Flamingo: A framework for smartphone security context management

Md Shahrear Iqbal, Mohammad Zulkernine
School of Computing
Queen's University, Kingston, Ontario, Canada
{iqbal,mzulker}@cs.queensu.ca

ABSTRACT

The availability of powerful smartphones and the necessity of security in mobile devices have made researchers propose multiple security modes (e.g., home, office, outdoor, and financial) for such devices. In each mode, a user can install a different set of apps. However, in most of the cases, the user has to select the mode manually. If we can sense the smartphone's security context accurately, then it is possible to switch between different security modes automatically. Also, smartphone operating systems are becoming ubiquitous. As a result, mobile apps need to behave differently based on the security context (e.g., not sending the data if the network is insecure). There exist other research work that may detect the physical context of a smartphone. However, we focus on sensing different security parameters (e.g., location, is-network-encrypted) and calculating the security context from the parameters. In this paper, we propose Flamingo, a security context management framework that maintains a cache of security contexts and parameters to be used by the operating system and third-party applications. As detecting contexts requires the use of power-hungry smartphone sensors, a comprehensive framework for sharing security parameters among various applications can be beneficial in terms of energy and other resource expenses. The implementation of Flamingo as a part of the Android operating system shows that it is effective in managing security contexts and parameters.

CCS Concepts

•**Security and privacy** → **Mobile platform security;**

Keywords

Smartphone security; Multiple security modes; Security context;

SAC 2017,April 03-07, 2017, Marrakech, Morocco

http://dx.doi.org/10.1145/3019612.3019726

1. INTRODUCTION

Smartphones have made communications much easier and nowadays, they are the hub of people's work and entertainment. However, due to the age-old battle between security and usability, smartphones are often designed with the preference of usability. The reason is that rigorous security requires more involvement and knowledge from users. All these conditions lead us to a vulnerable mobile ecosystem where security and privacy are often compromised. The situation is even worse with the devices that are released with smartphone operating systems (e.g., smart TVs, smart cars, smart fridges, smart ovens, smart thermostats).

User contexts are important as smart devices are used for many different purposes. In the context of security, humans can sense danger from external entities. In a similar way, in our opinion, the smartphone operating system should also sense its security context and act appropriately. Millions of apps, available from both trusted and untrusted sources, offer services to users. However, users usually have no or little idea about what these apps do in the background. An operating system should not depend on its users to take proper security measures all the time which is tedious and often users do not have the knowledge to do so correctly. As a result, an automatic detection of security contexts and switching to an appropriate security mode are necessary.

In the ubiquitous computing research, the term "context" is associated with different meanings and interpretations. For example, a context can be user's location, his physical activity (walking, biking, running, etc.), or a list of nearby devices. Many researchers work on context-aware applications. In this paper, we are particularly interested in detecting a smartphone's "security context" so that the operating system can enforce appropriate security policies and facilitate security-aware applications. We define the smartphone's security context as follows:

> The security context of a smartphone is the degree of threat to the device's resources. It is a derived value from multiple parameters. The value of a parameter can be acquired from any smartphone sensors, or be derived from other parameters.

To manage the security context and a set of security re-

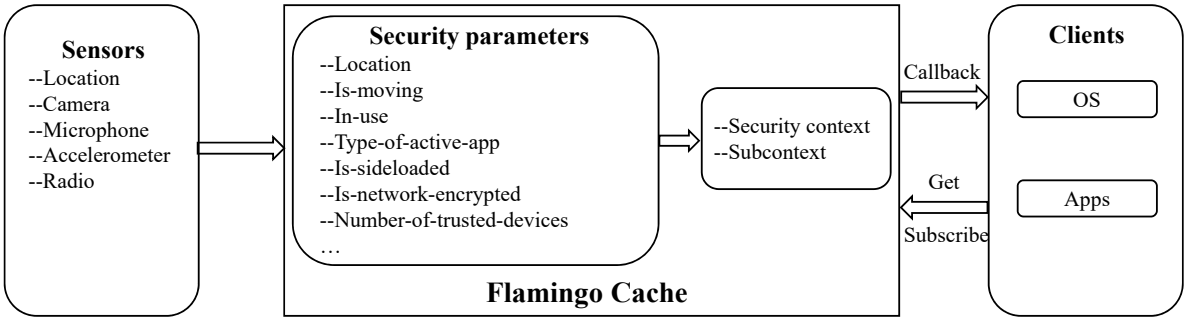

Figure 1: Flamingo maintains a cache of security parameters and context, calculated from smartphone sensors. Clients can get the values using the Flamingo API.

lated parameters, we design Flamingo[1]. Flamingo is capable of returning the phone's context or the value of a security parameter upon request. Flamingo uses different phone sensors to determine the context and manages a cache to reduce power consumption (by avoiding redundant recalculation of security parameters). The initial contexts and security parameters for Flamingo are identified based on an online survey involving actual users.

The remainder of the paper is organized as follows. We describe Flamingo in Section 2 and evaluate it in Section 3. A brief discussion on the related work is provided in Section 4, and we summarize the paper in Section 5.

2. THE FLAMINGO SYSTEM

The architecture of the Flamingo system is shown in Figure 1. Flamingo acts as a middleware between its clients (security-aware applications, operating system components, etc.) and phone sensors. Flamingo maintains a cache where it updates the context and security parameters on a regular time interval. A comprehensive framework for managing the security context and parameters will help immensely to reduce power consumption.

Flamingo uses different sensors to calculate a set of security parameters. The context and its subcontexts (a context within another context) are derived from these parameters. The context, as well as the values of these parameters, are shared between applications via a system API. The API exports the current context and also notifies subscribed clients any context changes via callbacks. For example, the operating system itself can be a client of Flamingo and can use the context value to switch to a different security mode. Third-party apps can use these values to behave differently in different situations.

In this section, we examine the sensors available in a typical smartphone and describe how these sensors are used to manage the security context and parameters. Some of the descriptions will be Android specific as we implemented Flamingo in Android.

[1]We choose this name due to the fact that Flamingos can sense natural disaster early [1].

2.1 Smartphone sensors

One of the reasons that made phones smart is the availability of sensors. Different types of sensors are now available including accelerometer, gyroscope, rotation sensor, ambient light sensor, proximity sensor, temperature sensor, etc. In addition to that, user's location can be detected using GPS, WiFi, Cell tower ID, and/or a combination of these. To calculate the security parameters, Flamingo uses the following sensors: location, camera, microphone, accelerometer, and radio.

2.2 A user survey

We performed an online survey to determine the security contexts and parameters necessary for an average user. The participants completed an online questionnaire (hosted in `fluidsurveys.com`) about their smartphone usage and their perception of security and privacy. 88% of the survey participants are between 19 and 39 years old and 97% of them have at least an undergraduate degree. 87.5% participants claim that they use their smartphones for work-related tasks. 23.8% participants give their phones to their children and they are not sure if it can cause any security or privacy issues. 56.7% participants often connect to open unsecured public WiFi networks. 62.1% participants do not feel secure using financial apps on mobile devices. 64% participants think that a security app is necessary to protect smartphone resources, however, 71% of them never installed one.

We also asked participants about different security contexts and parameters that they think should be provided by the underlying operating system. According to their answers, we find that the essential security contexts for an average smartphone user are home, office, and outdoor. Moreover, in the home context, the phone can be in two subcontexts, namely, casual and private. In the office context, the phone should detect whether the user is in a meeting. In addition, we identify another two subcontexts (side-loaded, financial) and a number of security parameters. The parameters are *Location, Place-type, Type-of-user-activity, Is-moving, In-use, Is-locked, Type-of-active-app, Is-side-loaded, Network-type, Is-network-encrypted, Is-camera-on, Is-mic-on, Is-storage-encrypted*, and *Number-of-trusted-devices*. In the following subsections, we describe how each of the parameter values is collected.

2.3 Security parameters

Flamingo maintains the value of *Location* using a number of power efficient techniques. In smartphones, GPS takes the highest power followed by WiFi and GSM. This is why Flamingo uses a wide range geofence (120 meters) to detect user's home or office. It also responds to the location-changed events generated by the Android location manager (set to update every five minutes or if the location changes more than 200 meters). If the location is neither home nor office, then Flamingo sets the location as outdoor. At the time of updating the *Location*, if the user is connected to the internet, Flamingo tries to detect the type (*Place-type*) of the location (restaurant, gym, park, cafe, hospital, etc.) using the Google places API.

Flamingo detects whether the smartphone is moving and updates *Is-moving*. It uses Google activity recognition API which detects a user's following activity: IN_VEHICLE, ON_BYCYCLE, ON_FOOT, RUNNING, STILL, TILTING, WALKING, and UNKNOWN. The detected activity is saved in *Type-of-user-activity*. If the activity is STILL, Is-moving is set to NO.

Flamingo updates *In-use* based on the on/off status of the smartphone screen. In addition to that, it also uses the status of the phone microphone and speaker. Flamingo detects whether the phone is currently locked or unlocked (*Is-locked*).

Flamingo categorizes (*Type-of-active-app*) every pre-installed application as well as any new apps that the user installs. This information is obtained from the Google Play.

Flamingo detects whether the currently active app is side-loaded (*Is-side-loaded*). Side-loaded means that the app is not installed using the default market (i.e., Google Play). Many popular paid apps are repackaged with malware and distributed freely in third-party app markets.

Flamingo maintains the type of network (*Network-type*) that is being used to connect to the internet. This can be 2G, 3G, LTE, or WiFi. An unencrypted network is considered insecure for a number of functionalities. As a result, Flamingo updates *Is-network-encrypted* on a regular interval and when any network change is detected.

Flamingo maintains whether the camera and the mic are currently in use (*Is-camera-on* and *Is-mic-on*). The operating system may alert the user if it detects any suspicious activity around these two sensors in certain contexts.

Is-storage-encrypted tells the encryption status of the currently mounted file system of the external storage.

Flamingo communicates with other devices in the same network and maintains a list of available trusted devices in each context (*Number-of-trusted-devices*). If the current location is at home, then Flamingo may find the user's spouse's phone, his smart TV, etc. and update the number. This parameter can be effective in some cases. For example, if Flamingo detects that a number of trusted devices are missing while the user is at home, then it can generate an alert.

2.4 Security context detection

Using the values of the security parameters, Flamingo calculates the security context. In Table 1, we list the contexts and subcontexts that Flamingo maintains. Note that the choice of these contexts and subcontexts are based on our user survey. However, Flamingo is completely flexible about the choice of the contexts. In each context, multiple subcontexts can be activated simultaneously. For example, when the current context is outdoor, the subcontext can be Restaurant (*Place-type*) and side-loaded. It means that the user is actually using a side-loaded app while located in an outdoor restaurant.

Security Context	Subcontext
Home	Casual, Private, Financial, Side-loaded
Office	Casual, In-meeting, Financial, Side-loaded
Outdoor	Place-type, Financial, Side-loaded

Table 1: Security contexts in Flamingo.

In Algorithm 1, we show how Flamingo switches to different contexts[2]. Flamingo always starts with sensing the current location. If the location is detected as home, it sets the subcontext as casual. However, if Flamingo detects that the user turns on the camera and/or the microphone, it sets the subcontext as private. This is due to the fact that media files (e.g., images, audio, video) captured inside a user's home are normally private. The operating system may decide to store the files securely so that other applications or malware cannot have an easy access to them.

In the office context, Flamingo switches the subcontext from casual to in-meeting according to the user's meeting schedule (as found in the calendar app) or when it detects that the user is inside a meeting room. The operating system can disable apps that can record audio or video in this subcontext.

Flamingo also updates the subcontext if *Is-side-loaded* is true or if *Type-of-active-app* is financial.

2.5 The Flamingo Cache

Currently, Flamingo implements a simple cache. At first, it calculates the values of all the security parameters and determine the current context and subcontexts based on the values. After that, Flamingo inserts these values in the cache with an expiry time. It updates the values of the security parameters when they expire in the cache. Operating system components or third-party applications interact with Flamingo by requesting the value of a security parameter or the current context. If the requested value is not expired in the cache, Flamingo returns the value right away saving an access to the sensors. Clients can also subscribe to Flamingo for a set of parameters. In that case, Flamingo informs the clients of any detected changes via callbacks.

[2]New contexts and security parameters can be introduced to Flamingo with a very little effort.

Algorithm 1 Security Context Detection

```
1: procedure FLAMINGO CONTEXT                ▷ Detect security context
2:     while True do
3:         if Location = HOME then
4:             Context ← Home
5:             subcontext ← Casual
6:             while True do
7:                 if Mic = ON | Cam = ON then
8:                     subcontext ← Private
9:                 else
10:                    subcontext ← Casual
11:                end if
12:                Call UPDATE
13:            end while
14:        else if Location = OFFICE then
15:            Context ← Office
16:            subcontext ← Casual
17:            while True do
18:                if In-Meeting = True then
19:                    subcontext ← In − Meeting
20:                else
21:                    subcontext ← Casual
22:                end if
23:                Call UPDATE
24:            end while
25:        else if Location = OUTDOOR then
26:            Context ← Outdoor
27:            while True do
28:                if Internet = ON then
29:                    subcontext ← Place − Type
30:                else
31:                    subcontext ← Other
32:                end if
33:                Call UPDATE
34:            end while
35:        end if
36:    end while
37: end procedure

38: procedure UPDATE                         ▷ Update Context
39:    if AppType = Sideloaded then
40:        subcontext ← Sideloaded
41:    end if
42:    if AppType = Financial then
43:        subcontext ← Financial
44:    end if
45:    if Location is changed then
46:        exit loop
47:    end if
48: end procedure
```

3. EVALUATION

In this section, we evaluate Flamingo in terms of accuracy, operational overhead, and usability. We implement a prototype of Flamingo by modifying the Android Open Source Project (Marshmallow version 6.0.0 r1 MRA58K as of 2015/10/17). We deploy the resulted operating system to a Google Nexus 5. It has Qualcomm MSM8974 Snapdragon 800 CPU (Quad-core 2.3 GHz) and Adreno 330 GPU with 2GB memory. It has the following sensors: accelerometer, gyroscope, magnetometer, light, proximity, pressure, and GPS.

3.1 Data Collection

Flamingo updates its internal cache with the values of the security parameters and context every 5 minutes. After it calculates a value, it inserts the value as a tuple in the cache with the expiry time (e.g., {<Location, Home>, 300}). During this update, Flamingo writes the values with a timestamp in a log file. We also develop an Android app that requests the values of different security parameters randomly using the Flamingo API (to access the cache periodically). If the current value of the requested parameter is expired, Flamingo recalculates the value, updates the cache, and returns the value to the app via a callback. Otherwise, the value is returned from the cache directly. We develop another app (to create the ground truth) that interacts with the user and asks the values of different parameters time to time. The user inputs are also logged with a timestamp.

3.2 Accuracy

We compare the two log files generated from both Flamingo and our user interaction app. Flamingo is 100% accurate in detecting the context (Home, Office, Location) described in Section 2. Flamingo also successfully detects the private subcontext whenever a user turns on the camera or the microphone while being inside his or her home. The type of the currently opened app is also logged properly as side-loaded, financial or other. However, Flamingo fails to detect whether the user is in a meeting if the meeting information is unavailable in the user's calendar. Accurately detecting a room inside a building is an active research area. However, by detecting user steps and with the help of an indoor floor plan, researchers [10] are able to pinpoint the indoor location with an acceptable accuracy. Unfortunately, this kind of technique consumes too much power to be considered for Flamingo. Alternatively, we plan to use the smart door locks that are available currently for enterprises. If a meeting or conference room in an office has smart door locks, the smartphone can communicate with it to confirm the meeting location of the user. Currently, Flamingo relies on the user calendar to update the "In-meeting" parameter.

3.3 Operational overhead

In this subsection, we evaluate Flamingo in terms of performance, memory and storage usage, and power consumption. In each case, we show that there is a very little to negligible overhead.

3.3.1 Performance

We quantify performance using a popular benchmarking app (AnTuTu) available from the Android stores. The app tests CPU and memory performance, 2D/3D graphics, Multitasking, etc. It gives a score for each test which can be used to compare relative performance between devices. All numbers from the benchmarking app are averaged over 10 runs. Table 2 shows the comparison of scores

resulted from the app. In our understanding, the difference between the scores are really insignificant and it is clear that the overall performance is not hampered by activating Flamingo.

Test Group	Score	
	Stock	Flamingo
3D	8680	8720
UX	15993	16026
CPU	16797	16598
RAM	6628	6525
Total	48098	47869

Table 2: Individual test scores from the AnTuTu benchmarking app.

3.3.2 *Memory and storage*

Flamingo does not incur any memory overhead. Our modification changes two system files, namely, framework.jar and services.jar. However, the changes in sizes (2.76K bytes for the framework.jar and 11.15K bytes for the services.jar) are insignificant. The storage requirement of Flamingo is also negligible as it maintains a cache of a few security parameters.

3.3.3 *Power consumption*

Although smartphones are becoming increasingly powerful day by day, they are still limited in battery capacity. As a result, we design Flamingo to share security parameters and thus avoid recalculation. Indeed, we find that Flamingo does not incur any overhead in terms of power. We show that in Figure 2. We observe that the battery charge level drops from 100% to 91% when Flamingo is enabled and from 100% to 92% when Flamingo is disabled during a period of 20 hours. During this time, we turned the WiFi off so that the location service only uses GPS. We moved the phone from office to home and then again home to office. We also requested different parameter values randomly every minute using multiple clients. However, the phone's screen remained off during the experiment. The total number of processes running on the phone during the experiment is 206. This experiment demonstrates that sharing the security parameters among different applications is a good idea if multiple clients (apps) in the system are requesting them.

3.4 Usability

In our opinion, a framework like Flamingo is necessary as an integral part of the operating system. The security model of the smartphone operating system cripples the power of the third-party anti-viruses. To fight against the rising threat of malware in smartphones and other smart devices (cars, TVs, fridges, etc.), the operating system needs to be a smart anti-malware itself [7]. Also, third-party applications should be aware of the security context of the phone and behave accordingly. For example, a financial app may not want to send sensitive personal information over an unencrypted network. Depending on the context, the operating system may decide to maintain separate data profiles for applications. In that way, data used

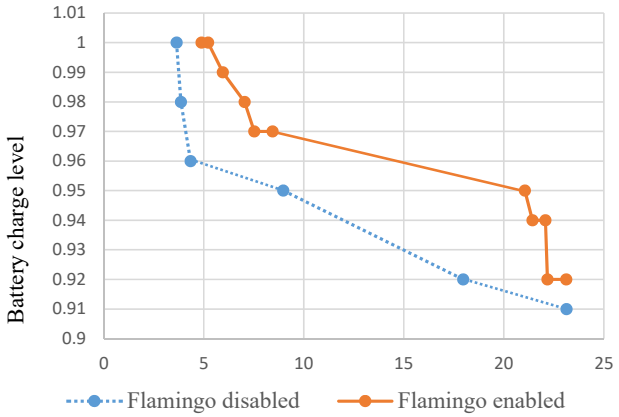

Figure 2: Battery charge level during a period of 20 hour.

in the office context can be protected from malicious apps in other contexts. Certain apps can also be made disabled in some secure contexts. Apps from unknown or untrusted sources can be blocked from accessing resources using a technique proposed in [8]. Also, Flamingo greatly reduces power consumption as it manages a cache of values that is shared among many applications. In our opinion, it is a much better approach than the case where each individual application is accessing the sensors to determine the values arbitrarily.

4. RELATED WORK

Smartphone sensors have been used for different types of application, e.g., detecting user activities [5], detecting location [11, 2], monitoring health status [15], finding people based on similar interests [3]. TransitGenie [4] uses WiFi, GPS, and accelerometer to distinguish between activities (e.g., walking, driving). Zheng et al. [18] use GPS history data to infer transport modes.

In [9, 12], the authors propose frameworks for context monitoring. They also investigate the energy consumption of their monitoring systems. Energy consumption is the main issue while designing a sensor-based system as smartphones are limited in battery capacity. Continuous use of sensors exhausts the battery quickly which the users may not like. The temporal correlation between different contexts to infer some values without accessing actual sensors is proposed by Nath [13] to enhance the energy profile. To consume less energy, adaptive sampling based on user's current context is used in systems like Jigsaw [12] and Sociable Sense [16].

A number of papers [6, 17, 14] address context-sensitive access control in Android devices. However, none of these address a comprehensive security context management framework and propose sharing of security parameters between applications.

5. CONCLUSION AND LIMITATIONS

We present Flamingo, a framework for managing smartphone's security context. In our opinion, it is helpful for

developing effective anti-malware capabilities in an operating system. It also facilitates security-aware third-party applications. Flamingo manages a cache of security parameters calculated from the smartphone sensors. The security context can be used by the operating system to enforce different restrictions. Flamingo also exports a system API that can be used to query the current context and/or the security parameters. The Flamingo cache enables sharing values among multiple clients which greatly reduces power consumption of the phone. We evaluated Flamingo and found that it accurately calculates different values of the security parameters. However, the Flamingo cache is not yet smart enough to infer the values of some parameters from other parameters which can further lessen the power requirement. In the future, we plan to implement an inference engine for the Flamingo cache.

6. ACKNOWLEDGMENTS

This work is partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Canada Research Chairs (CRC) program.

7. REFERENCES

[1] Can your pet sense disasters? https://blogs.cdc.gov/publichealthmatters/2012/10/pet-sense-disaster/. Accessed: 2016-10-06.

[2] M. Azizyan, I. Constandache, and R. Roy Choudhury. Surroundsense: mobile phone localization via ambience fingerprinting. In *Proceedings of the 15th annual International Conference on Mobile Computing and Networking*, pages 261–272. ACM, 2009.

[3] N. Banerjee, S. Agarwal, P. Bahl, R. Chandra, A. Wolman, and M. Corner. Virtual compass: relative positioning to sense mobile social interactions. In *Proceedings of the International Conference on Pervasive Computing*, pages 1–21. Springer, 2010.

[4] J. Biagioni, A. Agresta, T. Gerlich, and J. Eriksson. Transitgenie: a context-aware, real-time transit navigator. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 329–330. ACM, 2009.

[5] T. Y.-H. Chen, A. Sivaraman, S. Das, L. Ravindranath, and H. Balakrishnan. Designing a context-sensitive context detection service for mobile devices. *MIT Computer Science and Artificial Intelligence Laboratory, Tech. Rep.*, 2015.

[6] M. Conti, V. T. N. Nguyen, and B. Crispo. Crepe: Context-related policy enforcement for Android. In *Information Security*, pages 331–345. Springer, 2011.

[7] M. S. Iqbal and M. Zulkernine. SAM: A Secure Anti-Malware Framework for Smartphone Operating Systems. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC 2016)*, pages 1–6. IEEE, 2016.

[8] M. S. Iqbal and M. Zulkernine. ZoneDroid: Control your Droid through Application Zoning. In *Proceedings of the 11th International Conference on Malicious and Unwanted Software (MALCON)*, pages 113–120. IEEE, 2016.

[9] S. Kang, J. Lee, H. Jang, H. Lee, Y. Lee, S. Park, T. Park, and J. Song. Seemon: scalable and energy-efficient context monitoring framework for sensor-rich mobile environments. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, pages 267–280. ACM, 2008.

[10] F. Li, C. Zhao, G. Ding, J. Gong, C. Liu, and F. Zhao. A reliable and accurate indoor localization method using phone inertial sensors. In *Proceedings of the International Conference on Ubiquitous Computing*, pages 421–430. ACM, 2012.

[11] K. Lin, A. Kansal, D. Lymberopoulos, and F. Zhao. Energy-accuracy trade-off for continuous mobile device location. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, pages 285–298. ACM, 2010.

[12] H. Lu, J. Yang, Z. Liu, N. D. Lane, T. Choudhury, and A. T. Campbell. The jigsaw continuous sensing engine for mobile phone applications. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, pages 71–84. ACM, 2010.

[13] S. Nath. Ace: exploiting correlation for energy-efficient and continuous context sensing. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, pages 29–42. ACM, 2012.

[14] M. Nauman, S. Khan, and X. Zhang. Apex: extending Android permission model and enforcement with user-defined runtime constraints. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 328–332. ACM, 2010.

[15] T. Pascu, M. White, N. Beloff, Z. Patoli, and L. Barker. Ambient health monitoring: The smartphone as a body sensor network component. *InImpact: The Journal of Innovation Impact*, 6(1):62, 2016.

[16] 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*, pages 73–84. ACM, 2011.

[17] G. Russello, M. Conti, B. Crispo, and E. Fernandes. Moses: supporting operation modes on smartphones. In *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies*, pages 3–12. ACM, 2012.

[18] Y. Zheng, L. Liu, L. Wang, and X. Xie. Learning transportation mode from raw gps data for geographic applications on the web. In *Proceedings of the 17th International Conference on World Wide Web*, pages 247–256. ACM, 2008.