SenseLE: Exploiting Spatial Locality in Decentralized Sensing Environments

Nicolae Vladimir Bozdog, Marc X. Makkes, Alexandru Uta, Roshan Bharath Das, Aart van Halteren, Henri Bal
Department of Computer Science
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
{n.v.bozdog, m.x.makkes, a.uta, r.bharathdas, a.t.van.halteren, h.e.bal}@vu.nl

***Abstract*—Generally, smart devices, such as smartphones, smartwatches, or fitness trackers, communicate with each other indirectly, via cloud data centers. Sharing sensor data with a cloud data center as intermediary invokes transmission methods with high battery costs, such as 4G LTE or WiFi. By sharing sensor information locally and without intermediaries, we can use other transmission methods with low energy cost, such as Bluetooth or BLE.**

In this paper, we introduce Sense Low Energy (SenseLE), a decentralized sensing framework which exploits the spatial locality of nearby sensors to save energy in *Internet-of-Things* (IoT) environments. We demonstrate the usability of SenseLE by building a real-life application for estimating waiting times at queues. Furthermore, we evaluate the performance and resource utilization of our SenseLE Android implementation for different sensing scenarios. Our empirical evaluation shows that by exploiting spatial locality, SenseLE is able to reduce application response times (latency) by up to 74% and energy consumption by up to 56%.

I. Introduction

Distributed sensing and monitoring are rapidly evolving, as both the underlying technology and the number of connected devices are growing at a fast pace. Current state-of-the-art solutions [1], [2] for performing distributed sensing are centered around cloud computing. Such approaches offer efficient methods of analyzing and storing sensor-generated data. The most widely deployed sensing infrastructures perform traffic monitoring, air-quality measurements, disaster management etc.

We identify two types of distributed sensing applications: off-line data analysis (e.g., road quality monitoring), and real-time data analysis (e.g., traffic monitoring, athletes coaching, disaster management). While the cloud-centric approach is highly advantageous for off-line data analysis, it is less suitable for applications that require real-time decision making. For such applications, a decentralized architecture would significantly improve response times by performing direct, device-to-device communication. To achieve this, smartphones are equipped with significant numbers of sensor and networking technologies, such as WiFi, Bluetooth and NFC, and make ideal hubs for other IoT devices. Currently, many IoT architectures use the smartphone as a gateway for sending sensor data to be processed in the cloud [3], [4]. Modern smartphones have multi-core CPUs and large storage space, enabling them to perform significant amounts of local computation on sensor data. Such computing and storage capabilities have not yet been harnessed for *distributed sensing*.

By combining highly versatile communication technologies, and increasing compute and storage capacities, smartphones are becoming a highly promising computing platform. By means of *decentralization* and *collaboration*, smartphone-based IoT sensing platforms can improve upon the cloud-centric model, by analyzing data close to its source, and disseminating information through low-energy channels to nearby devices. In this way, spatial locality can be harnessed, leading to a highly decentralized and efficient architecture in which smartphones can collaboratively sense and share information without a cloud infrastructure that relays messages.

In this paper we investigate how smartphones can be used effectively and efficiently to locally perform sensor data processing and sharing, as opposed to the traditional cloud-centric approach. Instead of sending all the collected data to a cloud, in our vision, smartphones act as a middleware that collects data in ad-hoc fashion from smart devices, and then processes it locally, according to the needs of the applications. This approach is beneficial for real-time applications that communicate with sensing devices over short ranges (less than 100 m). Such applications operate optimally if the density of nearby sensing devices is high and the bandwidth requirements are low, which is the case in sensor applications like crowd analysis [5], indoor applications for smart offices [6], or sports monitoring [7]. By using only local information, such applications can achieve better response-times than their cloud-centric counterparts. For example, a context-aware application can determine waiting times at queues by checking how long the phones of the nearby people have been staying around. We implement this application and discuss it later in the paper. Another example is a navigation application for groups of tourists that, instead of using the onboard GPS sensor, obtains the GPS coordinates from another device in the group. This approach could greatly reduce the energy used by the group of phones, as shown later in the paper.

In this work, we improve upon the results of our earlier SWAN-Lake system [8] by reducing the response times and power consumption, which are both requirements of IoT. We achieve this through a complete system re-design, resulting in a new framework for opportunistic acquisition and processing of sensor data, called SenseLE (Sense **L**ow **E**nergy).

A key challenge in designing SenseLE is to enable real-time sensor data collection and processing with minimal energy overhead. To achieve this, we use Bluetooth Low Energy (BLE) as underlying technology for sharing sensor data between mobile devices. With low energy consumption, low latency and similar range as standard Bluetooth, BLE is the best candidate for our scenario. The main drawback of BLE is its small bandwidth, but most sensors produce only numeric data that can be transferred over low bandwidth connections. However, BLE was created mostly for connecting to peripherals, such as fitness trackers or beacons, which have a fixed and relatively small number of sensors. To adapt it for sharing multi-sensor data between devices, we had to create a mechanism that dynamically chooses which sensors to share based on the context. This mechanism automatically detects what sensors are needed by close-by devices and makes these sensors remotely accessible over BLE. To our knowledge, this is the first attempt to use BLE for sharing sensor data between mobile devices.

In summary, our contributions are as follows:

- We develop SenseLE as a solution for decentralized sensing in mobile environments. SenseLE employs mechanisms for detecting nearby devices and accessing their sensors remotely over BLE.
- We implement SenseLE as library for Android and evaluate it under multiple sensing scenarios involving small groups of smartphones. Our evaluation shows that our framework achieves good performance with low energy footprint.
- We prove the usability of SenseLE by building a simple application for estimating waiting times at queues.

The remainder of the paper is structured as follows: in Section II we discuss previous work that we base upon, Section III describes the architectural elements of our framework, the results of our evaluations are presented in Section IV, Section V discusses related work, and Section VI concludes the paper.

II. Background

Previously, we studied the feasibility of using standard Bluetooth for opportunistic distributed sensing [8]. In this paper we build upon that previous work and implement a much more efficient mechanism for sharing sensor data based on Bluetooth Low Energy (BLE). Despite sharing the name "Bluetooth", the two communication technologies differ in many respects, from the way connections are managed to the API exposed to the applications. Therefore, in SenseLE we had to take a completely new approach in order to match the functionality offered by BLE to the requirements of distributed sensing. We have also tested WiFi Direct [9] as a solution for remote sensing, but were hindered by the fact that it requires *manual* pairing between devices, which makes it unusable in opportunistic sensing scenarios. Moreover, it was previously shown that WiFi Direct consumes significantly more energy than Bluetooth [10].

SenseLE uses the SWAN library [11] for accessing the sensors of the mobile device (Fig. 1). SWAN was designed to support easy programming of context-aware applications for Android. To this end, it provides a high-level domain specific language that makes it easy to fetch and combine data coming from various sensors. SWAN has built-in support for most of the sensors present in modern smartphones. External sensors, like heart rate monitors or wearables can be integrated via plug-ins. In addition to the ease of use, SWAN provides centralized access to sensors, thus eliminating the redundancy caused by multiple sensing applications running in parallel.

Applications can access the SWAN framework by using the SWAN-Song language [12]. With SWAN-Song it is possible to access various sensors in a uniform manner, without having to know their particular APIs. As an example, the following expression gets the current light intensity:

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

In this expression, `self` is the *location identifier* of the sensing device, `light` is the sensor identifier, `lux` is the sensor attribute that has to be read, `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.

While SWAN has support for distributed sensing, this support is rather limited. Access to remote devices is done in SWAN via cloud messaging, which has high latency and high energy consumption. We notice that in the case of collaborating sensing devices that are in close proximity to each other, these issues can be overcome by using device-to-device communication protocols, like Bluetooth or BLE, which have low latency and reduced energy consumption. We show that by leveraging these techniques, we greatly reduce the energy footprint and improve the response times.

III. SenseLE Framework

The main motivation behind SenseLE is to create an easy to use and energy efficient tool for developing distributed mobile sensing applications for smartphones. To achieve this goal, our framework employs the following key features:

- **Discoverability.** SenseLE provides a simple mechanism for discovering neighboring devices, which is based on the BLE discovery service from Android.
- **Usability.** SenseLE seamlessly handles communication errors, churn and other issues that might result from the mobility of the collocated sensing devices. This is achieved by using a modular architecture in which the failure of one component does not affect the others. In case of such an event, the specific component is recovered with minimal overhead.
- **Efficiency.** As shown by our evaluations, SenseLE has good response times and minimal energy consumption under stress conditions.
- **Portability.** SenseLE exposes a simple API for accessing remote sensors on Android smartphones. The API is based on the SWAN-Song language [12], that provides

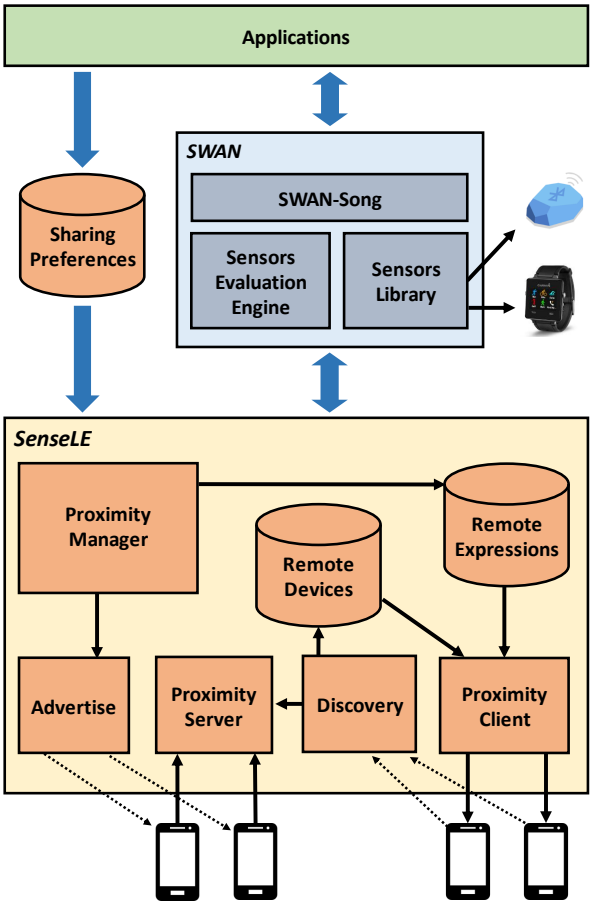

Fig. 1. SenseLE architecture

uniform access to sensors and facilitates the creation of context expressions. The API can be easily extended to other devices that run Android, like tablets or smartwatches.

In the following sections we describe the architecture of SenseLE, along with the design and implementation challenges that we encountered.

A. Architecture

The SenseLE architecture is shown in Fig. 1. As can be seen, SenseLE acts as a transport layer between the SWAN sensing library and other devices. The main function of SenseLE is to enable sharing of sensor data between collocated smartphones over device-to-device (D2D) connections. In particular, we use Bluetooth Low Energy for connecting devices due to its energy efficiency and low latency properties. In the past we studied the feasibility of using standard Bluetooth for the same purpose [8]. We show that the two protocols, despite being related, require different approaches in practice. We also show that Bluetooth Low Energy represents a better choice for distributed sensing both functionally and performance-wise.

An important reason for choosing BLE as communication protocol in SenseLE is the way information is structured within BLE, which is similar to the way we structure information in SenseLE. In SenseLE, each sensor is assigned an identifier, called *sensor entity* and one or more *value paths*. The value paths correspond to the different types of information produced by a sensor. For example, the GPS sensor has *latitude* and *longitude* as value paths. Similarly, BLE devices use Generic Attribute (GATT) profiles that structure data in *Services* and *Characteristics* [13]. Therefore, we can match each sensor entity in SenseLE with a BLE service and each value path with a characteristic.

Applications interact with SenseLE indirectly, through the SWAN library API, which has two methods.

```
registerExpression(id, body, callback);

unRegisterExpression(id);
```

The first method is used to register a context expression written in the SWAN-Song language. The first two parameters denote the expression ID and the expression body, while the last one is a callback that is invoked whenever new sensed data is available. SWAN-Song expressions can be of two types: local and remote. A local expression is prefixed with the *self* keyword and indicates that the sensed data comes from a local sensor. A remote expression is prefixed either with 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 SenseLE connects to the device having that ID (if the device is in proximity) and fetches sensor data from it. Otherwise, if NEARBY is used, then sensor data is continuously polled from all nearby devices having the requested sensor. In the next paragraphs we focus only on the remote expressions, as they are handled by SenseLE.

Upon receiving a remote context expression, SenseLE invokes the *Proximity Manager*, which first saves the expression in a local database (*Remote Expressions*) and then starts advertising the expression to other devices in proximity, so the latter can prepare for sending back sensor data. If one or more devices that have the requested sensor are found by the *Discovery* module, SenseLE connects to them and fetches the sensor data, which is sent back to the caller application. SenseLE supports *pull* and *push* techniques for getting remote sensor data. When the caller application no longer requires remote sensor data, it calls `unRegisterExpression`, so the connection with the remote device is closed and the expression is removed from the Remote Expressions database.

Advertising and Discovery. SenseLE uses the standard advertising and discovery mechanisms provided by the Android API for BLE devices. A BLE-enabled Android device can advertise a number of BLE *services*. Each of these services has several attributes, called *characteristics*. In SenseLE we assign a service to each *sensor entity* and a characteristic to each *value path*. Given that SenseLE currently supports more than 25 sensors and each sensor has on average 4 value paths, the total number of characteristics that have to be advertised

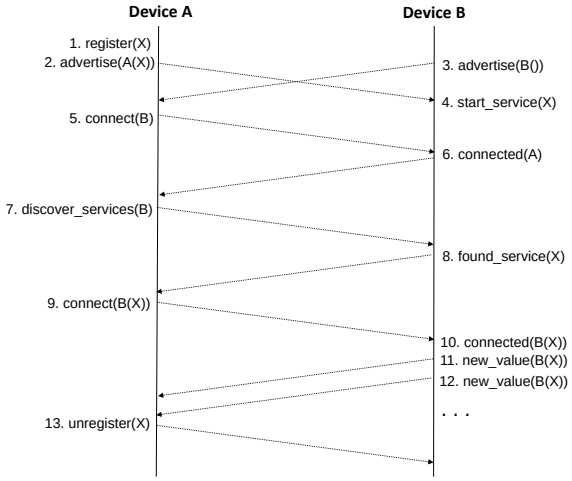

Fig. 2. Connection steps

goes above 100. After conducting several tests, we noticed that advertising this amount of characteristics results in discovery times longer than 30 seconds, which is unacceptable in the context of real-time sensing. To overcome this, we take an alternative approach by assigning BLE services to sensors on demand, based on the sensors requested by other devices. To realize this, each SenseLE device includes in the advertisement packets the IDs of the sensors that it is interested in. Whenever SenseLE detects a nearby device that is looking for a certain sensor, it creates a BLE service for that sensor, such that the other device will be able to discover and connect to that service. When a sensor is no longer needed by nearby devices, the BLE service for that sensor is disabled.

Remote Sensing. Fig. 2 shows the steps for connecting two devices in SenseLE in order to share sensor data. For brevity, in this example data from only one sensor is shared. However, SenseLE supports multi-sensor sharing, as shown by our experiments. Initially, a context expression requesting data remotely from sensor X is registered in SenseLE on device A (1). Then, device A starts advertising its presence (2). It includes in the advertising packets the ID of sensor X, so that other devices can prepare to share data from sensor X with device A. At the same time device B starts advertising its presence (3). When device B discovers device A, it notices that device A is looking for sensor X, so it starts a BLE service for serving data from sensor X (4). When device A discovers device B, it initiates a connection (5). Upon successfully connecting with device B, device A invokes a BLE service discovery on device B (7). After learning that device B has a service for sensor X (8), device A connects to the service and starts fetching data (9, 10, 11, 12). When data from the remote sensor is no longer needed, the sensor is unregistered (13).

Privacy. SenseLE employs a basic mechanism that lets the user decide which sensors are shareable and which are not. These preferences are stored internally in the *Sharing Preferences* key-value store (Fig. 1).

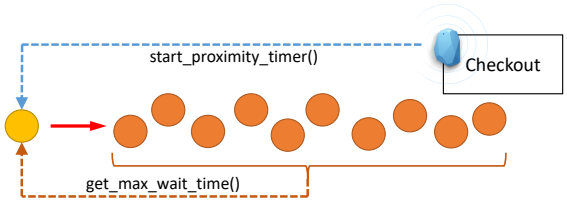

Fig. 3. SmartWait app

Incentivizing the user to share sensor data is a key aspect related to privacy. Since SenseLE is used by other applications merely as a tool for accessing remote sensor data, the proper functioning of these applications is conditioned by the user's consent to share sensor data required by the applications. For example, the SmartWait app (see section III-B) functions properly only if the user agrees to share information about her presence at a queue with other app users.

With privacy being outside the scope of this paper, we provide this mechanism merely as a mean for the end user to limit the sensor data that is shared with others, similar to the Android permissions required by applications. For more advanced privacy control, our framework can be further enhanced by applying the techniques described in our previous work [14], which use homomorphic encryption for anonymizing IoT data.

B. Use Case - The SmartWait app

As use case for SenseLE, we prototype a simple application for estimating waiting times at queues. Our application, called SmartWait, relies on the simple principle that a rough estimation of the waiting time at a queue can be obtained by computing the maximum waiting time among the other persons that are waiting. With SenseLE, computing this maximum can be done easily by remotely detecting the waiting times of others (Fig. 3).

The application works as follows: when the user arrives at the queue, her SmartWait app detects the queue by connecting to a beacon that advertises the presence of that queue, then starts a timer. After that, the SmartWait app uses SenseLE to gather the waiting times of the other people at the queue and computes the maximum among them in real-time. At the same time, SenseLE disseminates in the background the waiting time of the user to the people who arrived later.

Programming this application without the help of SenseLE would normally take much time, as it requires knowledge of the APIs for accessing beacons and smartphones over D2D links, as well as synchronizing simultaneous connections to other devices. With SenseLE, it only takes two context expressions to achieve the same functionality:

```
self@beacon_sensor:discovery

MAX(NEARBY@beacon_sensor:time_in_range)
```

The first expression is to get a notification when the beacon assigned to the queue is in range, while the second detects the

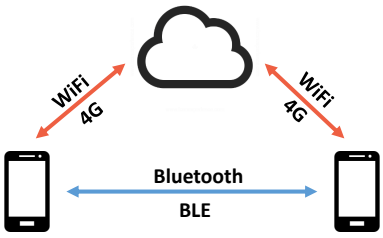

Fig. 4. Cloud-centric sensing vs. device-to-device sensing

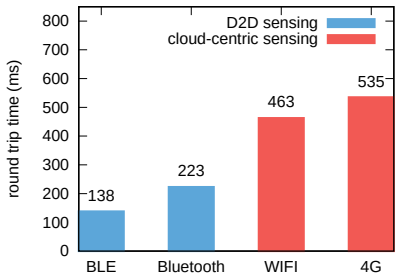

Fig. 5. Latency for different communication protocols

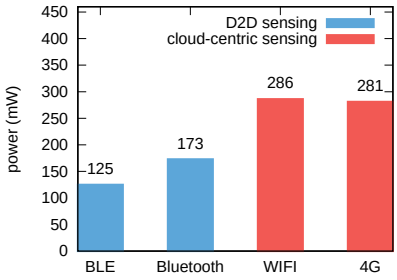

Fig. 6. Power usage for different communication protocols

waiting times of the other people waiting and computes the maximum among them.

We note that the SmartWait app provides only a rough approximation of the waiting time at a queue, its accuracy being highly dependent on the dynamics of the queue. Still, it proves the ease of building distributed sensing applications with SenseLE, without the need of a cloud infrastructure.

IV. Evaluation

We implemented SenseLE as a library for Android. We tested our implementation on a small group of smartphones, consisting of 2 Nexus 6P phones running Android 7, 3 Nexus 5X phones running Android 7 and one Nexus 5 phone running Android 6 (Fig. 7). We chose this combination of devices as they all have support for Bluetooth Low Energy and they all run unmodified versions of Android.

We benchmarked SenseLE against our own implementation of the SWAN-Lake framework, which we previously developed as a solution for opportunistic distributed sensing [8] (see Section II). We made this choice as we were interested to analyze the effectiveness of the power usage improvements employed by SenseLE compared to our previous work. Therefore, we focus our comparisons on energy efficiency.

In order to test how SenseLE performs compared to a cloud-centric solution, we also implemented a variation of SenseLE that is using the cloud as a proxy for sharing sensor data between phones (Fig. 4). For the cloud implementation we used an instance of Cowbird [15] installed on our university's cloud infrastructure.

Our experiments consist of small groups of phones (2-6) sharing sensor data with each other. The following sensors were used in our tests: light, accelerometer, gyroscope, magnetometer, battery, proximity, pressure, microphone and GPS. We made this selection as all of these sensors are commonly found in modern smartphones. In all experiments the phones exchange sensor data continuously for a duration of 5 minutes, using a sample interval of one second between sensor readings. All the results shown represent averages obtained from at least 5 sample runs. For measuring the power consumption of the phones we used the Trepn profiler [16]. All power measurements are relative to a baseline of 254 mW, which was measured with the phone being in idle state. For all experiments we kept the phone screen on and disabled all running applications.

Fig. 7. Experimental setup

Device-to-device (D2D) sensing vs. cloud-centric sensing. We first show that D2D sensing performs better than cloud-centric sensing for low-bandwidth applications like sensing in terms of power consumption and latency. To this end, we connect two phones either directly (over Bluetooth/BLE) or via a cloud infrastructure (over WiFi/4G) and have one of the phones sharing sensor data with the other over 5 minutes. We used the setup shown in Fig. 4 to test the two scenarios. When analyzing the round-trip time (Fig. 5), we can see that sensing is almost 4 times faster when using BLE, compared to the 4G case, 3 times faster compared to the WiFi case and considerably better compared to standard Bluetooth. We note that all time values presented here include the time taken by SenseLE to process the sensor data internally, which is 60 ms on average. Also, the cloud has only the role of a router, therefore the overhead of processing the data in the cloud is negligible. The long round-trip times for WiFi and 4G can be explained by the communication overhead of routing all data through the cloud, which involves at least two wireless links. A similar situation is revealed if we compare the power consumption of D2D sensing with cloud-centric sensing (Fig. 6). We can see here that remote sensing over BLE consumes two times less power compared to the cloud-centric approach. Also, sensing over BLE is more energy efficient than sensing over standard Bluetooth. The results above lead us to the conclusion that it is preferable to perform remote sensing over D2D links whenever the amount of shared data is small. In cases where large streams of data have to be transferred, like video or sound streams, then a cloud-centric solution is desirable as it has larger bandwidth.

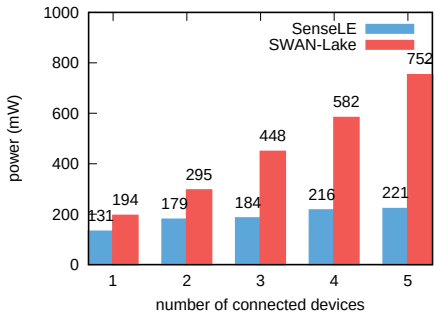

Fig. 8. Power usage for different numbers of connected devices

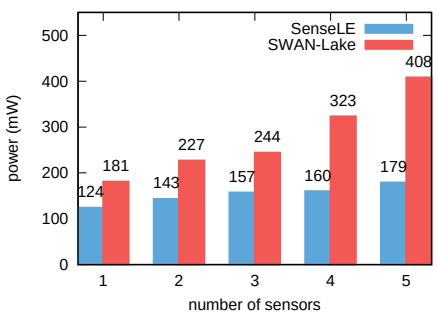

Fig. 9. Power usage for different numbers of remote sensors

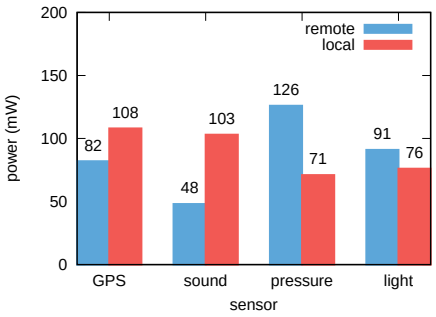

Fig. 10. Local sensing vs. remote sensing

Multi-device sensor sharing. It is interesting to see how SenseLE performs in settings where sensor data comes from multiple devices simultaneously. For example, the SmartWait app (see section III-B) connects to multiple close-by users to learn their waiting times. Therefore, in the next test we look at the power consumed by SenseLE when a phone fetches sensor data from 1-5 devices (Fig. 8). Here, we can observe that the overhead of having more than one connected device is minimal in SenseLE. However, we notice a considerable growth in the power consumption of SWAN-Lake. This is expected, as SWAN-Lake uses Bluetooth as underlying communication protocol, which has higher energy consumption than BLE. Still, the results show that SenseLE not only has better energy management, but also scales better when the number of concurrent connections increases.

Multi-sensor sharing. In the next experiment we study SenseLE's efficiency when data from multiple sensors is shared between two phones. In this experiment, one phone continuously polls data from up to 5 sensors of another phone (Fig. 9). Here, we can see that SenseLE scales much better in terms of power consumption with the number of shared sensors compared to SWAN-Lake. For 5 sensors, the power consumed by SenseLE is more than 2 times lower compared to SWAN-Lake. This can be attributed to the adaptive approach we implemented in SenseLE for assigning BLE services to requested sensors (see Section III-A). This is not possible in SWAN-Lake, as Bluetooth, which is the underlying communication protocol, does not support the notion of a service.

Local sensing vs. remote sensing. In Fig. 10 we compare the power consumed by getting data from the onboard sensors with the power consumed by transferring data from the same sensors on a remote device. For this experiment we used 4 sensors that detect changes of the environment that are common to a group of close-by smartphones: light, sound (microphone), pressure and GPS location. Here, the results are more balanced. We observe that reading the GPS and sound sensors remotely uses significantly less energy, while using the pressure and light sensors locally is more efficient. However, we do not see many potential useful applications that require users to share data from their light and pressure sensors. On the contrary, sharing of GPS and sound sensors can prove useful in many situations. For example a group of runners can use location data from only one of their phones in order to save energy, or a crowdsensing application for monitoring noise levels in a city can intelligently pick data from one device in each group of collocated users.

Group sensor sharing. To have a better understanding of the impact of sensor sharing (over BLE) on battery usage, we analyze next the battery consumption of a group of 3 smartphones that share data from one GPS sensor (Fig. 11). We choose to analyze the GPS sensor, as it is common for groups of people that perform activities together (like sport or travel) to use applications that require user's location. In this experiment, one phone collects data from the onboard GPS sensor and shares this data with the other 2 phones in the group. We let the experiment run for 3.5 hours and measure the average battery consumption of the group. The GPS readings were performed at the default rate, which is around one reading per second. We compare the case where the phones share data from one GPS sensor against the case where each of the phones uses its local GPS. The results show that sharing sensor data reduces the average battery usage of the group by 13%. We notice that having one phone sharing its sensor data within the group over extended periods of time can lead to a more rapid drain of its battery compared to the others. To achieve better load balancing, the role of sensor data provider can be assigned over time to different phones in the group in a round-robin fashion.

Discovery. Finally, we analyze the overhead of nearby device discovery in SenseLE. Fig. 12 contrasts the power usage of BLE discovery (used by SenseLE) with that of standard Bluetooth (used by SWAN-Lake). We tested two discovery modes: ON, where we keep discovery enabled for the whole duration of the experiment, and Intermittent, where we switch discovery between on and off. For the Intermittent case we use a duty cycle of 50%, as previous research [17] indicates that this value gives optimal energy consumption, and a period[1] duration of 20 seconds. We chose this value based on the fact that in standard Bluetooth a discovery cycle lasts around 10 seconds. Therefore, by choosing a discovery period of 20 seconds, we have discovery cycles of 10 seconds followed by

[1]a period is the time interval between two discovery cycles

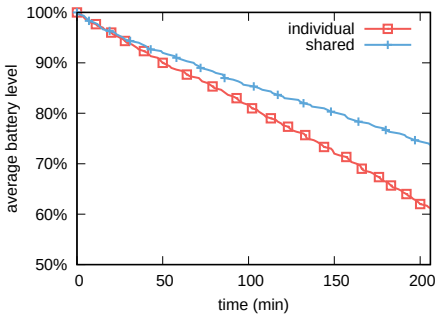

Fig. 11. Battery usage for group sensor sharing

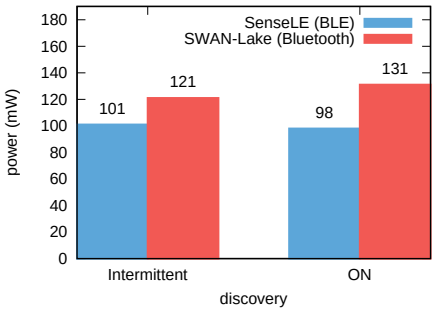

Fig. 12. Power usage for device discovery

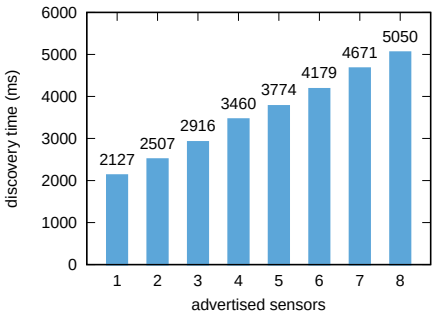

Fig. 13. Sensors discovery time

idle intervals of 10 seconds, thus preserving the 50% duty cycle. As expected, BLE is slightly more energy efficient than standard Bluetooth. Interestingly, BLE performs slightly better when discovery is left turned on for the whole duration of the experiment. Moreover, we found that having discovery turned on in SenseLE does not affect the performance of transferring sensor data in parallel. This is a major improvement over using standard Bluetooth for communication, as standard Bluetooth does not allow open connections while discovery is enabled.

We also analyze the time it takes for SenseLE to discover the sensors available on a nearby phone (Fig. 13). In this test up to 8 sensors are advertised by the remote device. We notice here a linear growth of the discovery time with the number of advertised sensors (the sensors are advertised as BLE services). Also, the discovery time is rather high, varying from 2127 ms to 5050 ms. We address this limitation in SenseLE by enabling and disabling BLE services according to the remote sensors demanded by nearby devices (see Section III-A).

V. Related Work

In this section we discuss several categories of work related to SenseLE. We begin by discussing systems that employ *collaborative and distributed sensing*. Then, we analyze systems that use *BLE communication protocols* to optimize communication between mobile devices. Finally, we discuss *IoT designs and models* related to our sensing framework.

A. Collaborative and Distributed Sensing

MOSDEN [18] is a mobile framework that exploits the opportunistic collocation of mobile devices to perform collaborative sensing. MOSDEN distinguishes itself from other solutions by having a modular and extensible architecture that makes it easy to develop collaborative sensing applications for Android. Unlike SenseLE, MOSDEN uses only WiFi for communication and it does not support sharing of sensor data with anonymous devices.

Another body of related work includes solutions for collaborative context monitoring. CoMon [19] is a platform for cooperative context monitoring between collocated devices that uses a benefit-aware negotiation mechanism in order to improve the outcome of the cooperation in terms of energy usage. A similar approach is taken by Panorama [20]. In addition to collocated devices, Panorama also involves clouds and cloudlets in the cooperation process in order to increase the performance of continuous context monitoring. While CoMon and Panorama focus on high-level models for collaboration between devices, in SenseLE we focus on the practical challenges that arise when connecting groups of devices in a peer-to-peer fashion.

B. BLE Communication Protocols

In [17], Radhakrishnan et al., provide a first empirical evaluation of the BLE communication protocol for smartphones. In this study, the authors show that, depending on usage patterns, BLE is only marginally consuming less power than Bluetooth. For a lower power consumption, the authors recommend a low *duty cycle* (i.e., the fraction of time the device is actively scanning). In SenseLE, we also implement this optimization.

In [21], Levy et al. introduce Beetle, a new operating system service that mediates the communication between applications and BLE-enabled devices. Beetle provides a separation between applications and the sensors they are entitled to access. Furthermore, Beetle offers application developers the ability to *share* sensors between multiple applications, *fine-grained access control*, and efficient *many-to-many* communication through BLE. Such research is orthogonal to SenseLE, as it offers another layer of abstraction on top of raw BLE communication capabilities.

A project with some similarities is CoTrust [22], a decentralized middleware for establishing trusted connections between co-located mobile devices. CoTrust employs a model that enables trusted collaborations between mobile devices based on the social network interactions of their users. While CoTrust focuses on trust mechanisms for device collaboration (i.e., offloading computation, sensing, chat, and file transfer), the main focus of SenseLE is the efficient sharing of sensor data between devices.

C. IoT Designs and Models

Initially, IoT solutions were designed in cloud-centric fashion [3], where all data generated by IoT devices are stored and analyzed in clouds. However, this scenario is not suitable for applications where response times are critical, as the latency to access clouds is generally large. To improve on the cloud centric system, fog or edge computing has been proposed [23], [24], [25]. Such techniques focus on optimizing

latencies by placing computation at the "network edge", or in the "fog", i.e., closer to the data sources. Our approach takes such optimizations even further: we exploit spatial locality of mobile devices by performing direct, device-to-device low energy sharing of data.

VI. Conclusions

Distributed sensing and monitoring are becoming increasingly important as they are able to improve and optimize several aspects of modern life. From traffic monitoring and routing, to disaster management and real-time athlete coaching, all such applications can highly benefit from efficient data dissemination frameworks. However, such applications are less suitable for the traditional *cloud-centric* model, where data are sent and processed in clouds, as such solutions incur large latency penalties, limiting real-time responsiveness.

This paper discusses the design and implementation of SenseLE, a framework for opportunistic acquisition and processing of sensor data from nearby devices. SenseLE is able to exploit spatial locality in order to improve response times and minimize power consumption.

Our evaluation shows that SenseLE is able to reduce communication latencies by up to 3X compared to the cloud-centric model. Instead of querying clouds or reading energy-hungry local sensors (e.g., GPS), SenseLE performs remote low-energy sensing, resulting in power consumption reduced by up to 56%. Furthermore, we learned that BLE is much better suited than standard Bluetooth for low-volume sensor communication. However, the communication software needs to be redesigned to exploit the advantages of BLE. This is illustrated by the fact that SenseLE is a complete redesign of our earlier SWAN-Lake system.

As future work, we plan to improve the privacy mechanisms of SenseLE and to investigate the scalability of our framework by conducting a large scale study involving many sensing devices with high mobility.

Acknowledgements

This work is funded by the Dutch public-private research community COMMIT/. We thank Kees Verstoep for his comments that greatly improved the manuscript.

References

[1] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Sensing as a service model for smart cities supported by internet of things," *Transactions on Emerging Telecommunications Technologies*, vol. 25, no. 1, pp. 81–93, 2014.

[2] 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, 2010.

[3] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[4] T. Zachariah, N. Klugman, B. Campbell, J. Adkins, N. Jackson, and P. Dutta, "The internet of things has a gateway problem," in *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*. ACM, 2015, pp. 27–32.

[5] A. Stopczynski, J. E. Larsen, S. Lehmann, L. Dynowski, and M. Fuentes, "Participatory Bluetooth sensing: A method for acquiring spatio-temporal data about participant mobility and interactions at large scale events," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013 IEEE International Conference on*. IEEE, 2013, pp. 242–247.

[6] 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*. ACM, 2011, pp. 73–84.

[7] S. B. Eisenman, E. Miluzzo, N. D. Lane, R. A. Peterson, G.-S. Ahn, and A. T. Campbell, "Bikenet: A mobile sensing system for cyclist experience mapping," *ACM Transactions on Sensor Networks (TOSN)*, vol. 6, no. 1, p. 6, 2009.

[8] N. V. Bozdog, R. Bharath Das, A. Van Halteren, and H. Bal, "Swan-lake: Opportunistic distributed sensing for Android smartphones," in *Proc. of the 11th EAI Int. Conf. on Body Area Networks*. EAI, 2016.

[9] D. Camps-Mur, A. Garcia-Saavedra, and P. Serrano, "Device-to-device communications with Wi-Fi Direct: overview and experimentation," *IEEE wireless communications*, vol. 20, no. 3, pp. 96–104, 2013.

[10] A. Fahim, A. Mtibaa, and K. A. Harras, "Making the case for computational offloading in mobile device clouds," in *Proceedings of the 19th annual international conference on Mobile computing & networking*. ACM, 2013, pp. 203–205.

[11] R. Kemp, "Programming frameworks for distributed smartphone computing," Ph.D. dissertation, Vrije Universiteit, Amsterdam, 2014.

[12] N. Palmer, R. Kemp, T. Kielmann, and H. Bal, "Swan-song: a flexible context expression language for smartphones," in *Proc. of the Third Int. Workshop on Sensing Applications on Mobile Phones*. ACM, 2012.

[13] C. Gomez, J. Oller, and J. Paradells, "Overview and evaluation of Bluetooth low energy: An emerging low-power wireless technology," *Sensors*, vol. 12, no. 9, pp. 11 734–11 753, 2012.

[14] M. X. Makkes, A. Uta, R. Bharath Das, N. V. Bozdog, and H. Bal, "P2-swan: Real-time privacy preserving computation for iot ecosystems," in *International Conference on Fog and Edge Computing*. IEEE, 2017.

[15] R. Bharath Das, N. V. Bozdog, and H. Bal, "Cowbird: A flexible cloud-based framework for combining smartphone sensors and IoT," in *Proceedings of the 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*. IEEE, 2017.

[16] "Trepn power profiler," https://developer.qualcomm.com/software/trepn-power-profiler, accessed: 2017-07-10.

[17] M. Radhakrishnan, A. Misra, R. K. Balan, and Y. Lee, "Smartphones and BLE services: Empirical insights," in *Mobile Ad Hoc and Sensor Systems (MASS), 2015 IEEE 12th Int. Conf. on*. IEEE, 2015.

[18] P. P. Jayaraman, C. Perera, D. Georgakopoulos, and A. Zaslavsky, "Efficient opportunistic sensing using mobile collaborative platform Mosden," in *Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom), 2013 9th Int. Conf. on*. IEEE, 2013.

[19] Y. Lee, Y. Ju, C. Min, S. Kang, I. Hwang, and J. Song, "Comon: Cooperative ambience monitoring platform with continuity and benefit awareness," in *Proceedings of the 10th international conference on Mobile systems, applications, and services*. ACM, 2012, pp. 43–56.

[20] K. Alanezi, X. Zhou, L. Chen, and S. Mishra, "Panorama: A framework to support collaborative context monitoring on co-located mobile devices," in *International Conference on Mobile Computing, Applications, and Services*. Springer, 2015, pp. 143–160.

[21] A. A. Levy, J. Hong, L. Riliskis, P. Levis, and K. Winstein, "Beetle: Flexible communication for Bluetooth low energy," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2016, pp. 111–122.

[22] K. Alanezi, R. I. Rafiq, L. Chen, and S. Mishra, "Leveraging BLE and social trust to enable mobile in situ collaborations," in *Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication*. ACM, 2017, p. 98.

[23] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[24] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

[25] A. V. Dastjerdi and R. Buyya, "Fog computing: Helping the internet of things realize its potential," *Computer*, vol. 49, no. 8, pp. 112–116, 2016.