Energy Efficient Proximity Alert on Android

Muhammed Fatih Bulut
Computer Science & Engineering
University at Buffalo, SUNY
mbulut@buffalo.edu

Murat Demirbas
Computer Science & Engineering
University at Buffalo, SUNY
demirbas@buffalo.edu

***Abstract*—The proximity alert service on Android is important as an enabler of ubiquitous location-based services, however, it is also limited in this role due to its excessive energy expenditure. In this paper, we present the design and implementation of an energy-efficient proximity alert service for Android. Our method utilizes the distance to the point of interest and the user's transportation mode in order to dynamically determine the location-sensing interval and the location providers (GPS, GSM, or Wi-Fi) to be used. We implement our method as a middleware service in the Android open source project. Our service, for a realistic scenario, reduces GPS usage by 96.66% and increases battery life time by 75.71% compared to the baseline proximity alert in Android.**

***Keywords*-proximity alert; smartphones; energy-efficiency**

I. Introduction

In the last couple of years, the number of smartphone users in US alone has proliferated to 100 million [1]. Equipped with a touch screen, built-in sensors, one hop access to web, and supreme portability, smartphones position themselves as an alternative to personal computers. In addition, smartphones also enable novel transformative location-based services, thanks to their ubiquitous access to GPS and network localization.

Proximity alert is a prominent example of location-based services, in which a user is monitored to detect whether the user passes nearby of a specific location. All major smartphone operating systems, including Android and iOS, expose a proximity alert functionality via their APIs. Proximity alert enables various applications including: personal location-based reminders, safety apps notifying user of dangerous nearby neighborhoods, location-based auto check-in, call, or text.

A noteworthy category of apps enabled by high-precision proximity alert is *pervasive collaboration applications*. Recently, in our LineKing [1] work, we have utilized proximity alert on the participants' smartphones to detect the time spent at the coffee shop in our campus. Then using the aggregated crowdsourced data and by leveraging on the computational power of the cloud, our system is able to forecast future wait times with less than 3 mins mean absolute error.

[1]Android smartphones constitute 50% of the US market and 75% of the global market for smartphones.

Another significant category of apps enabled by high-precision proximity alert is novel social networking applications. Although some smartphone users may not be acquainted with each other, they may share the same spaces regularly (a concept referred to as familiar strangers). For example, students taking the same class, people working in the same building, or people living in the same neighborhood may fit into this category. These people have relevant information to share with each other, but not the appropriate social connections/channels to do so. High-precision proximity alert service can help fill in this gap.

Unfortunately, the proximity alert service depends on frequent checking of physical location with expensive sensors such as GPS, Wi-Fi, or GSM. However, smartphones remain severely battery constrained: According to the survey conducted by TIME [2], almost 62% of US mobile users wish improvements in battery life; more than any other attractive features such as screen size or lower price.

To alleviate the energy expenditure problems of the proximity alert service, in this paper we propose energy-efficient methods to implement proximity alert on smartphones. We restrict our focus to the Android platform, and on high-precision proximity alert requirements, as in the LineKing and auto check-in applications. (As we discuss later, our method readily extends to low-precision proximity alert requirements and achieve even further energy savings for them.) Our method of proximity alert utilizes distances to the point of interests (POIs) and user's transportation mode (idle, walking or driving) to dynamically decide on location providers and location-sensing interval. Specifically, our contributions are as follows:

1) We identify the problems for proximity alert in Android framework that lead to wasting energy as follows: static location provider strategy (GPS-first strategy), static and frequent periodicity of the location checking, and the lack of utilization of less costly inertial sensors.
2) We propose a method to monitor proximity to POIs in an energy-efficient way. Our method keeps track of distances to the POIs and detects the user's transportation mode (idle, walking or driving) to dynamically

[2]http://bit.ly/QO8Dpn

decide on the location provider strategy and location-sensing interval.

3) We investigate low-power transportation mode classification by utilizing the accelerometer to differentiate between idle, walking and driving modes.
4) We implement our method as a system level service by modifying the LocationManagerService of the Android Open Source Project (AOSP).
5) We conduct experiments in various scenarios and show the strengths and weakness of our proposal. Our results indicated that, for a realistic scenario, *we saved 75.71% of the battery comparing to the baseline implementation of Android's proximity alert by reducing GPS usage by 96.66%.*

Outline of the rest of the paper. In Section II we present the problems in the current Android implementation and discuss possible solutions. We present transportation mode classification in Section III, and in Section IV, we discuss the overall design of our proximity alert framework. We discuss our implementation in Section V, evaluation results in Section VI, and related work in Section VII.

II. Proximity Alert in Android

In Android (similarly in iOS), proximity is defined with respect to a geographical region which is delineated with three parameters: latitude, longitude, and radius. Once a device is within the radius of a region, the entered alert is fired, and once the device is out of the radius, the exited alert is fired. Below we present the problems in the current Android proximity alert framework and discuss solutions to overcome the limitations.

A. Static use of location providers

Android provides a couple alternatives to learn location. As a location provider, Android can use GPS, cell tower or Wi-Fi AP (the latter two collectively referred to as network location). These have different accuracy, availability, and energy consumption. As for accuracy and availability, GPS is more accurate in outdoors but unavailable indoors, whereas network location is mostly available, but more accurate indoors when Wi-Fi AP (WAP) is available. As for energy, Figure 1 shows the energy consumption of different location providers [3]. In 10 hours, GPS spent 82% of the battery in comparison to Network which only spent 36%.

Unfortunately, for proximity alert in Android, location is requested from each provider following a static order without considering the availability, accuracy or energy consumption. If GPS is found to be available then the other providers are ignored, however, this causes the Wi-Fi to turn on, even when it goes unused. Also, even when network location has better accuracy and availability than GPS (e.g., indoors), a high energy-cost is still incurred for GPS.

[3]Experiments are done with the devices that only request corresponding measurement in the background with minimal settings

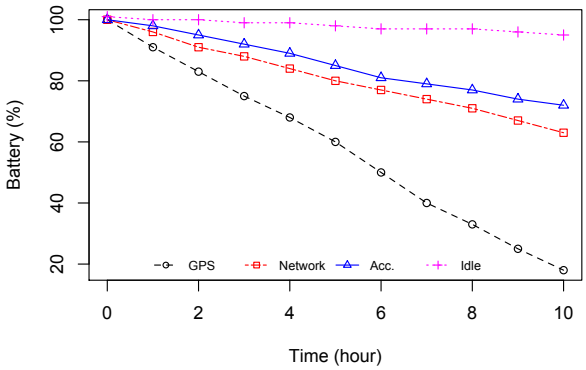

Figure 1. Battery consumption in 10 hours for continuous sensing of GPS, network localization, accelerometer and idle phone.

In addition, distance to the POI, the most important parameter in proximity alert, is also not considered in Android's provider selection process. There is a need for high accuracy localization when the user is close the POI, however if user is far away, then accuracy requirement can be lowered, for example by using cell tower based localization.

B. Static and frequent periodicity of location updates

Android's proximity alert request location in frequent intervals (in seconds), however, this is often unnecessary. When the user is far away from the POI, there is no need to request location with high frequency, as there is no way for the user to pass through the location in such a short time. For a stationary user, this is also an overkill.

Rather than fixing the interval, changing it dynamically based on the distance to the POI and the speed of the user would lead to significant energy savings. However, to realize this, an energy efficient method for determining the user's speed is needed. We resolve this problem by utilizing the accelerometer to decide user's transportation mode and estimate the time of arrival to the POI.

C. Underutilized sensors

Smartphones are equipped with a variety of sensors, which can provide valuable information for proximity alert, but these are unfortunately underutilized by Android. As we discussed above, using the accelerometer, it is possible to determine the mode of transportation of the user (stationary, walking, or driving), using which we can duty cycle the location sensing interval to conserve energy.

Unfortunately, using the accelerometer is not free, rather its energy cost is comparable to GPS and Wi-Fi. Figure 1 shows the continuous cost of using accelerometer in 10 hours in comparison to continuous sampling of GPS and network localization. As the figure indicates, cost of accelerometer is almost similar to network localization only with a small margin of superiority.

As the distance to the POI increases, we can increase the interval of sampling location, so there is a sweet point where sampling the location periodically is more energy-efficient than sampling accelerometer (and deciding the mode for energy saving). Figure 2 shows the sampling periodicity versus the battery level in 10 hours. Horizantal dashed green line represents the remaining battery from the use of accelerometer with 1 minute duty cycle periods. (Assuming that the user is not changing her mode too frequently, this is a reasonable duty-cycle period). As the figure indicates any interval more than 1 minute is good enough for the network localization to be more energy efficient than accelerometer. Similarly, any interval more than 10 minute is good enough for GPS to be more energy efficient than accelerometer. Therefore, we limit the use of accelerometer with distances that can be reached in 1 minute when network location is available, and 10 minute in any other cases. We elaborate on these tradeoffs in Section IV.

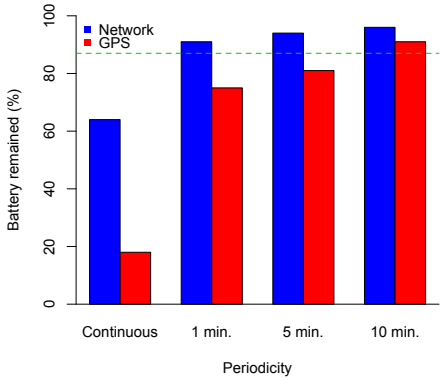

Figure 2. Different periodicity of network and GPS localization for 10 hours. Horizantal green line represents the battery remained for the accelerometer with 1 min. duty cycle period.

III. Deciding on Transportation Mode

In this section, we explain our method for differentiating between different transportation modes. For the sake of simplicity, we focus on three different modes; *idle*, *walking* and *driving*. These modes exhibit differences in accelerometer readings. In idle mode, we do not move much, and this results in regular low variances. Walking causes regular/patterned changes to acceleration in short periods, which leads to large variances. Finally, while driving occasional changes in acceleration occur, but not as much as in walking.

Based on these observations, our method works as follows. We record the accelerometer readings at a sampling rate of 50 Hz for 10 seconds and obtain a total of 500 samples. In order to deal with orientation issues, we adopt the approach in [2] and compute the L2-norm of the raw data. We then calculate the variance and classify the user based on this value. If the variance turns out to be high ($\sigma > 1$), then we classify the activity as walking. If the variance is very low ($\sigma < 0.01$) and persists across consecutive intervals, then we classify it as idle. Finally, if the variance is in between the high and low thresholds, we classify the activity as driving.

	Idle	**Walking**	**Driving**
Idle*	0.98	0.00	0.02
Walking*	0.09	0.80	0.11
Driving*	0.00	0.00	1.00

Table I
CONFUSION MATRIX FOR TRANSPORTATION MODE CLASSIFICATION. ROWS ARE GROUND TRUTHS, WHEREAS COLUMNS ARE THE FINDINGS

This algorithm runs for 6 consecutive 10 seconds interval. In order to eliminate dithering, a majority voting is performed at the end of the minute and outputs the detected transportation mode. The accelerometer is then duty-cycled and turned off for the next minute.

Experiments: We collected a week of accelerometer data and manually labeled the time period when the experimenter is idle, walking, or driving. The resulting confusion matrix from our algorithm is shown in Table I. We classify idle and driving behavior with a high percentage (98% and 100%). Walking is occasionally confused with driving and idle behavior, however, we observe that this happens only in transition states.

IV. System Design

Our proximity alert service consists of three basic components, namely Proximity Alert Manager (PAM), Transportation Mode Classifier (TMC) and Phone State Receiver (PSR). PAM initiates, processes, and controls all the operations including processing location updates and sending directives to other components to start/stop them. TMC is responsible for classifying user's transportation mode (as explained in the previous section). PSR is responsible for letting PAM know about whether Wi-Fi is enabled/disabled and GPS is enabled/disabled. Before delving into the details, we define the following terms:

$\boldsymbol{l_u}$: Location of the user. l_u has latitude ($l_u.lat$), longitude ($l_u.lng$), accuracy ($l_u.accuracy$) and provider ($l_u.pr$, i.e. Network or GPS) information.

$\boldsymbol{l_p}$: Location of the POI. l_P has latitude ($l_p.lat$), longitude ($l_p.lng$) and radius information ($l_p.radius$).

$\boldsymbol{t_u}$: User's transportation mode: idle, walking or driving

$\boldsymbol{d_n}$: User's distance to the targeted geo-point using the WGS84 ellipsoid.

$\boldsymbol{d_c}$: Conservative distance to the target geo-point, defined as:

$$d_c = d_n - l_u.accuracy - l_p.radius \quad (1)$$

$\boldsymbol{d_{TMC}}$: Threshold distance to activate TMC. After d_{TMC}, periodic checking becomes more advantageous than running TMC. Since the cost of location providers differ, we specify two different thresholds, (i.e. d_{TMC_1} and d_{TMC_2}). Former

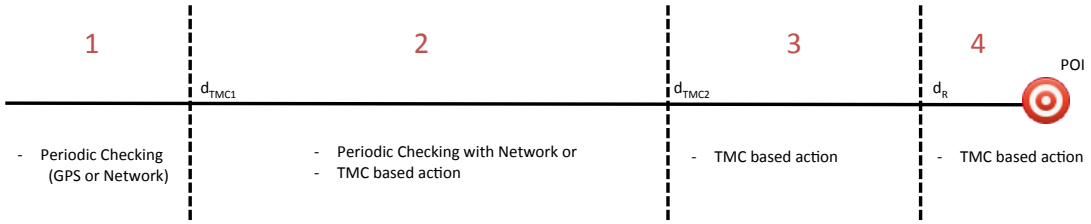

Figure 3. Threshold distances to the POI. Distance is decreasing from left to right.

is the threshold where periodic GPS checking is preferable, and the latter is where periodic network location checking is preferable to TMC.

$\boldsymbol{d_R}$: Critical distance to end periodic checking. After this point our system either listens location continuously or run TMC in frequent periodicity to understand departure from the idle mode.

$\boldsymbol{v_{max}}$: Maximum velocity for driving ($v_{d_{max}} = 60$ mph) and walking ($v_{w_{max}} = 3.1$ mph). These are used for setting the alarm for periodic checking.

Algorithm 1 Proximity Alert Algorithm

```
1:  procedure ProximityAlert(l_u)
2:      d_cmin = Min. d_ci for all of the monitored regions
3:      if d_cmin ≥ d_TMC1 then
4:          Check loc. after d_cmin/v_dmax
5:      else if d_TMC2 ≤ d_cmin < d_TMC1 then
6:          if l_u.pr == Network then
7:              Check loc. after d_cmin/v_dmax
8:          else
9:              if t_u == DRIVING then
10:                 Check loc. after d_cmin/v_dmax
11:             else if t_u == WALKING then
12:                 Check loc. after d_cmin/v_wmax
13:             else if t_c == IDLE then
14:                 Stop location updates
15:             end if
16:         end if
17:     else if d_R ≤ d_cmin < d_TMC2 then
18:         if t_u == DRIVING then
19:             Keep requesting location updates
20:         else if t_u == WALKING then
21:             Check loc. after d_cmin/v_wmax
22:         else if t_c == IDLE then
23:             Stop location updates
24:         end if
25:     else
26:         if t_u == DRIVING OR t_u == WALKING then
27:             Keep requesting location updates
28:         else if t_u == IDLE then
29:             Stop location updates
30:         end if
31:     end if
32: end procedure
```

PAM keeps track of all proximity alerts by monitoring the user's transportation mode and the minimum distances to the registered POI regions. Whenever a location update arrives or user's transportation mode changes, PAM updates the periodic checking parameters by running Algorithm 1.

Threshold distances: We define 3 threshold distances from the user's location, which yields 4 distance intervals to consider (see Figure 3). The first one is d_{TMC_1} which is regarded as a far away location and requesting GPS location periodically is better than observing the transportation mode (similar argument for the second threshold d_{TMC_2} for network localization). As inferred from Figure 2, these distances roughly corresponds to 10 minute for GPS and 1 minute for network. Considering $v_{d_{max}}$, this approximately corresponds to 1 mile and 10 mile from the POI. Third and final threshold is d_R which is regarded as a close point to the POI. Once this threshold is reached, location or TMC needs to be run frequently in order not to miss any POI. We chose d_R as 100 meters. Based on these threshold distances, we dynamically decide the next step as shown in Algorithm 1.

Algorithm 1: We first calculate d_c to all of the monitored regions and select the one with the minimum distance ($d_{c_{min}}$) as the target point. The rest of the operations are based on this min distance and its corresponding distance interval (see Figure 3). If the distance falls into interval 1, then our system stops if TMC is running and sets up an alarm to check location again after $d_{c_{min}}/v_{d_{max}}$ interval.

If the min distance is at interval 2, then we first check if the location provider is network. Since this is a threshold where periodic network location checking is more preferable to TMC, if the location provider is network, then we request location after $d_{c_{min}}/v_{d_{max}}$ interval. Otherwise, we run TMC and observe the user's transportation mode. If the user is driving, then we check the location again after $d_{c_{min}}/v_{d_{max}}$ interval. If the user is walking then we check the location after $d_{c_{min}}/v_{w_{max}}$. Finally if the user is in idle mode, then we stop requesting location updates and run TMC to detect departure from this mode.

If the min distance falls into the third interval, we follow a more conservative approach. If TMC is not running, we start it and observe the user's transportation mode. If the user is driving, then we request location updates continuously as this implies user will be at the target soon. If user is walking then we set up an alarm to check location again after $d_{c_{min}}/v_{w_{max}}$ interval. If the user is idle, then

we stop requesting location updates. Finally if min distance is at the fourth interval, it means user is almost at the target point, so our system request location updates continuously unless user is in idle mode.

Location provider selection: Location is first requested as a network location. If the user's distance is more than d_R and location is not accurate enough, then GPS location is requested. However, if user's distance is smaller than d_R, then we request from GPS unless GPS is unavailable. This way, as the user is getting closer to the POI, we ensure that we get more accurate location information.

V. Implementation

We implemented our proximity alert as a middleware service in Android by modifying Android open source project (AOSP) [3]. Specifically, we modified the parts where the Android proximity alert is implemented, namely the LocationManagerService class located in the framework folder. This class is implemented as a singleton to simplify the management of resources and access control.

All our changes were done in Android JellyBean 4.1 and we approximately added 1KLOC to this service to enable our proposed system. After making the modification, we generated a new custom build with the changes for Nexus S 4G phones.

In addition to the custom build, we also developed an application to perform the experiments in Section VI. This application provides an ability to register and unregister proximity alert on the map. Finally, we recorded battery stats periodically by registering and unregistering the ACTION_BATTERY_CHANGED provided by the Android framework. This action calls the provided BroadcastReceiver to report the battery level. All our logs are stored internally in the device and evaluated offline after the experiments.

VI. Experiments

To evaluate the performance of our service, we conduct experiments for various scenarios. All comparisons are done with the original implementation of proximity alert in the Android platform. We use the Nexus S 4G Sprint phones in all our experiments. We record the GPS usage, network usage, TMC usage, and the battery information during the experiments.

A. Scenario A: Idle device

In this scenario, we conduct an experiment with idle devices which register a proximity alert with different distances. Figure 4-a shows the battery consumption with respect to the distance of the registered proximity alert for 10 hour periods. As shown in the figure, as the distance increases ($d_1 < d_R < d_{TMC_2} < d_2 < d_{TMC_1} < d_3$) battery consumption is reduced considerably. Especially once the distance is greater than d_{TMC_2} (d_2 and d_3) where periodic checking is taking place, the battery consumption is much less than the baseline proximity alert (B-P).

B. Scenario B: Day planning

In this scenario, Bob sets up two proximity alerts before he leaves the home (08:00am) to arrange his day. He uses the first alert to remind him to meet with his colleague when he arrives at work. He uses the second one to remind him to buy some groceries at the grocery store. Bob might go to grocery store either in the morning or after work, so he specifies a large expiration time (12 hours) for the grocery store, but he turns off the alert for his work when he gets there. Bob returns home back at 8:00pm. Figure 4-c shows the route Bob takes during the day. Point A is his home, B is the grocery store, and C is his workplace. Distance between A and B is approximately 0.6 miles and the distance between B and C is around 2.2 miles. As shown in Figure 4-b, within this 12 hour period, baseline proximity alert spends most of the battery (30% remained), while our implementation spends only 17% of the battery (83% remained). Although, baseline proximity alert uses GPS and network location almost continuously (12 hour), for our implementation the GPS usage is 24 minutes, which is mostly traded with periodic network localization (22 minutes) and TMC (37 minutes). These findings corresponds to 75.71% improvements in battery consumption along with reducing GPS usage by 96.66%, network localization usage by 96.94% which are traded with periodic checking and TMC component which relies on relatively low-cost accelerometer sensor. We repeat the experiment several times and get the similar results.

VII. Related work

A. Location Sensing

Our work is the first work to investigate the energy-efficiency of high-precision proximity-alerts in Android, however, energy consumption of location sensing in smartphones has received interest in recent years. In [4], authors identify four factors that waste energy: static use of location sensing mechanism, absence of use of other sensors, lack of cooperation among applications, and finally ignoring battery level while sensing. In [5], authors argue that using a history of cell-id sequences, one can determine the user's location with accuracy comparable to GPS. In [6] authors utilize the location-time history of the user along with user's past velocity and activity ratio to duty-cycle GPS. In a related vein, in SensLock [7], authors explore the possibility of continuous location tracking in an energy efficient way. They utilize Wi-Fi AP beacons for localization, use accelerometer to duty cycle sensing, and GPS for path tracking. In [8], Ryoo et al. aim to monitor geofences defined as polygons. Their method of detecting entry and exit from geofences is to use GPS and cell tower signal strength along with user's activity state.

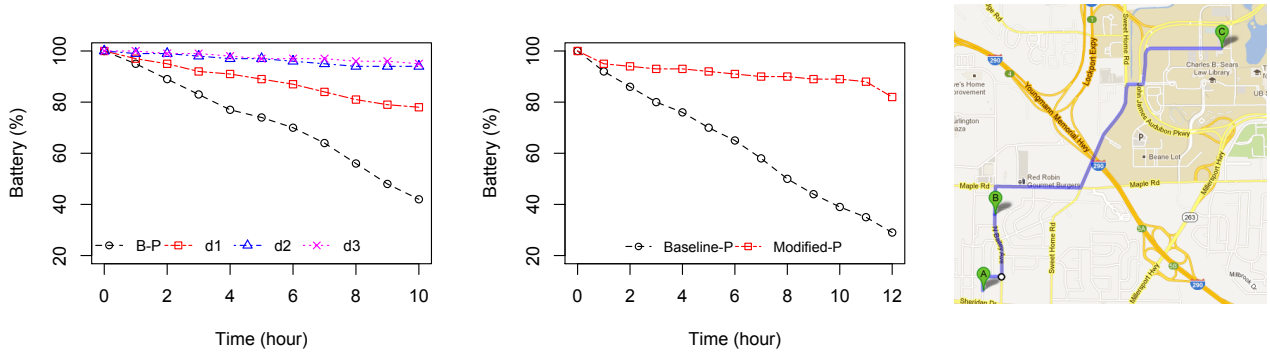

Figure 4. From left to right: a) Scenario A: Battery consumption of an idle phone with different distances. b) Scenario B: Battery consumption for a day planning scenario. c) Route for the Scenario B. Point A is Bob's home, B is the grocery store and C is Bob's workplace.

B. Activity Recognition

Due to the proliferation of sensors in commodity mobile devices, identifying the physical activity of a user has recently gained attention in pervasive community. Aside from using sensor motes to recognize user's activity [9], there has been an increasing interest on using smartphones to perform activity recognition. In [10] authors use accelerometer in smartphones to recognize different activities including walking, jogging and standing. In [11], authors use sensors to infer the user's status to share it on user's social network. This work adopts a split-level classifier to perform some part of the classification on the server. Finally, similar to our work, in [12], authors use smartphones to determine transportation mode of a user. Different than our approach, they utilize both accelerometer and GPS sensors. Due to the energy consumption of GPS, we opt-out of GPS and instead rely entirely on accelerometer.

VIII. Conclusion

We presented the design, implementation and the experiments of an energy efficient proximity alert for Android phones. Our method utilizes distances to the POIs and user's transportation mode to dynamically decide on location providers and location sensing interval. We tested our proposal in different scenarios and shows the strengths of our approach. Our evaluation results showed that, for a realistic scenario, we increase battery lifetime by 75.71% and reduces GPS usage by 96.66%. These results place applications which require high precision proximity alert service into the realm of possibility using the constrained battery capacities of today's smartphones. In our future work, we will improve the service by considering user's mobility profile and additional transportation modes.

References

[1] M. F. Bulut, Y. S. Yilmaz, M. Demirbas, H. Ferhatosmanoglu, and N. Ferhatosmanoglu, "Lineking: Crowdsourced line wait-time estimation using smartphones," in *MobiCASE: Fourth International Conference on Mobile Computing, Applications and Services*, 2012.

[2] S. Reddy, M. Mun, J. Burke, D. Estrin, M. Hansen, and M. Srivastava, "Using mobile phones to determine transportation modes," 2010.

[3] "Android open source project," http://source.android.com.

[4] 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*, ser. MobiSys '10. New York, NY, USA: ACM, 2010, pp. 315–330.

[5] J. Paek, K.-H. Kim, J. P. Singh, and R. Govindan, "Energy-efficient positioning for smartphones using cell-id sequence matching," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*, ser. MobiSys '11. New York, NY, USA: ACM, 2011, pp. 293–306.

[6] 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*, ser. MobiSys '10. New York, NY, USA: ACM, 2010, pp. 299–314.

[7] 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*, ser. SenSys '10. New York, NY, USA: ACM, 2010, pp. 43–56.

[8] J. Ryoo, H. Kim, and S. R. Das, "Geo-fencing: geographical-fencing based energy-aware proactive framework for mobile devices," in *Proceedings of the 2012 IEEE 20th International Workshop on Quality of Service*, ser. IWQoS '12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 26:1–26:9.

[9] P. Zappi, C. Lombriser, T. Stiefmeier, E. Farella, D. Roggen, L. Benini, and G. Tröster, "Activity recognition from on-body sensors: accuracy-power trade-off by dynamic sensor selection," in *Proceedings of the 5th European conference on Wireless sensor networks*, ser. EWSN'08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 17–33.

[10] J. R. Kwapisz, G. M. Weiss, and S. A. Moore, "Activity recognition using cell phone accelerometers," *SIGKDD Explor. Newsl.*, vol. 12, no. 2, pp. 74–82, Mar. 2011.

[11] E. Miluzzo, N. D. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S. B. Eisenman, X. Zheng, and A. T. Campbell, "Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application," in *Proceedings of the 6th ACM conference on Embedded network sensor systems*, ser. SenSys '08. New York, NY, USA: ACM, 2008, pp. 337–350.

[12] S. Reddy, M. Mun, J. Burke, D. Estrin, M. Hansen, and M. Srivastava, "Using mobile phones to determine transportation modes," *ACM Trans. Sen. Netw.*, vol. 6, no. 2, pp. 13:1–13:27, Mar. 2010.