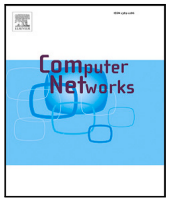
Computer Networks 222 (2023) 109560

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



DynGATT: A dynamic GATT-based data synchronization protocol for BLE networks

Christian Hirsch [a,1], Luca Davoli [b,*], Radu Grosu [c], Gianluigi Ferrari [b]

[a] Independent Researcher, Wien, Austria
[b] Internet of Things (IoT) Laboratory, Department of Engineering and Architecture, University of Parma, Parma, Italy
[c] Institute of Computer Engineering, Vienna University of Technology, Wien, Austria

ARTICLE INFO

Keywords:
Internet of Things (IoT)
Bluetooth Low Energy (BLE)
Generic ATTribute (GATT)
Internet Gateway
Dynamic network
Smart agriculture

ABSTRACT

Bluetooth Low Energy (BLE) is a wireless communication technology for power-constrained Internet of Things (IoT) applications. BLE data can be transmitted via either the IPv6 or the Generic ATTribute (GATT) Profile protocol, with the former supporting dynamic IoT structures and the latter being application-friendly. In fact, GATT requires the data layout to be known in advance by peer devices, in order to properly interpret the received data. In this paper, we introduce DynGATT, a protocol that achieves the benefits of both IPv6 and GATT, by extending GATT in a seamless fashion to support dynamic IoT structures. The key idea of DynGATT is to use GATT descriptors, originally intended to specify data in static IoT scenarios, to also specify IoT systems whose structures may dynamically evolve. Peer devices reading these descriptors will know how to interpret the data of GATT characteristics provided by devices joining the IoT network. Because no additional data have to be transmitted, the connection time is then reduced with respect to classical BLE. DynGATT has been implemented and tested in an agricultural IoT application, with different types of sensor nodes. Our experimental evaluation shows that DynGATT is very power-efficient, despite its added flexibility. Its worst-case power consumption is only around 19.37 μA per data transmission and around 41.37 μA overall. This consumption can be further reduced by using the methods discussed in this paper. To the best of our knowledge, this work is the first to support dynamic IoT structures in a GATT-based setting.

1. Introduction

In recent years, the Internet of Things (IoT) has become increasingly popular as a way of connecting in a unified fashion different kinds of sensors and actuators, for different types of applications, from smart city scenarios and Industry 4.0, to smart agriculture and smart health-care, just to name a few. In order to create a unified network, all different things, also denoted as smart objects (SOs), need to be connected. As in many IoT applications the provision of cable-based power and network access may be challenging, SOs thus need to be battery-powered and rely on wireless communications. One technology providing wireless data transmission and holding the promise to save power is BLE.

In detail, BLE is a wireless communication technology introduced by the Bluetooth Special Interest Group (SIG) with the Bluetooth Core Specification 4.0 in 2010 [1], with the goal to save power. With BLE, data can be transmitted in two possible ways: (a) *connectionless*, via advertising and scanning, or (b) *connection-based*, via data channels. In the *first* case, on one side, a device, denoted as *broadcaster*, periodically broadcasts information, while, on the other side, another device, denoted as *observer*, scans for the advertised packets to receive data. Once an observer receives advertisement data during its scanning period, it knows that a broadcaster is within its connection range and can potentially interact with itself. Hence, this leads to a connectionless communication and, thus, to the most commonly one-way data transmission. On the other end, the *second* approach establishes a connection between the two devices and, afterwards, relies on this connection to exchange data in both ways.

One major advantage of BLE is its availability in many devices on the market, especially in the Commercial Off-The-Shelf (COTS) consumer area—e.g., modern laptops and PCs, smartphones, as well as many Single Board Computers (SBCs) used for IoT applications, are equipped with this communication technology. Hence, no additional hardware is necessary to communicate with other devices, such as sensors, actuators or beacons, by using BLE.

* Corresponding author.
E-mail addresses: christian@hirsch.zone (C. Hirsch), luca.davoli@unipr.it (L. Davoli), radu.grosu@tuwien.ac.at (R. Grosu), gianluigi.ferrari@unipr.it (G. Ferrari).

[1] He was with the Institute of Computer Engineering of the Vienna University of Technology when contributing to this work.

https://doi.org/10.1016/j.comnet.2023.109560
Received 17 July 2022; Received in revised form 3 December 2022; Accepted 3 January 2023
Available online 10 January 2023

With respect to competing technologies, such as IEEE 802.11 (Wi-Fi), Long Range (LoRa) protocol, or ZigBee, BLE uses less power to transmit data [2,3]. However, it does not implement an Internet Protocol (IP) stack in its core, which makes it difficult for Internet applications to rely on BLE. Although IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) allows to support the IP version 6 (IPv6) protocol stack over constrained (IEEE 802.15.4-based) devices [4], the majority of devices implementing BLE communicate relying on the GATT Profile stack, which is designed to support a Device-to-Device (D2D) communication with a master, thus forming a star topology. While the 6LoWPAN implementation is flexible, as the device can create its data stream and send it to any remote device connected to the Internet, the GATT stack is more stringent, as the peer device needs to know how to interpret the data. An example is given by a smart watch connecting to a person's smartphone to exchange data. This watch is paired to the user's smartphone and is only meant to be connected to this device to exchange data. The app running on the smartphone knows which GATT services and characteristics to connect to, so that it can receive data and interpret them correctly. For different sensors, applications need to be reconfigured or new specialized apps need to be implemented, since one needs to know exactly how to interpret the transmitted data.

For certain types of applications, power consumption is critical and, therefore, data transfer shall be reduced. For example, consider a farming scenario with different sensors, all battery-powered, that either measure environmental impact, track vehicle data, or observe a livestock's health. In such a scenario, the GWs need to know all possible sensor nodes, in order to receive, interpret and transmit the data. Unfortunately, this approach cannot be generalized, since every new sensor node added to the infrastructure requires a reconfiguration of the system itself. In such a dynamic scenario, where livestock and vehicles are involved, this is even trickier. Consider a barn that is equipped with GWs that can connect to such a sensor node or to a tracking device. In this case, it might happen that, at some point, the livestock or vehicle moves out of the GW range. Consequently, the sensor nodes and trackers need to create timestamped data, store them locally temporarily, and synchronize them later, as soon as they are within the transmission range of the GW.

The main contributions of this paper are the design and development of a GATT-based synchronization protocol, denoted as *DynGATT*. Our starting point is to abide by BLE specifications, in order to easily implement it on any BLE-powered device – such as smartphones, PCs, and microcontrollers – and targeting IoT-oriented applications, thus reducing the number of interactions needed to decrease the power consumption. In order to do so, we *first* introduce the infrastructure needed and the challenges faced in gathering agricultural information. *Second*, we propose an approach to overcome these challenges by designing a protocol for an IoT scenario with BLE-powered sensor nodes and GWs. *Last*, we describe the implementation of our IoT-oriented data synchronization protocol. In particular, we develop: a SSGW application to connect BLE-powered devices to the Internet; a GATT descriptor-based characteristics paradigm to interpret the data of the BLE-powered devices.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 provides an overview on the proposed IoT-oriented infrastructure. Section 4 describes the sensor identification and device discovery as part of the SSGW application, while Section 5 describes the proposed synchronization procedure and the corresponding protocol. Section 6 describes the experimental setup considered to evaluate the proposed approach and Section 7 is dedicated to experimental performance analysis. Finally, Section 8 presents our conclusions and discusses future work directions.

2. Related work

A dynamic data synchronization requires, as a first step, a self-configuring and self-managing paradigm for autonomous discovery of services, providing data to the different IoT devices belonging to the involved network(s). To this end, discovery mechanisms have to be defined, allowing interested clients to discover service providers in contexts in which neither one knows the existence of the other. This requires: (a) the definition of a language for describing the services and their selection criteria; (b) a common protocol for exchanging the descriptions; and (c) rules for matching criteria with service descriptions. Components (a–c) have to be managed in an efficient way and in challenging scenarios, such as those of IoT contexts [5,6]. To this end, discovery paradigms should consider proactive service advertisement, in order to ensure and exploit interoperability among heterogeneous IoT devices, taking into account their dynamic presence in a network – in topological terms – over a certain time interval.

Examples of discovery mechanisms that can be adopted in IoT scenarios are Multicast Domain Name System (mDNS) with Service Discovery (mDNS-SD) [7] (often adopted jointly with another protocol, denoted as ZeroConf [8]) and the Constrained RESTful Environments (CoRE) Link Format [9]. In detail, the former mechanism exploits the DNS interaction mechanism to resolve host names in networks that do not include a local name server [10], while the latter mechanism is specifically designed to be used jointly with the Constrained Application Protocol (CoAP) [11]. Unfortunately, mDNS-SD may not be a viable solution for low-power or non-IP-compliant networks, while CoRE Link Format would require that both CoAP clients and servers *a priori* know a common lexicon. Then, another issue regarding heterogeneous networks may be related to their natural fragmentation—sometimes, there is the need to discover services and data hosted by several nodes belonging to separate networks.

With regard to BLE-oriented discovery mechanisms, a well-known one is iBeacon [12], proposed by Apple and enabling location-awareness features for devices in the proximity of the beacon broadcaster itself, leveraging the *advertisement* mode of the BLE protocol. Although iBeacon is widely adopted (thanks to the availability on a wide plethora of Apple devices), it is a one-way advertisement mechanism allowing only to determine when a user enters or leaves the broadcaster's proximity region. To this end, iBeacon requires Universally Unique IDentifiers (UUIDs) pre-allocation for every scenario in which it would be adopted. Moreover, devices compatible with iBeacon are, in general, always active in advertisement mode, thus not being able to selectively deactivate this operational mode.

Another BLE-based discovery paradigm is Eddystone [13], proposed by Google and providing functionalities for transparency and robustness in the communication among the parties. Moreover, being able to support different types of payload in the frame format – namely, Eddystone-UID, Eddystone-URL, Eddystone-TLM, and Eddystone-EID, each one being a good candidate to cover part of the heterogeneity of IoT scenarios – Eddystone represents a reliable discovery mechanism which a dynamic data synchronization protocol can build on. Examples of such scenarios could be smart parking management [14,15], as well as indoor navigation systems [16,17], with the goal of exploiting retrieved information for high-layer processing purposes. As this technology is versatile from an utilization point of view, it is also included in the proposed work for device discovery.

Regarding data transfer, in [18] three different ways for connecting BLE devices to the Internet, using an Internet GW, are presented. The *first* one is a smart system with a REpresentational State Transfer (RESTful) Application Programming Interface (API) Internet GW. In detail, sensor nodes act as GATT servers and the Internet GW acts as a GATT client, with the GW connecting to sensor nodes and collecting data upon request. Then, a user can access the Internet GW via a RESTful API, while the GW accesses the information via GATT communication from the sensor node [19]. Furthermore, a Generic Access Profile (GAP) RESTful API is present on the Internet GW and allows to carry out device discovery, connection establishment, and other relevant operations [20]. The *second* application is a smart system

with an HTTP proxy Internet GW. In this scenario, the sensor node acts as a GATT client, connecting to the Internet GW, which acts as a GATT server and forwards the data to a Cloud server. The *third* application, proposed in [18], is a smart system with an Internet Protocol Support Profile (IPSP) [21] border router application, where the 6LoWPAN protocol is implemented. In detail, the server implements the IPv6 stack and the Internet GW acts as a border router. Since our proposed work is based on top of GATT, the ideas at the basis of the first two examples of an Internet GW are taken into account.

Spörk et al. [22] propose a IPv6 over BLE approach which dynamically changes connection parameters to save energy. In their work, the connection parameters are tuned so that, when no data is transmitted, the connection intervals are increased to reduce power consumption. When a data stream is transmitted, however, connection intervals are shortened to transmit data faster.

Another BLE-based data transfer is proposed in [23], where data related to motion tracking of human movements are sent from slaves to a master exploiting BLE data packets. Even though experimental results represent a relevant benchmark for BLE-based data transmission, the chosen payload format lacks generality, since the receiver needs to know exactly the position of each data type in order to correctly parse the data.

With specific reference to smart agriculture scenarios and use cases, there exist various approaches based on short-range (e.g., BLE [24,25]), medium-range (e.g., IEEE 802.15.4 and IEEE 802.11 [26,27]), and long-range (e.g., cellular LTE [28,29], LTE-M [30], and 5G [31]) communication technologies. The rationale behind this array of technologies lies, as expected, on the adoption of the most suitable connectivity type for the specific scenario—e.g., depending on the farm size [32,33], the availability of an Internet access [34,35], and the need to interact with high-level cyber-physical systems (CPSs) [36] (also in a secure way [37,38]). As an example, IoT systems deployed on open fields (e.g., aiming at livestock localization) normally rely on long-range power-efficient technologies (namely, Low-Power Wide Area Networks, LPWANs), such as Sigfox (e.g., for cow geo-localization [39]), Narrowband IoT (NB-IoT) [40], and LoRaWAN [41]. Examples of developments or deployments based on short-range and medium-range connectivity are instead proposed in [42], where a Wireless Sensor Network (WSN)-based farming ecosystem, based on embedded IEEE 802.11-enabled devices and featuring a control mobile App, is presented. In [43], a low-cost soil moisture data collection architecture, based on the Ad-hoc On-demand Distance Vector (AODV) routing protocol, is discussed. In [44], an IEEE 802.15.4-based WSN, aiming at assessing a particular crop parameter, denoted as Leaf Area Index (LAI), is proposed, while the use of LoRa for monitoring stations and Wi-Fi/LTE for control stations is detailed in [45]. Finally, the adoption of BLE in heterogeneous smart farming-like contexts is discussed in [46], while an IoT platform (denoted as SheepIT), integrating sets of sensors into mobile (coupled to livestock) and static (distributed along vineyards) nodes, is presented in [47].

3. Overview of the IoT infrastructure

The reference IoT infrastructure in this work is shown in Fig. 1 and targets a smart agriculture scenario containing (a) a swarm of static BLE-enabled sensor nodes (Fig. 1a), measuring weather status and soil properties, and (b) dynamic BLE-equipped sensor nodes placed on livestock and vehicles. In order to transmit the sensed data, a Fog layer (Fig. 1b) consisting of at least one BLE-enabled GW with permanent power supply and Internet connection, is deployed, in turn allowing to connect the sensors' swarm to the IoT's Cloud infrastructure (Fig. 1c).

Each component of the IoT infrastructure has its own requirements. As the swarm of sensor nodes is deployed outdoor, with a lack of permanent power supply, the majority of sensor nodes have to run on batteries. In order to reduce manual intervention and increase system's reliability, a major focus is on power consumption, which needs to be reduced in order to increase battery duration and, as a consequence, extend operational lifetime. Since sensor nodes may be placed on mobile objects – e.g., livestock or vehicles – and may have a limited access to GWs, they need to temporarily store internally timestamped measurements. As a consequence, sensor nodes have to be equipped with a local clock, that can be exploited for this task.

Unlike battery-powered sensor nodes, GWs are powered in a permanent way and, therefore, power consumption is not an issue. Keeping in mind that most of the power is consumed by the wireless sensor nodes during data transmission, the aim of a GW is to support sensor nodes in reducing the time their wireless network interfaces need to stay active. With BLE communications, this can be done by tuning the connection parameters, which configure the amount of data packets to be transmitted and their timing. This means that the time and duration the radio module is turned on, to support an established BLE connection, can be tuned to minimize the power consumption. However, maintaining a connection alive – even with properly tuned parameters – does not eliminate the following issues.

- A GW has limited resources and can keep only a limited amount of simultaneous connections alive.
- Since mobile sensors can go out of range of a GW, the connection to the sensors may be lost.
- Because of tuned connection parameters, the GW may not recognize immediately a connection loss. For instance, this might depend on the *connection interval*–investigated further in this work, and being one of the three main BLE connection parameters, together with *slave latency* and *connection supervision timeout* [48]–specifically chosen for specific interactions between GW (master) and BLE slave nodes. Should the connection interval be too long, it may thus have an impact on internal data exchanges operated by the GW between two consecutive connection events. This is especially true since, generally, the three connection parameters mentioned above are related as follows:

$$T_{\mathrm{C-suptimeout}} > 2 \cdot T_{\mathrm{C-interval}} \cdot (1 + T_{\mathrm{S-latency}})$$

where: $T_{\mathrm{C-suptimeout}}$ is the connection supervision timeout (dimension: [s]); $T_{\mathrm{S-latency}}$ is the slave latency (adimensional); and $T_{\mathrm{C-interval}}$ is the connection interval (dimension: [ms]) [49,50].

Besides connecting to sensor nodes, the GW has also other tasks to perform in the proposed DynGATT infrastructure: it needs to receive the sensor nodes' measurements, interpret the data, convert them properly, and forward them to a Cloud service. Therefore, it can be concluded that a GW is responsible for providing sensor nodes with the possibility to transmit data and, if necessary, for pre-process the received data to fit into packets that can be handled by servers in the Internet.

In order to implement such a dynamic IoT scenario, the GW *first* scans for BLE sensor nodes and, as soon as an advertising packet – emitted by a sensor node – is received, it starts establishing a BLE connection to the sensor node. *Then*, data from the sensor node are synchronized, pre-processed on the GW, and transmitted to a Cloud server that, in turn, stores them into a DataBase (DB). This approach also overcomes the limitations of a GW which can only keep a limited amount of connections alive. In other words, if many sensor nodes are permanently connected to a GW and new sensor nodes come close to the GW itself, the GW may not be able to connect to these new sensor nodes. In order to connect to a new sensor node, the GW would need to tear down an already established BLE connection.

With reference to the methods discussed in [18], the proposed DynGATT IoT-oriented mechanism relies on the use of an HTTP proxy Internet GW, with the difference that sensor nodes act as GATT servers, while the GW is the GATT client. Furthermore, data transfer and exchange with sensor nodes is initiated by the GW, instead of a sensor node. Hence, when compared to the RESTful API approach presented in [19], data accessed in the Cloud are always cached from a DB. With

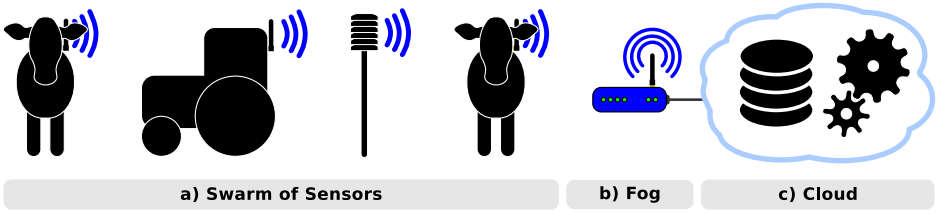


Fig. 1. Proposed BLE-based network infrastructure, composed of (a) swarm of sensors, (b) Fog layer, and (c) Cloud layer.

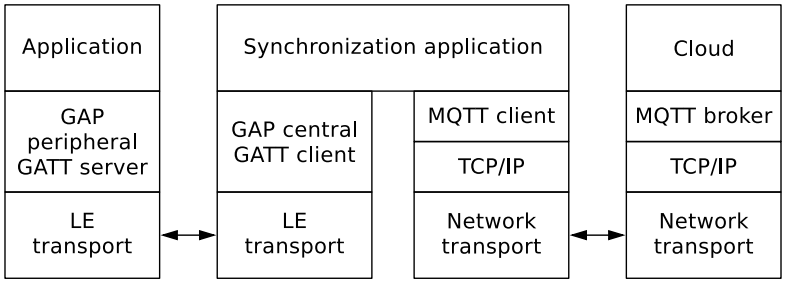


Fig. 2. Layered representation of the SSGW.

Table 1
Comparison among Internet GWs.

	HTTP proxy GW	RESTful API GW	SSGW
Multiple GWs support	✓	×	✓
Synchronization sequence	×	✓	✓
No device conn. limitation	×	✓	✓

the proposed setup, one can thus establish a connection for a short time interval, which gives the GW the opportunity to connect sequentially to a large number of sensor nodes.

4. Smart synchronization Internet GW

In traditional Internet GW applications, the logic behind data transfer resides at the application layer of the BLE device, such as a smart system with an HTTP proxy Internet GW or a smart system with an IPSP [18]. On the other hand, the logic can be implemented on the client in a smart system with a RESTful API Internet GW.

In this work, we introduce a Smart Synchronization Internet GW (SSGW) application. In comparison with its competing technologies, the logic in charge of *when* and *how* to connect to BLE devices is implemented in the GW (see Fig. 2). This, is turn, gives some additional benefits, as summarized in Table 1 and discussed below.

- Unlike a RESTful API GW, it is possible to have multiple SSGWs in the IoT infrastructure: this is because the Cloud does not need to know different GWs to access the sensor nodes.
- The SSGW can alter the sensor nodes' synchronization sequence, based on various properties (e.g., Received Signal Strength Indicator, RSSI, last cached synchronization time, etc.) as well as on their location (e.g., on the basis of their distance from the GW).
- There are no limitations on the amount of sensors in the IoT infrastructure; at the opposite, an HTTP proxy GW application – where the logic for data synchronization resides in the sensor node – may reach its connectivity limits.

4.1. Synchronization application

The proposed DynGATT data synchronization approach is based on the paradigm shown in Fig. 3, where a SSGW performs beacon-oriented sensor node identification, before being able to connect to the sensor node itself. In detail, this is done by scanning for BLE advertisement packets through the `SCAN()` function, to discover sensor nodes in its proximity. These sensor nodes (a) can have their BLE radio interface off or can be out of range of the GW, or (b) can have available data to be synchronized, can have their radio interface on, and can advertise themselves through the `ADV()` function, by broadcasting, for example, Eddystone-URL beacons $B_{\text{E-url}}$.

If the GW discovers advertising sensor nodes $\{S_i\}_{i=1}^{N_S}$, then it verifies, through the `CHECK()` function, if one or more advertising node(s) contain(s) the Eddystone-URL which the GW itself is interested on. Assuming that the verification procedure classifies the advertising sensor node S_k ($k \in \{1, \ldots, N_S\}$) as a trusted node, then the GW will *first* connect to S_k with the `CONNECT`(S_k) function and, *then*, will start the data synchronization procedure by: (a) retrieving S_k's services, characteristics, and descriptors; (b) enabling indications; (c) listening for incoming data packets $\{S_{k-\text{DATA}-j}\}_{j=1}^{N_{\text{DATA}}}$; and (d) handling them on the basis of previously retrieved information.

As soon as the GW receives the data, it processes them and sends them to a Cloud server, e.g., via an Message Queuing Telemetry Transport (MQTT) client over a TCP/IP connection. On the Cloud server, the data are further processed, e.g., stored into a DB.

5. GATT-based data synchronization

As mentioned in Section 1, each type of sensor node implements its own GATT server structure, which needs to be known by the remote device in order to interpret correctly the data. This is even true for the considered agricultural IoT scenario with, for example, (a) cows equipped with sensing collars, (b) sensor nodes on the fields, measuring soil and weather status, and (c) actuators that can control irrigation. As all these devices communicate via BLE, each one typically defines its own GATT server structure to access the data.

In order to receive the data, a GATT client, corresponding to a SSGW, *first* needs to connect to a GATT server (a sensor node or an actuator). *Then*, the SSGW (i.e., the GATT client) can access the sensor node or actuator's values (i.e., the GATT server) through the corresponding GATT characteristics hosted by the GATT server, either by activating notifications or directly by reading these characteristics. Many of these GATT services and characteristics are pre-defined and standardized by the SIG [51].

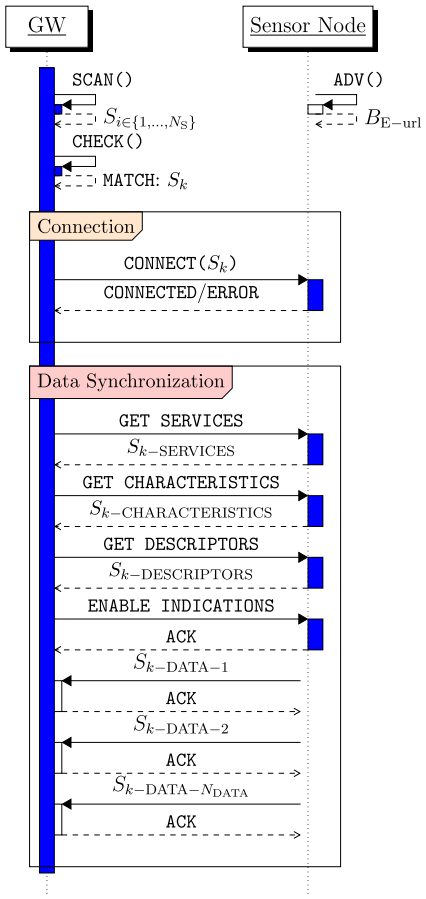


Fig. 3. Sensor node identification and GATT-based synchronization adopted in the proposed DynGATT approach.

But the question is: *how can recorded historical measurements be synchronized and what is necessary in order to obtain a history of these measurements?*

The first essential requirement to support historical data transfer is to add a timestamp to each measurement. However, as the numerous SIG-defined GATT services and characteristics are standardized, it is not suggested (although possible) to change the data format and to add a timestamp to, for example, a temperature reading. Nevertheless, reading a BLE characteristic should always be meant to retrieve the most recent data, rather than historical ones.

One possible solution is to combine different characteristics, in order to read historical data. Different SIG-defined characteristics are available to be used to read the time from a remote device: one of them is the *Current-Time* characteristic, another one is the *Date-Time* characteristic. To this end, one approach to read historical data can require to use one of these characteristics giving the time of a historical measurement, followed by, for example, notifications or a reading of the BLE characteristics carrying the measurement. However, this seems to be odd and difficult to implement and control. Some questions (such as *how are lost connections handled?*) arise, beside the fact that using *Current-Time* for transmitting past time indexes (stamps) is not what a BLE characteristic is supposed to be. As a consequence, this method does not comply with the SIG standard and would increase the data transfer, as many read operations and notification packets would be necessary.

Another possibility is to create user-defined GATT services and characteristics, where a characteristic can combine, at the same time, a timestamp and one or multiple measurements. Assume, as an example, that historical temperature readings shall be communicated: every time the characteristic is "read", it returns the timestamp combined with the measurement reading. Then, either the GATT client carries out read operations, until no further temperature measurement is available or notifications are enabled when all the available data are sent. Assuming that a sensor node measures temperature and relative humidity, this can be extended either by creating an additional GATT characteristic, holding timestamped relative humidity readings, or by combining timestamp, temperature and relative humidity in a unique GATT characteristic. While the *first* approach has the advantage of being more flexible, the *latter* mechanism saves bytes and, as such, reduces power consumption. This leads to the creation of different synchronization characteristics, identified by their UUIDs; then, a SSGW or the GATT client will know how to interpret the data according to the UUID itself.

While this approach solves the problem of transmitting historical measurements, it creates another problem: every GW needs to *a priori* know every possible synchronization characteristic, in order to understand and interpret the byte streams correctly. In other words, every GW needs to know all possible devices (sensor nodes, actuators, etc.) and their synchronization characteristics in advance, in order to communicate with them. Should new devices be integrated into an existing IoT infrastructure, all the GWs would need to be re-programmed, thus moving back to a static infrastructure.

Besides services and characteristics, a GATT server can also hold descriptors to provide more details on a BLE characteristic—e.g., defining its possible values range. Despite the predefined descriptors, it is possible to create new custom user-defined descriptors. The fact that there is no limitation on how many descriptors can be used in a service has been exploited in the definition of the proposed DynGATT synchronization approach.

5.1. GATT server layout

In order to move from the static infrastructure generated by a static approach towards a dynamic approach, our work proposes the use of descriptors, in order to define the layout for services' and characteristics' synchronization. This approach leads to the identification of a single characteristic used to synchronize data and known by all devices, where the layout of the transmitted data packets is described by the corresponding descriptors.

In the left side of Fig. 4, the GATT layout for the proposed SSGW service is shown, highlighting the following blocks: (a) the synchronization service; (b) the corresponding synchronization characteristic used to receive the data, in turn followed by the next descriptors (c)–(e); (c) the (mandatory) Client Characteristic Configuration Descriptor (CCCD), which is defined by the SIG and used to enable or disable notifications or indications; (d) a newly introduced (and a non-SIG-defined) mandatory *Blob Count Descriptor* descriptor, used to tell the GW the expected amount of data blobs; (e) a sequence of (optional) descriptors, telling the GW how to interpret the stream of data received by the characteristics handler.

Moreover, consider Table 2 as the list of the UUIDs known in the DynGATT infrastructure. Based on these UUIDs, it is possible to create and introduce new sensor nodes with all possible combinations of the specified measurements. An illustrative instance is given at the right side of Fig. 4, relative to a weather station that measures ambient temperature and relative humidity. Additionally, the weather station stores and synchronizes the battery level. On the basis of the description shown in the left side of Fig. 4, this is done by defining the synchronization service and characteristic, followed by the CCCD, and the blob count (mandatory for all types of sensors). Hence, the following descriptors are specific for this weather station example: the *first* descriptor tells that the first element of the data stream is a timestamp; the *second* descriptor is associated with the battery status information; the *third* descriptor is related to the ambient temperature; and, finally, the *last* descriptor is for the ambient relative humidity.

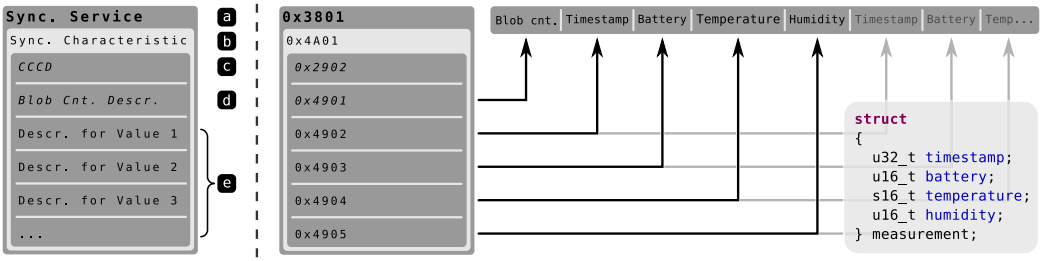


Fig. 4. GATT-based layout, with descriptors used to define characteristics layout: (left) generic layout, (right) example instance for a weather station.

0x4901	0x4902	0x4903	0x4904	0x4905	0x6388D8FB	0x04A5	0x05F0	0x17DE

Fig. 5. Representative example of a descriptor related to a weather station, including timestamp, battery voltage level, ambient temperature, and ambient relative humidity.

0x4901	0x4902	0x4903	0x4907	0x4908	0x6388D8FB	0x04A5	0x0726	0x2134

Fig. 6. Representative example of a descriptor related to a soil status node, including timestamp, battery voltage, soil temperature, and soil moisture.

What the GW knows from this GATT layout is how to interpret the received data bytes of the GATT characteristic. For a better understanding, let us assign numerical values to the example in Fig. 4. Recall that Table 2 specifies some UUIDs, with their corresponding descriptions and types of data. Based on this information, the GW can assume that one blob of data (corresponding to the gray box on the right side of Fig. 4) will consist of 10 B, where, as shown in Fig. 5: the first 4 B represent the Unix-like timestamp (dimension: [s]; in Fig. 5, they are equal to 0x6388D8FB that, once converted to decimal, lead to a final value equal to 1669912827); the next 2 B correspond to the battery voltage level (dimension: [V]), with a resolution of 0.01 V (in Fig. 5, they are equal to 0x04A5 that, once converted to decimal, lead to a final value equal to 11.89 V); the next 2 B correspond to the ambient temperature (dimension: [°C]), with a resolution of 0.01 °C (in Fig. 5, they are equal to 0x05F0 that, once converted to decimal, lead to a final value equal to 15.20 °C); and the last 2 B correspond to the ambient relative humidity (adimensional: [%]), with a resolution of 0.01 % (in Fig. 5, they are equal to 0x17DE that, once converted to decimal, lead to a final value equal to 61.10 %). The very first byte is the number of blobs contained in one characteristic reading, notification or indication. Please note that, in this example, the specified (16 bit) UUIDs are for demonstration purposes only. For real applications, full 128 bit UUIDs need to be defined as specified by the SIG.

Based on the descriptors in Table 2, other sensor nodes can also be defined. As an example, consider a sensor node that measures the soil status, e.g., soil temperature and soil moisture. If this sensor is introduced, the GATT descriptor sequence (block (e) in the left side of Fig. 4) can be equal to 0x4902, 0x4903, 0x4907, and 0x4908, as shown in Fig. 6, thus representing a 10-byte stream containing timestamp (equal to 0x6388D8FB that, once converted to decimal, leads to a final value equal to 1669912827), battery voltage (equal to 0x04A5 that, once converted to decimal, leads to a final value equal to 11.89 V), soil temperature (equal to 0x0726 that, once converted to decimal, leads to a final value equal to 18.30 °C), and soil moisture (equal to 0x2134 that, once converted to decimal, leads to a final value equal to 85 %), respectively.

A concluding example is related to a weather station measuring ambient temperature, atmospheric pressure and precipitation. Its GATT descriptor sequence (block (e) in the left side of Fig. 4) would look like as 0x4902, 0x4903, 0x4904, 0x4906, and 0x4909, as shown in Fig. 7, representing a 14-byte stream containing information on timestamp (equal to 0x6388D8FB that, once converted to decimal, leads to a final value equal to 1669912827), battery voltage (equal to 0x04A5 that, once converted to decimal, leads to a final value equal to 11.89 V), ambient temperature (equal to 0x05F0 that, once converted to decimal, leads to a final value equal to 15.20 °C), atmospheric pressure (equal to 0x000F7BAC that, once converted to decimal, leads to a final value equal to 101.47 kPa), and precipitation (equal to 0x0019 that, once converted to decimal, leads to a final value equal to 25 mm).

Table 2
GATT information known in the infrastructure.

UUID	Description	Resolution	Format
0 × 3801	Synchronization Service	–	–
0 × 4A01	Synchronization Characteristics	–	–
0 × 4901	Blob count	–	uint8
0 × 4902	Unix time in seconds	–	uint32
0 × 4903	Battery in V	0.01 V	uint16
0 × 4904	Ambient temperature in °C	0.01 °C	int16
0 × 4905	Ambient relative humidity in %	0.01 %	uint16
0 × 4906	Atmospheric pressure in Pa	0.1 Pa	uint32
0 × 4907	Soil temperature in °C	0.01 °C	int16
0 × 4908	Soil moisture in %	0.01 %	uint16
0 × 4909	Precipitation in mm	1 mm	uint16

It is noteworthy to highlight that the eight illustrative GATT descriptors shown in Table 2 (0x4902 to 0x4909) allow to handle $2^8-1 = 255$ different sensor nodes or synchronization streams, without the need of re-configuring the infrastructure, namely the GW.

5.2. Connection tuning

According to the Bluetooth Core specifications [1], as anticipated in Section 3, there are a couple of parameters which allow to tune the data connection: (a) the *connection interval*, defining the time interval between two consecutive connection events, during which the radio interface should be turned on if data have to be exchanged between the devices, and ranging from 7.5 ms to 4.0 s; and (b) the *data payload*, defining how many bytes can be transmitted within one connection event, and ranging from 0 B to 251 B per transmitted/received packet.

By tuning these parameters, two different goals can be achieved. *First*, the data throughput can be increased or decreased, by either changing the connection interval or the data payload. *Second*, the power consumption can be controlled properly tuning the timing at which the radio is turned on. In particular, a data payload's change has a high impact on power consumption. Moreover, for every connection event, additional control information (e.g., a header, a Cyclic Redundancy Check, CRC, or a Message Integrity Code, MIC) may be included. If the same amount of information is contained in a single

0x4901	0x4902	0x4903	0x4904	0x4906	0x4909	0x6388D8FB	0x04A5	0x05F0	0x000F7BAC	0x0019

Fig. 7. Representative example of a descriptor related to an alternative weather station, including timestamp, battery voltage, ambient temperature, atmospheric pressure, and precipitation.

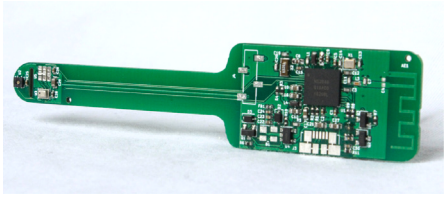


Fig. 8. nRF52840 SoC-based sensor node measuring ambient temperature, ambient relative humidity and atmospheric pressure.

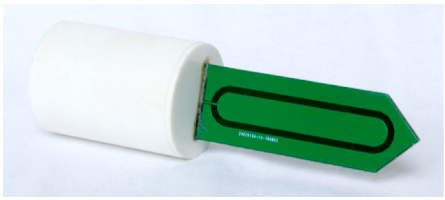


Fig. 9. nRF51822 SoC-based sensor node measuring soil temperature and soil moisture.

transmitted packet, the overhead needs to be sent only once, thus saving wireless transmission time. However, this is only true in an ideal scenario with no or negligible wireless interference. At the opposite, in noisy environments, large data payloads can also lead to an increased power consumption, as a whole dataset needs to be resent if not correctly received. With small data payloads, only limited amounts of data need to be re-transmitted. The connection timing only influences the power consumption on established connections when no data are transmitted, as a limited amount of data has to be exchanged in order to keep a connection alive. Otherwise, if a connection is only used to transmit data and terminated right after data transmission, the connection interval has an impact on the data throughput between the master device (i.e., the GW) and the peripheral device (e.g., a longer connection interval results in longer data transmissions and lower data throughput [49]). At the opposite, the impact of connection establishment on the overall power consumption is negligible with respect to the consumption incurred by data transfer.

In order to achieve short connection times in the aforementioned scenario, the data throughput is increased by setting the Maximum Transmission Unit (MTU) size, which defines, at L2CAP level, the maximum payload size that can fit into one data packet. Because one data packet may fit more than the measurements size, the previously introduced blob count (block (d) in the left side of Fig. 4) is used to help filling the packet with as many measurements as possible. Both the remote device and the GW can define their own MTU sizes and, as soon as a connection is established, the GATT client (the GW in this scenario) can start negotiating the MTU size that will be used throughout the connection. Depending on the negotiated MTU size and on the size of a single measurement, the sensor node calculates how many measurements can fit into a single data packet: this amount will correspond to the content of the very first byte transmitted by the GATT server. Therefore, the GATT client (namely, the SSGW) will know how many measurements are transmitted within a single reading.

6. Experimental evaluation

In order to evaluate the proposed SSGW with the DynGATT synchronization protocol, we consider an experimental setup in both a static scenario and a dynamic scenario, deploying the DynGATT protocol (proposed in Section 5) on the following self-developed sensor nodes.

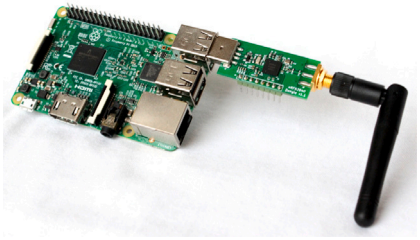


Fig. 10. Raspberry Pi 3 with external USB Bluetooth adapter.

SN1 A Nordic Semiconductor nRF52840 System-on-Chip (SoC)-based sensor node measuring ambient temperature, ambient relative humidity, and atmospheric pressure (Fig. 8).

SN2 A Nordic Semiconductor nRF51822 SoC-based sensor node measuring soil temperature and soil moisture (Fig. 9).

SN3 A Nordic Semiconductor nRF52840 SoC-based sensor node measuring ambient temperature and acceleration (similar to SN1 shown in Fig. 8).

The firmware for all sensor nodes is based on the Zephyr Real-Time Operating System (RTOS) [52] and implements the proposed DynGATT synchronization protocol.

On the SSGW side, DynGATT was implemented through a Python application, which interacts with the Linux OS Bluetooth driver via the BlueZ Desktop Bus (D-Bus) API [53]. In order to verify and evaluate different BLE solutions, the SSGW application was tested on the following systems:

- a PC running Ubuntu 18.04 (a) with the internal Bluetooth adapter, as well as (b) with an external USB Bluetooth adapter, based on the nRF52840 SoC and the Zephyr RTOS;
- a Raspberry Pi 3 SBC running Raspbian Wheezy, with the same external USB Bluetooth adapter used for the PC (Fig. 10).

As discussed in Section 4.1, the synchronization mechanism is implemented as follows. *First*, the SSGW, through its BLE interface (either internal or external), scans for sensor nodes broadcasting a specific Eddystone-URL and stores the retrieved sensor nodes in an array. *Then*, the SSGW:(a) connects sequentially to the sensor nodes, one by one; (b) discovers the GATT server layout; (c) looks for the synchronization service, characteristics and descriptors; and (d) enables the indications. As soon as the indications are enabled, the sensor node, which the SSGW is currently connected to, starts sending its data to the SSGW via indications. Upon data reception, the SSGW unpacks the received data, encodes them in JavaScript Object Notation (JSON) format and prints the JSON-formatted output to the `stdout` interface. Then, this output is sent to an MQTT client connecting to an MQTT broker and publishing the JSON data. Additionally, the SSGW prints out debug information (such as timings) into the `stderr` interface.

6.1. Static scenario

In the static scenario, one SSGW and different sensor nodes are deployed. The SSGW, as well as the sensor nodes, are placed at fixed

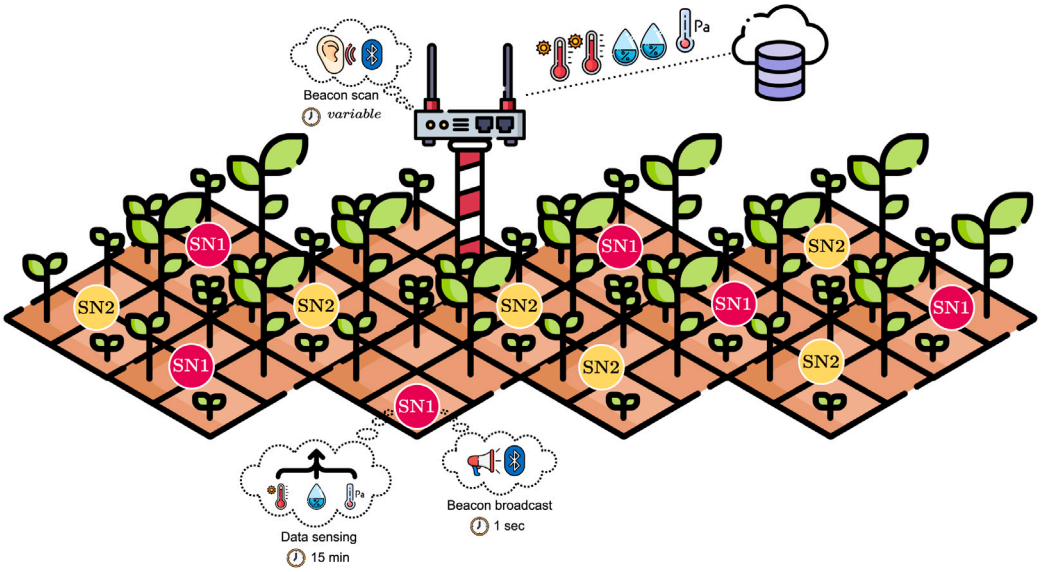


Fig. 11. Pictorial representation of the considered static scenario.

locations and do not move throughout the experiment. With this experiment, the impact of timings can be evaluated, as the distance between the SSGW and each sensor node does not change. The results can also be used for power consumption estimation based on real measurements, as well as using a power profiler (e.g., Nordic Semiconductor offers an online power profiler [54] that can be used for this purpose).

In detail, as shown in the pictorial representation in Fig. 11, the sensor nodes are configured to take a measurement every 15 min and to broadcast an Eddystone-URL once a second. Additionally, the SSGW is configured to scan for sensor nodes and, if any is found, to synchronize the data available at the sensor nodes. Hence, one may alter the scan period on the SSGW in order to get different timings. In this experiment, sensor nodes SN1 and SN2, storing 14 B and 10 B in one data blob, respectively, are used.

With this setup, we measure timings on the SSGW side to perform a power consumption estimation. More precisely, the considered timings are as follows: (a) the overall interaction time, from the instant at which the BLE connection is established to the instant at which the BLE connection is terminated; (b) the time required for sending the *raw* data via indications; (c)the amount of bytes and datasets; and (d) the amount of indications used to transmit data.

As anticipated before, a significant component of the power consumption at the deployed sensor nodes is related to the power consumption of the SoC during data transmission, which can be measured through the online power profiler offered by Nordic Semiconductor. In detail, the considered sensor nodes are typically in two operational states: (a) broadcasting their own Eddystone-URL or (b) connected to a SSGW for data synchronization purposes.

6.2. Dynamic scenario

As discussed in Section 3, one of the aims of the proposed DynGATT protocol is to handle dynamic scenarios, such as tracking livestock in a farm. Similarly to the static scenario shown in Section 6.1, we deployed one SSGW and different sensor nodes. However, in order to make the scenario dynamic, the following modifications were introduced, as shown in the pictorial representation in Fig. 12: (a) we mounted SN3 on a cleaning cart, and (b) we simulated the behavior of an animal on SN2. Similarly to the other sensor nodes, SN3 broadcasts an Eddystone-URL once per second and measures the ambient temperature and the current three-axes (x, y, z) acceleration every 5 min. Additionally, the acceleration sensor in SN3 sends an interrupt to the MCU in the sensor node when moved: this will generate an additional acceleration measurement with the corresponding timestamp. In comparison to the static scenario, the cleaning cart permanently moves around. This means that, most of the time, the cleaning cart is sufficiently far from the SSGW, which cannot thus detect SN3 on board the cart. However, it can be assumed that the cleaning cart moves near the SSGW at least once a day, except for weekends. As soon as the cleaning cart is sufficiently close to the SSGW, SN3 will be detected by the SSGW, which will start the synchronization procedure.

The simulated behavior is based on some data retrieved from a highly automated farm. On this farm, cows can go to a milking robot in order to get milked. While they get milked, they are simultaneously getting fed. The farmer can decide, on the basis of the cows' status, health, and time from the last calving, how often a cow is allowed to go to the milking robot. Then, it is assumed that cows are allowed to go to the milking robot two to five times a day (with an average equal to three), with an estimated milking time of about eight minutes (as suggested by literature works [55–58]). From this information, we can implement a simulated behavior, by randomly switching off and on the radio module of a single sensor node. First, the sensor node switches its radio module off and calculates the time its radio module stays off, representing the cow away from the milking robot. Then, the radio module switches on, representing the cow at the milking robot, and stays on as long as the milking procedure lasts. After milking, the radio module switches off again. In order to calculate the timings, we use a normal distribution with a mean value of 4 h and a standard deviation of 1 h. In this experiment, we assume that the SSGW is placed right next to the milking robot and that cows sleep about 12 h. That is also the reason why the mean value was chosen to 4 h: since the SSGW is switched off for 12 h a day (during the simulated sleep), the chosen distribution corresponds, on average, to 3 milkings per day. The milking duration, i.e., the time a cow is close to the SSGW, is calculated with a mean value of 480 s and a standard deviation of 120 s. During this time, the SSGW needs to detect and connect to the sensor node to synchronize the data. Unlike the static scenario, in the dynamic scenario the SSGW scans for sensor nodes every minute (but for an incomplete synchronization). In order to avoid to connect to a sensor node that has recently been synchronized, the SSGW skips all sensor nodes that were successfully synchronized in the last hour.

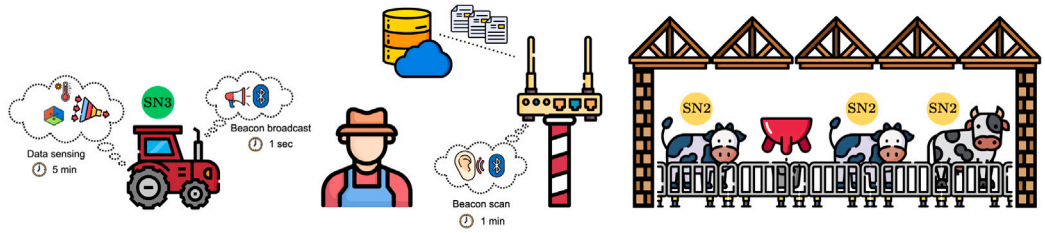


Fig. 12. Pictorial representation of the considered dynamic scenario.

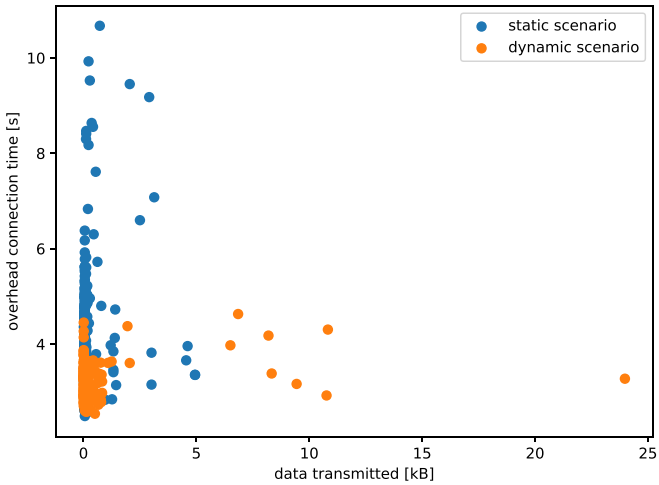


Fig. 13. Connection overhead T_{OH} as a function of the number of transmitted bytes.

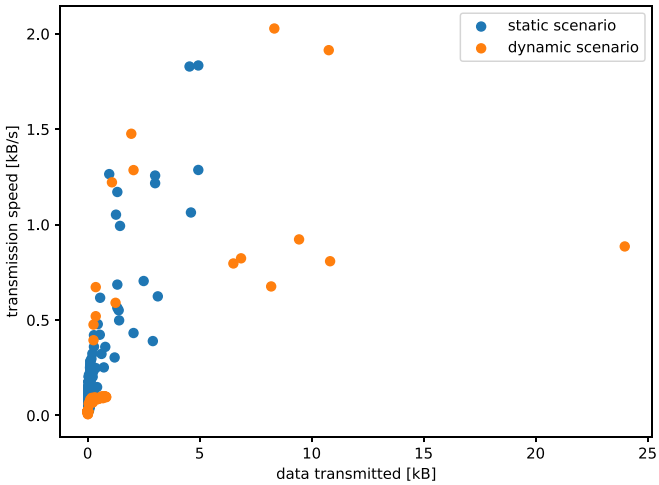


Fig. 14. Transmission rate as a function of the transmitted data.

7. Results and discussion

On the basis of the experimental evaluation scenarios described in Section 6, a first performance evaluation, associated with the connection overhead (including connection establishment, service discovery, time synchronization, and connection termination), is carried out by analyzing the timing logs at the SSGW. In Fig. 13, the connection overhead, denoted as T_{OH}, is shown as a function of the number of transmitted bytes. As can be seen, T_{OH} ranges from 2.49 s to 10.67 s. From the results in Fig. 13, it can be seen that there is no correlation between T_{OH} and the number of transmitted bytes, i.e., the amount of data transferred does not influence the overhead time. However, the time required for data transfer heavily depends on the amount of available information: on the basis of our experimental evaluation, the achieved data rates vary between 6.95 B/s and 2,027.82 B/s, corresponding to a large interval. Fig. 14 shows the transmission rate (dimension: [kB/s]) as a function of the transmitted data.

In order to estimate as precisely as possible the worst-case power consumption, taking into account that the larger the amount of data transferred at once, the higher the throughput for the synchronization procedure, we considered the timings for transmitting only a single dataset. This quantity, denoted as T_{TX}, ranges from 0.27 s to 2.36 s in the static scenario.

In the *broadcasting state*, a nRF52840 SoC consumes, on average, $I_{\mathrm{ADV}} = 22\,\mu\mathrm{A}$ when broadcasting 31 B of payload every 1000 ms [54]. In this state, the power consumption heavily depends on the BLE advertising interval.[2] If, for example, we increase the interval to 2000 ms, the power consumption drops to $I_{\mathrm{ADV}} = 12\,\mu\mathrm{A}$. However, since we use a one-second advertising interval, we consider this scenario as the worst-case for power consumption, as the largest possible payload is transmitted.

In the *synchronization state*, i.e., when connected to a central device, the SoC usually consumes, on average, $I_{\mathrm{SYNC}} = 1.36\,\mathrm{mA}$, with peaks up to 10.9 mA. This performance is obtained using a transmission power equal to 4 dBm, the largest possible payload, and a connection interval of 30 ms [54].

In order to calculate the overall worst-case power consumption, both broadcasting and synchronization states need to be combined together. Consider that the sensor node collects a measurement every $T_{\mathrm{DATA}} = 15\,\mathrm{min}$ and that the SSGW connects to the sensor node as soon as data are available. This would result in a connection time $T_{\mathrm{CONN}} = T_{\mathrm{OH}} + T_{\mathrm{TX}} = 10.67\,\mathrm{s} + 2.36\,\mathrm{s} = 13.03\,\mathrm{s}$ every $T_{\mathrm{DATA}} = 15\,\mathrm{min}$, in which the SoC consumes $I_{\mathrm{SYNC}} = 1.36\,\mathrm{mA}$. During the rest of the time, namely during an interval of duration $T_{\mathrm{ADV}} = T_{\mathrm{DATA}} - T_{\mathrm{CONN}} = 900\,\mathrm{s} - 10.36\,\mathrm{s} = 886.97\,\mathrm{s}$, the sensor node is in advertising state and consumes $I_{\mathrm{ADV}} = 22\,\mu\mathrm{A}$. Hence, the average worst-case power consumption of the SoC is the following:

$$I_{\mathrm{SoC}} = \frac{(T_{\mathrm{OH}} + T_{\mathrm{TX}}) \times I_{\mathrm{SYNC}} + T_{\mathrm{ADV}} \times I_{\mathrm{ADV}}}{T_{\mathrm{DATA}}} = \frac{13.03\,\mathrm{s} \times 1.36\,\mathrm{mA} + 886.97\,\mathrm{s} \times 22\,\mu\mathrm{A}}{900\,\mathrm{s}} = 41.37\,\mu\mathrm{A}. \quad (1)$$

Moreover, even considering the worst-case battery energy scenario, with a sensor node equipped with a 350 mAh battery, the sensor node could still operate for 8460 h (approximately 352 days), assuming that only the SoC is affecting the power consumption and the battery does not self-discharge.

Hence, these results highlight the impact of advertising and synchronizing states on power consumption. Regarding power consumption, the advertising state represents a fixed lower bound specified by the

[2] For the sake of clarity, in this paper, unless stated otherwise, we implicitly assume that the voltage is fixed. Therefore, the power consumption is associated with the current intensity.

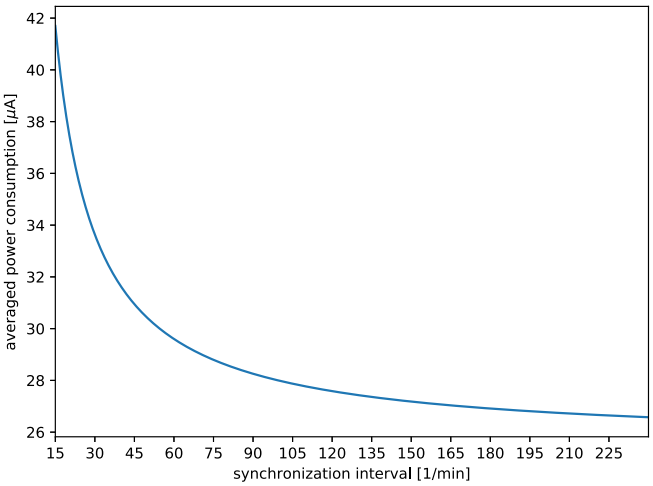


Fig. 15. Average worst-case power consumption for synchronizing data.

SoC. However, in order to control the energy consumed by the SoC, it is possible to configure the SSGW for the scenario's needs. The more often data are synchronized, the higher is the energy consumption. Fig. 15 shows the correlation between the average power needed for synchronizing data and the synchronization interval in the worst-case scenario, on the basis of Eq. (1). From Fig. 15, it can be concluded that increasing the synchronization period decreases the average power consumption of the SoC. Additionally, this has another impact as well: because transmitting more data at once increases the transmission rate, the power consumed to transmit one dataset decreases as well (this, however, is not considered in Fig. 15), resulting in even lower power consumption. However, this comes at the cost that data are available in the Cloud layer at a delayed time. For example, if the SSGW increases the synchronization interval from 15 min to 120 min, the average power consumption of the SoC drops from 41.37 μA to 27.58 μA (worst-case), resulting in an increased lifetime of 528 days with data being updated in the cloud every 2 hours.

On the SoC side, the advertisement period can be altered to further decrease the power consumption. But this has an impact on the responsiveness of the system, as it takes longer for a SSGW to detect and connect to the sensing device—the longer the advertising interval, the longer the scan time and the slower the system responsiveness. Moreover, this may have a negative impact especially in dynamic scenarios, such as in smart farming, where cows can move close to or walk away from a SSGW very quickly, thus preventing the SSGW from *on-time* detection of the presence of an animal and synchronization of its data. This further highlights why the trade-off between responsiveness and power consumption has to be carefully considered and adjusted according to the application needs.

Another strategy allowed by DynGATT is related to the possibility to quickly synchronize data through the use of dynamic trackers: our experimental evaluation confirms that communicating more data at once increases the data rate. In the best-case scenario, it was possible to synchronize 8335 B in 4.11 s transmitted via indications without overhead.

Besides power consumption, the proposed SSGW with the DynGATT dynamic synchronization protocol offers the following advantages. *First*, since BLE connections are established only if necessary to synchronize data, a (theoretically) unlimited amount of BLE devices can be integrated into the IoT infrastructure. This would not be possible for scenarios in which connections are kept alive, as devices have only a limited number of (a) resources per connection and (b) peripherals to connect to. *Second*, BLE peripherals dynamically appearing and disappearing from the SSGW's visibility can be handled, as data on the peripherals are stored temporarily. *Third*, thanks to the fact that the structure of both synchronization service and characteristics is defined by GATT descriptors, the SSGW only needs to know a specific synchronization GATT characteristic, but does not need to *a priori* know its data byte structure. *Finally*, another advantage of the proposed approach is that new sensor nodes can be integrated in an existing IoT infrastructure without updating or changing the SSGW. However, this is true only to some extent: as long as the descriptors are pre-defined and known by the SSGW, then no update is necessary. As soon as new measurements are introduced and new descriptors thus need to be introduced in the system, the SSGW also needs to be updated in order to handle these new data types.

8. Conclusion and future work

In this paper, we have presented a new Smart Synchronization Internet GW (SSGW) and a dynamic BLE GATT-based synchronization mechanism, denoted as DynGATT, to be used in dynamic IoT scenarios, such as those in agricultural domains. In these scenarios, there are *on-field* BLE sensor nodes, collecting data and advertising themselves through Eddystone-based beacons, and BLE-equipped SSGWs, in charge of discovering sensor nodes and synchronizing with them in a timely- and energy-preserving way.

The flexibility of the proposed DynGATT approach makes it attractive in the future, to place GWs on mobile carriers, such as Unmanned Aerial Vehicles (UAVs) and Unmanned Ground Vehicles (UGVs), thus exploiting the potentialities of our protocol.

Security needs also to be taken into account. Securing end-to-end the communication between *on-field* sensor nodes and the Cloud allows to mitigate threats from data harvesting to data processing, but completely prevents the possibility to pre-process data at GW level. The investigation of end-to-end security mechanisms between sensor nodes and a SSGW is an interesting extension, possibly through group key-based algorithms, exploiting the dynamicity of BLE sensor nodes and sharing keys with sensor nodes at their first interaction. Obviously, these security *add-ons* will have an impact on power consumption, that should be carefully considered.

Finally, in order to improve energy management policies, a dynamic power tuning strategy can be defined, in which the advertising interval can be regulated based on the battery state. Then, additional information may be considered to be sent inside the beacon packets, to share further knowledge between a sensor node and the SSGW exploiting all the available features.

CRediT authorship contribution statement

Christian Hirsch: Conceptualization, Methodology, Software, Validation, Investigation, Writing – original draft, Writing – review & editing. **Luca Davoli:** Conceptualization, Methodology, Software, Validation, Investigation, Writing – original draft, Writing – review & editing. **Radu Grosu:** Conceptualization, Investigation, Writing – original draft, Supervision, Funding acquisition. **Gianluigi Ferrari:** Conceptualization, Investigation, Writing – original draft, Writing – review & editing, Supervision, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

This work received funding from the European Union's Horizon 2020 research and innovation program ECSEL Joint Undertaking (JU) under grant agreement No. 783221, AFarCloud project - "Aggregate Farming in the Cloud". The work of L.D. and G.F. is also partially supported by the Agritech project - "National Research Centre for Agricultural Technologies", project code CN00000022, funded under the National Recovery and Resilience Plan (NRRP), Mission 4 Component 2 Investment 1.4 - Call for tender no. 3138 of 16/12/2021 of Italian Ministry of University and Research funded by the European Union – NextGenerationEU, Concession Decree no. 1032 of 17/06/2022 adopted by the Italian Ministry of University and Research; SMALLDERS research project - "Smart Models for Agrifood Local vaLue chain based on Digital technologies for Enabling covid-19 Resilience and Sustainability", funded by the PRIMA Program - Section 2 Call multi-topics 2021; and by the European Union's Horizon 2020 research and innovation program ECSEL Joint Undertaking (JU) under grant agreement No. 876019, ADACORSA project - "Airborne Data Collection on Resilient System Architectures". The JU received support from the European Union's Horizon 2020 research and innovation programme and the nations involved in the mentioned projects. The work reflects only the authors' views and opinions; neither the European Union nor the European Commission can be considered responsible for any use that may be made of the information it contains.

References

[1] Bluetooth SIG, Core Specification 4.0, 2010, https://www.bluetooth.com/specifications/specs/. (Accessed on 23 May 2021).
[2] M. Siekkinen, M. Hiienkari, J.K. Nurminen, J. Nieminen, How low energy is bluetooth low energy? Comparative measurements with ZigBee/802.15.4, in: 2012 IEEE Wireless Communications and Networking Conference Workshops (WCNCW), IEEE, Paris, France, 2012, pp. 232–237, http://dx.doi.org/10.1109/WCNCW.2012.6215496.
[3] B.B. Olyaei, J. Pirskanen, O. Raeesi, A. Hazmi, M. Valkama, Performance comparison between slotted IEEE 802.15.4 and IEEE 802.1 lah in IoT based applications, in: 2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), IEEE, Lyon, France, 2013, pp. 332–337, http://dx.doi.org/10.1109/WiMOB.2013.6673381.
[4] J. Decuir, Bluetooth smart support for 6LoBTLE: Applications and connection questions, IEEE Consum. Electron. Mag. 4 (2) (2015) 67–70, http://dx.doi.org/10.1109/MCE.2015.2392955.
[5] J. Quevedo, M. Antunes, D. Corujo, D. Gomes, R.L. Aguiar, On the application of contextual IoT service discovery in information centric networks, Comput. Commun. 89–90 (2016) 117–127, http://dx.doi.org/10.1016/j.comcom.2016.03.011.
[6] G. Codeluppi, A. Cilfone, L. Davoli, G. Ferrari, LoRaFarM: A LoRaWAN-based smart farming modular IoT architecture, Sensors 20 (7) (2020) http://dx.doi.org/10.3390/s20072028.
[7] S. Cheshire, M. Krochmal, Multicast DNS, RFC, (6762) IETF, 2013, URL https://tools.ietf.org/html/rfc6762.
[8] Zero Configuration Networking (Zeroconf), 2020, http://www.zeroconf.org/. (Accessed on 16 July 2022).
[9] Z. Shelby, Constrained RESTful Environments (CoRE) Link Format, RFC, (6690) IETF, 2012, URL https://tools.ietf.org/html/rfc6690.
[10] S. Cheshire, M. Krochmal, DNS-Based Service Discovery, RFC, (6763) IETF, 2013, URL https://tools.ietf.org/html/rfc6763.
[11] Z. Shelby, K. Hartke, C. Bormann, The Constrained Application Protocol (CoAP), RFC, (7252) IETF, 2014, URL https://tools.ietf.org/html/rfc7252.
[12] iBeacon Technology, 2020, https://developer.apple.com/ibeacon/. (Accessed on 16 July 2022).
[13] Google Beacon Platform - Eddystone, 2020, https://github.com/google/eddystone. (Accessed on 16 July 2022).
[14] A. Mackey, P. Spachos, K.N. Plataniotis, Smart parking system based on bluetooth low energy beacons with particle filtering, IEEE Syst. J. 14 (3) (2020) 3371–3382, http://dx.doi.org/10.1109/JSYST.2020.2968883.
[15] H.-T. Chen, P.-Y. Lin, C.-Y. Lin, A smart roadside parking system using bluetooth low energy beacons, in: Web, Artificial Intelligence and Network Applications, Springer International Publishing, Cham, 2019, pp. 471–480, http://dx.doi.org/10.1007/978-3-030-15035-8_44.
[16] F. Campaña, A. Pinargote, F. Domínguez, E. Peláez, Towards an indoor navigation system using bluetooth low energy beacons, in: 2017 IEEE Second Ecuador Technical Chapters Meeting, ETCM, IEEE, Salinas, Ecuador, 2017, pp. 1–6, http://dx.doi.org/10.1109/ETCM.2017.8247464.
[17] G. De Blasio, A. Quesada-Arencibia, C.R. Garcia, J.C. Rodriguez-Rodriguez, R. Moreno-Diaz, A protocol-channel-based indoor positioning performance study for bluetooth low energy, IEEE Access 6 (2018) 33440–33450, http://dx.doi.org/10.1109/ACCESS.2018.2837497.
[18] J. Decuir, C. Hansen, C. Gordon, I. Blair, C. Pfister, K. Schultz, D. Richkas, T. Wei, V. Zhodzishsky, K. Kambhampati, Y. Kwan, J. Tien, N. Hunn, Internet Gateways, White Paper, (v01) Bluetooth SIG, 2016, p. 22, https://www.bluetooth.com/wp-content/uploads/2019/03/InternetGateways_WP_v01.pdf. (Accessed on 16 July 2022).
[19] M. Andersson, R. Heydon, K. Schulz, M. Olsson, A. Larsson, GATT REST API, White Paper, (V10r01) Bluetooth SIG, 2014, https://www.bluetooth.com/wp-content/uploads/2019/03/GATT-REST-API_WP_V10r01.pdf. (Accessed on 16 July 2022).
[20] M. Andersson, A. Larsson, R. Heydon, M. Olsson, T. Howes, GAP REST API, White Paper, (V10r01) Bluetooth SIG, 2014, https://www.bluetooth.com/wp-content/uploads/2019/03/GAP-REST-API_WP_V10r01-1.pdf. (Accessed on 16 July 2022).
[21] Bluetooth SIG, Internet Protocol Support Profile, 2014, https://www.bluetooth.com/specifications/specs/internet-protocol-support-profile-1-0. (Accessed on 25 November 2022).
[22] M. Spörk, C.A. Boano, M. Zimmerling, K. Römer, BLEach: Exploiting the full potential of IPv6 over BLE in constrained embedded IoT devices, in: Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems, SenSys '17, ACM, 2017, pp. 1–14, http://dx.doi.org/10.1145/3131672.3131687.
[23] J. Tosi, F. Taffoni, M. Santacatterina, R. Sannino, D. Formica, Throughput analysis of BLE sensor network for motion tracking of human movements, IEEE Sens. J. 19 (1) (2019) 370–377, http://dx.doi.org/10.1109/JSEN.2018.2877102.
[24] N. Islam, B. Ray, F. Pasandideh, IoT based smart farming: Are the LPWAN technologies suitable for remote communication? in: 2020 IEEE International Conference on Smart Internet of Things (SmartIoT), 2020, pp. 270–276, http://dx.doi.org/10.1109/SmartIoT49966.2020.00048.
[25] A. Ilapakurti, C. Vuppalapati, Building an IoT framework for connected dairy, in: 2015 IEEE First International Conference on Big Data Computing Service and Applications, 2015, pp. 275–285, http://dx.doi.org/10.1109/BigDataService.2015.39.
[26] T. Truong, A. Dinh, K. Wahid, An IoT environmental data collection system for fungal detection in crop fields, in: 2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering, CCECE, 2017, pp. 1–4, http://dx.doi.org/10.1109/CCECE.2017.7946787.
[27] M.-S. Liao, S.-F. Chen, C.-Y. Chou, H.-Y. Chen, S.-H. Yeh, Y.-C. Chang, J.-A. Jiang, On precisely relating the growth of phalaenopsis leaves to greenhouse environmental factors by using an IoT-based monitoring system, Comput. Electron. Agric. 136 (2017) 125–139, http://dx.doi.org/10.1016/j.compag.2017.03.003.
[28] F. Edwards-Murphy, M. Magno, P.M. Whelan, J. OHalloran, E.M. Popovici, b+WSN: Smart beehive with preliminary decision tree analysis for agriculture and honey bee health monitoring, Comput. Electron. Agric. 124 (2016) 211–219, http://dx.doi.org/10.1016/j.compag.2016.04.008.
[29] S. Trilles, J. Torres-Sospedra, A. Belmonte, F.J. Zarazaga-Soria, A. Gonzalez-Perez, J. Huerta, Development of an open sensorized platform in a smart agriculture context: A vineyard support system for monitoring mildew disease, Sustain. Comput.: Inf. Syst. (2019) http://dx.doi.org/10.1016/j.suscom.2019.01.011.
[30] R. Ratasuk, N. Mangalvedhe, A. Ghosh, B. Vejlgaard, Narrowband LTE-M system for M2M communication, in: 2014 IEEE 80th Vehicular Technology Conference (VTC2014-Fall), 2014, pp. 1–5, http://dx.doi.org/10.1109/VTCFall.2014.6966070.
[31] Y. Tang, S. Dananjayan, C. Hou, Q. Guo, S. Luo, Y. He, A survey on the 5G network and its impact on agriculture: Challenges and opportunities, Comput. Electron. Agric. 180 (2021) 105895, http://dx.doi.org/10.1016/j.compag.2020.105895.
[32] F.K. Shaikh, S. Karim, S. Zeadally, J. Nebhen, Recent trends in internet of things enabled sensor technologies for smart agriculture, IEEE Internet Things J. (2022) 1, http://dx.doi.org/10.1109/JIOT.2022.3210154.
[33] G. Manogaran, M. Alazab, K. Muhammad, V.H.C. de Albuquerque, Smart sensing based functional control for reducing uncertainties in agricultural farm data analysis, IEEE Sens. J. 21 (16) (2021) 17469–17478, http://dx.doi.org/10.1109/JSEN.2021.3054561.
[34] R.K. Singh, R. Berkvens, M. Weyn, AgriFusion: An architecture for IoT and emerging technologies based on a precision agriculture survey, IEEE Access 9 (2021) 136253–136283, http://dx.doi.org/10.1109/ACCESS.2021.3116814.
[35] K. Grgic, D. Zagar, J. Balen, J. Vlaovic, Internet of things in smart agriculture – possibilities and challenges, in: 2020 International Conference on Smart Systems and Technologies, SST, 2020, pp. 239–244, http://dx.doi.org/10.1109/SST49455.2020.9264043.
[36] K. Dineva, D. Parvanov, T. Atanasova, G. Mateeva, P. Petrov, G. Kostadinov, Towards CPS/IoT system for livestock smart farm monitoring, in: 2021 International Conference Automatics and Informatics, ICAI, 2021, pp. 252–255, http://dx.doi.org/10.1109/ICAI52893.2021.9639460.
[37] A. Yazdinejad, B. Zolfaghari, A. Azmoodeh, A. Dehghantanha, H. Karimipour, E. Fraser, A.G. Green, C. Russell, E. Duncan, A review on security of smart farming and precision agriculture: Security aspects, attacks, threats and countermeasures, Appl. Sci. 11 (16) (2021) http://dx.doi.org/10.3390/app11167518.

[38] J. Prakash, P. Thorwe, T.Q.S. Quek, AgriAuth: Sensor collaboration and corroboration for data confidence in smart farms, in: Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks, WiSec '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 383–385, http://dx.doi.org/10.1145/3448300.3468260.
[39] A. Llaria, G. Terrasson, H. Arregui, A. Hacala, Geolocation and monitoring platform for extensive farming in mountain pastures, in: 2015 IEEE International Conference on Industrial Technology, ICIT, 2015, pp. 2420–2425, http://dx.doi.org/10.1109/ICIT.2015.7125454.
[40] L. Wen, Z. Qingfang, H. Ke, L. Xueke, G. Kai, Design of agricultural irrigation hydropower dual control intelligent equipment based on NB-IoT technology, in: Proceedings of the 7th International Conference on Informatics, Environment, Energy and Applications, IEEA '18, Association for Computing Machinery, New York, NY, USA, 2018, pp. 42–48, http://dx.doi.org/10.1145/3208854.3208897.
[41] D. Davcev, K. Mitreski, S. Trajkovic, V. Nikolovski, N. Koteli, IoT agriculture system based on LoRaWAN, in: 2018 14th IEEE International Workshop on Factory Communication Systems, WFCS, 2018, pp. 1–4, http://dx.doi.org/10.1109/WFCS.2018.8402368.
[42] K. Aliev, M. Moazzam Jawaid, S. Narejo, E. Pasero, A. Pulatov, Internet of plants application for smart agriculture, Int. J. Adv. Comput. Sci. Appl. 9 (4) (2018) http://dx.doi.org/10.14569/IJACSA.2018.090458.
[43] J. Lloret, S. Sendra, L. Garcia, J.M. Jimenez, A wireless sensor network deployment for soil moisture monitoring in precision agriculture, Sensors 21 (21) (2021) http://dx.doi.org/10.3390/s21217243.
[44] J. Bauer, N. Aschenbruck, Design and implementation of an agricultural monitoring system for smart farming, in: 2018 IoT Vertical and Topical Summit on Agriculture - Tuscany (IoT Tuscany), 2018, pp. 1–6, http://dx.doi.org/10.1109/IOT-TUSCANY.2018.8373022.
[45] J.A. Brenes, G. Marín-Raventós, When one wireless technology is not enough: A network architecture for precision agriculture using LoRa, Wi-Fi, and LTE, in: A.K. Nagar, D.S. Jat, G. Marín-Raventós, D.K. Mishra (Eds.), Intelligent Sustainable Systems, Springer Nature Singapore, Singapore, 2022, pp. 103–112.
[46] P.K. Reddy Maddikunta, S. Hakak, M. Alazab, S. Bhattacharya, T.R. Gadekallu, W.Z. Khan, Q.-V. Pham, Unmanned aerial vehicles in smart agriculture: Applications, requirements, and challenges, IEEE Sens. J. 21 (16) (2021) 17608–17619, http://dx.doi.org/10.1109/JSEN.2021.3049471.
[47] L. Nóbrega, P. Gonçalves, P. Pedreiras, J. Pereira, An IoT-based solution for intelligent farming, Sensors 19 (3) (2019) http://dx.doi.org/10.3390/s19030603.
[48] R. Katila, T. Nguyen Gia, T. Westerlund, Analysis of mobility support approaches for edge-based IoT systems using high data rate bluetooth low energy 5, Comput. Netw. 209 (2022) 108925, http://dx.doi.org/10.1016/j.comnet.2022.108925.
[49] P. Bulić, G. Kojek, A. Biasizzo, Data transmission efficiency in bluetooth low energy versions, Sensors 19 (17) (2019) http://dx.doi.org/10.3390/s19173746.
[50] A. Jaimin, BLE power optimization parameters, 2021, https://buildstorm.com/blog/ble-power-optimization-parameters/. (Accessed on 22 November 2022).
[51] Bluetooth GATT Specifications, 2020, https://www.bluetooth.com/specifications/gatt/. (Accessed on 23 May 2021).
[52] Zephyr Project, 2020, https://www.zephyrproject.org/. (Accessed on 16 July 2022),
[53] DBus API, 2020, https://git.kernel.org/pub/scm/bluetooth/bluez.git/tree/doc. (Accessed on 16 July 2022).
[54] Online Power Profiler, 2020, https://devzone.nordicsemi.com/power/w/opp. (Accessed on 16 July 2022).
[55] M. Ayadi, Optimization of milking frequency in dairy ruminants, in: N. M'Hamdi (Ed.), Lactation in Farm Animals, IntechOpen, Rijeka, 2019, http://dx.doi.org/10.5772/intechopen.87303.
[56] J. Wang, D. Lovarelli, N. Rota, M. Shen, M. Lu, M. Guarino, The potentialities of machine learning for cow-specific milking: Automatically setting variables in milking machines, Animals 12 (13) (2022) http://dx.doi.org/10.3390/ani12131614.
[57] A. Castro, J. Pereira, C. Amiama, J. Bueno, Estimating efficiency in automatic milking systems, J. Dairy Sci. 95 (2) (2012) 929–936, http://dx.doi.org/10.3168/jds.2010-3912.
[58] J. Hogenboom, L. Pellegrino, A. Sandrucci, V. Rosi, P. D'Incecco, Hygienic quality, composition, and technological performance of raw milk obtained by robotic milking of cows, J. Dairy Sci. 102 (9) (2019) 7640–7654, http://dx.doi.org/10.3168/jds.2018-16013.

Christian Hirsch is an independent researcher focusing on IoT solutions, hardware prototyping and development, especially based on near-range Wi-Fi and BLE and low-power Narrowband-IoT and LTE-M cellular. He has been a University Assistant at the Institute of Computer Engineering at the Vienna University of Technology until February 2022.

Luca Davoli is a non-tenured Assistant Professor at the Internet of Things (IoT) Laboratory, Department of Engineering and Architecture, University of Parma, Italy. He obtained his Dr. Ing. degree in computer engineering and his Ph.D. in information technologies at the Department of Information Engineering of the same university, in 2013 and 2017, respectively. His research interests focus on IoT, Pervasive Computing, Big Stream and Software-Defined Networking.

Radu Grosu received the Ph.D. degree in computer science from the Technical University of Munich, Munich, Germany, in 1994. He is currently a Professor and the Head of the Cyber-Physical Group, Faculty of Informatics, Vienna University of Technology. Before receiving his appointment at the Vienna University of Technology, he was an Associate Professor with the Computer Science Department, State University of New York, Stony Brook, where he co-directed the Concurrent-Systems Laboratory and co-founded the Systems-Biology Laboratory. He was a Research Associate with the Computer Science Department, University of Pennsylvania. He is a Research Professor with the Computer Science Department, State University of New York. His research interests include modeling, analysis and control of cyber–physical, and biological systems. His application focus include green operating systems, mobile ad-hoc networks, automotive systems, the Mars rover, cardiac-cell networks, and genetic regulatory networks. He is a member of the International Federation of Information Processing WG 2.2. He was the recipient of the National Science Foundation Career Award, the State University of New York Research Foundation Promising Inventor Award, the ACM Service Award.

Gianluigi Ferrari received the Laurea (*summa cum laude*) and Ph.D. degrees in electrical engineering from the University of Parma, Parma, Italy, in 1998 and 2002, respectively. Since 2002, he has been with the University of Parma, where he is currently an Associate Professor of telecommunications and also the coordinator of the Internet of Things (IoT) Laboratory, Department of Engineering and Architecture. His current research interests include signal processing, advanced communication and networking, and IoT and smart systems.