electronics

MDPI

Article

Building IoT Applications with Raspberry Pi and Low Power IQRF Communication Modules

Isidro Calvo [1,*], José Miguel Gil-García [2,*], Igor Recio [1], Asier López [1] and Jerónimo Quesada [2]

[1] Department of Systems Engineering and Automatic Control, University of the Basque Country (UPV/EHU), E.U.I. of Vitoria-Gasteiz, Nieves Cano, 12, 01006 Vitoria-Gasteiz, Spain; irecio003@ikasle.ehu.eus (I.R.); alopez273@ikasle.ehu.eus (A.L.)

[2] Department of Electronic Technology, University of the Basque Country (UPV/EHU), E.U.I. of Vitoria-Gasteiz, Nieves Cano, 12, 01006 Vitoria-Gasteiz, Spain; jeronimo.quesada@ehu.eus

* Correspondence: isidro.calvo@ehu.eus (I.C.); jm.gil-garcia@ehu.eus (J.M.G.-G.); Tel.: +34-945-01-3254 (I.C.); +34-945-01-4126 (J.M.G.-G.)

Academic Editors: Steven J. Johnston and Simon J. Cox
Received: 16 May 2016; Accepted: 1 September 2016; Published: 8 September 2016

Abstract: Typical Internet of Things (IoT) applications involve collecting information automatically from diverse geographically-distributed smart sensors and concentrating the information into more powerful computers. The Raspberry Pi platform has become a very interesting choice for IoT applications for several reasons: (1) good computing power/cost ratio; (2) high availability; it has become a de facto hardware standard; and (3) ease of use; it is based on operating systems with a big community of users. In IoT applications, data are frequently carried by means of wireless sensor networks in which energy consumption is a key issue. Energy consumption is especially relevant for smart sensors that are scattered over wide geographical areas and may need to work unattended on batteries for long intervals of time. In this scenario, it is convenient to ease the construction of IoT applications while keeping energy consumption to a minimum at the sensors. This work proposes a possible gateway implementation with specific technologies. It solves the following research question: how to build gateways for IoT applications with Raspberry Pi and low power IQRF communication modules. The following contributions are presented: (1) one architecture for IoT gateways that integrates data from sensor nodes into a higher level application based on low-cost/low-energy technologies; (2) bindings in Java and C that ease the construction of IoT applications; (3) an empirical model that describes the consumption of the communications at the nodes (smart sensors) and allows scaling their batteries; and (4) validation of the proposed energy model at the battery-operated nodes.

Keywords: IoT gateways; IoT applications and drivers; Wireless Sensor Networks (WSN); low power solutions; remote sensing; IQRF; energy consumption model

1. Introduction

The Internet of Things (IoT) aims at connecting a world of networked smart devices typically equipped with sensors and radio frequency identification to the mainstream Internet, all sharing information with each other without human intervention. In the future, the Internet might be considered as comprised of billions of intelligent communicating 'things' that will further extend the borders of the current Internet with physical entities and virtual components [1]. According to a report by Cisco delivered in 2011, the number of devices connected to the Internet for the year 2020 is expected to be around 50 billion, yielding to 6.6 devices per person [2]; other estimations are even more generous (e.g., a number of seven trillion wireless devices serving seven million people is expected by 2017 [3]). An increasing number of IoT applications is found in different domains, such as

transport, energy, home, healthcare, logistics or industry. IoT applications produce many data that might be useless unless they are conveniently processed for extracting information. Since the volume of data produced could be considerable, the process of converting raw data into information should be automated, preferably at the nearest possible place of their acquisition to reduce the communication bandwidth. This can be achieved with the so-called 'data collectors' or 'gateways' that: (1) collect data from proximity sensors; (2) convert data into information near the acquisition location; and (3) send them to higher-level computers used for storage, analysis or monitoring purposes, typically using cloud computing techniques [4,5].

IoT infrastructures present several common characteristics, such as: (1) dealing with heterogeneity; (2) use of resource-constrained devices; (3) applications that require spontaneous interaction; (4) ultra-large-scale networks and large number of events; (5) dynamic network behavior requirements; (6) context-aware and location-aware applications; and (7) the need for distributed intelligence [3].

Two important issues in distributed applications are the use of low-cost hardware platforms and the management of the available resources at the nodes, typically processor, memory, network usage and energy usage [6].

Raspberry Pi is very adequate for this kind of applications, since it provides a very powerful/low-cost platform with good hardware expansion capabilities (different ports, General Purpose Input/Output (GPIO), pins) and standard connectivity (Ethernet, WiFi interfaces) [7]. Even though alternative Single-Board Computers (SBC) providing similar characteristics are available in the market, the price of the Raspberry Pi is very competitive because, initially conceived of for education, it has become a mass product [8]. Currently, there are several examples of IoT systems working on this technology [9–13]. Some of them are commercial products adapted for industrial automation [14].

One of the most critical resources in IoT applications is energy usage. Distributed nodes (i.e., smart sensors, actuators and data collectors) are typically operated on batteries and are required to work unattended for long periods of time (e.g., several months or even years). Communication technologies have become one of the major battery 'killers' for smart sensors. There exist some communication protocols and standards (e.g., ZigBee or Bluetooth) frequently used for IoT applications [1]. Some IoT applications may require low power solutions that provide a longer life-time of the batteries. In other cases, the IoT sensors must be distributed over long distances, causing propagation problems. In this scenario, new technologies are emerging for specific applications, such as smart metering [15]. Some examples of these technologies are LoRa/LoRaWAN (Long Range Wide Area Network) [16], Sigfox [17] or IQRF [18], which are aimed at saving energy, working on longer distances and providing a higher degree of flexibility to adapt to the requirements of the applications.

The creation of IoT gateway architectures and libraries that help designers and programmers to create new applications is a matter of interest, since it will allow integrating field data into higher level applications, such as cloud applications, while minimizing energy consumption.

This work proposes a possible gateway implementation with specific technologies. It solves the following research question: how to build gateways for IoT applications with Raspberry Pi and low power IQRF communication modules. The following steps [13,19] were followed to structure our research: (1) technology/literature review; (2) design of a gateway architecture for IoT applications; and (3) evaluation of an empirical model for the discharge of the batteries.

The literature/technology review is carried out in Sections 2 and 3: Section 2 is devoted to analyzing previous related work, and Section 3 is dedicated to providing a short overview of the IQRF technology. The design of our approach is presented in Section 4, which is divided into several subsections, each aimed at explaining different issues of the IoT gateway. The evaluation is provided in Section 5, where a simple mathematical model based on empirical data is presented. This section also discusses the obtained results. Section 6 draws some conclusions about the work.

2. Related Work

The concept of IoT is still under definition [1]. Kevin Ashton firstly proposed the concept in 1999, and he referred to the IoT as uniquely identifiable interoperable connected objects with Radio-Identification (RFID) technology. A commonly-accepted definition is: "A dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual 'Things' have identities, physical attributes and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network" [20]. Typically, IoT applications involve diverse technologies, including Wireless Sensor Networks (WSN), barcodes, intelligent sensing, RFID, NFC and low-energy wireless communications [1]. Creating IoT applications is a challenging task, since it requires working with heterogeneous, resource-constrained, location- and context-aware distributed infrastructures in order to provide complex applications. Several middleware solutions are available in order to help programmers [3].

The limitations of the IoT devices in terms of storage, network and computing, as well as the requirements of complex analysis, scalability and data access benefit from a technology like cloud computing. The IoT infrastructure can generate large amounts of varied data that must be quickly analyzed by means of different techniques [5,21] feeding the cloud computing infrastructure. Several authors have proposed different architectures that cover all layers. As a matter of example, a service-oriented architecture for IoT applications presented in [1] defines the following layers:

(1) Sensing layer: integrated with available hardware objects to sense the status of the things.
(2) Network layer: provides the infrastructure that supports the link among the things over wireless or wired connections.
(3) Service layer: creates and manages the services required by the users or applications.
(4) Interface layer: consists of the interaction methods with users or applications.

In this scenario, new challenges arise [22]; some of them are technical issues: (1) integrating social networking with IoT solutions; (2) developing green IoT technologies; (3) developing context-aware IoT middleware solutions; (4) employing artificial intelligence techniques to create intelligent things or smart objects; and (5) combining IoT and cloud computing [20]. Some of these challenges may be addressed by intelligent gateways that bridge sensor networks with traditional communication networks [23]. These gateways are responsible for collecting data from field sensors with different technologies, mainly wireless sensor networks, and sending them to the cloud infrastructure by means of TCP/IP-based communication networks.

A prototype architecture aimed at monitoring applications based on the Raspberry Pi is presented in [7]. It is relatively easy to find in the literature other examples of data acquisition systems aimed at different domains, such as smart cities [11,24], industrial process monitoring [10] or home automation [9,12,25], that use the Raspberry Pi. One interesting example of its capabilities in these applications may be found in [13], where an IoT-enabled emergency information architecture for elderly people is presented. We may conclude from these works that the Raspberry Pi is an inexpensive, extremely versatile and small computer, with network connectivity (via Ethernet or WiFi), supported by a large open-source community, which is adequate for building embedded applications by means of the GPIO pins.

The use of adequate communication means is another issue of interest when building IoT applications. Some technologies may be better suited than others to solve the requirements of particular applications. It is easy to find in the literature survey papers aimed at helping application engineers to select the most appropriate protocols [1,26]. ZigBee and Bluetooth are specifically designed for IoT applications. These competing technologies present different characteristics including range, data rate, network latency, power profile, security and complexity [27]. There are other kinds of WSN that could be used in IoT applications, but they are less common than the previous two [28]. Energy efficiency is also a key issue in IoT applications, especially at the sensor nodes [29,30].

Although ZigBee is relatively new, since 2004, after discovering that due to its high complexity and difficult usage, the implementation is not economic in smaller and some medium-sized applications, several lighter protocols were soon established, sometimes even by the original ZigBee Alliance members [31].

We chose IQRF technology [18] due to its versatility. It allows several communication modes by combination of: (1) the ISM band; (2) channel; (3) transmission bit rate; (4) transmission power level; and (5) reception level model. It also allows changing these parameters at run-time, as shown in this paper. Several papers provide wireless sensory network solutions based on this protocol [4,32–34].

This work focuses on the following contributions: (1) providing a possible IoT gateway architecture with specific technologies (Raspberry Pi and IQRF communication modules); (2) bindings in Java and C that ease the construction of IoT applications; (3) an empirical model that describes the consumption of the communications at the nodes (smart sensors) and allows scaling their batteries; and (4) validation of the proposed energy model at the battery-operated nodes.

3. Overview of IQRF Technology

IQRF is a platform for low speed, low power, reliable and easy to use wireless connectivity for telemetry, industrial control and building automation that can be used with different electronic equipment [18]. It is aimed at providing wireless connectivity in applications that require remote control, monitoring, alarming, displaying remotely-acquired data or connecting several devices to a wireless network. IQRF is a complete ecosystem, including hardware, software, development support and services [32].

The IQRF ecosystem covers hardware, software and protocols. At the heart of the system, there are several RF modules that can operate at the 433-MHz, 868-MHz and 916-MHz ISM (Industrial, Scientific and Medical) bands. Among other circuits, the modules hold an RF transceiver and an eight-bit microcontroller, which executes an operative system (OS) responsible for, among others, the communications and mesh networking functions. The final system can (1) extend the capabilities of the OS new programming functions by an end-user or (2) add a ready-to-use software layer of a Hardware Profile (HWP) plug-in responsible for supporting a dataflow-oriented Direct Peripheral Access (DPA) mechanism to interact with all of the peripherals fitted in the module. In the second case, there is no need for additional programming from the end-user. These modules are known as Data-Controlled Transceivers (DCTR) in the IQRF ecosystem. It is also possible to program a third layer of custom software to handle situations not covered by the DPA. Figure 1 represents schematically the three possible scenarios regarding the firmware development.

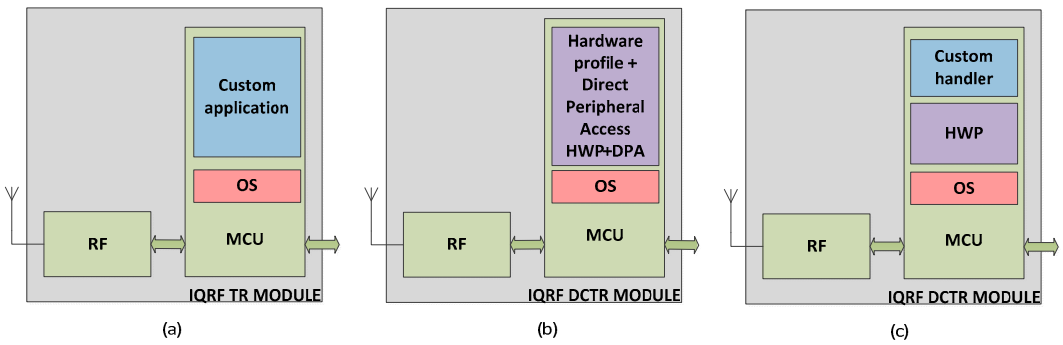

Figure 1. (**a**) Common transceiver; (**b**) data-controlled TR; (**c**) Data-Controlled Transceivers (DCTR) running custom software. HWP, Hardware Profile; DPA, Direct Peripheral Access.

The link layer is byte oriented with a maximum packet size of 64 bytes (56 if DPA is used). Transmission distance is claimed to be in the range of 100 metres. In this work, a TR-52D module was employed achieving up to 90 m without losing efficiency in the tests (other authors also report similar

distances [33]). The manufacturer claims that the range is longer when properly oriented, but this has not been tested in this work. According to the specifications, 240 hops between transceivers are allowed before discarding a packet.

The energy consumption varies depending on several factors, such as transmission power, reception and execution modes at the MCU. If the RF transceiver is disabled, the module current consumption ranges from 1.9 μA in sleep mode to 1.6 mA in run mode. During the transmission, the supply current depends on the selectable seven levels of transmission with power ranges that vary from 14 mA to 24 mA. When receiving, the current drained from the power supply starts at 13 mA in Standard mode (STD), but can be reduced to 25 μA if operated in the Extra Low Power mode (XLP) [35].

There is built-in support in the OS for a Serial Peripheral Interface (SPI) protocol to command the module from a locally-attached controller.

The modules offered by IQRF can achieve several network topologies, but the most versatile one is the mesh topology. In general, one module plays the coordinator role, while the others are considered plain nodes. One of these nodes can play the role of coordinator for a subnetwork. The OS supports node bonding, network discovery, routing packets and unbonding from the network with an easy to use Application Programming Interface (API).

The whole ecosystem is completed with a full set of gateways, routers, development tools and a fully-documented Software Development Kit (SDK) package for hardware deployment and cloud services for data exchange between IQRF networks and end-users.

4. IoT Gateway Architecture and Wrappings

4.1. Description of the IoT Gateway Architecture

The proposed architecture is comprised of three levels: (1) **Concentrator**, implemented on a Raspberry Pi SBC; (2) **Coordinator**, implemented on a privileged IQRF module attached to the concentrator that plays the role of coordinator; and (3) **End nodes**, implemented on IQRF modules, which acquire field information by means of different attached sensors (see Figure 2). This architecture involves different types of communication technologies at every level. The connection between the Raspberry Pi and the IQRF coordinator is implemented by means of an SPI connection, and the connection between the coordinator and the end nodes is carried out by means of IQRF wireless technology. In the proposed IoT gateway architecture, the Concentrator implements the *interface* and *service layers* so that it respectively provides the interaction methods and services required by users and higher level applications. The Coordinator implements the IoT *network layer* since it holds the IQRF WSN configuration and routing information, and it is responsible for interrogating the end nodes. End nodes implement the *sensing layer* by means of different sensors that let the nodes acquire field information. End nodes will send the data when required by the Coordinator.

One of the characteristics of the IQRF technology is that devices implement the full stack allowing them to behave either as Concentrators or End nodes. A unique type of device, the IQRF TR-52DA [35] was used in the architecture for the roles of both coordinator, as well as end nodes. In addition to the communication capabilities, the IQRF TR-52DA has: (1) a PIC16LF1938 microcontroller with interrupt capability; (2) a SPI interface used at the IQRF Coordinator to establish the communication with the Raspberry Pi; (3) an embedded temperature sensor; (4) two colors (green and red) of LEDs to be manipulated from the microcontroller; (5) six general purpose I/O pins available to connect external sensors; (6) a two-channel A/D converter; (7) a hardware timer; (8) power supply connections; and (9) battery monitoring capabilities. These end nodes were powered with lithium-polymer batteries to guarantee their autonomy.

The hardware architecture was implemented as follows (see Figure 2): (1) the **Concentrator** was implemented with a Raspberry Pi B+ with Raspbian; (2) the **Coordinator** was implemented with a TR-52DA IQRF node acting as coordinator; and (3) the **End nodes** were implemented by means of TR-52DA IQRF nodes powered with 400-mAh lithium-polymer batteries.

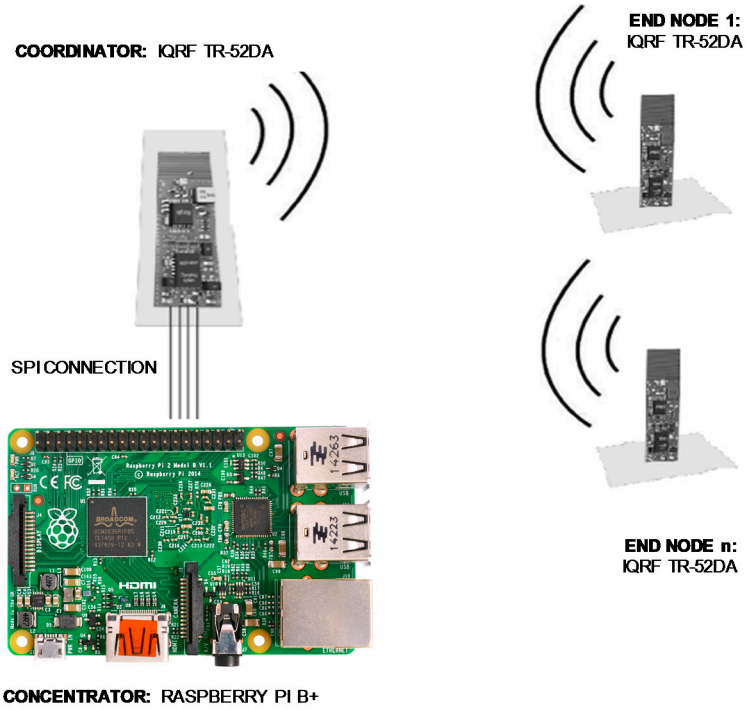

Figure 2. Implementation of the IoT gateway architecture showing the hardware modules.

4.2. Raspberry Pi IQRF Coordinator Connection

A prototype board (shown in Figure 3) was created to implement the connection between the Concentrator (Raspberry Pi) and the Coordinator (IQRF node) with an ad hoc Printed Circuit Board (PCB). This board provides the interface between the Raspberry Pi and the IQRF coordinator node. This board also includes connectors to carry out the energy measurements at the IQRF Coordinator used to obtain the empirical model discussed in the next section.

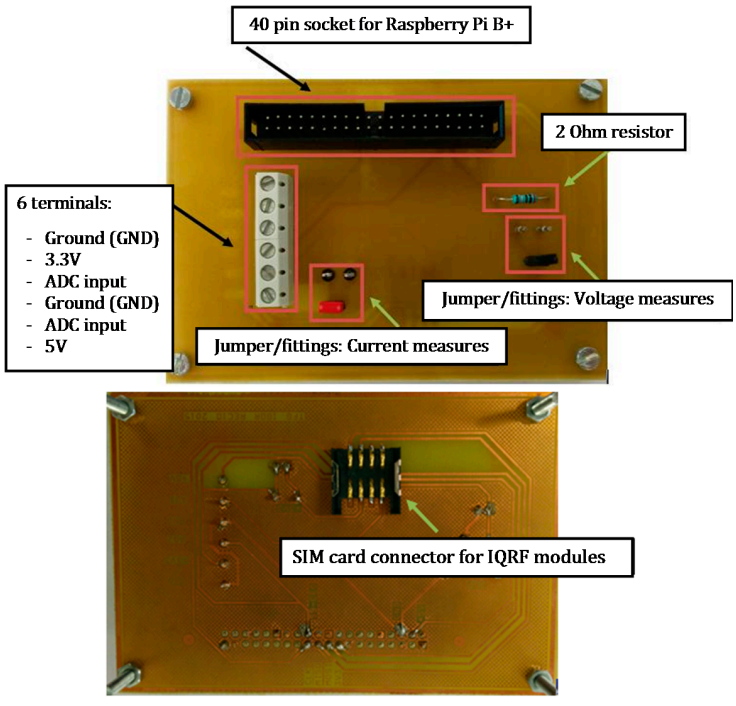

Figure 3. Prototype Raspberry Pi-IQRF Coordinator connection PCB.

The connection between the Raspberry Pi and the IQRF coordinator is carried out by means of the SPI protocol, the Raspberry Pi acting as the SPI master and the IQRF coordinator as the slave. The SPI frames the protocol defined at the Raspberry Pi to manage all nodes of the IQRF IoT network.

4.3. Wireless Communication Coordinator: End Nodes

To create and configure the IQRF network, it is necessary to specify the following parameters at the IQRF Coordinator and End nodes: (1) RF band; (2) RF channel number; (3) data transmission bit rate; (4) transmission power level; and (5) reception level mode. The available values may be found in Table 1. Since reconfigurations at run-time are allowed, these data were embedded in the protocol issued by the Raspberry Pi to the IQRF Coordinator.

Table 1. Configuration parameters of the IQRF network.

Band (MHz)	RF Channel	TX Bit Rate	TX Power	RX Level Mode
433 868 916	0 to 16 0 to 67 0 to 255	1.2 Kb/s (Experimental) 19.2 Kb/s 57.6 Kb/s (Experimental) 86.2 Kb/s (Experimental)	0 (min) to 7 (max)	STD (Standard) LP (Low Power) XLP (Extra Low Power)

Furthermore, the IQRF nodes must be bonded to the network in order to be available. Network bonding requires executing the process depicted in Figure 4. The IQRF coordinator expects that all nodes issue bond requests in which they send their *Module ID*, so it assigns the *Network ID* and *Address* of the end node. Once this operation is carried out, all end nodes will be bonded to the IQRF network, and normal operation will be started. End nodes need to know all configuration parameters (i.e., band, channel, transmission bit rate and power and reception level mode) previous to the bonding operation in order to establish the link. The topology of the network will be internally managed by the Coordinator using the IQMESH algorithm [31]. The IQRF technology is able to adapt to a wide range of topologies; the authors only tested the architecture with two topologies: linear (all nodes connected in line, in order to reach a maximum distance) and star (all nodes scattered in order to cover a broad area).

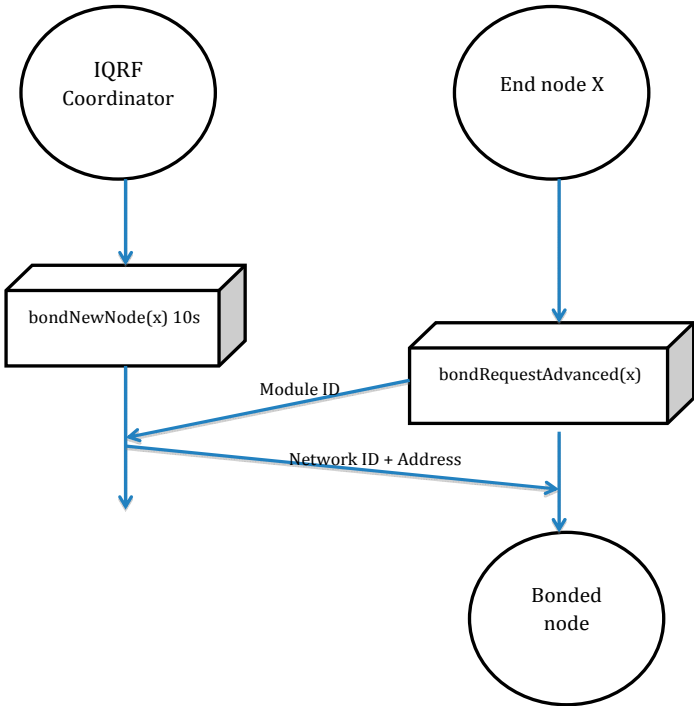

Figure 4. Bonding diagram of the IQRF end nodes to the coordinator.

4.4. Protocol Description

This section describes the operations allowed by the system and the protocol that implements it. This protocol is aimed at the TR-52DA module [35], but it could be easily adapted to other modules by the same provider. The protocol defines the operations issued by the Raspberry Pi to the IQRF coordinator by means of SPI, which will be respectively sent to the corresponding end node.

As shown in Table 2, the protocol allows *bonding* and *unbonding* ('U') end nodes to the IQRF network dynamically, so the topology of the network may be changed in time. Furthermore, the protocol defines the commands used to acquire data from the end nodes. These data may be *Temperature* ('T'), *Voltage* ('V') or *Analog data* ('A'), by means of the embedded sensors and available connectors at the end nodes. There are also some commands to manipulate the *Red* ('R') and *Green* LEDs ('G'). Another command allows one to *reset* ('r') all modules. There is one command to *send data* via SPI to the end nodes ('I'). Finally, there is another command to issue the network *reconfiguration* ('C'). This operation will be described in detail in the next subsection.

Table 2. Available commands to control the end nodes. SPI, Serial Peripheral Interface.

Function	Command	Parameters	
Network bonding	Node_ID	None	
Network unbonding	'U'	Node_ID	
Temperature acquisition	'T'	Node_ID	
Voltage acquisition	'V'	Node_ID	
Analog data acquisition	'D'	Node_ID	
Red led control	'R'	'0', switch off '1', switch on	Node_ID 'A'
Green led control	'G'	'0', switch off '1', switch on	Node_ID 'A'
Modules reset	'r'	Node_ID	
Data sending via SPI	'I'	-	
RF reconfiguration	'C'	-	

4.5. Dynamic Network Reconfiguration

One of the strengths of the IQRF technology is flexibility. For example, IQRF nodes may work both as coordinators or end nodes since they have the same stack. Furthermore, IQRF nodes may work in different modes according to the quality of service (QoS) requirements of the applications, including different frequency bands, channels, data transmission rates, transmission power levels and reception level modes (see Table 1). The authors allowed in the protocol the ability to change these parameters 'on the fly', allowing one to select a new configuration for all IQRF end nodes from the Concentrator (Raspberry Pi) in order to adapt the IoT wireless communications to new operational requirements. This characteristic is useful in different situations, e.g., when the end nodes are low on batteries, in order to use a new configuration, which demands lower energy or when different operational modes are demanded, such as the alarming operation of the IoT system.

This behavior is achieved by means of the *reconfiguration* command, issued by the Raspberry Pi, which requires the following information about the new configuration: (1) new transmission power level (from zero, minimum, to seven, maximum); (2) new data transmission rate; (3) new frequency band; and (4) new channel (see Table 1 for more information about the parameters). This command is used by the IQRF coordinator to send all end nodes the new configuration parameters. There is one default configuration, which is comprised by the following parameters: (1) transmission power level: maximum, 7; (2) transmission rate: 19.2 Kb/s; (3) frequency band: 868 MHz; (4) channel: 52. The default configuration allows avoiding the IQRF nodes possibly remaining in any unknown configuration in the case of failures, for example in the case of battery exhaustion.

4.6. C Wrapping of the Architecture

This section describes the C bindings provided by the authors to create IoT applications with the IQRF technology from the Raspberry Pi. The aim is to provide an easy to use wrapper that allows programmers to integrate this technology into higher level applications. Table 3 enumerates the available functions that map the protocol described above. These functions are provided as the RPi_IQRF.C library. These bindings wrap: (1) the code used to connect the Raspberry Pi with the IQRF coordinator through the SPI connection; (2) the code embedded at the IQRF coordinator to manipulate the IQRF end nodes; and (3) the code at the IQRF end nodes to measure the field data (temperature, voltage or analog data from the connected sensor.)

Table 3. C bindings for controlling the IQRF network from the Raspberry Pi.

C Heading	Function
int bond (char node)	End node bonding to the IQRF network
int unbond (char node)	End node unbonding from the IQRF network
int reset (char node)	Reset of one or all IQRF end nodes
char *temp (char node)	Acquisition of the temperature at one IQRF end node
char *voltage (char node)	Acquisition of the voltage at one IQRF end node
char *ADC (char node)	Acquisition of the value read at the analog converter at one IQRF end node
int LEDR (char onOff, char node)	Red led control of one IQRF end node
int LEDG (char onOff, char node)	Green led control of one IQRF end node
int config (char power, char speed, char, band, char channel1, char channel2, char channel3)	Reconfiguration of the IQRF network

4.7. Java Wrapping

Modern IoT applications, e.g., cloud applications, are frequently based on Java technology, so the authors also provide a Java wrapping that allows using the proposed architecture from Java. Figure 5 shows the UML diagram that allows programmers to create Java applications.

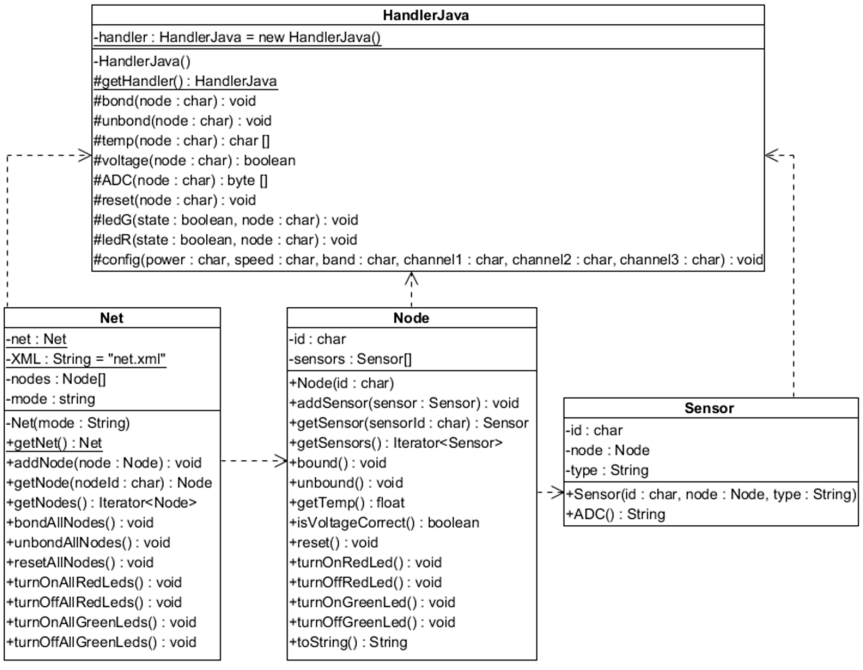

Figure 5. UML diagram for the Java wrapping.

The *HandlerJava* class is responsible for enabling a direct communication link between the Java interface and the native C libraries. It provides similar methods to the functions found in Table 3. The *Net* class represents the IQRF network and allows its use from the Java interface. The first time that it is used, it gets the IQRF network configuration from "net.xml", an xml formatted file. This class allows the execution of serial commands aimed at communication issues. The *Node* class represents every node at the IQRF network, allowing the execution of operations over the physical nodes. The *Sensor* class represents every sensor connected at every node. It enables reading the data of every physical sensor in different formats.

5. Energy Model Validation and Discussion

Some key points that influence energy consumption must be determined for a given design, such as transmission strength and duration, amount of time elapsed between active states, as well as the consumption in sleep and active states. The strategy followed to read the sensor is also important; e.g., a module that is regularly pulled to know the state of a door might run out of battery faster than a similar module that wakes up and transmits the occurrence of an opening or closing event. All of these factors should be considered when calculating the power budget required for a given application and the type of battery required.

Some applications may require anticipating the duration of the batteries to schedule maintenance tasks to replace them. One typical approach involves querying field devices for their power state, but this introduces undesired overhead on the remote devices. Most IoT infrastructures rely on data collectors that store and post-process the magnitudes read by the sensors. An alternative approach would involve running a model of the power consumption for the remote devices (based on communication issues such as transmission and reception events, payload and topology of the network) to determine in which nodes batteries must be replaced. This model also allows new functionalities, such as to extend the periods between queries of a starving node or to reduce the amount of data transmitted.

The accuracy of the model relays on understanding the power consumption characteristics of the sensor nodes and the conditions of the data transmission. Some realistic models for power consumption in wireless sensor network devices are presented in [36,37]. These works focus on obtaining realistic models that split the overall consumption into different sources. They provide criteria to choose design parameters once the practical aspects of the communications have been measured. Some works measure the power consumption that the modules need to carry out transmission and reception events, isolated from other forms of current consumption sources [38,39]. We have followed a similar approach to obtain a model of the discharge of the remote devices.

The charge drained from the battery can be estimated following Equation (1) by measuring the current flowing through the module over time.

$$Q(t) = \int I(t)\, dt \tag{1}$$

As the battery capacity units are given in mAh, the obtained measurements should be converted into submultiples of these units to estimate the duration of the battery charge. Current consumption measurement is carried out indirectly recording the current profile on an oscilloscope by measuring the voltage drop over a fixed 2 Ohms 1% resistor, as depicted in Figure 6.

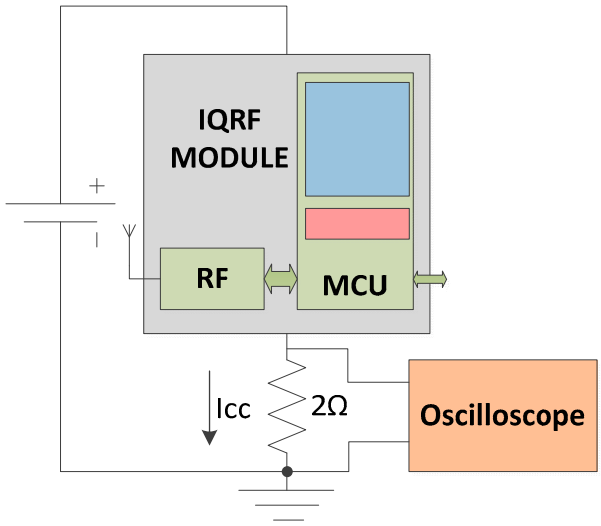

Figure 6. Measurement setup for obtaining the voltage drop over a fixed resistor.

The three transmission modes vary in the preamble duration from 3 ms in the Standard mode (STD_TX), 50 ms for the Low Power mode (LP_TX) to 900 ms for the Extra Low Power mode (XLP_TX). These different preamble times are required not to miss a packet in the reception side when configured to work in low power modes.

Firstly, the required current to transmit the information varies according to the RF transceiver power and the amount of bytes sent. Figure 7 shows the current profile obtained for a 64-byte transmission carried out at full power (level 7).

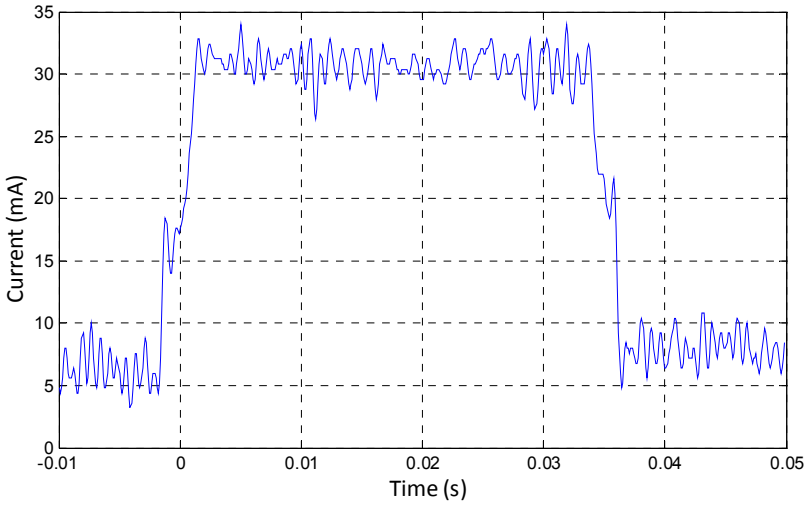

Figure 7. Current profile for a 64 byte transmission at full power (level 7).

The average current is found to be around 23 mA higher than the measured in run mode, which is consistent with the datasheet provided by the vendor. By integrating (Expression 1) the measured current over the transmission period, the charge drained from the battery can be calculated. Tables 4 and 5 summarize the currents measured for maximum (Level 7; green in Figure 8), minimum (Level 0; red in Figure 9), medium transmission power (Level 4; blue in Figure 8) and for several payloads.

Table 4. Transmission current for several payloads.

Payload (Bytes)	Full TX Power (mA)	Medium TX Power (mA)	Minimum TX Power (mA)
1	20.44	13.37	10.69
16	20.61	14.03	11.39
32	21.45	14.20	12.13
48	21.98	14.40	12.41
64	22.98	15.44	12.42

Table 5. Transmission electric charge for several payloads.

Payload (Bytes)	Full TX Power (nAh)	Medium TX Power (nAh)	Minimum TX Power (nAh)
1	64.07	39.32	26.15
16	104.78	68.38	49.88
32	151.36	97.87	74.13
48	201.75	128.49	97.62
64	241.96	157.93	120.07

Regarding the transmission side, an empirical mathematical model may be obtained from these measurements. Since nodes may work in different working modes depending on the selected transmission power (from zero, minimum, to seven, maximum, transmission levels), three different linear regressions have been calculated in order to predict the electric charge at maximum (seven), minimum (zero) and medium (four) levels (see Table 6). These regressions allow one to predict the electric charge, expressed in nAh, required for a number n of transmitted bytes $tec_P(n)$.

Table 6. Empirical model for transmission: required electric charge to transmit n bytes.

TX Power Level	Linear Fit (nAh)
7	$tec_7(n) = 45.275\,n + 16.96$
4	$tec_4(n) = 29.732\,n + 9.2006$
0	$tec_0(n) = 23.558\,n + 2.8963$

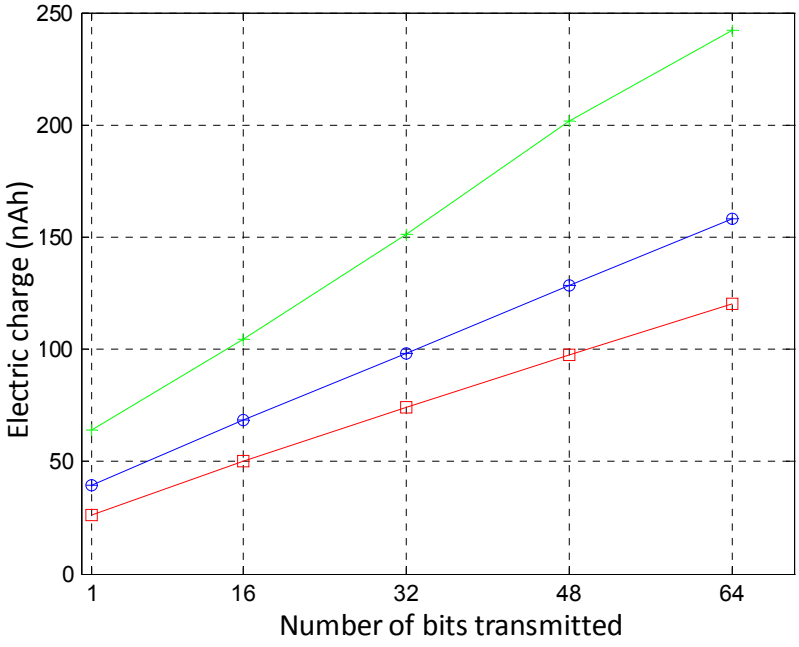

Figure 8. Electric charge for different transmission power and payloads.

At the reception side, the TR modules implement three reception modes: Standard (STD), Low Power (LP) and Extra Low Power (XLP). At reception, the mode control function relays on a parameter called *toutRF*, which indicates the number of times the module checks for incoming packets before exiting the reception function. In STD_RX mode, the receiver listens actively for incoming data in intervals that are multiples of 10 ms, as determined by the parameter *toutRF*. In LP_RX mode, the module listens for 10 ms and then sleeps to complete a 46-ms cycle that will be repeated *toutRF* times. In XLP_RX mode, the sleep period reaches 790 ms to complete a cycle. Several experiments have been conducted in order to measure the current in each mode. Tables 7 and 8 respectively summarize the average current and electric charge measured at every node in reception mode.

By means of a regression fit, it is possible to obtain an empirical model that allows the authors to predict the energy consumption at reception depending on the reception mode used (STD/LP/XLP) and consequently scaling the capacity of the batteries.

Table 9 shows the electric charge drained from the battery at reception operations as a function of the selected *toutRF* parameter.

Table 7. Average current drained in reception mode.

RX Mode	ToutRF (Times)	Current (mA)	Average (mA)
STD	50	12.6	12.8
	100	12.5	
	200	13.3	
LP	11	0.24	0.24
	30	0.23	
	43	0.24	
XLP	1	0.13	0.014
	2	0.14	
	3	0.14	

Table 8. Electric charge drained in reception.

RX Mode	ToutRF (times)	Charge (nAh)
STD	50	1758.89
	100	3490.56
	200	7409.46
LP	11	67.59
	30	182.05
	43	264.39
XLP	1	2.87
	2	6.16
	3	9.12

Table 9. Empirical model for reception: electric charge drained in reception mode.

RX Modes	Linear Fit
STD	$rec_{STD}(tout_{RF}) = 37.887\ tout_{RF} - 200.56$
LP	$rec_{LP}(tout_{RF}) = 6.1408\ tout_{RF} - 0.5983$
XLP	$rec_{XLP}(tout_{RF}) = 3.125\ tout_{RF} - 0.2$

Figure 9 represents the charge drained from the battery for the three modes when configured for equivalent periods of time by translating the *toutRF* parameter into seconds.

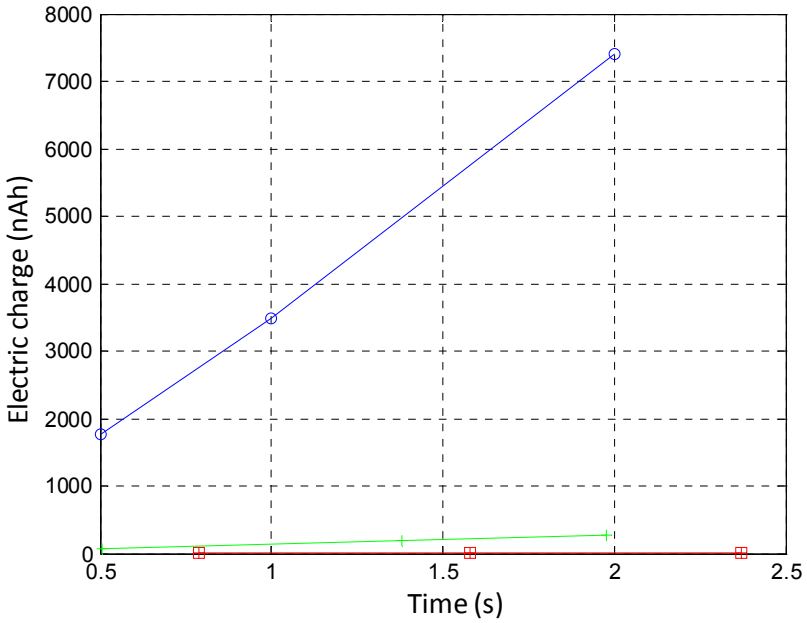

Figure 9. Charge drained from battery over time (blue, STD; green, LP; red, XLP).

Once the current consumption is characterized by the given empirical model (see Tables 6 and 9) expressed with equations that depend on the operating conditions (transmission level from zero to seven and reception level, STD/LP, XLP), this model can be used to schedule the transmission and reception events that fit an application according to the battery availability. The amount of energy used during the acquisition of the sensors, their processing or the energy budget required by additional hardware have not been taken into account, and their contribution should not be neglected in practical applications.

6. Conclusions

This work is devoted to facilitate the construction of low-cost/ low-energy IoT applications. Its major outcomes are: (1) one architecture for IoT gateways that integrates data from sensor nodes into higher-level applications (e.g., cloud applications); (2) bindings in Java and C that ease the construction of IoT applications; (3) an empirical model that describes the consumption of the communications at the nodes (smart sensors) and allows scaling the batteries; and (4) validation of the proposed energy model at the battery-operated nodes.

The proposed architecture is based on the Raspberry Pi platform due to its remarkable characteristics, namely: (1) good computing power/cost ratio; (2) high availability; currently, it has become a de facto hardware standard; and (3) ease of use, since it is based on operating systems with a big community of users. The IQRF WSN versatile technology is also proposed to acquire information from scattered sensors. From the energy management perspective for communications, the IQRF technology presents interesting benefits, such as (1) modifying the power level at the sending nodes and (2) adapting the reception times in order to save energy at the end nodes. These properties allow specifying different network configurations that adapt better to the QoS and energy requirements of the applications. Such low consumption levels were confirmed in this work by means of experimental results. In addition, according to the literature, the IQRF technology provides good propagation distances when compared with other technologies, especially in outdoor applications.

The wrappings presented by the authors allow designers to build high level applications in C and Java programming languages easily. These wrappings allow acquiring data from scattered sensors in an easy to use way, while keeping deep control of the IQRF WSN network energy consumption. The programmers of the applications will be able to build new energy-efficient IoT applications

that integrate field information without major difficulty. These wrappings can be used to specify the configuration of the IQRF parameters of the network or even modify them at run-time (reconfiguration) when the energy or QoS requirements of the applications demand it.

Since the management of the resources in IoT applications may become a key issue, the authors also present an experimental model for the energy consumption of the communications at the IQRF nodes that allows designers of IoT applications to scale the capacity of the batteries of the scattered sensors.

In summary, this article solves the research question presented at the beginning of the article: how to build gateways for IoT applications with Raspberry Pi and low power IQRF communication modules. Its major outcome is providing application designers all components to build IoT applications quickly and easily. This includes: (1) an architecture for the IoT gateways, together with all of the selected hardware components (Raspberry Pi, IQRF nodes, etc.); (2) bindings to build high level applications with the most common programming languages, such as C and Java (bindings for other programming languages, like Python, could be easily created); (3) a protocol that allows collecting data from the IQRF remote nodes (such as temperature, voltage, analog sensors, and LED manipulation); and (4) an empirical model for selecting the capacity of the batteries that helps with predicting their lifetime due to communication issues.

Supplementary Materials: The source code for the wrapper is available under the following doi: http://dx.doi.org/10.5281/zenodo.61071.

Acknowledgments: This work was supported in part by the University of the Basque Country (UPV/EHU) under projects EHU13/42 and UFI11/28 and by the Basque Government (GV/EJ) under projects CPS4PSS ETORTEK14/10 and Thinking Factory ETORGAI14.

Author Contributions: I.C. and J.M.G.-G. conceived of and designed the architecture and conceived of the experiments. I.R. implemented the IQRF code and the C wrappings. A.L. provided the Java wrapping. J.Q. supervised the work and the manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Li, S.; Xu, L.D.; Zhao, S. The internet of things: A survey. *Inf. Syst. Front.* **2015**, *17*, 243–259. [CrossRef]
2. Evans, D. The Internet of Things. How the Next Evolution of the Internet is Changing Everything. Cisco Internet Business Solutions Group (IBSG), April 2011. Available online: http://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf (accessed on 15 April 2015).
3. Razzaque, M.A.; Milojevic-Jevric, M.; Palade, A.; Clarke, S. Middleware for internet of things: A survey. *IEEE Internet Things J.* **2016**, *3*, 70–95. [CrossRef]
4. Bazydło, P.; Dąbrowski, S.; Szewczyk, R. Distributed temperature and humidity measurement system utilizing IQMESH wireless routing algorithms. In *Progress in Automation, Robotics and Measuring Techniques*; Szewczyk, R., Zieliński, C., Kaliczyńska, M., Eds.; Advances in Intelligent Systems and Computing, 352; Springer International Publishing: Basel, Switzerland, 2016; pp. 1–9.
5. Díaz, M.; Martín, C.; Rubio, B. State-of-the-art, challenges, and open issues in the integration of Internet of things and cloud computing. *J. Netw. Comput. Appl.* **2016**, *67*, 99–117. [CrossRef]
6. Noguero, A.; Calvo, I.; Almeida, L.; Gangoiti, U. A model for system resources in flexible time-triggered middleware architectures. In *Information and Communication Technologies*; Szabó, R., Vidács, A., Eds.; Lecture Notes in Computer Science (LNCS), 7479; Springer: Heidelberg, Germany, 2012; pp. 215–226.
7. Molano, J.I.R.; Betancourt, D.; Gómez, G. Internet of things: A prototype architecture using a raspberry Pi. In *Knowledge Management in Organizations*; Uden, L., Heričko, M., Ting, I.-H., Eds.; Lecture Notes in Bussiness Information Processing (LNBIP) 224; Springer International Publishing: Basel, Switzerland, 2015; pp. 618–631.
8. Bruce, R.F.; Dean Brock, J.; Reiser, S.L. Make space for the Pi. In Proceedings of the IEEE SouthEastCon, Fort Lauderdale, FL, USA, 9–12 April 2015; pp. 1–6.

9. Pavithra, D.; Balakrishnan, R. IoT based monitoring and control system for home automation. In Proceedings of the Global Conference on Communication Technologies, GCCT 2015, Thuckalay, India, 23–24 April 2015; pp. 169–173.
10. Raguvaran, K.; Thiyagarajan, J. Raspberry PI based global industrial process monitoring through wireless communication. In Proceedings of the 2015 International Conference on Robotics, Automation, Control and Embedded Systems, RACE 2015, Chennai, India, 18–20 February 2015.
11. Segura-Garcia, J.; Felici-Castell, S.; Perez-Solano, J.J.; Cobos, M.; Navarro, J.M. Low-cost alternatives for urban noise nuisance monitoring using wireless sensor networks. *IEEE Sens. J.* **2015**, *15*, 836–844. [CrossRef]
12. Jain, S.; Vaibhav, A.; Goyal, L. Raspberry Pi based interactive home automation system through E-mail. In Proceedings of the 2014 International Conference on Reliability, Optimization and Information Technology (ICROIT2014), Faridabad, India, 6–8 February 2014; pp. 277–280.
13. Gill, A.Q.; Phennel, N.; Lane, D.; Phung, V.L. IoT-enabled emergency information supply chain architecture for elderly people: The Australian context. *Inform. Syst.* **2016**, *58*, 75–86. [CrossRef]
14. Harting MICA Computing System. Available online: http://www.harting-mica.com/en/home/ (accessed on 30 April 2015).
15. Margelis, G.; Piechocki, R.; Kaleshi, D.; Thomas, P. Low Throughput Networks for the IoT: Lessons learned from industrial implementations. In Proceedings of the IEEE World Forum on Internet of Things, WF-IoT, Milan, Italy, 14–16 December 2015; pp. 181–186.
16. LoRaWAN: A Technical Overview of LoRa and LoRaWAN. Available online: https://www.lora-alliance.org/portals/0/documents/whitepapers/LoRaWAN101.pdf (accessed on 18 August 2016).
17. SIGFOX. Available online: http://sigfox.com/en (accessed on 18 August 2016).
18. IQRF Web Page. Available online: http://www.iqrf.org (accessed on 15 April 2016).
19. Duffy, A.; O'Donnell, F.J. A design research approach. In Proceedings of the AID'98 Workshop on Research Methods in AI in Design, Lisbon, Portugal, 19 July 1998; pp. 20–27.
20. Xu, L.D.; He, W.; Li, S. Internet of things in industries: A survey. *IEEE Trans. Ind. Inform.* **2014**, *10*, 2233–2243. [CrossRef]
21. Botta, A.; De Donato, W.; Persico, V.; Pescapé, A. Integration of Cloud computing and Internet of Things: A survey. *Future Gener. Comput. Syst.* **2016**, *56*, 684–700. [CrossRef]
22. Miorandi, D.; Sicari, S.; De Pellegrini, F.; Chlamtac, I. Internet of things: Vision, applications and research challenges. *Ad Hoc Netw.* **2012**, *10*, 1497–1516. [CrossRef]
23. Gazis, V.; Gortz, M.; Huber, M.; Leonardi, A.; Mathioudakis, K.; Wiesmaier, A.; Zeiger, F.; Vasilomanolakis, E. A survey of technologies for the internet of things. In Proceedings of the IWCMC 2015 International Wireless Communications and Mobile Computing Conference, Dubrovnik, Croatia, 24–28 August 2015; pp. 1090–1095.
24. Leccese, F.; Cagnetti, M.; Trinca, D. A smart city application: A fully controlled street lighting isle based on Raspberry-Pi card, a ZigBee sensor network and WiMAX. *Sensors* **2014**, *14*, 24408–24424. [CrossRef] [PubMed]
25. Sapes, J.; Solsona, F. FingerScanner: Embedding a Fingerprint Scanner in a Raspberry Pi. *Sensors* **2016**, *16*, 220. [CrossRef] [PubMed]
26. Lee, J.S.; Su, Y.W.; Shen, C.C. A Comparative Study of Wireless Protocols: Bluetooth, UWB, ZigBee, and Wi-Fi. In Proceedings of the 2007 33rd Annual Conference of the IEEE Industrial Electronics Society, IECON 2007, Taipei, Taiwan, 5–8 November 2007; pp. 46–51.
27. Baker, N. ZigBee and Bluetooth strengths and weaknesses for industrial applications. *Comput. Control Eng. J.* **2005**, *16*, 20–25. [CrossRef]
28. Yick, J.; Mukherjee, B.; Ghosal, D. Wireless sensor network survey. *Comput. Netw.* **2008**, *52*, 2292–2330. [CrossRef]
29. Abbas, Z.; Yoon, W. A survey on energy conserving mechanisms for the internet of things: Wireless networking aspects. *Sensors* **2014**, *15*, 24818–24847. [CrossRef] [PubMed]
30. Anastasi, G.; Conti, M.; Di Francesco, M.; Passarella, A. Energy conservation in wireless sensor networks: A survey. *Ad Hoc Netw.* **2009**, *7*, 537–568. [CrossRef]
31. Seflova, P.; Sulc, V.; Pos, J.; Spinar, R. IQRF wireless technology utilizing IQMESH protocol. In Proceedings of the 2012 35th International Conference on Telecommunications and Signal Processing (TSP), Prague, Czech Republic, 3–4 July 2012; pp. 101–104.

32. Hajovsky, R.; Pies, M. Use of IQRF technology for large monitoring systems. *IFAC PapersOnline* **2015**, *48*, 486–491. [CrossRef]
33. Bazydło, P.; Dąbrowski, S.; Szewczyk, R. Wireless temperature measurement system based on the IQRF platform. In *Mechatronics: Ideas for Industrial Applications*; Awrejcewicz, J., Szewczyk, R., Trojnacki, M., Kaliczyńska, M., Eds.; Advances in Intelligent Systems and Computing, 317; Springer International Publishing: Basel, Switzerland, 2015; pp. 281–288.
34. Pies, M.; Hajovsky, R.; Ozana, S.; Haska, J. Wireless sensory network based on IQRF technology. In Proceedings of the 2014 the 4th International Workshop on Computer Science and Engineering-Winter, WCSE 2014, Dubai, UAE, 22–23 August 2014.
35. TR-52D Transceiver Module Data Sheet. Available online: http://www.iqrf.org/products/transceivers/tr-52d (accessed on 30 April 2015).
36. Wang, Q.; Hempstead, M.; Yang, W. A realistic power consumption model for wireless sensor network devices. In Proceedings of the 3rd Annual IEEE Communications Society Sensor Ad Hoc Communications and Networks (SECON), Reston, VA, USA, 28 September 2006; Volume 1, pp. 286–295.
37. Martinez, B.; Vilajosana, X.; Chraim, F.; Vilajosana, I.; Pister, K. When scavengers meet industrial wireless. *IEEE Trans. Ind. Electron.* **2015**, *62*, 2994–3003. [CrossRef]
38. Kamath, S.; Lindh, J. AN092—Measuring Bluetooth Low Energy Power Consumption. Texas Instruments. Available online: http://www.ti.com.cn/cn/lit/an/swra347a/swra347a.pdf (accessed on 30 April 2015).
39. Kamath, S.; Lindh, J. AN079—Measuring Power Consumption of CC2530 with Z-Stack. Texas Instruments. Available online: http://www.ti.com/lit/an/swra292/swra292.pdf (accessed on 30 April 2015).