Technical Report

Energy-Efficient GPS-Based Positioning in the Android Operating System

Jochen Streicher, Olaf Spinczyk

03/2011

technische universität dortmund

SFB 876 Verfügbarkeit von Information durch Analyse unter Ressourcenbeschränkung

Part of the work on this technical report has been supported by Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876 "Providing Information by Resource-Constrained Analysis", project A1.

Speaker: Prof. Dr. Katharina Morik
Address: TU Dortmund University
Joseph-von-Fraunhofer-Str. 23
D-44227 Dortmund
Web: http://sfb876.tu-dortmund.de

Abstract

We present our ongoing collaborative work on EnDroid, an energy-efficient GPS-based positioning system for the Android Operating System. EnDroid is based on the *EnTracked* positioning system, developed at the University of Aarhus, Denmark. We describe the current prototypical state of our implementation and present our experiences and conclusions from preliminarily evaluating EnDroid on the Google Nexus One Smartphone. Although the preliminary results seem to support the approach, there are still several open questions, both at the application interface, as well as at the hardware management level.

1 Introduction

Today's smartphones exhibit a multitude of environment sensing possibilities, allowing a plethora of context-oriented applications to provide a rich user experience. One particular type of context-aware applications are *location-based applications* (LBAs) that have to be provided with the phone's absolute position to fulfill their tasks. Examples of such applications are geo-based information applications [1] or proximity and social networking applications [10, 7] just to mention a few. Some of these applications actually require continuous high-accuracy location updates, like those focusing on traffic [5] or healthcare.

With limitations, a smartphone can determine its current position using the mobile network or WiFi. However a lot of LBAs need more location information that is more accurate than the position calculated by GSM signal triangulation. WiFi-based positioning may yield good results when done indoors or in dense urban areas, but performs poorly in less densely inhabited areas, where it might not be available at all [3]. For outdoor positioning in suburban or rural areas, the *global positioning system* (GPS) is the option of choice, when it comes to accuracy. However, GPS is by far the most power-consuming positioning method and also one of the most power-hungry components even in modern smartphones [2]. As battery life is probably the scarcest resource on these devices, mobile phone owner's tend to switch off GPS positioning entirely when they do not need it explicitly, which is in fact a roadblock for the widespread use of emerging location-based applications [9].

Usually, mobile operating systems give their applications some limited possibilities to reduce positioning costs. This ranges from simply choosing a less power-consuming, but also less accurate location provider (like GSM-based positioning) to more fine-grained parameters, like the demanded frequency of position updates from the GPS. However, the influence of those parameters on actual power consumption and the resulting quality of the positioning service also depends not only on various external conditions, but also on the specific hardware executing the application. We do not believe that application developers should be responsible for optimizing application's for every possible device. Rather, we believe that this should be part of the system software.

Thus, our goals can be structured as follows: 1.) Location-based Applications should be able to state their demanded quality parameters (e.g., the accuracy) directly, instead of wildly guessing tuning parameters. 2.) The phone owner should not have to manually deactivate location sensing in favor of longer battery life.

The rest of this report is outlined as follows: Section 2 gives a short survey about existing work on energy-efficient positioning. The system we are using for our current work, EnTracked, is discussed briefly in Section 3. The current interface of Android's positioning framework is summarized in Section 4, while Section 5 describes the integration of EnTracked into Android. Section 6 presents an excerpt of our ongoing (preliminary) evaluation, while Section 7 discusses current problems and future work. Section 8 concludes the report.

2 Energy-Efficient Positioning

Various methods have been proposed to reduce the power consumption of location sensing, while still being able to guarantee adherence to given accuracy requirements. One possibility is to concentrate on demanded accuracy alone, while assuming robust values for changing context, like the current speed and the inherent position uncertainty of the current GPS fix. Several adaption techniques have been proposed, ranging from the *substitution* of GPS, in favor of less power-demanding location providers, to finer-grained parameters like the frequency used to get location updates from the GPS.

Incorporating context information allows more aggressive adaption and, thus, less power consumption. Context information includes the error estimation of the obtained positions and information about the phone's movement state, but can also include static or history-based knowledge, like typical routes, maps of signal qualities and observed accuracies for given providers. Context information also provides further adaption methods like sensing *supression.* Besides the aforementioned techniques, power-consumption for positioning can also be reduced by inter-device *cooperation*, for example, by distributing an obtained GPS fix via low-power communication to nearby devices.

Zhuang *et al.* use data from the acceleration sensor, to suppress location sensing when the device is not moving. They furthermore keep a continuously updated map containing information about the network-based location accuracy. Using this map, they proactively substitute the GPS with network-based positioning, wherever this is possible [11].

EnLoc [4] schedules different location providers while trying to maintain a demanded *average* accuracy. For this purpose it also incorporates the habitual mobility of the phone owner to predict locationts. Bluetooth-based localization is considered as an additional location provider in the *a-Loc* positioning system *[8]*, which uses a provider scheduling algorithm based on cost- and accuracy-models for location providers. They furthermore try to dynamically infer an application's accuracy requirements. For example, they use the spatial density of a specific type of interesting locations (e.g., restaurants) as an input therefore, when one of these locations has to be found. Besides the already introduced concepts of fix interval adaption and GPS availability mapping, *RAPS [9]* also implements inter-device cooperation via Bluetooth.

While the work of Zhuang and also a-Loc were implemented on top of Android, none of the existing approaches were actually integrated into the operating system.

3 Entracked

EnTracked focuses solely on GPS-based positioning. It uses dynamically changing context as well as a static power model of the device to schedule GPS fixes in an energy-efficient way, while maintaining a given location accuracy. The accuracy is provided as the maximum allowed distance d_{limit} between the current *position belief* and the phone's actual position. While this section just gives a brief summary on EnTracked's principles, the original EnTracked publication [6] provides a detailed technical introduction.

3.1 QoS- and Context-Driven GPS Scheduling

When a GPS fix is obtained, various contextual information is used to determine when the next absolute position is required in order to maintain the demanded accuracy of the current position belief. The location information obtained from today's GPS devices usually contains also information about the uncertainty u_{GPS} of the position. Together with the time since the last GPS position fix t_{GPS} and a robust speed estimation v_{est}, derived from GPS data, the following error model inspired by Farell *et al.* for the current time t is formed:

$$e_{model} = u_{GPS} + (tt_{GPS})v_{est} \tag{1}$$

Given the accuracy requirement d_{limit}, the time until the next location information has to be acquired can then be inferred using Equation 2.

$$t_{limit} = \frac{d_{limit} - e_{model}}{v_{est}} \tag{2}$$

Optionally, EnTracked first detects if the device is moving and, if this is not the case, skips the above calculations and puts the device into still mode. In still mode, continuous motion detection takes place to re-trigger actual location sensing when the device is moving again. The motion detection works by sampling the acceleration sensor every 50ms for about 1.2s, and then computes their per-axis variance. If that variance is above a certain threshold, EnTracked decides that the device has started moving again and leaves still mode. If that is not the case, the device can sleep again for 3 seconds. In still mode, the GPS operates with extremely low frequency, with an interval of 5 minutes between position fixes.

3.2 Device Modeling

The exact point in time, when the next position fix has to be available, is not yet sufficient. The *time to fix* (TTF) the GPS needs to obtain a position fix after being switched on, depends on the length of the preceding period of inactivity. The reason is that GPS chips usually stay in *tracking mode* for a certain time, after being switched off, which enables them to serve possibly following requests much faster than after a "cold" start. Once the

tracking mode is left, calculation the position again needs more time. Staying in tracking mode, after being switched off also means that the phone's power consumption does not instantly return to the level it was before activating the GPS.

Therefore, EnTracked accurate model of the timing and the energetic behavior of the GPS device in order to determine 1.) whether it is beneficial to switch off the GPS device until the next position fix and 2.) when it has to be switched on again, so as to obtain the next location information in time.

4 Positioning in Android

The positioning API in Android is callback-based: Applications interested in the phone's location can register for being informed when new location information is available. Either for a single update or for continuous updates. Applications requesting continuous updates can control their frequency in two ways simultaneously: Firstly, by specifying a *minimum interval* between subsequent location updates and, secondly, by specifying a *minimum distance* the phone has to travel before the application is told about the new location.

If an application registers for GPS-based location updates, the *interval* parameter directly influences the frequency of actual GPS position fixes. Thus, an application can save energy by increasing this value. If several applications requested periodic location updates, the smallest of all interval parameters is used. The *distance* parameter does not have an effect on the frequency of GPS fixes at all, but rather just decides whether the corresponding application is told about a freshly obtained position.

Applications can further lower the cost of location sensing by choosing a less expensive provider like WiFi or the mobile network. This can be done either explicitly or implicitly, by specifying prioritized coarse criteria regarding the levels of accuracy (5 different settings) and power consumption (3 different settings).

5 EnDroid

The idea for EnDroid was to leave the decision about fix intervals entirely to the existing EnTracked system. To be able to easily integrate updates from the ongoing work on EnTracked itself, we decided to keep the existing Python code base. As Android is essentially Linux-based, it is also able to run a Python interpreter natively. However, the Android API and all its services like positioning and sensor access are not accessible from Python. Furthermore EnTracked needs *exclusive* control over the GPS configuration, which is not even available for Android applications. Thus, EnDroid consists of an algorithmic part, the *Head*, running in a Python VM and a control part, the *Body*, residing in in the application framework that also contain's Android's positioning subsystem, the *Location Manager*. An overview of EnDroid's positioning system, its components and their communication paths, is depicted in Figure 1, which will be explained in the remainder of this section.

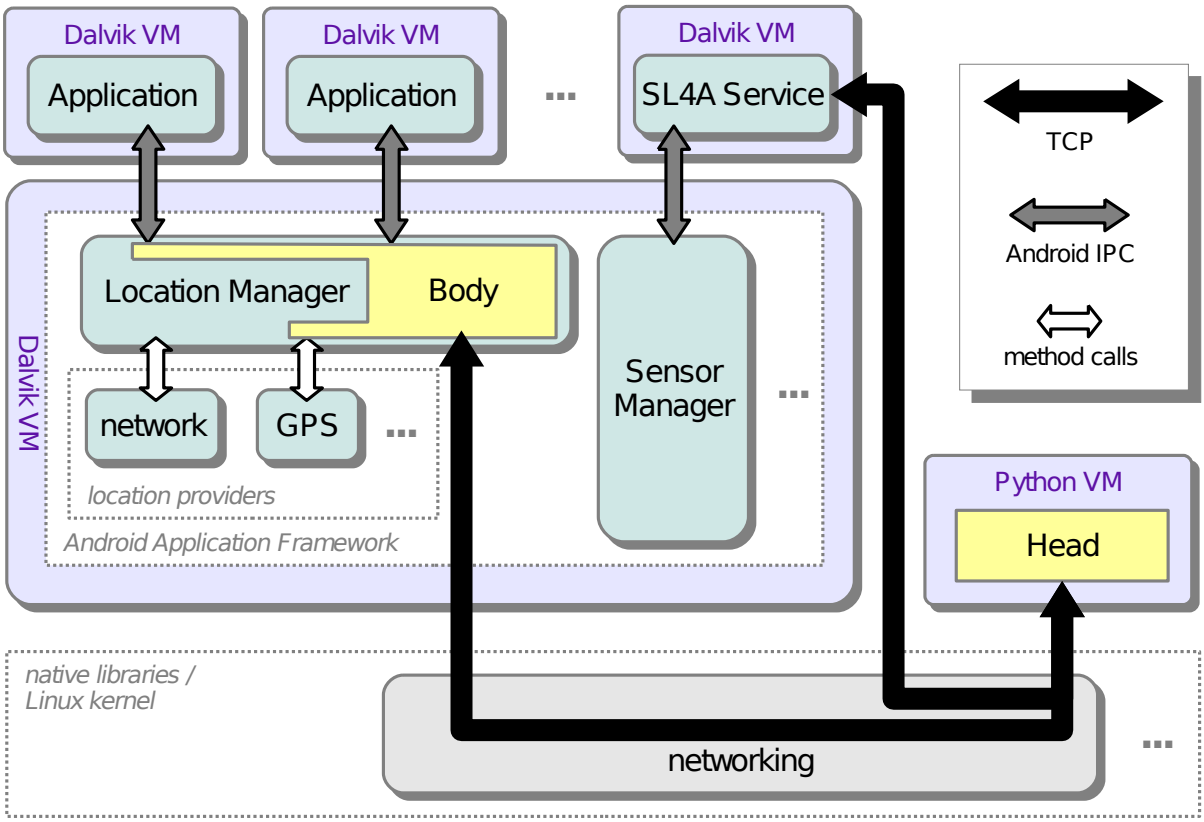

Figure 1: The current EnDroid architecture and its communication paths (bigger and darker means more overHead).

5.1 Android API Access

To be able to use the acceleration sensor via the Android API, we use the *scripting layer for android (SL4A)* application[1], which provides an Android API facade for several scripting languages. The language-independent core consists of an Android service that listens for incoming RPCs (remote procedure calls) from scripts on a local TCP port. These RPCs are encoded as a JSON string containing a method name and respective parameters, which loosely correspond to Android API calls. After reception of an RPC, the service invokes the corresponding method of the Android API. If that method has a return value, it is transported back to the script via the same mechanism.

On the script side resides a corresponding RPC interface. In the case of Python, this is an object that sends an RPC when one of its methods is called. When the SL4A service receives asynchronous responses from Android, like it is the case with location or sensor updates, their data is stored in an event buffer. Scripts can fetch events from this buffer via an according RPC call. This call can optionally block if no event is available, allowing the script to wait for an event without using the CPU.

[1] http://code.google.com/p/android-scripting/

5.2 Exclusive GPS Management

Every request for location updates by Android applications is handled and being kept track of by the *Location Manager*, which is also responsible for distributing obtained location information to these applications. The Location Manager further keeps track of available location providers and decides which providers to use in order to fulfill the request parameters (criteria, interval and distance).

The EnDroid Body intercepts requests from applications to the location manager in order to be informed about the current positioning requrements. Currently, this is just the information if the GPS is needed at all. Furthermore, it has to control the GPS location provider. The GPS location provider's interface is, apart from the various types of status update it reports, quite simple. The Location Manager can request a single fix, activate and deactivate location tracking and adjust the fix interval.

The communication with the Head is again performed by a TCP channel, which is more of a quick hack than a design decision. However, this channel is currently used only for GPS and alarm commands (Head to Body) and location information (Body to Head), which are sent with relatively low frequency ($\leq$ 1Hz). The amount this channel adds to the total energy consumption should therefore be negligible.

The alarm command is used to wake up the Head after a certain time. As the Head is just a port of EnTracked, it performs GPS scheduling like it did on its original platform, the *Nokia N series*: 1.) enable the GPS, 2.) receive a position fix, 3.) disable the GPS, 4.) *sleep* for the calculated time, and 5.) request the next position fix. When a process "sleeps", it tells the operating system to be omitted from CPU scheduling for a certain time. When that time is up, the process takes part again in CPU scheduling. In Android, however, it may sleep for a much longer time, since Android *suspends* when this is not *explicitly prevented*. This is unlike desktop Linux systems, which only go into suspend mode (also known as *suspend to RAM*) when this is *explicitly requested*. This means, if the Head simply executes the first four of the five steps stated above, GPS tracking may stall completely for a long time.

However, if an application requests periodic GPS fixes in an *unmodified* Android, the device may suspend between the fixes, but *does* wake up when the inter-fix time has elapsed.

Modern GPS chips schedule periodic position updates on their own, and wake up the CPU, when a they have obtained a position fix. If the GPS chip is not capable of scheduling, the location manager makes use of the *alarm manager*, which arms a special hardware timer that is able to wake the device from suspend mode.

Therefore, to ensure this behavior, the Head sends an alarm command to the Body and waits for an answer on the TCP channel after switching off the GPS. The Body uses an alarm to resume device operation at the given time and sends its answer to the Head.

Apart from the need of an alarm, disabling the GPS after every fix is generally not very smart, as we thereby restrict the chip's own energy-saving possibilities. Therefore we extended the Body such that the Head can alternatively specify an interval instead of switching the GPS off and on again.

6 Preliminary Evaluation

We performed some preliminary measurements to get an impression as to whether the EnDroid approach is beneficial at all in terms of power consumption. This must not be taken for granted since the original measurements were performed on an older generation of smartphones and with EnTracked just being an application and not an operating system part.

6.1 Measurement Method

Our evaluation was done entirely on a *Google Nexus One* (N1) smartphone, running as well an original and an EnDroid variant of Android 2.3.5. If not stated otherwise, Radio (2G/3G) was disabled, while WiFi was enabled and being associated with an access point in order to enable AGPS communication.[2]

A convenient way to obtain indicative power measurements is the internal sensor of the N1 battery. It provides values for current, averaged current, voltage and battery charge level. These values are updated approximately every 50 seconds. The sensor readings are quantized in steps of 4.886mV for Voltage, 67 ţA for current and averaged current, and 1.6mAh for the charge level. Their actual precision is unknown to us, but they are known to be influenced by temperature. In summary, the coarse temporal and spatial resolution, the temperature sensitivity, probably also the precision in general, and the general variability of battery characteristics over time render the internal sensor readings hardly suited for an extensive evaluation.

However, since many of the measurements had to be performed outside, in an environment hardly suited for expensive measurement hardware, we decided to base the preliminary comparisons on the internal measurements.

To compute the approximate average power consumption during a certain time interval, we tried two different methods. The first one calculates the approximate average power consumption $\overline{P}^I$ based on the voltage and average current readings. The second one, which we assumed to be more robust, calculates the approximate average power consumption $\overline{P}^Q$ based on the voltage and battery charge level readings.

From the battery sensor readings, we used the charge level, represented by $Q^B(t)$, the voltage, represented by $U^B(t)$ and the averaged current, represented by $I^B(t)$. Since the readings are not updated continuously, these functions are defined as right-continuous step functions.

[2]Modern GPS chips are usually capable of *Assisted GPS (AGPS)*. When a position fix is requested from an AGPS-capable chip, it first connects to an AGPS server and downloads an *almanach,* which contains information about current satellite positions. Using this information, the first position fix can be obtained in seconds instead of minutes.

6.1.1 Calculation Based on Average Current Readings

For the calculation based on the average current reading, we calculate the approximate energy consumed in an interval $[t_o; t_1]$:

$$W^I([t_o; t_1]) = \int_{t_0}^{t_1} U^B(t) I^B(t) dt \tag{3}$$

The approximate power is the calculated as follows:

$$\overline{P}^I([t_o; t_1]) = \frac{1}{t_1 - t_0} W^I([t_0; t_1]) \tag{4}$$

6.1.2 Calculation based on Battery Charge Level Readings

For the discharge-based calculation, we do not use the sensor reading for average current, but rather define the average current in a given time interval $[a; b]$ as:

$$\overline{I}^Q([a; b]) = Q^B(b) - Q^B(a) \tag{5}$$

Unfortunatly, since the readings are not updated continuously, this is not valid for arbitrary intervals $[a; b]$. Therefore, we define a function T that splits a given time interval into the subintervals between sensor updates, with the set S being defined as the set of all points in time that correspond to a new charge level reading.

$$T([t_o; t_1]) = [x, min\{y \| y \in S \cup t_1 \wedge y > x\}) \| x \in S \cup \{t_0\}) \tag{6}$$

The energy consumed in an interval $[t_o; t_1]$, based on the charge state readings, is approximated as:

$$W^Q([t_o; t_1]) = \frac{1}{t_1 - t_0} \sum_{J=T([t_0;t_1])} \overline{I}^Q(J) \int_J U^B(t) dt \tag{7}$$

The approximate average power, based on the battery charge level reading, is then calculated as follows:

$$\overline{P}^Q([t_o; t_1]) = \frac{1}{t_1 - t_0} W^Q([t_0; t_1]) \tag{8}$$

Of course, an additional error is introduced, if t_o or t_1 are not in S.

6.2 Measurements

For the measurements, we developed a simple application that requests periodic location updates with a given fix interval, and writes them to the internal SD card. Additionally, the current battery sensor readings are written.

For the Android-based measurements we tried different fix intervals. We compared these to EnDroid with manual GPS scheduling, both with movement detection disabled (MS) and and enabled (MD), but also with chip-based scheduling (CS). The combination of chip scheduling and movement detection is not implemented yet. The desired accuracy (distance limit) for EnTracked was always 50m. The experiments were run either outside or inside, near a window.

Since one of our goals is to eliminate the need for manually switching the GPS off when it is actually not required, we are especially interested in the power consumption during still periods. Table 1 shows the average current values obtained from these measurements with the N1 being in different motion states. However, these figures are to be understood only tentatively, since several experiment parameters that should have been under explicit control were not, e.g., temperature, battery charge state and possibly disrupting WiFi signals. Furthermore, we performed no assessment regarding the actually resulting positioning error, as it was done in the original EnTracked paper. A few of the measurements were indeed obtained simultaneously (those with *), underlying almost identical external conditions, both with fully charged batteries. For the other two non-still scenarios, the same route was taken for every system variant.

The figures obtained by the charge-level-based calculation are in black, while those obtained by the average-power-based calculation have blue color. The figures obtained by the charge-level-based method obviously make less sense, since there is no reason why the inside measurements should be better than those performed outside. The quality of the GPS signals is expected to be better outside and, thus, the power consumption should be lower. A follow-up experiment showed, that the charge state reading is, in fact, strongly influenced by temperature. Measuring the charge level while letting the phone cool down outside, and warming it up back inside, resulted in a rising charge level reading, while the battery was actually discharging.

6.3 Results

Although being tentative, the results seem to indicate that EnDroid can, in fact, prolong battery life. Given the demanded maximum distance error 50m, assuming a walking speed of $2\frac{m}{s}$ and a *worst-case* GPS positioning error of 30m, the interval between GPS position fixes has to be no larger then 10s, according to Equation 2. EnDroid adapts the interval according to the actual GPS positioning accuracy, which may be much better.

Furthermore, we learned that chip-based scheduling probably yields better results than manual scheduling. However, with rapidly changing fix intervals, like it is the case when the phone transitions between different movement states, the actual position fixes often arrived with several seconds delay. This, however, invalidates the robustness assessment performed in the original EnTracked paper. The results also indicate that the energy savings gained by motion detection and still mode are currently almost negligible. To allow the phone's user to be oblivious to whether GPS is activated or not, while the location does not change, we should come close to the energy consumption in the suspended state, which is consumes about 20mW on the Nexus One under the conditions stated at the beginning and about 17mW when WiFi is switched off.

	Android									EnDroid (50m)		
	1s	2s	3s	5s	10s	15s	20s	30s	40s	CS	MS	MD
still, inside (20℃)			425 421		272 276					317 298	347 346	233 231
still, outside (6℃)	674 527	573 428		476 374	338 313	359 326	298 284	275 241	229 181	288 262	364 253	372 234
walking, outside (15℃)*					437 395					262 245		
walking, outside (15℃)*	549 497									437 359		
driving, car (6℃)			571 452							550 411		539 416
walking, outside (15℃)					243 322							224 161

Table 1: Average power consumption in mW for different configurations and scenarios.

Further investigation showed that, during the acceleration measurement phase, the phone consumed 300mW. After experimenting with lower sampling frequencies, we discovered a lower bound at 200mW, which stems from SL4A always requesting the highest sampling rate (which is 20Hz in this case) and just filtering afterwards. The additional 100mW result in fact from the communication mechanism between SL4A and Python.

6.4 Future Evaluation

As soon as the remaining issues are resolved, we plan to perform a much more extensive evaluation of our system. This will include a much a better coverage of hardware platforms, tracking parameters, and scenarios as well as measurement accuracy and comparability. Furthermore the resulting positioning error will be evaluated.

7 Discussion and Future Work

7.1 Positioning API

Up to now, we just examined whether the adaption of location sensing according to given quality parameters is beneficial. We used one of these parameters, namely accuracy (or

distance limit) and specified it globally, per experiment. In the future, however, the applications should specify their positioning quality requirements.

7.1.1 State of The Art

To get an impression of the potential energy savings gained by dynamically specified accuracy requirements, we first want use information from the existing API utilization as much as possible. For example, the *minimum distance* parameter of Android's positioning API could be interpreted as a distance limit.

However, since the location updates in Android are in fact also a wake-up mechanism for Android applications we have to be careful about 1.) altering the application's behavior this way and 2.) measuring actually the energy savings caused by the smaller wake-up frequency and thus the smaller CPU utilization.

Therefore, we want to investigate existing location-based applications, whether they use the distance parameter at all, and if they do whether they use it as a wake-up mechanism and or perform energy-intensive operations as a response to the position update.

7.1.2 API Evolution

Obviously, the existing positioning API is not optimally suited to enable application-directed adaption for energy-efficient positioning. On of our later goals is to come up with a better API that clearly separates the concerns of wakeup frequency and accuracy and encourages application developers to specify requirements for both of them independently.

Also, we want to investigate allowing finer-grained possibilities of specifying positioning accuracy. Up to now, we regarded only the maximum allowed distance error (or *worst-case distance error*) as a positioning quality requirement. However, applications might instead or additionally have requirements regarding the average distance error, which was considered in *[4]*. Other applications might have requirements regarding the *worst case distance error*, while still being fine if their position belief violates this requirement for a certain percentage of time. The potential for further reduction of power-consumption by utilizing such finer-grained requirements for GPS scheduling is yet to be determined.

7.2 Under the Hood

Although EnDroid can reduce the power consumed by GPS, especially when the device is not moving, there are still unresolved issues regarding the hardware device management.

7.2.1 GPS Interface

We believe that there is further optimization potential in the decision when and under which circumstances AGPS data should be downloaded. However. looking at the Android sources, it seems that the decision *when* to download AGPS data is entirely done by the chip or at least a proprietary driver. An additional idea we want to investigate is to

utilize *Mobile Station Assisted AGPS (MSA)* for the some of the low-frequency position fixes during motion detection. Using MSA, the chip does not infer its position on its own, but rather sends all available data on current GPS signals to the AGPS server. The AGPS server then calculates the position and sends it back to the chip.

While we learned that chip-based GPS scheduling generally uses less power than manual scheduling and works well for fixed or only slightly altered intervals, it seems to be not very well suited for rapidly adaptive scheduling like EnDroid does with changing transportation modes and speeds. However, we need to do more experiments to get a better understanding of this problem. As a possible solution, we could resort to manual scheduling when the interval decreases too much. Another, probably better solution, would be adapting the interval in advance, *before* the next fix arrives.

7.2.2 Motion Detection

Regarding the motion detection, it became clear that the current SL4A-based implementation is not very efficient. However, even a native Android implementation consumes 200mW when performing the measurements while holding a wakelock.[3] Yet, it is not clear that the power drain is caused mainly by the sensor, since simply holding a *wakelock* and doing nothing else already consumes about 80mW. The wakelock has to be held, because otherwise Android would suspend between the single acceleration measurements. Trying to replace the wakelock with alarms between the measurements resulted in too long time intervals between the measurements (up to 200ms). Since the motion detection is crucial for our second goal as stated in Section 1, we will have to come up with a much less power-draining implementation. However, 80mW seem to be the lower bound for this.

8 Conclusion

We presented an overview on the current state of EnDroid, our energy-efficient positioning system for Android. While tentative results suggest the validity of the approach, a more extensive evaluation is needed. There are still unresolved issues, regarding the application's interface as well as the hardware device management. We take these issues also as hint that positioning should be an integral part of the operating system rather than an application or middleware just set on top of it.

References

[1] Y. Cai and T. Xu. Design, analysis, and implementation of a large-scale real-time location-based information sharing system. In *Proceeding of the 6th international conference on Mobile systems, applications, and services*, pages 106–117. ACM, 2008.

[3] A wakelock is an operating system abstraction that can, amongst other things, prevent Android from suspending.

[2] Aaron Carroll and Gernot Heiser. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, USENIXATC'10, page 21, Berkeley, CA, USA, 2010. USENIX Association.

[3] Y.C. Cheng, Y. Chawathe, A. LaMarca, and J. Krumm. Accuracy characterization for metropolitan-scale wi-fi localization. In *Proceedings of the 3rd international conference on Mobile systems, applications, and services*, pages 233–245. ACM, 2005.

[4] I. Constandache, S. Gaonkar, M. Sayler, R.R. Choudhury, and L. Cox. Enloc: Energy-efficient localization for mobile phones. In *INFOCOM 2009, IEEE*, pages 2716–2720. IEEE, 2009.

[5] J. Eriksson, L. Girod, B. Hull, R. Newton, S. Madden, and H. Balakrishnan. The pothole patrol: using a mobile sensor network for road surface monitoring. In *ACM MobiSys*, 2008.

[6] M.B. Kjærgaard, J. Langdal, T. Godsk, and T. Toftkjær. Entracked: energy-efficient robust position tracking for mobile devices. In *Proceedings of the 7th international conference on Mobile systems, applications, and services*, pages 221–234. ACM, 2009.

[7] Axel Küpper and Georg Treu. Efficient proximity and separation detection among mobile targets for supporting location-based community services. *SIGMOBILE Mob. Comput. Commun. Rev.*, 10:1–12, July 2006.

[8] K. Lin, A. Kansal, D. Lymberopoulos, and F. Zhao. Energy-accuracy aware localization for mobile devices. *Proceedings of ACM MobiSys '10*, 2010.

[9] 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*, pages 299–314. ACM, 2010.

[10] R. Schmohl and U. Baumgarten. The contextual map-a context model for detecting affinity between contexts. In *MOBILe Wireless MiddleWARE: Operating Systems and Applications. Second International Conference, Mobilware 2009, Berlin, Germany, April 28-29, 2009. Revised Selected Papers*, volume 7, page 171. Springer Verlag, 2009.

[11] 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*, pages 315–330. ACM, 2010.