A System-call based Model of Software Energy Consumption without Hardware Instrumentation

Shaiful Alam Chowdhury, Luke N Kumar, Md. Toukir Imam
Mohomed Shazan Mohomed Jabbar, Varun Sapra, Karan Aggarwal, Abram Hindle, Russell Greiner
Department of Computing Science
University of Alberta, Edmonton, Canada
Email: {shaiful, lkumar, mdtoukir, mohomedj, vsapra, kaggarwa, abram.hindle, rgreiner}@ualberta.ca

***Abstract*—The first challenge to develop an energy efficient application is to measure the application's energy consumption, which requires sophisticated hardware infrastructure and significant amounts of developers' time. Models and tools that estimate software energy consumption can save developers time, as application profiling is much easier and more widely available than hardware instrumentation for measuring software energy consumption. Our work focuses on modelling software energy consumption by using system calls and machine learning techniques. This system call based model is validated against actual energy measurements from five different Android applications. These results demonstrate that system call counts can successfully model software energy consumption if the idle energy consumption of an application is estimated or known. In the absence of any knowledge of an application's idle energy consumption, our system call based approach is still useful to compare the energy consumption among different versions of the same application.**

I. Introduction

Battery-driven mobile devices such as smartphones and tablets have become an indispensable part of our modern lives. As of now, two-thirds of the Americans are reported to carry a smartphone, and a significant portion of these users are dependent on these devices to access the Internet[1]. Globally, at least 1.4 billion smartphones were in use by 2014 [2], coinciding with a 70% increase in mobile data traffic observed within the same year [4]. With recent technological advancements, these mobile devices are equipped with different sensors and peripherals that inspire the development of more sophisticated mobile applications. Software developers continuously update, maintain, and improve upon their existing applications to satiate the user requests. Such updates, however, may lead to increased software energy consumption that reduces the availability of these battery constrained mobile devices, thus harming the overall user experience.

In a recent survey, longer battery life has been reported as the most desired smartphone feature [11]. Unfortunately, the improvement in battery technology is insignificant compared to the advancement in computing capability, implicitly advocating the necessity of energy efficient software development. Pinto *et al.* [14] observed that energy measurement related questions are the most perplexing; these questions had fewer acceptable answers on StackOverflow[2].

The fundamental aim of our research is to save developers from the complication of measuring actual energy consumption. In this paper, we propose multiple system call based energy models for Android mobile devices with very acceptable accuracy. Capturing system call traces of an application is much easier than profiling energy consumption directly; profiling a run of an application with *strace* provides the developer with a count of system calls invocations. Developers can directly use these system call counts with our model to estimate energy consumption for their applications.

Our contributions include: a proposal and validation of an accurate model that can estimate energy consumption of unseen applications based on their system call counts; and a demonstration that these models perform well at predicting relative change in energy consumption performance between multiple versions of the same application. Predicting energy consumption of a completely new application is very challenging for two reasons: 1) System calls can not capture the idle or mostly CPU bound energy consumption of an application and; 2) The set of system calls between two different applications can be significantly different. In order to deal with the first issue, our proposed models are evaluated for estimated idle energy. For the second issue, we demonstrate a system call grouping approach to generalize existing measurements. Our system call grouping approach is accurate at modelling energy consumption.

II. Background

In this section, we define some of the common terms that are frequently used in this paper.

a) Power and Energy: Power is the rate of work or the transmission rate of energy. Energy is defined as the capacity of doing work or the total power integrated over time [1]. The unit of power measurement is the *watt* (W) whereas the unit of energy measurement is *joule* (J) ($1J = 1W \cdot 1s$).

b) System Calls: A system call is a fundamental interface between an application and the operating system's kernel [1], [13], that provides the application with access to hardware and process related services. Some of the common operations accessed via system calls are *process control*—create and terminate a process; *file management*—create and read a file; and *device management*—request and access a device[3]. Consequently, the number of different invoked system calls can be used to estimate resource utilization of an application.

c) Tail Energy and Application's Idle Energy: Tail energy leak is the phenomenon when a device component

[1]http://www.pewinternet.org/2015/04/01/us-smartphone-use-in-2015/ last accessed: 22-Jun-2015

[2]http://stackoverflow.com/ last accessed: 22-Jun-2015

[3]http://man7.org/linux/man-pages/man2/syscalls.2.html last accessed: 05-May-2015

978-1-5090-0172-9/15/$31.00

stays in high power usage state for a period of time even after finishing its operation [1], [12], [13]. Such an energy leak is one of the most undesirable features of mobile applications to the energy-aware application developers. In contrast, we define an application's idle energy as the energy consumed by that application when the application is running without any user interaction. System calls typically do not capture such idle energy profiles, which led us to design our generic energy models in a way that works without application's idle energy (as described in Section V-C1). For simplicity, our Android device's idle energy consumption, approximately 0.7 joules consumed per second, is included within each application's idle energy consumption estimation.

III. Related Work

Prior studies related to modelling energy consumption can be divided into three broad categories: utilization based modelling, instruction based modelling, and modelling by using system call traces.

Utilization Based Modelling: Utilization based energy models [3], [19], [5] take into consideration the utilization statistics of individual components and settings of a system such as CPU, screen brightness, and Wi-Fi. Regression analysis is conducted to model an application's energy consumption by using the utilization statistics and the corresponding energy consumption. Utilization based models, however, suffer from several issues [13]. For example, tail energy can not be modelled by this approach of energy modelling.

Instruction Based Modelling: Seo et al. [16] estimated energy consumption of distributed software systems implemented in Java. Being a distributed system, their proposed framework integrates component level energy estimation with component's communication cost as well. In order to estimate software energy consumption using the cost of program instructions, Shuai *et al.* proposed *eLens* [6]—a tool that can estimate energy profiles at instruction level, method level, and thus can estimate the energy consumption of the whole software system. Instruction based energy models, however, are platform dependent and complex to develop.

System Call Based Modelling: The drawbacks of the widely explored utilization and instruction based approaches motivated new energy modelling approaches, such as energy model based on system call invocations. A system call based approach is able to overcome the shortcomings of the utilization based approaches for several reasons [13]. System calls are only gateways that provide access to different I/O components—thus capturing all the system calls invoked by an application will indicate the majority of I/O components accessed by an application. Moreover, system call based approaches do not suffer from tail energy phenomenon as such approaches do not use resource utilization statistics directly.

Pathak *et al.* [13] modeled energy consumption of different Android and Windows mobile phones using system call traces, and the logs of context switch events were captured to estimate CPU power consumption. Their proposed model is based on finite state machines (FSM), where each state in FSM represents a power state of a specific component or a set of components. Although such a model offers high accuracy, producing a FSM for each hardware component requires expertise, time, and measurement equipment.

Aggarwal *et al.* [1] followed a much simpler approach to understand energy consumption of mobile applications. Instead of using complicated FSMs, the authors used system call counts as the model input. The proposed model can compare two versions of an application to suggest if the difference in energy consumption is statistically significant. Their model, however, can not deal with unseen system calls and might fail for new applications. Moreover, the actual amount of application energy consumption can not be reported.

In our work, we are more interested to exploit different machine learning approaches towards predicting an application's actual energy consumption so that a developer does not need to worry about developing and maintaining complicated hardware based measurement systems like the GreenMiner [9].

IV. Methodology

This section describes our hardware based energy profiler (GreenMiner [9]), the process of collecting energy profiles of different applications, the algorithms used for modelling energy consumption, and the method of evaluating the performance of our energy consumption models.

A. GreenMiner

In order to capture the energy profiles of the applications, we used GreenMiner [9] as the test bed. GreenMiner consists of five basic components: a power supply for the phones (YiHua YH-305D), 4 Raspberry Pi model B computers for test monitoring, 4 Arduino Unos and 4 Adafruit INA219 breakout boards for capturing energy consumption, and 4 Galaxy Nexus phones as the systems under test. The GreenMiner is fully described in the prior literature [9], [15].

B. Data Collection

Data was collected using four sequential steps.

Step 1: We chose five open source applications (Firefox, Calculator, Bomber, Blockinger and FeedEx) for the Android platform to conduct our experiments; we selected these five applications as they are from different domains and their version control repositories are readily available. Multiple versions of these applications were built from their version control repositories. Table I describes each of the applications.

Step 2: A separate test case for each application was designed to emulate the realistic usage of the application by an average user. For example, to simulate an average user scenario of reading a web page in the Firefox web browser, the test opened and slowly scrolled through the Wikipedia web page on American Idol for 4 minutes. For Calculator, another test performed several simple calculations such as metric conversions, and solves a quadratic formula. For games, such as Bomber and Blockinger, simple game playing scenarios were designed. Likewise, FeedEx's test case was to emulate reading RSS feeds.

Step 3: GreenMiner was used to run the applications and test scripts to collect the energy consumption. In order to collect system call profiles, the *strace -c* option was used. The *strace* profiler was run separately from the actual energy measurement test in order to avoid the overhead that might contaminate the actual measurement.

Step 4: As variation was observed in both energy consumption and system calls over different runs, multiple runs (10) were measured to calculate averages for both system call counts and energy consumption measurements [1], [8].

Applications	Type	No. of versions	No. of system calls	Time period of commits of versions
Firefox	Browser	156	52	Jul, 2011 - Nov, 2011
Calculator	Android Calculator	101	25	Jan, 2013 - Feb, 2013
Bomber	Android game	79	47	May, 2012 - Nov, 2012
Blockinger	Android game	74	56	Mar, 2013 - Aug, 2013
FeedEx	Android news reader	21	70	May, 2013 - Apr, 2014

TABLE I. DESCRIPTION OF THE APPLICATIONS

C. Algorithms and Implementations

In order to model applications' energy consumption, we employed two of the most commonly used machine learning approaches: linear regression and support vector regression (SV). Linear regression has a tendency to overfit training data, which we addressed by using regularization with 10-fold cross validation. We used the linear regression algorithm implemented within MATLAB, considering system calls as our feature vector and energy consumption as the observed values.

In ϵ-SV regression [18], the main goal is to find a function $f(x)$ that does not deviate more than ϵ from the true values. In this work, a kernelized SV regression model [17] is used using the SVM^{light} [10] implementation with a linear kernel. Success using linear kernel—instead of more complicated RBF, polynomial, and sigmoid kernels—is beneficial, as linear features tend to be more interpretable [7].

D. Evaluation

In order to show the accuracy of our proposed models, we compare the performance of our models, in terms of mean squared error (MSE), with a theoretically controlled baseline model. Predictions in the baseline model are the actual energy measurements summed with normally distributed errors. The mean of the distribution is 0, and the standard deviation is 10% of the average of the actual measurements. Taking average for the standard deviation is valid for all of our applications (except FeedEx), as the variation in energy consumption among different versions is very low; for FeedEx, we used 10 joules as the standard deviation. For the baseline model, the MSE of the actual values and the predicted values are taken for 1000 different runs[4], and then the average MSE is reported. This simple way of evaluating our models' accuracy alleviates the difficulty of comparing with other earlier methodologies described in Section III. The fixed 10% error for standard deviation in our baseline model is inspired by the performance of the previous models. For example, the error of *eLens* in predicting software energy consumption was with in 10% of the actual measurements [6]. This gives us implicit knowledge on the performance of our proposed models compared to the earlier models.

V. EXPERIMENTS

We have three different experiments with three different objectives, each intended to address a different prediction problem relevant to software developers. These three experiments include: 1) train on a single application and test on the same application; 2) train on a set of applications and test on the same set of applications; and 3) train on multiple applications and predict on an unseen application.

A. Train on a Single Application and Test on the Same Application

In this experiment, the software energy consumption prediction model is trained on a single application (80% training data) and tested on the same application (20% test data). This type of model is proposed for software developers who are working with a single application, but continuously developing new versions. For example, Firefox is not immediately deployed to the end-users. The developers build nightly versions each day known as Mozilla nightly. A more stable nightly version is known as Aurora. The Beta version comes after successful testing of Aurora. And a completely stable Beta is released as Firefox[5]. If the energy consumption profiles can be captured for the existing versions of Firefox using test bed like GreenMiner, then an energy consumption prediction model can be developed. As a result, new version's energy profile may be estimated rather than expensively measured.

Table II shows the performance comparison between system call based models, linear and SV regression, and the baseline model. Both of our models outperform the baseline with a clear margin in all of the cases. Not surprisingly, the more robust SV regression outperforms the simple linear regression model; FeedEx has a very low number of examples (only 6 versions for testing and 15 for training), which could be one of the reasons for its worse performance of SV regression compared to the linear regression. For Firefox, although the MSE for linear regression is much better than the baseline, it is still much higher compared to the MSE for other applications. Figure 1(a) depicts the prediction accuracy of SV regression for Firefox. This shows the accuracy of our proposed models; both the models were capable of capturing the fluctuations between versions. For Firefox, however, there were three versions in the test data that have different energy profiles than other versions. Although our linear regression model could capture the shape for those versions, the scale was not very accurate[6], thus contributing to an MSE that is larger than for other applications. SV regression's performance shows that it handled such variation better.

B. Train on a Set of Applications and Test on the Same Set of Applications

In this experiment, we address organizations and software developers that develop multiple applications. Instead of modelling each application's energy profile separately, such developers seek a generalized energy consumption model for all software applications. In order to build such a model, we used both the linear and SV regression with all the 5 applications in training (using combined 80% random measurements from

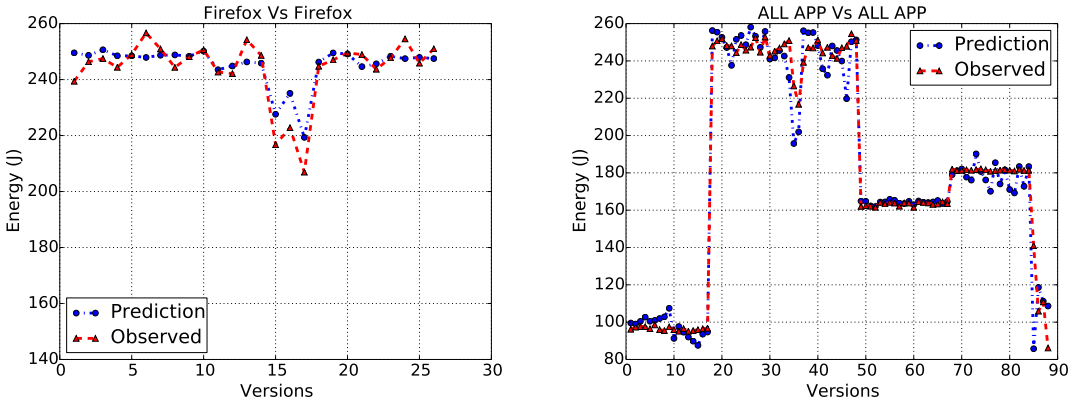
[4]different errors are produced in different runs from a normally distributed error function, so multiple runs are required to have a stable average MSE

[5]http://mozilla.github.io/process-releases/draft/development_overview/ last accessed: 20-May-2015

[6]Figure is not included because of the lack of space

Applications	Linear regression (MSE)	SV regression (MSE)	Baseline (MSE)
Firefox	234.15	**32.17**	884.02
Calculator	0.62	**0.47**	80.39
Bomber	0.98	**0.73**	255.68
Blockinger	0.22	**0.10**	190.07
FeedEx	**1.77**	31.39	33.91

TABLE II. PERFORMANCE COMPARISON (MSE) FOR TRAIN ON A SINGLE APPLICATION AND TEST ON THE SAME APPLICATION



(a) Firefox with SV regression

(b) Multiple apps with Linear regression

Fig. 1. Energy consumption estimates based on project history (80/20): actual versus predicted (for both single application and multiple applications)

each application), and then evaluate the prediction on the unused measurements (using rest of the 20% measurements from each application). Each application has its own system call profile. Some applications do call system calls unique to their domain, while some system calls are called by all applications. One method to address the variation in system call profiles between applications is to either consider the union or the intersection of the system calls invoked between applications. For our mixed model (training on Firefox, Calculator, Bomber, Blockinger, and FeedEx), we took the union of all system calls from all the five applications. After conducting 10-fold cross validation, we tested our models for the test data.

As in this experiment, the training data consists of all the five applications, taking the average of the training values as the standard deviation for the baseline model is not recommended as different applications have different scales of energy profiles. For simplicity, we chose standard deviation as 10 (joules) and thus get MSE of 485.72 for the baseline model. For our models, the MSE for the linear regression and SV regression models are 96.50 and 61.16 respectively, showing the much superior performance of system call count based models. Figure 1(b) shows the effectiveness of our proposed models (using linear regression as the example) more clearly by depicting the actual and predicted data for the mixed model.

When compared against the application specific models of Section V-A, the MSEs of these multiple application models are higher but still lower than the baseline of each set of models (except FeedEx): average single application linear regression MSE was 47.54 versus 96.50 for multiple applications; average single application SV regression MSE was 12.97 versus 61.16 for multiple applications. Regardless, the multiple application models all had MSE lower than the average baseline model in almost all cases, and Figure 1(b) demonstrates that the model performance is visually very acceptable.

C. Training on Multiple Applications and Predicting on an Unseen Application

In this experiment, the energy consumption prediction models are trained on multiple applications and tested on an unseen application. Realistically, a model capable of predicting unseen application's energy consumption would be very useful, as any organization can use this model for any of their applications. Predicting energy consumption for an unseen application, however, is very challenging, as the new applications can have a set of system calls that were never encountered in the training phase. Furthermore, system calls cannot capture the idle energy consumption of an application, making it even more difficult to predict a new application's energy consumption profile. We describe addressing these issues below:

1) Addressing Application Idle Energy: System calls do not capture idle or CPU bound energy consumption of an application. This led us to control and deduct applications' idle energy from our dataset. Using the GreenMiner, we observed the power use time series for some of the versions of the five applications. In the time series, we observed flat periods at regular intervals for all the applications; this is the time when the user (i.e., our automated test script) was not interacting with the system—thus, these intervals represented the applications' idle behaviour.

Although with models like this, the idle energy of an application must be estimated in order to predict its total

Groups	System calls	Semantics
Memory_mapping	mremap, munmap, brk, madvise, mprotect, mmap2	System calls that are related to address mapping or similar functions.
Stat	lstat64, fstat64, stat64, statfs64, access	System calls that return file information.
Socket	setsockopt, socket, bind, connect,getsockname,socketpair etc.	System calls related to socket operations.

TABLE III. EXAMPLES OF GROUPING SYSTEM CALLS

Applications	Linear Regression (MSE)					SV Regression (MSE)					Baseline (MSE)
	0.7	0.8	0.9	1	X	0.7	0.8	0.9	1	X	
Firefox	985.85	109.36	253.45	1418.09	253.45	1478.39	272.84	**87.85**	923.42	**87.85**	628.43
Calculator	**5.92**	79.56	373.10	886.54	**5.92**	6.65	85.62	384.49	903.27	6.65	89.57
Bomber	4313.15	2968.06	1873.81	1030.41	5.26	4480.84	3107.35	1984.71	1113.05	**2.69**	255.53
Blockinger	541.73	127.17	47.00	301.20	47.00	627.61	166.56	**39.89**	247.59	**39.89**	324.87
FeedEx	1563.18	1184.63	869.26	617.09	662.47	1084.21	791.98	562.94	397.10	425.21	**98.97**

TABLE IV. PERFORMANCE COMPARISON (MSE) FOR UNSEEN APPLICATIONS

energy consumption, these types of models are still very useful to compare the energy consumption between two versions of an application (regardless of the actual idle energy consumption).

The idle power usage deducted from our dataset are 0.7, 0.9, 1.30, 0.9 and 0.98 *watt* for Calculator, Firefox, Bomber, Blockinger and FeedEx respectively. These power usages were multiplied by the duration of the experiments to get the total idle energy for each of the applications. After deducting the idle energy consumption, the average energy consumption of Firefox, Calculator, Bomber, Blockinger, and FeedEx were 44.79, 22.65, 18.00, 64.94 and 43.17 joules respectively. This shows that our selected applications are very different in their energy consumption patterns. The similar energy profiles for Firefox and FeedEx is expected because of the similar nature of the applications—they are both HTTP clients.

2) Addressing Unseen System Calls: In our dataset, Firefox has five system calls (*brk*, *getpeername*, *sigaction*, *utimes* and *sendmsg*) that are not present in any other application. Similarly, three system calls of FeedEx (*bind*, *socketpair*, and *ftruncate*) are also absent in the combined set of the four other applications. This is problematic for our energy prediction model as these system calls can not be modeled during the training phase. The two simple approaches for dealing with unseen system calls, *i.e.*, intersection and union of all the system calls, thus fail to offer an accurate energy consumption model. This led us to design a new feature encoding technique—grouping similar system calls together so that the group names can be used as the features instead of using the actual system calls.

Table III shows a snapshot of our grouping mechanism from a total of 28 groups. We collected the function/behaviour of a system call using the *man* command in Linux. Using the semantics, we grouped similar system calls together. For example, if an application uses *getuid* whereas another uses *geteuid*, we counted the occurrences of those system calls under the same group name—GetUid. This is because of our hypothesis that these types of system calls should contribute to the same amount of energy consumption. This proposed grouping approach can be used to model energy consumption of other platforms besides Android.

3) Modification of the Algorithms: The offset parameter for both Linear and SV regression—although is valid when a fraction of data of the test application is used with the training dataset—is problematic in case of predicting energy consumption for unseen application; this parameter adds an extra offset value in our model without having any knowledge of the unseen application. In the case of developing a generalized model that can be used for any new application, our model must rely only on the weights of the system calls. In order to resolve this issue, we set the intercept (which is not system call) of all the examples as 0 (forcing offset to be 0) instead of the offset found by the linear regression model. Similarly, when using SV regression we set the parameter b as zero while running *svm_learn* in SVM^{light}[7].

We also validated our trained models in a different way than the previous experiments. Instead of applying 10-fold cross validation over the 80% training data, we select one of the applications for cross validation, and three other applications for training. This way we have better understanding on how our model is going to work for a completely unknown application. When both the training and cross validation accuracies are very good and close to each other, we test our model on our test application—the application that was never used for training or cross validation.

Table IV shows the performance of both the algorithms with the baseline. We present the performance of our models with different estimations of idle power usage (0.7, 0.8, 0.9, and 1.0 watt) for all the applications in order to show how inaccurate the results would be if the developers estimate their application's idle energy different than the actual idle energy (X in the table). This statistics makes more sense when the MSE of our models are very low (e.g., Calculator and Bomber). For example, if the developers estimate Calculator's idle power usage as 0.8 instead of 0.7 *watt*, the MSE becomes 79.56 for linear regression, which is still lower than the baseline model's MSE.

In general, the performance of our models show that in many cases a 0.1 idle *watt* error in estimation still results in a model that performs better than the baseline model. When the correct idle energy was added—i.e., the X columns—both the models outperform the baseline model for all the applications except FeedEx. The superior performance of SV regression over Linear regression could be its resilience to not overfitting the training data. The poor performance with FeedEx measurements for both of our models is not surprising as the variation in energy consumption among different versions of FeedEX is the highest of all the applications, thus the idle energy consumption we estimated for FeedEx could be

[7]http://svmlight.joachims.org/ last accessed: 01-May-2015

inaccurate. We observe, however, very accurate prediction for Calculator and Bomber, and to a lesser extent for Blockinger. This is because of the fact that these applications do not have any system calls that are not present in the training data comprised of other applications, which is not true for Firefox and FeedEx. This implies that collecting more applications' measurements containing more possible system calls will help us to develop a more robust generalized model.

VI. Discussion

The accuracy of both of our models with linear features is encouraging because it means we can interpret what the models have learned. Furthermore, it suggests that counts of system calls do model energy consumption behavior in an application. The theory behind the success of the linear models is that they simplify the states of the FSMs in *eprof* by Pathak *et al.* [12]. A specific system call represents the usage pattern of an specific resource, e.g., *read* for reading a file. Thus, energy profiles of all the resources that an application uses can be captured by profiling the different system calls counts invoked by that application. Although system calls typically do not directly capture CPU usage, they are often correlated with memory usage. Thus sometimes memory related system calls can be expected to capture some CPU usage.

Our final generic system call based model is useful to any application developer as one can estimate the energy consumption of an application without any hardware instrumentation; rather an estimation of application's idle energy and a measurement of system calls are sufficient. Even if the idle energy estimation is incorrect (more than 0.1 *watt* off) the model works relatively, as the error from a constant offset—application idle parameter—will disappear when 2 versions of the application are compared. Only the change in CPU usage between the 2 versions will have an unseen affect.

We recommend that developers follow a rule of thumb for estimating the idle energy of their application: if their application only operates on user input and has no background processes—audio, animation, networking threads, etc.—then set application idle to the idle of the device (for our smartphones it was near 0.7 *watt*); if the application has background processes, especially networking, then idle usage should be higher (0.8 *watt* or 0.9 *watt*); where as if the application is a game with a 30 fps or higher event loop and utilizes graphics then the idle should be even higher (1.0+ *watt*).

A. Threats to Validity

The idle energy that we estimated for our five applications were from simple time-series observations. It would be more accurate to develop separate test cases with no user interaction for each of the applications so that actual idle energy can be obtained. The test cases we designed might not have covered all the significant features of our applications. External validity is hampered by using only one brand and make of smartphone.

VII. Conclusion and Future Work

In this paper, we have presented multiple system call based energy models using multiple versions of five Android applications: Firefox, Calculator, Bomber, Blockinger, and FeedEx. Our experiments provide support that system calls can be used to predict the energy consumption of mobile applications. Furthermore, we show that the energy consumption of unseen applications can be estimated without hardware instrumentation using measurements from 3rd-party applications measured with hardware instrumentation—primarily by accounting for the idle energy use of a particular application. Using a dynamic trace of system calls and a estimate of idle application energy usage, a developer can accurately, within a certain error tolerance, estimate the energy use of their application without a hardware power meter. This context is useful for the average developer who does not have access to energy measurement infrastructure, but seeks to control for application energy consumption during software evolution.

Future work includes building a more general model trained on more systems with more features including CPU usage, and different code features, such as depth of inheritance. Our long term goal is to help software developers during development by notifying them of potential negative effects their code changes might have on an application's energy consumption.

References

[1] K. Aggarwal, C. Zhang, J. C. Campbell, A. Hindle, and E. Stroulia. The Power of System Call Traces: Predicting the Software Energy Consumption Impact of Changes. In *CASCON '14*, 2014.

[2] Banerjee, Abhijeet and Chong, Lee Kee and Chattopadhyay, Sudipta and Roychoudhury, Abhik. Detecting Energy Bugs and Hotspots in Mobile Apps. In *FSE 2014*, pages 588–598, Hong Kong, China, Novemeber 2014.

[3] A. Carroll and G. Heiser. An Analysis of Power Consumption in a Smartphone. In *Proceedings of the USENIXATC'10*, 2010.

[4] Cisco. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2014-2019. Technical report, Cisco, February 2015.

[5] M. Dong and L. Zhong. Self-constructive High-rate System Energy Modeling for Battery-powered Mobile Systems. In *Proceedings of the MobiSys '11*, pages 335–348, June 2011.

[6] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan. Estimating Mobile Application Energy Consumption Using Program Analysis. In *ICSE '13*, pages 92–101, 2013.

[7] T. Hastie, R. Tibshirani, and J. Friedman. Support vector machines and flexible discriminants. In *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics.

[8] A. Hindle. Green Mining: Investigating Power Consumption Across Versions. In *ICSE '12*, pages 1301–1304, June 2012.

[9] A. Hindle, A. Wilson, K. Rasmussen, E. J. Barlow, J. C. Campbell, and S. Romansky. GreenMiner: A Hardware Based Mining Software Repositories Software Energy Consumption Framework. In *MSR 2014*, pages 12–21, Hyderabad, India, May 2014.

[10] T. Joachims. Making large-scale support vector machine learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods*, pages 169–184. MIT Press, 1999.

[11] I. Moura, G. Pinto, F. Ebert, and F. Castor. Mining Energy-Aware Commits. In *MSR 2015*, Florence, Italy, May 2015.

[12] A. Pathak, Y. C. Hu, and M. Zhang. Where is the Energy Spent Inside My App?: Fine Grained Energy Accounting on Smartphones with Eprof. In *EuroSys '12*, pages 29–42, Bern, Switzerland, April 2012.

[13] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang. Fine-grained Power Modeling for Smartphones Using System Call Tracing. In *EuroSys '11*, pages 153–168, Salzburg, Austria, April 2011.

[14] G. Pinto, F. Castor, and Y. D. Liu. Mining Questions About Software Energy Consumption. In *MSR 2014*, pages 22–31, 2014.

[15] K. Rasmussen, A. Wilson, and A. Hindle. Green Mining: Energy Consumption of Advertisement Blocking Methods. In *GREENS 2014*, pages 38–45, Hyderabad, India, June 2014.

[16] C. Seo, S. Malek, and N. Medvidovic. Component-level energy consumption estimation for distributed java-based software systems. volume 5282 of *Lecture Notes in Computer Science*, pages 97–113. Springer Berlin Heidelberg, 2008.

[17] A. J. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 14(3):199–222, August 2004.

[18] V. Vapnik. *The nature of statistical learning theory*. Springer, 2000.

[19] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang. Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones. In *Proceedings of the 8th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, 2010.