SWAN-Lake: Opportunistic Distributed Sensing for Android Smartphones

Nicolae Vladimir Bozdog
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
n.v.bozdog@vu.nl

Roshan Bharath Das
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
r.bharathdas@vu.nl

Aart van Halteren
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
a.t.van.halteren@vu.nl

Henri Bal
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
h.e.bal@vu.nl

ABSTRACT

Today, smart devices like mobile phones and fitness trackers have many sensors. Formatting, combining and querying sensor data from such devices can be a challenging and cumbersome task. In this paper we introduce SWAN-Lake, a distributed mobile framework that provides structured access to sensor data. SWAN-Lake provides access to local sensors as well as access to sensors of nearby devices. Here, we take a distributed approach by using device-to-device communication for exchanging sensor data between smartphones. We evaluate the performance and resource utilization using different sensing scenarios for our SWAN-Lake Android implementation. As a result, we show that our framework is energy efficient and performs well under high load.

CCS CONCEPTS

•Human-centered computing →Ubiquitous and mobile computing systems and tools; Smartphones;

KEYWORDS

Distributed sensing, ad-hoc data sharing, device-to-device communication, mobile computing, Internet of Things, Android

1 INTRODUCTION

The Internet of Things starts to become a reality. New mobile devices emerge, starting from fitness trackers to high-end smartphones. All these devices have in common the ability to sense the environment by means of a large variety of sensors. While fitness trackers are equipped with only few sensors that are required for tracking a particular activity, smartphones have plenty of sensors that allow them to perform complex activities. The simplest smartphone sensing applications use only one sensor to provide very specific functionality. For example, Shazam uses the microphone to identify a song being played and Pedometer uses the accelerometer to count the steps. While these applications are able to detect simple activities like walking or listening to music, they are not aware of the complete context of the user (e.g., walking in the park, while listening to music). To detect more complex scenarios, applications like Tasker [13], Trigger [14] and IFTTT [6] combine information from *many* sensors and from the internet to trigger user defined actions based on context. Still the ability of these applications to identify the context is limited by the given onboard sensors of the smartphone.

In this paper we investigate the possibility of sharing sensor data between nearby smartphones in order to enhance their sensing capabilities in the context of the Internet of Things. For example, a context-aware application could determine the waiting time at a bus stop, traffic light or supermarket by checking how long the phones of the people around have been staying still. To support this and other scenarios, we propose a framework for sharing sensor data between smartphones. Our framework, called SWAN-Lake, extends the work from [8] by adding support for building context-aware applications that not only use the sensors available on the smartphone, but also the sensors from nearby devices. SWAN-Lake exposes a simple API to developers and makes use of an intuitive language for defining context expressions. More specifically, our contributions are as follows:

- We introduce SWAN-Lake, which extends the SWAN framework [8] by adding support for sharing sensor data between close-by smartphones. In SWAN-Lake, we implement mechanisms for detection of nearby smartphones and sharing of sensor data with them over Bluetooth and WiFi links.
- We extend SWAN-song [11], the high-level language for defining context expressions in SWAN, in order to support sharing of sensor data between co-located devices.
- We evaluate SWAN-Lake under different sensing scenarios that use up to 7 remote sensors. Our measurements focus on the performance of the framework, energy efficiency and tolerance to failures.

The remaining of the paper is structured as follows: section 2 describes previous work that our research is based upon, the architecture of SWAN-Lake is explained in section 3 followed by

BODYNETS 2016, December 15-16, Turin, Italy

DOI 10.4108/eai.15-12-2016.2267797

evaluation results in section 4, section 5 presents a concise literature review, and section 6 concludes the paper.

2 BACKGROUND

In this section we describe briefly the SWAN framework [8], which we base our work upon. SWAN (Sensing With Android Nodes) is a framework for easily building context-aware applications for smartphones. SWAN acts as a middleware between applications and sensors. It focuses on helping application developers by providing a high-level abstraction for accessing the sensors. It also eliminates redundancy caused by multiple sensor-based applications running in parallel by providing a centralized solution for collecting and storing sensor data locally.

Multiple applications can interact with SWAN using its domain specific language called SWAN-song. The application registers a SWAN-song expression to SWAN, which evaluates the expression and sends back the result. As an example, the following expression gets the current light intensity:

```
self@light:lux{MAX,1000ms}
```

where `self` is the *location identifier* of the device, `light` is the sensor name, `lux` is the value path, `MAX` represents the history reduction mode and `1000ms` represents the history window. The expression computes the maximum among the values generated by the light sensor in the last 1000 milliseconds.

SWAN's architecture comprises all the components depicted in Fig. 1 except the Proximity Manager, which we introduce in SWAN-Lake. Applications can register SWAN-song expressions using the SWAN API. On registering an expression, the Expression Evaluation Engine invokes the Sensor Library in order to query the sensors used in expressions.

SWAN also supports sharing of sensed data with other devices. For this, it uses the Google Cloud Messaging (GCM) service to register context expressions remotely for evaluation. This method adds much overhead when the devices that are connecting are in proximity, in which case more efficient technologies like Bluetooth or WiFi Direct can be used. In the remaining of the paper we address this issue, by extending SWAN to support efficient collaborative sensing between co-located devices.

3 SWAN-LAKE FRAMEWORK

The main motivation behind SWAN-Lake is to create a tool that simplifies the development of context-aware applications by adding support for real-time sharing of sensor data with nearby smartphones.

3.1 Architecture

SWAN-Lake's architecture is shown in Fig. 1. There are two main changes compared to the original architecture of SWAN. First, we implement the Proximity Manager, which handles all the exchanges of sensor data with remote devices. Second, we enhance SWAN-song to allow registration of remote context expressions by using the *NEARBY* keyword as specifier for the location where the expression should be evaluated.

3.1.1 Remote expressions in SWAN-song. In SWAN-Lake, we extend the semantics of the *location identifier* from SWAN-song

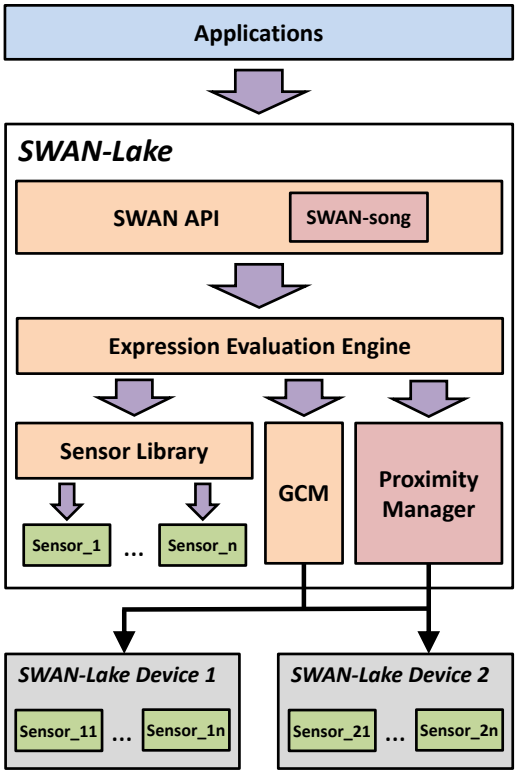

Figure 1: Architecture of SWAN-Lake

(see section 2) by allowing it to take as values a Bluetooth ID or the *NEARBY* keyword, like below:

```
bluetooth_ID@light:lux{MAX,1000ms}
NEARBY@light:lux{MAX,1000ms}
```

If a Bluetooth ID is used, then the Proximity Manager looks around for a device having that ID and, if found, it registers the expression to the remote device. Then, the remote device evaluates the expression and sends back the result.

If the *NEARBY* keyword is used, then the expression is sent to *all* nearby devices for evaluation (we will detail later how SWAN-Lake discovers nearby devices). Upon receiving the expression, all nearby devices register it in their instances of SWAN-Lake and, whenever the result of the expression changes, it is sent back to the device that issued the expression. This process continues until either the expression is unregistered by the device that issued it, or the latter moves out of range.

3.1.2 The Proximity Manager. The Proximity Manager (Fig. 2) handles the registration of remote context expressions, discovery of the nearby SWAN-Lake enabled devices and exchange of sensor data with them. The communication with the rest of the SWAN-Lake components is done through the Expression Evaluation Engine, to which it exposes an API with two methods:

```
registerExpression(exprId, exprBody, deviceId);
unRegisterExpression(exprId);
```

The first method is used to register a context expression written in the SWAN-song language with one device, if `deviceId` is a

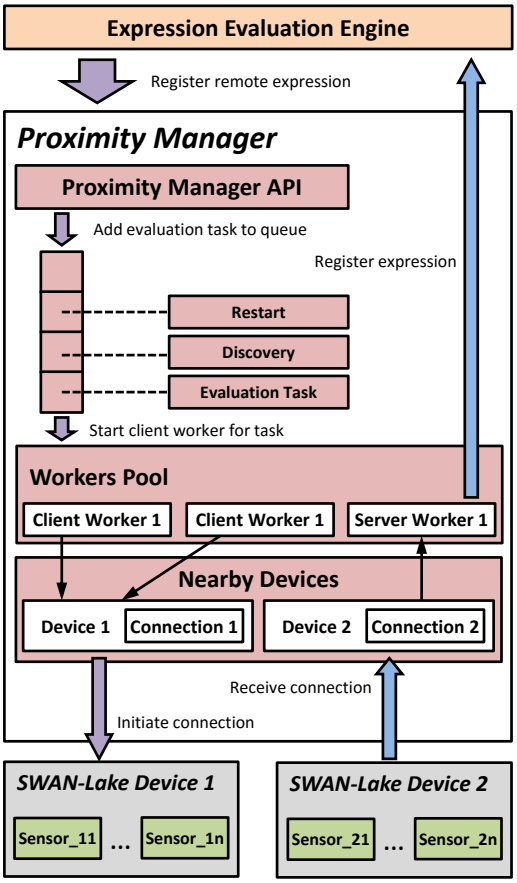

Figure 2: The Proximity Manager

Bluetooth device ID, or with all nearby devices, if `deviceId` is set to *NEARBY*. The other two parameters denote an unique ID for the expression (`exprId`) and the expression body (`exprBody`), which is the part after `@` in the example we gave earlier. If an expression is registered with a specific Bluetooth device, a new *evaluation task* is created for that device and introduced in the processing queue. Otherwise, if the expression is registered with all nearby devices, a new task is created for each device detected in proximity and added to the queue.

The purpose of the processing queue is to synchronize different types of tasks that cannot happen at the same time. Besides evaluation tasks, whose purpose is to register expressions to other devices, there are also *restart tasks* that restart the Bluetooth adapter in case of failures and *discovery tasks*, which trigger a Bluetooth discovery session of the close-by devices. Each time a new device is found, it is added in the *Nearby Devices* list. To differentiate between SWAN-Lake devices and other Bluetooth devices, our framework alters the Bluetooth ID of the smartphone by appending "swan" at the end of it followed by a unique ID. These three types of tasks cannot happen in parallel because evaluation tasks imply the opening of Bluetooth connections, which cannot happen while the adapter is restarting or discovering. Also, we do not want to restart the adapter while discovery is taking place. While restart tasks are scheduled only in the presence of failures, discovery tasks are scheduled periodically, with a frequency depending on how loaded the Proximity Manager is.

3.2 Operation Modes

SWAN-Lake supports two operation modes: *opportunistic* and *connected*. While the opportunistic mode is suitable for large loosely connected groups of devices (e.g., random people in a supermarket exchanging data about waiting times at checkout queues), the connected mode is better for small groups of devices that are strongly connected (e.g., people in a house playing a multiplayer game). Below we explain how they work.

3.2.1 Opportunistic mode. When working in opportunistic mode, a SWAN-Lake enabled smartphone should be able to share data efficiently with surrounding devices regardless of their number or their mobility. This is difficult to attain, given that Bluetooth supports at most 7 concurrent connections. To overcome this, we close connections with remote devices as soon as the sharing of sensor data ends to make room for other connections. Also, to reduce the overhead of opening and closing connections, we group requests for the same remote device on the same connection.

Another limitation of Bluetooth that we discovered in our tests is that if two Android smartphones try to connect to each other at the same time, the connection attempts might block indefinitely. Since this is very likely to happen when operating in *opportunistic mode*, we fixed this issue by alternating connections between each pair of devices in a group.

3.2.2 Connected mode. When running in connected mode, SWAN-Lake enabled smartphones are supposed to efficiently stream sensor data over long lived connections that are less affected by churn. To support this scenario, we implement a mechanism for caching connections in SWAN-Lake. Connection caching is done by keeping connections open even if no worker thread is using them, as opposed to the opportunistic mode, where unused connections are closed. This way, if a new context expression is registered, a cached connection can be used immediately without having to wait for a new connection to be established.

4 EVALUATION

We implemented SWAN-Lake as a library for Android. In our implementation we used Bluetooth for communication between nearby smartphones when operating in *opportunistic mode* and both Bluetooth and WiFi for the *connected mode*. In the experiments with WiFi we connected all phones to a local area network. We also considered WiFi Direct [4] and Bluetooth Low Energy (BLE) as alternatives, but were hindered by various limitations. In the case of WiFi Direct, the main problem is that it requires *manual* pairing between devices, while in the case of BLE only high-end smartphones have the capability of advertising their presence. Therefore, we currently use standard Bluetooth, which is supported by most smartphones and does not require manual pairing [1].

[1] Although standard Bluetooth is mostly used by manually pairing devices, the Bluetooth Android API also offers the option to connect smartphones without pairing them first.

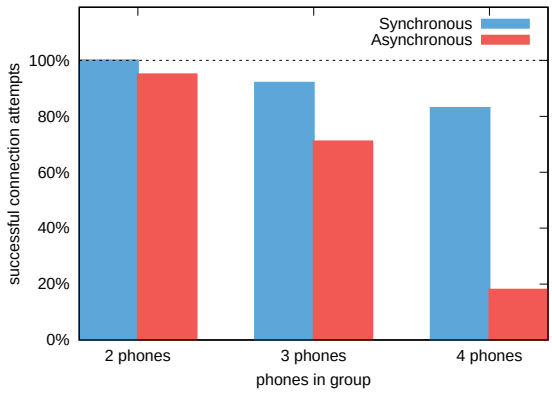

Figure 3: Success rate of connection attempts

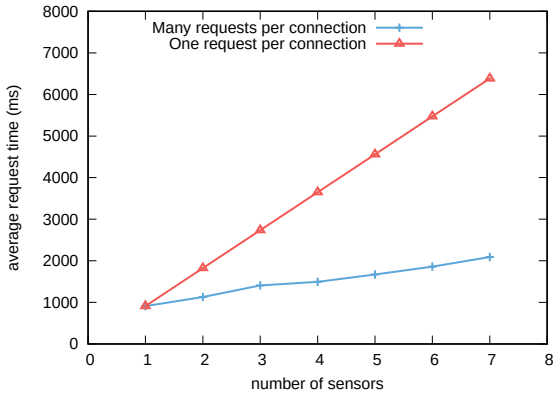

Figure 4: Average total time spent per request

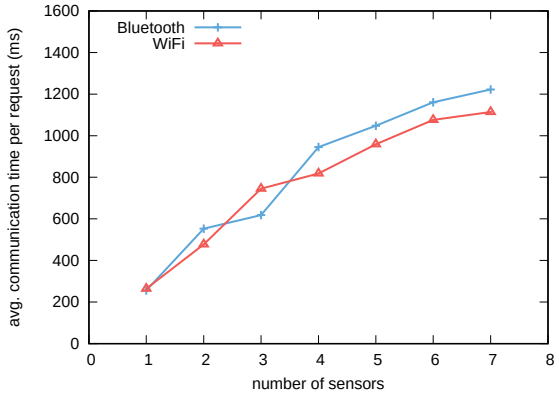

Figure 5: Average communication time spent per request

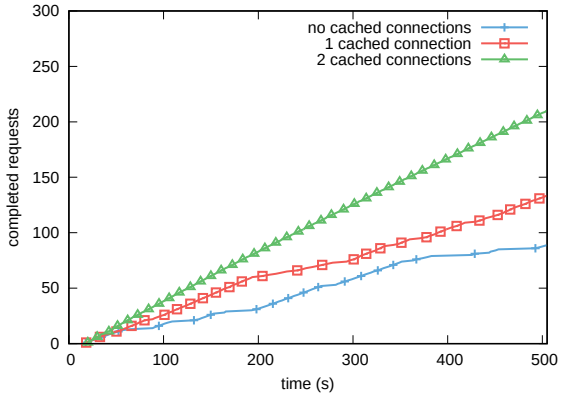

Figure 6: Number of completed requests for different numbers of cached connections

We used 4 phones in our measurements: Huawei Nexus 6P, LG Nexus 5 and two Motorola Moto G 4G. We ran experiments with 2, 3 or 4 phones. In every experiment, each phone requests data from N remote sensors on all the other phones in the group, where N ranges between 1 and 7. The requests are done by registering *NEARBY* expressions (see section 3.1.1) in a round-robin fashion to all phones in the group with a sample interval of one second between them. In all experiments we kept the screens of the phones on, except for those that measured the energy consumption. We used the following sensors in the tests: light sensor, accelerometer, gyroscope, magnetometer, battery level sensor, proximity and microphone. All these sensors are present in all the test phones.

4.1 Performance

4.1.1 Connection attempts. We start by looking at the number of successful connection attempts during 5 minutes experiments (Fig. 3) with groups of 2, 3 and 4 phones. In these experiments, each phone attempts to connect to the other phones in the group in a round-robin fashion (as needed by the *NEARBY* construct). If a connection fails, the initiator moves to the next phone in the group. We note here that SWAN-Lake handles connection failures gracefully, so they will not crash the system or slow down other tasks. The percentage of successful connection attempts is computed out of the total number of connection attempts to all phones in the group for the whole duration of the experiment. We explained in section 3.2 that alternating connections between Android smartphones can increase the chances of connection attempts to be successful. Therefore, in this experiment we compare the cases when connections between devices are alternating (synchronous) or not (asynchronous). As can be seen, the percentage of successful attempts drops steeply with the number of collaborating phones when connections do not alternate. When the optimization is used, the rate of failed attempts is reduced with up to 65%. Still, the failure rate is not reduced to 0 even with the optimized version, most likely due to interference.

4.1.2 Processing time. In SWAN-Lake we minimize the request processing time by grouping requests for many sensors on the same connection (see section 3.2). Hence, in Fig. 4 we compare the average request processing times achieved by the optimized version against those obtained if each request was made on a separate connection. We estimate the latter by multiplying the average processing time for one sensor with the number of sensors used. In

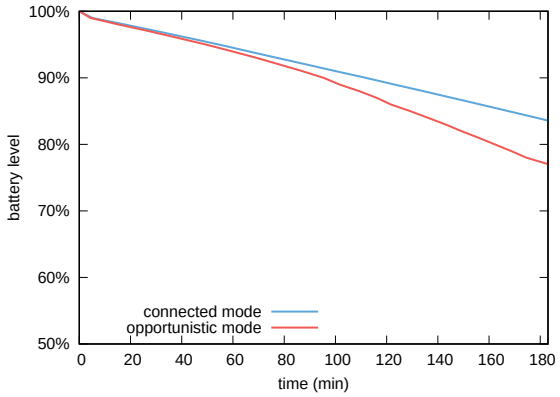

Figure 7: Battery usage for the two operation modes

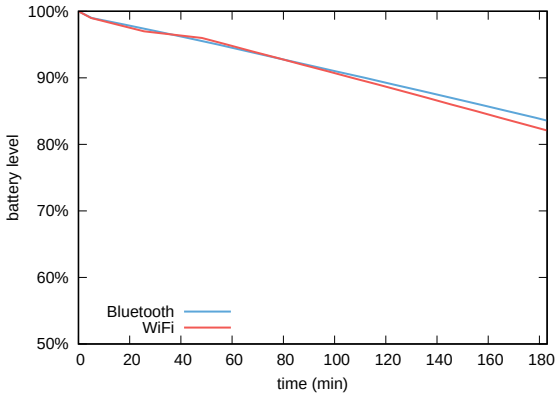

Figure 8: Battery usage for Bluetooth and WiFi in connected mode

this experiment we had a group of 3 phones collaborating in opportunistic mode. As we can see, the optimized version is performing much better regardless of the number of sensors used.

4.1.3 Communication time. In the next experiment we study how Bluetooth and WiFi work for a group of 3 phones exchanging sensor data in *connected mode* (Fig. 5). The results show that with both techniques our framework performs reasonably well, with a small advantage for WiFi when more than 3 sensors are queried. Each additional sensor adds approximately 160 ms for Bluetooth and 140 ms for WiFi, which are close to the communication overhead when operating in opportunistic mode. Therefore, we can conclude that the latency of the communication protocol influences the performance of the framework.

4.1.4 Cached connections. SWAN-Lake uses cached connections in order to reduce communication time (see section 3.2). In the next experiment we keep the same setup as in the previous one and analyze how the number of successful requests in a fixed time interval is affected by the number of cached connections (Fig. 6). As we can see, for every point in time, the number of requests completed by that moment increases with the number of cached connections.

4.2 Resource Usage

In this section we examine several scenarios that highlight the resource utilization of SWAN-Lake. All the experiments presented in this section were done under stress conditions, therefore the results presented here represent an upper limit of the SWAN-Lake's resource utilization.

We start by looking at the energy consumption of our framework when operating in opportunistic and connected modes (Fig. 7). In this experiment we let 2 phones exchange sensor data over Bluetooth continuously for 3 hours. All the 7 available sensors were used. As the figure shows, there is a clear difference in favor of running the framework in connected mode. This is expected, as in opportunistic mode the two phones keep connecting and disconnecting from each other for each request, which causes the Bluetooth adapters of the phones to consume more energy.

Next, let us see which of Bluetooth and WiFi performs better when operating in connected mode. As Fig. 8 shows, at first sight they are both almost equally good, with Bluetooth saving 2% of the battery. However, for the 3 hours of the experiment there were only 2686 requests completed over WiFi compared to 2844 requests completed over Bluetooth. We determined that this behavior was due to the fact that the WiFi adapter goes into a low-power state if the screen is inactive, as opposed to the Bluetooth adapter, which operates at full potential all the time.

Finally, we looked at the CPU load and memory usage. In our tests the CPU load did not exceed 11% and the memory usage did not go beyond 20 MB regardless of the number of queried sensors. These values are normal given the high number of sensors used and the high frequency of the readings. The Mosden sensing framework [12], which is the closest work related to ours, uses up to 20% of the CPU and 23 MB of memory in similar usage conditions.

5 RELATED WORK

Mosden [7, 12] is a platform that enables collaborative processing of sensor data. It extends GSN [1], a middleware for processing sensor data in the cloud, by adapting it to run on resource constrained mobile devices. Mosden and SWAN-Lake are similar, as they both follow a distributed architecture in which mobile devices can perform sensing tasks collaboratively. Unlike SWAN-Lake, Mosden does not allow for sharing of sensor data with anonymous nearby devices.

Sharing of mobile data over D2D links has already been proven useful for context computation offloading [3] and group activity recognition [2]. Also, emerging protocols like Zigbee [9] or Z-Wave [5] offer new opportunities for applications relying on ad-hoc sharing of data.

When it comes to commercial solutions, most of them are focused on storing and analyzing data collected by the Internet of Things [10]. To our knowledge, the only commercial tool for collaboratively sensing in proximity is Google Nearby, which uses a combination of short range communication techniques and cloud services to connect Android smartphones with nearby services.

6 CONCLUSIONS

In this paper we introduced and evaluated SWAN-Lake, a framework for building collaborative mobile sensing applications. Our framework leverages sharing of sensor data within groups of co-located devices in order to improve the accuracy of context detection in smartphone applications. We tested SWAN-Lake under various sensing scenarios on a small group of devices. Our measurements showed that our framework performs well under stress conditions and has low resource consumption.

As future work, we intend to make SWAN-Lake more customizable in terms of privacy, by extending SWAN-song to support various privacy directives. These directives should give users control about which other phones to trust or not (based on contact lists, location, etc.). Also, different privacy settings could be used for sensitive sensors (e.g., the user's heartbeat level) and less sensitive ones (e.g., ambient noise). To prevent the interception of sensitive data by third-parties, we consider encrypting sensor data that is being shared. Finally, we plan to study the performance of our framework for groups of phones with high mobility. To this end, we aim to incorporate SWAN-Lake in a crowdsensing application and test it during an event with a large number of participants.

ACKNOWLEDGEMENTS

This publication was supported by the Dutch national program COMMIT/. We thank Marc Makkes for his comments that greatly improved the manuscript. We are also grateful to the anonymous reviewers for their useful insights.

REFERENCES

[1] Karl Aberer, Manfred Hauswirth, and Ali Salehi. 2007. Infrastructure for data processing in large-scale interconnected sensor networks. In *2007 Int. Conf. on Mobile Data Management*. IEEE, 198–205.

[2] Amin B Abkenar, Seng W Loke, Wenny Rahayu, and Arkady Zaslavsky. 2016. Energy Considerations for Continuous Group Activity Recognition Using Mobile Devices: The Case of GroupSense. In *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*. IEEE, 479–486.

[3] Khaled Alanezi, Xinyang Zhou, Lijun Chen, and Shivakant Mishra. 2015. Panorama: A Framework to Support Collaborative Context Monitoring on Co-located Mobile Devices. In *Int. Conf. on Mobile Computing, Applications, and Services*. Springer, 143–160.

[4] Daniel Camps-Mur, Andres Garcia-Saavedra, and Pablo Serrano. 2013. Device-to-device communications with Wi-Fi Direct: overview and experimentation. *IEEE wireless communications* 20, 3 (2013), 96–104.

[5] Carles Gomez and Josep Paradells. 2010. Wireless home automation networks: A survey of architectures and technologies. *IEEE Communications Magazine* 48, 6 (2010), 92–101.

[6] IFTTT. 2016. https://ifttt.com. (2016). Accessed: 2016-11-16.

[7] Prem Prakash Jayaraman, Charith Perera, Dimitrios Georgakopoulos, and Arkady Zaslavsky. 2013. Efficient opportunistic sensing using mobile collaborative platform mosden. In *Collaborative Computing: Networking, Applications and Worksharing, 2013 9th Int. Conf. on*. IEEE, 77–86.

[8] Roelof Kemp. 2014. *Programming Frameworks for Distributed Smartphone Computing*. Ph.D. Dissertation. Vrije Universiteit, Amsterdam.

[9] Patrick Kinney and others. 2003. Zigbee technology: Wireless control that simply works. In *Communications design conference*, Vol. 2. 1–7.

[10] Julien Mineraud, Oleksiy Mazhelis, Xiang Su, and Sasu Tarkoma. 2015. Contemporary Internet of Things platforms. *arXiv preprint arXiv:1501.07438* (2015).

[11] Nicholas Palmer, Roelof Kemp, Thilo Kielmann, and Henri Bal. 2012. SWAN-song: a flexible context expression language for smartphones. In *Proceedings of the Third International Workshop on Sensing Applications on Mobile Phones*. ACM, 12.

[12] Charith Perera, Prem Prakash Jayaraman, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. 2014. Mosden: An internet of things middleware for resource constrained mobile devices. In *2014 47th Hawaii International Conference on System Sciences*. IEEE, 1053–1062.

[13] Tasker. 2016. http://tasker.dinglisch.net. (2016). Accessed: 2016-11-16.

[14] Trigger. 2016. http://gettrigger.com. (2016). Accessed: 2016-11-16.