Data management of sensor signals for high bandwidth data streaming to the Cloud

Theodoros Soultanopoulos[1], Stelios Sotiriadis[1,2], Euripides G.M. Petrakis[1], Cristiana Amza[2]
[1]School of Electrical and Computer Engineering, Technical University of Crete (TUC), Chania Greece
[2]Computer Engineering Research Group, University of Toronto, Toronto, Canada
tsoultanopoulos@gmail.com, s.sotiriadis@utoronto.ca, petrakis@intelligence.tuc.gr, amza@ece.utoronto.ca

Abstract—Internet of Things (IoT) and Cloud computing highlight new requirements for data collection, storage and analysis of user data. In this work we focus on the data collection from Bluetooth Low Energy (BLE) devices in order to identify and track data from things in real time, as well as to enhance data analysis by allowing processing in the Cloud. The paper proposes an IoT service architecture and a system implementation for a gateway service running on smart devices (tablets, smartphones) that supports generic access to BLE compliant sensors and allows on-the-fly sensor registration using an easy to use data information schema. The gateway is designed to support the processing (e.g. filtering) of sensor data before they are transferred to the cloud in example of FOG computing (that defines high bandwidth and low latency IoT distributed system). The experimental results demonstrate that the system supports real time communication and fast data collection, as the total time for data transmission (from smart sensors to the cloud) is less than 130 ms.

Index Terms—Cloud Computing, Internet of Things, FOG computing, Bluetooth Low Energy sensors, Modular Cloud services, Internet of Things gateway

I. Introduction

The idea of the Internet of Things (IoT), combined with Cloud computing, opens new horizons in the field of real time data collection and analysis. The use of wearable sensors and their capability of Internet connectivity provides significant benefits in the area of healthcare where organizations allow a continuous, easy and efficient monitoring of patients´ vital data from anywhere, leading to cost savings such as equipment and nursing staff [3, 6]. Also patients themselves can now easily and safely perform their daily activities, wearing the appropriate sensors feeling safe and secure. In this work, we propose the implementation of an IoT gateway service that takes advantage of state-of-the-art technologies and smartphones capabilities to communicate and process data acquired by wearable Bluetooth Low Energy (BLE) [1] devices and sensors. It transforms a smartphone to a gateway allowing continuous and fast communication of vital data with the Cloud wherein third party users (such as doctors or medical personnel) can access easily and on demand.

The service allows the collection, processing and storage of vital data that the BLE sensors produce (e.g. blood-oxygen saturation level and pulse rate) as well as the transfer of useful information to the Cloud through the Internet. Firstly, an internal (on the gateway) processing of the signal data that are received from BLE sensors takes place in the example of FOG computing[2]. This gateway service is dynamic allowing automatic discovery and registration of new sensors (each sensor is declared by its XML schema) and, in addition, has the ability to recognize and register new characteristic values of BLE sensors (e.g. pulse rate in addition to blood-oxygen saturation) by editing their XML schema.

Building upon principles of service oriented design, it takes full advantage of Cloud services for processing potential big data streams produced by an ever-increasing number of users and sensors while being expandable with new features without effecting existing ones. The use of Cloud services, allows for central (on the Cloud) collection, communication, storage and further processing (e.g. big-data analysis) of sensor readings as well as of users' information in conjunction with several additional functions (potentially available in the form of modular Cloud services) helping in the overall system extensibility and interoperability with other systems.

Section II presents the background, Section III the IoT gateway service architecture and implementation, Section IV the performance analysis and Section V conclusions and future research directions.

II. Background

This work is focused on the implementation of an IoT gateway service for BLE devices. The solution is based on a Service Oriented Architecture (SOA) that makes use of modular services implementing fundamental functionalities communicating with each other over the network (typically using http protocol) and offering important benefits, such as multi-tenancy, increased accessibility and security through powerful APIs for seamless application integration. The design is based on our earlier work [4, 6] and we use the BLE standard to support low latency and short-range networks (less than 50m) for low-power devices [5]. BLE includes two core protocols referred to as ATT (Attribute Protocol) and GATT (Generic Attribute Profile).

ATT is a wire application protocol; it provides the basic information structure, the information discovery process from a remote device and the ability to interact with the generated information.

[1] https://www.bluetooth.com/specifications/bluetooth-core-specification

[2]https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf

GATT is built on top of ATT and defines how devices that follow the BLE protocol can transfer data when paired. It uses the attributes provided by ATT, organizing them into a hierarchy of data structures. It further divides information into logical pieces (e.g. heart rate service) and the "features" that are information specific to a particular sensor (e.g. heart rate measurement). The format includes the Generic Access Profile (GAP) that defines the mechanism for BLE devices to communicate with each other, by broadcasting (get data) or observing (listen to data).

We utilize FIWARE Enablers that are open source and Cloud services that offer open APIs for interconnection [1]. The FIWARE platform comprises of a set of cloud enablers on OpenStack [7] that are considered general purpose and common to several current and future usage areas. In particular, developers build applications by utilizing already deployed services offered as Cloud enablers encapsulating common functionalities (e.g. user authentication, data storage, context data management etc.), instead of re-engineering and implementing services from scratch. Cloud enablers are executed over virtualized hardware or other software stacks that can be highly scalable according to the needs of the users (e.g. elasticity in terms of resources). In addition, they

- Share common physical resources with other Cloud applications or services, thus supporting multi-tenancy.
- Provide interfaces using APIs that allow combination and remote management.
- Utilize 3rd party services that are decentralized.
- Are interoperable but could be based on different semantics e.g. different models with regards to communication (i.e. JSON) or protocols (i.e. BLE etc.).

III. IoT Service Architecture

In this section we present the IoT service architecture for BLE sensors based on a generic reference model. Fig. 1 demonstrates the service oriented reference architecture for IoT data collection and forwarding to the Cloud.

The producers are the users and their devices, the front-end is the data generator point, the back-end is the system logic including functionalities like storage, user authentication and publish/subscribe services that allows on demand data fetching from third user parties. Finally, the consumers are users registered to the back-end for getting particular data. In our case the "Sensor Data Collector" module implements the front-end system for BLE devices. This is comprised of a set of modular services that develop the key functionalities of the system such as data collection using BLE protocol, local storage within the smart phone device, notification service, back-end connectivity and system logic

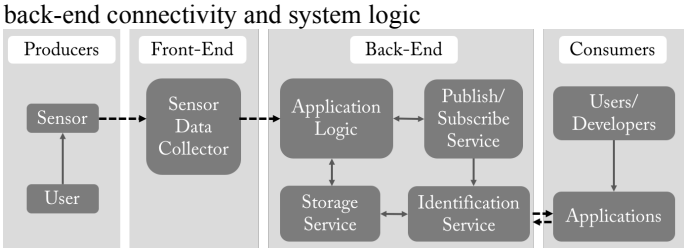

Fig. 1. The reference architecture of the service oriented architecture for data collection from IoT devices

Fig. 2 demonstrates the architecture of the BLE IoT service following the reference model of Fig. 1 and includes the next:

(a) The producers that are the BLE devices and the users with the smart phones. These generate data in real time that are captured and encoded into the desired format (i.e. JSON) from the front-end system.
(b) The gateway includes front-end services such as notification service, sensors data collector service, connectivity service, local storage and the application logic. The gateway establishes a GATT connection with the sensors while it uses the notification and the GCM service for forwarding alerts to the user.
(c) The Cloud (back-end) includes the connectivity service, the identity management, the local storage (separated over two database systems such as the data historic values from sensors and JSON storage service), the identity management service and the application logic.
(d) The consumers are the third party users and services accessed in the form of Web applications. The consumers can subscribe to certain sensors and get notified when new data is available.
(e) The technicians (administrator users) can register sensors by defining an XML schema (as detailed in Section IV). Also, they setup the system with regards to which devices are associated with certain users.

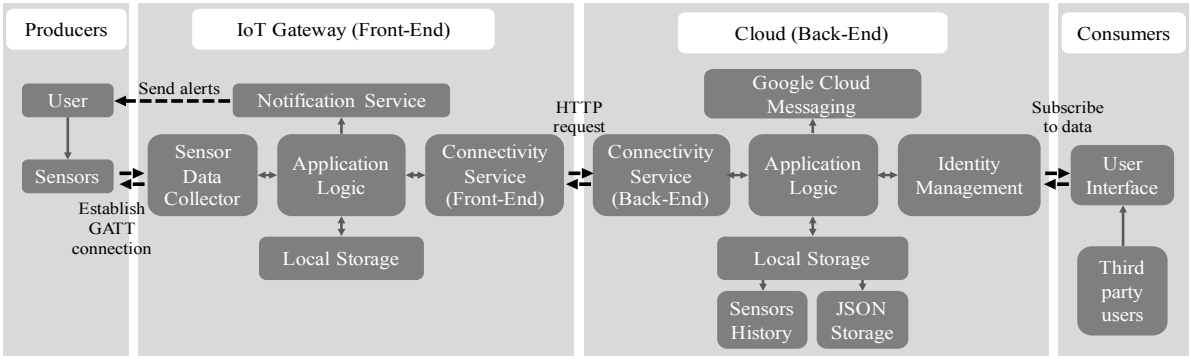

Fig. 2. Architecture of the BLE IoT Service data collection and processing.

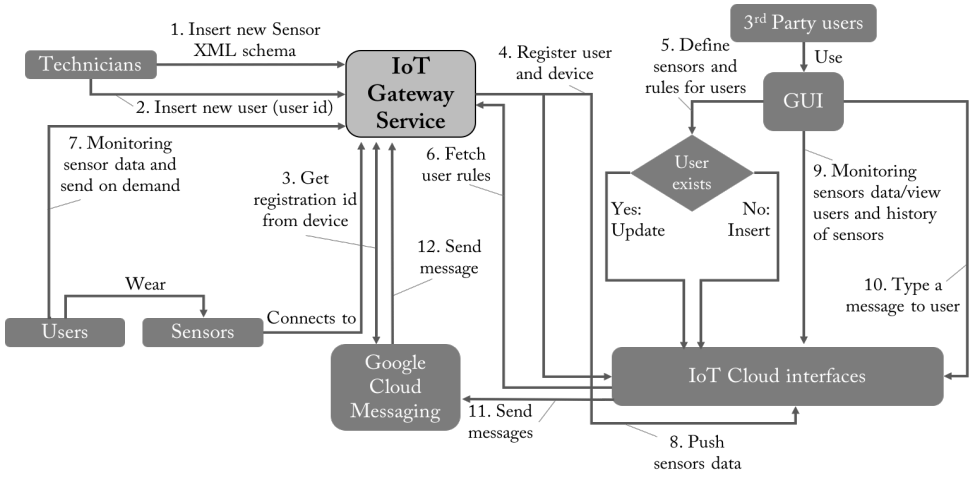

Fig. 3. The data flown of the IoT gateway service

IV. IoT Services Implementation

Fig. 3 presents the data flow and data transfer in a logical time sequence. We present the use case of a patient monitoring case from a doctor, where the patients are the service users and the doctors are the consumers. This includes the following

(a) Registration of a new sensor: An XML schema describes each sensor. The system administrator accesses and modifies the XML schemas of the sensors (as presented in Appendix). These hold information for the sensors so to be recognized by the gateway service.

(b) Registration of a new user: The system administrator uses the authentication mechanism to register new users to the service using the email as registration identifier (id). The user's profile is stored locally and in the Cloud authentication service.

(c) Request to the GCM service (detailed in Section IV): The service provides identification per device using a device identifier (id) from the GCM service.

(d) User and device registration in Cloud: The service uses the local storage (in JSON format) for storing data associated with a user identifier (user email address). The data are forwarded to the Cloud using an HTTP post command (following the gateway REST API).

(e) Sensor assignment and user rules: This is used to associate sensors to user and to assigns rules applicable to each sensor. For example, in a healthcare scenario, medical personel can define sensor category (e.g. Nonin pulse oximeter [3]), and set lower and upper values for sensor measurements, thus to be notified when these values are violated using GCM service.

(f) Rules for mobile devices: The service transfers the rules from back-end to front-end based on an HTTP post request using as parameters the user id, and an appropriate identification code to get rules and sensors related to the specific user and store them in the device's local storage.

(g) User Interaction: The user does not require making any configuration, as the service is automated and dynamic. He/she only has to wear a portable BLE sensor and carry a mobile device. The sensor is automatically paired to mobile device. Then the service can monitor sensor data in real time.

(h) Data forwarding to the Cloud: The data that is forwarded to the Cloud include

- Device Name identifier (i.e. the sensor identifier name) and Measurements i.e. name and value of the measurement).
- Measuring Time: It refers to the date format (year-month-day) and time (hours: minutes: seconds) information model including the measurements interval. Average sensor values are transmitted to back-end per regular intervals or on demand. We distinguish between two modes operation namely a) the smooth mode: The measurement values generated by sensors are within the range of the maximum and minimum threshold and are transmitted to back-en per regular time intervals and b) the risk mode: The measurement values generated by sensors violate one of the rules and are transmitted to back-end immediately. In addition, the use (patient) has to option to transmit data to back-end on demand (on his/her own initiative).
- The violation rule: In this case we have a rule violation, so the service activates the automated measurements per second.

(i) Monitoring measurements in real time to the Cloud: One of the most important functions of the system is the fact that the third party user has the option to monitor real-time measurements of the users.

(j) Messaging to user: GCM forwards messages to the user in real time. It offers a messaging middleware for information exchange (e.g. text messages from third party users to end users).

[3] http://www.nonin.com/PulseOximetry

A. The XML Schema to add new sensor

The service allows the definition of a generic XML schema that accepts the recognition of attribute values of sensors based on the general characteristics profile (GATT). The description of the XML schema includes the following information:

- <sensor>: This tag includes the BLE devices that are registered into the system.
- <device>: A specific sensor which is characterized by its category (e.g. Polar H7).
- <values>: Indicate the measurement type and the measurement units.
- <format>: It is the format of produced values (e.g. FORMAT_UINT8 that is an 8-bit integer).
- <position> & <multi>: In some cases, BLE sensor manufacturers define complicated function for decoding their features.

An example XML schema is presented in the Appendix section (for two BLE sensors).

B. Sensor data collector

The sensor data collector, shown in Fig. 4, collects data from the device based on the XML as described in the Appendix. It uses the "Device Scan Activity" that receives and stores an appropriate data structure of the XML schema for BLE sensors. Before that, it has been converted to a Java object through a Java document object model parsing process with the aim of identifying attribute values of sensors. Then, it activates the mechanism for detecting peripheral devices to determine which peripheral devices advertise data to the service. Next phase involves a decoding of information that includes identifying the measurement, its value and the unit of the measurement. The sensors generate data per second, using a broadcast receiver, and information is forwarded to the application logic.

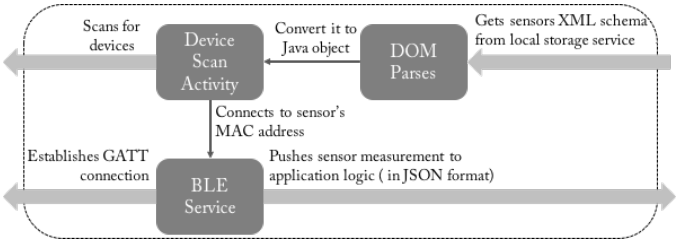

Fig. 4. Sensor Data Collector internal processes

C. Local storage

The local storage allows insertion, deletion and update of data and it includes the database manager, the data encryption and decryption and the SLQLite [4] database engine as shown in Fig. 5. Firstly, the database manager defines the appropriate tables for different types of data and predefined queries such as the "add sensor measurement" which stores the measurement of a sensor, the "delete sensor measurement" for deletion of a specific sensor, the "return sensors" that returns the identifier of a sensor. The "data encryption and decryption" is a component that runs on top of local storage for decoding/encoding data based on the Advanced Encryption Standard (AES) [1, 2]. Finally, the SQLITE database includes tables such as the "user device" that is comprised by the user id and the paired device (email and device registration id), the "user rules" that includes the user id (email) and sensors (device name id) along with the rule thresholds, the "sensor measures" that store data locally in case of failure in communication. The service contains the so called "asset folder" that includes the XML format of each sensor.

Data can be stored locally in the SQLITE database and are deleted upon transmission to the JSON cloud storage, for example data can be stored on local SQLITE storage when WIFI connection is lost and transmitted back when connection is up again.

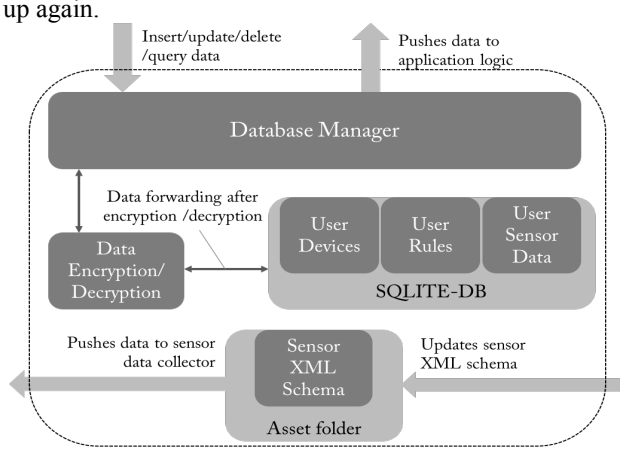

Fig. 5. The Local Storage internal processes

D. Connectivity Service

The Connectivity Service is the communication channel between the gateway service and the Cloud. Data is transformed to JSON and then with the suitable asynchronous calls are send securely to the Cloud. The connectivity service operations include (a) the recovery of device registration identifier from the Cloud (b) it sends the registration id to the Cloud using the appropriate HTTP headers, (c) it downloads rules per user using a post API call so data is forwarded to the application logic, which in turn transfers it to the local storage for further use, and (d) it sends sensor data using the on time interval, the on demand or the violation rule functions.

E. Notification Service

The notification service informs the users with appropriate prompts namely "toast messages" for Android devices and "alerts" for important changes occurring in the flow of operation of the service. The notification service works independently from GCM service that handles user prompted communication.

F. Google Cloud messaging (GCM)

The Google Cloud messaging [5] is a free service that allows developers to send messages to multiple platforms including

[4] https://www.sqlite.org/

[5] https://developers.google.com/Cloud-messaging/

Android and iOS. It is comprised by a GCM server deployed in the cloud and a GCM client that is part of the front-end service. The steps to send a message are as follows:

- The smart phone service sends an HTTP request to the GCM server using credentials designated by Google via AppEngine. The Google Connection server responds to the device with a message and assigns a unique Registration ID to it.
- Then the client service sends the Registration ID to the backend (application logic) for later use. The server stores the registration id of the device to a local database until the user decides to send a message. In our case we store it in the JSON storage GE.
- Using the Registration ID along with the text message, it makes a request to the GCM server to send in information as message to the application using a secured way as it can keep the message until the device is available online.

G. Application Logic (Gateway service)

The application logic is the centralized module that orchestrates services in sections B and E above. The service includes (a) the management module for comparing the measurements produced by the sensors and the rules stored in the Cloud, (b) a module for processing measurements that calculates the average values and periodically, (c) the Cloud data submission module where the user select the condition for data collection (on time-interval, on demand, and on rule violation) and (d) the system synchronization module.

H. Backend JSON Storage:

The backend JSON Storage service is based on MongoDB[6] that is a NoSQL Database. It is accessible as a RESTful API.

I. Identity Management

The Identity Management offers the authentication mechanism. We use it to integrate identity management into the sensor data collector and to authenticate and authorize the access to the system. It includes an interface that secures communication among the various actors of the system.

V. Experimental Evaluation

To explore the performance analysis of the IoT gateway service we measure the time required for data transmission using the REST API including service belonging to the front-end (notification service, sensor data collector, application logic, connectivity service and local storage service) and the Cloud that are the back-end services (connectivity services, GCM, application logic, identity management and data storage). The experimental configuration includes a gateway service that is installed in a smart phone device (running Android) and the back-end services that are deployed on OpenStack and on the FIWARE platform following the system of Fig. 2. Fig. 6 demonstrates the HTTP request/response times for the IoT gateway service. In particular, the method called "data transmission" is responsible for transferring sensor data to the Cloud using the gateway service. For experimental purposes we used the Polar H7 [7] for measuring pulse rates and the Nonin Onix 3250[8] for measuring oxygen saturation (SpO2) and pulse rates. Both sensors generate new measurements every 1 second.

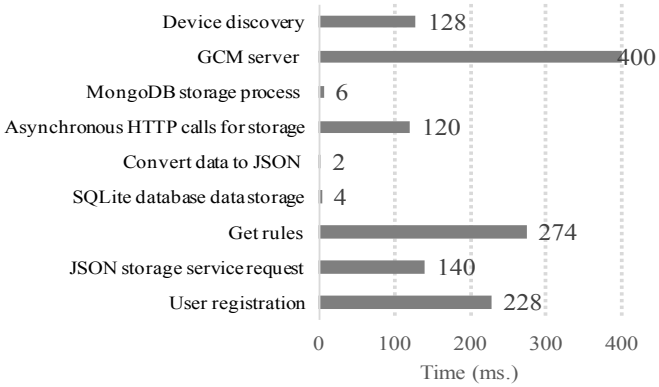

Fig. 6. IoT gateway service times for the different HTTP call methods

The values reported in this experiment are average over 100 method executions and account for the times for signal transmission and storage. We report measurements for the following methods:

a) The method "user registration" to register a new user and his/her gateway device to the Cloud. Every time a new registration happens, the device requests a new id from the GCM server that is also registered into the application logic of the system. The HTTP calls for this action require 228 milliseconds (ms) on the average. The request is forwarded to the JSON storage service using an HTTP post request that takes 140 ms, so data is stored to the local storage (an action that requires 4 ms). Based on this measurement, the whole process for this method takes 372 ms on the average. This method is executed only once per user.

b) The method "get rules" retrieves rules from the Cloud using the appropriate HTTP post call. This takes 274 ms on the average, then the method stores data to SQLite database (i.e. the internal database of the gateway device) an operation that takes 4 ms. The service converts data to JSON format in an average time of 2ms. The service uses an asynchronous HTTP call to push data to the Cloud local storage in an average time of 120 ms. The local storage processing time is calculated in 6 ms. Based on these measurements, the whole time process for the "data transmission" method is calculated to 128 ms. We consider this time significant low, bearing in mind the number of interacting services and the overhead that may be involved.

c) The "message notification" method sends messages through GCM to the device. The user submits the main text and the device's registration id that is send to GCM server (400 ms), which in turn discovers the device for which the message should be forwarded (an operation that takes 128 ms). The whole processing time is 528 ms on the average.

[6] https://www.mongodb.org

[7] http://www.polar.com

[8] http://www.nonin.com/Finger-Pulse-Oximeter/Onyx-Vantage-9590

To measure the effectiveness of the data storage in the MongoDB database, we simulated 100 users that store data every second (each data row includes 3 records of data). We used simulated (but realistic) data generated from real time data of the devices by adding some noise. We deployed the MongoDB in OpenStack using a virtual machine (VM) with 4 CPU Cores, 4 GB RAM and 40 GB hard disk. We also measured the utilization levels of the VM with regards to the CPU percentage, Virtual memory percentage and disk usage percentage. We calculated the average run for 1 hour as follows. The CPU percentage is averagely 7%, the virtual memory is 69% and the disk usage in 55%. The CPU percentage does not show an increasing trend, while the virtual memory is slightly increasing meaning that at a certain point the system will need to scale up (however the trend line is slowly increased 67.75% to 68.75% for the selected experiment). Finally, the disk usage is slightly increased. We conclude that the system could support a high number of concurrent users while a medium flavor VM can manage the storage needs for the 2 selected sensors. In future, we aim to evaluate this in more detail thus to include the overhead coming from the network (i.e. send the request to MongoDB from an external point).

VI. Conclusions

We focused on developing a dynamic (with universal BLE sensors support) IoT data collection service for BLE devices. We implemented the system using general-purpose services. The characteristics of the proposed service are: flexibility when adding new sensors via an XML schema, ability to monitor vital measurements produced by the BLE sensors from the users and consumers, smart mode of data transmission to the Cloud including on time interval, on demand-violation and rules. We provided an experimental analysis and we concluded that the data transmission method is relatively fast (taking less than 130 ms). Future work includes, exploring more performance features, for example to evaluate the number of spontaneous sensors and users that the system can handle as well as, exploring how the system performance is affected when it scales up by adding new MongoDB databases.

References

[1] Theodore Zahariadis, Andreas Papadakis, Federico Alvarez, Jose Gonzalez, Fernando Lopez, Federico Facca, and Yahya Al-Hazmi. 2014. FIWARE Lab: Managing Resources and Services in a Cloud Federation Supporting Future Internet Applications. In Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC '14). IEEE Computer Society, Washington, DC, USA

[2] Treffyn Lynch Koreshoff, Toni Robertson, and Tuck Wah Leong. 2013. Internet of Things: A Review of Literature and Products. In Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration. Haifeng Shen, Ross Smith, Jeni Paay, Paul Calder, and Theodor Wyeld (Eds.). ACM, New York, NY, USA, pp. 335-344.

[3] Stelios Sotiriadis, Euripides G.M. Petrakis, Stefan Covaci, Paolo Zampognaro, Eleni Georga and Christoph Thuemmler, An architecture for designing Future Internet (FI) applications in sensitive domains: Expressing the software to data paradigm by utilizing hybrid Cloud technology, in IEEE 13th International Conference on Bioinformatics and Bioengineering (BIBE), 2013 pp.1-6, 10-13 Nov. 2013

[4] Alexandros Preventis, Kostas Stravoskoufos, Stelios Sotiriadis, and Euripides G. M. Petrakis. 2014. Interact: Gesture Recognition in the Cloud. In Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC '14). IEEE Computer Society, Washington, DC, USA, pp. 501-502.

[5] Christopher J. Hansen. 2015. Internetworking with Bluetooth Low Energy. GetMobile: Mobile Comp. and Comm. 19, 2 (August 2015), pp. 34-38.

[6] Konstantinos Douzis, Stelios Sotiriadis, Euripides G.M. Petrakis, Cristiana Amza, Modular and Generic IoT Management on the Cloud, Future Generation Computer Systems, Available online 18 June 2016.

[7] OpenStack, Open source software for creating private and public clouds, Available online: https://www.openstack.org, Accessed 3 August 2016.

Appendix

The XML schema for adding BLE sensors into the gateway. It includes the Nonin Onix sensor and the properties oxygen percentage range in blood (SpO2) and heart rate (pulse_Rate) and the Polar H7 with the property “heart rate” (pulse_Rate).

```
<? xml version="1.0" encoding="UTF-8" ?>
  <sensors>
        <device name="NoninOnix">
            <uuid id="0aad7ea0-0d60-11e2-8e3c-0002a5d5c51b">
                <values name="SPO2" type="%">
                    <format>FORMAT_UINT8</format>
                    <position>7</position>
                    <multi>1</multi>
                </values>
                <values name="pulse_Rate" type="bpm">
                    <format>FORMAT_UINT8</format>
                    <position>[8,9]</position>
                    <multi>[256,1]</multi>
                </values>
            </uuid>
        </device>

              <device name="Polar H7">

            <uuid id="00002a37-0000-1000-8000-00805f9b34fb">
                <values name="pulse_Rate" type="bpm">
                    <format>FORMAT_UINT8</format>
                    <position>1</position>
                    <multi>1</multi>
                </values>
            </uuid>

        </device>
 </sensors>
```