Kraken.me Mobile: The Energy Footprint of Mobile Tracking

Immanuel Schweizer*, Roman Bärtl*, Benedikt Schmidt*, Fabian Kaup†, Max Mühlhäuser*
* Technische Universität Darmstadt, Telecooperation Lab
{schweizer,benedikt.schmidt,max}@tk.informatik.tu-darmstadt.de
† Technische Universität Darmstadt, Peer-to-peer Systems Engineering Lab
fkaup@ps.tu-darmstadt.de

***Abstract*—Power consumption can make or break the success of mobile applications. This is especially true for applications requiring constant access to sensor readings as sensors tend to consume considerable amounts of energy. A lot of attention has been focused on reducing power consumption for hardware sensors both from a hardware and software perspective. However, mobile phones enable applications to also gather software artifacts employing so called *soft sensors*, e.g., calendar, contacts, browsing history, etc.**

Soft sensors are especially important when considering personal assistant systems, life logs etc. They provide additional deep insight into human behavior patterns and user goals. Unfortunately, most tracking application do not consider these soft sensors and their power consumption is mostly unknown.

In this paper we introduce the Kraken.me mobile tracking application. It is part of the Kraken.me framework, tracking mobile, desktop, and social interactions. The application tracks both hard and soft sensors to enable the creation of rich user profiles. Our main contribution is a thorough evaluation of the power consumption of each individual sensor used and the combination in the Kraken.me application. Using a high-accuracy measurement setup, we provide an in-depth power analysis of both soft and hard sensors. We believe these insights can help researchers and developers of mobile tracking applications to design more power efficient and, thus, more successful tools.

***Keywords*—Power consumption, Mobile Tracking, Soft sensors**

I. Introduction

Mobile phones are battery-constrained and mobile sensing applications are among the most energy consuming. This creates an area of conflict. Mobile sensing, especially for real-time use cases, requires high sampling rates while applications with high power consumption see only limited success [1].

Recently, this trade-off was explored in great detail by Kansal et al. [2]. They argue that a programmer building mobile context sensing apps should be able to specify two dimensions: (i) the *latency* at which context change is detected and (ii) the *accuracy* of the inferred context. They propose the latency, accuracy, and battery (LAB) abstraction to specify these dimensions. Their Senergy API is then supposed to provide the most energy-efficient context sensing algorithm to fulfill the specified requirements. This is a powerful approach lending app developers a tool to improve both programmer productivity and energy efficiency.

However, this is not the reality when implementing mobile context sensing applications today. Developers and researchers build their own approaches relying on access to the raw sensor data provided by the API, they use the default context provided by the operating system, or, if applicable, alternatives provided by the operating system. Kansal et al. introduce these three possibilities as (i) *Raw*, (ii) *One Default*, or (iii) *Fixed Predefined Modes*. Using these the developer has only limited choices when it comes to accuracy or latency. Also, the availability of accurate energy measurements for the standard APIs is very limited.

This was a challenge for us, as we are currently developing the Kraken.me suite [3]. Kraken.me is a comprehensive suite of tracking applications designed as building blocks for context-aware application, e.g., personal assistance etc. Kraken.me mobile tracks the user's behavior using thirteen different hard and soft sensors currently. To provide this comprehensive data gathering, we – as many other developers of mobile context sensing applications – must rely on the standard APIs, if possible. To provide valuable insights for other researchers and developers, we wanted to understand the power consumption for each sensor individually and in combination. Especially, since we employ soft sensors, which have not really been examined in literature.

Hence, this paper contributes the following:

1) The Kraken.me mobile application as a showcase of mobile context sensing across hardware and software sensors
2) Accurate power consumption measurements of the standard Android API for most hardware sensors
3) Accurate power consumption measurements for three custom software sensors (calender, contacts, call logs)

The paper is , therefore, organized as follows. Section II will introduce the overall Kraken.me framework and most importantly the Kraken.me mobile application. Section III reports the evaluation results. Section IV summarizes the related work, before Section V concludes the paper.

II. Kraken.me

Kraken.me mobile is one building block in the overall Kraken.me framework [3][1]. While this paper focuses on Kraken.me mobile we will first introduce the overall system to give some context. Afterward the mobile application and especially the implementation of software (soft) sensors is given.

[1]https://kraken.me

MobiCASE 2014, November 06-07, Austin, United States
Copyright © 2014 ICST
DOI 10.4108/icst.mobicase.2014.257823

The goal of Kraken.me is to gather information about people's lives. Kraken.me, therefore, offers an unprecedented, vendor-agnostic tracking framework. Kraken.me has two properties that – together – distinguish it from other such frameworks. First, Kraken.me is a multi-device tracking framework. It offers tools to tracks social networks, desktop interaction, and mobile interaction. Second, Kraken.me tracks *hard* (e.g., activity, location, accelerometer, etc.) and *soft* sensors.

The overall architecture of Kraken.me is depicted in Figure 1.

All tools send their data to a central instance where the data is stored and later processed. The webpage provides a full list of all collected data points[2]. We do not expect people to voluntarily give their data for research, but want to offer comprehensive visualizations and use cases to grow our user base. For a more comprehensive overview of Kraken.me the reader is refereed to [3].

In the following section we concentrate on Kraken.me mobile for Android and, later, focus on the crucial question of energy consumption. Draining the battery is a big challenge for continous tracking and we will provide valuable insight into the energy consumption of different soft and hard sensors available.

A. Kraken.me Mobile

Kraken.me strives to collect a comprehensive picture about a person's interaction with both the environment and technology. Mobile devices are deeply entrenched into a person's life and equipped with an increasing number of sensors. A mobile application is, thus, an essential building block of Kraken.me.

A mobile device offers access to multiple soft and hard sensors. The idea was to collect data using a greedy approach: We want to collect as much data as is acceptable from (i) a *privacy perspective* and (ii) a *energy consumption perspective*.

Given the implication of continuous tracking, (i) is a discussion best reserved for another paper. To give a brief introduction into the Kraken.me sign-up process: We try to make sure – multiple times – that people understand the implications when signing up with Kraken.me (through any of the tools) (cf. Figure 2(a)). People can – after accepting these implications – sign up with their social media account. Inside the application we offer different tracking profiles (cf. Figure 2(b)), which can be changed at any time.

The collected data is stored on the device and is pushed to the Kraken.me server on a regular basis. Data are presented with one of two possible views. The first view gives you an idea about your app usage today (cf. Figure 2(c)), while the second compares your activity with your app usage. Based on this information, users get a better understanding of their smartphone usage behavior and see the benefit of the data they collect.

The challenge and implications of (ii) are at the core of this paper's contribution and will be discussed in the evaluation. Before being able to understand power consumption, however, we need to understand just how much data we can and do collect. For easier reference the hard sensors are given in Table I and the soft sensors in Table II.

Data	Description
Sound pressure	dBSPL value captured with the microphone
Ring mode	Is the phone on silent, vibration, etc.
Acceleration	Readings from the accelerometer
Activity	Activity Recognition provided by the Android API
Illuminance	Lux as provided by the Android API
Network Connectivity	The active and all available network connections

TABLE I: Data collected from/derived from physical sensors by Kraken.me mobile

Data	Description
Browsing History	All visited web sites excluding incognito mode
Contacts	All saved contacts
Calendar	All saved calendar events and reminders
Location	Location as provided by the Android API
Call Log	The call history
Foreground apps	Time span apps are in foreground
Keyboard	Statistics about key strokes

TABLE II: Data collected from software sensors by Kraken.me mobile

To offer such a comprehensive set of sensors, we decided to develop for Android first. Currently, iOS is too restricted for the same tracking, especially considering soft sensors.

All sensors in Kraken.me need to rely on the Android API in one way or the other. If applicable we tried to use the data provided by the API without any further processing, e.g., accelerometer, activity or location. Other sensors implement further processing on the raw data given by the API. This is especially true for the soft sensors. Here we rely on two general techniques – (i) content observer and (ii) accessibility – which are explained in the following section.

1) ContentObserver API: A small number of soft sensors can be implemented using a standard Android API, e.g., browser history. However, most soft sensors in Kraken.me are implemented using Android's concept of content observers. Here, a content observer can register on any URIs which points to a system database, e.g., calendar data. The observer receives a call back whenever changes to the underlying database occur.

Following such a call back we always retrieve the whole data due to two reasons: (i) Even though the documentation states that it should be possible to only retrieve updated data since Android 4.1, we were not able to consistently reproduce this with any of the devices we use for testing and (ii) even if it would work, we would miss any updates during times when Kraken.me is not running. Every callback will contain the complete database. Hence, we are required to keep the current state of the database stored in Kraken.me to calculate a diff whenever we receive a callback. This concept is currently used to implement three sensors: call logs, calendar, and contacts.

While call logs are only changed whenever a call occurs the other two pose one additional challenge. Whenever a user makes a change the Android operating system will synchronize this change with the Google backend (if activated). This

[2]https://kraken.me/#!/datenschutz

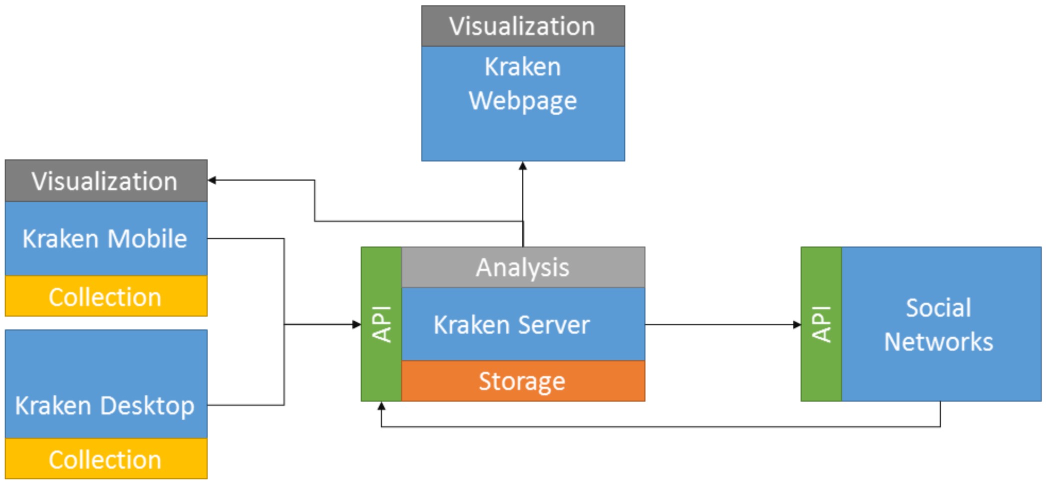
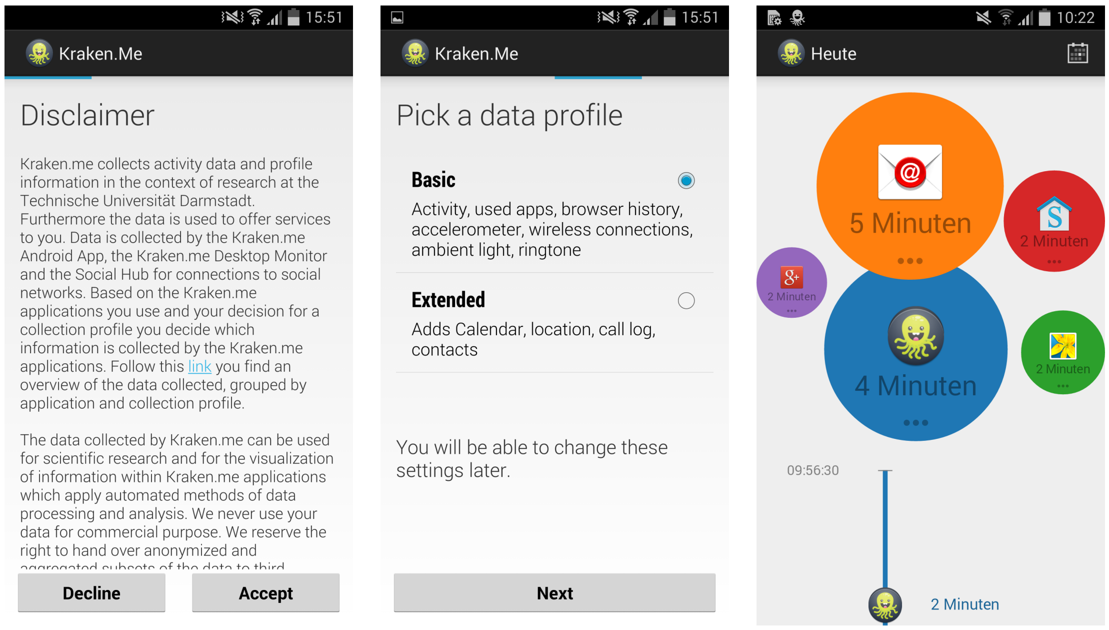


Fig. 1: Overview of the Kraken.me System Architecture



Disclaimer

Kraken.me collects activity data and profile information in the context of research at the Technische Universität Darmstadt. Furthermore the data is used to offer services to you. Data is collected by the Kraken.me Android App, the Kraken.me Desktop Monitor and the Social Hub for connections to social networks. Based on the Kraken.me applications you use and your decision for a collection profile you decide which information is collected by the Kraken.me applications. Follow this link you find an overview of the data collected, grouped by application and collection profile.

The data collected by Kraken.me can be used for scientific research and for the visualization of information within Kraken.me applications which apply automated methods of data processing and analysis. We never use your data for commercial purpose. We reserve the right to hand over anonymized and

Pick a data profile

Basic ●
Activity, used apps, browser history, accelerometer, wireless connections, ambient light, ringtone

Extended ○
Adds Calendar, location, call log, contacts

You will be able to change these settings later.

(a) Kraken.me disclaimer (b) Select collection profile (c) Apps Statistics

Fig. 2: Screenshots of Kraken.me Mobile

synchronization will set flags inside the database triggering yet another callback to the content observer. Additionally users tend to update information in close time proximity leading to large amounts of subsequent triggers. We implemented a simple timeout mechanism to filter these subsequent changes and synchronizations. If a new call back occurs during the timeout the timer is reset until it eventually triggers the diff calculation. The impact of this small change will be further discussed in the evaluation.

2) Accessibility API: Some information is not easily accessible given any of the official APIs. To record statistics on key presses and foreground apps, we had to rely on a backdoor build into the system. The accessibility API delivers information about the user interface, gestures, touch and the status of the phone to be used for the realization of barrier-free apps. Kraken.me, however, is registered as an accessibility app – which needs to be activated by the user – to implement even more sophisticated soft sensors. After registration, the application is allowed to subscribe to accessibility events. Based on the events delivered by the Accessibility API we can generate events we are interested in (e.g. change of foreground app, interaction with text fields, etc.).[3] Given the number of sensor included in the application, the inherent question remains: How can we realize continuous tracking with the

[3]These soft sensors have just been implemented and were not available for the power evaluation. We include them here to share the employed idea with other researchers.

highest possible granularity? In the next section we share our findings on power consumption and discuss implications.

III. Evaluation

Kraken.me is designed to provide real-time personal tracking data. The main objective is gathering this data at the maximal possible sampling rate. However, the application will gain no real-world usage, if battery life is severely affected. Thus, a balance between measurement accuracy and battery consumption must be found. This optimization problem requires profound knowledge about the power consumption of different sensors and their interaction.

Before we introduce the results, we will start off by explaining our measurement setup. Next, we will investigate the energy consumption of different sensors individually. Afterward we measure the energy consumption for two different sensor configurations of Kraken.me and end the section by providing some recommendations for sensing applications in general.

Before diving into the results, let us first introduce the measurement setup.

A. *Measurement Setup & Methodology*

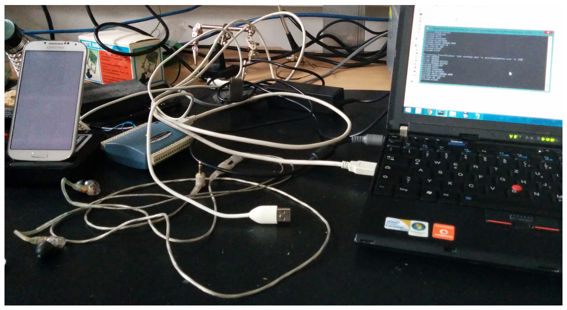
Measuring energy consumption on mobile phones is not a trivial task. Different background processes and inter-dependencies between applications make it more difficult to single out any given application. Also, the battery level is only a crude estimate of the power consumption.

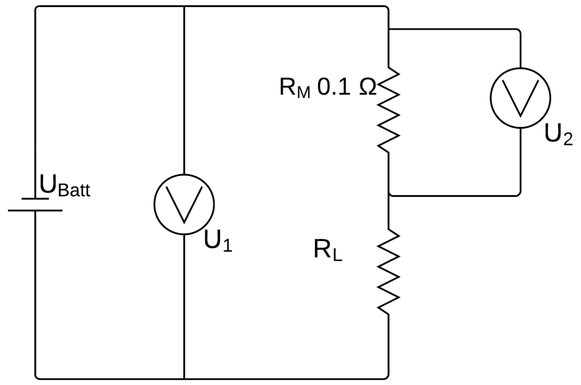
Thus, we employ a custom measurement setup to directly measure the power drained from the battery of a Galaxy S4 given different load scenarios. Figure 3 illustrates the measurement setup.

The battery of the phone is removed and the phone and battery is connected to a 16 bit measurement card, the USB-1608FS Plus (cf. Fig. 3(a)). Given the measurement setup – as illustrated by the simple circuit diagram in Figure 3(b) – the power consumption can be calculated as $P = \frac{(U_1 \cdot U_2)}{R_M}$, where U_1 and U_2 are the voltages measured using the measurement card. The maximum absolute measurement error of this setup is 2.47%. For a detailed analysis of the error the reader is referred to [4].

From the software perspective, we preloaded the Kraken.me application on a Galaxy S4 using the standard Android 4.4.2 ROM. All background services not required for running the operating system or accessing the sensors are deactivated. The display was switched on but set to the lowest possible brightness.

Each measurement scenario – which usually consists of several different configurations for one sensor – was measured as follows: First all sensors were deactivated to measure the current idle consumption. Then each of the configurations was tested consecutively. Between different configurations all sensors were again deactivated, the display was set to the maximum brightness and vibration was switched on for 4 seconds. The resulting spike in energy consumption was used to automatically separate different configurations (cf. Figure 4).


(a) Ongoing measurement



(b) Circuit diagram

Fig. 3: Measurement Setup

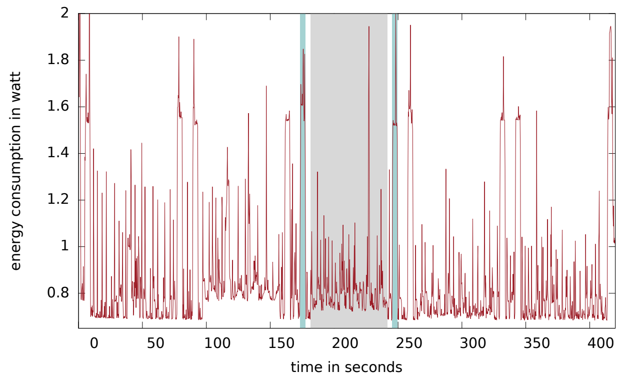


Fig. 4: Plot of energy consumption for one evaluation run with spikes in consumption for the automatic separation of different configurations (Raw data is measured at 10000Hz. It was averaged over 500ms to produce this plot)

For any given scenario this measurement process was repeated at least five times. To quantify the energy consumption for each configuration, we subtract the corresponding idle consumption. Given these values, we are able to calculate average, median and standard deviation of any given configuration.

B. Energy consumption of individual sensors

The mobile application of Kraken.me captures data from several sensors at once. However, we want to understand the consumption pattern of individual sensors first, before going into detail about the overall energy consumption. This will give us a deeper understanding of the options available for hardware sensors. It is also the first evaluation of the software sensors as implemented for Kraken.me. We will first provide an in-depth lock into a subset of the hardware and software sensors. All other sensors are evaluated in comparison.

1) Location: In literature location is well established as one of the worst sensors with respect to energy consumption [5]. Hence, it will provide a base line for other sensors to compare against. As Kraken.me uses the Android LocationManager this does also provide an evaluation of the different possible accuracy settings (no power, low power, balance, high accuracy) available. The sensor was activated for 60 seconds and two refresh rates (10 and 60 seconds) are evaluated. The resulting plot is given in Figure 5.

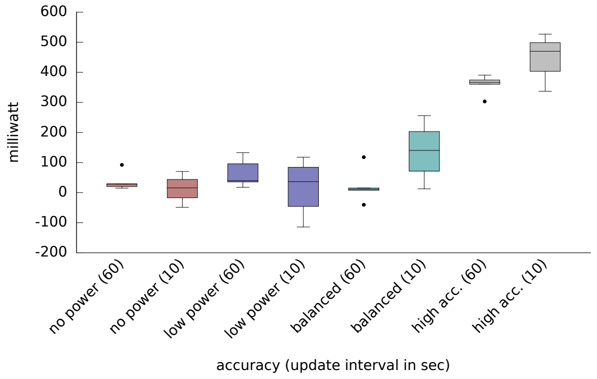


Fig. 5: Power consumption of the location sensor given different refresh rates and accuracy settings

As expected higher accuracy settings lead to higher energy consumption. The settings *No Power* and *Low power* on average consume 13.4 mW and 19.2 mW respectively, *High accuracy* consumes 451.0 mW on average. However, only high accuracy activates GPS, while all other settings use either GSM or wifi triangulation. Wifi triangulation, especially on Android, has been shown to be rather accurate [6]. We therefore recommend to use highest accuracy only if an application relies on location-only, e.g., navigation.

2) Accelerometer: The accelerometer is used for different applications from display orientation to games. It is, hence, save to say that a lot of effort has gone into providing good APIs and energy-efficient accelerometers hardware in the past. Android offers four different settings for the accelerometer (normal, UI, game, fastest). These API levels match envisioned application scenarios, with the lowest sampling rate offered by *normal* all the way up to *fastest*.

The higher power consumption of a higher sampling rate is clearly visible in Figure 6. Interesting is the power consumption of the configuration *game*. It shows a high variance in power consumption, leading us to conclude that the sampling rate is adjusted according to the user's activity. Measurements

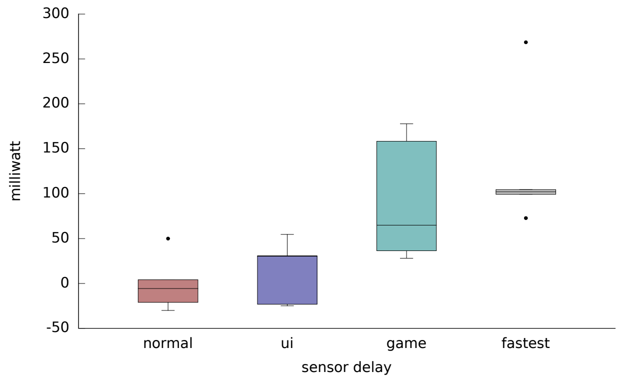


Fig. 6: Power consumption of the accelerometer

show that the light sensor – which offers the same settings – exhibits a similar behavior, hence it is excluded from detailed discussion. In the next section, we focus on the *activity* sensor as the last hard sensor.

3) Activity: Android offers an API to retrieve a guess on the activity of the user. It uses a variety of sensors to reason on the current activity giving results such as *In vehicle*, *Still* etc. In the following evaluation the sensor is queried at intervals of 5, 10, 20, and 60 seconds with a total measurement time of 60 or 120 seconds depending on interval length. Again, we would expect the power consumption to increase as the granularity increases.

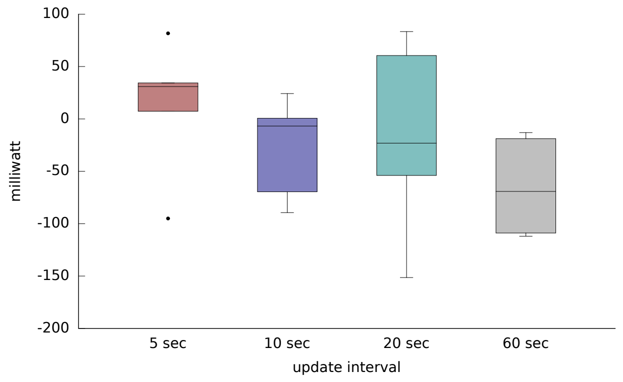


Fig. 7: Power consumption of the activity sensor

However, the results in Figure 7 indicate that the sensors requires close to no power. The results are even below 0. This is possible due to the variance between measurements. Comparing the consumption against the idle consumption (cf. Fig. 8) the confidence intervals overlap.

However, this implies no additional power usage of the activity sensor, as Android seems to be gathering this data in the background. A simple query of the activity sensor is not going to use additional energy. This is important for any application considering activity detection. Implementing any custom activity recognition approach will always consume additional energy, while Android's approach won't. Even if the accuracy is slightly better the trade-off might not be satisfying.

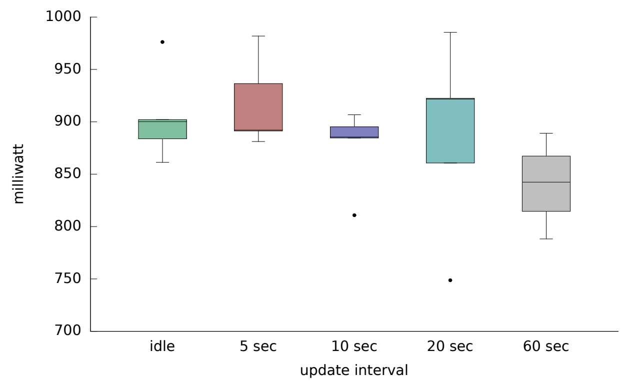


Fig. 8: Power consumption of the activity sensor compared to the idle consumption

4) Calendar: Until now we have focused on physical sensors. Here, we were able to verify expected results. The next step is to focus our attention on software sensors. These differ in a number of key aspects. First, there is no dedicated sensor hardware which drains power. Second, there is usually no API available for easy access and with different configurations. These sensors are custom implementations. Thus, this evaluation is valid only for our implementation. However, we expect our experience to help others designing their own sensors.

The soft sensor described in detail is the calender sensor. Kraken.me requires a full view of the calendar and all subsequent modifications. This is realized by doing an initial scan of all calender items. Afterward the sensor is queried only after the call back, if the calendar entries are changed. As stated before, this requires us to retrieve the whole database again and manually calculate the difference.

To evaluate both the initial scan and the calculation of the difference we set up the following two scenarios. The database was initially filled with 50 or 100 entries respectively. After the initial scan was done one entry was changed and a rescan was done. Hence, we can report results for initial scan and rescan for both 50 and 100 entries.

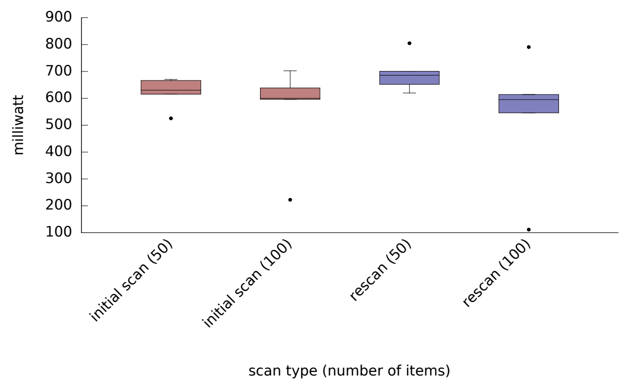


Fig. 9: Power consumption of the calendar sensor

Figure 9 illustrates the results. All runs need virtually the same amount of energy independent of the number of entries. Also initial scan and rescan are basically the same. Querying the API and storing the entries in the database seems to consume a high amount of energy. The amount of energy is even higher than querying GPS for location. Hence, there is a second takeaway. Do not query this API very often. As already stated this is not trivial as Android fires calendar update events not only when something is changed but also after synchronization. Our implementation uses timeouts to filter these events, but optimization could go even further. If no real time tracking is needed, it is possible to calculate the difference once per day or only when charging. We might even want to limit the consumption to a certain percentage of the overall battery power of the phone. Notable the other software sensors behave similar and are, thus, not given in full detail.

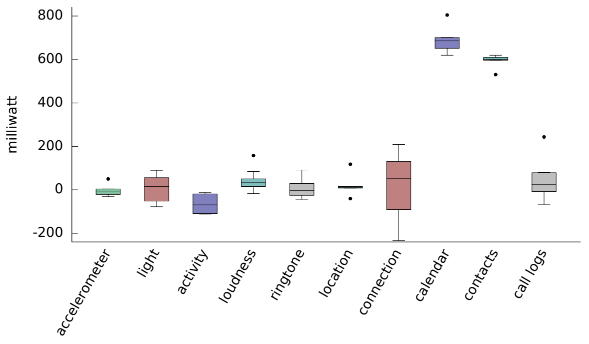
5) Comparison: This section will summarize the results by plotting the power consumption for each sensor given the lowest and highest setting for sampling rate. For ringtone and loudness there was no significant difference between sensor configurations. Hence, all measurements are used to calculate the box plot. An overview of the settings used is given in Table III.

Sensor	*High sampling rate*	*Low sampling rate*
Accelerometer	SENSOR_DELAY_ FASTEST	SENSOR_DELAY_ NORMAL
Activity	5 sec.	60 sec.
Light	SENSOR_DELAY_ FASTEST	SENSOR_DELAY_ NORMAL
Location	HIGH_ACCURACY	BALANCED_POWER_ ACCURACY
Location Interval	10 sec.	60 sec.
Loudness	5, 10, 20, 60 sec.	5, 10, 20, 60 sec.
Ringtone	5, 10, 20, 60 sec.	5, 10, 20, 60 sec.
Calendar	Rescan, 50	Rescan, 50
Contacts	Rescan, 50	Rescan, 50
Call-Log	Rescan, 10	Rescan, 10

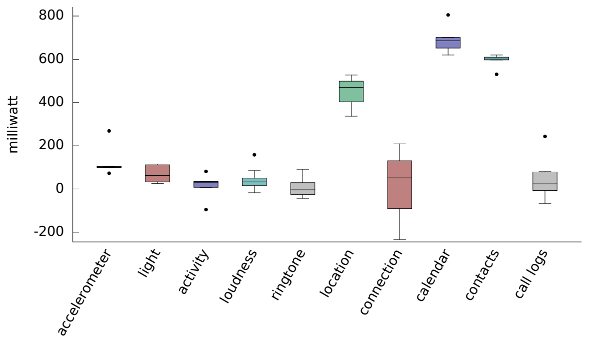
TABLE III: Configuration parameters for each sensor

Figure 10 summarizes the results. For most physical sensors there seems to be no difference between both configurations. We will see in the next section that the difference will add up if all sensors are activated at once. Location is an outlier. This is due to the reasons stated above. Location relies on additional hardware to provide better accuracy yielding a much higher power consumption.

The software sensors consume a rather large amount of energy, but only when triggered. Hence, a lot of thought should be given on when to trigger these sensors. The standard trigger would be using the callback provided by the operating system. This would then very much depend on the user behavior. Techniques with fixed intervals might work, depending on the real-time constraints of the application. In summary the work that has been put into making physical sensors as energy-efficient as possible has been paying of. Depending on the sensor it might basically come for free from a power consumption perspective. The same is not true for software sensors and there is still room for optimization and research.



(a) Low sampling rate



(b) High sampling rate

Fig. 10: Power consumption of each sensor individually

C. Energy Consumption of Kraken.me mobile

Applying the knowledge we gained from evaluating individual sensors, we implemented two different configurations for Kraken.me mobile[4]. The first configuration (fast) is about gathering real-time data with the highest possible granularity, while the second configuration (slow) is about a more balanced approach. The configurations are given in Table IV. Software sensors can be ignored as they are currently independent of the configuration.

Sensor	*fast*	*slow*
Accelerometer	SENSOR_DELAY_ FASTEST	SENSOR_DELAY_ NORMAL
Activity	10 sec.	120 sec.
Light	SENSOR_DELAY_ FASTEST	SENSOR_DELAY_ NORMAL
Location	HIGH_ACCURACY	BALANCED_POWER_ ACCURACY
Loudness	10 sec.	120 sec.
Ringtone	10 sec.	120 sec.

TABLE IV: Configuration parameters for Kraken.me

The hypothesis is obvious: While *fast* provides high tracking granularity, *slow* should decrease power consumption significantly. Figure 11 illustrates the difference in power consumption.

Overall *fast* consumes $599.4mW$ on average, *slow* consumes only $128.9mW$ on average. Thus, *slow* consumes only around 22% of the power. Reducing the granularity further yields only diminishing returns in terms of power consumption, but reduces the data quantity to a point were real-time tracking for e.g., personal assistance, is not possible anymore.

Most of the reduction comes from using the balanced location method. However, the overall reduction is much more pronounced than anticipated. The small difference for each of the single sensors adds up to explain these results. In summary, the results have lead us to configure the Kraken.me application using the *slow* configuration. This still has an impact on battery life, but it is much less than anticipated for multi-sensor all day tracking.

[4]Please note that these configurations do not match the settings for the sensor comparison.

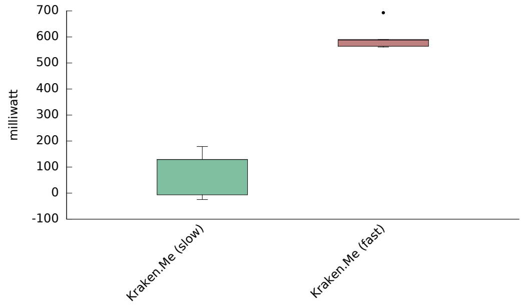


Fig. 11: Energy consumption for Kraken.me given the *fast* and *slow* configurations

IV. Related Work

Smartphones are expected to last through at least one working day. Hence, a lot of research has gone into understanding and measuring power consumption. Here, the power consumption of smartphones may be derived by either measuring the power consumption directly, or using device dependent power models in combination with the system utilization [7], [8]. Nacci et al. [9] extend this approach by proposing a framework allowing automatic power model generation. These are then used to suggest the user certain energy conserving actions.

The increasing number and use of sensors make them a major source of power consumption in modern smartphones. However, only due to these sensors has the field mobile sensing gained considerable traction [10], [11]. Both for mobile crowd sensing [12], [13], [5] and mobile context sensing [14], [15], [16].

Considering the power drain, continuous sensing is still hard to achieve. Recently, people have started considering energy-efficient continuous context sensing algorithms [15], [17], [18], [5]. Others have focused on optimizing the network power consumption as the second largest consumer after the display [19], [20]. Based on some of this work, Kansal et al. [2]

analyze the trade-off between accurate sensing and the power consumption of the smartphone, from which they derive a sensor abstraction layer considering the energy consumption. Still, in their current implementation sensors are limited to location context. Given our results, it might be worthwhile extending their abstraction to more sensors and context modalities.

It is important to continue understanding and optimizing the power consumption of these sensors as the use and impact will only increase.

V. Conclusion

With Kraken.me we built a comprehensive mobile tracking application within the Kraken.me framework. It tracks data from hard and soft sensors to derive a comprehensive profile of any given user. We quickly realized that power consumption is the main drawback of any tracking application.

Given the rich tracking profile, we contributed our in-depth analysis on the impact of different sensors and configurations on power consumption. This confirmed some obvious results, e.g., wifi and cell location is cheaper than GPS. It uncovered some interesting observations, e.g., activity recognition is always on in Android, thus consuming no additional power. And last but not least, that software sensors are the least optimized.

This implies an interesting field for further optimization. Either in scheduling software sensors, finding more efficient implementations or sophisticated trade-off energy mechanism.

Acknowledgment

This work has been co-funded by the DFG as part of the CRC 1053 MAKI.

References

[1] N. Banerjee, A. Rahmati, M. D. Corner, S. Rollins, and L. Zhong, *Users and batteries: Interactions and adaptive energy management in mobile systems*. Springer, 2007.

[2] A. Kansal, S. Saponas, A. B. Brush, K. S. McKinley, T. Mytkowicz, and Z. Ryder, "The Latency, Accuracy, and Battery (LAB) Abstraction: Programmer Productivity and Energy Efficiency for Continuous Mobile Context Sensing," in *ACM SIGPLAN international conference on Object oriented programming systems languages & applications*, 2013.

[3] I. Schweizer and B. Schmidt, "Kraken.me - multi-device user tracking suite," in *2nd Workshop on Human Activity Sensing Corpus and Its Application*, 2014.

[4] F. Kaup, P. G. Gottschling, and D. Hausheer, "Powerpi: Measuring and modeling the power consumption of the raspberry pi," in *39th Annual IEEE Conference on Local Computer Networks*, 2014.

[5] Z. Zhuang, K.-H. Kim, and J. P. Singh, "Improving energy efficiency of location sensing on smartphones," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 315–330.

[6] N. Brouwers and M. Woehrle, "Detecting dwelling in urban environments using gps, wifi, and geolocation measurements," in *2nd Intl Workshop on Sensing Applications on Mobile Phones*, 2011.

[7] L. Zhang, B. Tiwana, R. P. Dick, Z. Qian, Z. Mao, Z. Wang, and L. Yang, "Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones," in *CODES + ISSS'10*. ACM, Oct. 2010.

[8] L.-T. Duan, B. Guo, Y. Shen, Y. Wang, and W.-L. Zhang, "Energy analysis and prediction for applications on smartphones," *Journal of Systems Architecture*, vol. 59, no. 10, pp. 1375–1382, Nov. 2013.

[9] a. a. Nacci, F. Trovò, F. Maggi, M. Ferroni, A. Cazzola, D. Sciuto, and M. D. Santambrogio, "Adaptive and Flexible Smartphone Power Modeling," *Mobile Networks and Applications*, Oct. 2013. [Online]. Available: http://link.springer.com/10.1007/s11036-013-0470-y

[10] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell, "A survey of mobile phone sensing," *Communications Magazine, IEEE*, vol. 48, no. 9, pp. 140–150, 2010.

[11] W. Z. Khan, Y. Xiang, M. Y. Aalsalem, and Q. Arshad, "Mobile phone sensing systems: A survey," *Communications Surveys & Tutorials, IEEE*, vol. 15, no. 1, pp. 402–427, 2013.

[12] I. Schweizer, R. Bärtl, A. Schulz, F. Probst, and M. Mühlhäuser, "NoiseMap - Real-time participatory noise maps," in *2nd Intl Workshop on Sensing Applications on Mobile Phones*, 2011.

[13] M.-R. Ra, B. Liu, T. F. La Porta, and R. Govindan, "Medusa: A programming framework for crowd-sensing applications," in *Proceedings of the 10th international conference on Mobile systems, applications, and services*. ACM, 2012, pp. 337–350.

[14] H. Lu, A. B. Brush, B. Priyantha, A. K. Karlson, and J. Liu, "Speakersense: energy efficient unobtrusive speaker identification on mobile phones," in *Pervasive Computing*. Springer, 2011, pp. 188–205.

[15] D. H. Kim, Y. Kim, D. Estrin, and M. B. Srivastava, "Sensloc: sensing everyday places and paths using less energy," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*. ACM, 2010, pp. 43–56.

[16] K. K. Rachuri, M. Musolesi, C. Mascolo, P. J. Rentfrow, C. Longworth, and A. Aucinas, "Emotionsense: a mobile phones based adaptive platform for experimental social psychology research," in *Proceedings of the 12th ACM international conference on Ubiquitous computing*. ACM, 2010, pp. 281–290.

[17] J. Paek, J. Kim, and R. Govindan, "Energy-efficient rate-adaptive gps-based positioning for smartphones," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 299–314.

[18] Y. Wang, J. Lin, M. Annavaram, Q. A. Jacobson, J. Hong, B. Krishnamachari, and N. Sadeh, "A framework of energy efficient mobile sensing for automatic user state recognition," in *Proceedings of the 7th international conference on Mobile systems, applications, and services*. ACM, 2009, pp. 179–192.

[19] U. Rathnayake, H. Petander, M. Ott, and A. Seneviratne, "EMUNE: Architecture for Mobile Data Transfer Scheduling with Network Availability Predictions," *Mobile Networks and Applications*, vol. 17, no. 2, pp. 216–233, Jun. 2012.

[20] A. Blenk, W. Kellerer, F. Wamser, T. Zinner, and P. Tran-Gia, "Dynamic HTTP Download Scheduling with Respect to Energy Consumption," in *Tyrrhenian International Workshop on Digital Communications - Green ICT (TIWDC)*, 2013.