A Battery Lifetime Guarantee Scheme for Selective Applications in Mobile Multimedia Electronics

Jung-wook Cho

Computer Systems and Network
Electrical and Computer Engineering
Graduate School of UNIST

A Battery Lifetime Guarantee Scheme for Selective Applications in Mobile Multimedia Electronics

A thesis
submitted to the School of Electrical and Computer Engineering
and the Graduate School of UNIST
in partial fulfillment of the
requirements for the degree of
Master of Science

Jung-wook Cho

20. 6. 2011
Approved by

____

Major Advisor
Euiseong Seo

A Battery Lifetime Guarantee Scheme for Selective Applications in Mobile Multimedia Electronics

Jung-wook Cho

This certifies that the thesis of Jung-wook Cho is approved.

20. 6. 2011

Thesis Supervisor: Euiseong Seo

Jongeun Lee

Younho Lee

Abstract

Unpredictable battery lifetimes because of multitasking significantly harm the availability of mobile devices. Existing research has focused on improving energy efficiency in order to extend battery lifetimes or guaranteeing the battery lifetime of the whole system. However, some applications are prioritized over others and these are required to stay in operation for certain durations. This paper suggests a battery lifetime guarantee method for prioritized applications in multitasking mobile systems. Our approach profiles and analyzes the battery usage patterns of each task and suggests preserving the energy budget for prioritized tasks while maintaining service quality by scheduling non-prioritized tasks to use only surplus energy. The suggested method is evaluated in an Android smartphone. Our approach shows 1%~10% positive errors with respect to lifetime guarantee but no negative errors.

Contents

List of figures

List of tables

I. INTRODUCTION

Over 100 million smartphones were sold during quarter 4 of 2010, which amounted to 80% year-on-year growth (Canalys, 2011). The performance of smartphones is also rapidly improving, and some multicore processors already overwhelm the desktop processors of the early 2000s. Propelled by this high performance, the third party smartphone app market is dramatically expanding. For example, more than 500,000 third party apps are currently sold at app stores.

In spite of the explosive growth of smartphone hardware and software, battery lifetime remains the greatest cause of user dissatisfaction (CFI Group, 2009). The battery capacity of mobile devices, which primarily determines their battery lifetimes, is limited by the form and external design of the devices. Therefore, the displeasure of mobile device users from low battery lifetimes is expected to remain for the forthcoming years.

Moreover, the continual additions of diverse peripheral components such as GPS (global positioning system), NFC (near-field communication) and ambient sensors to smartphones are broadening the application fields of third party applications. Both the increase in the number of third party applications and the expansion of their application fields have lead to the high degree of multitasking that results in a shortened battery lifetime. Besides this shortened battery lifetime, the unpredictability of the battery lifetime because of the high degree of multitasking becomes another source of user dissatisfaction.

Most smartphone systems provide tools for monitoring the remaining battery charge and estimating the remaining lifetime of the battery. However, these estimation methods are based on a rule of thumb approach that divides the remaining battery charge by the observed energy draining rate of the entire system. This approach cannot predict the actual battery lifetime when applications start to execute and finish spontaneously because such sudden changes affect the battery draining rate. In spite of this unpredictability in battery lifetime, users have virtually no means to control the system's operation time except terminating certain applications to save energy and extending the operation times of other important applications.

To overcome these limitations, researchers have suggested diverse approaches that guarantee the system's operation time by restrictively scheduling applications according to their priorities and the remaining battery charge available. However, third party smartphone applications have different lifetime requirements depending on their purposes. For example, the importance of the lifetime guarantee of a movie player takes precedence over that of other background applications until the movie finishes, at which point the need for the lifetime guarantee of the movie player disappears.

Research has attempted to control the energy consumption of each application according to its characteristics and priorities. However, existing approaches require applications to be redesigned so that they can notify their energy requirements and change their behaviors in response to the remaining battery charge. In addition, operating systems would have to be thoroughly modified because they have to arbitrate between the energy requirements of various applications. Such major restructuring implies complicated programming interfaces to precisely control energy usage, and all applications, even background applications, would have to explicitly notify the kernel about their energy requirements through these interfaces.

This paper proposes a battery lifetime guarantee method that guarantees the battery lifetime of each application in a mobile system. Instead of making sweeping changes to operating systems and applications, our approach does not require any changes to existing applications and it can be constructed with only minor modifications to operating systems. Furthermore, our approach is able to provide lifetime-guaranteed scheduling to chosen applications, while providing best-effort scheduling to others. For instance, the prototype of the suggested method has already been implemented in the Google Android operating system and it is being evaluated by the commercial smartphone, Google Nexus One.

The organization of this paper is as follows. The background and related works are introduced in Section 2. In Section 3, we suggest the battery lifetime method for selected applications in a mobile device. Section 4 introduces the implementation of the prototype and evaluates its results. We conclude our research in Section 5.

REFERENCES

1. Canalys, Google's Android becomes the world's leading smart phone platform, 2011.

2. CFI Group, Smartphone Satisfaction Study 2009.

II. MOTIVATION AND RELATED WORK

2.1 Motivation

Lithium-ion polymer batteries (Li-ion batteries) are currently the most popular energy storage for modern mobile devices. Li-ion batteries use electronic control circuits to protect themselves from overcharging and overheating (Martin and Siewiorek, 2004). In many cases, these charge control circuits provide standard interfaces such as ACPI (Advanced Configuration and Power Interface) to the upper software layers including the operating systems so that the software can check battery status on the go.

Most mobile electronics including laptops and smartphones predict the remaining battery lifetime based on the system-wide power consumption history and remaining battery charge. Although such a simple estimation provides users with a degree of information, few commercial systems provide an effective means to control battery lifetime (Roy et al., 2011).

For example, Google Android, which is a commercially successful open source smartphone operating system, provides an energy usage monitoring method. This method is able to not only measure the energy consumption of the whole system, but also estimate the amount of energy used by each application by monitoring the amount of system resources it uses. Currently, Android monitors the use of the following resources to estimate energy consumption:

1) Processors: Most smartphone application processors can change their operating clock frequencies dynamically through the dynamic voltage and frequency scaling (DVFS) feature in order to improve energy efficiency (Carroll and Heiser, 2010). Android calculates the energy consumption of the processor by counting both the processor’s utilization time and the clock frequency at that time.

2) Networks: Four types of network resources are monitored, namely phone, radio, Wi-Fi and Bluetooth. Phone and radio are for phone calls and stand-by, respectively. The wireless network resources consume static power, which does not change, while they are waiting for incoming data. In addition to the static power, they consume dynamic power when they send or receive data.

3) Screen: The power consumption of the screen varies according to the brightness of the backlight.

With the above-mentioned power consumption model, Android estimates how much energy each resource contributes to overall energy consumption. However, this model is only about the contribution ratio. The absolute values of the system's energy consumption come from the sensor built into the battery. By aggregating the information from these two sources, Android can calculate the amount of energy each resource uses and as well the amount of energy each application consumed.. The kernel battery driver notifies the user-level energy management service following every 1% decrease in the remaining charge, and actual energy consumption is then calculated by each resource and by each application in response to this notification.

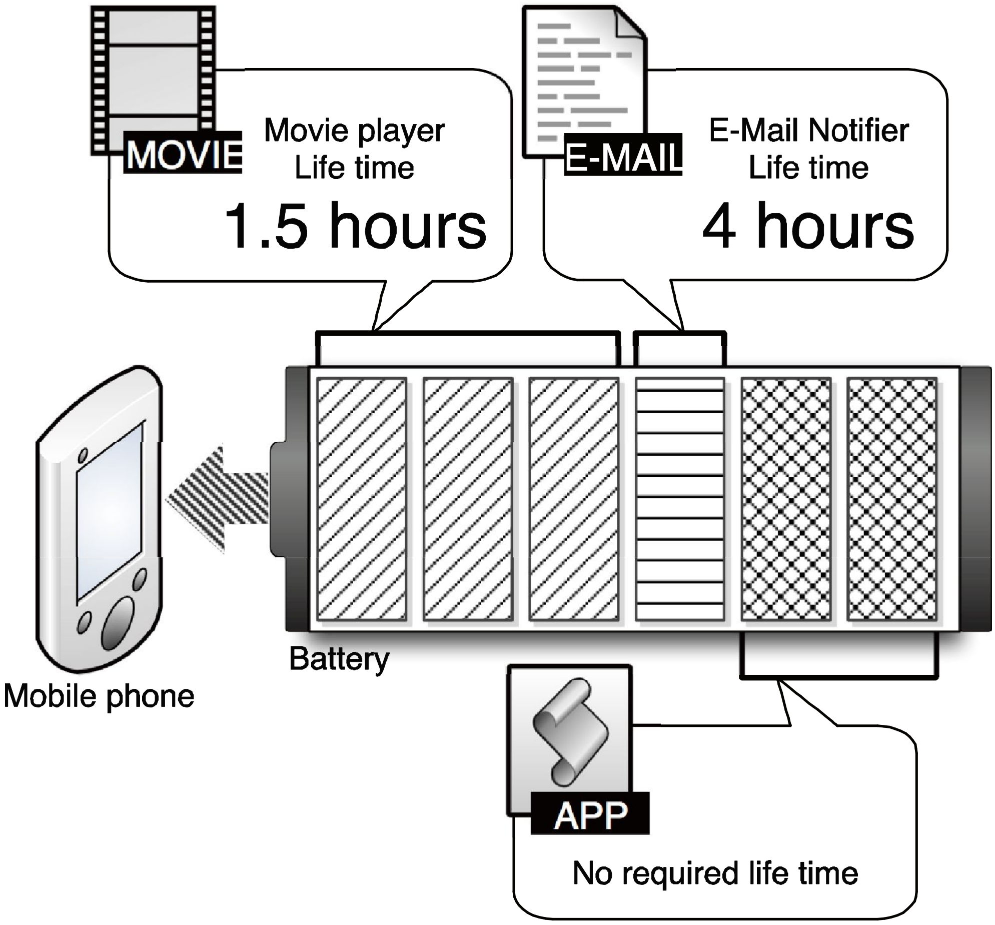


Figure 2-1. Applications in a smartphone may have their own lifetime requirements.

Although this approach is typical of most mobile electronics and has been proven useful for predicting the remaining battery lifetime as mentioned before, Android has no means to control the battery lifetime of the entire system or to guarantee a predefined lifetime for an application.

In this research, we thus assume that (i) each application in a system has different battery lifetime requirements according to their characteristics and (ii) multiple applications with different battery lifetime levels can run simultaneously. For example, Figure 2-1 shows that a movie player should play for an hour and a half, while e-mail notification should work normally for four hours. Some applications do not have specific time requirements. Such applications should run without restriction as best-effort way, while the system should reserve sufficient energy.

New applications are continually added or updated to mobile systems. Indeed, identical applications can consume different amounts of energy depending on the underlying hardware of the device. In addition, coexisting applications can affect the energy consumption of an application. Therefore, the amount of energy used by an application is difficult to predict in its design or implementation stage, and therefore the proposed battery lifetime guarantee method aims to react to the dynamically changing energy consumption of applications.

Users can run applications spontaneously at any time. Accordingly, they should be able to request a battery lifetime guarantee for new applications whenever necessary. The proposed system allows lifetime guarantee requests as long as the system condition allows them.

To guarantee the system operation time, existing studies have typically controlled energy consumption by restricting scheduled applications. For example, if the remaining battery charge is insufficient, the battery lifetime can still be guaranteed by not executing applications that are of low priority or maintaining energy consumption by limiting application runtimes. However, existing methods fail to take into account the behavioral characteristics of multimedia or interactive applications.Since most applications that run in mobile systems are multimedia or interactive applications, failing to schedule for a certain period results in poor user experiences. For example, if a video player does not wake up once every ten milliseconds or so, the frame rate will drop. However, simply limiting scheduled applications by that amount of energy may prevent applications from waking up on time. Moreover, multimedia players consist of multiple threads that use up the processor's resources for buffering if they remain. Therefore, the processor's resources diminish for the thread rendering frames on screen, which causes the application to lower its service quality. In order to maintain service quality, these issues should be resolved.

Finally, the battery lifetime guarantee method should be easily applicable to existing operating systems and applications and provide simple and intuitive user interfaces. In order to meet these objectives, the proposed method cannot depend on applications directly providing energy usage information. Instead, it has to obtain this information from the operating systems and device drivers and control energy consumption through the task scheduler in the operating systems.

2.2 Related Work

The most intuitive approach to overcoming the insufficient battery lifetimes of mobile devices is increasing the energy efficiency of the system through the DVFS feature of processors (Flautner and Mudge, 2002, Yuan and Nahrstedt, 2003, Choi et al., 2004, Seo et al., 2008) or through the dynamic power management (DPM) feature of peripheral devices (Seo et al., 2010, Cheng and Goddard, 2006).

If the processor lowers its operating clock speed using DVFS, the energy consumption needed to execute an instruction decreases. However, a lowered clock speed prolongs the response times of applications. Like DVFS, DPM saves energy by making peripheral devices sleep during their idle times, but the wake-up delay from this sleep state may induce unexpected response delays. Therefore, researchers have focused on ways to increase energy efficiency while minimizing the sacrifice of the quality of services. However, improving the energy efficiency of mobile devices alone does not always guarantee to increase the lifetimes of applications.

The accurate control of energy requires the ability to dynamically monitor the amount of energy used by each application or device in a system (Roy et al., 2011). In many research efforts (Flinn and Satyanarayanan, 1999, Zeng et al., 2002, Economou et al., 2006, Snowdon et al., 2009, Kim et al., 2011), the energy consumption by an application or a device has been obtained by estimating the number of device accesses or the used processor time. However, the accuracy of such estimation methods is reported to be low, and because of the ever-increasing diversity of components in a mobile system, improved accuracy does not seem likely in the future (McCullough et al., 2011). Calculating the energy consumption of each device, application or virtual machine is becoming an important issue in the server domain as well (Kansal et al., 2010).

Many studies have suggested approaches for predicting and controlling the battery lifetimes of mobile systems. For example, the Odyssey Operating System (Flinn et al., 1999) predicts future energy consumption and available battery lifetime by monitoring prior energy consumption. When the remaining battery charge is unable to provide a sufficient battery lifetime, the application will receive a request to reduce its energy consumption. Applications thus reduce their energy consumption levels by adjusting service fidelity, which means switching to one of several predefined low-power modes. However, operating systems still find it difficult to control the amount of energy consumed in these power-saving modes.

Neugebauer and McAuley (2001) introduced the Nemesis operating system, which registers the power consumption of each device during the calibration process in order to calculate the exact amount of energy consumed by each application. Based on this information, the amount of resources applications have used can be totaled. This management of resources is similar to Odyssey's approach. When an application is required to spend more than its limited resources, it will receive a feedback signal and its behavior will be adapted accordingly.

Bellosa (2000) proposed the concept of ‘energy-aware scheduling’. Based on an analysis of the energy pattern per thread, a throttling thread, which forces the halt cycle to reduce energy consumption, will execute if the consumed energy exceeds the predefined average energy. However, the proposed method is impractical in real-life mobile systems because it includes only the energy consumption patterns of the CPU and memory and fails to consider the wireless adapter and screen, which actually account for a large proportion of energy use.

Compared with existing operating systems that aim to maximize performance and fairness, Vahdat et al. (2000) designed a system that considers the energy efficiency of its power management policy. Likewise, Zeng et al. (2002, 2003) proposed a prototype ECOSystem (Energy-centric Operating System) that considers energy the primary resource of the operating system. They introduced the Currentcy abstraction, which means the amount of energy used by devices such as the CPU, disk drive and network during a certain period. In the ECOSystem, in order to guarantee a user's desired battery lifetime, the available Currentcy of the entire system is calculated for each interval and distributed to each application according to the user's preferences. The battery lifetime is thus guaranteed since applications only execute within their given Currentcy limits.

Banga et al. (1998) regarded the processor as the principal resource in conventional operating systems and introduced a resource container to account the amount of resources necessary for a specific activity. This resource container could control fine-grained resource management in the server system.

Roy et al. (2011) extended Banga's model to assess the energy consumed by an application in a mobile system. They proposed a hierarchical resource container in the Cinder Operating System, which could allocate energy and account energy usage by thread based on detailed information. While the ECOSystem uses a flat structure of resource containers, the Cinder Operating System utilizes hierarchical resource containers, which helps control the energy consumption of each thread and prevents malicious applications from consuming excessive energy.

Most existing studies (Flinn and Satyanarayanan, 1999, Neugebauer and McAuley, 2001) are limited because they modify applications in response to battery status. In addition, some guarantee the battery lifetime of the whole system rather than guaranteeing the lifetime of each application (Flinn and Satyanarayanan, 1999, Bellosa, 2000, Zeng et al., 2002, Zeng et al., 2003). To guarantee the battery lifetime of each application, the operating system provides the specific interfaces to control the complex energy flow (Rumble et al., 2010, Roy et al., 2011).

Our study proposes a software framework that guarantees the battery lifetime of each application according to a user's needs. In our study, we distinguish between prioritized and non-prioritized tasks depending on a user's requests and profiles and analyze the battery usage of each application. Based on this information, when the battery lifetime of each application cannot be guaranteed, non-prioritized applications are no longer scheduled and prioritized applications are only scheduled according to their energy histories. In this way, the battery lifetime of each application can be guaranteed. Our system can be used universally across mobile systems because it does not need to modify existing applications or the power management of devices.

REFERENCES

1. Martin, T. and Siewiorek, D., A power metric for mobile systems, In 1996, Proceedings of the 1996 international symposium on low power electronics and design, 37–42.

2. Roy, Arjun and Rumble, Stephen M. and Stutsman, Ryan and Levis, Philip and Mazieres, David and Zeldovich, Nickolai, Energy management in mobile devices with the cinder operating system, In 2011, Proceedings of the sixth conference on computer systems, EuroSys 11, 139–152.

3. Carroll, Aaron and Heiser, Gernot, An analysis of power consumption in a smartphone, In 2010, USENIX Association 21–21.

4. Flautner, Krisztian and Mudge, Trevor, Vertigo: automatic performance-setting for Linux, In 2002, Proceedings of the fifth symposium on operating systems design and implementation, 105–116.

5. Yuan, Wanghong and Nahrstedt, Klara, Energy-efficient soft real-time CPU scheduling for mobile multimedia systems, In 2003, Proceedings of the nineteenth ACM symposium on operating systems principles, 149–163.

6. Choi, Kihwan and Soma, Ramakrishna and Pedram, Massoud, Fine-Grained Dynamic Voltage and Frequency Scaling for Precise Energy and Performance Trade-Off Based on the Ratio of Off-Chip Access to On-Chip Computation Times, In 2004, IEEE Computer Society, 10004

7. Seo, Euiseong and Park, Seonyeong and Kim, Jinsoo and Lee, Joonwon, TSB: A DVS algorithm with quick response for general purpose operating systems, In 2008, J. Syst. Archit., 1–14.

8. Seo, Euiseong and Kim, Sangwon and Park, Seonyeong and Lee, Joonwon, Dynamic alteration schemes of real-time schedules for I/O device energy efficiency, In 2011, ACM Trans. Embed. Comput. Syst., 23: 1–32.

9. Cheng, Hui and Goddard, Steve, Online energy-aware I/O device scheduling for hard real-time systems, In 2006 Proceedings of the conference on design, automation and test in Europe:, 1055–1060.

10. Roy, Arjun and Rumble, Stephen M. and Stutsman, Ryan and Levis, Philip and Mazieres, David and Zeldovich, Nickolai, Energy management in mobile devices with the cinder operating system, In 2011, Proceedings of the sixth conference on computer systems, EuroSys 11, 139–152.

11. Flinn, J. and Satyanarayanan, M., Energy-aware adaptation for mobile applications, In 1999, Proceedings of the seventeenth ACM symposium on operating systems principles, 48–63.

12. Zeng, H. and Ellis, C.S. and Lebeck, A.R. and Vahdat, A., ECOSystem: managing energy as a first class operating system resource In 2002, ACM SIGPLAN Notices, 123–132.

13. Economou, Dimitris and Rivoire, Suzanne and Kozyrakis, Christos, Full-system power analysis and modeling for server environments, In 2006, In Workshop on Modeling Benchmarking and Simulation (MOBS).

14. Snowdon, D.C. and Le Sueur, E. and Petters, S.M. and Heiser, G., Koala: A platform for OS-level power management, In 2009, Proceedings of the fourth ACM European conference on computer systems, 289–302.

15. Nakku Kim, Jungwook Cho and Euiseong Seo, Energy-Based Accounting and Scheduling of Virtual Machines in a Cloud System, In 2011, Proceedings of the 2011 International Conference on Green Computing and Communications.

16. John C. McCullough and Yuvraj Agarwal and Jaideep Chandrashekar and Sathyanarayan Kuppuswamy and Alex C. Snoeren and Rajesh K. Gupta, Evaluating the Effectiveness of Model-Based Power Characterization, In 2011, Proceedings of the 2011 USENIX Annual Technical Conference.

17. Kansal, Aman and Zhao, Feng and Liu, Jie and Kothari, Nupur and Bhattacharya, Arka A., Virtual machine power metering and provisioning, In 2010, Proceedings of the first ACM symposium on Cloud computing, 39–50.

18. Neugebauer, R., and McAuley, D., Energy is just another resource: Energy accounting and energy pricing in the nemesis OS, In 2001, Hot Topics in Operating Systems, 67–72.

19. Bellosa, F., The benefits of event: driven energy accounting in power-sensitive systems, In 2000, ACM, 37–42

20. Vahdat, A. and Lebeck, A. and Ellis, C.S., Every joule is precious: The case for revisiting operating system design for energy efficiency, In 2000, Proceedings of the ninth workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system, 31–36.

21. Zeng, H. and Ellis, C.S. and Lebeck, A.R. and Vahdat, A., Currentcy: Unifying policies for resource management, In 2003, Proceedings of the USENIX 2003 Annual Technical Conference.

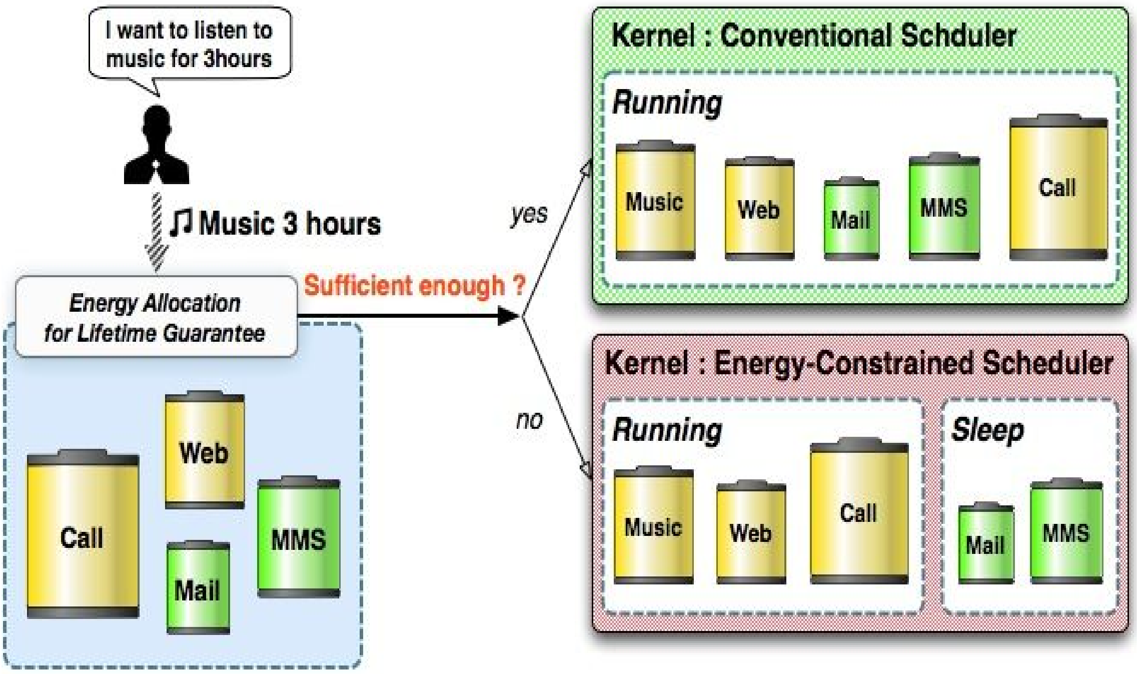
22. Banga, G. and Druschel, P. and Mogul, J.C. Resource containers: A new facility for resource management in server In 1998. Operating Systems Review, 45–58.

23. S. Rumble, R. Stutsman, P. Levis, D. Mazieres, and N. Zeldovich, Apprehending joule thieves with cinder, In 2010, ACM SIGCOMM Computer Communication Review, 106–111.

III. OUR APPROACH

The proposed system, shown in Figure 3-1, compares the monitored information such as the amount of energy each application consumes with the remaining battery charge in order to predict the available battery lifetime. When the lifetime requirement of each application cannot be guaranteed by the remaining battery charge, a signal is sent to the scheduler to restrict the energy consumption of all applications. Applications that have lifetime requirements are thus guaranteed by suspending applications that do not have lifetime requirements.

This section explains each component of the proposed system architecture and introduces the model used for estimating operation time and system modes in order to ascertain system status. In addition, the scheduler, which can limit energy consumption, is described without any degradation of quality of service (QoS) with respect to multimedia applications.



I want to listen to music for 3hours
♫ Music 3 hours
Energy Allocation for Lifetime Guarantee
Call
Web
Mail
MMS
Sufficient enough ?
yes
no
Kernel : Conventional Schduler
Running
Music
Web
Mail
MMS
Call
Kernel : Energy-Constrained Scheduler
Running
Music
Web
Call
Sleep
Mail
MMS

Figure 3-1. Conceptual diagram of our approach.

3.1 System Architecture

The suggested system is implemented on Google's Android platform. Android is a good open source software for research because users can easily modify and test proposals according to their purposes. In addition, Android can be considered to be a standard mobile system since it uses Linux and Dalvik Java Virtual Machine as its kernel and runtime environments. Thus, the system architecture proposed in this study is thought to be readily adaptable to other mobile operating systems.

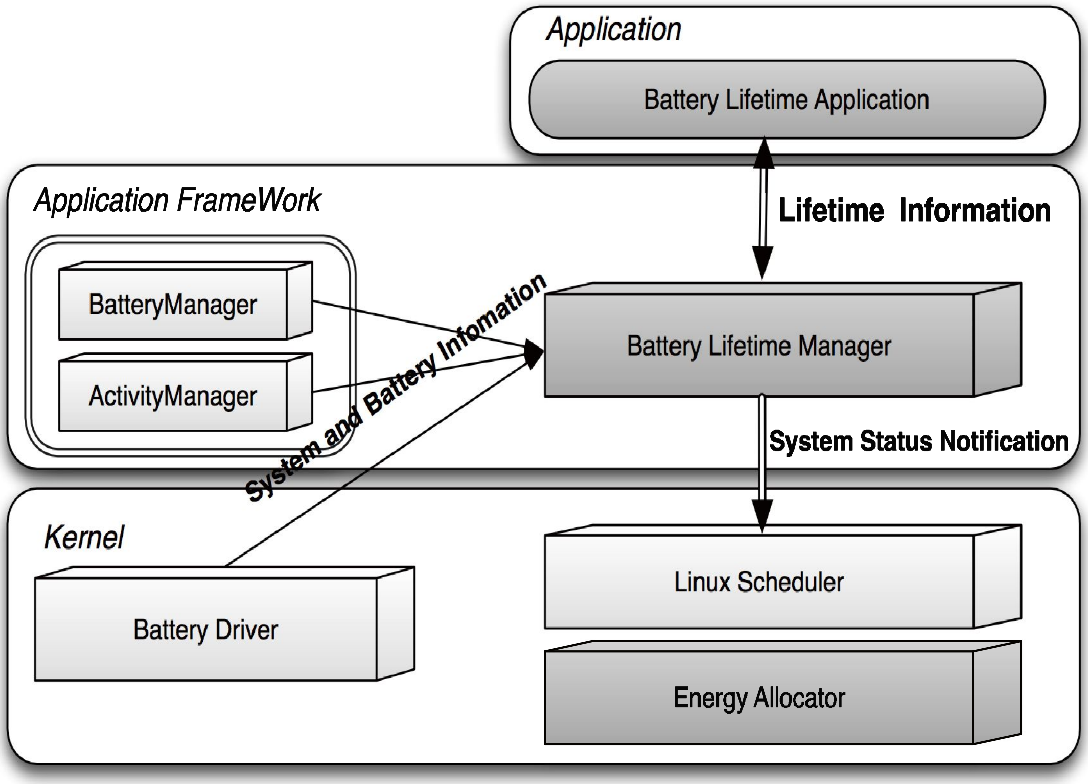


Figure 3-2. System architecture.

The system shown in Figure 3-2 is divided into three layers and seven parts. Of the components in the system, the Battery Manager, Activity Manager, Battery Driver and the Linux Kernel Scheduler are already included in the Android platform by default but we do propose extending their functionality by modifying them. The other components are newly proposed in this study.

All the managers in an application framework, which is similar to the UNIX system daemon, run as a service. The Battery Manager is responsible for monitoring the current battery status of the system. This service periodically monitors the remaining battery charge and battery temperature, which is obtained from the kernel, and analyzes the proportion of overall system energy consumption that each device accounts for and the amount of energy consumed by each application based on the usage of devices obtained and the scheduled time of application through the Battery Driver and Activity Manager.

The Activity Manager records how much time each application runs during a specific period, which can help analyze energy consumption per application. The Battery Driver is a kernel device driver that reads the remaining battery charge from the circuit built into the battery and provides this information to the application framework.

The Battery Lifetime application, like other applications, has graphical user interface (GUI) . Users can request the lifetime requirements of running applications and adjust or clear preset requirements. The lifetime requirement of each application is then delivered to the Battery Lifetime Manager in the application framework.

The Battery Lifetime Manager use the information from the Battery Manager, Activity Manager and Battery Driver to assess the current status of the system. The system can be classified into five energy management states as defined in section 3.2 and it has different strategies for each state. To this end, the Battery Lifetime Manager periodically monitors the amount of consumed energy from the Battery Driver and analyzes how much energy each application consumes based on the information delivered by the Battery Manager and Activity Manager. If the battery is quickly depleted so that the lifetime requirements of applications cannot be guaranteed, the Kernel Scheduler receives a signal to limit the use of energy by certain applications.

The Battery Lifetime Manager provides the Kernel Scheduler with the current system status and the battery usage of each application. This information is passed to the Energy Allocator. If limiting the use of energy is necessary to guarantee the lifetime requirement of each application, the Energy Allocator will determine how much time each application is scheduled for a certain period. This decision is then sent to the scheduler for scheduling.

Applications that do not require a lifetime are suspended if the remaining battery charge is insufficient. The energy-constrained scheduler is carried out to prevent the energy usage rates of the applications that require lifetimes from exceeding the current level. For more detail on energy-constrained scheduling, see section 3-4.

3.2 System Modes

The Battery Lifetime Manager divides the system into the following five energy management states according to the remaining battery charge, the battery lifetime required by each application and how quickly energy is being consumed by applications.

1) Normal: Applications currently running on the system do not require lifetimes or the system is recharging. In this state, there are no scheduling limitations.

2) Green: Some applications currently running on the system require lifetime guarantees but the intervention of the scheduler is not needed in response to the remaining battery charge and the rate of energy consumption. In other words, if the current system status remains, the lifetime requirements of applications will be guaranteed even if they are heavily scheduled.

3) Yellow: If the current system status is retained without limitations on battery use, the longest lifetime of an application cannot be guaranteed. In other words, using the energy-constrained scheduler can be postponed but it should end up guaranteeing the lifetimes of applications.

4) Red: The system is in the energy-constrained scheduling mode. An immediate scheduling limitation is required to guarantee the lifetimes of applications. Applications that do not have lifetime requirements are no longer scheduled and others are prohibited from exceeding the current level of energy consumption.

5) Black: Despite using energy-constrained scheduling, lifetime guarantees cannot be kept because applications that have lifetime requirements suddenly consume large amounts of energy or the user demanded lifetimes that were impossible to guarantee.

When considering an operating system, the Yellow state is the same as is the Green state. However, the Yellow state needs to give the user the signal to take action before entering the Red state because the execution of applications that do not have lifetime requirements will be limited.

When applications start to execute and finish spontaneously, the rate of energy utilization in the whole system is greatly changed. In addition, sudden increases or decreases in the rate of energy utilization can occur according to the characteristics of applications without changing the number of running application. Depending on these situations, the energy management states in the system can change (see Figure 3-3).

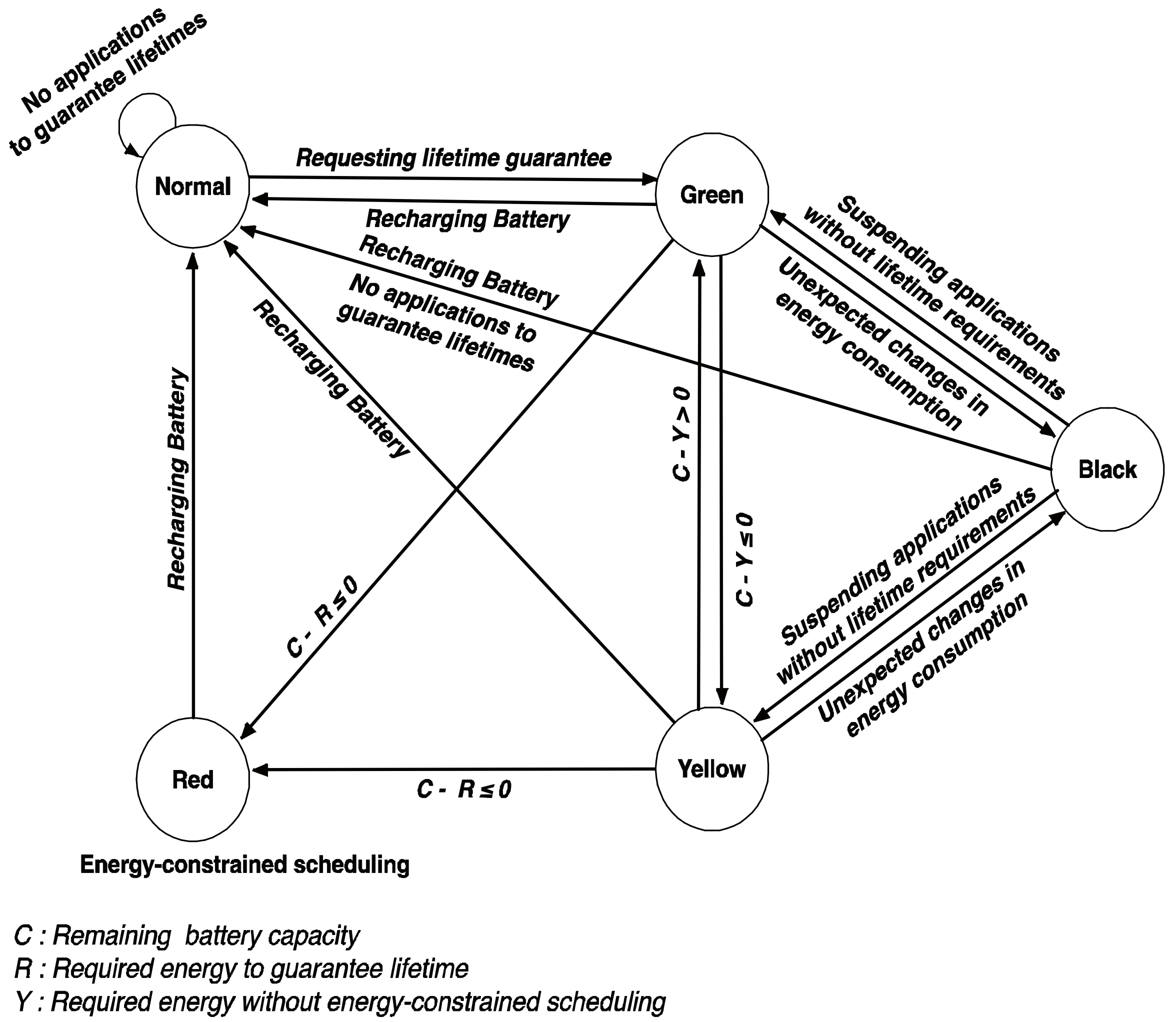

No applications to guarantee lifetimes
Normal
Requesting lifetime guarantee
Green
Recharging Battery
Recharging Battery
No applications to guarantee lifetimes
Suspending applications without lifetime requirements
Unexpected changes in energy consumption
Black
Recharging Battery
Recharging Battery
C - Y > 0
C - Y ≤ 0
C - R ≤ 0
Suspending applications without lifetime requirements
Unexpected changes in energy consumption
Red
Yellow
C - R ≤ 0
Energy-constrained scheduling
C : Remaining battery capacity
R : Required energy to guarantee lifetime
Y : Required energy without energy-constrained scheduling

Figure 3-3. Transition diagram of the five energy management states.

If the mobile phone recharges the battery in all states, the energy management state will switch to Normal. The energy management state will then switch to the Green state if the application requires its lifetime in the Normal state.

If the current rate of energy utilization remains and the longest lifetime of a user's required application cannot be guaranteed, the Green state will switch to the Yellow state. When the energy management state stays in the Yellow state, C is increasingly reduced and it finally enters the Red state before ensuring the longest lifetime requirement. When the energy management state switches to Red, the Battery Lifetime Manager tells the kernel to enter the energy-constrained scheduling mode.

If the difference between the values of C and R is low and the rate of the energy utilization of applications that have lifetime requirements rapidly increases, the Yellow or Green states may switch to the Black state. The Black state cannot guarantee their lifetime requirements in response to current system status. Thus, users should give up the lifetime guarantee of an application when the energy management state becomes Black. If surging the energy consumption of applications that do not have lifetime requirements threatens other applications, the energy management state will switch to the Red state and the energy consumption of the applications that have suddenly consumed a lot of energy will be restricted. Therefore, switching to the Black state does not happen except when there is a surge in the energy consumptions of applications that have lifetime requirements.

Once the system goes into the Red state, energy management state does not change until the battery is recharged or there are no applications that have lifetime requirements. Maintaining the Red state aims to prevent the excessive consumption of energy when a temporary decrease in energy consumption is allowed in order to exit from the Red state. Therefore, the energy consumption rates of applications should be monitored for enough long to avoid short-term rate increases.

A variety of system services exists to provide applications with a wide range of services and manage the system. Although applications receive the appropriate degree of the processor time, applications that use the system services will not run properly if these services do not run properly. If the energy management state is not Normal, we assume that the system services have the same lifetime requirements as the application that required the longest lifetime guarantee. Using this method, system services continue to run as long as the system is operating.

3.3 Lifetime Estimation

The amount of energy that each application consumes is obtained from the Battery Manager provided by Android, which keeps track of the approximate system status, but does not aim to predict battery lifetimes accurately. Therefore, in order to meet these lifetime requirements, we herein introduce a new evaluation model in order to calculate the energy usage rate for each application.

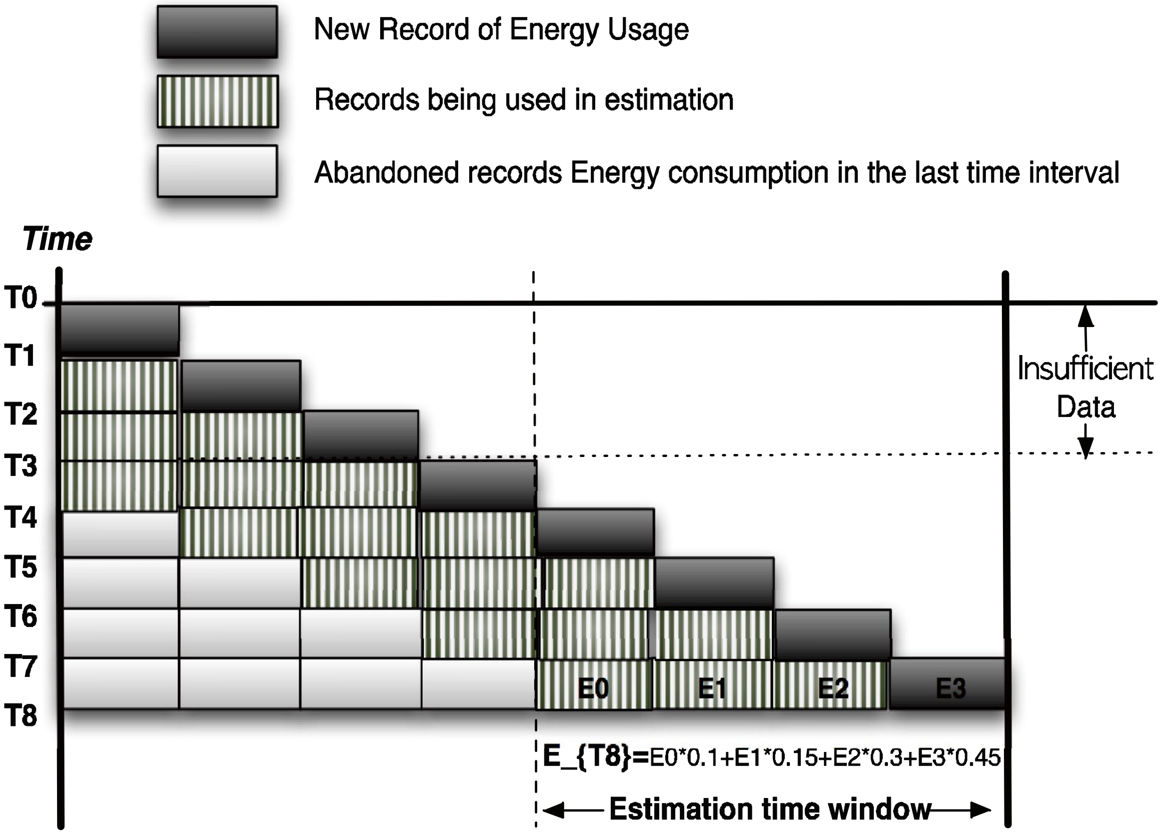

New Record of Energy Usage
Records being used in estimation
Abandoned records Energy consumption in the last time interval
Time
T0
T1
T2
T3
T4
T5
T6
T7
T8
Insufficient Data
E0
E1
E2
E3
E_{T8}=E0*0.1+E1*0.15+E2*0.3+E3*0.45
Estimation time window

Figure 3-4. Profile with Estimation time window.

The Activity Manager reads the runtimes of each application from the proc filesystem. Based on this information and the amount of devices used by each application, the Battery Manager is able to calculate the proportion of energy consumed by a particular application over a certain period. Most applications consist of multiple threads. In Android, the threads in one application have the same UID. Therefore, we consider the energy consumed by threads with the same UID to calculate the energy consumption of the application.

For a more sophisticated analysis, the Battery Lifetime Manager could ask the Activity Manager to update recent information on the runtimes of applications. In our prototype, we used the variable of 1 minute to analyze scheduling information and the energy consumption of each application. The battery in the Nexus One takes advantage of the implementation of a prototype to update the registers storing information on the battery every 3.5 s with a minimum draining battery capacity of 1.6 mAh. Therefore, an overly short sampling period does not improve accuracy and can increase overheads.

Sudden changes in energy consumption may happen depending on the characteristics of applications. If the energy management state is decided by a temporary surge in energy usage, it leads to an overly fast transition to the Red state and thus the applications that do not have lifetime requirements can no longer be executed. This temporary increase in energy consumption does not cause a transition to the Red state. From diverse experiments, we verified that calculating the amount of energy consumption every minute is sufficient to accurately guarantee the lifetime for tens of minutes, as shown in Figure 3-4. Of course, the monitoring period and the time window can be changed depending on the characteristics of the system.

When an application that has a lifetime requirement is executed for the first time, whether the lifetime requirement is guarantee or not is delayed for filling the estimated time window (i.e. in our case is 4 minutes). Even though the user requires a lifetime guarantee for the application, the energy management state will not change for 4 minutes. If the rate of energy utilization is obtained after 4 minutes and the energy management state switches to Black, the Battery Lifetime Manager should notify the user that guaranteeing the lifetime requirement is impossible and thus deny the user's request.

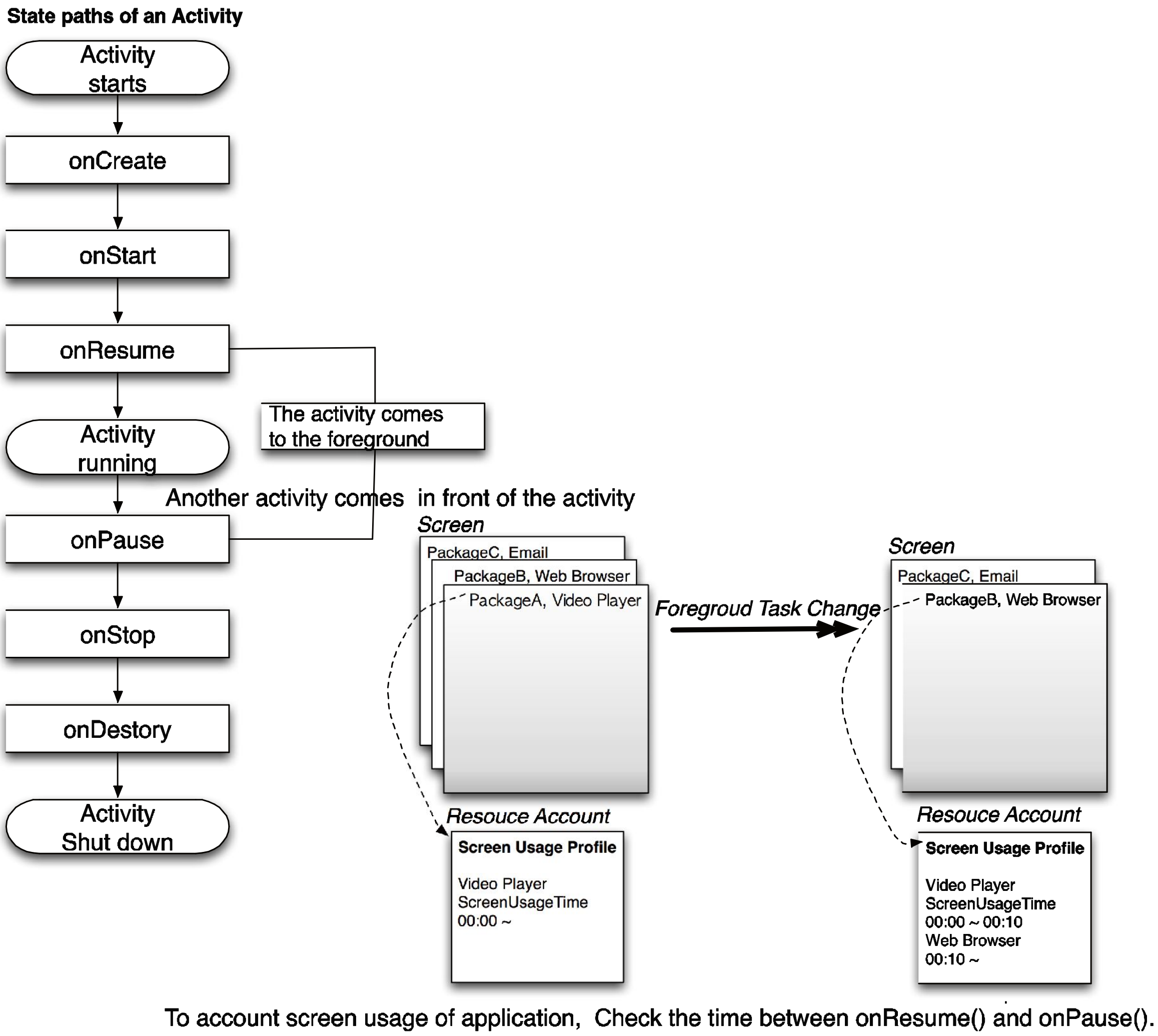


Figure 3-5. Mobile display activity.

We use the energy consumption model introduced in section 2-1 to measure the amount of energy that applications consume. However, in its calculation of total energy, the default Android Battery Manager excludes the amount of energy consumed by the display, which is a large proportion of the total energy consumption. Thus, the amount of energy that the display of each application consumes should be included. The current system does not provide the proportion of display usage for each application. Therefore, we modified Android's application framework to measure the amount of screen usage for each application. The applications with GUI have a life cycle (Figure 3-5). These applications use the screen when it is in between the onResume and onPause states; thus, by recording this time we can measure the amount of screen usage for each application.

3.4 Energy-constrained Schedule

When energy management state is Green or Yellow, the system should reserve sufficient energy for applications that have lifetime requirements only if other applications can run with surplus energy (see Figure 3-6).

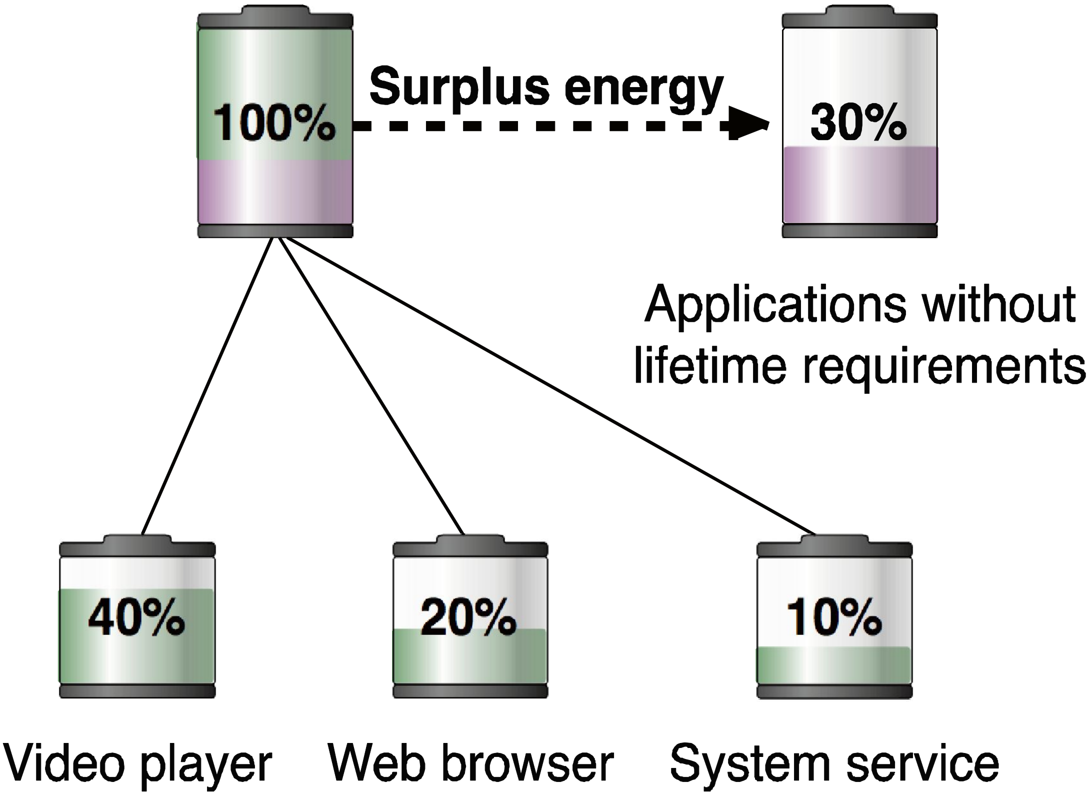


Figure 3-6. Resource allocation.

When the energy management state switches to Red after this surplus energy has been exhausted, the Kernel Scheduler begins working in the energy-constrained scheduling mode. The energy-constrained mode makes two major changes. First, the applications that do not have lifetime requirements are no longer scheduled. Second, the applications that have lifetime requirements are prevented from suddenly consuming an excessive amount of energy.

The scheduler prepares one freezing queue in the energy-constrained scheduling mode. The applications that are unscheduled because of energy constraints are inserted into this queue. When the energy management state gets out of Red, applications in the freezing queue are released and the normal schedule begins.

In the Red state, the scheduler limits the amount of energy each application consumes in each ‘fiscal interval’, which is a short period of tens of seconds.

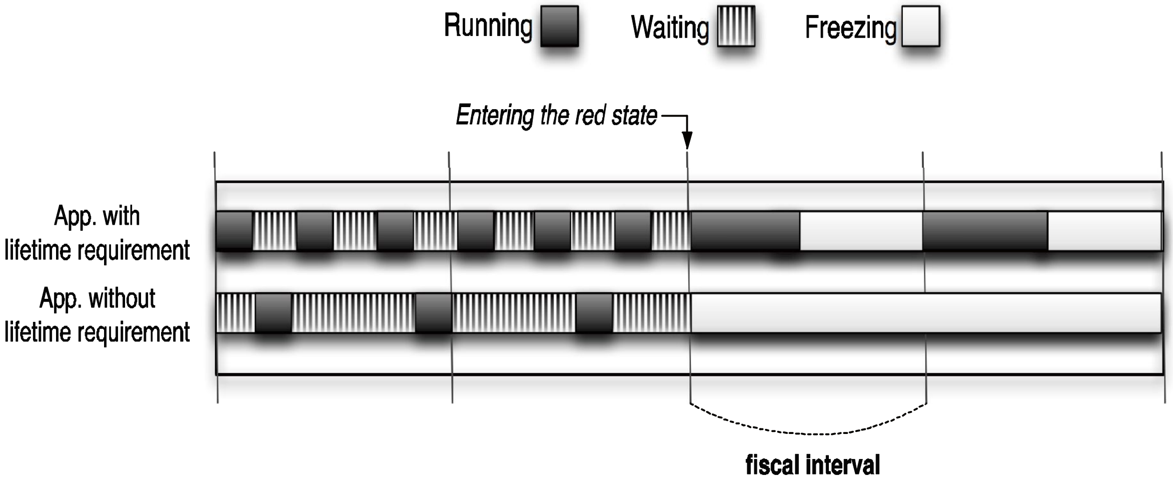
When the energy management state switches to Red, the Energy Allocator calculates the average energy usage of applications that have lifetime requirements during a fiscal interval by dividing the reserved energy for the applications into the remaining lifetime of each application. Thus, the scheduler prevents applications that have used more than their given energy quota during a fiscal interval from further scheduling by putting them into the waiting queue.

After a fiscal interval, the energy that is available to be consumed by applications in the waiting queue during a new fiscal interval is reassigned to those applications that have lifetime requirements by the Energy Allocator. The applications that receive the energy will continue rescheduling until their given energy quota runs out.

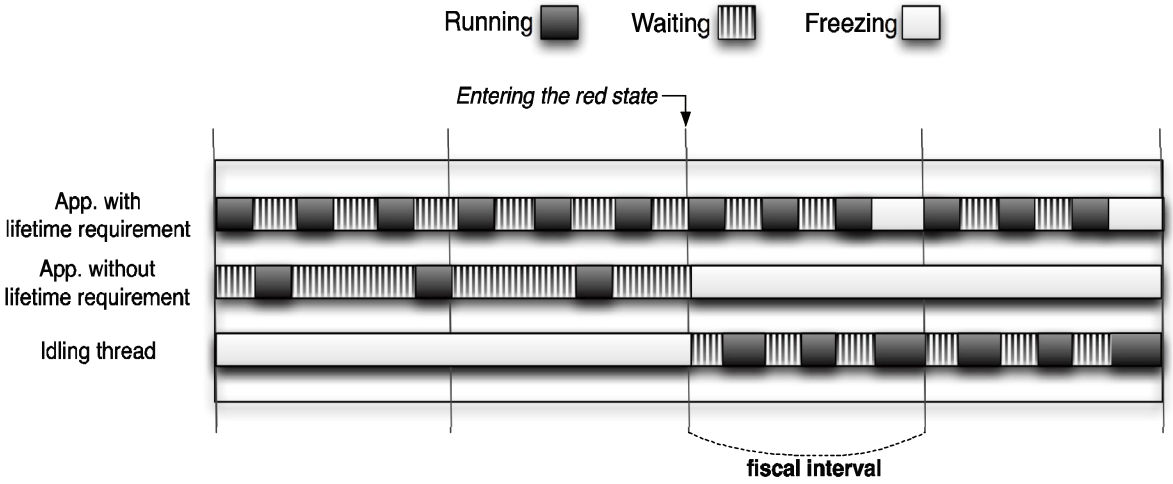
Monitoring the energy consumed by applications during a fiscal interval in real-time leads to significant overheads. Therefore, after entering the Red state, the Energy Allocator notifies the scheduler of the available scheduling time of each application during every fiscal interval, which is calculated based on the energy consumption characteristics of each application. These calculations are made only once at the entry point of the Red state. The ensuing overheads can be significantly reduced because the scheduler sets the time instead of the energy as the main criteria for scheduling.

If the given time for an application cannot be spent during a fiscal interval, the remaining time will carry over to the next fiscal interval.

We discovered from our experiments that this scheduling method causes multimedia applications such as movies and games to seriously degrade QoS in the energy-constrained scheduling mode, even though they receive the same amount of processor time as in Green or Yellow. For example, we confirmed that the frames per second (FPS) of the video player seriously drops despite receiving the same amount of processor time.


Running
Waiting
Freezing
Entering the red state
App. with lifetime requirement
App. without lifetime requirement
fiscal interval

(a) Without compulsory idle spreading.



(b) With compulsory idle spreading

Figure 3-7. Energy-constrained scheduling changes the scheduling patterns of applications.

As shown in Figure 3-7(a), QoS degradation occurs in the energy-constrained scheduling mode because some applications that do not have lifetime requirements are frozen, whereas others are quickly scheduled in succession. Multimedia applications spend their given time at the beginning of the fiscal interval; they are put into the waiting queue and stop running for the remaining fiscal interval. At this time, QoS degradation occurs, such as not responding to user inputs or stopping playing videos.

Multimedia players periodically render frames using processor resources and then use any remaining processor resources for data buffering. When there are surplus processor resources, they keep filling the buffer after rendering frames. Consequently, after the given resources for buffering run out, they are unable to be scheduled before the beginning of the next fiscal interval. We confirmed that the buffered data are not actually used for rendering frames and thus they were discarded from our experiments.

In the energy-constrained scheduling mode, in order to prevent this QoS degradation, the pattern of scheduling shown in Figure 3-7(b) should be forced to provide the same experiences, such as running other applications together or using a low-performance processor. To this end, we suggest compulsory idle spreading.

The proposed method attempts to keep the scheduling patterns of running multimedia applications after they enter the energy-constrained scheduling mode by executing the kernel thread as much processor time as the frozen applications are allocated and leading to competition between the kernel thread and applications that have lifetime requirements for the processor’s resources

To this end, the kernel thread not only competes for the processor’s resources but also minimizes additional energy consumption by actually running the idle loop. Using this method, we expect the application to avoid QoS degradation because of the change in scheduling patterns in the energy-constrained scheduling mode.

IV. Evaluation

4.1 Evaluation Environment

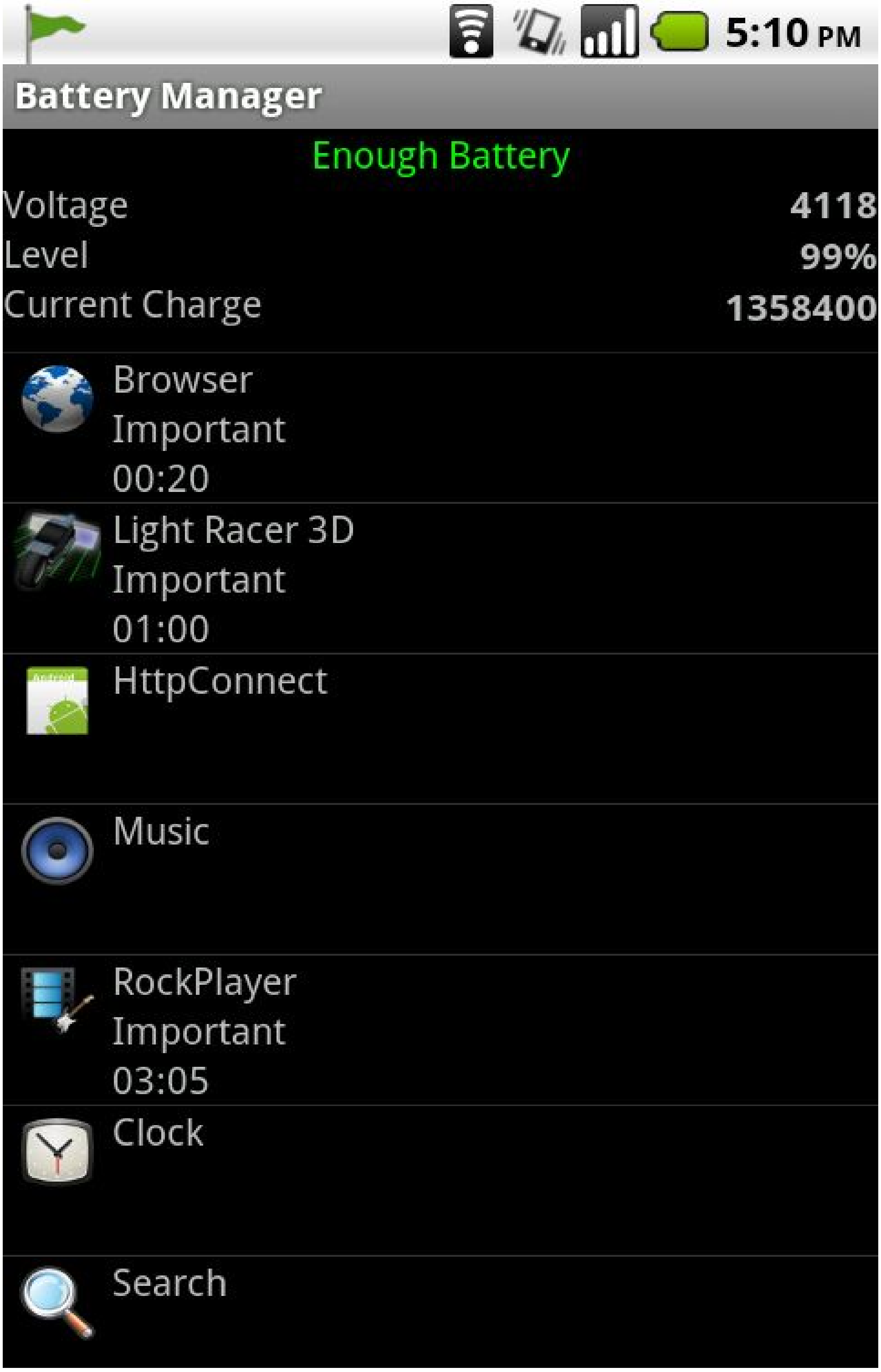
The proposed system was implemented in the Android 2.2 Froyo System and the implemented system was ported to a commercial smartphone, Google Nexus One, for evaluation.

The smartphone displays the current energy management state in its notification area, as shown in Figure 4-1. The user can see a list of currently running applications and asks for the lifetime of each application using the Battery Lifetime Application. To enable evaluation, all Android's default services were kept running and the applications described in Table 4-1 were used for the experiment.

We proved the accuracy of the lifetime guarantee by running several applications at the same time and setting the lifetimes of some applications among them. In addition, the effect of compulsory idle spreading was evaluated by measuring the number of FPS when the media player played a video.

Name	Description	Necessary Resources
Movie	Movie player with Rock Player, a media player app.	Processor, Display and Audio
Game	Automated play of a 3D game, Light Racer.	Processor, Display and Audio
Web	Visiting 16 sites randomly with random thinking times from 500 ms to 13 s.	Processor, Display and Wi-Fi
Music	Playing an MP3 file using Android's Music Player.	Processor and Audio
Socket	Downloading small files from 420 bytes to 4200 bytes with random pause times from 1 s to 10 s.	Processor and Wi-Fi
Loop	Repeating simple calculations continually.	Processor
Random	Repeating simple calculations for random time intervals from 1 s to 10 s. with random pause times from 1 s to 10 s.	Processor

Table 4-1. Applications used in the evaluation.



5:10 PM

Battery Manager

Enough Battery

Voltage	4118
Level	99%
Current Charge	1358400

Browser
Important
00:20

Light Racer 3D
Important
01:00

HttpConnect

Music

RockPlayer
Important
03:05

Clock

Search

Figure 4-1. A screenshot of the Battery Manager Application (the green flag in the upper left-hand corner shows that the device is in the Green state).

4.2 Lifetime Guarantee Accuracy

Figure 4-2 shows the proportion of energy consumed in each energy management state after guaranteeing the lifetime requirement of each application when one of the three applications required a lifetime guarantee. When any application did not require a lifetime guarantee, the available time of the system was approximately two hours. In our experiments, we required more than two hours to judge the accuracy of the lifetime guarantee.

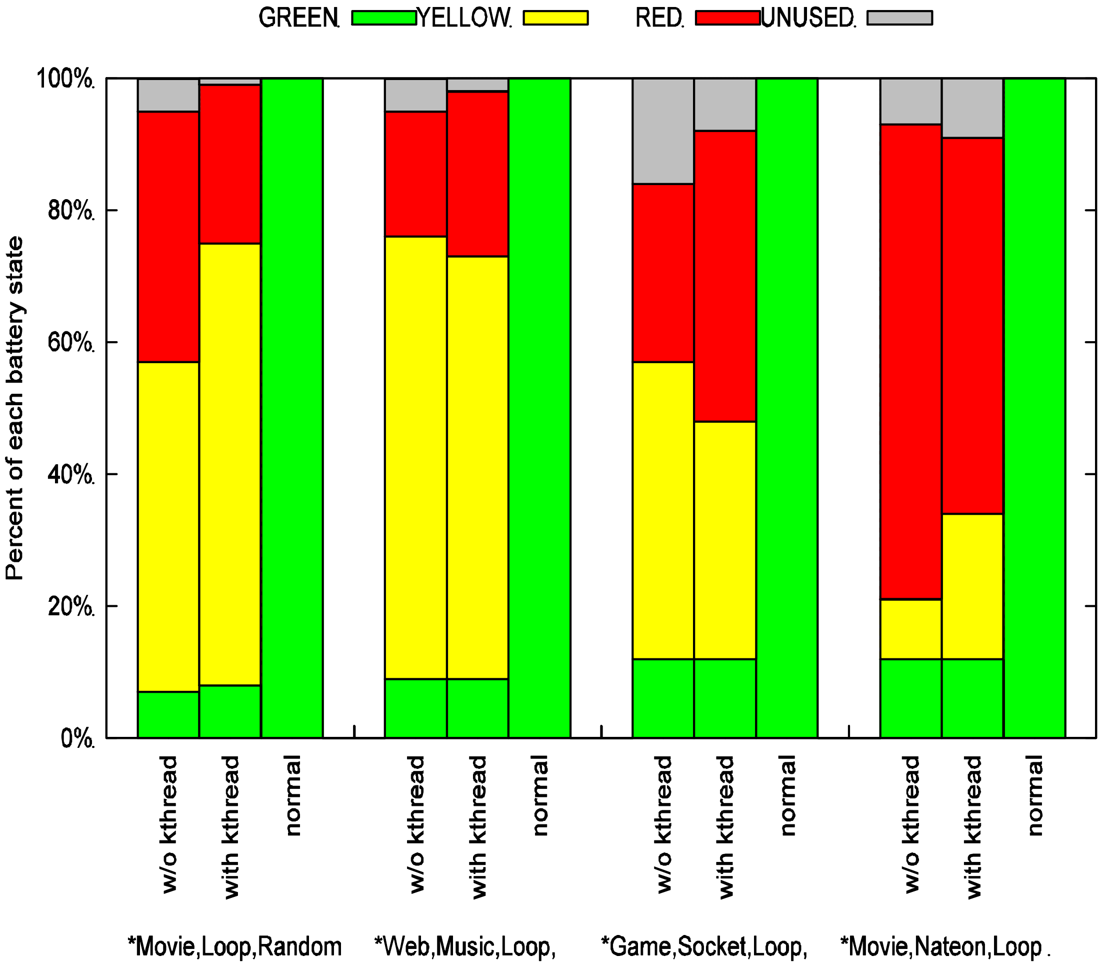


Figure 4-2. Accuracy of the proposed method when only one application requests a lifetime guarantee (applications marked by asterisks have lifetime requirements).

The criteria 'unused' means that the remaining battery charge after the lifetime requirements are guaranteed. There are three cases for this:

1) Insufficient energy (Unused < 0): the lifetime guarantee of applications failed.

2) No unused energy (Unused = 0): all the lifetime requirements are guaranteed and all surplus energy was used for applications that do not have lifetime requirements.

3) Remaining energy (Unused > 0): all lifetime requirements are guaranteed but some surplus energy unnecessarily remained. In other words, energy management is so conservative that the applications that do not have lifetime requirements can no longer run in the Red state even though there was sufficient energy.

The ideal case is no unused energy.

At the end of our experiments, there was an amount of unused energy but this amount was different following every experiment. Such prediction errors occur because the number of context switching and the number of timer interrupts in the Red state are lower than those in other states. Overheads could be reduced by reducing these factors. The recent operating system entered the tickless kernel mode (Siddha et al., 2007), which does not call timer interrupts when the system is idle. Therefore, the number of timer interrupts is significantly reduced after applications that do not have lifetime requirements are frozen after entering the Red state. As a result, the unused energy remained after the battery lifetime guarantee by reducing the amount of consumed energy.

This tendency was observed more strongly when compulsory idle spreading, which forces the idle thread to run, is not used. As shown in Figure 4-2, the proportion of unused energy increases more by not using compulsory idle spreading.

By entering the Red state earlier than the ideal case the execution of applications do not require a lifetime are affected to run, but in every experiment, we can confirm the battery lifetime of each application is guaranteed.

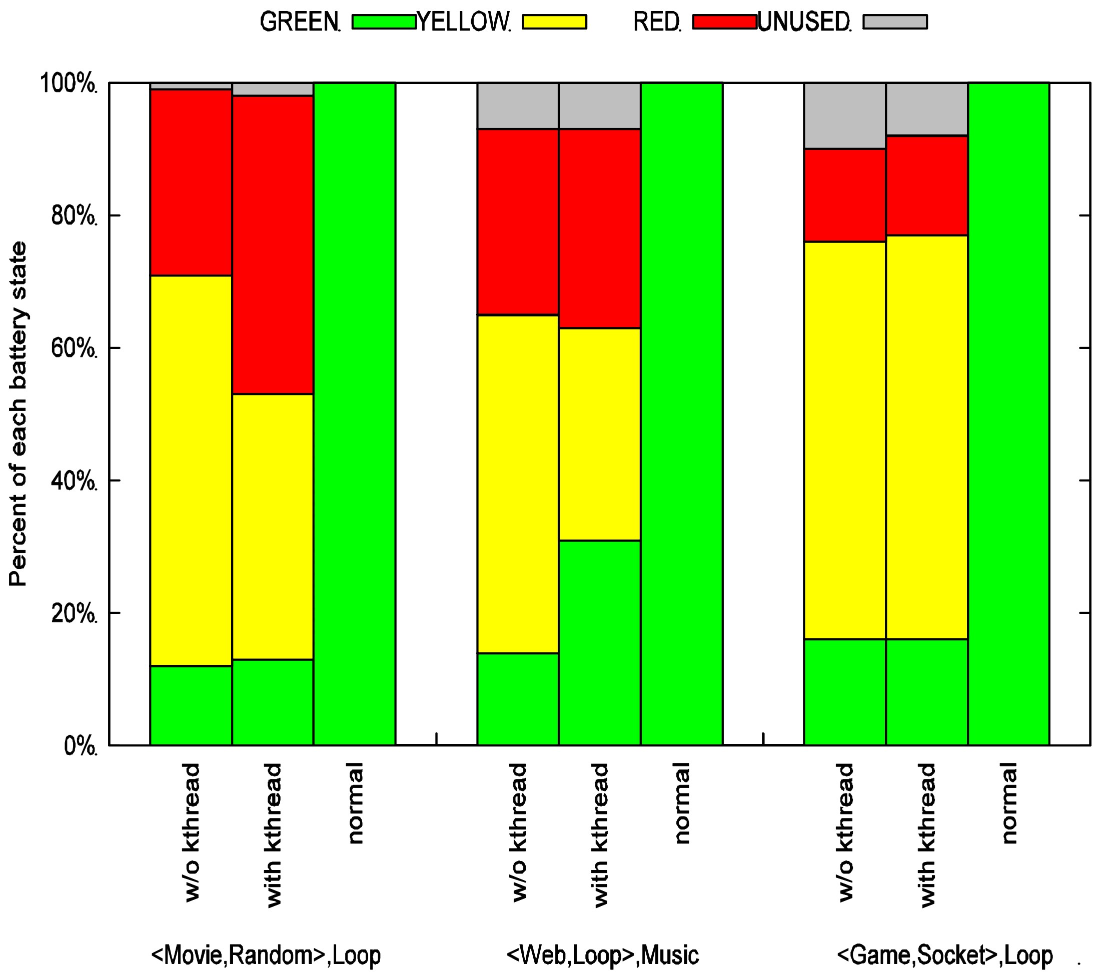


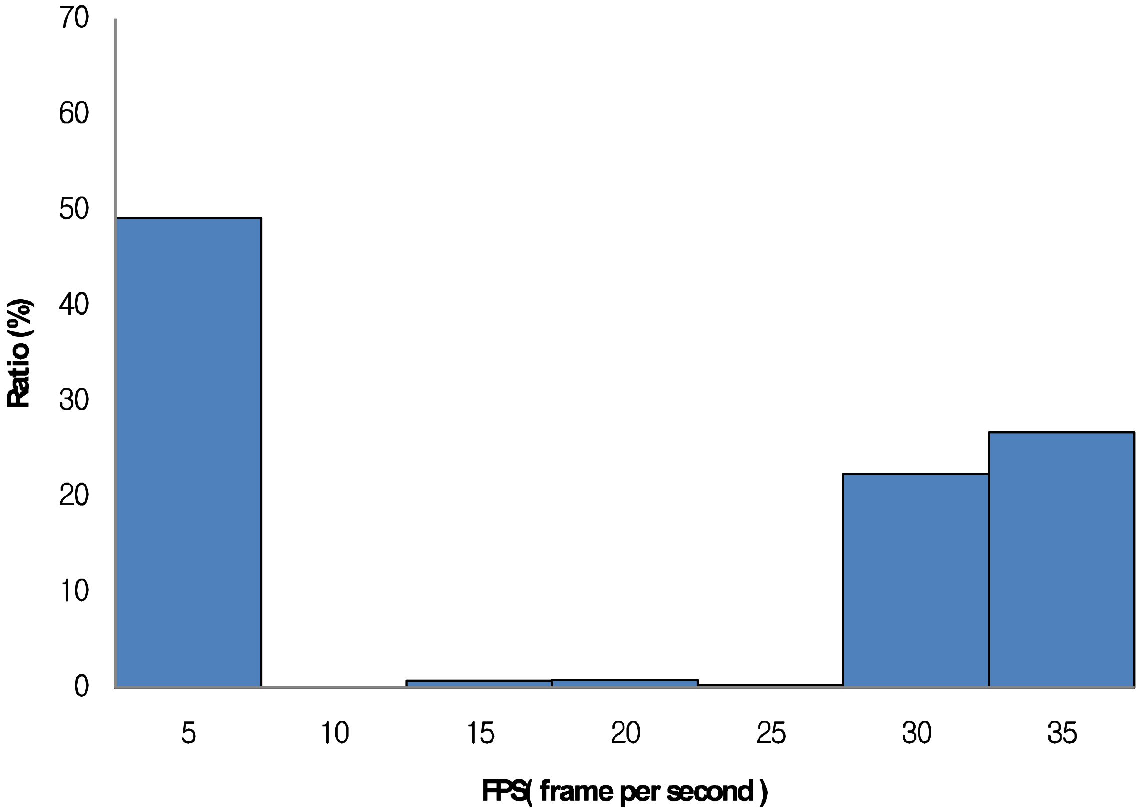
Figure 4-3. Accuracy of the proposed method when only two applications request a lifetime guarantee (applications in brackets have lifetime requirements).

These characteristics are found when multiple applications require a lifetime guarantee at the same time. Figure 4-3 shows the results of our experiment when two of the three applications required a lifetime guarantee. In all cases, the proportion of unused energy ranged from 1% to 10%.

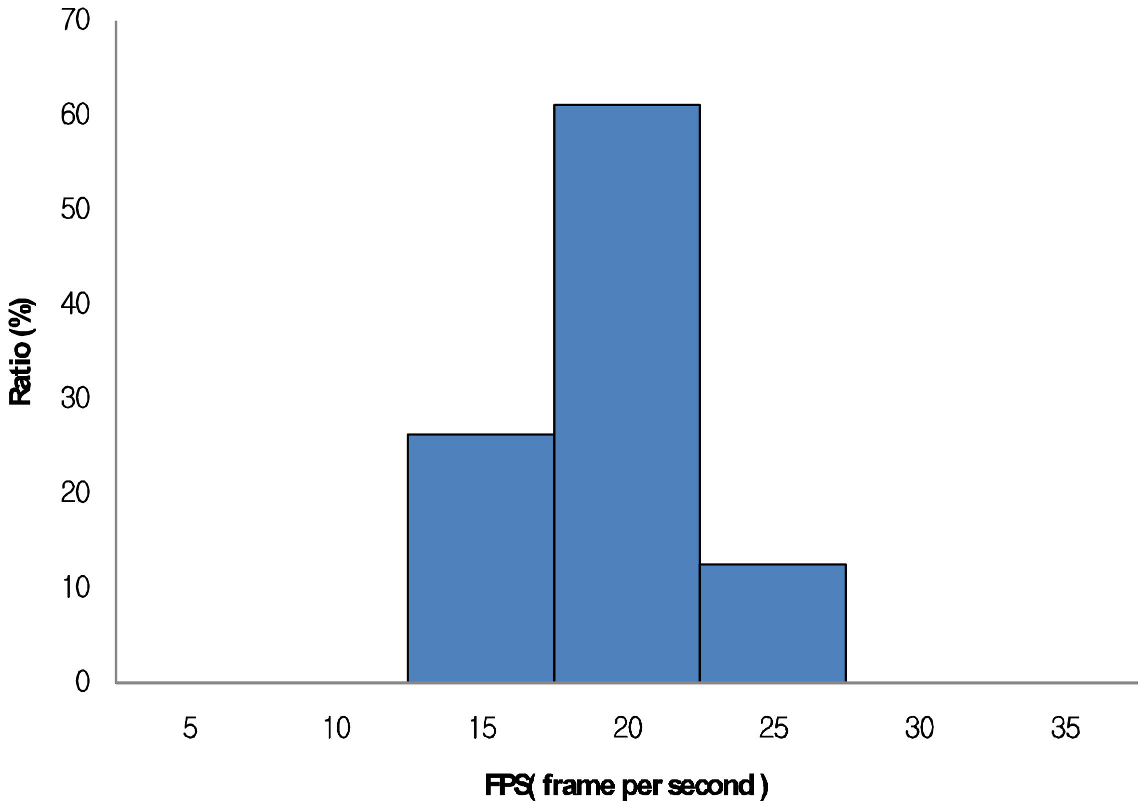
Therefore, to use energy more aggressively, we should introduce a method that postpones entering the Red state by considering the surplus energy caused by frozen applications. However, the chance of failing to guarantee the lifetime requirements is likely to be higher because of the incorrect predictions.

4.3 Quality of Multimedia Applications

To evaluate the effect of compulsory idle spreading, we composed a tool to record the change in the degree of FPS of the video player. In our experiments, we ran a video player with background applications that require 50~60% of the processor's resources. At the end of the profile period, we forced entry into the Red state. In conclusion, the system allowed the video player to use 40~50% of the processor's resources, which was not enough to play the video. Therefore, the FPS were degraded. Furthermore, we extended the length of a fiscal interval in the energy-constrained scheduling mode to two seconds to effectively show the change in FPS value.



(a) Scheduling without compulsory idle spreading



(b) Scheduling with compulsory idle spreading

Figure 4-4. Distribution of FPS values under energy-constrained scheduling.

Figure 4-4 shows the distribution of FPS values when the video player played for one hour in the Red state. As expected, without compulsory idle spreading, the FPS showed two extreme value distributions. The video player showed fast movement over 30 FPS for the first second by using its given energy, while it showed a FPS value below 5 for the other second because its given energy had been exhausted in the first second.

This situation becomes more severe because the video player buffers the data in the background. The given energy budget is used for rendering on screen and the remaining energy budget is used for buffering. Therefore, the energy budget is quickly exhausted, and eventually frames stored in the buffer cannot not be drawn on time.

In terms of sound, more severe losses occur. Without compulsory idle spreading, the user cannot hear the sound properly because it stutters. However, we noted that the number of jitters significantly reduced when compulsory idle spreading was used.

REFERENCES

1. Siddha, S. and Pallipadi, V. and Ven, A. V. D., Getting maximum mileage out of tickles, In 2007, Proceedings of the Linux Symposium, 201–208.

V. Conclusion

In this paper, we suggest a scheduling method that guarantees the battery lifetimes of chosen applications in a mobile system. In contrast to existing studies, the proposed method not only guarantees the battery lifetime requirement of each application but also requires no changes to existing applications. In addition, running applications do not need to consider the amount of energy they consume or the remaining battery charge. In addition, we show that multimedia applications may cause the loss of QoS by scheduling a limited energy consumption. To solve this problem, we suggest energy-constrained scheduling algorithms that can conserve scheduling patterns by preventing the degradation of QoS. To the best of our knowledge, this is the first study to point out the reduced quality of services with respect to multimedia applications because of changes in scheduling patterns.

Our experiments show 1%~10% positive errors with respect to lifetime guarantee but no negative errors. In other words, battery lifetime is guaranteed but excessive energy conservation could not be forced on applications that do not have lifetime requirements. In addition, we confirm that maintaining scheduling patterns by force prevents the significant degradation of service quality in the case of multimedia applications when energy consumption is limited.

The use of smartphones is growing and the number of simultaneously running applications is increasing through multitasking. The proposed approach of deliberately allocating energy resources to applications improves reliability by guaranteeing the battery lifetimes of applications. In addition, the approach proposed in this study can be utilized for mobile devices used for military, surveillance and scientific research as well as for smartphones. Our approach is likely to help these mobile systems achieve their diverse mission-critical jobs by guaranteeing the lifetime of each mission.

The proposed method is only effective for applications that have steady energy consumption patterns. Applications that have changeable energy consumption patterns (e.g. web browsers) should be profiled for a longer time. Thus, we are conducting further research to minimize the loss of QoS and guarantee battery lifetime by using different profiling methods according to the characteristics of the applications under study.

Acknowledgements

I would never have been able to finish my dissertation without the guidance of my committee members, help from friends, and support from my family.

I am deeply grateful advisor, Professor Euiseong Seo who gave the opportunity to be his first student. I was able to finish writing my dissertation because of his guidance and unwavering support. Above all, I really appreciate he provided me with a chance to conduct project in my areas of interest.

Besides my advisor, I would like to thank my committee members, Prof. Jongeun Lee and Younho Lee. They gave me their encouragement and insightful comments with respect to my dissertation.

I would like to thank members in CSL, Nakku Kim, Youngjoo Woo, Moohyeon Nam, Suntae Kim, Daeyoon Jin, Woohyuk Choi and other students who have developed LMS. They were always willing to help and give their best suggestions. Especially, Youngjoo and Suntae helped me complete my dissertation.

I would like to thank other graduate and undergraduate students I met in UNIST. I would have been lonely without them.

Finally, I would also like to thank my parents and sister. They were always supporting me and encouraging me with their best wishes. They were always cheering me up and stood by me through the good times and bad.

I'll do my best in return for their care and love.