Extending Battery Lifetime in Smartphones with Power Efficient Task Management and Energy Aware Design Tool

Hiroshi Inamura†, Takeshi Kamiyama†, Teppei Konishi†, and Ken Ohta†

†Research Laboratories, NTT DOCOMO, Inc.
inamura@nttdocomo.com, kamiyamat@nttdocomo.co.jp, teppei.konishi.xt@nttdocomo.com, ohtak@nttdocomo.co.jp

Abstract - In the use of smartphone, the battery lifetime is important factor that determines how long users can use the applications on the device. The application developers need to be aware of energy consumption for their application. In order to extend the battery lifetime, we recognize two issues; 1) Allowing developers to know the power-efficiency of their applications in real user environments for applications in design, 2) Optimizing the system to be power-efficient in the execution of background tasks for existing applications. We implemented a tool for developers to visualize how their code consumes the energy, with low overhead for data collection in runtime. We also improved the power efficiency in the background task management in Android OS and it shows up to 40% power reduction in call waiting time.

Keywords: Power Management, Android, Background Task, Global Synchronization, Power Modeling

1. INTRODUCTION

With the wider use of smartphone, it has become important to improve the battery lifetime of devices. Comparing to the featurephone, the smartphones are flexible both in development and deployment of 3rd party applications. Among major platforms of smartphones, a large percentage of devices are running Android OS [1]. Users are attracted to smartphones for the variety of applications available through application market. It is preferable all the applications are power efficient. The application developers need to be aware of energy consumption for their application. In order to extend the battery lifetime, we recognize two issues;

1) **Allowing developers to know the energy-efficiency of their applications in real user environments for applications in design**. The power consumption of device is determined by the utilization or the state of each hardware components such as CPU, which depends on the behavior of the application software. It is possible to prolong the battery lifetime by eliminating inefficiency from the applications, such as too frequent data transfer, for example. As a case study on a particular application, Furusho [2] showed that the energy consumption could be decreased by optimizing the application's behavior through an analysis of real users' actions. Given the competitiveness of the smartphone market, it is critical that developers be aware of how much power their applications consume and of the "Power Cost" of each function. The energy visualization integrated in SDK tool will be great help for developer to know the "Power Cost" in early stage of design.
2) **Optimizing the system to be power-efficient in the execution of background tasks for existing applications.** The background task (BG task) is a type of application component that does care-taker for processes with long waiting time and the role of service provider. A class of BG task has arbitrariness for the timing to be executed and does not cause any visible change to the user interaction. We can utilize this characteristic to shift the execution timing to minimize the utilization of hardware components. On the managing the execution timing, we need to consider the possibility of the synchronization of task executions among device, since it may leads network congestion if the communicating tasks are involved.

In this paper we describe an approach to extending the battery lifetime with our solutions for these two issues. With the combination of solutions, the extension can be made both for applications to be designed as well as existing applications on the market. We implemented a software tool for developer to visualize how their code consumes the energy, with low overhead for data collection in runtime. Also, we improved the power efficiency in the background task management in Android OS and it shows up to 40% power reduction for call waiting time.

2. ALLOWING DEVELOPERS TO KNOW THE POWER-EFFICIENCY OF THEIR APPLICATIONS

Software-based energy profiling demands an accurate power estimation scheme. The scheme should satisfy the following points. **a) Accurate estimation based on modeled characteristics of hardware components. b) Estimation is based on the data obtained through actual usage of applications.**

We propose an energy profiler for Android applications [3] based on a power model of devices and data obtained during applications usage.

To achieve a), we extend the existing power model proposed in [3]**[5]** to take account of new features of hardware components such as multi-core CPUs with dynamic frequency scaling and 3G/LTE RRC State transition. Regression analysis is used to allow the model to catch the relationship between power consumption and the behavior of each hardware component. In other words, the model is an ensemble of formulas, each with its own coefficients and

parameters that express the power consumption of each component.

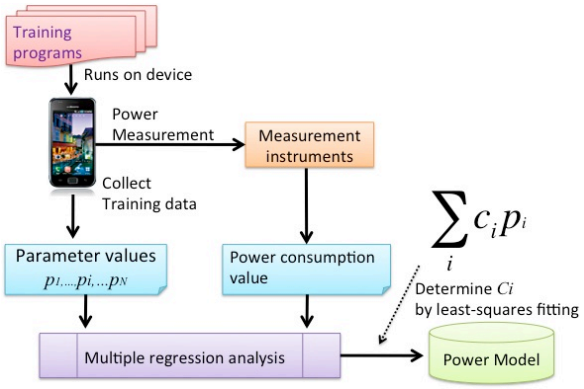

Figure 1: The process of power characterization

To achieve b), we log target applications while running to obtain the data needed generate the model parameters. The data reflects the impact of actual application usage on real devices, so the estimation results are practical. Once the model is generated and loaded into the profiler, there is no need to measure power consumption directly.

Based on the above ideas, we pose 2 requirements for the profiler as follows; 1) Accuracy of energy estimation targeting overall system is sufficient. 2) Logging overhead is small.

2.1 Related Works

Kaneda [5] proposed a system-wide power model that accurately estimates the power consumption. The model covers the main components and the power consumed by each component is parameterized by common data which is easily obtained at the OS-level, such as CPU utilization. Thus the good feature of their model design is not only that it's extendable for system-wide estimation but also each component can be logged with small overhead. However, it doesn't consider the recent features of components such as frequency scaling of multi-core CPUs and mobile wireless interfaces such as 3G/LTE.

ARO (Application Resource Optimizer) is a profiler that does address such recent features [6]. It features a power model of the mobile wireless interface. In 3G/LTE networks, the wireless link between device and base station is managed by RRC (Radio Resource Control), which handles the bearer setup and release, and mobility management [8][9]. RRC State used as model parameters can be estimated using the RRC State machine that specifies the states from packet capture data. However, special privilege such as root is needed on common devices to permit packet capture for logging and such logging incurs high CPU time overhead.

The previous works don't satisfy our energy profiler requirements. We describe an energy profiler by following the model design of [3][5]. This approach makes it easy to cover the overall system, has low logging overheads, and allows the model to be extended to account for the features of recent hardware components like ARO

.

2.2 Energy Estimation using Power Model

We describe here our device power model for energy estimation. As follows, the device model is expressed as the sum of the power consumption of each hardware component.

$$P_{est} = c_{offset} + \sum_{i}^{N} c_i p_i \qquad (1)$$

Where P_{est} is the total power consumption of the device. N is the number of hardware components. p_i is a parameter that covers the power consumption of hardware component i. c_{offset} and c_i are coefficients to be determined through multiple regression analysis.

Figure 1 shows the process of power characterization to generate the power model of a specific device. As in the existing scheme [5], power characterization is done by multiple regression analysis using the above equation and training programs to collect training data used in the analysis.

To collect the data, sets of power consumption values and parameter values, p_i, are measured under many test cases by the programs that run on the device. The coefficients of the equation, c_{offset} and c_i, are then obtained as a result of analyzing the training data.

One of our key points is that the parameters should be easy-to-obtained common values, such as CPU utilization, because it allows the process of power characterization to be as independent of device type as possible. We follow the basic design of our existing power model but extend it to cover the new features of multi-core CPUs and 3G/LTE.

2.2.1 Multi-core CPU

Recent smartphones use multi-core CPUs that can switch the number of active cores and scale frequency for energy-efficiency. Because our existing model assumes a single core CPU and only CPU utilization is used as a parameter, it has difficulty in accurately estimating the power consumption. We introduce a new model that can handle the complexity of multi-core operation.

Our new multi-core CPU power model is expressed as follows.

$$\begin{aligned} P_{cpu} &= P_{core1} + P_{core2} + \cdots + P_{coreN} \\ &= \sum_{i}^{N} c_i p_i = \sum_{i}^{N} c_i f_i u_i \end{aligned} \qquad (2)$$

Where P_{cpu} and P_{core1} to P_{coreN} are the overall power consumption of target CPU and the power consumption of each core in the CPU. We assume power consumption of each core can be separately modeled because recent frequency scaling technology allows cores to work independently.

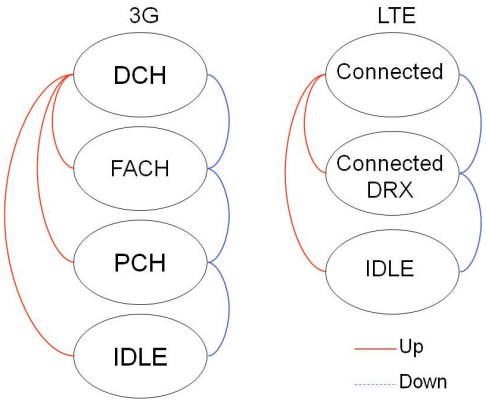

Figure 2: RRC State transition

Table 1: Power Model of Target Device

Component	Model Equation
CPU	$c_1 f_{core1} u_{core1} + c_2 f_{core2} u_{core2}$
Display	$c_3 p_{brightness}$
3G	$(c_4 + c_5 p_{sendBps} + c_6 p_{rcvBps}) p_{pch} + c_7 p_{fach} + c_8 p_{pch} + c_9 p_{idle}$
LTE	$(c_{10} + c_{11} p_{sendBps} + c_{12} p_{rcvBps}) p_{LTEconnected} + c_{13} p_{LTEcdrx} + c_{14} p_{LTEidle}$
Wi-Fi	$(c_{15} + c_{16} p_{sendBps} + c_{17} p_{rcvBps}) p_{flagWiFi}$
GPS	$c_{18} p_{gps}$
Disk	$c_{19} p_{readByte} + c_{20} p_{writeByte}$
GPU	$c_{21} p_{gpuLoad}$

Table 2: Values of Coefficients

i	Coefficient	i	Coefficient
0	1.462e-01	11	8.000e-08
1	1.375e-09	12	2.048e-08
2	1.162e-07	13	3.892e-02
3	2.340e-04	14	1.723e-02
4	1.563e-01	15	3.210e-02
5	3.265e-07	16	1.633e-07
6	1.107e-07	17	1.218e-07
7	1.179e-01	18	3.712e-02
8	1.384e-02	19	3.307e-10
9	1.402e-02	20	1.336e-09
10	2.054e-01	21	7.797e-10

c_i, p_i, f_i and u_i are the coefficient and parameter for core i, frequency value of core I, and CPU utilization rate of core I, respectively. As shown in (2), we define the core's work load, the product of f_i and u_i, as a new power consumption parameter for each core. This is because we think the parameter should allow consideration of the change of performance due to dynamic frequency scaling.

2.2.2 3G/LTE

In 3G/LTE networks, wireless link between device and base station is managed by RRC (Radio Resource Control), which handles the bearer setup and release, and mobility management [8][9]. The multiple link states, called RRC State, are defined in Fig.2, and the state is automatically switched depending on data transfer from/to device for the purpose of efficient use of wireless resources.

For example in the case of 3G, DCH is the state for the high throughput data transfer when using applications. FACH is the state for only low throughput. PCH is the stand-by state (entered when no data is transferred within a certain period). IDLE is the state of wireless link released. State transitions occur in various conditions as defined in RRC, such as data transfer defined in each state. For example, upper transition to DCH from lower states occurs when data transfer is requested. A lower transition to FACH, PCH or IDLE occurs after the inactivity timer, which runs while no data is being transferred, times-out. From the point of power consumption, it is known that more power is consumed in the upper state.

ARO accurately estimates the power consumption of the 3G/LTE interface using RRC State as a model parameter. In addition, ARO obtains the parameter value by RRC State estimation because the state cannot be directly accessed on general devices. Therefore, ARO loads the RRC State machine; it estimates the states by detecting data transfer from packet capture data (pcap), that is collected by the device. However, this can impose big CPU time overhead and needs super-user privileges. These limitations do not yield a widely provided, easy-to-use, profiler for general application developers.

In our work, we follow the basic idea of ARO's RRC state machine to extend our existing power model of wireless interface, and avoid the limitations noted. In detail, we replace packet capture logging with lightweight periodic logging of network statistics, which are easily accessed at the OS or API-level, such as /proc/net/dev. This yields much lower overheads because data volume is less than with than packet capture.

2.2.3 Result of Power Characterization

We show a result of power characterization for a real Android device that uses Qualcomm's MSM8960 [9] chipset. Model equations including parameters are shown in Table 1, and values of coefficients are shown in Table 2.

2.3 Application Energy Profiler

This chapter describes our application energy profiler equipped with the power model generated in the last chapter. Figure 4 shows the functional design of the profiler. The system consists of the Logger application, which runs on the device to collect the data needed for energy profiling, and the energy profiler, which analyzes energy consumption of the target application.

The logger application collects the log data needed for parameter generation at one second intervals while the target application is active. The energy profiler calculates the device's average power consumption in one second periods using the parameter values generated from the log data. Finally, the profiler shows the energy consumed by the application on a time chart, see Fig.3. The power breakdown for each hardware component is also shown in the chart. It enables the application developer to become aware of what factor in driving power consumption.

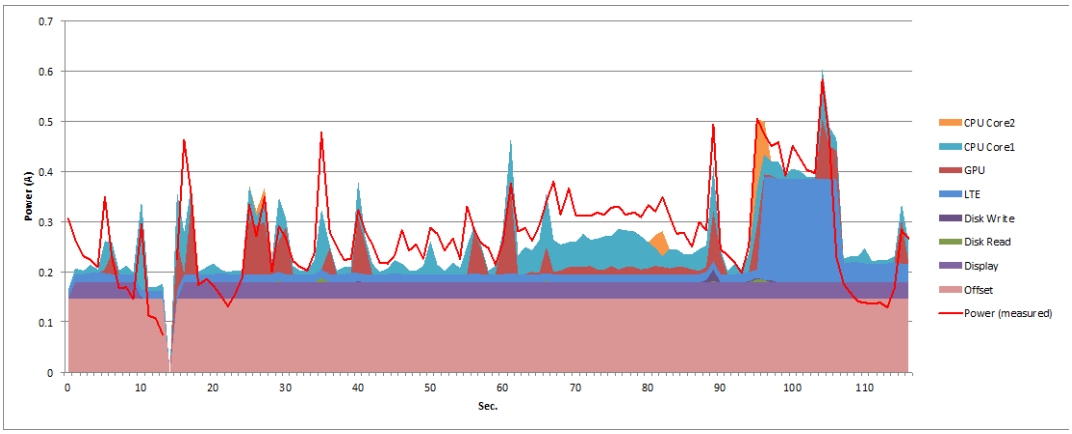

Figure 3: Estimated and measured power consumption for the mail application scenario.

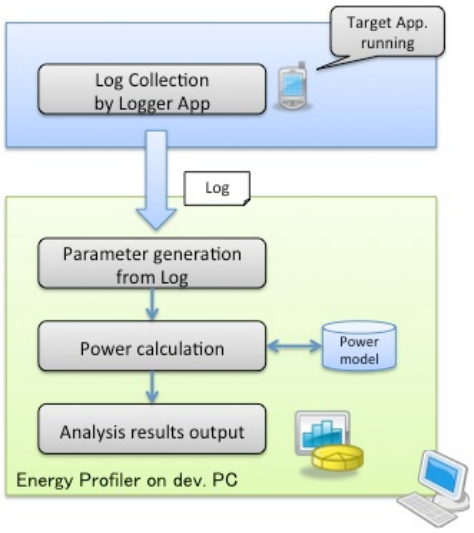

Figure 4: Functional design of Energy Profiler

2.4 Evaluation

We evaluated our energy profiler according to the two requirements, 1) accurate power estimation and 2) lightweight online logging, respectively.

2.4.1 Estimation Accuracy

To evaluate the accuracy of the energy estimates we actually measured the consumption under the following commonly used application usage of Mail, Map, Calendar, Movie player and Phonebook.

Figure 3 compares measured and estimated power consumption for the mail scenario 1). It is confirmed that the estimation basically follows the actual consumption for the entire period.

Through the all scenario, the profiler estimates energy consumption with about 10% error in application-mix as shown in Fig.3. Thus, it is considered that the profiler can works with constant accuracy.

However, at most 0.1A errors are observed around at 20 second, the period of 40 to 90 second and around at 110 second in Fig.3. We suppose the feature of display causes these errors because the device used in this time is equipped with OLED display; however our power model originally assumes LCD display. According to previous work [7], power consumption of OLED depends on the color and is able to be explained by RGB value of each pixel. As a result, dynamic change of color tone on the screen occurred and became the error factor.

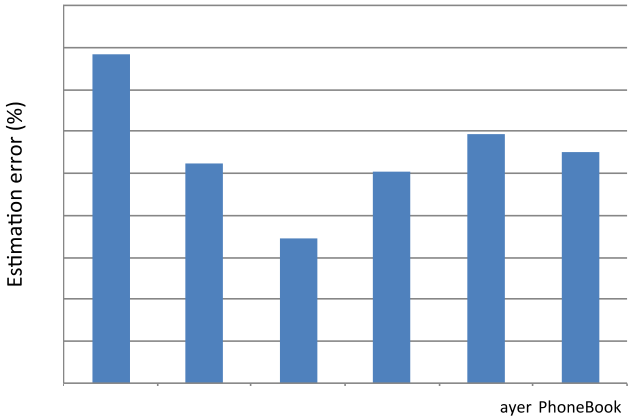

Figure 5: Estimation error in each scenario.

2.4.2 Overhead of logging

According to requirement 2), lightweight logging is important so as not to disturb natural use of the application in the tests. We pointed out the problem of ARO's logging overhead and presented a more lightweight logging approach for RRC State estimation to satisfy the requirement. Thus we compared the proposal to ARO in terms of CPU time overhead.

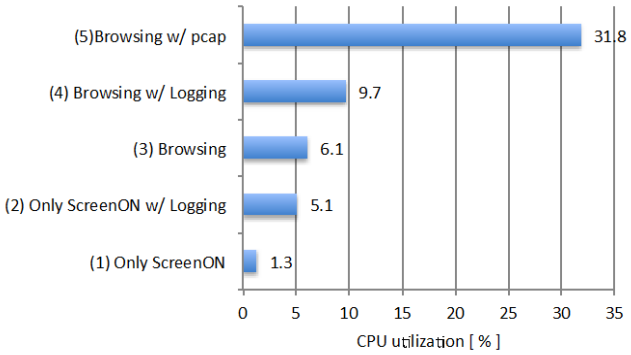

Figure 6: Comparison of CPU time overhead for logging

The logger application collects log data every second while the target application is running. First, we measured CPU utilization in the following test cases in order to understand the application's overhead.

(1) Leave the device as it is without logging

(2) Leave the device as it is with logging proposal

The results, shown in Fig.6, indicate that logging imposes 3.8 % of CPU time overhead (compare (1) to (2)).

Next, we test following test cases to compare the overhead of our logging with that of ARO.

(3) Automatic browsing without logging

(4) Automatic browsing with logging proposal

(5) Automatic browsing with ARO logging (pcap)

These test cases must involve data transfer because ARO's logging is packets capture and work on only condition that data transfer occurs. We use a test program that automatically download and display fixed Web pages on browser.

From the results of (3)-(5), the overhead of our logging proposal is 3.6% (compare (3) to (4)), whereas ARO's overhead is 25.7 %. As CPU utilization is one of the parameters of the CPU power model presented in section 2.2, the power consumption increases due to this overhead. Therefore, energy consumption estimated using packet capture could differ from the real consumption in normal usage of the application. Offsetting this effect is rather difficult because CPU load of packet capture depends on traffic pattern/volume of application usage. Furthermore, many applications for smartphones generally involve data transfer. Thus ARO logging may disturb natural application usage in many cases.

CPU overhead of our periodic logging proposal is small and always constant because it doesn't depend on the kind of application used. As stated above, our method achieves both accurate and lightweight estimation of 3G/LTE power consumption.

3. OPTIMIZING THE SYSTEM POWER-EFFICIENT IN BACKGROUND TASKS FOR EXISTING APPLICATIONS

We describe it is possible to improve the power efficiency in the execution of background tasks by shifting the execution timing. On the managing the execution timing, we need to consider the possibility of the synchronization of task invocations among device, since it may leads to network congestion if the communicating tasks are involved.

3.1 The BG task management mechanisms

Android OS has AlarmManager (AM) [12] that is a mechanism to invoke registered tasks at specified time and Broadcast Intent (BI) [13] that start tasks on the specified state change in the device during the circulation of event message. In this paper, we define wakeup task for tasks calling AM API having time spec with waking up directive (namely, RTC_WAKEUP, ELAPSED_REALTIME_WAKEUP). When the specified time has come, they wake up the device if it asleep. We also define non-wakeup task for tasks using non-wake up directive (namely, RTC, ELAPSED_REALTIME). The non-wakeup tasks are executed at exact time specified if the device already running, and at the most recent wakeup if the device asleep at the specified time.

The BG task invocation on state change happen when the devices' status has changed such as the screen turned on/off, change in the battery status and receipt on SMS, so on. For example, suppose a battery meter application that displays the remaining amount of electric energy in the battery. If the amount changed, the Android OS notify the event using a Broadcast Intent (BI) and the application updates the displayed info. In order to circulate the BI, the device is waked up. Therefore, the wake up for BI circulation will initiate the BG task invocation scheduled in AM. We define the BG task that is executed on BI, as system event (BI).

3.2 Avoiding the network congestion caused by the simultaneous invocation of BG tasks

We need to avoid the network congestion while enabling the simultaneous execution of background tasks for power efficiency.

3.2.1 Network Congestion by simultaneous BG task invocations

Since it is possible to specify the time of BG task invocation precisely in AM API, the network congestion is concerned caused by simultaneous BG tasks invocation at the specific time among many devices with the software created and operated carelessly [14][15]. The interworking/interference between wakeup task and non-wakeup task can synchronize devices globally at specific time. We define such unintended synchronization as "sync by interference" that occur when the device wakes-up with wakeup task, the device runs non-wakeup tasks. The issue is the invocation of non-wakeup tasks aligning to the specific time the wakeup task stated. In order to avoid the network congestion, we need to suppress the "synch by interference" among tasks that may initiates network communication when it is invocated. For preventing the network congestion by application activities, suppressing the "synch by interference" is important issue.

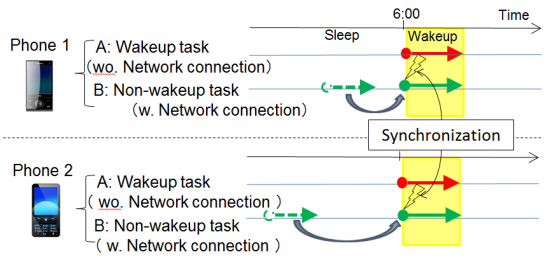

Figure 7: Non-wakeup task is invoked with wakeup task, which causes unintended global synchronization of task invocation among devices

3.3 Case: A Global Synchronization by SBI

Figure 1 shows the two devices, namely phone1 and phone2, each has been installed app A and app B. The app A wakes up device at 6am every day, and it does not start any

communication but does some housekeeping task. The app B is a non-wakeup task and will start connection to other servers. As shown in Fig.1, even if the start timing for app B set to different time stamp such as 5:58 and 5:55, the invocation of app B is pending towards 6am, the app A will invokes app B in both cases by the device wakeup. So the app B will start communication at 6am for both devices at globally synchronized. It is easy for developers to expect the synchronization caused by communicating app that wakes up at specific time. The difficulty is to foresee the wakeup task without networking can cause congestion. Under such circumstances, the developer may specify every 00 minutes or 30 minutes of much "aligned" time for the invocation in the application. The developer of app B pays no attention for the possibility of congestion, since it does not specify exact time for invocation. Still, as shown here, there is possibility for global synchronization by the combination of different type of BG tasks.

3.4 Related Works

We could not find any single previous work to cover our issue. For the power optimization by simultaneous task execution, Calder [17], Kononen [18] showed the effectiveness. But we need consideration for network congestion. Yamamoto [19] and other related study [20] worked on power optimization and avoiding network congestion, still we need to preserve developer' s design for controlling the invocation timing. Although the task killer applications [16] are solving the power issues utilizing user intervention, they do not keep original behavior in applications.

3.5 Issues in BG task management mechanism and a proposal for a new control scheme

We propose a new BG task management mechanism that simultaneously invokes non-wakeup task with wakeup task in inexact type only. The mechanism preserves their semantics since the invocation timing in non-wakeup tasks have arbitrariness in BG and in wakeup task with inexact type is chosen by system inherently. Our proposal can reduce the energy consumption in BG task and avoid the network congestion.

3.5.1 Implication in API semantics provided by AlarmManager

In order to modify the invocation timing of BG tasks with less impact to intended behavior in the applications, we need to control them keeping the real time semantics designed in AM API. According to the API design, the real-time semantics can be classified into following three categories.

- Exact real time(API: wakeup task with 'exact' specifier)

The execution time needs to align with real time clock's specific time stamp.

- Exact interval(API: wakeup task with 'inexact' specifier)

The invocation need not be exactly specified time, but the interval between the invocations should be specified amount of time. AM has an SetInexactRepeating API for the purpose that keeps the interval, with the first invocation the system specified.

- Non real time (API: Non wakeup task)

There is no strong requirement in the invocation time and their interval in BG execution. Since it does not wake up the device, the invocation is opportunistic that any time after the designated time is acceptable, during the screen is off.

3.5.2 Avoiding the global synchronization between task invocations

The current implementation of AM invokes the non-wakeup tasks on device wakeup caused by the system events (BI) or wakeup tasks and they are executed altogether The "synch by interference" (SBI) happens when simultaneous execution of wakeup tasks with exact designation and non-wakeup task. Since system chooses specific timing for the wakeup task with inexact designate, it does not cause SBI issue. In order to prevent SBI, we propose a mechanism that exclude wakeup task with exact designate from simultaneous execution with non-wakeup tasks.

3.5.3 Power Reduction in simultaneous execution of Non-wakeup tasks

Using a model equation, we describe the energy consumption can be reduced by simultaneous execution of non-wakeup tasks and wakeup tasks better than non-wakeup tasks and system events (BI). As showed in Sec 2, non-wakeup tasks are executed on the device wakeup later than specified time stamp. So, non-wakeup task (N) is simultaneously executed with either wakeup task (W) or system events (BI).

$$\sum_{\{BI,W,N\}}^{i} t_i (P_{off} + P_c)$$

The power consumption with the simultaneous execution of all the three types (W, N, BI) is described as follows. Let P_{off} as the baseline power consumption of zero utilization in all resources. P_c as the averaged power consumption over unit time. $t_{\{N,W,BI\}}$ are the execution time for each type of BG tasks respectively non-wakeup (N), wakeup (W) and system events (BI). With the optimized simultaneous execution in our proposal, the power consumption modeled as follows. For timing manipulation, because we cannot change the timing of wakeup (W) and system event (BI), we will set the non-wakeup (N) aligned to either one of (W, BI) to be invoked. The power offset are shared during the simultaneous execution.

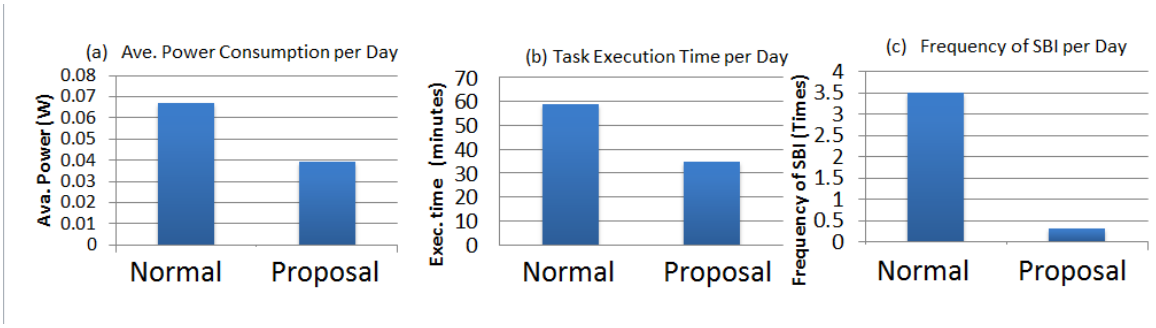

Figure 8: Evaluation of Our Proposed Method

Based on Sec 3.1, suppose the execution time is the longest among the tasks executed in parallel and the resource utilization will be the total of each, the power consumption is given in;

$$P_c \sum_{\{BI,W,N\}}^{i} t_i + P_{off} \min \begin{pmatrix} \max(t_{BI}, t_N) + t_W \\ \max(t_W, t_N) + t_{BI} \end{pmatrix}$$

Looking at Table 3 summarized the result in our user study, comparing the averaged execution time for non-wakeup task and system event are 7.8[s] and 0.5[s], respectively and differs 1:16 ratio. Considering with the averaged execution time in wakeup of 8.2[s], we see $t_W \cong t_N > t_{BI}$; therefore, the non-wakeup (N) task should be executed only with wakeup (W). With the consideration, we derive the optimized power consumption in model as;

$$P_c \sum_{\{BI,W,N\}}^{i} t_i + P_{off} (\max(t_W, t_N) + t_{BI})$$

The non-wakeup task should be executed with wakeup task in the same time, which improves the power consumption of the device as a whole.

Table 3: Average execution time of BG task

Execution Time	AM tasks		system event
	wakeup task	non-wakeup task	
Average (sec)	8.2	7.8	0.5
Std. Dev. (sec)	3.1	2.4	0.1

3.6 BG task control scheme

We propose a mechanism that invokes non-wakeup task only with Exact Interval wakeup. Since the execution timing will be chosen by system and randomized in Exact Interval wakeup, there is no risk for causing SBI. We have a chance to optimize power efficiency in BG task by simultaneous execution of non-wakeup tasks. As we discussed in 3.5.2, execution with exact real-time may cause SBI. For power efficiency, we exclude BI from execution opportunity as seen in 3.5.3. The remaining timing opportunity for execution is with Exact Interval wakeup in AM.

With this scheme, it is possible to apply the optimized BG task invocation to existing applications as is, without re-designing them to adopt to any new API. Our idea is preserving real-time semantics in existing AM API with minimum modification in task invocation policy.

3.7 Implementation and Evaluation

We have implemented our proposed scheme on AOSP Android r4.2.1. We have modified the task invocation policy in AlarmManager into following;

1. Classify the trigger of wakeup
2. Only if the wakeup is caused by SetInexactRepeating, invoke the pending non-wakeup task

The power consumption of the device with the implementation of our proposal and without are shown in Fig.8(a) as 0.039[W] and 0.067[W], respectively. It means 40% reduction while screen turned off (call waiting). The result explained as the reduction of execution time in BG shown in 8(b). In addition, the synchronization between non-wakeup task and exact designated wakeup task is 0.3 [times/hour] in our scheme, while 3.5 [times/hour] in normal device. So, we could eliminate 90% of SI issue during the screen is turned off.

Our proposed method ensures the real time constraint designed in application and avoids re-design them for improvements. So there is least impact for user and developer perspective while achieving energy reduction. Also we could effectively eliminated the issue of “synch by interference” and network congestion that may be caused by concentration of task execution among devices. Since the amount of traffic from devices during the screen turned off is determined the behavior of BG tasks, our proposal is successful in prevention of network congestion as well.

4. SUMMARY AND CONCLUSION

In this paper we describe an approach to extending the battery lifetime with these two issues and our proposals. With the combination of solutions, the extension can be made both for applications to be designed as well as existing applications on the market.

We implemented a tool for developer to visualize how their code consumes the energy, with low overhead for data collection in runtime. Our implementation is a model-based energy profiler for Android applications, taking account of the features of multi-core CPUs and 3G/LTE; Experiments showed that it estimates energy consumption with about 10% error in the application-mix examined, and that logging incurs a CPU time overhead of only 3.8%, which superior to other profilers. We improved the power efficiency in the background task management in Android OS and it showed up to 40% power reduction for call waiting time. We also pointed out the issue of “synch by interference” and our scheme successfully suppresses the issue as well.

REFERENCES

[1] Google.Inc : Android http://www.android.com/about/. (accessed 2013-04-21).

[2] H. Furusho , K. Hisazumi , T. Kamiyama , H. Inamura , T. Nakanishi, and A. Fukuda, ``An Energy Profiler for Android Applications Used in the Real World,'' Proc. of the 10th International Conference on Mobile Systems, Applications and Services (MobiSys'12), pp.517-518 (2012).

[3] T. Kamiyama, H. Inamura, and K. Ohta, ``Evaluation of Model based Energy Profiler for Android Applications,'' Proc. of DICOMO2013 Symosium, pp286-292 (2013).

[4] T. Kamiyama, and M. Katagiri, ``Visualization of mobile terminal power consumption based on OS-level analysis,'' NTT DOCOMO Technical Journal, Vol.11 No.3, pp.72-76 (2009).

[5] Y. Kaneda, T. Okuhira, T. Ishihara, K. Hisazumi, T. Kamiyama, and M. Katagiri, ``A Run-Time Power Analysis Method using OS-Observable Parameters for Mobile Terminals,'' Proc. of International Conference on Embedded Systems and Intelligent Technology, pp.39-44 (2010).

[6] F. Qian, Z. Wang, A. Gerber, Z. Morley Mao, S. Sen, and O. Spatscheck, ``Profiling Resource Usage for Mobile Applications: A Cross-layer Approach,'' Proc. of MobiSys 2011, pp.321-334 (2011).

[7] M. Dong, and L.Zhong, ``Chameleon: A Color-Adaptive Web Browser for Mobile OLED Displays,'' Proc. of MobiSys 2011, pp.85-98 (2011).

[8] 3GPP TS 25.331 Radio Resource Control (RRC); Protocol specification, 3GPP Website, http://www.3gpp.org/ftp/Specs/html-info/25331.htm.

[9] Qualcomm Inc., http://www.qualcomm.com/snapdragon.

[10] 3G Specifications, 3GPP Website, http://www.3gpp.org/3GPP-specifications.

[11] 3GPP TS 36.331 Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Resource Control (RRC); Protocol specification, 3GPP Website, http://www.3gpp.org/ftp/Specs/html-info/36331.htm.

[12] Android API: Alarm Manager (online) available from http://developer.android.com/intl/ja/reference/android/app/AlarmManager.html (accessed 2013-04-21).

[13] Android API: Broadcast Receiver http://developer.android.com/reference/android/content/Context.html#sendBroadcast(android.content.Intent (accessed 2013-04-21).

[14] T. Konishi, T. Kamiyama, S. Kawasaki, H. Inamura, ``Reducing the use of Energy and Wireless Resources on Android Devices during Screen Inactive Periods,'' DPS Workshop IPSJ, pp. 249-256 (2012).

[15] S. Kawasaki, T. Kamiyama, S. Ohkubo, H. Inamura, ``A congestion avoidance method for event delivery mechanism,'' IPSJ SIG Technical Report Vol 2012-CDS-6, pp.1-8 (2012).

[16] Google Play: Advanced task killer. (online) available from https://play.google.com/store/apps/details?id=com.rechild.advancedtaskkiller&hl=ja (accessed 2013-04-21).

[17] M. Calder, and M. Marina, ``Batch Scheduling of Recurrent Applications for Energy Savings on Mobile Phones,'' Proc. of 7th Annual IEEE Communications Society Conference on Sensor Mesh and Ad Hoc Communications and Networks (SECON2010), pp. 1–3 (2010).

[18] V. Kononen, and P. Paakkonen, ``Optimizing Power Consumption of Always-On Applications Based on Timer Alignment,'' Proc ˆ of COMSNETS2011, pp.1-8 (2011).

[19] T. Yamamoto, S. Saruwatari, M. Minami, H. Morikawa, ``Piggyback Transport Protocol: Energy-efficient Upload Engine for Participatory Sensing,'' Journal of IPSJ, Vol.53, No.1, pp.274–285 (2012).

[20] E.J. Vergara, and S. Nadjm-Tehrani, ``Energy-aware Cross-layer Burst Buffering for Wireless Communication,'' Proc. of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet. e-Energy '12, pp.1-10 (2012).

(Received December 29, 2013)
(Revised March 26, 2014)

Hiroshi Inamura joined NTT DOCOMO, Inc. in 1998. His research interests include system research on mobile device and distributed system. Before DOCOMO, he was a research engineer in NTT labs since 1990. From 1994 to 1995, he was a visited researcher in the Department of Computer Science, Carnegie Mellon University. He received B.E., M.E. and D.E. degree in Keio University, Japan. He is a member of IPSJ, IEICE, ACM and IEEE.

Teppei Konishi His main research interests include energy-efficient task scheduling for Android OS. He received B.E. and M.E. degree in Osaka University, Japan. He is currently a research engineer at Research Laboratories, NTT DoCoMo Inc. He is a member of IPSJ.

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 DOCOMO, he received MS degree in University of Tokyo, Japan in 2006. Also, he was co-founder and CEO in e-jis, Inc. from 2003 to 2006. He is a menber of IPSJ.

Ken Ohta received the BE, ME, and DE degrees from Shizuoka University, Japan in 1994, 1996, and 1998, respectively. In 1999, he joined NTT Mobile Communications Network Inc. (NTT DOCOMO). His research interests include mobile computing, distributed systems, and system security. He is a member of the Information Processing Society of Japan and of the Institute of Electronics, Information and Communication Engineers.