Chulhong Min *KAIST, Daejeon, South Korea* **Youngki Lee** *Singapore Management University, Singapore*
Chungkuk Yoo *KAIST, Daejeon, South Korea* **Seungwoo Kang** *KOREATECH, Cheonan, South Korea*
Inseok Hwang *IBM Research, Austin, TX, USA* **Junehwa Song** *KAIST, Daejeon, South Korea*

Editors: Robin Kravets and Nic Lane

PowerForecaster: Predicting Power Impact of Mobile Sensing Applications at Pre-Installation Time

Excerpted from "PowerForecaster: Predicting Smartphone Power Impact of Continuous Sensing Applications at Pre-Installation Time" from *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems* with permission. http://dx.doi.org/10.1145/2809695.2809728 © ACM 2015

Mobile application markets are important for users to select desirable applications to users' needs; they provide diverse information, such as features, screenshots, and user comments. However, they still miss a key information, *power consumption* by an application. Users recognize whether an application is power hungry or not, only after they install and use it. They usually have no choice but to count on such trial and error to decide if the application is worth such power consumption. Emerging mobile sensing applications [2][3][4], however, make such users' practices of "use it a while and see what happens to battery" no longer effective. Mobile sensing applications continuously drain battery in the background without users' explicit awareness. For example, Accupedo, a commercial pedometer application, reduces the phone's battery life by hours, causing the phone to be shut down early [6].

What if an application market provided a sensing application's estimated power use? It will help users make informed decisions even prior to installing the applications. Users are relieved of exhaustive trial and error to install an application, and may be less embarrassed with rapid battery drain which was already expected.

It is not straightforward to realize such a function. While there have been extensive efforts to build accurate models for power consumption of mobile applications [1][8][10], they can provide power information only after running the applications and observing their power behaviors. A trivial way to provide power information prior to installing an application is for developers to post the average power of the application for common-use cases. We notice that such average power information is neither accurate nor useful for many individual users. The main reason is that the sensing application's power consumption largely deviates from one user to another, depending on each user's physical activities, phone usage, and environmental factors. Such differences are amplified when an application applies various optimization techniques. Our previous study showed that users' mobility patterns affect battery drain of commercial sensing applications significantly [7]. We also observed that active sensing duration of an application varies up to three times with 27 people's sensor traces collected over three weeks [6]. For example, a location tracking application caused nontrivial differences of its daily GPS activation time per user, which ranges from 0.7 to 2.4 hours (mean: 1.4, stdev: 0.4). It mainly attributes to the optimization technique that activates GPS only when a user is moving.

In our earlier paper [6], we introduced PowerForecaster, a system to provide an *instant, personalized* power estimation of sensing applications at *pre-installation* time. In this article, we give an overview of the system and present the key results. Figure 1 shows mockup screenshots of application markets when PowerForecaster is integrated. PowerForecaster provides unique user experiences. First, it provides users with power estimation before installing applications, removing the hassles of trying and uninstalling applications one after another until users find satisfactory power-efficient applications. Second, estimation is highly personalized and thereby more accurate as an individual user's physical activities and phone usage are major input variables upon which predictions are

Photo, istockphoto.com

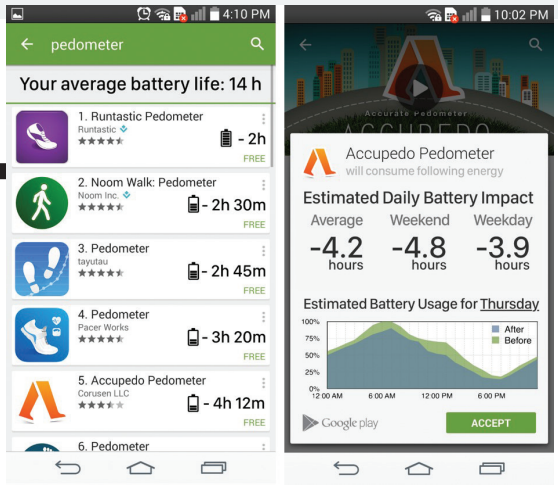


FIGURE 1. Power impact at pre-installation time.

made. Third, the system neither requires any changes of the application binaries nor additional information from developers.

We devise a *personalized user behavior-aware power emulation* approach. The key idea is to reproduce individual users' execution environment for a target application to track its power use. The execution environment is reproduced by replaying pre-collected personal traces of sensor and device usages. The approach has three advantages: (1) It accounts for major user-dependent variables causing large power consumption deviations – physical activities and phone usage. (2) It estimates power use of arbitrary applications exactly as they are on the application market; the emulator tracks hardware use of the applications by running its executable. (3) It considers shared hardware use with existing applications by reproducing their resource use.

We further optimize PowerForecaster to speed up emulation and reduce power overhead for trace collection. First, PowerForecaster accelerates emulation by fast-forwarding replays to capture changes in the power states of hardware, while parallelizing replays on multiple emulator instances. For responsive service, it also progressively updates power estimation with interim results. Our evaluation shows that PowerForecaster emulates 18-hour long traces within 30 seconds at 6 to 7% power estimation errors. Second, it employs a balanced duty-cycling policy to minimize data necessary for power estimation while keeping its accuracy high. We found that, even with the data collected at the ratio of 1/12, the estimation accuracy still remains above 90%.

PowerForecaster OVERVIEW

PowerForecaster takes a target application's executable as an input and estimates its power impact, i.e., net power increase by the application as an output. Figure 2 shows the architecture overview. Prior to power estimation requests, the mobile-side collector pre-collects user behavior traces. Upon a request, the cloud-side power emulator estimates power impact of a target application with these traces.

Mobile-side components collect user behavior traces in the background. The *sensor trace collector* logs sensor data that captures users' physical activities. The *device usage trace collector* logs hardware component usages by existing applications, which are potentially sharable with a new sensing application. Once collected, the traces are reused for various sensing applications.

Cloud-side components run a target application and estimate its power use. The *user behavior-aware power emulator* executes the application and monitors its hardware usage. Simultaneously, it replays the sensor and device usage traces to reproduce the personalized power-related execution environment. As a result, the emulator obtains detailed hardware usage statistics including which, when, and how long hardware components are used. The *power impact estimator* computes the net power increase of the application based on the cumulative hardware usage statistics and a system call-based power model [10].

USER BEHAVIOR-AWARE POWER EMULATION

Existing mobile emulators, such as Android emulator and iOS simulator, are not feasible solutions to emulate the power use of the sensing applications. They do not support power monitoring during emulation and cannot reproduce users' behavioral factors, which are essential for personalized power estimation.

To address the issues, we developed a novel *user behavior-aware power emulator* based on Android emulator (Figure 2). Overall operation flows are as follows.

Pre-emulation: Upon a request, the *power emulator* prepares necessary inputs for emulation. It first obtains sensor and device usage traces from the *user trace manager* as

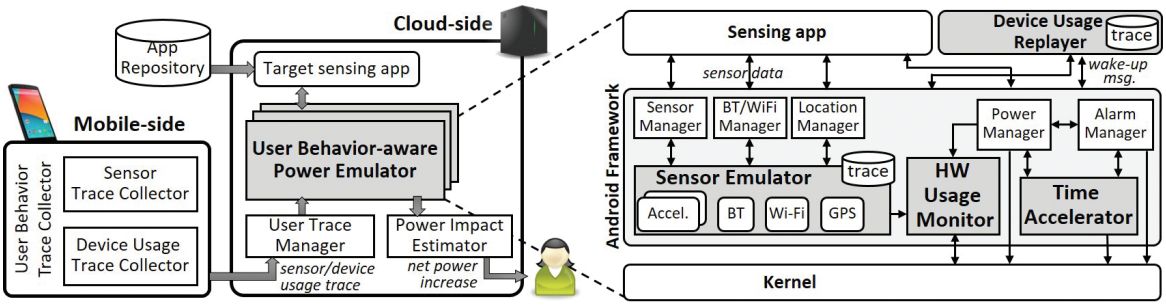
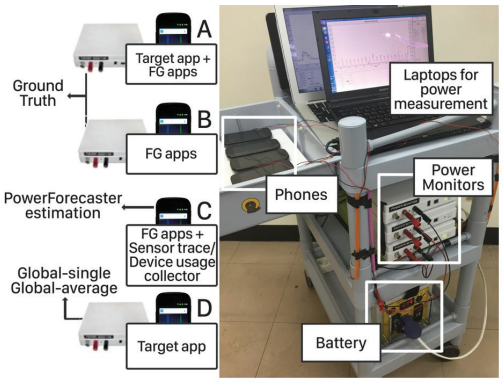
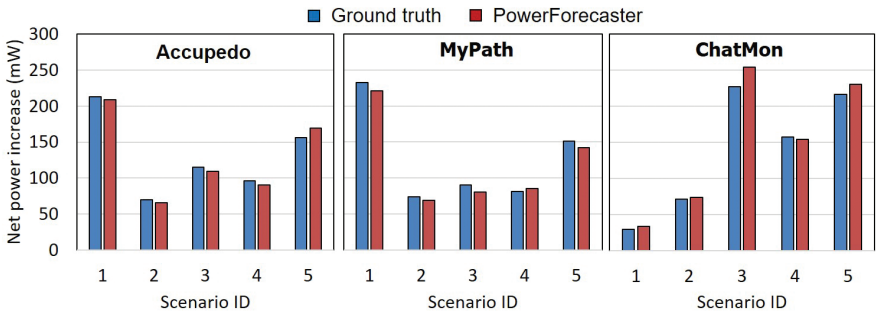


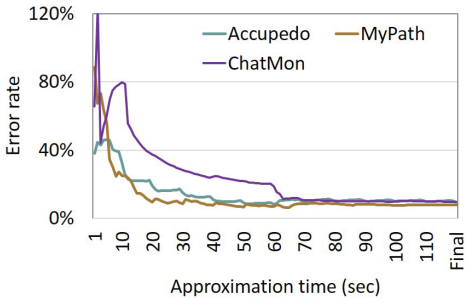
FIGURE 2. PowerForecaster architecture.



(a)



(b)



(c)

FIGURE 3. PowerForecaster evaluation: (a) setup (left), (b) estimation accuracy (upper right), and (c) latency (lower right).

well as the target application's executable. It then initiates multiple power emulator instances in parallel for speedy estimation. Each instance installs the executable and receives a part of traces from the emulator manager. If necessary, it replays pre-generated user interaction, such as clicking a start button to bootstrap the application.

Power-emulation: After the preparation, the emulator instance executes the application. While the application is running, the *sensor emulator* mimics the sensor operations to fulfill the application's sensing requests. It reads sensor traces and feeds sensor data at right timing and rate as the application demands. It also emulates the hardware states of the corresponding sensor devices and account for power estimation afterwards. To consider the sharing effect, the *device usage replayer* reproduces the existing applications' device use simultaneously based on the device usage trace. The *hardware usage monitor* tracks system calls made to use hardware components such as sensors and CPU. Finally, it generates hardware usage statistics.

Post-emulation: With the hardware usage statistics, the p*ower impact estimator* computes the increase in power consumed by the target application. For power estimation, we adopt a system call-based method using power profiles obtained by offline profiling. Currently, PowerForecaster considers the following hardware components: *CPU, GPS, inertial sensors, microphone, Bluetooth/Wi-Fi scans.*

EVALUATION

We present the key results to show the power estimation accuracy and latency of PowerForecaster. More details can be found in our original paper [6].

Setup: The current implementation supports Nexus 5 (Android 4.4.4) and Nexus S (Android 4.1.2). PowerForecaster can easily incorporate other devices as we modularize it to include new power models. For evaluation, we mainly used three sensing applications including a commercial pedometer application, *Accupedo*[1], and the two research applications we developed, *MyPath*, a location tracker and *ChatMon*, a conversation monitor, inspired by prior works [5][9]. These applications involve diverse hardware usage; Accupedo uses accelerometers and CPU. MyPath uses accelerometers, GPS, and CPU while ChatMon uses Bluetooth, a microphone, and CPU.

We performed scenario-based experiments. We crafted five one-hour scenarios with four parameters: mobility, encounter, indoor/outdoor status, and phone usage. The scenarios have different combinations of parameter values and their time durations, e.g., 10 minutes of walking alone outdoor while using a map application, and then 20 minutes of staying indoor with friends. For each scenario, we measured the accuracy by using four phones (P_A, P_B, P_C, and P_D) and three Monsoon power monitors as in Figure 3(a). P_A and P_B are used to collect ground truth; P_A runs a target sensing application with the same existing foreground applications while P_B runs the existing applications only. The difference in power consumption between the two phones is considered as the ground truth. P_C is used to collect sensor and device usage traces upon which PowerForecaster estimates the power impact of a target application. For comparison, P_D measures power while running the target application only; we omit the comparative results. It is interesting to note that Figure 3(a) clearly depicts the challenges in measuring power consumption of sensing applications in the real world.

Accuracy: Figure 3(b) shows the power measurement with three sensing applications for the five scenarios. PowerForecaster accurately estimates net power increases for the applications and scenarios. The error rates are 5.3 to 7.7%, indicating that PowerForecaster closely traces the power use of the target sensing applications.

We analyze the characteristics for each scenario and application. For Accupedo, Scenario-1 shows the largest net power increase. The scenario involves the longest movement (50 minutes), causing higher power use to process accelerometer data.

[1] https://play.google.com/store/apps/details?id=com.corusen.accupedo.te

Also, short use of other applications (5 minutes) makes the net power increase high due to little sharing effect. Scenario-2 is the opposite. Scenario-4 and Scenario-5 reveal the resource-sharing effect. They have the same movement duration but different usage time of other applications, 30 minutes and 1 minute, respectively. Thus, Scenario-5 shows twice net power increase. The accuracy results prove the PowerForecaster's capability of handling various physical and phone usage behavior of users. MyPath shows a similar trend since it has accelerometer-based triggering, as Accupedo does. ChatMon exhibits different trends due to different sensing logic and related user behavior. In Scenario-1, there is no Bluetooth encounter occurred, and thus, audio processing is not triggered, resulting in the smallest net power increase. Scenario-3 has the longest encounter time, 60 minutes, thereby performing sound processing for the entire duration. Thus, it shows the largest net power increase. Although it has greater encounter time of 20 minutes compared to Scenario-5, it shows similar net increase due to resource sharing.

Latency: Figure 3(c) shows the average errors versus emulation time. The error quickly decreases within 10 seconds for MyPath and Accupedo. Within 30 seconds, the error almost saturates. ChatMon takes about 60 seconds until saturation. ChatMon's longer saturation is attributed to the small absolute power consumption of Scenario-1, 29mW, causing high relative error; Scenario-3 and Scenario-4 of ChatMon reach 5% error rate in 12 seconds. The results show that PowerForecaster achieves the latency of around 20 seconds by adopting combination of acceleration techniques. We acknowledge that the latency needs to be further improved for commercial adoption of PowerForecaster.

CONCLUSION

We presented PowerForecaster, which provides instant, personalized power impact of mobile sensing applications at pre-installation time. We first uncovered that individual user behavior is a key to estimate power impact of mobile sensing applications. Then, we developed a novel power emulator using a user's behavioral traces.

While we demonstrated a proof-of-concept prototype and its feasibility, several important issues need to be further addressed to achieve the ultimate vision of PowerForecaster. First, we focused on providing an accurate power estimate of sensing applications given users' behavioral traces of a specific day. However, there could be daily variations and patterns of user behaviors. An important future work will be to model the routines and patterns of user behaviors that can affect the power estimation and to provide comprehensive power impact estimates adjusted for the different patterns. Second, we need to address a server-side scalability issue to integrate and deploy the system in real application markets. According to our estimation [6], several thousands of physical servers might be required to handle the worldwide workload. This would be reasonably practical considering today's commercial cloud services, but the ever-growing popularity of sensing applications will also aggravate the scalability issue. To resolve the problem, we are working on a novel approach to reducing emulation workloads. Third, PowerForecaster needs to incorporate more complex structure of mobile sensing applications and their diverse hardware usage patterns. For example, mobile sensing applications increasingly employ wearable devices beyond smartphones while the applications' operation might require cloud-side computation. Also, their sensing logic becomes sophisticated for accurate and power-efficient user behavior inference. To this end, we are extending the current power emulator to handle such an advanced application architecture. ■

Chulhong Min received a PhD in computer science from KAIST, now working as a postdoctoral researcher at KAIST. His research interests include mobile and pervasive computing systems, mobile sensing and context monitoring, human computer interaction, and IoT.

Chungkuk Yoo is pursuing a PhD at KAIST. Within the broad spectrum of mobile computing, his research interests lie in mobile systems and applications for in-situ social interaction in real world.

Junehwa Song received a PhD in computer science from the University of Maryland at College Park. He is a professor in the School of Computing at KAIST. His research interests include mobile and ubiquitous systems, ubiquitous services, human computer interaction, and social and culture computing.

Youngki Lee received a PhD in computer science from KAIST. He is an assistant professor at the School of Information Systems of Singapore Management University. His focus is on building experimental and creative software systems; his interest lies in mobile and social sensing, human behaviour analytics, and IoT systems.

Seungwoo Kang received a PhD in computer science from KAIST, and is now an assistant professor in the School of Computer Science and Engineering, KOREATECH. His research interests include mobile and ubiquitous computing, mobile sensing systems, and IoT.

Inseok Hwang received a PhD in computer science from KAIST. He is a Research Staff Member at IBM Research, Austin. His focus is sensory applications delivering cognitive computing user experiences in everyday life, and more broadly, his interests lie in the intersection of mobile computing and human computer interaction.

REFERENCES

[1] Dong, M., Lan, T., and Zhong, L. Rethink energy accounting with cooperative game theory. In Proc. ACM MobiCom, 2014.

[2] Jain, S., Borgiattino, C., Ren, Y., Gruteser, M., Chen, Y., Chiasserini, C. F. LookUp: enabling pedestrian safety services via shoe sensing. In Proc. MobiSys, 2015.

[3] Lane, N. D., Miluzzo, E., Lu, H., Peebles, D., Choudhury, T., and Campbell, A. T. A survey of mobile phone sensing. Communications Magazine, IEEE, 48(9), 140-150.

[4] Lee, Y., Iyengar, S. S., Min, C., Ju, Y., Kang, S., Park, T., Lee, J., Rhee, Y., and Song, J. (2012). Mobicon: a mobile context-monitoring platform. Communications of the ACM, 55(3), 54-65.

[5] Lee, Y., Min, C., Hwang, C., Lee,J., Hwang, I., Ju, Y., Yoo C., Moon, M., Lee, U., and Song, J. SocioPhone: everyday face-to-face interaction monitoring platform using multi-phone sensor fusion. In Proc. ACM MobiSys, 2013.

[6] Min, C., Lee, Y., Yoo, C., Kang, S., Choi, S., Park, P., Hwang, I., Ju, Y., Choi, S., and Song, J. PowerForecsater: Predicting Smartphone Power Impact of Continuous Sensing Applications at Pre-installation Time. In Proc. ACM SenSys, 2015.

[7] Min, C., Yoo, C., Hwang, I., Kang, S., Lee, Y., Lee, S., Park, P., Lee, C., Choi, S., and Song, J. Sandra helps you learn: the more you walk, the more battery your phone drains. In Proc. ACM UbiComp, 2015.

[8] Oliner, A. J., Iyer, A. P., Stoica, I., Lagerspetz, E., and Tarkoma, S. Carat: Collaborative energy diagnosis for mobile devices. In Proc. ACM SenSys, 2013.

[9] Paek, J., Kim, J., and Govindan, R. Energy-efficient rate-adaptive GPS-based positioning for smartphones. In Proc. ACM MobiSys, 2010.

[10] Pathak, A., Hu, Y. C., and Zhang, M. Where is the energy spent inside my app?: fine-grained energy accounting on smartphones with eprof. In Proc. ACM EuroSys, 2012.