An Implementation for Dynamic Application Allocation in Shared Sensor Networks

Carmen Delgado, Sergio Batista, María Canales, José Ramón Gállego, Jorge Ortín, Matteo Cesana

An Implementation for Dynamic Application Allocation in Shared Sensor Networks

Carmen Delgado, Sergio Batista,
María Canales, and José Ramón Gállego
Aragón Institute of Engineering Research
Universidad de Zaragoza
Zaragoza, Spain
Email: cdelga@unizar.es

Jorge Ortín,
Centro Universitario de la Defensa and
Aragón Institute of Engineering Research
Universidad de Zaragoza
Zaragoza, Spain
Email: jortin@unizar.es

Matteo Cesana
Dipartimento di Elettronica,
Informazione e Bioingegneria
Politecnico di Milano
Milano, Italy
Email: matteo.cesana@polimi.it

***Abstract*—We present a system architecture implementation to perform dynamic application allocation in shared sensor networks, where highly integrated wireless sensor systems are used to support multiple applications. The architecture is based on a central controller that collects the received data from the sensor nodes, dynamically decides which applications must be simultaneously deployed in each node and, accordingly, over-the-air reprograms the sensor nodes. Waspmote devices are used as sensor nodes that communicate with the controller using ZigBee protocol. Experimental results show the viability of the proposal.**

***Keywords*—** Implementation, Over-The-Air Programming, Shared Sensor Networks, Wireless Sensor Networks, ZigBee.

I. Introduction

Wireless Sensor Networks (WSNs) are one of the key enabling building blocks for the Internet of Things. Looking back at the evolution of WSNs, a clear trend can be observed moving from stand-alone, application-specific deployments to the emergence of *shared sensor networks*, where highly integrated wireless sensor systems are used to support multiple services and applications in deployment domains such as Smart Cities, Smart Home and Buildings and Intelligent Transportation Systems.

The aforementioned trend calls for novel design good practices to overcome the limits in flexibility, efficiency and manageability of vertical, task-oriented and domain-specific WSNs. Being the research field that recent, a common terminology is still missing and the technical papers often use different wording for similar concepts; as an example, *shared sensor networks*, *virtual sensor networks* and *multi-application sensor networks* are often used almost interchangeably in the literature. The interested reader may refer to the following surveys on the topic [1], [2]. In this work, we will use the term *shared sensor network* (SSN) to define a physical sensor network infrastructure which can be used to support multiple concurrent applications and services, and where the ownership of the physical infrastructure is decoupled from the ownership of the applications and services. Generally speaking, the efficient realization of SSNs requires technologies and solutions in different domains ranging from the node level, where the sensor nodes must be able to support and run applications in a transparent way, up to the network level, where effective platforms and solutions are required to manage and reconfigure on-the-fly the network resources.

At the node level, the design of abstraction layers and primitives on a single sensor node to overcome the problem of application-platform dependency has been already addressed. As an example, Mate [3], ASVM [4], Melete [5] and VMStar [6] are frameworks for building application-specific virtual machines over constrained sensor platforms. At the network level there are two main building blocks which are usually tightly coupled: (i) management platforms to support multiple application sharing a common physical infrastructure, and (ii) tools/algorithms to allocate the physical resources to the multiple applications. Representative examples of management platforms are SenSHare [7] and UMADE [8], which create multiple overlay sensor networks which are "owned" by different applications on top of a shared physical infrastructure. As far as resource allocation in SSNs is concerned, the authors of [9] focus on environmental monitoring applications and propose a centralized optimization framework which allocates applications to sensor nodes minimizing the variance in sensor readings. Ajmal *et al.* [10] propose a decision algorithm to dynamically "admit" applications to a physical sensor network infrastructure. The research streamline on *sensor mission assignment* [11], [12] also addresses a resource allocation problem in WSNs; namely, the problem is to jointly allocate the physical sensor resources to incoming applications while incorporating admission control policies. In our past work [13], we extended those works considering networking-related aspects, including the possibility to re-configure the sensor network by moving applications which were previously deployed, and further modelling situations where multiple applications can be concurrently deployed at a sensor node.

While our work in [13] focuses on a mathematical programming framework to model the joint problem of application/service admission control and resource allocation to the admitted applications, in this work we present a system architecture and implementation that provides the basis to deploy those features in an actual SSN. The remaining manuscript is organized as follows: Section II describes the system architecture and its implementation (communications, sensor nodes, applications, central controller). In Section III

ISBN 978-3-903176-03-4

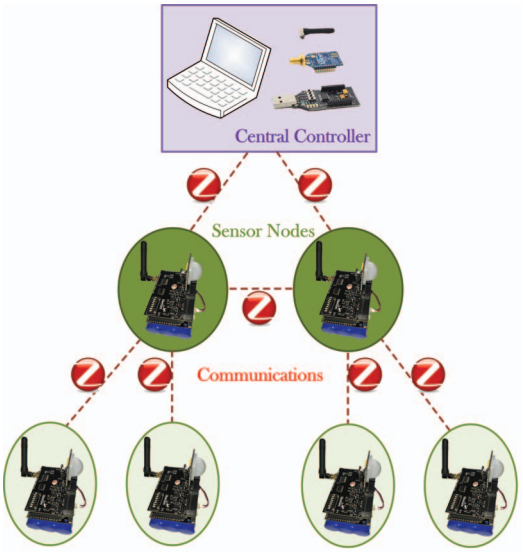


Fig. 1. Network topology scenario

experimental results validating the implementation are shown and concluding remarks are finally reported in Section IV.

II. System Architecture and Implementation

The proposed system architecture aims to manage a shared physical WSN over time. As stated above, we look at the reference scenario where a single SSN infrastructure provider (SSN-IP) owns an infrastructure which can be accessed by multiple application providers which issue requests to deploy specific applications/services.

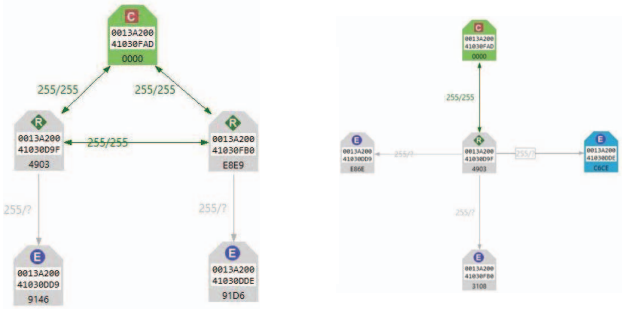
In the aforementioned scenario, different applications will be running in the sensor nodes. Each of these nodes will send the monitored data through multihop wireless communications to a sink node, which acts as a central controller. This controller not only collects the received data from the sensor nodes, but dynamically decides which applications must be deployed in each node and, accordingly, reprograms the sensor nodes. Figure 1 schematically shows this architecture composed by three main elements: central controller, sensor nodes and communications. In the following we describe in detail these elements and their practical implementation.

A. Communications

ZigBee protocol [14] has been chosen to implement wireless multihop communications between the sensor nodes and the central controller. ZigBee is an IEEE 802.15.4-based standard characterized by high reliability, low cost, low power, scalability and low data rate (up to 250 kbps), which is commonly used in WSN implementations [15]. More specifically, XBee ZigBee-Pro S2 modules [16] have been used.

It must be noted that in a ZigBee network there are three different device types: Coordinator, Router and End device:

- **Coordinator**: each ZigBee network must have one. It selects the channel and PAN ID to start the network, allows routers and end devices to join the network, assists in routing data and has to be always awake.



(a) (b)

Fig. 2. Network topology examples

- **Router**: It must join a ZigBee network before it can transmit, receive or route data. After that, it can allow routers and end devices to join the network and route data. It has to be always awake.
- **End device**: It must join a ZigBee network before it can transmit or receive data. To do so, it must be associated to a router or a coordinator (its parent). It must always transmit and receive RF data through its parent and can neither allow devices to join the network nor route data. On the contrary, it can sleep.

In our implementation, the central controller acts as the coordinator, creating and managing the network whereas the sensor nodes are configured either as routers or end devices, depending on the network requirements and topology. As stated above, routers and coordinator can route data and are interconnected forming a mesh topology with multihop routing. However, each end device is only associated to a parent node (router or coordinator). This can be seen in Figure 1, where the coordinator and all the routers (darker green) within transmission range can directly communicate with each other while the end devices can only communicate with their parents. In the end, the deployed topology enables a bidirectional communication between all the devices.

The XBee modules for each type of device have been configured with the XCTU software [17]. Figure 2 shows some of the configured reference scenarios. As previously explained, only end devices are allowed to sleep in a ZigBee network. Thus they are configured in sleep mode and wake up periodically to send the gathered information or to check if they have anything to receive, as it is detailed in section II-B. If an end device is sleeping when its parent has something to send to it, the router keeps the data in its buffer until the end device wakes up and asks if there is any message for it.

Finally, it is important to note that the communication between the XBee RF Modules and the host device (sensor node or central controller) is done through a logical-level asynchronous serial port [16]. The maximum transmission rate for this serial communication is 115200 bps, being the value configured for the sensor nodes in our implementation. For the coordinator, the value has been configured to 38400 bps, as it is explained in section II-C. Therefore, as shown in Figure 3, even in a single-hop wireless link between two

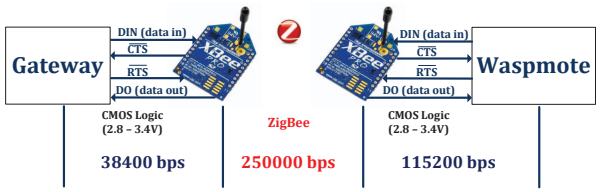


Fig. 3. Serial and wireless communications between XBee-based devices

devices, there are three communications links, which can limit the transmission data rate of a ZigBee application.

B. Sensor nodes

1) Hardware: The devices used as sensor nodes are Waspmotes Pro (v1.2) developed by Libelium [18]. Waspmote is an open source wireless sensor platform specially focused on the implementation of low consumption nodes that are completely autonomous and battery powered, offering a variable lifetime between 1 and 5 years depending on the duty cycle and the radio used. They have an ATmega 1281 processor at 14.7456 MHz, SRAM of 8 KB, EEPROM of 8KB and Flash of 128 KB. They also have a micro SD slot with a 2GB SD card. In addition, they are equipped with a Real Time Clock (RTC) that controls the low consumption modes, and a battery connector with a rechargeable Lithium Ion battery of 6600 mAh. The Waspmotes are connected to the XBee ZigBee-Pro S2 module through Socket 0 and to an Events Sensor Board v2.0 [19] through the I/O Sensor pins. This board has 8 sockets to connect different sensors, where Socket 7 is used for digital signals and the others for analogical ones. Four different sensors have been used in our implementation: the MCP9700A Temperature Sensor (connected to Socket 5), the 808H5V5 Humidity Sensor 3.3V (Socket 6), the LDR Luminosity Sensor (Socket 2) and the PIR Presence Sensor (Socket 7). Detailed features of these sensors can be found in [19]. Table I summarizes the energy consumption involved in the Waspmotes operating as sensor nodes in our network.

2) Software: Four different applications can be deployed on the sensor nodes in our testbed: temperature monitoring, humidity monitoring, luminosity monitoring and presence detection. The first three ones monitor the corresponding variable of interest periodically at a configurable sensing interval, whereas the last one works by events that are generated when presence is detected. These applications have been developed using the Integrated Development Environment for Waspmote (Waspmote IDE) [20], which is compatible with the Arduino programming environment. Thus, Waspmotes can only execute a given firmware at a time. Since our proposal aims to simultaneously deploy any subset of the available applications at a sensor node, the adopted solution has been to develop 15 different firmwares, one for each possible combination of the four considered applications. They are shown in table II. For instance, firmware 1 just monitors temperature, while firmware 7 monitors luminosity, humidity and temperature. It is worth mentioning that even if only one firmware can be executed at a time, the remaining ones can be stored in the SD card and loaded when needed, as it will be later explained.

TABLE I
ENERGY CONSUMPTION VALUES

Module	Mode	Consumption
Waspmote	ON	15 mA
	Sleep	55 μA
XBee ZigBee PRO	ON	45.56 mA
	Sleep	0.71 mA
	Sending	105 mA
	Receiving	50.46 mA
Events Sensor Board	Minimum (constant)	3.6 μA
	Connector2 + LDR S.	(32-64) + 0 μA
	Connector5 + Temperature S.	32 + 6 μA
	Connector6 + Humidity S.	32 + 180 μA
	Connector7 + PIR S.	0 + 100 μA
	Reading register (<20ms)	150 μA
SD	ON	0.14 mA
	Reading	0.2 mA
	Writing	0.2 mA
	OFF	0 mA

TABLE II
FIRMWARE CODIFICATION

Firmware	PIR	Luminosity	Humidity	Temperature
1				x
2			x	
3			x	x
4		x		
5		x		x
6		x	x	
7		x	x	x
8	x			
9	x			x
10	x		x	
11	x		x	x
12	x	x		
13	x	x		x
14	x	x	x	
15	x	x	x	x

These firmwares have to perform two main functionalities: monitoring/sensing the parameters required by the applications (with configurable periodic sensing intervals or by events) and listening to the messages sent by the central controller (mainly firmware updates or changes in the applications sensing intervals). These two functionalities have been handled through alarms that allow waking up the Waspmotes from the low energy consumption state to perform the sensing and listening procedures, as it is explained next.

Figure 4 shows schematically the execution of these functionalities. Once the required libraries are included and variables are declared, the setup block configures the following modules: (i) XBee module for communications; (ii) RTC

module for alarm programming; and (iii) Events Sensor Board for interpreting data from the sensors and transmitting them to the microcontroller. Once a device is connected to the network, it sends an information message to the central controller to inform about which applications are running and with which parameters. This message has the following format:

$$|\#||INFO||\#||APP:x||\#||SENSOR:y||\#||LSTN:z|$$

where # is the delimiter, INFO is the frame identifier, APP:x is the firmware that it is running x $\in$ [1,...15], SENSOR:y is the sensing interval (in seconds) used by each application (SENSOR $\in$ [TEMP, HUM, LDR]; PIR is not included since it works by events), and LSTN:z is the listening interval (in minutes), with z $\in$ N.

Finally, the alarms are initialized. *Alarm1* is used for the sensing intervals and *alarm2* is used to program the listening intervals, which allow checking with the controller for updates of the firmware. We set this interval to 1 minute.

Then, the device starts executing the loop() block, where the Waspmote remains in the Sleep mode until any interruption happens. This low consumption mode allows configuring the modules that should or not sleep. This is necessary since, as stated in section II-A, the XBee module cannot be switched off in the routers, where applications can also be deployed.

PIR detection is activated with an asynchronous interruption when the sensor detects any movement, so it is originated by the Events Sensor Board and it is not programmable. Since this interruption is not predicted, to avoid missing any other interruption that may occur at time instants close to it, *alarm1* and *alarm2* are reprogrammed after this detection. When the *alarm1* interruption happens (i.e., when it is time to measure the temperature, humidity, luminosity or a combination of them), data are sensed, encapsulated in a frame and sent to the central controller. This frame also includes the information about the remaining battery life in the sensor node, that will be used by the central controller to reallocate applications when needed. Finally, the next *alarm1* is programmed.

As stated before, the central controller can send both firmware updates or changes in the application parameters. Firmware updates are performed using the OTA-Shell provided by Libelium [21], a functionality which allows the Over-The-Air Programming of Waspmotes through a multihop ZigBee network, which will be further explained in section II-C.

The XBee module distinguishes between the reception of standard and OTA frames: standard ones are directly processed and data is extracted, whereas for OTA frames, before the processing, it is checked which command contain. In order to process both type of frames using only one interruption, in our implementation the controller first sends a message indicating which kind of frame (standard or OTA) will send.

So when the *alarm2* interruption happens, the Waspmote sends a message to the central controller advising that it will start listening, thanks to the #LISTEN command. After this, the Waspmote starts listening if any message from the central controller has been sent. If nothing is received, it goes to the sleep mode, whereas if it receives something, it first checks if

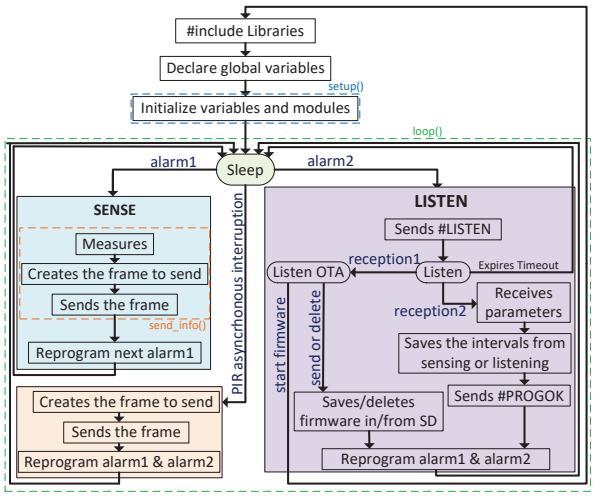


Fig. 4. Sensor node firmware schema

```
# ./otap
You have to specify one command
usage: otap
 -delete_program       deletes a specified program
 -get_boot_list        get the boot list of nodes
 -info_program         shows info about hex file
 -scan_nodes           scan for active nodes
 -send                 send firmware to nodes
 -start_new_program    start specified program
 -version              show version of the program
```

Fig. 5. Commands from the OTA-Shell

it is an OTA or a standard frame to set the suitable reception mode. For OTA frames, if the $-send$ command is received, the device will save the firmware in the SD card, whereas if the $-delete\_program$ command is received, the device will access the SD storage and will delete the specified firmware. If the received command is a $-start\_new\_program$, it will stop its execution to restart the device and execute the new firmware.

On the other hand, when a standard frame with application configuration parameters is received, the device will process them and it will send an acknowledgment message (#PROGOK) to the central controller to confirm the update of the application parameters. Finally, both *alarm1* and *alarm2* are reprogrammed. The reprogramming of *alarm1* is required because if the reception of a new firmware takes too much time, it could cause the lost of a preceding *alarm1* and consequently that alarm would not be reprogrammed anymore.

C. Central controller

The central controller has been implemented through a Java application deployed in a computer with a Java virtual machine. In order to provide ZigBee connectivity, an USB Waspmote Gateway from Libelium [18] with an XBee ZigBee-Pro S2 module has been used (see purple rectangle of central controller in Figure 1). The OTA-Shell specifications require configuring the serial communications between this Gateway and the XBee module to 38400 bps [21]. The election of

Java allows using the XBee Java Library, required to interact with the XBee module. Along with the Java application it is also used the aforementioned OTA-Shell provided by Libelium (Figure 5) to perform the OTA Programming of the Waspmotes. In addition, to manage all the collected data from the sensor nodes in the Java application, a database has been implemented with an XAMPP server managed with MySQL through PhpMyAdmin. Finally, as a proof of concept, a simple graphic web interface hosted in the XAMPP server has been designed with PHP to show in real time the data from the nodes.

*1) **Java application**:* The Java application is the main module of the central controller, where the system intelligence will reside. From the perspective of the SSN-IP that owns the infrastructure, this central controller should address the problem on how to optimally manage the application requests coming from different application providers to maximize the total revenue, deciding whether and when to admit an application request and how to reconfigure consequently the current physical resource assignment to applications, as we show in [13]. To this purpose, parameters such as the activity time (the required lifetime of the application in the network) and a set of requirements to deploy the application can be considered in the decision process. These requirements could be the sensing area to be covered, the required processing and storage capabilities of the node(s) hosting the application, and the required communication bandwidth to deliver application data remotely across the physical sensor network. It is worth mentioning that in this work we focus on the implementation of the practical mechanisms to deploy multiple applications on the sensor nodes or reallocate them using OTA, rather than the decisions of where and when to perform these allocations, which we analyzed in [13].

Figure 6 shows the flow diagram of the Java application that the controller runs. When a new application arrives at the system, an algorithm should select the best node or nodes to allocate the application. As a simple example, we choose the node with the highest battery life. According to the status of the selected node, four different cases are possible:

1) Application is already running with the same sensing interval: Nothing is required. The current deployed application can provide the required data.
2) Application is running in the node with different sensing interval: A new configuration fulfilling both sensing intervals must be generated by the controller and sent to the node. As can be seen in the purple area of Figure 6, the controller waits until it receives the #LISTEN command from the node and then sends the new configuration frame to the node. When the acknowledgment is received, the database is updated.
3) Application firmware is saved in the SD card of the node but it is not running: The $-start\_new\_program$ OTA command is sent after receiving the #LISTEN command. Once the acknowledgment is received, the database is updated. Note that this stored firmware must include the code of the active applications at that moment plus the new one. For example (as can be seen in Table II), if firmware 4 is running (luminosity) and a humidity application arrives to the system, firmware 6 must be activated.
4) Application firmware is not in the node: The $-send$ OTA command with the new firmware is sent after receiving the #LISTEN command. When the acknowledgment is received, the database is updated. After this, it is necessary to start this firmware, so the previous case (the blue one of the Figure 6) is also executed.

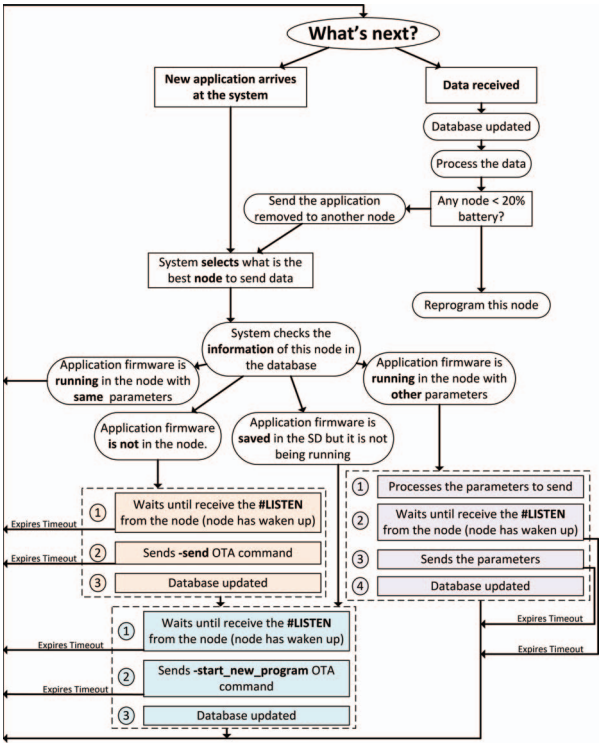


Fig. 6. Java application schema

In addition, and also as a proof of concept to validate the application reallocation, the battery information included in all the frames from the sensor nodes (see section II-B) is compared to a predefined threshold. This threshold is set to 20% because the manufacturer does not guarantee the correct reception and execution of the OTA-Shell commands under this value. Then, the central controller reprograms the node deleting the application with the highest energy consumption (i.e., starting a new firmware without this application, which should be also sent to the node if it is not already stored in the SD card) and sends it to another node with enough energy following the same steps as if it were a new one.

Finally, we describe in the following example how the controller informs to a sensor node about the sensing intervals of its different applications.

Let us assume we have to sense temperature (1) every 5 seconds, humidity (2) every 10 seconds and luminosity (3) every 15 seconds. This combination of applications corresponds to firmware 7. Figure 7 shows the time instants in which each

application will be sensed. Since every 30 seconds the same structure will be repeated, the following data frame is sent:

|2||<5><5><5><5><5><5><5>||-||<7><1><3><5><3><1><7>||

where the first |2| means that this is a standard data frame with sensing intervals, | <5><5><5><5><5><5> | are the time intervals between two sensings, and | <7><1><3><5><3><1><7> | represents the indexes of the firmwares whose applications must be sensed at each time.

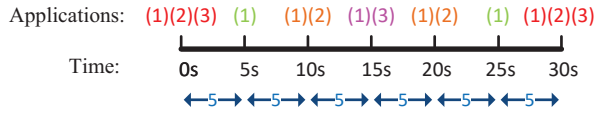


Fig. 7. Mechanism to process the intervals

*2) **Database**:* To manage the database MySQL through PhpMyAdmin are used, integrated in XAMPP, from where new databases and their tables can be created and registers can be inserted with SQL sentences. Figure 8 shows the entity relationship database diagram that will help to understand its functionality. The database is formed by four tables: *Register, Devices, Firmwares* and *OTAFirmwares*, where primary and foreign keys have been used. Primary ones are identified with *PK*, while the foreign ones are identified with *FKx*. The register table has all the registers that the controller receives from the nodes. Its primary key is the ID, whose value is increased everytime a new data is received. Foreign keys are *DeviceID* and *FirmwareID*, which allow to access to the device registers and the firmware registers. The *Devices* table has as primary key the *DeviceID* attribute, which is the identifier of each node. It contains the information of the nodes: battery, listening time and sensing intervals of the applications. The firmware table contains the information of the firmwares that can be executed in the nodes, and *FirmwareID* is its primary key. *OTAFirmwares* saves all the FirmwareIDs stored in the SD memory card of the network devices. Its primary key is *FirmwareOTAID*, and its foreign keys are *DeviceID* and *FirmwareID*.

*3) **Graphic web interface**:* A simple graphic web interface has been designed to monitor all the data sensed by the nodes in real time. It has been designed with PHP and is hosted in the XAMPP server. The Highcarts library has been used for the graphic design. The stored parameters can be shown by device or by application. Figure 9 shows a screenshot of the graphic web interface monitoring several applications.

III. Experimental Results

In this section, we evaluate the performance of the system implementation. Two main aspects have been analysed: the time required to perform a firmware update and the energy consumption of the sensor nodes.

A. Firmware update

Firmware update is one of the key elements to perform the adaptive resource allocation aimed with this implementation.

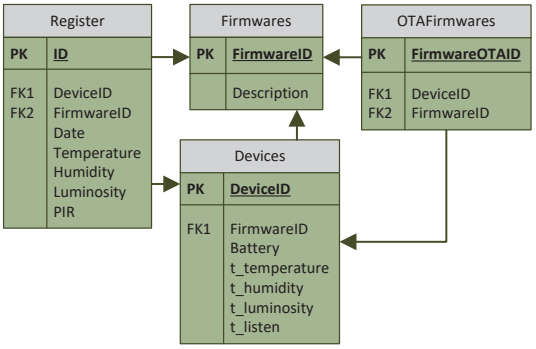


Fig. 8. Entity relationship database diagram

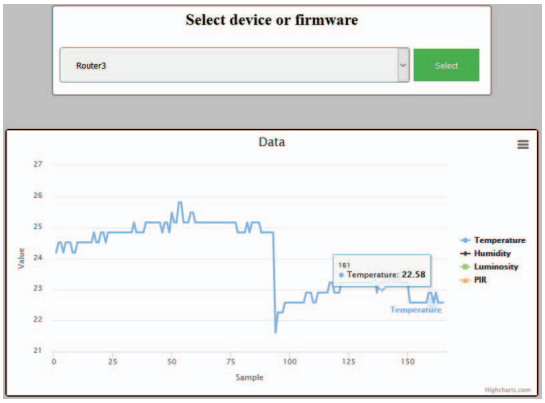


Fig. 9. Graphic web interface

Two possible scenarios can arise: If the firmware to be updated is already saved in the SD card of the sensor node, it only has to be started with the $-start\_new\_program$ command. However, if the firmware is not in the sensor node yet, it has to be sent from the central controller with the $-send$ command and then run with the $-start\_new\_program$ command.

Table III shows the results in terms of number of frames, firmware size, ZigBee data bytes sent (81 bytes per 802.15.4 frame), required time, and effective ZigBee data rate when updating firmware from the coordinator to a 1-hop router (Figure 2(a)) for different application sets.

As expected, the time required to activate an already stored firmware is much lower than the one required to send it, since only one frame has to be sent. A 2GB SD card allows to store more than 16000 firmwares, which are limited to 128 KB (flash memory of Waspmote, see section II-B1). In our example scenario, with 15 firmwares, all of them can be preloaded in the SD card.

It is also worth noting that when the firmware is sent, not only the amount of information to be sent is higher, but the effective transmission rate is lower. This result directly depends on the OTA-Shell implementation from Libelium, which is used to send the firmwares to the sensor nodes. As for the data rate obtained in the other case, in the XBee documentation [16] it is stated that empirical results show that the maximum achievable ZigBee data rates at one hop router

TABLE III
FIRMWARE UPDATE RESULTS FOR DIFFERENT APPLICATION SETS

Update	Firmware: Applications	Frames	Firmware size	Bytes sent	Time	Rate
***-send* firmware**	1: Temperature	982	74080 B	79542	115 s	5.53 kbps
	3: Temperature and Humidity	1058	79704 B	85698	129 s	5.31 kbps
	7: Temperature, Humidity and Luminosity	1061	79754 B	85941	132 s	5.2 kbps
	15: Temperature, Humidity, Luminosity and Presence	1071	80506 B	86751	134 s	5.17 kbps
-start new program	1: Temperature	1	-	19	0.011 s	13.17 kbps
	3: Temperature and Humidity	1	-	19	0.01 s	14.52 kbps
	7: Temperature, Humidity and Luminosity	1	-	19	0.0135 s	11.23 kbps
	15: Temperature, Humidity, Luminosity and Presence	1	-	19	0.009 s	15.24 kbps

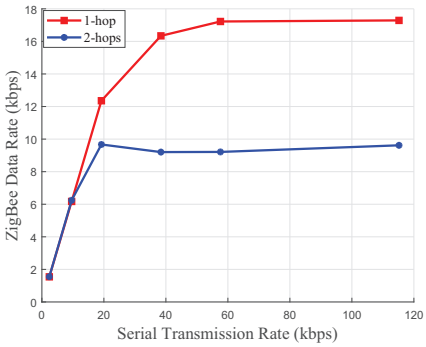


Fig. 10. Transmission rate from the controller

to router links with security disabled is 35 kbps when the serial communications between XBee module and device are set to 115200 bps. Note that in our scenario, serial communications between XBee module and central controller must be set to 38400 bps.

We have analyzed the maximum achievable data rate in our scenario both for 1-hop communications between the coordinator and a router and 2-hop communications between the coordinator and an end device (Figure 2(a)) varying the serial communications rate between XBee module and coordinator. Figure 10 shows the obtained results, slightly lower than the maximum values shown in [16], but confirming the data rates shown in table III.

B. Energy consumption

In this section, as an example to check the expected energy consumption in the sensor nodes, we have analyzed the case of a router in an extreme situation with very high energy consumption. The router is running a temperature application with a sensing interval of 10 s and it is listening to the coordinator 1 s every 60 s (as Figure 11 shows).

Eq. (1) shows its lifetime where EC is the total energy consumption, which is defined in eq. (2), and 6600 mAh is the battery capacity of the devices:

$$lifetime = \frac{6600mAh}{EC}, \quad (1)$$

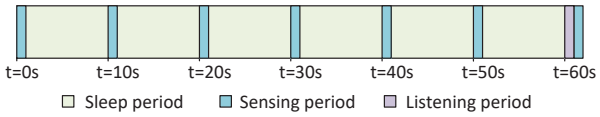


Fig. 11. Temporal diagram of a Waspmote program, sensing every 10s and listening every 1m

$$EC = SLPEC * t_{SLPEC} + SEC * t_{SEC} + LEC * t_{LEC}, \quad (2)$$

where $SLPEC$ is the sleep energy consumption, SEC is the sensing energy consumption, LEC is the listening energy consumption and t_{SLPEC}, t_{SEC} and t_{LEC} are the corresponding fractions of time at each state.

In the sleep period, the Waspmote is in Sleep mode (0.055 mA), the XBee module is always on, since it is acting as a router (45.56 mA), the Events Sensor Board consumes at its minimum (0.0036 mA) and the SD is always on (0.14 mA). t_{SLPEC} in this case is 0.9807.

SEC corresponds to the energy consumption related to the sensing itself and the sending of the sensed data when the Waspmote and its SD are always on (15mA and 0.14 mA, respectively). In this case, we distinguish between the sensing time, the ZigBee packet sending time and the ZigBee ACK receiving time. In the first case, the Xbee module is on (45.56mA) and the Events Sensor Board is sensing temperature (0.1916 mA, which also takes into account the reading register), which takes less than 20ms according to the specifications of the manufacturer [19] (fraction of time for sensing is 0.002). Secondly, the XBee is sending (105 mA) and the Events Sensor Board is in its minimum (0.0036 mA). The 802.15.4 temperature data frame has 102 bytes at PHY level, (around 3.3 ms, fraction of time for sending is 0.00033). Thirdly, the XBee will be receiving (50.46 mA) and the Events Sensor Board will be in its minimum again (0.0036 mA). The 802.15.4 frame for the ZigBee ACK frame has 49 bytes at PHY level (around 1.6 ms, fraction of time for receiving of 0.00016).

Finally, for the listening intervals, we have to consider that the node first sends the #LISTEN command (and the

corresponding ZigBee ACK from the coordinator is received), and then it waits 1 s for receiving messages from the central controller (fraction of time of 0.01667). Note that in this case no message from the central controller is sent. The Waspmote is on (15 mA), the Events Sensor Board is in its minimum value (0.0036 mA) and the SD is on (0.14mA). First, the XBee is sending (105mA) and since the 802.15.4 frame for the #LISTEN command has a length of 87 bytes, the transmission takes around 2.8 ms (fraction of time of 0.000046). The 802.15.4 frame for the ZigBee ACK frame has 49 bytes at PHY level (around 1.6 ms, fraction of time for receiving of 0.000026).

With these parameters, we theoretically estimate a lifetime of 5.96 days. According to the experimental results, obtained measurements have shown that in 7 hours running the temperature application the battery dropped 11%. From this result, we could estimate a battery duration of 2.65 days, which is in the same magnitude order. It must be noted that this reduced lifetime is due to the excessively short intervals chosen for the sensing and the listening and because calculations have been made at a router. Since ZigBee routers must have the XBee module active all the time, they are usually plugged into the grid or have an external energy source. In fact, the theoretical estimation of the lifetime of an end device (where the XBee module can be in sleep mode during the Waspmote sleep periods) running the same firmware, is 129.58 days.

IV. Conclusion

In this paper we have presented the implementation of a system architecture for wireless SSNs that allows dynamically allocating applications arriving to the system or reallocating already deployed applications through over-the-air programming of the sensor nodes, which support multiple concurrent applications. Experimental results have shown the viability of the implementation. It is worth noting that storing in the sensor nodes all the possible deployed firmwares highly improves the performance of the over-the-air programming, so the firmwares should be either preloaded before the deployment or stored after the first update. As future research lines, the deployment of a larger network, where some of our previously proposed resource allocation algorithms could be tested and the comparison with other schemes is foreseen.

Acknowledgment

This work was supported in part by the Spanish Government through the Ministerio de Ciencia e Innovación under grant TEC2014-52969-R, Universidad de Zaragoza through the grant UZ2018-TEC-04, Gobierno de Aragón, the European Social Fund (ESF), grant CAS17/00624 from the mobility program of the Ministerio de Educación, Cultura y Deporte and Centro Universitario de la Defensa through project CUD2017-18.

References

[1] I. Khan, F. Belqasmi, R. Glitho, N. Crespi, M. Morrow, and P. Polakos, "Wireless sensor network virtualization: A survey," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 553–576, Firstquarter 2016.

[2] C. M. D. Farias, W. Li, F. C. Delicato, L. Pirmez, A. Y. Zomaya, P. F. Pires, and J. N. D. Souza, "A systematic review of shared sensor networks," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 51:1–51:50, Feb. 2016. [Online]. Available: http://doi.acm.org/10.1145/2851510

[3] P. Levis and D. Culler, "Mate: A tiny virtual machine for sensor networks," *SIGARCH Comput. Archit. News*, vol. 30, no. 5, pp. 85–95, Oct. 2002. [Online]. Available: http://doi.acm.org/10.1145/635506.605407

[4] P. Levis, D. Gay, and D. Culler, "Active sensor networks," in *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, ser. NSDI'05. Berkeley, CA, USA: USENIX Association, 2005, pp. 343–356. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251203.1251228

[5] Y. Yu, L. J. Rittle, V. Bhandari, and J. B. LeBrun, "Supporting concurrent applications in wireless sensor networks," in *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, ser. SenSys '06. New York, NY, USA: ACM, 2006, pp. 139–152. [Online]. Available: http://doi.acm.org/10.1145/1182807.1182822

[6] J. Koshy and R. Pandey, "Vmstar: Synthesizing scalable runtime environments for sensor networks," in *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, ser. SenSys '05. New York, NY, USA: ACM, 2005, pp. 243–254. [Online]. Available: http://doi.acm.org/10.1145/1098918.1098945

[7] I. Leontiadis, C. Efstratiou, C. Mascolo, and J. Crowcroft, "Senshare: Transforming sensor networks into multi-application sensing infrastructures," in *Proceedings of the 9th European Conference on Wireless Sensor Networks*, ser. EWSN'12, 2012, pp. 65–81.

[8] S. Bhattacharya, A. Saifullah, C. Lu, and G. Roman, "Multi-application deployment in shared sensor networks based on quality of monitoring," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE*, April 2010, pp. 259–268.

[9] Y. Xu, A. Saifullah, Y. Chen, C. Lu, and S. Bhattacharya, "Near optimal multi-application allocation in shared sensor networks," in *Proceedings of the Eleventh ACM International Symposium on Mobile Ad Hoc Networking and Computing*, ser. MobiHoc '10. New York, NY, USA: ACM, 2010, pp. 181–190. [Online]. Available: http://doi.acm.org/10.1145/1860093.1860118

[10] S. M. Ajmal, S. Paris, Z. Zhang, and F. N. Abdesselam, "An efficient admission control algorithm for virtual sensor networks," in *High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS), 2014 IEEE Intl Conf on*, Aug 2014, pp. 735–742.

[11] T. L. Porta, C. Petrioli, C. Phillips, and D. Spenza, "Sensor mission assignment in rechargeable wireless sensor networks," *ACM Trans. Sen. Netw.*, vol. 10, no. 4, pp. 60:1–60:39, Jun. 2014. [Online]. Available: http://doi.acm.org/10.1145/2594791

[12] N. Edalat and M. Motani, "Energy-aware task allocation for energy harvesting sensor networks," *EURASIP Journal on Wireless Communications and Networking*, vol. 2016, no. 1, p. 28, Jan 2016. [Online]. Available: https://doi.org/10.1186/s13638-015-0490-3

[13] C. Delgado, M. Canales, J. Ortn, J. R. Gllego, A. Redondi, S. Bousnina, and M. Cesana, "Joint application admission control and network slicing in virtual sensor networks," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 28–43, Feb 2018.

[14] *ZigBee Specification*, r20 ed. ZigBee Standards Organization. ZigBee Alliance, Inc., sep 2012.

[15] T. Alhmiedat, "A survey on environmental monitoring systems using wireless sensor networks," *Journal of networks*, vol. 10, no. 11, pp. 606–615, 2015.

[16] *ZigBee RF Modules: XBee2, XBeePro2, Pro S2B. User Guide*. Digi International, jun 2017.

[17] [Accessed 13-April-2018]. [Online]. Available: https://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu

[18] [Accessed 13-April-2018]. [Online]. Available: http://www.libelium.com/

[19] *Events 2.0: Technical Guide, Waspmote*, v5.1 ed. Libelium Comunicaciones Distribuidas S.L., jan 2016.

[20] *Waspmote IDE, User Guide*, v4.1 ed. Libelium Comunicaciones Distribuidas S.L., jan 2014.

[21] *Over the Air Programming with 802.15.4 and ZigBee: Laying the groundwork*, v4.4 ed. Libelium Comunicaciones Distribuidas S.L., sep 2013.