DOI. 10.1109/ICUMT.2017.8255160

Application layer protocol for IoT using Wireless Sensor Networks communication protocols

Valentin Stangaciu, Madalina Stanciu, Loredana Lupu, Mihai V. Micea, Vladimir Cretu
Department of Computers and Information Technology
Politehnica University Timisoara
Timisoara, Romania
valentin.stangaciu@cs.upt.ro, madalinastanciu12@gmail.com, loredanalupu349@yahoo.ro,
mihai.micea@cs.upt.ro, vladimir.cretu@cs.upt.ro

***Abstract*— This paper aims at providing a communication architecture in order to enable the integration of traditional Wireless Sensor Networks into the Internet of Things paradigm. We focus especially on a communication protocol for providing connectivity between the smart objects and a central IoT hub. From the implementation and testing of the proposed IoT protocol, we measured the smallest memory footprint, which demonstrates that such a protocol may be easily integrated into smart objects represented by small embedded systems low on hardware resources**

Keywords—Internet of things, wireless sensor networks, real-time applications, real-time systems, communication protocol.

I. INTRODUCTION

During a very short period of time, a new research domain and paradigm has appeared, represented by the concept of Internet of Thing (IoT). This concept aims at integrating all smart devices, sensors and general appliances into a network and providing high connectivity. Furthermore, each device, usually designated as smart object [1] , is connected to such a network and usually has to be able to be uniquely identified, addressed and controlled by the user or by the system.

The study and the development of platforms integrating into the Internet of Things paradigm has been greatly stimulated by the high applicability opportunities such as Smart Homes. This concept has the ability to transform mainly the houses and the office buildings into user-aware and energy-aware environments [2]. The high connectivity of the smart objects is usually accomplished by Internet related protocol such as TCP or UDP, IPv4 or IPv6 [3].

Another important domain which received a high interest from the research community is represented by the Wireless Sensor Networks (WSN). Such a network is formed by numerous interconnected, battery powered, cheap devices, with low hardware resources but high sensing abilities. The idea of such networks is that with very small but numerous interconnected nodes, high complexity tasks may be performed. The WSNs are used in sensing and monitoring application for critical or non-critical environments [4, 5]. In such networks, the device interconnection is guaranteed by using dedicated protocols and standards for Wireless Sensor Networks such as ZigBee [6].

The main part of this paper will propose a communication protocol for providing connectivity between the smart objects and a central IoT hub. Our main idea is to provide an architecture to enable the integration of Wireless Sensor Networks into the new Internet of Things concept. We also present initial data related to the implementation of the proposed protocol in a real IoT network.

II. RELATED WORK

As stated before, one of the highest applicability domain for Internet of Things is represented by Smart Homes or Home / Office Automation. In such environments many of the current implementations and concepts are oriented on classic Internet communication protocols. Many of the smart objects are connected using Wifi through classical access points or by using the standard Ethernet infrastructure [2]. In many of the approaches, central hubs are used to provide the interconnection of various types of devices using several communication interfaces in order to offer maximum connectivity [7].

The user interaction in many of the current IoT implementations is performed by various cloud services, which have become a common practice. Users are encouraged to access their IoT devices through cloud services. Furthermore, the data storage and analysis is performed using cloud computing [8] and interfaces using Web Services [9]. In many of such implementations the communication architecture is based on Wifi and Internet.

Even though the WLAN communication method is preferred in most implementations of IoT networks, other communication method have been suggested from the early stages of the IoT paradigm [10]. Wireless Sensor Networks are important candidates for joining the Internet of Things concept by adapting the communication protocols [11] and proposing new architectures for ad-hoc sensor networks in order to be integrated [12].

This work was supported by a grant of the Romanian National Authority for Scientific Research and Innovation, CNCS – UEFISCDI, project number PN-II-RU-TE-2014-4-0731.

In order to further maximize the communication capabilities, more daring solutions have been proposed using Cognitive Radio [13] which a novel approach in radio communication but needs further research investment.

III. System Architecture

The communication architecture of the system that this paper will be based on has two main components: the smart devices and the central IoT Hub.

The smart devices are normally small embedded systems, low on hardware resources but with high sensing or control capabilities. Each smart device has wireless communication capabilities implemented using the ZigBee stack [6]. In our perspective the smart device is actually the node of a wireless sensor network.

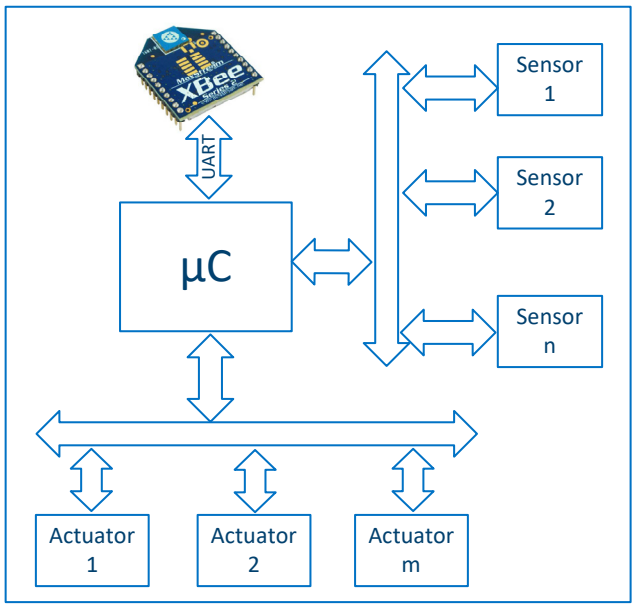

Fig. 1. Smart object block diagram

The block diagram of the smart devices is presented in Fig. 1 where the main component is a microcontroller which manages the whole sensor node. This component is mainly used to acquire the sensing information from the connected sensors and if needed to control the actuators when present as well as to implement the communication protocol. Depending on sensing/control role of each sensor node, a number of similar/different sensors or actuators may be present on the sensor node. From a black box perspective, the sensor nodes may be seen identical but exporting their capabilities through the communication protocol.

The communication is assured by XBee Series 2 modules [14], which are ZigBee certified and rely on the IEEE 802.15.4 Standard [15] for medium access protocol. The communication stack provides all the necessary levels offering medium access, routing and application level.

Each smart object node shall be uniquely identified by a 64 bit serial number provided by the XBee Series 2 module. The uniqueness of the serial number is guaranteed by the producer.

At the software level, in order for the smart object to be able to also function in an environment with time restrictions, we have chosen a hybrid real time operating system for small embedded devices [16]. Such an operating system supports different types of task scheduling and offers guarantees that the task will be executed according to their time constraints.

All the smart objects form a ZigBee network which, according to the standard, has to be coordinated by a ZigBee Coordinator node. This coordinator node shall be connected to an Embedded IoT Hub. Considering these aspects the general system architecture is similar to the one presented in the figure Fig. 2

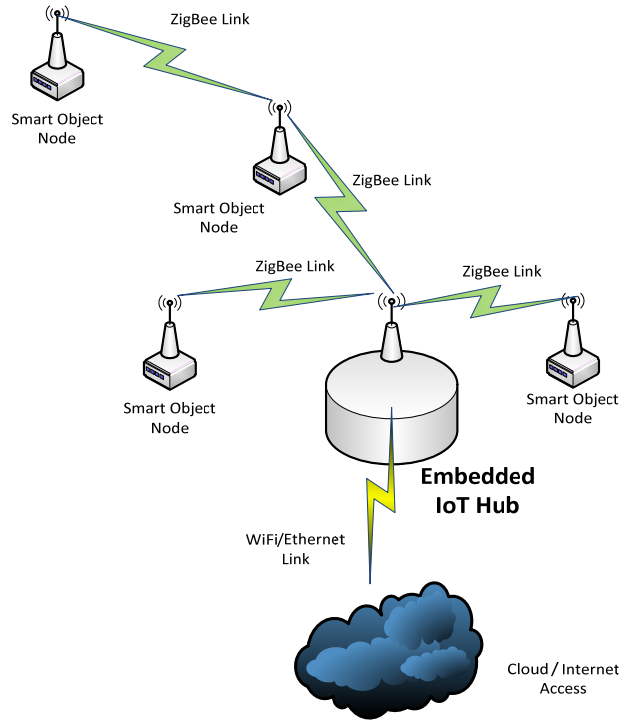

Fig. 2. General System Architecture

The Embedded IoT Hub has the main role to manage the ZigBee wireless sensor network through a Coordinator Xbee module that shall be connected to it. Another important role is to communicate with the smart objects in order to request data, request parameter change or to control the various actuators within the smart objects. The IoT Hub device will also have the role of a sink node (acting like a data concentrator), providing storage for the data sent by the smart objects.

Another communication interface of the IoT Hub will provide the communication with the exterior through classical Internet access. This function is mandatory in order to provide the interface for uniquely accessing each smart object remotely. Practically this function is responsible for implementing the Internet of Things paradigm.

This paper shall not concentrate on presenting how the IoT Hub will be accessed from the exterior but it will be oriented on providing the data encoding and communication protocol available over the ZigBee link. In the next sections the communication protocol between the smart objects and the IoT Hub shall be presented.

IV. Communication Protocol over the ZigBee Link

The communication protocol between the IoT Hub and the smart objects must be aware of some restrictions that are imposed by the limitations of the XBee module and the smart objects themselves. The most important restriction imposed by the XBee module refers to the fact that the maximum amount of data that the XBee device can transfer is of 72 bytes [14]. Considering this aspect the protocol must offer means of data segmentation.

Furthermore, from a communication point of view, the smart objects will be considered identical and the protocol must encode the data in order to distinguish between the different types of sensors and actuators controlled by the smart device.

Along with these considerations, the proposed communication protocol has been designed as having a highly used smart metering protocol as a starting point. The smart metering protocol considered is represented by the DLMS/COSEM (Device Language Message Specification / Companion Specification for Energy Metering) suite which is standardized by the International Electrotechnical Commission [17, 18]. The data encapsulation shall be based on the encoding specified by the Abstract Syntax Notation ASN.1 standard using the Basic Encoding Rules format [19]. Such an encoding is ideal for data structure serialization.

According to the standard, the main idea behind the Basic Encoding Rules (BER) may be summarized to the following general form [20]:

TABLE I. BER GENERAL FORM

TAG	DATA

The Tag field represents a byte specifying the type of the data that follows in the packet. The data field represents the actual data. As an example we will consider encoding a 32 bit unsigned integer of value 0x0011223344 in hexadecimal. The data structure specified by the standard suitable for the encoding of such a number is *DoubleLongUnsigned* having tag 0x06 (hexadecimal). The encoding of this data in conformance with the BER encoding is represented in

TABLE II. BER ENCODING EXAMPLE

TAG	DATA	Resulted encoded value
06	0011223344	060011223344

The basic data structures that shall be supported by the protocol will be referred to as *GenericData* and will contain data types and structures that are described by the ASN.1 standard. The encoding method of the data types are ASN.1 complaint with the only difference that the tags have overridden according to the rules specified by the standard. The *GenericData* shall have the following definition using Abstract Syntax Notation:

```
GenericData ::= CHOICE {
      BERNullData            [0]       IMPLICIT NULL,
      BERArray               [1]       IMPLICIT SEQUENCE OF Data,
      BERStructure           [2]       IMPLICIT SEQUENCE OF Data,
      BERBoolean             [3]       IMPLICIT BOOLEAN,
      BERBitString           [4]       IMPLICIT BIT STRING,
      BERDoubleLong          [5]       IMPLICIT Integer32,
      BERDoubleLongUnsigned  [6]       IMPLICIT Unsigned32,
      BEROctetString         [9]       IMPLICIT OCTET STRING,
      BERVisibleString       [10]      IMPLICIT VisibleString,
      BERInteger             [15]      IMPLICIT Integer8,
      BERLong                [16]      IMPLICIT Integer16,
      BERUnsigned            [17]      IMPLICIT Unsigned8,
      BERLongUnsigned        [18]      IMPLICIT Unsigned16,
      BERlong64              [20]      IMPLICIT Integer64,
      BERLong64Unsigned      [21]      IMPLICIT Unsigned64,
      BEREnum                [22]      IMPLICIT Unsigned8,
      BERRawData             [254]     IMPLICIT OCTET STRING,
      BERDontCare            [255]     IMPLICIT NULL
}
```

Code Sequence 1. Definition of GenericData

The protocol shall support the following types of packets (IoTPDU):

- *IoT_GetDataRequest* – this IoTPDU is sent by the IoT Hub to a smart object in order to request the data read from the sensors
- *IoT_GetDataResponse* – this IoTPDU is sent by the smart object to the IoT Hub as a response to a *GetDataRequest* packet, containing the acquired data, encoded in BER format
- *IoT_GetParameterRequest* – this IoTPDU is sent by the IoT Hub to a smart object requesting the value of a specified internal parameter
- *IoT_GetParameterResponse* – this IoTPDU is sent by the smart object to the IoT Hub as a response to a *GetParameterRequest* containing the requested information
- *IoT_SetParameterRequest* – this IoTPDU is sent by the IoT Hub to the smart object requesting the node to set the value of a specified parameter
- *IoT_SetParameterResponse* – thie IoTPDU is sent by the smart object to the IoT Hub with the response for the previous *SetParameterRequest* IoTPDU
- *IoT_CallActionMethodRequest* – this IoTPDU is sent by the IoT Hub to the smart object requesting a remote call of a smart object's method
- *IoT_CallActionMethodResponse* – this IoTPDU is sent by the smart object to the IoT Hub as a response to a previous *CallActionMethodRequest* IoTPDU

Using an Abstract Syntax Notation style the proposed IoTPDU can be described as following:

```
IoTPDU ::= CHOICE {
    IoT_GetDataRequest          [20]      NULL
    IoT_GetDataResponse         [21]      GetDataResponsePDU
    IoT_GetParameterRequest     [22]      GetParameterRequestPDU
    IoT_GetParameterResponse    [23]      GetParameterResponsePDU
    IoT_SetParameterRequest     [22]      SetParameterRequestPDU
    IoT_SetParameterResponse    [23]      SetParameterResponsePDU
    IoT_CallActionMethodRequest [24]      CallActionMethodRequestPDU
    IoT_CallActionMethodResponse [25]     CallActionMethodResponsePDU
}
```

Code Sequence 2. Definition of IoTPDU

The *IoT_GetDataRequest* IoTPDU is not designed to receive any paramters thus no selection of the requested data is needed in this stage.

The *IoT_GetDataResponse* IoTPDU is designed to have the following format:

```
GetDataResponsePDU ::= SEQUENCE {
    IsMultiPacketDataPDU        BERBoolean,
    TotalPDUPackets             BERUnsigned,
    CurrentPDUPacketNumber      BERUnsigned,
    IsLastPacketOfPDU           BERBoolean,
    DataSize                    BERLongUnsigned
    NodeDataPacket              NodeData,
}
```

Code Sequence 3. Definition of *GetDataResponsePDU*

The *IsMultiPacketDataPDU* informs the receiver whether this current *GetDataResponsePDU* is part of a multi-packet transmission. This situation may occur when the amount of data to be transmitted over the ZigBee link exceeds the maximum allowed of 72 bytes. In the same context are defined the members: *TotalPDUPacket* – specifying the total number of *GetDataResponsePDU* to be expected; *CurrentPDUPacketNumber* – specifying the current id of the PDU packet and IsLastPacketPDU a Boolean values specifying whether this is the last packet of the segmented PDU. These members are to be used by the receiver to correctly assemble the *NodeDataPacket* member if PDU segmentation is needed. The *DataSize* member specifies the size in bytes of the BER encoded data contained in the *NodeDataPacket* member.

The NodeDataPacket member contains the actual BER encoded acquired measurement data from the smart object. This is defined as an ASN.1 sequence as follows:

```
NodeData ::= SEQUENCE {
    NodeID                      BERLong64Unsigned,
    SensorDataArray             SEQUENCE OF SensorDataElement,
    NodeLocation                NodeLocationElement
}
```

Code Sequence 4. Definition of *NodeData* member

The *NodeData* member describes a data packet with the actual measurement transmitted via ZigBee link by the smart object. This structure shall contain the NodeID which is a mandatory node identification, the 64 bit number that uniquely identifies the node. The *SensorDataArray* member is a sequence of *SensorDataElement* members which contains the measurement data from each sensor within the smart object. The size of the *SensorDataArray* is equal to the number of sensors connected to the smart object. The latter member, *NodeLocation*, contains the location of the smart object.

The *SensorDataElement*, containing the actual measurement from a sensor shall have the following structure:

```
SensorDataElement ::= SEQUENCE {
          Timestamp                 BEROCtetString,
          SensorID                  BERUnsigned,
          SensorDataType            SensorDataTypeEnum,
          SensorDataObject          GenericData,
          MeasureUnit               MeasureUnitEnum,
          Scalar                    BERInteger
}
```

Code Sequence 5. Definition of *SensorDataElement*

The first member of this structure is represented by the *Timestamp* of the measurement. The *SensorID* represents the identification number of the sensor within the smart object. The *SensorDataType* is an enumerative member and defines the type of measurement contained. The *SensorDataObject* member contains the actual measurement value and depending on the enumerative value of *SensorDataType* it shall have one of the data types defined by the ASN.1 compliant *GenericData*. The *MeasureUnit* is also an enumerative value containing the measuring unit to be considered for the *SensorDataObject* value.

The *SensorDataType* is defined as enumerative type *SensorDataTypeEnum* and may be described as follows:

```
SensorDataTypeEnum ::= ENUMERATED {
    Voltage                     (0),
    Current                     (1),
    Temperature                 (2),
    Humidity                    (3),
    ActiveEnergy                (4),
    ReactiveEnergy              (5),
}
```

Code Sequence 6. Definition of *SensorDataTypeEnum*

The *MeasureUnit* is defined as enumerative type *MeasureUnitEnum* and may be described as follows:

```
MeasureUnitEnum ::= ENUMERATED {
    volt                        (0),
    ampere                      (1),
    degree_celsius              (2),
    percentage                  (3),
    watt-hour                   (4),
    var-hour                    (5),
    no-unit                     (255),
}
```

Code Sequence 7. Definition of *MeasureUnitEnum*

Both *SensorDataTypeEnum* and *MeasureUnitEnum* enumerative types have not been fully defined and thus they may be extended.

At this point, all the members of *NodeData* have been fully described leaving only *NodeLocation* to be detailed. *NodeLocation* is defined as a *NodeLocationElement* type in order to permit 2 ways of defining the location: as geographical coordinates or as a string designating its location. The type is presented as follows:

```
NodeLocationElement ::= SEQUENCE {
    NodeLocationType            NodeLocationTypeEnum,
    NodeLocationObject          NodeLocationObjectData,
}
```

Code Sequence 8. Definition of *NodeLocationElement*

The NodeLocationElement is used to specify the location of the node. In order to make a correct specification the type of location and the actual location is needed. The type of location

is designated by the member *NodeLocationType* and the actual location data by the member *NodeLocatioObject*. The *NodeLocationType* is defined as enumerative type *NodeLocationTypeEnum* and states if the location data is represented by the geographical coordinates or by a string. Depending of this enumerative value is the data type of the *NodeLocationObjectData* type. These data type shall be described as following:

```
NodeLocationTypeEnum ::= ENUMERATED {
    GeoCoordinates              (0),
    LocationString              (1),
}
```

Code Sequence 9. Definition of *NodeLocationTypeEnum*

```
NodeLocationObjectData ::= CHOICE {
    SensorLocationAsGlobalGeoCoordinates    SEQUENCE(SIZE(2)) OF BERInteger,
    SensorLocationAsString                  BEROCtetString,
}
```

Code Sequence 10. Definition of *NodeLocationTypeEnum*

At this point, the whole data packet represented by *NoteData* has been defined along with the most important protocol IoTPDU, *GetDataResponsePDU*. This type of packet may be transmitted by the smart object when requested by the IoT Hub using a *GetDataRequestPDU* or it may be transmitted unattended with a designated periodicity in seconds.

The next IoTPDU to be defined is *GetParameterRequestPDU* used by the IoT Hub to interrogate the value of a certain parameter of a certain smart object. This IoTPDU shall contain only an enumerative value designating the parameter as described in the following ASN.1 syntax:

```
GetParameterRequestPDU ::= SEQUENCE {
    ParameterType                           BEREnum,
}
```

Code Sequence 11. Definition of *GetParameterRequestPDU*

The *GetParameterResponsePDU* will be transmitted strictly by the smart object when a *GetParameterRequestPDU* is received from the IoT Hub. This response shall contain the parameter type same as the request and the actual parameter data:

```
GetParameterResponsePDU ::= SEQUENCE {
    ParameterType                           BEREnum,
    ParameterData                           GenericData
}
```

Code Sequence 12. Definition of *GetParameterResponsePDU*

The *SetParameterRequestPDU* is another IoTPDU that shall only be transmitted by the IoT Hub to a designated smart object in order to request a parameter value change. This IOTPDU shall contain the parameter type and the parameter value:

```
SetParameterRequestPDU ::= SEQUENCE {
    ParameterType                           BEREnum,
    ParameterData                           GenericData
}
```

Code Sequence 13. Definition of *SetParameterRequestPDU*

The *SetParameterResponsePDU* is the IoTPDU which provides a response from the smart object for a previous *SetParameterRequestPDU* received from the IoT Hub. This IoTPDU shall contain the type of the parameter which value was requested to be changed and the result of the operation, whether the change was successful or not:

```
SetParameterResponsePDU ::= SEQUENCE {
    ParameterType                           BEREnum,
    Result                                  BERBoolean
}
```

Code Sequence 14. Definition of *SetParameterRequestPDU*

This protocol was not only designed for transferring and reading or writing smart object parameters. It has also been designed for supporting remote method call. This procedure has been introduced mainly to request an action over the actuators that may be present on the smart object.

In order to remotely call a method of a smart object the IoT Hub must explicitly send a *CallActionMethodRequestPDU* to a designated smart object as described below:

```
CallActionMethodRequestPDU ::= SEQUENCE {
    MethodID                                BERUnsigned,
    MethodParameters                        GenericData
}
```

Code Sequence 15. Definition of *CallActionMethodRequestPDU*

The *MethodID* represents the identification number of the method provided by the smart object. If needed a field for parameters has been made available.

The smart object, after the reception of a *CallActionMethodRequestPDU* it must send the result of the remote method call using a *CallActionMethodResponsePDU* having the following structure:

```
CallActionMethodResponsePDU ::= SEQUENCE {
    MethodID                                BERUnsigned,
    Result                                  BERBoolean
    MethodAdditionalResponseData            GenericData
}
```

Code Sequence 16. Definition of *CallActionMethodResponsePDU*

This layer of the protocol does not define explicit measurement data, parameters, remote methods or method arguments. This layer of the protocol only offers a basis for a generic encoding which shall be offered to an upper layer protocol.

V. Discussions and Implementation Considerations

The proposed protocol has been implemented and tested on an IoT network with the same architecture as described in Fig. 2. The smart objects, forming the network, are extremely small embedded systems, having the architecture presented in Fig. 1, with an NXP LPC2138 [21] as a host microcontroller. As stated in the smart object architecture diagram, the communication is implemented using XBee Series 2 modules. The main reason for such a decision is the fact that such modules are suitable for small embedded systems.

TABLE III. Implementation parameters

Software component	Memory footprint (bytes)
Hybrid Operating system	30744
XBee Series 2 Library	7492
Protocol implementation libray	1670

In such a situation the memory footprint is critical, for both code and data memory. In TABLE III. the memory footprint of the main software components is presented. The largest memory footprint is represented by the hybrid operating system, which in our implementation, is necessary in order to provide a stable and predictable task execution environment.

Another mandatory software component is represented by the library managing the XBee module. Its main role is to implement the low level communication protocol with the module and provide the basic APIs for transmitting and receiving data.

As it can be observed, the lowest memory footprint belongs to our IoTPDU protocol implementation which demonstrates that such a protocol may be easily integrated into smart objects represented by small embedded system low on hardware resources.

VI. Conclusions and Future Work

In many of the situations the smart objects in a network are connected to an IoT Hub using TCP protocols through WLAN networks. In such architectures the WLAN communication medium is highly used even if the smart objects do not require such bandwidth. Our architecture is based on the fact that the smart objects communicate with the IoT Hub using Wireless Sensor Network oriented protocols such as ZigBee.

In this paper we present an application layer protocol for communicating between and IoT Hub and the smart objects through communication protocols designed for Wireless Sensor Networks. This proposed protocol is derived from smart metering communication protocols which have already been standardized. They have been adapted in order to fit the needs of IoT smart objects mainly by using the ASN.1 Basic Encoding Rules.

References

[1] C. Toma, C. CIUREA, and I. IVAN, "Approaches on Internet of Things Solutions," Journal of Mobile, Embedded and Distributed Systems, vol. V, no. 3, pp. 124 - 129, 2013.

[2] B. L. R. Stojkoska and K. V. Trivodaliev, "A review of Internet of Things for smart home: Challenges and solutions," Journal of Cleaner Production, vol. 140, no. pp. 1454-1464, 2017.

[3] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," Comput. Netw., vol. 54, no. 15, pp. 2787-2805, 2010.

[4] P. N. N. Reddy, P. I. Basarkod, and S. S. Manvi, "Wireless Sensor Network based Fire Monitoring and Extinguishing System in Real Time Environment," International Journal Advanced Networking And Application, vol. 3, no. 2, pp. 1070-1075, 2011.

[5] M. Marcu, C. Stangaciu, A. Topirceanu, D. Volcinschi, and V. Stangaciu, "Wireless Sensors Solution for Energy Monitoring, Analyzing, Controlling and Predicting Sensor Systems and Software." vol. 57, G. Par and P. Morrow, Eds., ed: Springer Berlin Heidelberg, 2011, ISBN: 978-3-642-23583-2, pp. 1-19.

[6] ZigBee Alliance, "ZigBee Specifications," version 1.0 r13 Dec 2006.

[7] V. Vujovi and M. Maksimovi, "Raspberry Pi as a Sensor Web node for home automation," Comput. Electr. Eng., vol. 44, no. C, pp. 153-171, 2015.

[8] N. S. Sirsath, P. S. Dhole, N. P. Mohire, S. C. Naik, and N. S. Ratnaparkhi, "Home Automation using Cloud Network and Mobile Devices," ITSI Transactions on Electrical and Electronics Engineering (ITSI-TEEE), vol. 1, no. 2, pp. 93-97, 2013.

[9] E. Bernardes, C. Barros, and A. d. R. L. Ribeiro, "An Self-configuration Architecture for Web-API of Internet of Things," in Proceedings of the 10th International Conference on Web Information Systems and Technologies Barcelona, Spain, 2016.

[10] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," Future Gener. Comput. Syst., vol. 29, no. 7, pp. 1645-1660, 2013.

[11] K. Kumar, S. Kumar, and O. Kaiwartya, "Distance, Energy and Link Quality Based Routing Protocol for Internet of Things," in Proceedings of the International Conference on Signal, Networks, Computing, and Systems, 2017, pp. 253-259.

[12] T. Qiu, N. Chen, K. Li, D. Qiao, and Z. Fu, "Heterogeneous ad hoc networks: Architectures, advances and challenges," Ad Hoc Networks, vol. 55, no. pp. 143-152, 2017.

[13] F. F. Qureshi, R. Iqbal, and M. N. Asghar, "Energy Efficient Wireless Communication Technique Based on Cognitive Radio for Internet of Things," Journal of Network and Computer Applications, no. in press, 2017.

[14] Digi International, "XBeeTM Series 2 OEM RF Modules Product manual v1.x.2x - ZigBee Protocol," Digi International Inc. 2007.

[15] IEEE Computer Society, "Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless personal Area Networks (LR-WPANs)," in 802.15.4d, ed: Institute of Electrical and Electronics Engineers, Inc., 2009.

[16] C. Stangaciu, M. Micea, and V. Cretu, "An Analysis of a Hard Real-Time Execution Environment Extension for FreeRTOS," Advances in Electrical and Computer Engineering, vol. 15, no. 3, pp. 79-86, 2015.

[17] International Electrotechnical Commission, "Electricity metering data exchange - The DLMS/COSEM suite - Part 1-0: Smart metering standardisation framework," in IEC 62056-1-0:2014, ed, 2014.

[18] International Electrotechnical Commission, "Electricity metering data exchange - The DLMS/COSEM suite - Part 5-3: DLMS/COSEM application layer," in IEC 62056-5-3:2016, ed, 2016.

[19] International Telecommunication Union, "Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER) " in Series X: Data Networks and Open System Communications, ed, 2002.

[20] J. Larmouth, ASN.1 Complete: Morgan Kaufmann, 2000, ISBN: 9780122334351.

[21] NXP Semiconductors, "LPC2131/32/34/36/38 Datasheet," NXP Semiconductors July 2011.