Environmental Monitoring Using Heterogeneous Wi-Fi and IEEE 802.15.4 Networks

Cristian Cocioabă, Dan Tudose
Computer Science Department
University POLITEHNICA of Bucharest
Bucharest, Romania
cristiancocioaba@gmail.com, dan.tudose@cs.pub.ro

Abstract—**Environmental monitoring is key for multiple applications and requires that devices used in acquiring environment data to be scattered over a wide area, but at the same time maintain accessibility of information from all sensor nodes. Although many Wireless Sensor Networks are based on the IEEE 802.15.4 standard for low-power, Wi-Fi networks offer the most accessibility and interoperability with other devices. We envision a system which integrates the two networks' main capabilities by monitoring data in a heterogeneous Wireless Sensor Network. Nodes can transmit sensor data over the Internet to other devices, server applications or cloud based solutions, making the whole process of environmental monitoring universally accessible.**

I. Introduction

Today, smart homes, smart cities, smart grids, intelligent transportation are infrastructure systems that connect our world more than we ever thought possible. For this vision, most representative is the concept of Internet of Thighs (IoT), in which the use of sensors to gather environment measurements is closely coupled with information and communication technologies. By using embedded devices, intelligent monitoring and management can be achieved, interconnecting them to transmit useful measurement information and control instructions through a distributed sensor network.

For environment monitoring, a Wireless Sensor Network (WSN) is used to detect physical phenomena such as light, heat, pressure, etc. A WSN consists of a large number of sensor nodes, each of them equipped with one or multiple sensors. WSNs are viewed as a revolutionary information gathering method, which is best suited to create complex communication systems.

In comparison with wired solutions, WSNs are easier to deploy and have better flexibility. With technological advance and the need to integrate more and more sensors and expand the monitored area, WSNs are becoming the key technology for IoT.

II. Related work

Environment monitoring has been around for a while and many hardware providers for Wireless Sensor Network focus on using 802.15.4 networks, mainly for their low-power. However, the offered solutions tend to lack accessibility and integration with common devices, such as smartphones, tablets or do not include easy cloud connectivity. Also, the high price of such implementations makes them even less attractive. Therefore, the concern is with finding cheaper and more community-oriented alternatives for monitoring the environment.

One of the most popular solutions for Wireless Sensor Networks is ZigBee [1], a protocol for wireless mesh networks based on the IEEE 802.15.4 standard. It was especially conceived for ultra low-power applications. ZigBee protocols are used in embedded applications that have low power consumption and do not require a large bandwidth.

The ZigBee protocol offers support for star, tree, and mesh network topologies, handling message routing between the nodes. Although a ZigBee node does not have a long transmission range, messages can travel to greater distances by adding other nodes to create a network and expand the coverage area.

ZigBee certified products tend to have a high price and do not include embedded sensors. Furthermore, for the applications to extract data from the ZigBee nodes, a hardware link with a node must exist. A solution to that problem is offered by a ZigBee IP Router [3], a gateway to Wi-Fi networks. However, this solution adds to the price of a system and it does not include integration with common devices.

Thread [4] is communication protocol designed for wireless networks, based on the IPv6 stack. It is designed to be user-friendly, always secure and cost-efficient. It runs on the IEEE 802.15.4 stack and it was conceived for a large range of applications for home automation. It is a proprietary protocol, and devices using it must have a certificate. The whole network can connect to the Internet through a Wi-Fi “Border router”.

III. General overview

A WSN generally consists of a varying number of sensor nodes and a gateway for the connection to the internet. The gateway can also have its own sensors. The general deployment stages in using a WSN are (see Fig. 1): first, after power-on, the sensor nodes broadcast their status, and the neighboring nodes respond with their own status to detect each other. Second, based on a configured topology (linear, star, tree, mesh, etc.) the nodes make a logical connection between themselves to create a fully organized network. Finally, the links are used to transmit the sensing data and commands.

For environment monitoring, sensor nodes need to be scattered on a large area, many times in remote locations, requiring devices to run for long periods without any intervention. Usually, sensor nodes are powered from batteries, limiting transmission power and active time. The transmission distance can be up to 800 to 1000 meters in an open outdoor environment within line-of-sight [4], but it will greatly decrease indoors to a few meters because of attenuation [5]. To expand the coverage area, WSNs use multi-hop routing to transmit a message from one node to another through intermediate nodes that redirect the message to its destination. The source node sends its data to the nearest neighbor and so on until it arrives at its destination. In a WSN, the gateway node, called Base Node, Coordinator, or Edge Router can reformat the message received from other nodes, retransmit it to another network, or just keep the data in memory for future usage.

The protocols for a WSN may vary, depending on the application it serves, but all have the same main features: self-organizing, self-adaptation, limited node energy and unstable transmission links.

Steps for data monitoring and aggregation:

1. Gather data from sensors
2. Serialize/Pack data
3. Send and route to coordinator
4. Coordinator sends to a sink node, an aggregation node or a web monitoring app

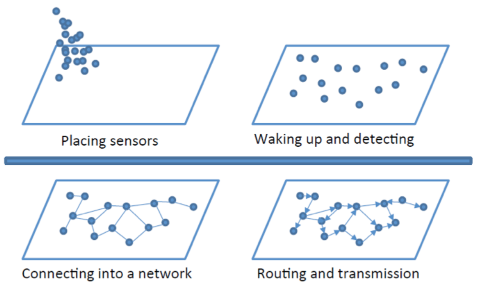

Fig. 1. Organizing and transmitting process of WSNs [6]

IV. Architecture

A. Overview

The most remarkable protocols for WSNs that are used in current commercial applications are: Bluetooth 4.0, which is oriented towards medical WSN, IEEE 802.15.4 oriented towards industrial WSN, and WLAN IEEE 802.11, which is currently the main networking protocol for IoT [6].

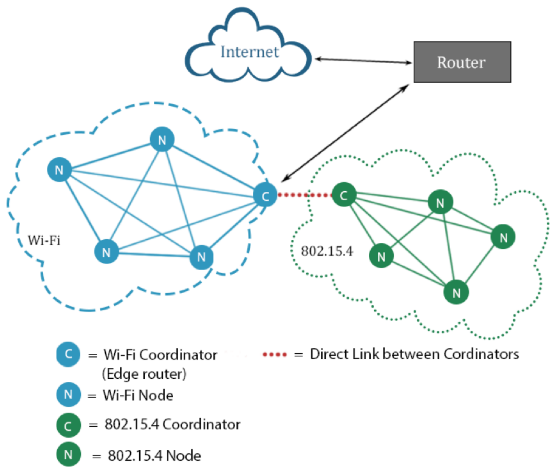

Fig. 2. Wireless Sensor Network with Wi-Fi and 802.15.4 nodes. The whole network has access to the Internet through the edge router (a Wi-Fi Coordinator).

The characteristics of WSN are quite similar to those of a low-speed WPAN, and thus most WSNs take IEEE 802.15.4 as the underlying communication standard. It is focused on low-cost and low-speed communication of nearby devices.

Wi-Fi (IEEE 802.11) offers a larger bandwidth but requires more power. For IoT, the main advantages of WLAN are:

- easy integration of WLAN clients and devices to the internet
- broad acceptance as a wireless communication technology in offices, homes, and industry
- widespread support on mobile devices
- power consumption levels acceptable for industrial applications and sensor networks

Predominant network topologies for IEEE 802.11 WLANs are star topologies, where mobile clients connect directly to access points.

To benefit from the low power of IEEE 802.15.4 and the integration of Wi-Fi, a hybrid network may be used, which is a heterogeneous WSN composed of both types of wireless networks and in which each node can transmit messages to any other node, regardless of its position and network type. In this way, all IEEE 802.15.4 nodes have access to the Internet.

In Wi-Fi network topologies, each individual node can transmit data directly to the Internet if access is granted, but in a WSN that is not usually the case, due to the large network size and coverage area. Nodes can create their own wireless mesh and route all messages to a node which has access to the Internet, named Coordinator or Edge router. The Coordinator can manage the whole network and can route all the data to a Sink (an aggregation node). The Sink node can be another node that can store and display data for all sensors or any kind of Web Service accessible from the Internet. The Wi-Fi Coordinator can also store sensor data, but it is constrained by hardware specific limitations in processing power and storage capability.

In the IEEE 802.15.4, a similar topology is recommended, consisting of a Coordinator and Devices (normal nodes that only collect data from sensors). Optionally, a Router node can be used, which redirects messages from Device nodes to a Coordinator or another Router node.

In this way, a hierarchical structure is created, a Tree network topology, in which the Coordinator is the base node and the Devices are the leaves.

To connect the two types of networks there must be a direct link between the Coordinator from the Wi-Fi network and the Coordinator from IEEE 802.15.4. This link can be implemented by a serial interface through UART.

B. Connections

In a WSN, all nodes must be connected and all gathered data must be accessible from the network. To achieve that in a heterogeneous network, we make use of each node's specific capabilities.

The ESP8266 nodes have built-in support for UDP and TCP. Because there is no need for a permanent connection between nodes and for energy saving, the radio may be turned off, messages can be sent as unicast or broadcast messages through UDP. A Wi-Fi connection is needed, which can be supplied by a Wi-Fi router or they can create their own access points for the other nodes to connect.

The Sparrow v4 nodes can communicate through 802.15.4, creating a wireless network mesh. Because the two networks do not have a way to send messages to one another, a hardware link must be made between two nodes from different networks. This can be done with a SPI or UART interface.

V. Implementation

A. Node hardware

In the 802.15.4 network, we used the Sparrow v4 board. It is an excellent platform for IoT, with ultra low-power wireless transceiver on the 2.4 GHz frequency band, designed to work with a range of wireless protocols, such as IEEE 802.15.4, 6LoWPAN and ZigBee. It also has multiple sensing capabilities, including inertia, gyroscope, temperature, humidity, luminosity, UV index, visible light index, and also a barometric and altimeter. This range of sensors, make it perfect for environment monitoring [7].

The ESP8266 [8] module is a very popular choice for the IoT world. It implements the 802.11 b/g/n protocol and also has integrated support for the TCP/IP and UDP stack protocols. It can be used as a Wi-Fi module, by sending AT commands, or it can be used as a standalone processing unit, having enough memory and processing power to handle a IoT application and gathering data from external sensors through GPIO pins. We used two types of development boards for this project, the NodeMCU v1[9] and WeMos D1 mini [10] for easy programing and debugging the implemented software.

B. Routing protocols

For Sparrow, the Constrained Application Protocol (CoAP) seems as the best choice for routing messages between nodes. The protocol is designed for machine-to-machine (M2M) applications such as smart energy and building automation [11].

We tried to use an existing CoAP implementation over an IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) stack ported for Sparrow v4. Because there is no support implemented yet for sensor data acquisition specific to Sparrow v4 and any software serial communication we chose to use the Arduino environment and libraries, which offer better flexibility and greater support for sensor data acquisition and built-in functions for handling software and hardware serial connections.

For routing the messages between the 802.15.4 nodes, we used the SparrowRadio library, which handles radio management and data transfers specific for Sparrow nodes. Inspired by CoAP, we kept the Coordinator and Device roles and we implemented simpler routing rules for sending Device nodes data to the Coordinator and then to the Wi-Fi Coordinator. The acquisition and transfer of sensor data is done through specific packet data structure.

WSN Protocol for ESP8266 is a lightweight protocol for creating a WSN over Wi-Fi with ESP8266 chips. It is a similar model to the CoAP protocol, but is based on IPv4 and UDP stack. The Coordinator is called a "Head node", which can also act as a Router and the Device is called simply "Node". It supports network discovery and auto-arranging nodes into star, tree and mesh networks. It extends the limitations of the Wi-Fi range by creating new access points for other nodes to connect and auto-redirect the messages from and into the subnets [12].

C. Interconnection

For the two distinct networks to communicate, we made a link between their coordinators. Because the we cannot access the hardware serial from the Sparrow node directly, this connection is represented by a software serial port, which allows the use of any two pins from each node to be used in communication.

For the Sparrow node, we used the MOSI and MISO pins from the ISP module on the Nest as RX and TX pins. These are connected to GPIO12 and GPIO14 on ESP8266, representing the RX and TX, respectively (Table 1.).

Table 1.

Sparrow		ESP8266	
RX	Pin 20 - MOSI	GPIO14	TX
TX	Pin 12 - MISO	GPIO12	RX

Optionally, an SPI connection between the two nodes can be used.

D. Packing and transmitting sensor data

In this paper, we analyzed data gathering from all the nodes in a WSN, not the auto-arrangement and extra messages that each protocol sends between its own nodes to create a wireless network. For the purpose of monitoring environment parameters, it is only needed that each node to send data from its own sensors to the Internet or a Sink node (data aggregation node). In this way sensor data travels in only one direction. For future configuration, or sending a specific message to the whole network or only one node, there should be very little to modify in implementation, the process just reversing the transmission direction and implement the handle action to that message.

In the heterogeneous network that we implemented, ESP8266 and Sparrow nodes both send their sensor data to their respective Coordinator, and since only the Wi-Fi Coordinator is the edge router, it receives all the data from the Sparrow node and passes it directly to the Sink. It also has the capability to store values from all sensors and display them in an HTML page, but the hardware limitation of the Coordinator node forces it to keep and display only a small number of values. This HTML page is useful for viewing a minimal status of the nodes and their sensors in the absence of an Internet connection or if a server or a Sink node is not accessible.

Data packets sent from all nodes to the Coordinator must include relevant information about the sensor type and value, but also an identification for the sender node. In the same time, the packet must have a minimum length, due to low power concerns.

The data structure we chose to use as a data container for sensor data is shown in Listing 1. It includes a unique identification number for the source node that generates the data, which represents a 32-bits integer of the MAC address. It also contains the sensor type, an enumeration of types, showed in Listing 3., and the sensor value. Because a single sensor can supply information for multiple environment variables, we used a structure to hold the data for a single type of sensor and combine all different sensor values type in a union. In this way, a single message can be sent for one sensor read.

The size of the whole data structure depends on the particular sensor values. It has 4 bytes for node_id, 1 byte for sensor_type_t if "-fshort-enums" flag is specified for gcc, and the maximum length of the all sensor value structures. For the sensors we used, the biggest sensor value structure has a size of 8 bytes, giving a total packet size of 13 bytes. Also, for the compiler to not add extra padding on the packet, it must be specified to pack the structure. Otherwise, the size of the packet may be larger, depending on each node architecture, and messages will not be decoded properly.

For packet integrity, a CRC of 4 bytes can be added on the end of the message for control and a start sequence, or a header, at the beginning. In our case this is only needed for the serial communication, because packet integrity for Wi-Fi and 802.15.4 networks are supplied by the protocols we used. The header for the serial transfer is a one-byte value chosen randomly, similar to the "magic" byte in other network protocols, but the same on all nodes. With these fields, the total length of the packet transferred through software serial is 18 bytes long.

Listing 1. Sensor data

```
struct sensor_data_t {
  uint32_t node_id;
  sensor_type_t sensor_type;
  sensor_value_t sensor_value;
};
```

Listing 2. Sensor value struct

```
union {
    sensor_temperature_t temperature;
    sensor_light_t light;
    sensor_battery_t battery;
} sensor_value_t;
```

Listing 3. Sensor types

```
enum sensor_type_t {
  SENSOR_TEMPERATURE = 0x0,
  SENSOR_LIGHT,
  SENSOR_BATTERY
};
```

For efficient data collection, all nodes should transfer their sensor data to the Coordinator and the Coordinator should pass that data further to the Sink or save and display it. The routing protocols can handle variable payload size and can pass the whole packet in a single message, containing data from one type of sensor. In the same manner, for a serial data transfer, a packet is transmitted within a single message.

When the 802.15.4 Coordinator receives a message from a child node, Device or Router, it passes the payload directly to the serial to the Wi-Fi Coordinator. And when the Wi-Fi Coordinator receives a message, either from the Wi-Fi network or the software serial, it stores it in a list of sensor data for displaying it in the HTML page and also encodes it as a JSON string for server upload. It can send data through UDP or TCP, as a broadcast or as a unicast message. Sending the message as a JSON string gives the possibility to integrate the whole WSN with any server API and can be easily decoded and managed. Optionally, it can encode data on a custom way, for the server to handle.

All sensor data packets are routed from a Device or a Coordinator to the main Coordinator, or Edge router. After a sensor data packet is created, it is not altered during the routing stage, also ensuring data integrity on the way. If a packet has a wrong format it is rejected.

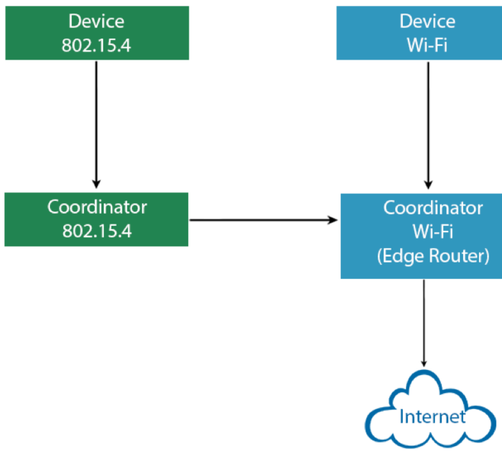

Fig. 3. Routing sensor data packets

VI. Gathering and Aggregating Sensor Data

A. *Temperature and humidity*

Sparrow has an integrated temperature and air humidity sensor produced by SiliconLabs, Si7021 [13]. It has a small print (3mm x 3mm) and can communicate with the microcontroller with a standard I2C protocol and thus transferring data fast. Measurement data consists of the first 8/12 bytes of the humidity value and on the last 12/14 bytes the temperature value.

To make use of the ESP8266 nodes, we used two types of digital temperature sensor: TMP102 [14] and DHT 11 [15], both transferring data through the same I2C standard. Also, DHT 11 has the capability to measure air humidity.

Sensor data packing is described in Listing 3.

Listing 4. Temperature and humidity
```
struct sensor_temperature_t {
  float temperature;
  float humidity;
};
```

In Fig. 4. we can observe the evolution of indoor temperature for one day for tree sensors. The first two (blue and orange) are Sparrow nodes, and the last one (green) is a ESP node. In this plot the difference in sensor measurement is evident, due to nodes placement in different room and to temperature sensors being of different types.

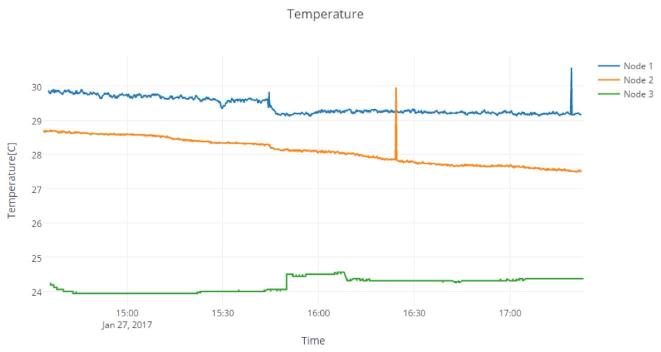

Fig. 4. Temperature[C] evolution on January 27,2007 from three nodes

B. *Light*

For gathering light parameters, we used the Si1145[16] sensor. It is a low-power, reflectance-based, infrared proximity, ultraviolet(UV) index, and ambient light sensor with I^2C digital interface. This sensor is included in Sparrow v4 board and we also used the digital light intensity sensor module GY-30 [17], which also has an I^2C interface.

Listing 5. Light data structure
```
struct sensor_light_t {
   uint16_t UV;
   uint16_t visible;
   uint16_t IR;
   uint16_t proximity;
};
```

In Fig. 5 we can observe visible light variation on three different locations: Node 1(blue) in a darker place, node 2(orange) with an average visible light, and node 3(green) near a window in daylight.

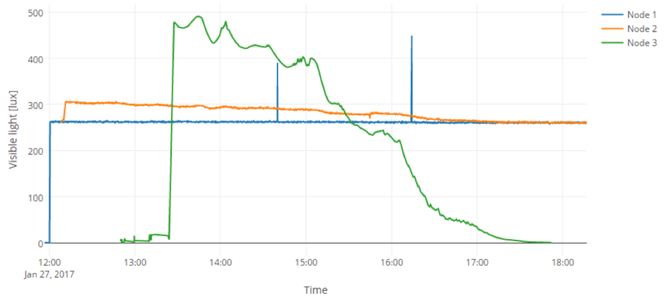

Fig. 5. Light visible index[lux] on January 27,2007 from three nodes

C. *Battery level*

Battery level is crucial for a node in a WSN. For monitoring the battery level from both Sparrow v4 and ESP8266, we read the A0 value. It gives an integer value ranging from 0 to 1024 and then converts it in voltage value. For the Sparrow V4, the maximum value represents 1.8V and for ESP8266 it handles a maximum voltage of 1V.

Listing 6. Battery data structure

```
typedef float sensor_battery_t;
```

For monitoring battery voltage over a day, we used a Sparrow node powered by two AA batteries, at approximatively 3V. We can see in Fig. 6. that the battery level obtained by reading the A0 value fluctuates and it is not a reliable source for battery monitoring.

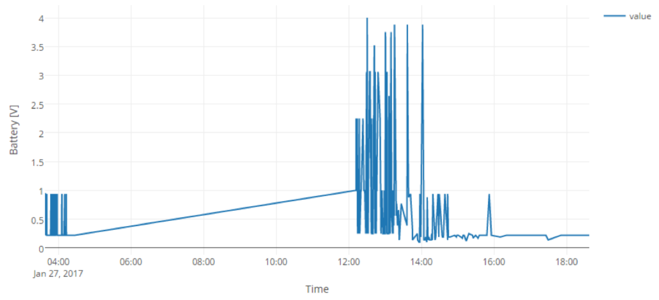

Fig. 6. Battery voltage[V] on January 27,2007 from a Sparrow v4 node

D. *Data aggregation*

All data from sensors arrive at the Wi-Fi Coordinator, each node being configured to read sensor value and send it at an interval of ten minutes, for tenting purposes, but in an actual environment monitoring situation, this interval should be changed based on the requirements and battery saving.

The Coordinator displays the last ten values from all sensors and the source node ID (MAC address). The format in which the data is shown can be either HTML or JSON. We implemented both, accessible at http://[ESP_IP]/sensors for the HTML page and at http://[ESP_IP]/sensors/json for the JSON output.

For data aggregation, the cloud platform is a popular choice, being always online and accessible from anywhere, perfect for gathering data from sensors scattered around the environment. We use services from devicehub.net [18], sending sensor data from the Edge Router to the cloud directly through a HTTP request for each sensor type.

VII. Conclusion

In this paper, we presented an architecture and a software solution for environment monitoring using sensor nodes which can communicate both in Wi-Fi and 802.15.4 networks. Thus, we can use the specific capabilities of both networks: the low-power of the 802.15.4 network and the accessibility of the Wi-Fi network.

The challenge of this project was to link the two networks in a manner that any node can supply data from its sensors, regardless of its position, distance or access to the Internet. We manage to successfully link the two networks, making the main nodes, the Coordinator, to communicate easily. We used a large range of sensors to demonstrate the usage of a heterogeneous WSN. During our evaluation process, no system freeze or auto restart was observed. Also, by using the accessible ESP8266, the whole system is easier to implement and offers a low cost. We hope that, by presenting this design and the issues we overcome, that other researchers can design their own solution for accessible and simple environment monitoring.

A problem in our architecture is that if the Wi-Fi Coordinator is not working, the whole network is inaccessible. This can be solved by using multiple Wi-Fi Coordinators in the same network or as child nodes to one Edge Router and linked with 802.15.4 Coordinators. In this way, if one stops working, the other can handle messages from the rest of the nodes.

On the future, we plan to add support to the CoAP 6LoPAN library for Sparrow v4 node, so that the routing can be made more reliable and extend the network area with sub-networks.

One other functionality we plan to add is the possibility to adjust the sleep time based on battery level and sensor values. In this way, the radio will be used only when the sensor value fluctuates, or the node receives a message.

References

[1] ZigBee Protocol. http://www.zigbee.org/zigbee-for-developers/applicationstandards/ Last accessed on 20 December 2016
[2] ZigBee IP. http://www.zigbee.org/zigbee-for-developers/network-specifications/zigbeeip/ Last accessed on 20 December 2016
[3] All About Circuits – Thread Network Protocol - http://www.allaboutcircuits.com/technical-articles/thread-network-protocol/, Last accessed on 18 November 2016
[4] Wireless sensor networks: a survey - I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci
[5] Fundamentals of Wireless Communication - David Tse,Pramod Viswanath 2005
[6] Internet of Things: Wireless Sensor Networks – White Paper – International Electrotechnical Commission
[7] Sparrow v4. https://elf.cs.pub.ro/wsn/wiki/sparrow_v4_en Last accessed on 14 January 2017
[8] ESP8266 Family http://www.esp8266.com/wiki/doku.php?id=esp8266-module-family Last accessed on 14 January 2017
[9] NodeMCU. http://www.nodemcu.com/index_en.html Last accessed on 14 January 2017
[10] WeMos D1 mini pro https://www.wemos.cc/product/d1-mini-pro.html Last accessed on 14 January 2017
[11] https://tools.ietf.org/html/rfc7252 - CoAP RFC
[12] Wireless Sensor Network Protocol for ESP8266. https://github.com/w01f6/esp8266-wsn-protocol Last accessed on 06 February 2017
[13] Si7021 Temperature and Humidity sensor. https://www.silabs.com/Support%20Documents%2FTechnicalDocs%2FSi7021-A20.pdf Last accessed on 20 January 2017
[14] TMP102 Temperature sensor https://www.sparkfun.com/datasheets/Sensors/Temperature/tmp102.pdf Last accessed on 20 January 2017
[15] DHT11. http://www.micropik.com/PDF/dht11.pdf Last accessed on 20 January 2017

[16] Si1145 Light sensor. https://cdn-shop.adafruit.com/datasheets/Si1145-46-47.pdf Last accessed on 20 January 2017
[17] GY-30 Light sensor. http://rohmfs.rohm.com/en/products/databook/datasheet/ic/sensor/light/bh1750fvi-e.pdf Last accessed on 21 January 2017
[18] DeviceHub.net cloud platform. https://www.devicehub.net Last accessed on 30 January 2017