GaaS: Adaptive Cross-Platform Gateway for IoT Applications

Mohamed Abdelaal, Mochamad Dandy, Frank Dürr, Kurt Rothermel
Institute of Parallel and Distributed Systems, University of Stuttgart
Email: first.last@ipvs.uni-stuttgart.de

Marwan Abdelgawad
German University in Cairo, Cairo, Egypt
Email: first.last@guc.edu.eg

***Abstract*—Internet of Things (IoT) is expanding at a rapid rate where it allows for virtually endless opportunities and connections to take place. In general, IoT opens the door to a myriad of applications but also to many challenges. One of the major challenges is how to efficiently retrieve the sensory data from "resources-limited" IoT devices. Such devices typically have a restricted energy budget, which broadly hinders their direct connection to the Internet. In this realm, modern mobile devices, e.g. smartphones, tablets, smartwatches, have been harnessed to bridge between the low-power IoT devices and the Internet. However, the current vision which mainly relies on designing *siloed* gateways, i.e. a separate gateway/App for each IoT device, is certainly impractical, especially with the rapid growth in the number of IoT devices. Furthermore, energy efficiency of the smart mobile devices hosting the IoT gateways has to be thoroughly considered.**

To tackle these challenges, we introduce GaaS (Gateway as a Service), a cross-platform gateway architecture for opportunistically retrieving sensory data from the low-power IoT sensors. Through Bluetooth low energy radios, GaaS is capable of simultaneously connecting to several nearby IoT sensors. To this end, we devise two distinct *priority-based scheduling* algorithms, namely the EP-WSM and FEP-AHP schedulers, which rank the detected IoT sensors, before estimating the connection time for each IoT sensor. The intuition behind ranking the IoT sensors is to improve the data retrieval rate from these sensors together with reducing the energy overhead on the mobile devices. Additionally, GaaS encompasses a self-adaptive engine to automatically balance the trade-off between energy efficiency and data retrieval rate through switching between schedulers according to the runtime dynamics. To demonstrate the effectiveness of GaaS, we implemented an IoT testbed to evaluate the energy consumption, the latency, and the data retrieval rate. The results show that using GaaS, compared to siloed gateways, we can identify up to 18% savings in the consumed energy while requiring much less data retrieval time.

***Index Terms*—Internet of Things, Opportunistic Gateways, Interoperability, Priority-Based Scheduling**

I. Introduction

Recently, the term Internet of Things (IoT) has been broadly utilized to express the connectivity of physical objects and sensors to the Internet to share and collect data. The growth in the IoT market is primarily driven by the technological advancements in sensing modalities, wireless communication, and cloud computing. These features open the door for a myriad of applications in health, industry, and smart homes [1]. Depending on the application, IoT designers may select between two communication models. In the first model, IoT sensors directly report their measurements to a cloud service. Nevertheless, these IoT sensors are typically "resource-constrained" devices in terms of the allocated energy and computing capabilities. Accordingly, the adoption of "energy-expensive" communication facilities, such as cellular and WiFi networks, to directly upload the measurements is hardly feasible in most IoT applications .

Alternatively, the second communication model involves deploying a gateway which effectively bridges the communication gap between the various IoT devices and a cloud service. In this context, a research question arises about the suitability of our smart mobile devices, e.g. smartphones, tablets, smartwatches, to act as gateways whenever they exist in the vicinity of the IoT devices. Indeed, the commercially-available mobile devices possess several features which enable them from playing such a role, including (1) their widespreadness where recent studies foresee that the number of smartphone subscriptions will exceed 6.1 billion by 2020 [3], (2) they are always connected thanks to being equipped with several communication interfaces such as WiFi, 3G/LTE, near-field communication (NFC), and Bluetooth low energy (BLE) and (3) they have relatively significant storage capability as well as computing power. Thanks to these features, several research activities have been devoted to explore the potential of employing smart mobile devices as data mules or opportunistic mobile gateways [4], [5], [6].

Nevertheless, relying on mobile devices as opportunistic gateways poses several challenges, including the siloed data collection architecture, failures of mobile devices, and handling the quality-of-service (QoS) metrics, e.g. energy overhead on the mobile devices, throughput, and latency. Driven by the growth of the IoT market, many companies and startups are rushing to produce new connected IoT sensors, e.g. Xiaomi MiBand [7] and Fitbit [8]. Each of these IoT devices has its own mobile gateway/App—designed by its vendor—to retrieve the sensory data. Such a siloed architecture requires the installation of a new gateway/App whenever an IoT device is to be connected to a mobile device. Accordingly, wireless connectivity to the IoT devices is restricted to those mobile devices which have the corresponding gateways. This closed and siloed architecture broadly hinders the growth of this class of IoT devices. Additionally, smart mobile devices introduce an unnecessary failure point for the connected IoT devices. If these mobile devices are not present or are discharged,

This work is supported by the German Federal Ministry of Education and Research (BMBF) grant 01DH17059.

Published in the 16th IEEE International Conference on Mobile Ad-Hoc and Smart Systems (IEEE MASS), November 4 - 7, 2019, Monterey, CA, USA.
© IEEE 2019

the attached IoT devices become completely unreachable. Therefore, there is an immense need to design a cross-platform gateway architecture for connecting low-power IoT sensors to the Internet.

Another major challenge of relying on mobile devices as opportunistic gateways is retrieving data from several IoT sensors at the same time. Being mobile implies that an opportunistic gateway will probably exist in the vicinity of various IoT sensors for a relatively short time. Therefore, a cross-platform gateway has to connect and swiftly retrieve as much sensory data as possible from all IoT sensors while considering the energy overhead on the mobile devices. In fact, the mobile devices—acting as cross-platform gateways—have to frequently scan the wireless channel to discover the IoT sensors. However, current mobile devices typically have limited energy sources, e.g. Sony Xperia Z5 has only 2900 mAh [9]. Hence, a challenge of compromising the trade-off between energy consumption and the other relevant QoS metrics, such as throughput and latency, emerges.

To tackle these challenges, we introduce GaaS, an adaptive *cross-platform* gateway architecture. The proposed architecture—developed on the Android operating system—is capable of simultaneously connecting to multiple IoT devices through Bluetooth low energy (BLE) connections. In this paper, BLE has been selected thanks to its low-power performance while providing moderate data rates. Additionally, BLE modules are nowadays embedded in most modern mobile devices, i.e. smartphones and laptops. Once a connection is established, GaaS simultaneously receives sensory data from active IoT sensors before either forwarding the data to a cloud service or processing (probably also visualizing) the collected data to provide the users with useful information. To this end, a scheduling strategy is required to determine the connection time and order. However, our application scenario is highly dynamic in terms of the number of available IoT devices and the remaining energy of the mobile devices hosting the gateways. Accordingly, GaaS employs an adaptive scheduling mechanism, based on the Monitor-Analysis-Plan-Execute (MAPE-K) control loop [10] combined with Fuzzy logic, to react to the runtime dynamics. In fact, the amount of data which needs to be communicated to a gateway differs among the IoT devices. Therefore, it is critical to assign longer time slots to the IoT devices which possess more sensory data. In this context, GaaS leverages two different multi-decision making algorithms to assign higher priority levels to those IoT devices before generating the connection schedule.

In detail, the paper provides the following contributions: (1) We define an architectural framework for a generic mobile gateway which can be connected to several IoT devices at the same time. Our proposed gateway leverages the so-called Android Interface Definition Language (AIDL) to provide services for the connected IoT devices. Such services ranges from data processing through visualization to data upload to cloud services. (2) We devise an adaptive scheduling mechanism, based on the exhaustive polling (EP) and fair exhaustive polling (FEP) strategies, to enable the generic gateway from simultaneously connecting to several IoT devices. The scheduling algorithm leverages the MAPE strategy to dynamically modify the generated schedule according to the number of connected IoT devices and the residual energy at the mobile device. In this context, the well-known Fuzzy logic theory has been exploited for reasoning the collected statistics about the mobile gateway. (3) We introduce a *priority-based* scheduling algorithm to further improve the efficiency of GaaS through ranking the IoT devices before generating a certain schedule. To this end, GaaS leverage two well-known *multi-criteria* decision making techniques, namely analytic hierarchy process (AHP) [11] and weighted sum model (WSM) [12] for ranking the nearby IoT devices. (4) We present a proof-of-concept implementation and evaluation of GaaS in a real-world scenario. We implemented the generic gateway on Android through which sensory data—from commercial IoT devices as well as sensors connected to Arduino kits—can be retrieved. The main goal of these evaluations is to quantify the power overhead, latency, and data throughput in different scenarios. The results show that using GaaS, we obtain a significant improvement of the energy efficiency relative to the siloed architecture with a comparable throughput and latency.

The remainder of this paper is organized as follows: Section II introduces the system model, the architectural framework of GaaS, and our assumptions. Section III presents the software architecture of the generic gateway with discussing each component. The adaptive and priority-based scheduling techniques are discussed in Section IV. In Section V, we present our real-world evaluation and our prototype implementation of the generic gateway architecture, before discussing the obtained results. Finally, Section VI highlights the main differences between GaaS and related work, before Section VII draws a conclusion with an outlook on future work.

II. System Overview

In this section, we explain the system model together with our assumptions. Figure 1 depicts the system model where a mobile device exists in the vicinity of a set of IoT sensors $\mathcal{S} = s_1, \cdots, s_n$ where $n \geq 1$. Each IoT sensor measures a certain phenomenon, e.g. temperature, speed, and electricity consumption, before periodically uploading these measurements to the opportunistic gateway implemented on the mobile device. To this end, the IoT sensors are assumed to be equipped with a BLE module through which it can connect to the opportunistic gateway. After collecting the sensory data, the opportunistic gateway employs a cellular or WiFi network to deliver the data to a cloud service or to a back-end server. If the gateway received sensory readings while it has no Internet connection, the readings are temporarily stored on the physical storage of the mobile device to be uploaded to the cloud as soon as an Internet connection is established.

To define the initial behavior of GaaS, the *system settings* component is an XML file comprising the main configurations. For instance, the system settings define the initial scheduling algorithm, timing constraints for scheduling the BLE connections, self-adaptation thresholds, and metadata

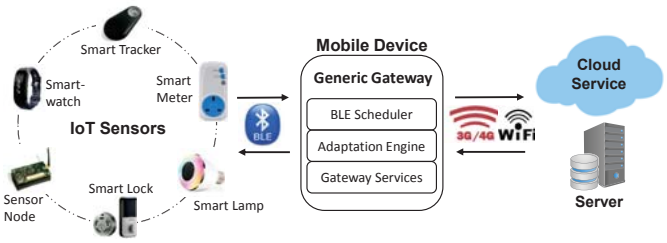


Fig. 1: GaaS system architecture

about the supported manufactures. In GaaS, we assume that the cross-platform gateway has knowledge about the IoT sensors. Specifically, GaaS comprises a list of all supported IoT sensors—from different manufacturers—in the *system settings*. Accordingly, GaaS is aware of the services required by each supported IoT sensor to properly handle the collected data.

III. Generic Gateway Architecture

In this section, we introduce the proposed architecture which defines our cross-platform gateway. Figure 2 depicts the main components of GaaS, including BLE interface, gateway main service, priority-based scheduling, and self-adaptation engine. At the outset, the *discovery manager* detects the nearby IoT sensors through scanning the available BLE interface, before obtaining their metadata, i.e. RSSI, MAC Address, available services, manufacturer, etc. To retrieve data from the detected IoT sensors, the *connection manager* establish BLE connections to *all* available BLE interfaces. Nevertheless, the gateway filters out the BLE interfaces which are not supported by the gateway. Below, we explain the role of each component to achieve a proper behavior of the gateway.

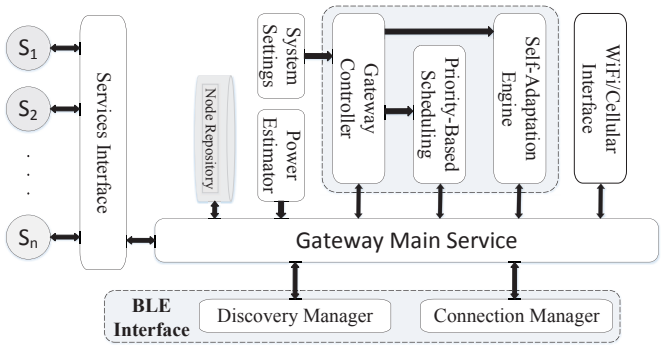


Fig. 2: GaaS software components

A. Gateway Controller

The *gateway controller* (GC) primarily serves as the starting point of the proposed architecture. In particular, the controller is responsible for executing the initial configurations retrieved from the system settings, e.g. initial scheduling algorithm and execution period of the self-adaptive engine. Furthermore, the GC initializes and terminates the gateway main service. Figure 3 illustrates the sequence diagram of our proposed gateway architecture. Once the gateway is activated, the GC controller automatically binds to the *gateway main service* which initializes the AIDL communication model. It is worth mentioning that GaaS leverages multithreading to enable the gateway from establishing connections to several IoT sensors at the same time. Once the `onServiceConnected()` method is triggered, two threads are created to simultaneously run the default scheduling algorithm and the self-adaptive engine. In this context, the scheduler thread is given the highest priority in the multi-threading mechanism to ensure seamless connections to the nearby IoT sensors.

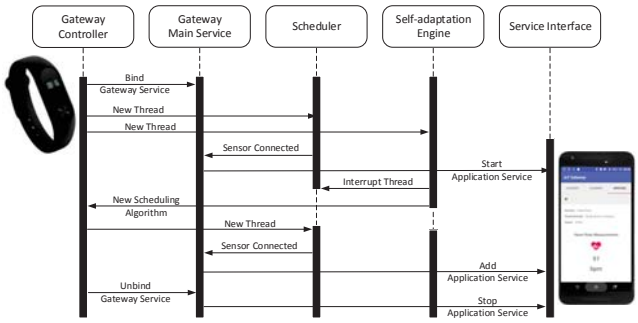


Fig. 3: Sequence diagram of the interactions between the components of GaaS

Whereas, the self-adaptive engine runs periodically to determine the most suitable scheduling algorithm based on the number of connected IoT sensors and the energy budget of the hosting mobile device. Once a new scheduling algorithm is elected by the self-adaptation engine, the scheduler thread is deliberately interrupted so that a new scheduling algorithm is adopted by the GC controller. The service interface is activated when at least one BLE-based IoT sensor has been connected, and sensory data, e.g. heart rate or electricity consumption, has been retrieved. Subsequently, the collected data is either visualized or uploaded to a cloud service. If the gateway is deactivated, the application service is interrupted, before unbinding the gateway service.

B. Gateway Main Service

The *gateway main service* (GMS) basically runs the inter-process communication to provide a programming interface by means of the AIDL method. As shown in Figure 2, the GMS component represents the heart of our proposed architecture where several processes, e.g. necessary database operations, hardware properties reading, activation and deactivation of the scheduler, etc., are offered using AIDL so that the other processes or threads may also use these services and bind to the GMS service via AIDL. Additionally, the GMS service serves as an abstraction of many Android peripheral operations, such as scanning for BLE devices, connecting to IoT sensors, connecting to the cloud services using Wi-Fi or Cellular networks, and handling the node repository. The GMS service has been designed to perform a long-running operation in the background without having to utilize the user interface. Thus, the GMS service complies with the intelligent *Job Scheduler* for optimizing the battery usage [13].

C. Service Interface

In a siloed gateway architecture, there exists typically a single *application service* that either visualizes the collected sensory data or uploads it to a cloud service. For instance, the gateway of a smart meter visualizes electricity consumption readings to enable users to make better decisions about their consumption. Alternatively, GaaS has been designed to simultaneously connect to several IoT sensors. Therefore, several application services are to be implemented to serve the various IoT sensors. According to the number of available IoT sensors, the number of active Android services is dynamically changing during the runtime. GaaS leverages the AIDL inter-process communication method to implement a *service interface* through which data can be exchanged between the main gateway service and the various application services. To perform a remote procedure call, AIDL generates two inner classes namely *Stub* and *Proxy* for handling data marshaling and unmarshalling at the main gateway service and the various application services, respectively.

D. Node Repository

For storing the various sensory data as well as the devices metadata, we exploit the SQLite database provided by the Android system. In GaaS, SQLite is mainly used to store metadata about the nearby IoT sensors, services available on each IoT sensor, and characteristics provided by the services of these nearby IoT sensors. This metadata is then linked to each other to connect between entities and relationship of the underlying databases. Figure 4 depicts the entity-relationship diagram employed by GaaS. Specifically, there exist four central databases, namely *BLE Device Data*, *BLE Service Data*, *Data Upload*, and *BLE Characteristic Data*. First, the BLE Device Data database typically stores metadata about the available nearby IoT sensors where the key is the device ID. Furthermore, this database comprises other entities representing the properties of each nearby IoT sensor, such as sensor name, MAC address, RSSI values, etc.

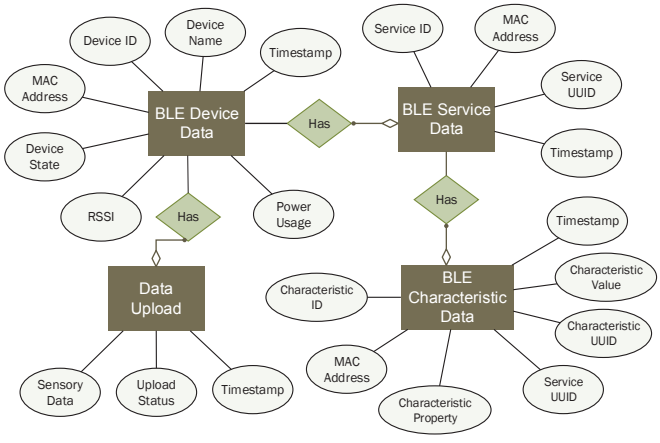


Fig. 4: Entity-relationship diagram of GaaS databases

Second, the BLE Service Data database stores metadata about services available in the IoT sensors where the key of each entry is the service ID. The relationship between such a database and the BLE Device Data database is a "has" operator where one BLE device has one or more services. Third, the BLE Characteristic Data database stores metadata about characteristics from the available services in the nearby IoT sensors where the key is the characteristic ID. Similarly, the relationship between this database and the BLE Service Data database is a "has" relationship where one service has one or more characteristics. Finally, the Data Upload database stores metadata about the sensory data ready to be uploaded to the cloud/back-end server. Each IoT sensor typically has one or more timestamped data entries where the primary key is the MAC address of the IoT device and the timestamp.

E. Self-Adaptation Engine

In this section, we introduce the self-adaptation engine implemented to further improve the relevant QoS metrics, including energy consumption and throughput. As explained in Section IV, the proposed schedulers perform distinctly in different situations. For instance, the gateway may favor a certain scheduler to improve the throughput if the number of connected IoT sensors is suddenly increased. Furthermore, data uploading to a cloud service may be drastically costly if the residual energy of the hosting mobile devices is below a minimum level. Therefore, GaaS has to proactively modify its configurations to save energy and/or retrieve more data from the IoT sensors. To this end, we employ the MAPE-K control loop [10] where two main criteria are frequently monitored while making the adaptation decision, namely the number of connected IoT sensors and the residual energy of the mobile devices hosting the cross-platform gateway.

Based on these monitored criteria, the self-adaptive engine decides upon which priority-based scheduler to adopt and whether to upload the sensory data to the cloud services. To make such decisions, GaaS adopts a lightweight implementation of a *Fuzzy logic* controller to analyze the collected metrics, i.e. residual energy and number of IoT sensors. Specifically, we convert the monitored metrics into a set of linguistic variables using triangular Fuzzy sets. We ran several experiments to accurately construct the Fuzzy sets and rules. For instance, if the residual energy is "low" and the number of sensors is "high", GaaS switches to a scheduler which favors energy over throughput. To facilitate the development of Fuzzy sets and rules on Android, a Java library for Fuzzy control language, referred to as `jFuzzyLogic` [14], has been integrated into our software architecture.

IV. Priority-Based Scheduling

In this section, we explain our proposed priority-based scheduling algorithms. In fact, GaaS has been designed to simultaneously connect to several IoT devices. To this end, it is necessary to construct a schedule which clearly defines the connection duration for each downlink between GaaS and the IoT sensors. To this end, we harness two well-known scheduling algorithms, namely *exhaustive polling* (EP) and *fair exhaustive polling* (FEP) [15]. Aside from the duration

of each connection, a question arises about the order of BLE connections that optimizes relevant quality metrics, such as the energy consumption on the mobile devices and the amount of retrieved sensory data (aka throughput). To determine the downlink priorities, GaaS ranks the detected IoT sensors according to the cost and efficiency of each BLE connection. Specifically, the IoT sensors having the least connection cost and highest connection efficiency will be given a higher priority. In this context, GaaS examines two different multi-criteria decision making algorithms for ranking the IoT sensors, including *weighted sum model* (WSM) and *analytical hierarchy process* (AHP). In both algorithms, we consider three decision criteria, viz. the received signal strength indicator (RSSI) of the BLE connection, the mobile device's residual energy, and the state of the IoT sensor (i.e. whether it is active and has data to upload). Below, we explain two priority-based schedulers, namely the FEP-AHP scheduler and the EP-WSM scheduler.

A. FEP-AHP Scheduler

The core idea behind the FEP-AHP is to poll the IoT sensors which has no data to report as rarely as possible. Specifically, the FEP-AHP algorithm classifies the IoT sensors as either *active* or *inactive*. Figure 5 demonstrates the main steps of the FEP-AHP scheduler. In the first cycle, GaaS annotates all the detected IoT sensors as active, before polling all sensors in a round robin fashion (line 5). The connection time T_{con} is evenly-distributed between the IoT sensors (line 11). Based on the feedback from the sensors, they are moved to the inactive state if they have no sensory data to upload (line 14). In fact, excluding the inactive sensors enables providing longer connection time T_{con} to the other sensors (line 17). The polling iteratively continues till the list of active sensors are emptied. To sidestep favoring the sensors generating packets at maximum rate, the FEP-AHP algorithm deliberately limits the polling interval to a priori-defined time t_p. Once the polling interval timer expires, the inactive sensors are turned into active state, thus allowing the gateway to poll them in the next cycles (line 4).

Once the list of detected IoT sensors $\mathcal{S}_{unsort}$ is recorded in the node repository. The FEP-AHP scheduler descendingly sorts the list through determining the priority of each IoT sensor (line 10). To this end, we develop a hierarchical structure with an objective at the top, criteria $\beta_1, \cdots, \beta_n$ at the second level, and a set of alternatives at the third level. As depicted in Figure 6, our main objective is to order the BLE connections according to the priority of each IoT sensor. To this end, GaaS considers three main criteria, including the RSSI values, the sensor state, and the residual energy of the mobile devices. Whereas, the alternatives represent, in our model, the nearby IoT sensors detected in the scanning phase. Subsequently, a pair-wise comparison matrix M_{ahp} is constructed to estimate the significance w_i of the adopted criteria relative to the main objective (line 8). Equation 1 defines a comparison matrix where each entry a_{ij} represents the ratio between the corresponding criteria, i.e. $a_{ij} = w_i/w_j$. In this context, we employ a scale of relative importance

Require: scanning time T_s, polling interval t_p

1: **Compute** matrices M_{RSSI}, M_{state}, and M_{energy} ▷ Equation 1
2: **for all** cycle $c_i \in \mathcal{C}$ **do**
3: $\mathcal{S}_{unsort} \leftarrow$ *ScanSensors*(T_s)
4: **if** $c_i == c_1$ **or** t_p is reached **then**
5: **Annotate** all sensors in $\mathcal{S}_{unsort}$ as *active*
6: **end if**
7: **for all** $s_i \in \mathcal{S}_{unsort}$ **do** ▷ AHP model
8: $K_{ahp} \leftarrow$ *weight*($s_i, M_{RSSI}, M_{state}, M_{energy}$)
9: **end for**
10: $\mathcal{S}_{sort} \leftarrow$ *SortAHP*($\mathcal{S}_{unsort}, K_{ahp}$)
11: $T_{con} \leftarrow \frac{T_{cycle}}{|\mathcal{S}_{sort}|}$ ▷ Distributing connection time evenly
12: **for all** sensors $s_i \in \mathcal{S}_{sort}$ **do**
13: **if** sensoryData(s_i) == Null **then**
14: **Annotate** s_i as *inactive*
15: **end if**
16: **end for**
17: $T_{con} \leftarrow \frac{T_{cycle}}{|\mathcal{S}_{sort}| - |\mathcal{S}_{inactive}|}$ ▷ Excluding inactive sensors
18: **end for**

Fig. 5: Priority-based FEP-AHP scheduling algorithm

ranging from equal importance, i.e. $w_i = 1$, to extreme importance, i.e. $w_i = 9$.

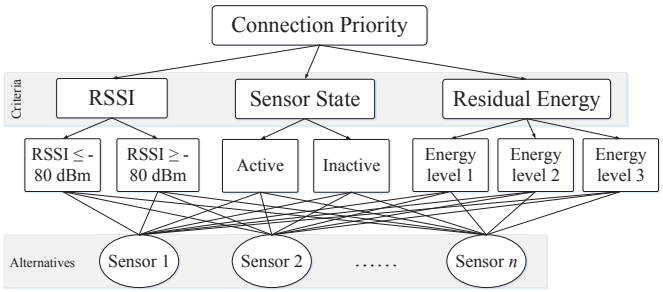


Fig. 6: AHP hierarchical structure

$$M_{ahp} = \begin{array}{c} \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{array} \begin{array}{c} \begin{array}{cccc} \beta_1 & \beta_2 & \cdots & \beta_n \end{array} \\ \begin{pmatrix} w_1/w_1 & w_1/w_2 & \cdots & w_1/w_n \\ w_2/w_1 & w_2/w_2 & \cdots & w_2/w_n \\ \vdots & \vdots & & \vdots \\ w_n/w_1 & w_n/w_2 & \cdots & w_n/w_n \end{pmatrix} \end{array} \quad (1)$$

Once the matrix M_{ahp} has been constructed, we compute the priority vector through determining the normalized eigenvector of the pair-wise comparison matrix, as expressed in Equation 2, where k is the priority vector and W is the vector of weights [16]. It is worth mentioning that the sub-criteria, e.g. RSSI ≤ -80dBm and energy levels, are also considered while estimating the final priority of each IoT sensor. For example, sensor-A has RSSI value of -90 dBm, device state is inactive and connection power consumption of 300 mW. Meanwhile, sensor-B has RSSI value of -50 dBm, device

state is active and connection power consumption of 90 mW. According to Equation 2, the AHP weights of sensor-A and sensor-B are circa 29% and 64%, respectively. Accordingly, sensor-B has higher priority than sensor-A, thus GaaS connects first to sensor-B.

$$M_{ahp} \cdot W = k \cdot W, \quad W = (w_1, w_2, \cdots, w_n)^T \tag{2}$$

B. EP-WSM Scheduler

Although being precise in sorting the IoT sensors, FEP-AHP is relatively complex owing to the various matrix computations (cf. evaluations in Section V). Therefore, we investigate, in this section, the EP-WSM scheduler as a lightweight priority-based scheduling algorithm. The core idea behind the EP-WSM algorithm is to poll the detected IoT devices in a consecutive order. GaaS continuously polls the first IoT sensor in a list of detected sensors till either retrieving all sensory data or the sensor becomes disconnected. Similarly, other IoT sensors are polled till covering all detected sensors. To avoid favoring the IoT sensors generating packets at higher rates, GaaS employs for each BLE connection a predefined time constraint t_{ep}. Accordingly, a BLE connection is deliberately terminated when either the data is completely retrieved or the connection timer, adjusted to t_{ep}, expires. Figure 7 depicts the various steps of the priority-based EP-WSM scheduling algorithm. In fact, BLE scanning is a major factor of the energy overhead on the mobile devices hosting the cross-platform gateway. To reduce the scanning time T_s, GaaS records the detected IoT sensors in the first cycle of the schedule, thus requiring less BLE scanning time in the subsequent cycles. Specifically, GaaS adopts, at the first cycle, an initial scanning time $T_s^{in} = \alpha$ (line 3). The detected IoT sensors, during this cycle, are recorded in the node repository. In the subsequent cycles, the gateway directly connects to the recorded sensors from the previous cycle. Therefore, a reduced scanning time $T_s^{red} = k \times \alpha, k \in [0, 1]$ can be adopted to solely detect the newly-arrived IoT sensors (line 6).

After recording the IoT sensors in the node repository, GaaS estimates the weighted sum score K_{wsm} to descendingly sort the list of sensors (lines 10-13). Equation 3 expresses the weighted sum score in case of n criteria and m alternatives, i.e. IoT sensors, where a_{ij} is the performance value of the i-th alternative with respect to the j-th criterion and w_j is the relative weight of importance of the j-th criterion. To express all criteria in exactly the same unit, we map the various criterion onto a scale between one and five, where the performance values of residual energy a_{i1}, device state a_{i2}, and RSSI a_{i3} are set—according to their significance—to five, three, and one, respectively. Once the list of sensors are sorted, GaaS then performs *synchronized* connections to the sensors starting with the one having the highest priority (line 15). The term "synchronized" emphasizes that the connection is accessible only by one thread to avoid the multithreading problems. Once a BLE connection is established, data retrieval continues till the timer t_{ep} expires or no further data exists (line 18).

Require: initial scanning time T_s^{in}, reduced scanning time T_s^{red}

1: **for all** cycle $c_i \in \mathcal{C}$ **do**
2: **if** $c_i == c_1$ **then**
3: $\mathcal{S}_{unsort} \leftarrow$ *ScanSensors*($T_s = T_s^{in}$)
4: **Record** $\mathcal{S}_{unsort}$ in the *BLE device data* database
5: **else**
6: $\mathcal{S}_{unsort} \leftarrow$ *ScanSensors*($T_s = T_s^{red}$)
7: **Update** $\mathcal{S}_{unsort}$ in the *BLE device data* database
8: **end if**
9: **for all** $s_i \in \mathcal{S}_{unsort}$ **do** ▷ WSM model
10: $A_{s_i} \leftarrow a_{i1}\omega_1 + a_{i2}\omega_2 + a_{i3}\omega_3$
11: **end for**
12: $k_{wsm} \leftarrow$ *maximum*(A_{s_i}), $i = 1, 2, 3, \cdots, m$
13: $\mathcal{S}_{sort} \leftarrow$ *SortWSM*$(\mathcal{S}_{unsort}, k_{wsm})$
14: **for all** $s_i \in \mathcal{S}_{sort}$ **do**
15: connection $d \leftarrow$ *synchronizedConnect*(s_i)
16: d.lock() ▷ keep the connection alive
17: **if** sensoryData(s_i) ! $= Null$ & t_{ep} is not reached **then**
18: **Record** sensoryData(s_i) in the *data upload* database
19: **end if**
20: d.unlock(s_i) ▷ kill the connection
21: **end for**
22: **end for**

Fig. 7: Priority-based EP-WSM scheduling algorithm

$$K_{wsm} = \max \sum_{j=1}^{n} a_{ij} \times \omega_j, \quad \text{for } i = 1, 2, 3, \cdots, m \tag{3}$$

V. Performance Evaluation

To demonstrate the effectiveness of GaaS, we tested our system in a real-world scenario. We first describe the setup of our evaluation, before we discuss the evaluation results. The performance is assessed in terms of the energy consumption of the mobile devices. Furthermore, we evaluate the throughput and latency of data collection from the various IoT devices.

A. Experimental Setup

To test our implementation of the cross-platform gateway, we designed a testbed composed of six temperature sensors connected to Faros BLE beacons and four commercial BLE-supported IoT devices, including a smart meter, a temperature sensor, a cycling speed sensor, and wearable Miband device (cf. Figure 8). Figure 8a depicts a low-cost and low-power Faros Board equipped with an Nordic nRF8001 BLE module. Such a board has specifically been designed for BLE devices operating as peripheral/slave nodes. The sensory data are stored as characteristics according to the GATT specification. In Faros board, data transmission is done using BLE's ATT protocol[1]. Similarly, the wearable MiBand device measures several phenomena, e.g. heart rate and steps count. In addition

[1]More details about the implementation of the Faros BLE beacon can be found under the following link: https://github.com/duerrfk/Faros

to these sensors, we also employed emulated sensors using *nRF Connect* built by Nordic Semiconductor [17]. The *nRF Connect* App—installed on LG Nexus 5X device—advertises BLE beacons and provides synthetic data embedded into the services and characteristics of the BLE connection mode.

(a) Faros Board with BLE Module (b) Commercial IoT devices

Fig. 8: IoT devices used in the evaluations

For implementing the cross-platform gateway, we have utilized a Sony Xperia Z5 mobile device with 2GB of RAM and 32GB of internal storage [9]. For BLE wireless communication, GaaS adopts a star topology where the mobile device—hosting the cross-platform gateway—acts as a master and the nearby IoT sensors are the slaves. such a topology has been selected to enable the mobile device from concurrently connecting to several IoT sensors. As a cloud service, we adopted Google Firebase, which provides several services for database post-processing, such as cloud storage, real-time database, and back-end service, and Google Analytic.

To measure the power consumption of our mobile gateway, the Trepn Profiler—a third party application built by Qualcomm [18]—has been used. Each run of these measurements were repeated ten times and the resultant values have been averaged. Furthermore, the Android Bluetooth Host Controller Interface (HCI) Snoop Log protocol has been utilized to provide a log of all Bluetooth HCI packets captured by the Android system. Wireshark [19] is then used to analyze the log data output from the Bluetooth HCI Snoop Log in the Android system for the sake of estimating the number of BLE packets sent and received by the mobile device acting as a mobile gateway.

B. Gateway Configurations

In this set of experiments, we examine various configurations of the cross-platform gateway together with assessing the impact of ranking the IoT sensors before generating a schedule. We selected to dynamically change the number of IoT sensors during the experiments through adding two IoT sensors every two minutes. Figure 9a depicts the impact of changing the scheduling cycle t_u—defined as the duration of a single round in which the gateway connects and retrieves data from all available IoT sensors. In this experiment, we adopted the FEP-AHP scheduler to examine its performance for different values of the scheduling cycle t_u. As the figure depicts, increasing the cycle t_u from 0.5 minute to one minutes leads to a slight reduction in the consumed power (at most by 10%). Clearly, power consumption is increased as the number of switching between the IoT sensors is also increased. Therefore, we selected $t_u = 1$ minute as a reasonable period while generating the various schedules in our next experiments. Similarly, changing the self-adaptation period t_v—defined as the time between successive executions of the self-adaptation engine—has a slight influence on the achieved throughput. Again, we set t_v to one minute while executing the self-adaptive engine in our next experiments.

Figure 9c shows a comparison between the standard FEP scheduler and the proposed priority-based schedulers. The main intuition here is to assess the impact of ranking the IoT sensors before generating a connection schedule. As the figure depicts, the FEP-WSM scheduler consumes, on average, 16% less energy than the standard FEP scheduler. Similarly, it achieves, on average, 17% higher throughput than the FEP scheduler. It is worth mentioning that the relatively loose distribution occurs owing to dynamically changing the number of IoT sensors during the experiment. Accordingly, we can conclude that ranking the IoT sensors broadly improves the relevant QoS metrics.

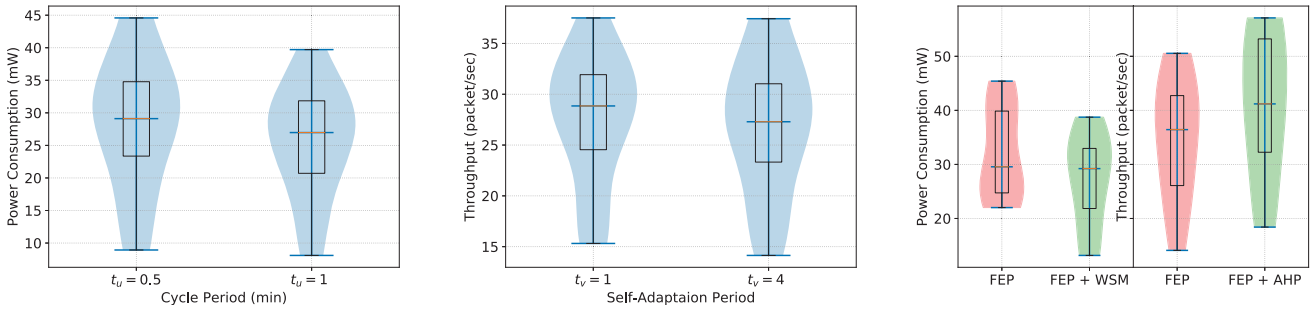
C. Priority-Based Schedulers

In this set of experiments, we aim at assessing the performance of the proposed priority-based schedulers in terms of the energy overhead and the amount of retrieved data, i.e. throughput. In addition to the EP-WSM and FEP-AHP schdulers, we also implemented two other variants, including the EP-AHP and FEP-WSM schedulers. Figure 9 demonstrates a comparison between the various schedulers in terms of the energy overhead for different number of connected IoT devices. As shown in Figure 10a, the FEP scheduler consumes less power than the EP scheduler (at least by 57%), regardless of the adopted ranking method. Such energy savings occur thanks to excluding the inactive sensors, thus allowing data retrieval from other sensors in shorter time. Through comparing Figures 10a-10c, it is obvious that the power consumption increases as the number of connected sensors n is increased. For instance, FEP-AHP consumes more energy (at most by 33%) when n is increased from two to ten sensors.

Similarly, Figure 11 illustrates the data retrieval rate of GaaS for different number of connected IoT sensors. For $n = 2$, the EP-WSM drastically outperforms the FEP-AHP scheduler where it achieves higher throughput (at least by 46%). Such high throughput of EP-WSM occurs thanks to retrieving all the data in each sensor before switching to the next sensor, thus collecting as much data as possible. Similar performance of the EP-WSM scheduler is achieved if n is increased to ten sensors (cf. Figures 11b and 11c). Based on the obtained results in Figures 10 and 11, it is clear that there exists a trade-off between the power consumption and throughput. Therefore, it is crucial to implement the self-adaptive engine to switch between the FEP-AHP and EP-WSM schedulers according to the application dynamics i.e. number of sensors and residual energy.

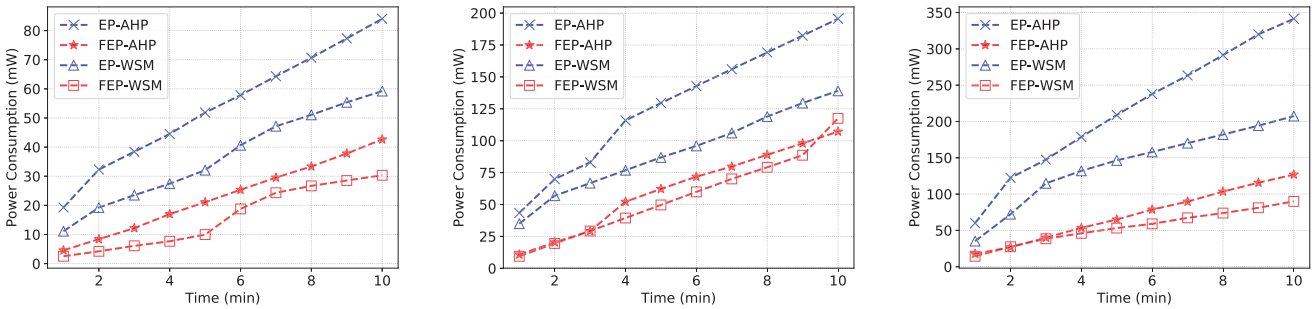
D. GaaS vs. Siloed Gateways

In this section, we compare the performance of GaaS and the siloed gateways in terms of the power consumption and



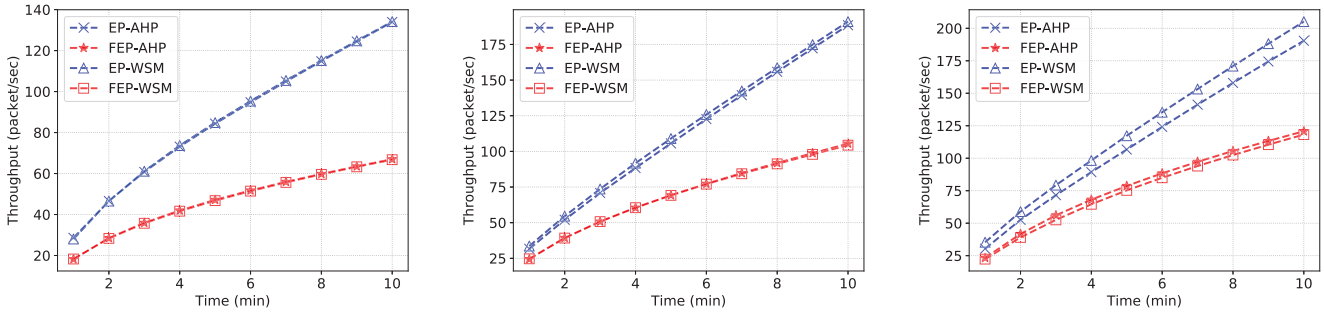
(a) Effect of changing the scheduling cycle (b) Effect of changing the self-adaptation cycle (c) Effect of ranking the IoT devices

Fig. 9: Comparing various configurations of GaaS



(a) $n = 2$ (b) $n = 6$ (c) $n = 10$

Fig. 10: Cumulative power consumption of GaaS for different numbers of the connected IoT devices



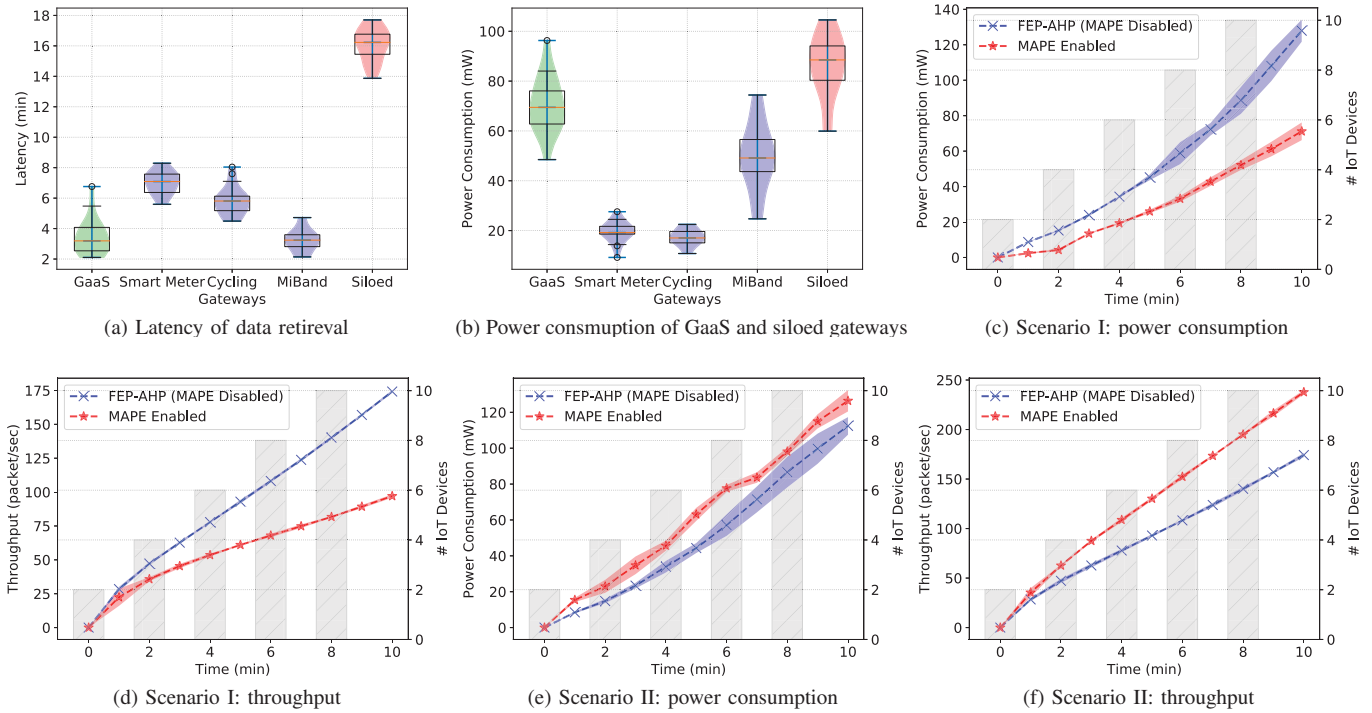
(a) Cumulative throughput of GaaS: $n = 2$ (b) Cumulative throughput of GaaS: $n = 6$ (c) Cumulative throughput of GaaS: $n = 10$

Fig. 11: Cumulative throughput of GaaS for different numbers of the connected IoT devices

the data retrieval latency. In this experiment, we utilized three gateways corresponding to the smart meter, the cycling (speed) sensor, and the MiBand device. For a fair comparison, we selected to retrieve 30 KB of sensory data using GaaS, while each siloed gateway collects only 10 KB of data, i.e. heart rate in case of the MiBand device. Accordingly, the power consumption and latency are measured for collecting 30 KB of data in each method. For GaaS, the FEP-AHP scheduler has been adopted while deactivating the self-adaptation engine. Figure 12a depicts the latency of data retrieval in case of GaaS and the three individual gateways, where the last entry "siloed" sums up the latencies of the individual gateways. Obviously, GaaS requires much less time (at least by 78%) to retrieve 30 KB of data from three IoT sensors. For GaaS, the distribution is highly tight where the maximum latency does not exceed circa 6.5 minutes compared to circa 17.6 minutes for the siloed gateways. Similarly, Figure 12b shows a comparison of the power consumption between GaaS and the siloed gateways. Apparently, GaaS consumes more energy than each individual gateway. Such a behavior of the individual gateways occurs thanks to their relatively short duty cycle along with collecting less amount of data. Nevertheless, we found that GaaS outperforms the set of all siloed gateways where it consumes, on average, 18% less power than all siloed gateways together.

E. Impact of Self-Adaptation

In this section, we assess the ability of our MAPE-based self-adaptation engine in improving energy efficiency and throughput of the proposed cross-platform gateway. In this set of experiments, we consider for the EP-WSM scheduler two



(a) Latency of data retireval (b) Power consmuption of GaaS and siloed gateways (c) Scenario I: power consumption

(d) Scenario I: throughput (e) Scenario II: power consumption (f) Scenario II: throughput

Fig. 12: Comparative study: GaaS against the siloed gateways and GaaS with/without the MAPE self-adaptation engine

distinct scenarios: *Scenario I* in which power consumption is dominating, i.e given larger weight than throughput, and *Scenario II* in which throughput is dominating the decision making process. At the beginning of each run, GaaS is initially connected to two sensors, where two additional sensors are added every two minutes. For Scenario I, Figure 12c shows the power consumption of GaaS with and without enabling the self-adaptation engine. Apparently, enabling the self-adaptation engine results in consuming, on average, 44% less energy. Furthermore, the figure shows that the difference between these two cases increases as the number of connected IoT sensors is increased from two to ten sensors. As expected, we find in Figure 12d that the self-adaptation engine achieves, on average, 38% less throughput than the baseline method.

For Scenario II, Figure 12e depicts the power consumption of GaaS with and without activating the MAPE-based self-adaptation engine. The figure shows that enabling the MAPE engine leads to consuming, on average, 19% more energy than the baseline method. Conversely, Figure 12f demonstrates that the MAPE engine achieves, on average, 28% higher throughput than the baseline method. Such a behavior of the self-adaptation engine proves that it can successfully react to the application dynamics, i.e. the residual energy and the number of detected IoT sensors, through favoring either energy consumption or throughput.

VI. Related Work

In this section, we review the most salient related work in the realm of opportunistic gateways with highlighting the novelty of our proposed solution. The problem of designing gateways for IoT applications has been tackled in several research work . For instance, Kang et al. [20] introduce a self-configurable *fixed* IoT gateway for the large-scale IoT environment. Such a gateway can automatically identify and connect to the nearby IoT sensors. These capabilities are already provided in GaaS through the BLE advertising mechanism. However, GaaS has the advantage of exploiting the mobile devices to retrieve data from a wide spectrum of IoT sensors in different locations. Aloi et al. [6] introduce a software architecture of smartphone-centric gateways to support the interoperability between different communication technologies. This architecture allows bi-directional information exchange between smartphones and the surrounding IoT sensors. Although the architecture does not consume excessive CPU power and memory space, it significantly increase the energy overhead on the host smartphones owing to simultaneously activating several radio interfaces and periodically scanning the radio channels to discover the surrounding IoT sensors. Alternatively, GaaS leverages BLE interfaces thanks to their widespreadness and their energy efficiency. Furthermore, GaaS considers energy efficiency through incorporating a self-adaptation engine.

Along a similar line, Zhalgasbekova et al. [5] introduce CollMule, an on-demand data collection mechanism which harnesses smartphones for aggregating air quality data from a set of IoT sensing devices. The core idea behind CollMule is to reduce the energy consumption of the smartphones that act as gateways through avoiding unnecessary connections

to unreliable IoT devices. Aguilar et al. [22] investigate the performance of BLE as a promising technology for opportunistic data collection. They consider two scenarios for data collection, including (1) piggybacking the sensor data in the advertisement phase and (2) data collection in the connection phase. Can et al. [4] investigate the feasibility of data collection by smartphones in the presence of rarely-visited regions through analyzing the GPS trajectories of the mobile devices. Furthermore, several duty cycling schedules were adopted to conserve energy on smartphones and on the sensor nodes. As alternative to these approaches, GaaS automatically discovers and connects to various IoT sensors at the same time thanks to its priority-based scheduling.

From the side of the IoT sensors, Wu et al. [21] devise a distributed algorithm which improves the network throughput via preforwarding the sensor data to other nodes that are often visited by smartphones. The preforwarding task has been formulated as an optimization problem that maximizes the network throughput. To solve this problem, two heuristic algorithms have been proposed that generate the data preforwarding plan based on the states of the direct neighboring nodes and on the mobility patterns of the available smartphones. We strongly believe that this approach can be readily augmented with GaaS to construct a highly reliable IoT application. In general, data collection from IoT sensors can also be performed using opportunistic communication in vehicular mobile networks. For instance, Tang et al. [23] present SWDCP-SCmules, a data collection framework which enables a set of mobile data mules, such as taxis, buses and pedestrians holding smartphones, from collecting sensory data, before forwarding them to data centers.

VII. Conclusion & Future Work

In this paper, we presented GaaS, a cross-platform gateway architecture for retrieving the sensory data from the low-power IoT sensors. The proposed architecture leverages multithreading to enable the gateway from simultaneously connecting to several IoT sensors. To regulate the BLE connections, we devised two distinct priority-based schedulers which rank the IoT sensors, before generating the connection schedule. Moreover, we developed a self-adaptation engine to regularly modify the gateway's configurations in accordance with the run-time dynamics. We demonstrated the efficiency of GaaS through carrying out several experiments on our IoT testbed. The results showed that GaaS reduces the energy overhead on the mobile devices together with reducing the amount of time required to retrieve data from the IoT sensors. In the future, we plan to further improve the energy efficiency of the mobile devices through reducing the duty cycle of GaaS. To this end, we may explore the potential of machine learning algorithms in precisely predicting the locations where IoT sensors most probably exist.

References

[1] L. Da Xu, W. He, and S. Li, "Internet of Things in Industries: A Survey," *IEEE Transactions on industrial informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.

[2] T. Zachariah, N. Klugman *et al.*, "The internet of things has a gateway problem," in *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, ser. HotMobile '15. ACM, 2015.

[3] "Ericsson Mobility Report," Ericsson Telecommunications Company, Tech. Rep., 11 2016. [Online]. Available: https://www.ericsson.com/assets/local/mobility-report/documents/2016/ericsson-mobility-report-november-2016.pdf

[4] Z. Can and M. Demirbas, "Smartphone-based data collection from wireless sensor networks in an urban environment," *Journal of Network and Computer Applications*, vol. 58, pp. 208 – 216, 2015.

[5] A. Zhalgasbekova, A. Zaslavsky, and S. Saguna, *Opportunistic Data Collection for IoT-Based Indoor Air Quality Monitoring*. Springer International Publishing, 2017, pp. 53–65.

[6] G. Aloi, G. Caliciuri, G. Fortino, R. Gravina, P. Pace, W. Russo, and C. Savaglio, "Enabling iot interoperability through opportunistic smartphone-based mobile gateways," *Journal of Network and Computer Applications*, vol. 81, pp. 74–84, 2017.

[7] Xiaomi, "MiBand 2," https://www.mi.com/en/miband2/, [Accessed : July, 2018].

[8] Fitbit, "Fitbit Devices," https://www.fitbit.com/es/home, [Accessed : July, 2018].

[9] S. Developers, "Xperia z5 dual," Sony Mobile Communications Inc., White paper, oct 2017, accessed August 2018.

[10] Y. Brun, G. D. M. Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw, "Engineering self-adaptive systems through feedback loops," in *Software engineering for self-adaptive systems*. Springer, 2009, pp. 48–70.

[11] A. Rangone, "An analytical hierarchy process framework for comparing the overall performance of manufacturing departments," *International Journal of Operations & Production Management*, vol. 16, no. 8, pp. 104–119, 1996.

[12] A. Kolios, V. Mytilinou, E. Lozano-Minguez, and K. Salonitis, "A comparative study of multiple-criteria decision-making methods under stochastic inputs," *Energies*, vol. 9, no. 7, p. 566, 2016.

[13] Google, "Android Optimize for Battery Life," https://developer.android.com/topic/performance/power/, [Accessed : July, 2018].

[14] P. Cingolani and J. Alcala-Fdez, "jfuzzylogic: a robust and flexible fuzzy-logic inference system language implementation," in *2012 IEEE International Conference on Fuzzy Systems*. IEEE, 2012, pp. 1–8.

[15] N. Johansson, U. Körner, and P. Johansson, "Performance evaluation of scheduling algorithms for bluetooth," in *Broadband communications*. Springer, 2000, pp. 139–150.

[16] T. L. Saaty, "How to make a decision: the analytic hierarchy process," *European journal of operational research*, vol. 48, no. 1, pp. 9–26, 1990.

[17] Nordic-Semiconductor, "Bluetooth Low Energy nRF8001," https://www.nordicsemi.com/eng/Products/Bluetooth-low-energy/nRF8001, [Accessed : July, 2018].

[18] Qualcomm, "Qualcomm Trepn Power Profiler," https://developer.qualcomm.com/software/trepn-power-profiler, [Accessed : July, 2018].

[19] Wireshark, "Wireshark Network Protocol Analyzer," https://www.wireshark.org/, [Accessed : July, 2018].

[20] B. Kang, D. Kim, and H. Choo, "Internet of everything: A large-scale autonomic iot gateway," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 3, no. 3, pp. 206–214, July 2017.

[21] X. Wu, K. N. Brown, and C. J. Sreenan, "Data pre-forwarding for opportunistic data collection in wireless sensor networks," *ACM Trans. Sen. Netw.*, vol. 11, no. 1, pp. 8:1–8:33, Jul. 2014. [Online]. Available: http://doi.acm.org/10.1145/2629369

[22] S. Aguilar, R. Vidal, and C. Gomez, "Opportunistic sensor data collection with bluetooth low energy," *Sensors*, vol. 17, no. 1, p. 159, 2017.

[23] Z. Tang, A. Liu, and C. Huang, "Social-aware data collection scheme through opportunistic communication in vehicular mobile networks," *IEEE Access*, vol. 4, pp. 6480–6502, 2016.

[24] M. Bonola, L. Bracciale *et al.*, "Opportunistic communication in smart city: Experimental insight with small-scale taxi fleets as data carriers," *Ad Hoc Networks*, vol. 43, pp. 43–55, 2016.

[25] H. Nishiyama, M. Ito, and N. Kato, "Relay-by-smartphone: Realizing multihop device-to-device communications," *IEEE Communications Magazine*, vol. 52, no. 4, pp. 56–65, April 2014.

[26] Z. Lu, G. Cao, and T. L. Porta, "Teamphone: Networking smartphones for disaster recovery," *IEEE Transactions on Mobile Computing*, vol. 16, no. 12, pp. 3554–3567, Dec 2017.