Enabling Dynamic and Lightweight Management of Distributed Bluetooth Low Energy Devices

Minkeun Ha
School of Technology and Health
KTH Royal Institute of Technology, Stockholm, Sweden
Email: minkeun.ha@sth.kth.se

Thomas Lindh
School of Technology and Health
KTH Royal Institute of Technology, Stockholm, Sweden
Email: thomas.lindh@sth.kth.se

***Abstract*—Bluetooth Low Energy (BLE) is a wireless communication technology for Internet of Things (IoT) applications. Although recent versions of BLE embrace the 6LoWPAN standard for IPv6 low-power device communication, this is often not the best choice for constrained-node networks due to downsides such as increased memory and computation overhead, heavier network stack, increased power consumption, backward incompatibility, etc. A crucial requirement for successful IoT solutions is an efficient strategy for monitoring and control of highly distributed sensor nodes. In this paper, we propose a dynamic and lightweight model for management of distributed constrained BLE nodes based on OMA Lightweight M2M (LWM2M) protocol. To this end, we have designed a new intermediate component, a BLE-LWM2M gateway (GW), which has dual roles: a BLE master that monitors and controls the BLE devices, and a LWM2M client that acts on behalf of an operator's LWM2M management server. The BLE-LWM2M GW has interfaces to dynamically manage connected BLE devices as LWM2M objects, called ObjectTwins, in runtime without a need for user configuration. All services that BLE devices provide are mapped to the ObjectTwins in the gateway, which are exposed to the LWM2M server. Hence, this model enables a LWM2M server to manage distributed BLE devices by interacting with the BLE-LWM2M GW using the LWM2M standard methods, and the constrained devices can still use a native Bluetooth communication protocol. To verify its feasibility, a prototype of a mobile healthcare system has been implemented in a testbed.**

I. Introduction

Bluetooth Low Energy (BLE) is a wireless personal area network technology aimed at smart applications in the healthcare, fitness, beacons, home entertainment, etc [1]. Compared to Bluetooth Classic, BLE technology is intended to significantly reduce power consumption at considerably lower cost. From version 4.2 the BLE specification includes IPv6 connectivity implemented as the Internet Protocol Support Profile (IPSP), which uses a Bluetooth-optimized version of 6LoWPAN [2]. The addition of 6LoWPAN means that each BLE device can be accessed globally and the number of addresses available for the Internet of Things (IoT) will be all but inexhaustible [3], [4]. However, this 6LoWPAN over BLE approach also comes with some downsides as follows. Increased memory space is necessary to generate and process relatively large IPv6 packets. Because the IP encapsulation and decapsulation are done over IPSP, the processing and computation overhead can be increased. In addition, the BLE device needs to have a heavier network stack. As a result, the BLE devices suffer from larger power consumption. Furthermore, IP-based protocols are not supported in early versions of BLE and many typical Bluetooth devices are not connected to the Internet. Due to the limitations, the 6LoWPAN over BLE approach is not always the best choice for constrained nodes.

An essential part of creating successful IoT solutions and services with BLE devices is to provide an efficient strategy for IoT operators and IoT users to handle continuous device management of distributed BLE devices in order to respond rapidly to unexpectedly changing device conditions or service requirements [5], [6]. However, efficient and remote runtime device management of non-IP BLE devices are facing difficulties and lacks a general solution.

In this paper, we propose a dynamic and lightweight model for management of distributed constrained BLE nodes based on Lightweight M2M (LWM2M). To this end, we have designed a new intermediate component, a BLE-LWM2M gateway (GW), which has dual roles: a BLE master that monitors and controls the BLE devices and a LWM2M client that acts on behalf of an operator's LWM2M management server. The BLE-LWM2M GW dynamically creates a special LWM2M object, named ObjectTwin, that is a logical replica of the connected BLE device. It has interfaces to the services and characteristics of connected BLE devices in runtime without any user configuration. The ObjectTwin is a LWM2M object which follows the standard LWM2M object model and exposes its resources to the LWM2M server. The LWM2M server has access to state information and can control the BLE device via the corresponding ObjectTwin provided by the BLE-LWM2M GW. This ObjectTwin can be created or deleted dynamically when BLE slaves are connected to or disconnected from the BLE-LWM2M GW. As a result, IoT operators and end users can monitor the connection of BLE peripherals, query BLE device metadata, configuration and state information, and control the BLE devices using the LWM2M standard protocol. To verify the feasibility of our approach, a testbed and prototype application for mobile healthcare has been implemented in a testbed with BLE sensors for heart activity monitoring.

The rest of this paper is organized as follows. A general background is outlined in Section II. In Section III, the proposed LWM2M-based dynamic and lightweight device management model for BLE devices is introduced. Section IV

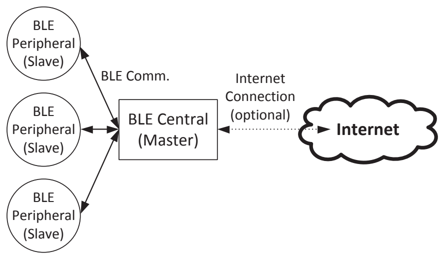


Fig. 1. An example of a network topology with BLE devices.

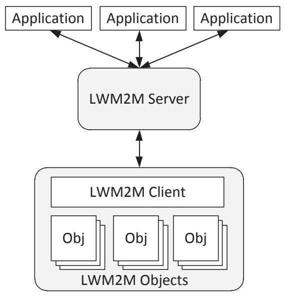


Fig. 2. An overview of the LWM2M architecture.

presents the testbed implementation and a prototype system for ECG monitoring in mobile healthcare, and finally the conclusions in Section V.

II. Background

A. Bluetooth Low Energy

The Bluetooth Low Energy (BLE), or Bluetooth Smart, is a power-conserving variant of Bluetooth personal area network technology. Unlike Bluetooth Classic, it has been optimized for ultra-low power consumption and low cost instead of focusing on the data rates. There are two types of BLE devices: the BLE central (master) and the BLE peripheral (slave). Fig. 1 shows a simple BLE network topology. The BLE interaction starts with establishment and pairing procedures to set up a connection between two BLE devices (master and slave). The BLE Central initiates a connection to the BLE peripheral. Note that, one master can connect to multiple slaves, but one slave can only be connected to a single master. BLE splits the ISM frequency band into 40 channels with a width of 2 MHz. All BLE devices independently broadcast short advertisement packets on one of three advertisement channels. The device then listens for a connection request on this channel. After a connection is established, one of the remaining 37 channels is used for exchanging larger data blocks. On top of the link layer, the Logical Link Control and Adaptation Protocol (L2CAP) resides to multiplex data from upper layers, store channel identifiers, and configure parameter options. The L2CAP data packet has a default 23-byte MTU, which can be extended to 65535 bytes depending on BLE hardware implementation. The Attribute Protocol (ATT) and the Generic Attribute Profile (GATT) are defined to manage profiles that contain configurations of the connection and application service profiles.

B. LWM2M Architecture and Object Models

OMA lightweight M2M (LWM2M) [7] is defined by the Open Mobile Alliance to support lightweight management of M2M and IoT devices. It provides a lightweight and secure communication interface along with an efficient data model. As shown in Fig. 2, the LWM2M architecture basically follows a traditional client-server communication model: a LWM2M server and a LWM2M client. The LWM2M server is a logical component typically deployed in a data center, a cloud system, or a private or public server system, that is hosted by service providers and operators. The LWM2M server acts as a resource directory that receives registrations from LWM2M clients and look-up requests from users, but it also issues necessary requests to their resources (LWM2M clients) [8]. LWM2M defines a simple resource model, which consists of objects and resources, where each piece of information is made available by the LWM2M clients. The objects are logical groups of resources. Further, there is an optional server, called LWM2M Bootstrap server, which provides the initial configuration settings and corresponding parameters to the LWM2M clients when bootstrapping the devices.

The LWM2M architecture is based on the RESTful design in the Constrained Application Protocol (CoAP) [9], so that LWM2M management operations and responses are mapped to CoAP method calls and response codes. The LWM2M protocol provides several logical interfaces over CoAP [9] between the server and client to perform management of a wide range of remote resource-constrained devices [10]: The *Bootstrap* interface allows the LWM2M Bootstrap server to manage the keying material of a LWM2M client approved to access a specified LWM2M server, the access control policies to check whether the LWM2M server has access right for performing a operation, and the initial configuration parameters for LWM2M clients. The *Device Discovery and Registration* interface allows a LWM2M client to make the specific LWM2M server aware of its existence and register the client's capability to the LWM2M server. The *Device Management and Service Enablement* interface allows the LWM2M server to perform device management operations for LWM2M clients. It enables LWM2M client services by querying specific resources of the LWM2M client using LWM2M methods. Lastly, the *Information Reporting* interface allows the LWM2M client to notify resource information to the LWM2M server. This report can be submitted periodically or triggered by special events.

The LWM2M specification introduces a simple two-level hierarchical data model, based on IPSO Smart objects [11], consisting of objects and resources that can be identified using Uniform Resource Identifiers (URIs). Open Mobile Alliance (OMA) also maintains a naming authority called the Open Mobile Naming Authority (OMNA) and assigns unambiguous unique identifiers instead of resource names. This approach enables objects and resources to be mapped to the CoAP resources and URI path components, in the standard format.

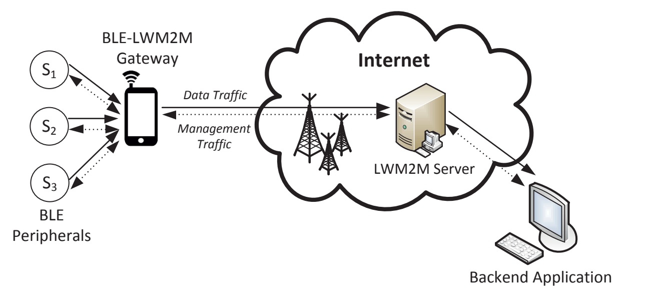


Fig. 3. A network architecture of the LWM2M-based dynamic and lightweight device management model for distributed BLE nodes.

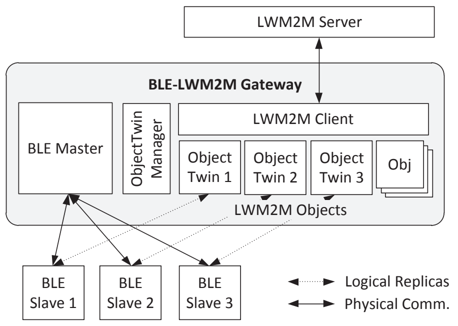


Fig. 4. The BLE-LWM2M GW software block diagram and its logical network connection to BLE slaves and a LWM2M server.

The LWM2M object model is reusable and extensible, so that any other organizations and users can define new objects for their own services.

III. Dynamic and Lightweight Management Model for constrained BLE nodes

In this section, we introduce a dynamic and lightweight model for management of distributed constrained BLE nodes based on LWM2M. Our goal is to efficiently support remote device management for BLE devices, which are resource-constrained non-IP devices and cannot be transparently accessed from the Internet.

A. Network Architecture

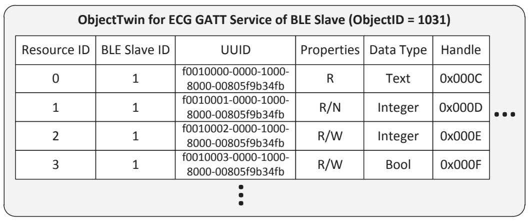
Fig. 3 shows a network architecture for a distributed LWM2M-based device management model. This model consists of four main components: BLE peripherals, a BLE-LWM2M GW, a LWM2M server, and a backend application. A group of BLE peripherals are BLE slaves, which are connected to a BLE-LWM2M GW and provide sensor/actuator services such as heart activity monitoring, blood pressure monitoring, etc. The BLE-LWM2M GW is a special device which has dual roles: a BLE master of connected BLE slaves and a LWM2M client. The LWM2M server maintains a collection of LWM2M clients and performs management operations. The users collect device state information and sensor data, perform management operations on BLE slaves, and visualize obtained information using backend applications. The LWM2M server can be placed in a data center, cloud systems, or a private server. Several IT companies offer cloud computing services for IoT, e.g. IBM Bluemix, Cisco Jasper, Amazon Cloud, and Microsoft Azure. In our prototype study, we used Microsoft Azure to support data storage and analysis.

B. BLE-LWM2M Gateway

The BLE-LWM2M GW is a key component to enable the LWM2M-based dynamic and lightweight management of BLE devices. Fig. 4 shows the software block diagram of the BLE-LWM2M GW. The BLE-LWM2M GW consists of three software modules and logical LWM2M objects. It has a BLE Master module to manage the access of the slaves as a BLE master and a LWM2M Client module to work as a LWM2M client. To support dynamic and lightweight device management, a new ObjectTwin feature is defined, which is a logical replica of its corresponding physical BLE slave. The ObjectTwin represents BLE services (and their characteristics) of connected BLE slaves as LWM2M objects (and resources) and exposes its control interfaces for READ, WRITE, NOTIFY, or INDICATE operations to move data between the BLE slave and the BLE master as standardized LWM2M methods. The remote users (or the LWM2M server) only need to interact with LWM2M Client module in the BLE-LWM2M GW to perform management operations on BLE slaves using LWM2M methods without any additional software stack. The difference between general LWM2M objects and ObjectTwins is that each ObjectTwin resource is mapped to a BLE characteristic of the BLE slave. When the BLE-LWM2M GW receives LWM2M messages for any of ObjectTwin resources, it sends the corresponding BLE commands to the connected BLE slave. It then reports received results from the BLE slave to the LWM2M server.

1) ObjectTwin Data Structure: As described above, the ObjectTwin is a virtual replica of each service of connected BLE devices. When a BLE service is discovered by the BLE-LWM2M GW, it assigns an object ID to the service and creates an ObjectTwin with the object ID. The BLE-LWM2M GW can manage an object ID pool and assign one of them to newly created ObjectTwins. The number of object IDs in this pool determines how many BLE slaves that can be connected to the BLE-LWM2M GW. This capacity depends on the hardware specification.

An ObjectTwin does not store actual data, but it manages the parameters to access the characteristic of BLE service and data type of the values such as UUID, properties, handle, etc. The size of each resource in an ObjectTwin is 22 bytes when a 128-bit UUID is stored. After an ObjectTwin is created, the BLE-LWM2M GW will send a request to the ObjectTwin manager when it receives LWM2M messages regarding an ObjectTwin resource. The ObjectTwin manager looks up the ObjectTwin and translates the request to a BLE message. It then queries the requested characteristic value from the BLE device or writes the value to the characteristic of the BLE device. Fig. 5 shows an example of an ObjectTwin data structure for the ECG GATT service.



ObjectTwin for ECG GATT Service of BLE Slave (ObjectID = 1031)

Resource ID	BLE Slave ID	UUID	Properties	Data Type	Handle
0	1	f0010000-0000-1000-8000-00805f9b34fb	R	Text	0x000C
1	1	f0010001-0000-1000-8000-00805f9b34fb	R/N	Integer	0x000D
2	1	f0010002-0000-1000-8000-00805f9b34fb	R/W	Integer	0x000E
3	1	f0010003-0000-1000-8000-00805f9b34fb	R/W	Bool	0x000F

Fig. 5. An example of a logical ObjectTwin for the ECG GATT service. Resource ID 0 is a description for the service (read only). Resource ID 1 is a resource to read ECG sensor value that can be readable or notifiable. Resource ID 2 is a resource to read current sampling rate of ECG sensor or write a new sampling rate (Read/Write/Notify). Resource ID 3 is an interface to turn on/off timestamping for each ECG reading.

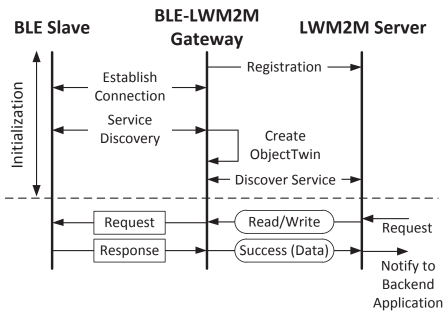


Fig. 6. A message diagram for ObjectTwin creation and a simple read/write operation.

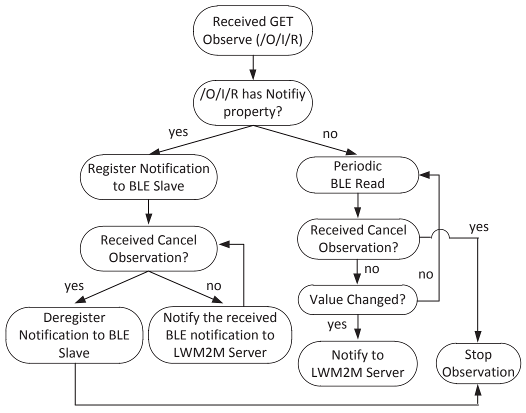

Received GET Observe (/O/I/R)
/O/I/R has Notifiy property?
yes
no
Register Notification to BLE Slave
Periodic BLE Read
Received Cancel Observation?
Received Cancel Observation?
yes
no
no
no
yes
Value Changed?
yes
Deregister Notification to BLE Slave
Notify the received BLE notification to LWM2M Server
Notify to LWM2M Server
Stop Observation

Fig. 7. A flowchart to handle the LWM2M Observe method on an ObjectTwin resource. In the figure, /O/I/R means /<Object ID>/<Object Instance ID>/<Resource ID>.

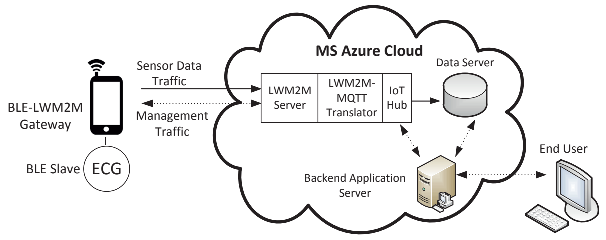


Fig. 8. A device management architecture of the prototype implementation for heart activity monitoring.

2) Device Management Operations for ObjectTwin: Fig. 6 shows the initialization process of the BLE-LWM2M GW. When it starts up, it registers its existence and its LWM2M objects to the LWM2M server. Later, if a BLE slave is connected to the BLE-LWM2M GW, the ObjectTwin Manager in the BLE-LWM2M GW performs service discovery to the connected BLE device. According to the results, the ObjectTwin manager dynamically creates ObjectTwins for discovered services. A LWM2M server can retrieve information about created ObjectTwins by searching the LWM2M object list. After creating the ObjectTwin, the LWM2M server can send READ/WRITE/EXECUTE messages regarding specific resources of the ObjectTwin. These messages are mapped to the corresponding BLE operations by the ObjectTwin manager. As a result, the BLE-LWM2M GW gets the latest state of the BLE slave or data generated by the BLE slave. The received data from the BLE slave will be delivered to the LWM2M server.

The BLE-LWM2M GW performs special activities to handle an Observation request regarding the resources of ObjectTwins. Fig. 7 shows how the BLE-LWM2M GW uses the GET Observe method in detail. When it receives an Observation request for the specific resource of ObjectTwin, it checks whether or not it has the BLE Notify property. If so, the ObjectTwin manager registers the notification for the corresponding attribute of the BLE slave. It then sends a response to the received notification to the LWM2M server. Later, when it receives an Observation Cancel message, it deregisters the notification and stops the observation reports to the LWM2M server. Otherwise, the ObjectTwin manager periodically reads the corresponding BLE characteristic mapped to the observed LWM2M resource. If the value is changed, it is reported to the LWM2M server until the Observation is canceled.

IV. Prototype Implementation

We have implemented a LWM2M-based device management model that dynamically adapts to changes in runtime without reconfiguration. The prototype application is a mobile healthcare system that monitors ECG data, stores the historical ECG data with timestamps in the data center, and visualizes the data on the web. As shown in Fig. 8, the system consists of an ECG BLE sensor node, a BLE-LWM2M GW, a cloud-based management server implemented in Microsoft Azure, and a backend application. The BLE sensor node and the BLE-LWM2M GW are built on Raspberry Pi 3 (RPi) Model B, which has a 1.2 GHz 64-bit Quad-core ARMv8 Cortex A53 processor (Broadcom BCM2837), 1 GB RAM, one Ethernet port, 802.11n Wireless LAN, and BLE 4.1. The RPi operating system is a Linux-based distribution (Raspbian Jessie 2016-09-23 Release). We used Bluez 5.42[1], which is an open-source Bluetooth library officially accepted in Linux, for the BLE network stack and Wakaama[2] for the LWM2M Client

[1]BlueZ, http://www.bluez.org/
[2]Wakaama, http://www.eclipse.org/wakaama/

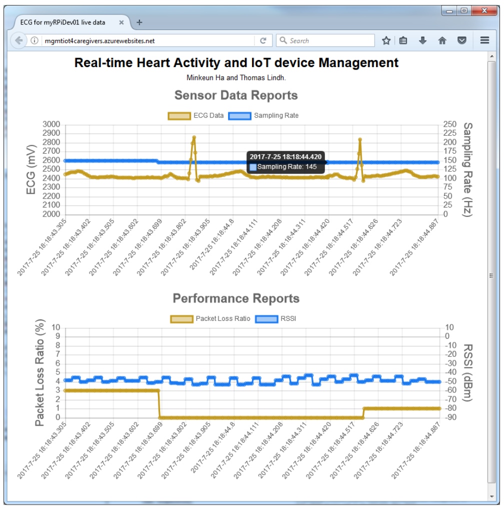
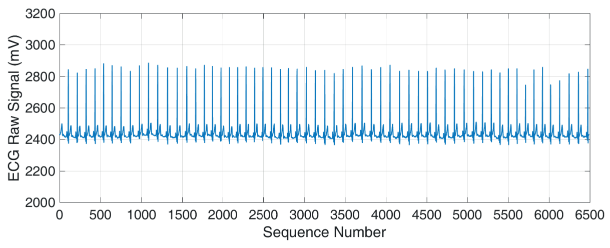
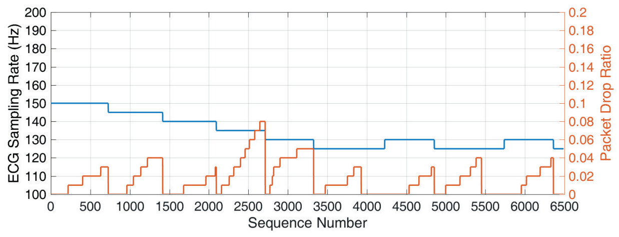


Fig. 9. Web-based data visualization of ECG recordings generated by a BLE slave. The upper graph shows real-time ECG data readings (yellow) and sampling rate (blue). The lower graph shows packet loss ratio (yellow) and RSSI (blue). The x-axis in both graphs shows timestamps for each ECG data sample.



(a) ECG data readings from a BLE slave using LWM2M.



(b) ECG sampling rate control according to the packet drop ratio of BLE link.

Fig. 10. A testbed experiment for LWM2M-based dynamic and lightweight device management.

module. Lastly, we implemented the LWM2M server using node.js based on the LWM2M library, lwm2m-node-lib[3]. In this prototype, the ObjectTwin manager creates ObjectTwins for GATT services, which can easily be extended to support other types of BLE services. In addition, the Microsoft Azure cloud system[4] generally supports MQTT but not natively the LWM2M. A protocol translator is therefore deployed between the LWM2M server and Microsoft Azure IoT hub, that translates LWM2M messages to MQTT-based messages and vice versa. The cloud-based server collects, stores and analyses data, and estimates the patients health condition. In addition, its analytics framework is used to visualize the collected ECG data on the web.

In this scenario, the backend application monitors packet drops in transmission of ECG recordings between the BLE slave and the BLE-LWM2M GW, and also controls the ECG sampling rate. The ECG sampling rate is set to 150 Hz. The packet drop ratio is monitored for 5 seconds and it is cleared every 5 seconds. The backend application then decreases or increases the ECG sampling rate (in steps of 5 Hz) when the observed packet drop ratio is higher or lower than the pre-defined thresholds (0.02), respectively. Fig. 9 shows the web-based data visualization display. As shown in Fig. 10b, when packet loss ratio is higher/less than 0.02, the backend application successfully decreases/increases the ECG sampling rate, respectively.

V. Conclusion

This paper presents a model for dynamic and lightweight device management of BLE peripherals. The main contribution is an ObjectTwin model of the BLE-LWM2M GW that combines the Bluetooth low energy and LWM2M standard specifications to achieve the new features. In addition, the feasibility of the proposed system is verified in a testbed prototype implementation of a mobile healthcare application. The case study demonstrates device management of ECG BLE sensor devices and a heart activity monitoring service running in a commercial cloud computing platform (Microsoft Azure IoT hub).

[3]lwm2m-node-lib, https://github.com/telefonicaid/lwm2m-node-lib

[4]Microsoft Azure IoT hub, https://azure.microsoft.com/en-us/services/iot-hub/.

Acknowledgment

We thank Microsoft for offering a grant and support to use the Azure IoT platform.

References

[1] C. Gomez, J. Oller, and J. Paradells, ”Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology,” *Sensors*, vol. 12, no. 9, pp. 11734-11753, Aug. 2012.
[2] Bluetooth SIG, https://www.bluetooth.com/
[3] L. Atzori, A. Iera, G. Morabito, ”The Internet of Things: A survey,” *Computer Networks*, vol. 54, no. 15, pp. 2787-2805, Oct. 2010.
[4] S. Hong, D. Kim, M. Ha, S. Bae, S. Park, W. Jung, and J. Kim, ”SNAIL: An IP-based Wireless Sensor Network Approach Toward the Internet of Things,” *IEEE Wireless Comm.*, vol. 17, no. 6, pp. 34-42, Dec. 2010.
[5] J. Höller, V. Tsiatsis, C. Mulligan, S. Karnouskos, S. Avesand, and D. Boyle, ”From Machine-to-Machine to Internet of Things: introduction to a new age of intelligence,” in Academic Press, 2014.
[6] Z. Sheng, C. Mahapatra, C. Zhu, and V. C. M. Leung, ”Recent Advances in Industrial Wireless Sensor Networks Toward Efficient Management in IoT,” *IEEE Access*, vol. 3, pp. 622-637, May 2015.
[7] OMA Lightweight M2M v1.0 LWM2M OMA, 2014.
[8] M. Kovatsch, M. Lanter, Z. Shelby, ”Californium: Scalable Cloud Services for the Internet of Things with CoAP,” *Proc. International Conference on the Internet of Things (IoT)*, Oct. 2014.
[9] C. Bormann, A. Castellani, and Z. Shelby, ”CoAP: An Application Protocol for Billions of Tiny Internet Nodes,” *IEEE Internet Computing*, vol. 16, no. 2, pp. 62-67, Mar. 2012.
[10] S. Rao, D. Chendanda, C. Deshpande, V. Lakkundi, ”Implementing LWM2M in constrained IoT devices,” *Proc. IEEE Conference on Wireless Sensors (ICWiSe)*, Aug. 2015.
[11] IPSO smart object guideline, Starter pack 1.0, IPSO, September 2014.