J Inf Process Syst, Vol.14, No.4, pp.948~960, August 2018
https://doi.org/10.3745/JIPS.04.0082

ISSN 1976-913X (Print)
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

Significant Motion-Based Adaptive Sampling Module for Mobile Sensing Framework

Muhammad Fiqri Muthohar*, I Gde Dharma Nugraha*, and Deokjai Choi*

Abstract

Many mobile sensing frameworks have been developed to help researcher doing their mobile sensing research. However, energy consumption is still an issue in the mobile sensing research, and the existing frameworks do not provide enough solution for solving the issue. We have surveyed several mobile sensing frameworks and carefully chose one framework to improve. We have designed an adaptive sampling module for a mobile sensing framework to help solve the energy consumption issue. However, in this study, we limit our design to an adaptive sampling module for the location and motion sensors. In our adaptive sampling module, we utilize the significant motion sensor to help the adaptive sampling. We experimented with two sampling strategies that utilized the significant motion sensor to achieve low-power consumption during the continuous sampling. The first strategy is to utilize the sensor naively only while the second one is to add the duty cycle to the naive approach. We show that both strategies achieve low energy consumption, but the one that is combined with the duty cycle achieves better result.

Keywords

Adaptive Sampling, Android Mobile Sensing Framework, Significant Motion Sensor

1. Introduction

Mobile sensing (or smartphone sensing) is a study that analyzes and processes mobile phone sensor data into meaningful information. Many systems or applications in a different domain can have benefit from this. Elderly monitoring systems [1,2], patient monitoring systems [3,4], or human fall detection [5,6] are the examples of the mobile sensing applications in the health domain. There are some implementations of the mobile sensing approaches for other areas, such as in transportation [7], personal and social behavior [8-11], and environment monitoring [12].

Other than that, researchers also try to use mobile sensing as implicit authentication method by analyzing user gait cycle [13,14].

Based on user involvement, there are two types of data gathering methods in mobile sensing. Those two data gathering methods are participatory sensing [15] and opportunistic sensing [16]. In participatory sensing, the user needs to interact with the mobile sensing. A user will interact with the application and decision to start the data collection is solely done by the user. In opportunistic sensing,

Manuscript received November 13, 2015; first revision March 7, 2017; accepted March 16, 2017.
Corresponding Author: Deokjai Choi (dchoi@jnu.ac.kr)
* Dept. of Electronics and Computer Engineering, Chonnam National University, Gwangju, Korea ({fiqri.muthohar, gdebig}@gmail.com, dchoi@jnu.ac.kr)

a user does not need to interact with the mobile sensing application. The mobile sensing application has the capability to determine when they should sense the data. Periodic smartphone sensing is one of the methods that usually used in opportunistic sensing.

Although mobile sensing is helpful, the primary function of a smartphone is a communication. Continuous sampling and processing the data from smartphone sensor makes tremendous energy consumption. Users do not want the mobile sensing application to deplete their smartphone battery quickly.

Research on offloading sensing computation to the powerful remote computers has been conducted to reduce battery consumption [17,18]. However, we believe that sending sensor data to a remote computer can generate privacy issue. Therefore, in this study, we try to decrease battery consumption in the sampling phase locally.

There are many frameworks developed to help researchers build sensor data collection or mobile sensing application. Those frameworks are built to ease application development so researcher can gather sensor data without requiring in-depth knowledge of mobile application programming. Moreover, power management in those frameworks is provided through time scheduling configuration or static duty cycle.

In this study, we present our design and implement adaptive sampling module for a mobile sensing framework. We have surveyed several mobile sensing frameworks and chose one to improve with our adaptive sampling module. Our goal is to develop an adaptive sampling module that can be used to continuous sampling sensors' data without quickly depleting the smartphone battery. In this study, we focus our approach on the location and motion sensor of the smartphone. In the adaptive sampling module for the location and motion sensor, we try to utilize a sensor called "significant motion sensor." The sensor sends a signal whenever it detects a motion event that might lead to a change in the user's location. Our hypothesis tells that we can decrease the power consumption by utilizing the significant motion sensor.

2. Android Mobile Sensing Framework Overview

Human behavior and activity recognition are some of the hot topics in the mobile sensing research. Android is the most popular platform for mobile phone operating system [19]. Since the Android platform is used broadly and easy to distribute applications, the Android platform is good target to conduct the mobile sensing research.

There are several mobile sensing frameworks available to help researcher develop Android-based mobile sensing application. Those frameworks are MoST (Mobile Sensing Technology; https://github.com/participactunibo/MoST), Funf (http://funf.org), and SensingKit (http://sensingkit.org) to name a few. In this section, we will provide overviews for Funf, SensingKit, and MoST framework.

2.1 Funf

This framework was developed at the MIT Media Lab for their academic research. It provides a reusable set of functions which enables us to collect and configure a broad range of data types.

In Funf, *probes* are basic data collection objects where each probe contains a unit responsible for collecting a specific type of information. Table 1 provides a list of probes that are available on the Funf framework by default. The Funf modularity design allows 3rd-party developers to add new probes. To add new probes to the Funf, 3rd-party developers need to code the new modules to log the sensor data, save it to disk, and then send it back to the servers. Additionally, 3rd-party developers also need to handle issues such as the increasing of battery consumption and memory storage for each probe they develop.

Table 1. Funf probes list

GPS	Cell tower ID	Running applications
Location	Call log	Installed applications
WLAN	SMS log	Screen on/off state
Accelerometer	Browser history	Music/Image/Video file scan
Bluetooth	Contacts	Battery status

To control the probes, the Funf uses time schedule configuration. The researcher can edit the JSON configuration file that provided by the Funf framework to match researcher's application needs. The schedule object includes the following fields [20]:

- **Interval** – This field configures the period of the schedule.
- **Duration** – This field configures the amount of time to run.
- **Opportunistic** – This field configures passive listening capability.
- **Strict** – This field configures the option to run at an exact time interval.

2.2 SensingKit

The SensingKit is a mobile sensing framework developed for the multi-platform and large-scale experiments. This framework currently works on both Android and iOS mobile platforms. The SensingKit authors provide two separate frameworks: a client library that runs on the user's mobile devices and the server framework [21]. However, only client library's source code available, and it is still a prototype. The SensingKit is also open source under the LGPL license.

The SensingKit currently is capable of capturing several data types:

- Motion: linear acceleration, gravity, and rotation
- Location: GPS location
- Proximity: proximity using Bluetooth sensor
- Battery: device power consumption

The SensingKit authors claim that this framework is easily extensible due to its modular design. They only mention energy saving feature in the SensingKit framework using Bluetooth Smart proximity sensing feature [21].

2.3 MoST Framework

The MoST framework is an improvement from the previous mobile sensing framework called Mobile Sensing Framework (MSF). The MSF was initially developed to overcome some weaknesses and limitations of the Funf framework [22]. The authors revamped the MSF architecture to be simpler and more lightweight and then renamed it as MoST. The MoST framework is open source application under the Apache license.

The MoST framework consists of two major parts: sensing subsystem and management subsystem. Sensing subsystem handles the input and output from the sensors [23]. Meanwhile, the management subsystem handles the sensing process from system event, sensing request and power management [23]. The MoST sensing subsystem consists of two major parts: *input* and *pipeline*. *Input* is the term used by the framework to define any source of sensing data whether it is physical or logical [23]. Input examples are the battery, Bluetooth, accelerometer sensor, and microphone to name a few. The *pipeline* is the term used to define "any component that receives, processes, and fuses sensed data collected from one or more inputs and send the processing result to the client application that requested it" [23].

The MoST management subsystem consists of *input manager*, *input arbiter*, *interaction layer*, *MoST service*, and *power management*. *Input manager* manages the *input* in the sensing subsystem. The *input arbiter* works as a proxy of the input manager for:

- The *interaction layer* from system and framework events,
- user and external apps that communicate with the *MoST service*,
- input policies in the *power management*.

The input manager will start or stop the input based on the votes that come from the input arbiter. All votes from the interaction layer, the MoST service, and the power management must agree to start the input sampling. If one or more votes are not agreed, the input manager will stop the input sampling.

The *interaction layer* receives events from the framework and the android system. The *interaction layer* process events, like incoming call received, to handle sensor that might collide with user's action. In the case of an incoming call received, the interaction layer will vote to stop the microphone input sampling process. In the MoST framework, the *MoST service* handles active pipelines that request one or more input for it. In the case of a new pipeline request, the MoST will request to start the input that the pipeline needs through the input arbiter.

The MoST provide power management through the input policies in the *power management*. By default, the framework provides two input policies implementation: duty cycle and asymmetric duty cycle. Both input policies provide a static duty cycle mechanism with predetermined cycle period and ratio.

2.4 Selecting Mobile Sensing Framework

For this study, we choose to develop our module based on the MoST framework. The reason we choose this framework over the others are as follows:

- The MoST framework has low computation resource (CPU & RAM) utilization compared to the Funf framework [22].
- All three frameworks are designed to be a modular framework. However, the Funf and the SensingKit modularity design is to support additional sensor modules. Meanwhile, the MoST modularity design is broader than those two.
- The MoST framework code documentation is easy to read and quite complete.

3. Adaptive Sampling Module Design and Implementation

3.1 Adaptive Sampling Module Design

We design our adaptive sampling module as an input policy for the power management in the MoST framework. For this module, we make several design guidelines as follows:

- This module should not increase local resource usage (CPU & RAM) significantly.
- This module should support broad sensor types and capable of assigning appropriate sampling strategy for each sensor.
- This module should be easily expandable to allow additional sampling strategies for the new and existing sensor.
- This module should support continuous sampling while keeping the application's power consumption as low as possible.

As an input policy, this module can easily help to manage the power consumption during continuous sampling. This module will affect the vote taken in the input arbiter. Our module will select the appropriate sampling-strategy according to the sensor type and configuration value. In the implementation, we create the sampling-strategy as a Java interface for the sensor to be recognized by our adaptive sampling module.

3.2 Proposed Sampling Strategies

In this subsection, we will provide an overview for the significant motion sensor at first. After that, we will explain our significant motion-based sampling strategies for location and motion sensors: significant motion-based duty cycle sampling strategy and naive significant motion-based sampling strategy.

3.2.1 Significant motion sensor overview

A significant motion sensor is a low-power sensor that detects motion which might lead to a change in the user location [24]. This sensor is a software-based sensor instead of hardware-based one. It uses underlying hardware sensor in the device, such as accelerometer sensor or other low-power sensors, as it sources to detect the *significant motion* event [25]. Fig. 1 shows the significant motion events occurrence with tri-axis accelerometer sensor reading.

When the significant motion is detected, the significant motion sensor triggers an event through the operating system to notify those who have interests in this event. Walking, riding a bike, or sitting in a moving bus are the examples of these significant motions [25]. As long as the smartphone is still engaged in those activities, the sensor should trigger or produce the significant motion event periodically. Meanwhile when the device is in the pocket of a person that is not moving, or the device is on the table that shaking a little. Those activities are not categorized as significant motion [25].

This sensor is relatively low power compared to other sensors such as accelerometer or gyroscope as shown in Table 2. Another benefit of using this sensor is that there is no manual computation by the application developer needed compared to other sensors. Another sensor case, for example using accelerometer sensor, the application needs to wake the CPU to classify whether the user is moving or not using that sensor. This classification process in the CPU consumes power too.

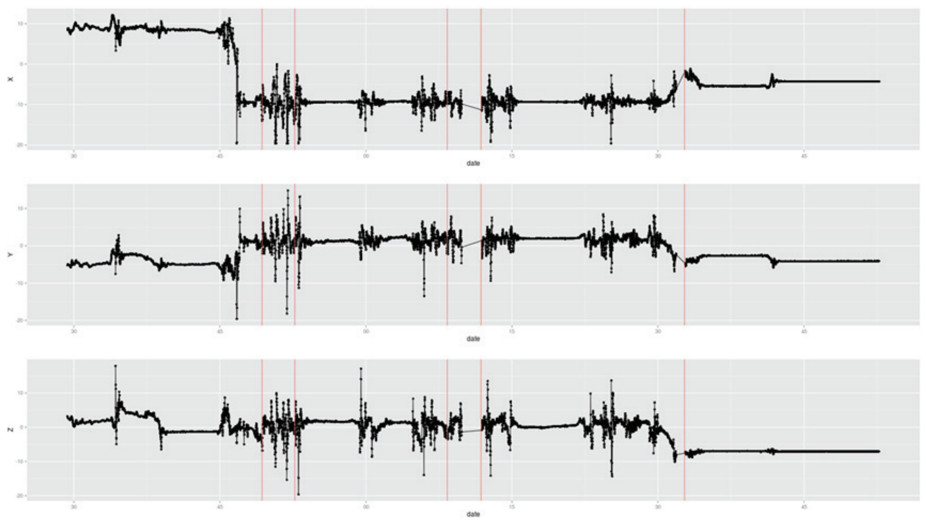

Fig. 1. Significant motion events occurrence (vertical red line) and tri-axis accelerometer signal.

Table 2. Sensor power requirement example on Samsung Galaxy S4 K330 (battery 3.8V)

Sensor name	Power requirement
Accelerometer sensor	0.25 mA (0.95 mW)
Magnetic field sensor	6.0 mA (22.8 mW)
Gyroscope sensor	6.1 mA (23.18 mW)
Significant motion sensor	0.3 mA (1.14 mW)
Gravity sensor	12.35 mA (46.93 mW)
Linear acceleration sensor	12.35 mA (46.93 mW)

The significant motion sensor makes a tradeoff between power consumption and the accuracy. The reason is this sensor may result *false positive*, e.g., an event triggered when the user is not moving. This *false positive* activates the CPU and consumes more power. Because of this reason, the significant motion sensor will have a delay to trigger the event. The delay is needed to avoid the *false positive* event. This delay is typically around 2 seconds with maximum up to 10 seconds [25].

Unfortunately, fragmentation is one of the issues in an Android-based platform. This fragmentation issue happens because phones with the old version operating system are still available with a huge percentage [26]. Another reason is that each device has different capabilities that depend on vendors [27]. The significant motion sensor is a relatively new sensor. It is only available in the recent Android-based smartphones. The Android operating system supports in the recent version with Android API level 19 that called Kit Kat (https://www.android.com/versions/kit-kat-4-4/). However, according to [27], the number of Android mobile devices that include significant motion sensor is increasing each year.

3.2.2 Significant motion-based duty cycle sampling strategy

In this strategy, we check whether the significant motion event is triggered or not. This idea is inspired by learning automata theory [28] which is called *linear reward-inaction learning* scheme. In this scheme, we need to choose an action. While doing that action, it checks whether it gets expected result or not. If it detects the expected result, it increases the probability to do the next action. Otherwise, it decreases the probability to do the next action.

In our adaptive duty cycle strategy, we modify the *linear reward-inaction learning* scheme to listen to the significant motion event. We listen to the significant motion event not only in the sampling period but also in the sleep period of the duty cycle. Therefore, our strategy to utilize the significant motion sensor is as follows: if there is a significant motion event detected in this duty cycle period then increase next sampling time. Otherwise, decrease next sampling time. Fig. 2 illustrates this process using a flowchart.

Our approach can be described using formula (1) and formula (2). The formula (1) described the process to increase the sampling duty cycle ratio, while formula (2) described the process to decrease the sampling duty cycle ratio.

T is the calculated sampling duty cycle ratio; $\tilde{T}$ is previous sampling duty cycle ratio, and α is a constant factor that adjusts the rate to increase or to decrease the sampling ratio.

$$T = \tilde{T} + \alpha(1 - \tilde{T}), \text{where } 0 < \alpha < 1 \tag{1}$$

and

$$T = \tilde{T} - \alpha\tilde{T}, \text{where } 0 < \alpha < 1 \tag{2}$$

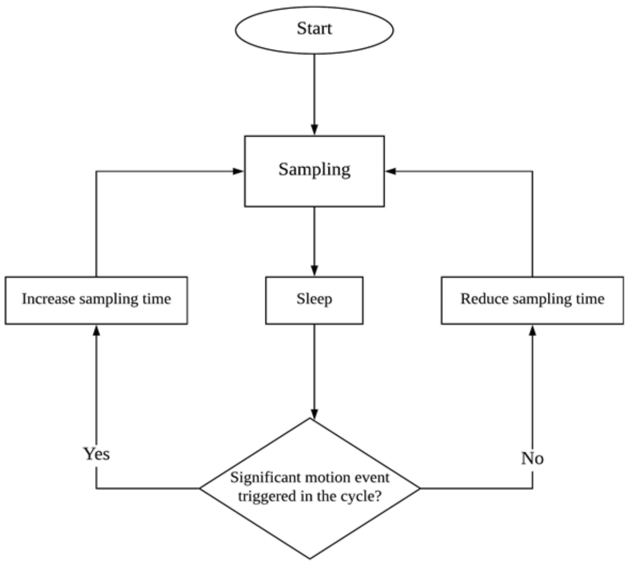

Fig. 2. Significant motion-based duty cycle sampling strategy flowchart.

3.2.3 Naive Significant Motion-based Sampling Strategy

This strategy uses the significant motion sensor naively to start motion and location sensor sampling. In this naive significant motion-based sampling-strategy, we directly start to sample the sensor data after a significant motion event triggered. The mobile application will listen to this significant motion event to begin the sampling. The sampling session will continue as long as the last significant motion trigger event is within the defined time threshold. If the last significant motion event exceeds that time threshold, sampling is stopped, and then the sensor will be put into sleep mode until another significant motion event occurs. Fig. 3 illustrates this sampling strategy using a flowchart.

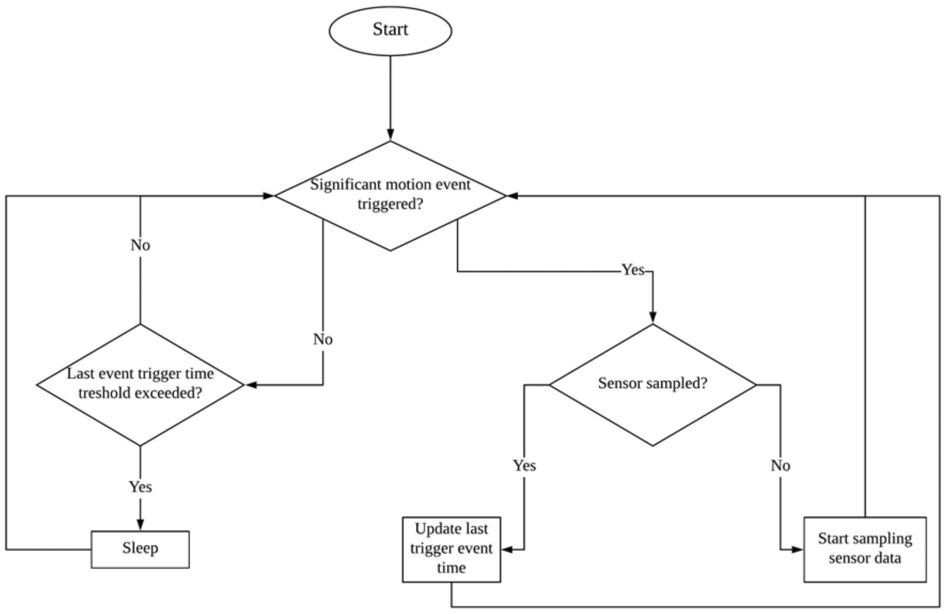
Start
Significant motion event triggered?
No
No
Yes
Last event trigger time treshold exceeded?
Yes
Sleep
Sensor sampled?
Yes
No
Update last trigger event time
Start sampling sensor data

Fig. 3. Naive significant motion-based sampling strategy flowchart.

3.3 Adaptive Sampling Module Implementation

3.3.1 Additional module implementation

For this study, we add a new module to manage the significant motion sensor. Thus, we call it "Significant motion manager module". We design the significant motion manager module to:

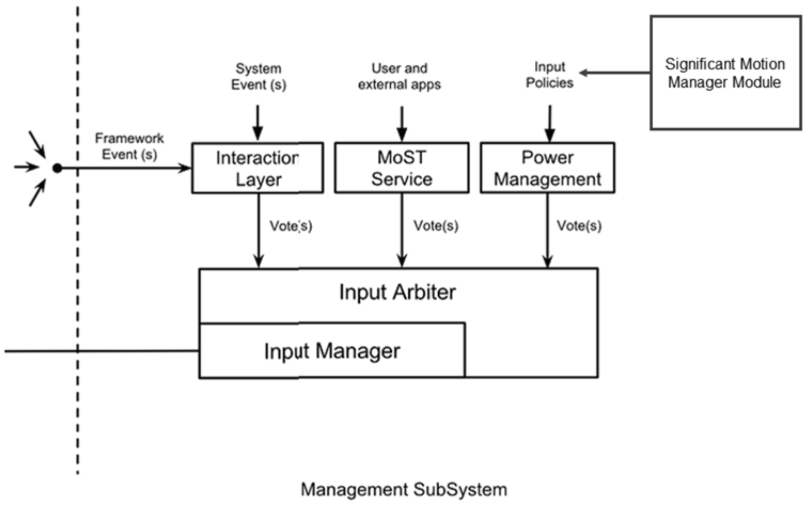

Fig. 4. The MoST Architecture management subsystem with our Significant Motion Manager module.

- Manage the significant motion sensor and handle the case of the sensor unavailability.
- Handle and process all significant motion events that are received in the framework when the sensor is available on the phone.
- Communicate with input policy that is interested in significant motion event.

This significant motion manager module will communicate with selected sampling strategy in our adaptive sampling module whenever it receives a significant motion event trigger. Fig. 4 displays our module design in the MoST architecture management subsystem.

3.3.2 Sampling strategies implementation

The α value, which the significant motion-based duty cycle sampling strategy uses in formula (1) and (2), is set to be 0.5 in both equations. The reason we choose this value is to balance the power consumption and the number of gathered data based on our trial. The duty cycle period for the accelerometer is 20 seconds, and the accelerometer will sample continuously during the sampling period. The duty cycle period for the location sensor is 60 seconds and the location sensor sampling is 30 seconds each during the sampling period.

We set the threshold value in the naive significant motion-based sampling strategy to be 20 seconds. This value is twice the time of maximum delay for the significant motion event to occur. We choose this value because it takes times for the application to receive the event signal from when the event signal sent from the sensor to the event listener. The application will sample the accelerometer sensor continuously within the sampling threshold time. The application samples the location sensor in 30 seconds each within the sampling threshold.

In both strategies, the accelerometer will be sampled at the fastest sampling frequency using the SENSOR_DELAY_FASTEST [24] configuration in Android API. By using this configuration, the application will sample the accelerometer sensor with a 0-microsecond delay based on this documentation [24]. However, because Android is not a real-time operating system, this value will give the lowest delay. For the location sensor, the application will sample it using the ACCESS_FINE_ LOCATION permission to be able to get location data from the network provider and GPS provider. The application needs that permission because sometimes GPS provider is unavailable and makes the application unable to retrieve location data.

To change which strategy used in the sampling, we provide an option to select one of those sampling strategies. We choose this approach to minimize the probability of running both applications at the same time. The option is implemented through "shared preferences" that is available in Android API.

4. Experiment and Result

4.1 Experiment Setup

We develop a mobile application using our modified MoST framework. Then, we install our mobile sensing application to two kinds of smartphone that have a significant motion sensor: Samsung Galaxy S4 (specification can be check at http://www.samsung.com/global/microsite/galaxys4/, we use the South Korea version of the phone) and LG G2 (specification can be check at http://www.lg.com/global/g2, we use the South Korea version of the phone). Both phones have their official latest Android operating system version, “Android Lollipop (https://www.android.com/versions/lollipop-5-0/)”, at the time we use it as the data collection device.

Our participants then do their usual activity with the data collection application run in the background. We set up the experiment this way to capture usage in real life condition where the user can do anything with their smartphone while doing their activities. The application gathers running (foreground) application data, accelerometer sensor data, location data, significant motion sensor event, and battery information data. The purpose of the data collection is to mimic the sampling process in the opportunistic mobile sensing. Each day, participants submit their data to us and have their system battery log data collected except for the weekend. After collecting the system log data, we reset the state of the system log data. Data collected for two weeks and each week participants collect data using different sampling strategy. We use those collected data to evaluate both sampling strategies.

4.2 Results and Discussion

We measured the battery consumption in each phone by using the Battery Historian (source code can be check at https://github.com/google/battery-historian) tools which is provided by Google to see how much an application consume power from Android phone’s battery. By using this tool, we can discern battery usage from one application to the other. The input for this tool is the system battery log that we collected from each participant every day.

Using the naive significant motion-based sampling strategies, user’s phone consumes around 7% to 10% of battery in one day. Fig. 5 shows the battery consumption results for the naive significant motion-based sampling strategy. While using the significant motion-based duty cycle sampling strategy, participant’s phone only consumes around 3% to 5% of battery in 1 day. Fig. 6 shows the battery consumption results for the significant motion-based duty cycle sampling strategy. In contrast, collecting location and motion sensor data without any power management policy consume about 11.5% to 14% of battery life in just 4 hours’ duration.

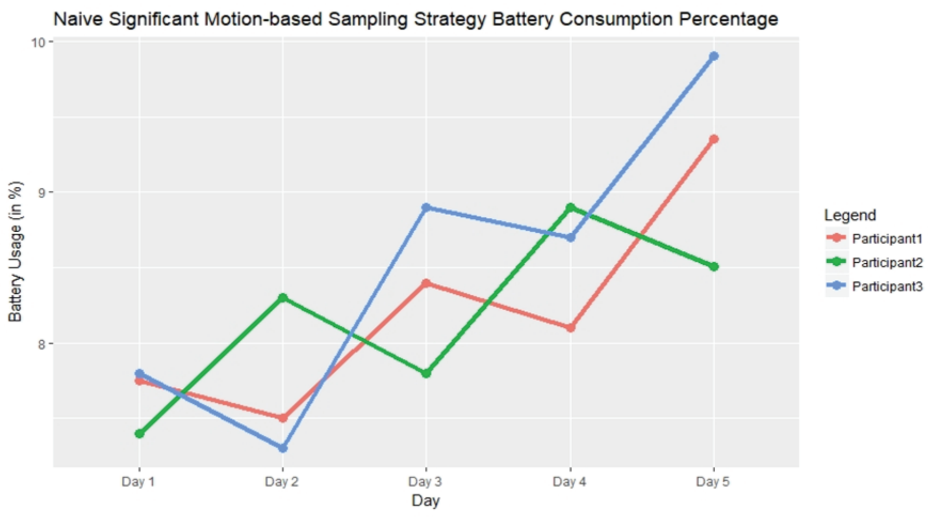

Fig. 5. Naive significant motion-based sampling strategy battery consumption.

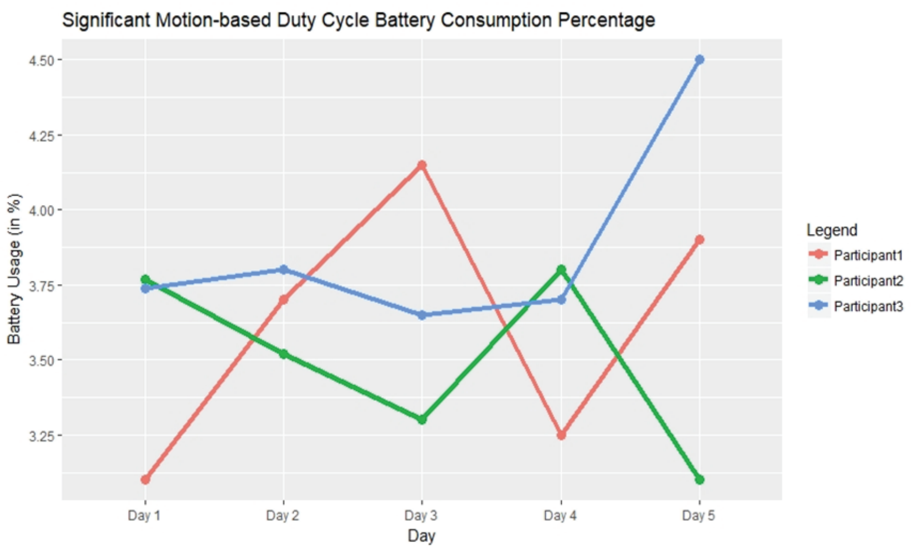

Fig. 6. Significant motion-based duty cycle sampling strategy battery consumption.

Those results differ because, in the significant motion-based duty cycle sampling strategy, sampling is done step by step before using all the duty cycles. While in naive significant motion-based sampling strategy, it samples the sensor until the threshold time is reached. Also, battery drain is varied depending on how active the person is. As our participants are all students, their activities were quite limited: go home (or dormitory), come to the lab, attend class, or take a break and go to eat. In this experiment, each student had similar activities in two weeks of data collection period.

5. Conclusions

In this study, we have presented our design and implementation of adaptive sampling module for the MoST framework. In our current work, we focus on the implementation of our adaptive sampling module in handling continuous location and motion sensors sampling. We have utilized significant motion sensor to help reduce energy consumption in continuous location and motion sampling.

We developed a data collecting application using the framework and our module to capture location, accelerometer sensor, and battery information. The application's battery consumption while uses naive significant motion-based sampling strategy that solely utilizes the significant motion sensor is lower than the case of without the sampling strategy. However, combining duty cycle with the significant motion sensor gives a better result in terms of power consumption rather than the naive significant motion-based sampling strategy. The significant motion-based duty cycle sampling strategy consumes battery 3.69% in average while the naive significant motion-based sampling strategy consumes battery 8.32% in average.

For our future work, we will test our implementation outside our lab with different kind of occupations. We also intend to add more sampling strategy support for other sensor types. Furthermore, we will use our application to do our activity recognition and human behavior research.

Acknowledgement

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (No. 2017R1D1A1B03035343).

References

[1] T. Faetti and R. Paradiso, “A novel wearable system for elderly monitoring,” *Advances in Science and Technology*, vol. 85, pp. 17-22, 2013.

[2] P. Pierleoni, L. Pernini, A. Belli, and L. Palma, “An android-based heart monitoring system for the elderly and for patients with heart disease,” *International Journal of Telemedicine and Applications*, vol. 2014, article no. 10, 2014.

[3] A. Raja, A. Tridane, A. Gaffar, T. Lindquist, and K. Pribadi, “Android and ODK based data collection framework to aid in epidemiological analysis,” *Online Journal of Public Health Informatics*, vol. 5, no. 3, article no. 228, 2014.

[4] S. Kumar, W. Nilsen, M. Pavel, and M. Srivastava, “Mobile health: revolutionizing healthcare through transdisciplinary research,” *Computer*, vol. 46, no. 1, pp. 28-35, 2013.

[5] L. Tong, Q. Song, Y. Ge, and M. Liu, “HMM-based human fall detection and prediction method using tri-axial accelerometer,” *IEEE Sensors Journal*, vol. 13, no. 5, pp. 1849-1856, 2013.

[6] O. Aziz, E. J. Park, G. Mori, and S. N. Robinovitch, “Distinguishing the causes of falls in humans using an array of wearable tri-axial accelerometers,” *Gait & Posture*, vol. 39, no. 1, pp. 506-512, 2014.

[7] P. Zhou, Y. Zheng, and M. Li, “How long to wait? Predicting bus arrival time with mobile phone based participatory sensing,” in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, Low Wood Bay, UK, 2012, pp. 379-392.

[8] R. LiKamWa, Y. Liu, N. D. Lane, and L. Zhong, “Can your smartphone infer your mood,” in *Proceedings of International Workshop on Sensing Applications on Mobile Phone (PhoneSense)*, Seattle, WA, 2011, pp. 1-5.

[9] A. Bogomolov, B. Lepri, and F. Pianesi, “Happiness recognition from mobile phone data,” in *Proceedings of 2013 International Conference on Social Computing (SocialCom),* Alexandria, VA, 2013, pp. 790-795.

[10] G. Chittaranjan, J. Blom, and D. Gatica-Perez, “Mining large-scale smartphone data for personality studies,” *Personal and Ubiquitous Computing*, vol. 17, no. 3, pp. 433-450, 2013.

[11] V. K. Singh, L. Freeman, B. Lepri, and A. S. Pentland, “Predicting spending behavior using socio-mobile features,” in *Proceedings of 2013 International Conference on Social Computing*, Alexandria, VA, 2013, pp. 174-179.

[12] N. Maisonneuve, M. Stevens, M. E. Niessen, and L. Steels, “NoiseTube: measuring and mapping noise pollution with mobile phones,” in *Information Technologies in Environmental Engineering*. Heidelberg: Springer, 2009, pp. 215-228.

[13] H. M. Thang, V. Q. Viet, N. D. Thuc, and D. Choi, “Gait identification using accelerometer on mobile phone,” in *Proceedings of 2012 International Conference on Control, Automation and Information Sciences (ICCAIS),* Ho Chi Minh, Vietnam, 2012, pp. 344-348.

[14] T. Hoang and D. Choi, “Secure and privacy enhanced gait authentication on smart phone,” *The Scientific World Journal*, vol. 2014, article ID. 438254, 2014.

[15] J. A. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and M. B. Srivastava, “Participatory sensing,” in *Proceedings of the 1st Workshop on World-Sensor-Web: Mobile Device Centric Sensory Networks and Applications (WSW 2006)*, Boulder CO, 2006.

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

[17] Q. Han, S. Liang, and H. Zhang, "Mobile cloud sensing, big data, and 5G networks make an intelligent and smart world," *IEEE Network*, vol. 29, no 2, pp. 40-45, 2015.
[18] K. K. Rachuri, C. Mascolo, M. Musolesi, and P. J. Rentfrow, "Sociablesense: exploring the trade-offs of adaptive sampling and computation offloading for social sensing," in *Proceedings of the 17th Annual International Conference on Mobile Computing and Networking*, Las Vegas, NV, 2011, pp. 73-84.
[19] Google Developers, "About Android," [Online]. Available: https://developer.android.com/about/.
[20] GitHub Inc., "funf-core-android," 2016 [Online]. Available: https://github.com/funf-org/funf-core-android/wiki/Configuration.
[21] K. Katevas, H. Haddadi, and L. Tokarchuk, "Poster: Sensingkit: a multi-platform mobile sensing framework for large-scale experiments," in *Proceedings of the 20th Annual International Conference on Mobile Computing and Networking*, Maui, HI, 2014, pp. 375-378.
[22] G. Cardone, A. Cirri, A. Corradi, L. Foschini, and D. Maio, "MSF: an efficient mobile phone sensing framework," *International Journal of Distributed Sensor Networks*, vol. 9, no. 3, article no. 538937, 2013.
[23] G. Cardone, A. Cirri, A. Corradi, L. Foschini, and R. Montanari, "Activity recognition for smart city scenarios: Google play services vs. MoST facilities," in *Proceedings of 2014 IEEE Symposium on Computers and Communication (ISCC),* Funchal, Portugal, 2014, pp. 1-6.
[24] Google Developers, "Sensors overview," [Online]. Available: https://developer.android.com/guide/topics/sensors/sensors_overview.
[25] Google Developers, "Sensor types," [Online]. Available: https://source.android.com/devices/sensors/sensor-types#significant_motion.
[26] Google Developers, "Android dashboard," [Online]. Available: https://developer.android.com/about/dashboards/.
[27] OpenSignal, "Android fragmentation (August 2015)," [Online]. Available: https://opensignal.com/reports/2015/08/android-fragmentation/.
[28] K. S. Narendra and M. A. Thathachar, *Learning Automata: An Introduction*. Mineola, NY: Dover Publications, 2012.

Muhammad Fiqri Muthohar https://orcid.org/0000-0003-3412-3271

He received his Bachelor degree from Institut Teknologi Bandung, Indonesia. He is currently a Master degree student in School of Electronics and Computer Engineering, Chonnam National University

I Gde Dharma Nugraha https://orcid.org/0000-0002-4960-7968

He received the B.Eng. in Electrical Engineering and Master degree in Computer Engineering from Universitas Indonesia. Since September 2014, he has been with the Network System Lab, Chonnam National University, Gwangju, Korea pursuing doctoral degree in Electrical & Computer Engineering.

Deokjai Choi https://orcid.org/0000-0001-9502-9882

He is a full professor of Computer Engineering Department at Chonnam National University, Korea. He received B.S. degree in Department of Computer Science, Seoul National University, in 1982. He got MS degree in Department of Computer Science, KAIST, Korea in 1984. He got Ph.D. degree in Department of Computer Science and Telecommunications, University of Missouri-Kansas City, USA in 1995. His interest on research spans from context awareness, pervasive computing, sensor network, future Internet and IPv6.