Reliable Data Communication System

S.H. Nijhuis
M. Zwalua

An implementation of the wireless communication system for the Life Counter appliance

Reliable Data Communication System

by

S.H. Nijhuis
M. Zwalua

to obtain the degree of Bachelor of Science

at the Delft University of Technology,

Project duration:	November 7, 2016 – February 2, 2017		
Student numbers:	S.H. Nijhuis		4157478
	M. Zwalua		4137671
Thesis committee:	L.C.N. de Vreede	Chair	TU Delft
	J. Hoekstra	Supervisor	TU Delft
	M.L. Gravemaker	Proposer	Life Counter
	I.E. Lager	Member	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Abstract

This report will present an implementation of a reliable data communications system, designed to transmit data from several 'sensor' nodes to one 'central' node and to display this data on a central display. Furthermore an uninterruptable power supply (UPS) was designed an implemented to keep the system running for approximately 9 hours.

The design was implemented with high-level embedded systems, such as the Raspberry Pi. On the Raspberry Pi an custom application protocol was developed to guarantee the transmission. For this transmission XBee Pro S2C modules were used. The transferred data is written to a SQL database.

The UPS and the XBee modules were tested individually on functionality. The UPS works as it should. The output voltage is stable when the battery voltage differs. The ZigBee network is able to transmit reliable, encrypted data over distances up to 550 meter. Furthermore, when this range not suffices, other ZigBee modules can be added to the network to relay messages.

S.H. Nijhuis
M. Zwalua
Delft, January 2017

Contents

List of Figures

List of Tables

1 Introduction

1.1. Background

According to the international convention of safety at sea [7], it's necessary to establish some kind of security to identify everyone on the ship. This convention is implemented in the "REGULATION (EC) No 725/2004 on enhancing ship and port facility security" [41]. When a ship is somewhere at the middle of the ocean, keeping track of who are exactly on board is not an issue. But when a vessel has to come to a dry dock for maintenance, this becomes a problem. Contractors, ship crew and shipyard personnel enter and leave the ship multiple times per day. To keep track of these persons, a system with paper cards is used. The problem with this system is that personnel tends to forget to move their card when they enter or exit the vessel, especially in case of an emergency, when the system is most needed. This results in a highly unreliable system.

1.2. Problem definition

To solve the problem described in section 1.1, Menno Gravemaker proposed a project assignment, as seen in appendix A. Summarized, the aim of the project is to:

"*design a detection system based on wireless detection and categorization of people, which communicates to a central place where it gives the safety officer of a ship a direct and reliable overview of who is on board the vessel at all times.*" [21].

The assignment is dividable into two parts: the recognition system and the processing and communication of multiple entrances on one boat. This report will only illustrate the latter. The RFID reader system will be reported by K.P. van der Mark en A. Lengyel [21].

An important factor is the reliability of the system; it has to work or it has to sense it doesn't and correct the error, or inform a supervisor.

1.3. Project goal

Proposed in this report is a design of a communication system that transmits the enter/exit information to a central location and displays this on a central screen. The system's 'sensor nodes' will be placed at the gangways leading to the entrances of the vessel. The *central node* will be located in a range of 100 meters from the *sensor nodes*. The project assignment is given in appendix A.

1.4. State-of-the-art analysis

Since the beginning of mankind reliable data communication was of great importance. Communication within villages made sure crops were harvest, goods were traded and enemies were beaten. Nowadays, a world without reliable communication is hard to imagine. From the cellphones in our pockets to the letter we bring to a mailbox. It all relies on reliable data communication. The wide range of applications also brings a wide range of possible solutions to transfer data.

Table 1.1: Communication options

Signal types	**Direct line-of-sight**	**Noise affected**	**effort to install**
Light	yes	high	low
Radio waves	no	medium	low
Wired	no	low	high

This isn't the first time a reliable data communication system was needed in combination with a RFID system. A system to follow cows within a dairy farm by using a RFID scanner was designed by G. Byrd [4]. This system uses XBee modules to enable communication between the RFID tag and a central node. Another system used the combination of RFID and XBee modules to keep track of food, medicine, general living goods, rescue equipment, vehicles, and on-site staff [48].

Infrared was also used to transfer data from one point to another. An article of F. R. Gfeller and U. Bapst describes a system for in house data communication (smaller than 50 m) using an infrared communication solution [14]. Also hybrid solutions are available. In the book "Study on Smart Home System Based on Internet of Things Technology"[22] a solution with ZigBee and infrared is mentioned.

These implementations have a lot of similarities compared to this project. All solutions require a communication system that is able to transmit data to a 'central place'. However, the communication part of these implementations were outdated. This means the network configuration is inconvenient and the used system boards are not available on the market anymore.

In our case, the reliability is the most important factor. This means that some extra software - such as a persistent[1] data cache - and hardware - such as an uninterruptable power supply (UPS) - have to be developed to guarantee a successful data transmission.

1.5. Thesis outline

This thesis will propose an implementation that is able to reliably transmit data from several sensor nodes to a single central node. The specific system requirements will be further clarified in chapter 2. Then, a global overview of the system will be given in chapter 3. The system is divided into four submodules. These submodules - the embedded system, the wireless system, the backup-up power system, and the software modules - will be discussed in chapters 4, 5, 7, and 6, respectively. After these submodules, chapter 3 will be devoted to the system overview.

Chapter 8 presents several tests, conducted to validate the function of the system. It's results will be discussed in sections 8.1.3 and 8.1.3.

This thesis will end with a report on the functionality of the developed system and a recommendation for future improvements. This report can be found in chapter 10.

[1] This means the data stored is not lost when suddenly a power outage occurs and the system has to reboot

2

Programme of requirements

The requirements are ordered according the MoSCoW method. The MoSCoW method defines four different levels of importantness of requirements: Must haves, Should haves, Could haves and Won't haves:

- Must haves
  - Requirements that form the essential base functionality.
- Should haves
  - Requirements that are also belong to the 'base functionality', but the system will be able to function, even if they are not implemented in the project.
- Could haves
  - Requirements that expand the functionality. These will be implemented depending on time and various other constraints, after implementing the 'must haves' and the 'should haves'
- Won't haves
  - Future requirements/functionality that will not be implemented in this project due to time or other constraints. These will not be part of this report.

2.1. General programme of requirements

The main assignment is designing a system that *must* reliably transmit information from a sensor node, located at a gangway, to central location, over a distance of 150 meters, with multiple data sources. A number of technical requirements are distinguished, separated into general and transmission requirements. The programme of requirements include vital functions of the system, except network- and transmission specifications. The transmission requirements include all fundamental design concerns in communication between the sensor nodes, such as the network topology and transmission error handling.

The system *must* display the number of people on-board. The data that has to be transmitted, *must* be read from a data input from the RFID sensor system.

To store data reliably, a database has to be maintained. To prevent errors from happening, the system *should* check and verify entries in database before updating. If a transmission is temporary not possible, the system *should* temporary (persistent) cache data.

Since RFID and wireless networks interfere when operating at the same frequency [6], no global ISM band as defined in the ITU Radio Regulations [43] is available for the RFID frequency, the system *should* track and report it's location. Most RFID readers have a setting in which region the reader is active, and tune the RFID

frequency accordingly.

In case of a power outage or unstable grid connection at the shipyard, the system *must* keep functioning in case of a power outage for at least 8 hours. Ideally, the system *should* keep function in case of a power outage for at least 24 hours. In order to achieve this with a battery of reasonable size and weight, the total system *should* consume less than 10 W of power. This is divided into a power consumption of 7 W for the RFID reader system and 3 W for the communication system.

It is possible that the system has to function outside. Therefore, the system *should* be rain and windproof. For easy use, the system *could* have easy setting buttons. If an error that cannot be solved automatically occurs, the system *should* be able to be rebooted, and therefore have a reset button.

For the supervisor it would be important to know the system's performance. Therefore, the system *could* display the number of connected ports on the central display and *could* display error messages on the central display. It also *could* have error LED's.

Summarized, the communication subsystem:

1. *must* reliably transmit data from a sensor node, located at the gangway to a central location
2. *must* display the number of people on-board
3. *must* read a data input from the RFID reader system
4. *must* keep functioning in case of a power outage for at least 8 hours
5. *should* consume less than 3 W of power
6. *should* keep function in case of a power outage for at least 24 hours
7. *should* be rain and windproof.
8. *should* temporarily (persistent) cache data
9. *should* track and report its location
10. *should* check and verify entries in database before updating
11. *should* have a reset button
12. *could* display the number of connected ports on the central display
13. *could* display error messages on the central display
14. *could* have error LED's.
15. *could* have easy setting buttons

2.2. Transmission requirements

In order to develop a reliable communication system, the system also *must* distribute time among the network nodes. This is, because otherwise data corruption or collisions could occur. If a transmission delay occurs, only the newest data has to be preserved.

Since a direct line-of-sight (LOS) is not guaranteed, infrared is not a suitable solution, according to table 1.1. Also, a wired connection is not a viable solution. It reduces the portability and takes a lot of time to set up (it has be installed carefully, so that personnel is not able to fall or break the cable). This leaves the wireless solution for the reliable data communication.

For transmitting data, a specific frequency band has to be used. Several options are available, the industrial, scientific, and medical (ISM) frequency band, the 863 MHz to 870 MHz band and any other (licensed) frequency band.

The so-called industrial, scientific and medical (ISM) bandwidths are defined in footnotes 5.138, 5.150, and 5.280 in the International Telecommunication Union (ITU) Radio Regulations [43]. These frequency bands are reserved for low-power communications systems, such as Bluetooth phones and wireless internet (Wi-Fi).

Besides the various ISM frequency bands, in Europe another frequency band is available for licence-free usage [9]. This frequency band is between 863 MHz and 870 MHz, and comes with limitations, regarding duty cycle, maximal effective radiated power (ERP) and application.

Almost every public, license-free transmission module uses one of these frequency bands.

If an emergency occurs, a lot of people will try to exit the vessel at the same time. Since a gangway is 0.5 to 3 meters wide, people can exit the vessel in rows of three people next to each other. In order to not create a large transmission lag, caused by a lack of transmission speed of the transmission system, it is vital that the system is able to transmit at least 5 actions per port per second. Since the system *must* support at least two sensor nodes, the system *must* be able to transmit a minimum of 10 actions per second, and *should* be able to transmit 20 actions per seconds.

For a reliable system, it has to recognize faulty modules. If a sensor node crashes, the central system needs to be informed about this and report it to a supervisor. Therefore, the central system *must* frequently contact all the active sensor nodes, to detect faulty modules.

Encryption can be of importance. If a good encryption scheme is implemented, the origin as well as the data itself, can be trusted. The data can be authenticated. Therefore, some form of encryption *should* be implemented.

Together with the already given requirements in the assignment (see appendix A), the communication subsystem:

1. *must* support at least two sensor nodes.
2. *must* detect transmission errors.
3. *must* distribute time among the network nodes.
4. *must* be able to transmit a minimum of 10 actions per second.
5. *must* frequently contact all the active sensor nodes, to detect faulty modules.
6. *should* support a transmission distance of 150 meters.
7. *should* support communication encryption.
8. *should* be able to transmit 20 actions per seconds.
9. *could* support more than 2 sensor nodes.
10. *could* support multiple displays.
11. *could* be able to withstand heavy weather.
12. *could* track and report it's location.
13. *could* verify transmitted data.

3

System overview

In this chapter a broad overview of the total system is given. It also shows the relations between all the submodules. An in depth explanation of the four submodules is discussed in the chapters. The four submodules of the communications sytem are:

- the embedded system, chapter 4
- the wireless system, chapter 5
- the back-up power system, chapter 7
- the software, chapter 6

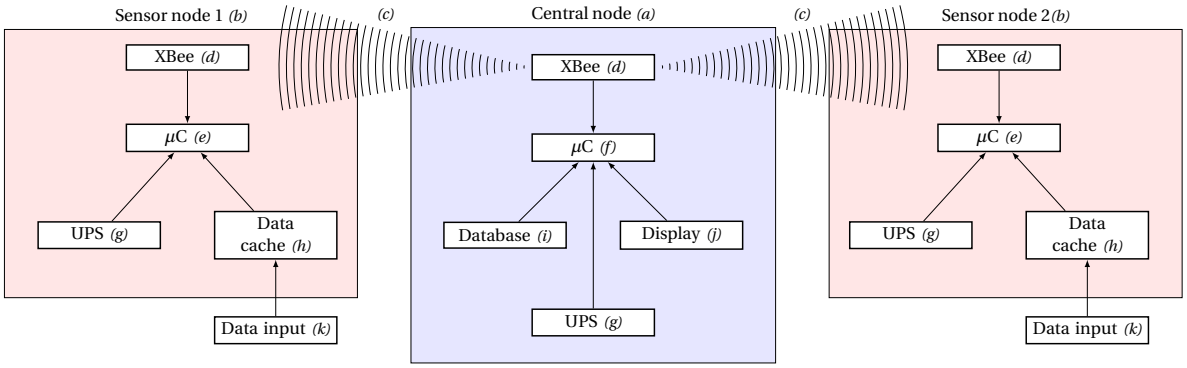
Figure 3.1 provides a block scheme overview of the system.

The communication subsystem is separated into two physical parts; a 'central' node *(a)* and a 'sensor' node *(b)*. The 'sensor nodes' consist of a micro controller *(e)*, a data cache *(h)*, an uninterruptable power supply (UPS) system *(g)*, and an XBee device *(d)*. The sensor nodes are located at the gangway of a vessel and collect information about people entering or exiting the ship *(k)*.

The micro controller is used to handle, process, and store information. This information is received from the RFID reader system and is stored in a persistent data cache, so no data will be lost if a power outage occurs, or the system has to restart because of an error. This precise implementation is described in chapter 6.5. In case a power outage or error occurs, the data will not be lost; after the power connection is restored, the system will continue transmitting its local cache.

A second precaution against power outages is the implementation of a UPS system. This is a small backup power system, connected to a sealed-lead-acid (SLA) battery, that will power the system for at least 8 hours if a power outage occurs (see chapter 7).

The XBee device enables the sensor node to transmit data to a central node. The central node is the location where all information is centrally stored and displayed to the supervisor. Therefore, the central node also features a micro controller *(f)*, an XBee device, a database *(i)* and a display *(j)*. Furthermore, the central node is protected against power outages, by storing all data persistently, and having a UPS system that is able to power the central node for at least 8 hours. If in an emergency the central node is not working, no data can be reported to the supervisor and the whole system is not very reliable.



(a)	Central node, where data from multiple sensor node is stored
(b)	Sensor node, which transmits information to a central node
(c)	Wireless data transmission
(d)	XBee PRO S2C wireless networking module
(e)	The micro controller (μC) on the sensor node, a C.H.I.P.
(f)	The micro controller (μC) on the central node, a Raspberry Pi 3.
(g)	Uninterruptable Power Supply
(h)	Persistent cache
(i)	The central database
(j)	Information display
(k)	Data input stream from RFID reader system

Figure 3.1: A brief system overview, consisting of a central node and two sensor nodes.

4

Embedded system

To process, store, and present the received data from the sensor nodes, located at the gangway of the vessel, a micro processor is needed. There are multiple ways of doing that. In this chapter the design choices and implementation of the microprocessor is explained. According to chapter 2, the central system must have:

- a monitor connection (HDMI, VGA or DVI)
- persistent storage
- enough processing power to process the data from the sensor node
- Universal asynchronous receiver/transmitter (UART) port [19]
- low energy comsumption (less then 3W)

Although there are many ways to connect a display, HDMI/VGA or DVI were the most suitable connections. These connections allow a high level display that is easy to read to be used.

Since the development time is limited, it's preferable to use an embedded system, or single board computer (SBC). These high-level micro processors, such as the Raspberry Pi 3B [44], are tiny computers working with a high-level operating system, such as '(Debian) Linux' or 'Windows 10 Embedded'. The system has all capabilities a modern computer has. Therefore, software written on it is not platform or chip specific, as it is on the various low-level micro processor such as the much simpler PIC16f628a [26].

Table 4.1 shows a comparison between five common micro processors with a UART or SPI connection. The UART or SPI connection is needed for the wireless communication system (further explained in chapter 5). The Arduino and PIC do not have a monitor connection. This makes them unusable for the central node. Furthermore, the C.H.I.P. is only fitted with a 'Radio Corporation of America' (RCA) monitor output. This connection is deprecated and only used for connecting old game consoles, video recorders (VCR's) and other old hardware to TV's and is therefore not considered for use in this system. However, a HDMI expansion board is available[1].

This leaves only the Raspberry Pi 3B and the Odroid C2. The Odroid C2 on the other hand, runs on a higher frequency, which results in a higher energy use. The higher frequency comes with extra processing power. Since no extra processing power is necessary, the Raspberry Pi 3B is left as the most suitable micro processor for the central node.

The requirements for the sensor nodes are almost the same as the central node, with the only difference being the monitor output, which the sensor node does not require. Because of this, the C.H.I.P. [30] is a suitable micro processor and in consultation with K.P. van der Mark and A. Lengyel, the C.H.I.P. is implemented in the sensor node. Since it runs Debian Linux, it's also well suited for fast development.

[1] The HDMI expansion board was not available at the time of this project due to being out of stock

Table 4.1: A list of five investigated micro processors.

Micro processors	Monitor connection	Persistent Storage	CPU	Energy use	Program. language	Approx. price
Raspberry Pi 3B[44]	HDMI, RCA	MicroSD	ARM A53	3 W[1]	Everything under Debian Linux, Windows 10 RT	€ 40
C.H.I.P.[30]	RCA	4GB NAND	ARM A8	2 W[30]	Everything under Debian Linux	€ 12
Odroid-C2[15]	HDMI	eMMC and MicroSD	ARM A53	5 W[15]	Everything under Ubuntu Linux, Android	€ 53
Arduino UNO[33]	No	1 kb EEPROM	Atmel ATmega328P	0.25 W[33]	Arduino C++	€ 12
Microchip PIC16f628a[26]	No	128 bytes EEPROM	PIC16f628a	0.2 mW[26]	C++	€ 1

4.1. Central node

The central node uses a Raspberry Pi 3B system. This embedded system is cheap, is equipped with an ready to use HDMI connector, and was already available when we started the project.

The Raspberry Pi 3B has one disadvantage. The on board power supply is slightly underdimensioned [13]. This was done to keep the costs of the device as low as possible [24]. As a result, under full load of the CPU, it's possible that the voltages drops enough to crash. If this happens during a write operation on the microSD card, the microSD card has a change to be corrupted [13]. If this happens, the system is not able to boot again and most of the data on the microSD card is unreadable. The risk of a low voltage crash gets bigger if a lot of energy is used for powering USB devices. During the development a few crashes happened because of this faulty power supply.

Although this means a Raspberry Pi 3B is not suitable for long-term operation, it is used for development purposes. The software, as explained in chapter 6, is compatible with any Linux or Windows operating system with a UART port available. This means the code is interchangeable with another embedded system designed for long-term operation.

One of these embedded systems is suitable for long term operation. This is the Odroid C2 [15]. It uses eMMC memory and has a better designed power supply. The eMMC memory is better designed, so corruption is less of a problem, compared to a regular microSD card [16]. The more thoroughly designed power supply makes the Odroid C2 more stable; the chances of crashing are smaller than the Raspberry Pi 3B. However, crashes are almost unavoidable, and are still possible to happen.

4.2. Sensor node

The sensor node requires less processing power than the central node. The C.H.I.P. is an excellent alternative to the Raspberry Pi 3B, equipped with less processing power. It is fitted with an internal persistent storage space, instead of an microSD card.

In contrast to the Raspberry Pi 3B, the C.H.I.P. does have a more suitable power supply. The C.H.I.P. uses on board flash memory with a controller instead of a microSD card without a controller. This makes the change of a corrupted memory a lot smaller, and makes the C.H.I.P. more suitable for long term operation.

5

Wireless Communication

Since a license for a global frequency band is not within possibilities due to money constraints, the system *must* transmit in an ISM (or likewise) band. Most off-the-shelf wireless transmission systems use one of the frequencies listed in table 5.1.

Table 5.1: Several ISM bands, within an acceptable frequency range

Frequency range		Availability
433.05 MHz	434.79 MHz	Region 1
863 MHz*	870 MHz	Europe
902 MHz	928 MHz	Region 2
2.4 GHz	2.5 GHz	Worldwide
5.725 GHz	5.875 GHz	Worldwide

Region 1: Europe, Africa, the Middle East (west of the Persian Gulf, including Iraq, the former Soviet Union and Mongolia).
Region 2: The US, Greenland and some eastern Pacific Islands.
*Not an ISM band, but defined by the Electronic Communications Committee (ECC).

One of the key features of the system is the communication between the central node and the sensor node. On this chapter the implementation and design choises are explained.

According to our requirements. The system needs to:

- cover a distance of maximal 150 meters
- support at least two sensor nodes
- be able to transmit at a minimum of 10 actions per second
- be low power

Described in table 5.2 are various standards for personal wireless communication, based on a comparative study of various wireless protocols.

Wi-Fi is used for many applications [20]. It offers high data rate and reasonable range. For our application, the range is not sufficient enough. Also, the ease of use in combination with micro controllers is not very high. Bluetooth offers more or the less the same specifications. Only the data rate is much lower.

The only suitable options are XBee modules and LoRA. Both offer low power, data transmission over a large distance. Also the micro controller software libraries support is high for these implementations. A lot

Table 5.2: Common used standard for wireless communication

Standard	**Bluetooth 1.2[2]**	**LoRa[23]**	**XBee Pro[8]**	**Wi-Fi[45]**
IEEE spec	802.15.1	802.15.4g	802.15.4	802.1 1a/b/g
Frequency band	2,4 GHz	861-870 Mhz	2,4 GHz	2,4 GHz
Max data rate	2 Mbit/s	0,3 kbit/s	250 kb/s	54 Mbs
Nominal range	100 m	22 Km	3,2 km	100 m

of libraries are available for rapid prototyping programming languages like Python and C(++). Because the development time is limited, this is an important feature.

Compared to XBee modules, LoRa has two drawbacks. The first one, is the frequency band. LoRa uses different frequency's for different parts of the world. One of the requirements is the use of world wide available frequency. The other drawback is the data rate. A minimum of 10 actions per second, would require for LoRa 4.48 Kbit/s. The data itself is 31 bytes, but the header for a LoRa message with this data is 25 bytes. This boils down 56 bytes or 448 bits per message. Although this is theoretically enough for the minimum requirement, it's not enough in practise. Since noise and multiple nodes lower the data rate to a level which is not acceptable.

This leaves XBee PRO modules as the only suitable option. It uses a world wide available frequency band, is able to cover the range and has a data rate that's high enough. Also, XBee is programmer friendly. It uses the UART or SPI port on a micro controller and has libraries available for Python [25], C [3] and more. As a result, XBee modules were selected in the wireless communication system. It's possible to connect XBee modules using UART. For development purposes, an UART to USB adapter was used. This implementation has a lower efficiency, due to losses caused by extra electronics needed by the USB to UART converter.

5.1. XBee background

Xbee PRO S2C (International) modules [8] are used for wireless communication. These XBee modules are widely used for wireless protocols by the industry. The XBee PRO S2C International has a maximal transmitting power of 63 mW, in order to make these modules usable all over the world.

The XBee modules do not require a gateway to the internet, can create networks with more than 65.000 devices and have a maximum transmission bandwidth of 250 kb/s. The modules have a maximal range of 3200 meter in a line-of-sight (LOS) situation. Furthermore, these modules support mesh networking, a technique where each node in the network is able to relay messages from and to various nodes on the network. The power usage of the XBee module is idle around 0.1 W and 0.3 W during transmission [8].

XBee offers a suitable form of encryption [37], a 128-bit Advanced Encryption Standard (AES) encryption. The payload of the message is encrypted by an key phrase distributed to every node on the network. AES is a symmetric encryption [34]. This means that the same key is used for encrypting and decrypting. On the complete system the same key is used for all the nodes. This is a secure implementation, because AES-128 bit is, at the time of writing, not compromised.

5.2. Firmware

XBee supports multiple firmwares. The most common ones are DigiMesh and ZigBee. Both offer a network in mesh topology [28]. The main difference between these firmwares is the type of nodes. The ZigBee protocol distinguishes three types of network nodes:

- Coordinator
- Router
- End Point

The coordinator-type device has a fixed 16 bit address (0x0000) and is always active. Only one coordinator can be active in a network. The router-type device is also unable to sleep, but has a variable address, which is assigned by the coordinator. The endpoint-type device is able to sleep and does not relay messages in a mesh configuration [17]. All of these devices are able to receive and send data.

The DigiMesh protocol does not distinguish different network node types; all nodes are routers. These routers are able to mesh and sleep [17]. Although sleep keeps the energy usage low, it negatively impacts the transmission speed and connection between nodes.

In a ZigBee network the coordinator has a known fixed address. Because of this, it's not required for the other nodes to know the hardware address of the coordinator. Instead, they transmit data to network address 0x0000. The advantage is that the software is identical for every system. The coordinator can be replaced without changing a single line of code.

Because of these advantages, ZigBee is chosen in favor of DigiMesh. The ZigBee protocol is used with only one coordinator - the central display - and routers - the sensor nodes.

5.3. ZigBee protocol

When joining a ZigBee network there are four scenarios (table 5.3) that can occur when a node joins a ZigBee network. If the joining node and the coordinator node have no encryption enabled, the joining will join the network and the encryption is not enabled. After the network is joined, it's possible to send and receive messages for both nodes. If the coordinator or the joining node has encryption enabled, there is no communication possible. If only the coordinator has encryption enabled, the node is able to join the network, but not able to communicate. In the case that both the coordinator and the joining node have encryption enabled, the communication is encrypted and both nodes are able to send and receive.

Table 5.3: ZigBee scenarios when joining network

Joining node encryption enabled	Coordinator node encryption enabled	Scenario
No	No	Joining node will join the network and messages are send unencrypted
No	Yes	Joining node will join the network, but will not be able to communicate
Yes	No	Joining node will not be able to join the network.
Yes	Yes	Joining node will join the network and messages are send encrypted

5.4. OSI Model

The OSI model provides an overview of the network attributes required to create a successful and useful network [27]. It give a good understanding of what happens on every layer of the network. The layers of the OSI model are:

1. **Physical network layer** This is the actual wave that is transmitted through a medium (copper, air, glass-fiber, etc.)
2. **Data Link layer** This layer handles error control on the physical layer.
3. **Network layer** This is the more higher level addressing scheme, providing ability to interconnect networks. Example: The IP protocol.
4. **Transport layer** Provides error and speed control on transmissions. Example: TCP.
5. **Session layer** Provides authentication and session management. Example: Database connection.
6. **Presentation layer** Provides representation of data. Example: JPG (pictures), AVI (video).

7. **Application layer** The actual data has been transmitted and has a semantic meaning to a specific program. Example: HTTP.

The ZigBee protocol provides an implementation of the upper six layers. It handles the physical, data, network, transport, session and presentation layer. The application layer is handled by the python ZigBee library and our own software.

5.5. Wireless Protocol

The XBee modules can be implemented using transparent mode (AT) [35] or using an Application Programmable Interface (API). When transparent mode is enabled, XBee modules act as a wireless serial port. Raw data is inputted at the serial connection of the sending module, and appears on the receiving module's serial port. The XBee module does not provide any feedback about the send or received data. An extra protocol has to be implemented for collision and error detection and handling.

The API allows the programmer to use a list of predefined functions. Amongst these functions are send and receive, but also several network control commands. Within this function error handling, collision detection and quality of service (QoS) is implemented.

Using API mode message delivery reports and network statistics are available. API mode is recommended when sending data to multiple destinations. From the API the commands listed in table 5.4 are used.

Table 5.4: ZigBee API commands used in the software

API Frame names	**API ID**	**Description**
ZigBee Transmit Request	0x10	Try to transmit data
ZigBee Transmit Status	0x8B	Receive a delivery report
ZigBee Explicit Rx Indicator	0x91	Data is received

It's also possible to use both of AT and API mode. In this case AT commands are allowed in the API packets. This can be used to configure certain settings or to receive remote parameters.

To configure the XBee modules it's not necessary to use the AT commands. It's also possible to use a GUI like XCTU. This program is developped by Digi International Inc. and is freely available [1].

Since the AT commands are not used in the use case, API mode is enabled without the possibility of AT commands. This reduces the risk of sending a possible AT command when this not meant to be. The implementation in our own protocol is described in section 6.3.

5.6. Prototype implementation

According to the requirements in chapter 2, the ZigBee protocol is the optimal solution. For the system three XBee modules (XBee Pro S2C with a nominal range of 3200 m and a datarate of 250 kb/s) were used. One for the central node (as depicted in figure 5.1a) and two for the sensors nodes (as depicted in figure 5.1b). The XBee modules were connected by UART. The XBee for the central node was provided with the ZigBee coordinator firmware. The two sensors nodes received the ZigBee router firmware.

The XBee were configured in API mode without escaping to AT mode. The PAN id was set to 1433, Encryption Enabled was set to 1 and the Encryption Key was set to "password". The other settings were not altered and left at their default.

The packet that is send from the sensor node to the central node is 48 bytes. 17 bytes are needed for the

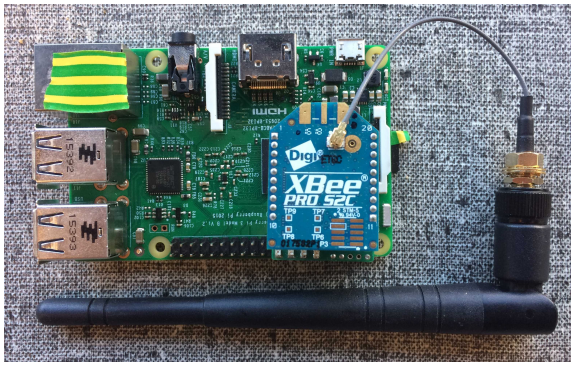
[1] XCTU can be found on `https://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu`

header and the CRC and 31 bytes are needed for the data to send. This results in a maximal packetrate per second of 651 packet/s (as shown in equation 5.1).

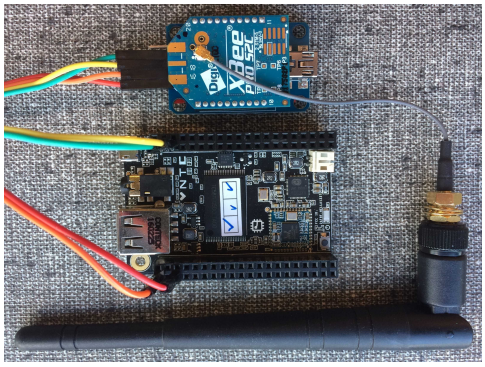
$$\frac{bitrate}{packetsize} = \frac{250\ kb/s}{48 * 8} \approx 651\ packets\ per\ second \tag{5.1}$$

This is more than enough to meet minimum actions per second requirement as seen in chapter 2.

A range test was conducted to see if the range requirement was met. The test results of this experiment can be found in chapter 8.



(a) The Raspberry Pi 3B connected to a XBee module



(b) The C.H.I.P. connected to a XBee module

6

Software

According to chapter 2, information needs to be transmitted from a sensor node, located at the gangway of a vessel, to a central node. No suitable low-power, low-bandwidth application protocol is available, since every generic protocol supports a lot of functions that are not used in this connection. Upon that, several special functions of the software have to be included - such as time distribution - that cannot be found in standard solutions. To be absolutely sure about the status of transmission and the validity of data, a custom software suite has been implemented on the micro controllers (μC), to take control of the storage and transmission of data.

This is done by three character 'commands' that can be transmitted, further explained in section 6.3. Every piece of data is acknowledged by the other party. Since the XBee module provides a Physical, Data link and Network layer (see section 5.4) implementations and the ZigBee protocol adds a reliable transmission protocol to the XBee network (see section 5.5) only a self-designed application protocol is needed.

Since reliability is an important issue in our system, the software has to be 'error aware'. This means that software has to be able to identify, and correct or report errors. In case of a total failure, the system is restarted and the ZigBee network is rebuilt. This all happens without losing data.

It is possible for data not to be transmitted chronological. To check which data is newer, time information has to be included with every piece of data. Therefore, the time needs to be synchronized, with an accuracy of several seconds. Both the Raspberry Pi 3B and the C.H.I.P. do not have a Real Time Clock (RTC) build in. This means, that whenever the power is removed, the time information is either lagging or unknown.

To indentify all the active nodes on the system (and therefore all the active gangway RFID sensor systems), the system needed a discover function. The ZigBee Node Identifier function (see chapter 5.5) does not provide enough reliability. This function only returns the XBee modules that have joined the network - it returns which nodes have an active input voltage. Therefore an alternative 'discover' function is implemented. This function will be handled by the custom application protocol presented in section 6.3, and will return the nodes that have joined the network and have the right software running.

According to chapter 2, the software has to be able to:

- Read data from the RFID sensor system
- Display the total number on-board
- Support multiple sensor nodes
- Save data with a persistent cache

The data from the RFID sensor system is read from a database file. This is done every 10 seconds. When

the data is send to the central node and a receive acknowledgement is received, the RFID data is removed from the database.

6.1. Language

There are a lot of programming languages available to use. When taking into account standard libraries, only C(#/+) and Python remained. Python was the best option, due to its extensive support for both the XBee library and many others, and its low-density and readable syntax.

Since both embedded systems work on a high-level Linux shell (see sections 4) it is possible to use a Github repository, for quick deploy and recovery of the source code. The code is fully exchangeable with any Linux or Windows computer.

The Python software is divided into four files.

- **main.py**
  The main source code, which collects all the libraries and starts the software
- **modules**
  - **base.py**
    The base source files, implementing the application protocol and the time distribution, discussed in section 6.3
  - **error.py**
    The error source files, implementing the error handling, covered in section 6.4
  - **database.py**
    The database source files, implementing the database handling, shown in section 6.5

6.2. Data transmission protocols

When using a computer to browse the internet, several protocols work together to show a good and valid website. Two important protocols are the transmission and application protocols, the upper two levels in the OSI model, as discussed in chapter 5.4. The transmission protocol takes care of how fast data is transmitted, how it is split into packets, how to check the contents of the packet and how to reconstruct the original data from the transmitted packets. This protocol can do this for every application on a system that is needed by the system.

The application protocol however, is part of the program that uses it. It defines the link between the web server and the browser. The application protocol is indeed a definition of language between a web server application and a client application.

6.3. Application Protocol

To make sure the data is transferred in a reliable way, the application protocol has to be able to:

- Transmit and verify data
- Transmit acknowledgements
- Distribute time over the network
- Identify active and faulty nodes
- Perform error handling

For these functions five possible commands are implemented (as seen in table 6.1). Basically, there are

Table 6.1: The five possible commands in the application protocol

Application Command	**Description**	**Transmitted Parameters**
'ACT' (ACTion)	Transmission of new data	• Node ID • Message ID • User Group • User ID • Direction • time stamp
'ACA' (ACtion Acknowledge)	Acknowledge an *'ACT'* packet	• Message ID • New time stamp
'DIS' (DIScover)	Discover all active sensors in network	• New time stamp
'DIA' (DIscover Acknowledge)	Acknowledge an *'DIS'* packet	• Node ID
'ERR' (ERRor) Transfer *'ERR'*	Transfer *'ERR'* packet from the node	• Node ID • Error type

three commands. The *'ACT'* and the *'DIS'* commands are acknowledged, respectively with an *'ACA'* and a *'DIA'* command. The *'ERR'* command is not acknowledged; an error that can't be resolved occured on a sensor node and the supervisor should correct the problem.

The node ID is the unique identifier of the gangway used.

The message ID is the unique identifier of the action in the sensor node database. By transferring this, the central node can easily confirm the particular message by returning this number.

The user group and user ID form a unique identifier for categorizing the individual walking on the gangway. It can also be used for system verification. If an action occurs - for instance, someone enters the vessel - the database can be checked if that particular user ID and group are already in the vessel. With this checks faulty sensor nodes or 'secret' gangways can be found.

The direction, stored and transmitted as a number from -3 to $+3$, is a combination of the number of scanned tags ($[1-3]$) and the direction of movement.

The 'time stamp' is the time stamp from when the action occured. If the transmission is delayed, the original time stamp will still be transmitted.

The 'new time stamp', that will be transmitted with the *'ACA'* and *'DIS'* command, will be a time synchonizing time stamp from the central Raspberry Pi. If a sensor node receives such a time stamp, the system time will be corrected to this time stamp.

'Error types' are the various error types: basic, moderate, critical, and bad critical errors are distinguished.

6.3.1. Transmitting data

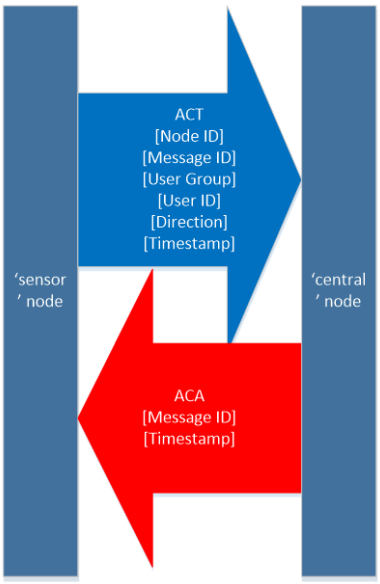


Figure 6.1: A graphical overview of the data transmission protocol

When there has been an action (someone walking through the gangway), the data is transmitted to the central node. The payload of the *'ACT'* command is:

```
ACT[Node ID],[Message ID],[User Group],[User ID],[Direction],[time stamp]
```

When an *'ACT'* command has been received, the data is parsed and stored in the database if the time stamp is greater than the time stamp already in the database. Otherwise it is possible for old information to be transmitted later. After completion, an *'ACA'* command is sent back. This package contains the following:

```
ACA[Message ID],[time stamp]
```

After receiving an *'ACA'* command, the sensor node system will change its system time to the time stamp that it received. This acknowledgement means also the data is received and processed correctly by the central node. Therefore all data with the particular 'message ID' can be deleted from the storage of the sensor node.

6.3.2. Discovering active nodes

Every 900 seconds (15 minutes) the central node tries to identify all sensor nodes, by dispatching a *'DIS'* command. This is done by the *'DIS'* command. The payload of the *'DIS'* command is:

```
DIS[time stamp]
```

When a *'DIS'* command has been received, the system time is updated and a *'DIA'* command is returned. The syntax is:

```
DIA[Node ID]
```

This will tell the central node that a port with particular node ID is active and running.

When a central node receives a *'DIA'* command, the node ID and the current time stamp will be written to a database, where all active nodes are listed.

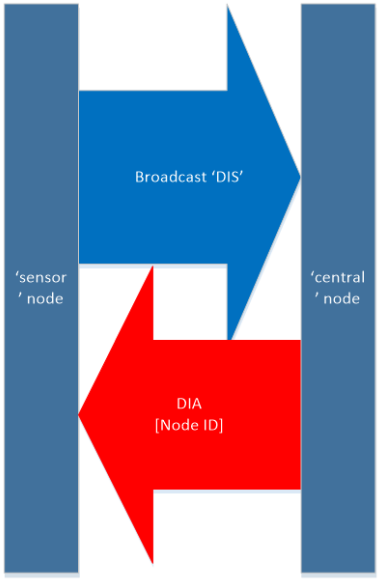


Figure 6.2: A graphical overview of the active node discovery protocol

6.4. Error handling

To design a reliable system, an error handler is required. If an error occurs, three steps are executed:

1. Save error to local log

2. Send error to central node
3. Activate error handler

The syntax of an *'ERR'* command is:

ERR[Node ID],[Error type]

There are three possible scenarios all depending on the severeness of the error. If the error is basic, the system will redo the action. If the error is moderate, the system will restart all the software. And if the error is critical, the complete system will reboot. In these last two situations, the central node will display a warning on the screen.

6.5. Database

To keep track of the number of people on-board the vessel, specific data needs to be stored. In order to store data structured and reliably, a database is used. In here the data, as explained in chapter 6.3, is stored: Message ID, User Group, User ID, Direction, and the time the action occured.

Two main options exist: NoSQL (Not Only Structured Data Language) and SQL (Structured Query Language). NoSQL is often used when the size, type, and format of data is not known. SQL databases are used when these characteristics of data is known. Since our data is very structured and predictable (we know how all the types and sizes of the variables), an SQL based database is used. Compared to a NoSQL database, it's easier to set up, search and insert data [18].

6.5.1. Database type

In order to function properly, the database software has to be:

- SQL based
- Lightweight
- Python compatible
- Reliable
- Open source

Since the database is used on an high-level embedded system (see chapter 4), the memory use should be limited. Also the processing power is a limiting factor.

An implementation that settles the mentioned requirements is SQLite [1]. It offers a lightweight and reliable SQL database, that is open source and easy to manage.

6.5.2. Database implementation: sensor node

In the node Pi there are two tables.

- actions
- service

When a person enters the vessel, the information of the RFID tag and the time stamp is written in the actions table. The database then generates a unique message ID. Stored are the 'user ID', 'user group', 'direction', 'time

[1] SQLite is freely available at `https://sqlite.org/`

stamp'. The user ID and user group are numbers to identify a person. The direction is an (signed) integer value, with six possible values: ±3, ±2, and ±1. This represents the direction of the person - *direction < 0* if the person is exiting and *direction > 0* if the person is entering the vessel - and the absolute value represents the scanned number of tags.

After the software has confirmed that the message has been received to the other side (the *'ACA'* command has been received), the entry is removed from the database. This is done to reduce the risk of an ever growing database.

Once every minute the RFID reader service stores a time stamp in the service table. The software checks this value every minute. If this time stamp is older than three minutes, an moderate error is raised an the RFID reader service is restarted.

6.5.3. Database implementation: central node

The central node, three tables are active:

- master
- node
- error

The 'master' table contains five columns. The 'user ID', 'user group', 'direction', 'tags scanned', 'first time active', and 'last time seen'. This information is received from the nodes. The user ID and user group are numbers to identify a person. The direction is, in contrast to the 'direction' value from the sensor node implementation, discussed in section 6.5.2, a value of ±1. The 'tags scanned' value represents the number of scanned tags. This information is split, for indexing purposes. Some simple maths has to be done when inserting the information into the database, but when acquiring information about the number of tags, almost no calculation has to be done. Furthermore, this data representation scheme is implemented to simplify a future implementation of a tag renewal system.

If new information is ready to be inserted into the table, the time last seen is compared to the time stamp included with the data. If the time stamp is newer than the time last seen, the new information is inserted into the database. If the time stamp is older than the time last seen, it is ignored. Due to this database layout, errors are corrected with the next boarding action.

The 'node' table contains all the active nodes in the system. Stored are the 'node address', 'node ID' and a 'time last seen'. The node address is the hardware address of the XBee module. The 'node ID' is the network ID of the sensor node. The 'last seen' field is updated whenever the node transmits an package. This can be a *'DIA'* command, an *'ERR'* command or an *'ACT'* command.

The 'error' table contains all the errors that occured on the connected sensor nodes and are transmitted using de *'ERR'* command (see section 6.4). Stored are the 'node ID', 'error type', and the 'time stamp'. The local errors from the central pi are not stored in this database. This is done to collect errors about the database.

6.6. User information display

In order to show who is on the vessel, a monitor is connected to the 'central pi'. This monitor displays the RFID tags and the user groups currently located on the vessel. Implemented for this project was a small 7 inch display, directly connected to the Raspberry Pi. Since its origin was unknown, no data sheet could be found with an indication of the power consumption. Estimated is that the LCD uses approximately 7 W. This is based on comparable LCD's since the original LCD didn't have a datasheet. This is comparable to the RFID reader system.

Information is displayed by a small web server (Apache[2]) embedded on the central node. On this web server runs a PHP script which reads information from the SQLite database. This page is automatically refreshed every 5 seconds.

[2]Apache is freely available at `https://www.apache.org/dyn/closer.cgi`

7

Uninterruptable Power Supply

In this chapter the uninterruptable power supply (UPS) of sensor node and the central node will be covered. System reliability is very important. Since this system is designed for emergency situations, it must keep functioning when a power outage occurs. Therefore an uninterruptable power supply (UPS) system is implemented. If the central node does not function; the number of people on board can't be displayed. Therefore, the central node needs to be online for a longer period than it's sensor nodes.

7.1. System design

The first option is directly connecting a battery to the system. This has the advantage that no extra circuitry is necessary. However, when a 12 V battery is connected directly to a 12 V bus, the battery is unable to charge, since this takes place at a higher voltage. Furthermore, this implementation has the disadvantage that there is no current limiting; if the battery is empty when connecting an external power source, the maximal charging current of the battery will be very high. This causes dangerous situations (e.g. the battery could explode, or the high current could melt wires and start a fire). Therefore, this implementation requires at least a current limiter, voltage regulator, and a fuse. Since an off-the-shelf UPS solution is exactly that, the decision is made to implement this.

Various basic UPS devices are available, such as the UPS Pico [32], OpenUPS [29], and the PicoUPS 120 [5]. An overview of this systems is available in table 7.1.

Table 7.1: An overview of three UPS systems, the UPS Pico, the OpenUPS, and the PicoUPS 120

	UPS Pico [32]	**OpenUPS [29]**	**PicoUPS 120 [5]**
Input voltage	5 V	6-30 V	6-18 V
Output voltage	5 V	6-24 V	12
Output current	2.5 A (3A peak)	6 A (10 A peak)	6 (10 with active cooling)
Charging current	750mA	3 A	1.2 A
Battery type	Li-Po	Li-Ion, Li-Po, LIFEPO4, Lead Acid	(sealed-)lead-acid
Battery capacity	3 Ah	configurable, up to 6 batteries	4-60 Ah
approximate price	€ 30	€ 120	€ 40

According to chapter 2, the UPS system needs to be able to:

- power a sensor node system for at least 8 hours
- deliver 10 W to the system
- have a 5 V and 12 V output

- switch from AC input to battery without shutting down the system
- charge a battery while connected to AC input

For the central node. The requirements are almost the same. It used 10 W, has two switch from AC to battery without shutting down and charge a battery while connected to the AC input. The only difference is that it powers a LCD instead of a RFID scanner. Therefor it lacks the need of a 12 volt output.

The UPS Pico system has a maximum battery size of 3 Ah. According to equation 7.1 the UPS Pico can't power the sytem for 8 hours.

$$t_{standalone} \approx \frac{V_{battery} * C-rate_{battery}}{P_{system}} = \frac{5V * 3Ah}{10W} = 1.5hours \tag{7.1}$$

The OpenUPS system has very extensive control options. It's able to charge different kind of batteries, balance different cells and measure input/output current. These functions are not necessary for the system.

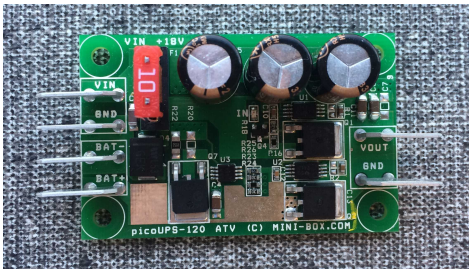


Figure 7.1: The Pico UPS 120

Therefor the PicoUPS 120 (shown in figure: 7.1) was implemented [5]. It can charge a sealed-lead-acid (SLA) battery [1] with a maximum current of 2.4 A - with the possibility of an alternative setting of 0.6 A and 1.2 A - and at 13.5 V, and can provide a node system with a maximum of 6 A at 12 V without active cooling. With active cooling the PicoUPS is able to deliver 10A at 12 V to the output. The PicoUPS is able to charge a (S)LA battery with a capacity from 4 Ah up to 60 Ah. Equation 7.2 shows that the maximal time a PicoUPS can power a node is

$$t_{standalone} \approx \frac{V_{battery} * C-rate_{battery}}{P_{system}} = \frac{12V * 60Ah}{10W} = 72hours \tag{7.2}$$

Figure 7.2 gives an overview of the electrical connections between the various components embedded in a sensor system. A system wide overview is available in figure: 3.1.

The function of the sensors in figure 7.2 - the Global Positioning Sensor (GPS), Passive Infrared Sensor (PIR), and the RFID reader - are not further explained in this report. These devices are covered in the report of the team developing the other part of the system [21]. The XBee module, the GPS sensor, and the PIR sensor are powered directly via the microcontroller (μC).

The PicoUPS is able to charge its battery when the input voltage is at least 13.5 V. According to the manual [5], the ideal input voltage of the PicoUPS is 16 V. An external input voltage of 12 VDC is applied to the boost converter. This boost converter raises the voltage level to 16 V, in order to charge the battery.

The output voltage of the PicoUPS is not regulated. If an input voltage is applied, the output voltage will match the input voltage minus 0.5 V. When the voltage input is removed, the voltage will drop to the battery voltage.

No 5 V output is available on the PicoUPS. Since the embedded system requires a stable voltage of 5 V, the voltage has to be brought down to 5V. The voltage regulator has to deliver a maximum of 0.6 A [2]. This can be done by:

- Linear Voltage Regulator
- Switching Voltage Regulator

[1] Regular lead-acid batteries can also be charged with the PicoUPS 120
[2] 3 W / 5 V = 0.6 A

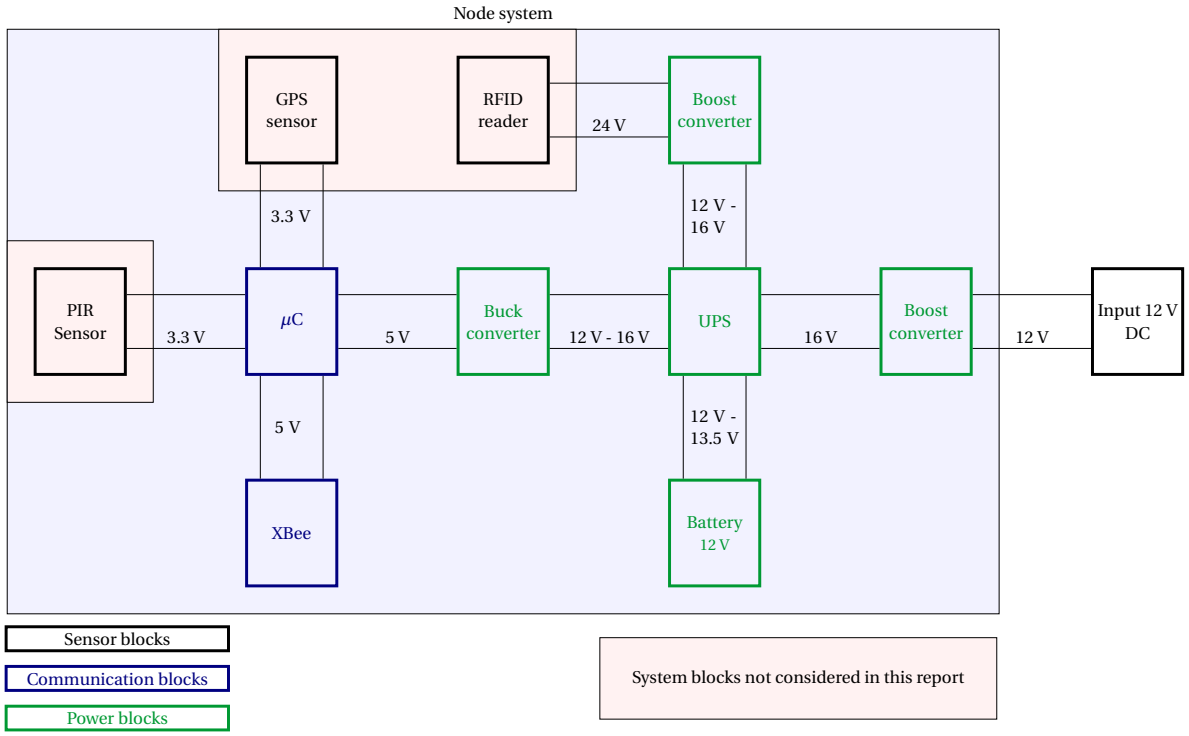
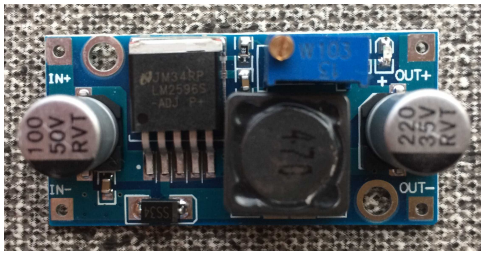


Figure 7.2: The implemented electrical scheme of a sensor node. An input voltage of 12 VDC is applied to the system. The central node lacks the boost converter; the central node does not require a RFID reader.

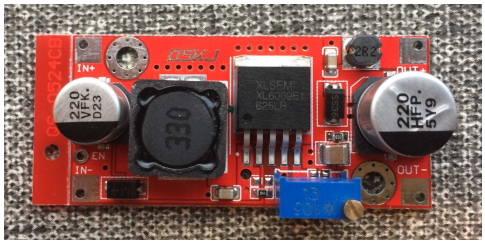
One of the possible linear voltage regulators is the LM7805 [40]. It's able to deliver 1.5 A at 5 V as long as the input voltage is at least 7.5 V. Although it's very cheap (a LM7805 can be bought for as little as € 0.12), it isn't very efficient [12]. The most optimal efficiency is shown in equation 7.3

$$\begin{aligned} V_{dif} &= V_{in} - V_{out} = 12 - 5 = 7\,V \\ P_{loss} &= V_{dif} * I_{in} = 7 * 0.6 = 4.2\,W \\ \eta &= \frac{P_{out}}{P_{in}} = \frac{7 - 4.2}{0.6 * 12} \approx 39\% \end{aligned} \tag{7.3}$$

Another option is a switching power supply like the Texas Instruments LM2596. This step-down converter (or buck converter [11]), depicted in 7.3b is able to reduce an input voltage between 1.2 V and 37 V to a stabilized output voltage, minimal 4% lower than the input voltage. The maximum output current is 3 A. Although this chip is more expensive (€ 0.81), it's also a lot more efficient. It's able to convert at least 80 % of the power according to the manual [39]. Since there was only a limited amount of power available, the LM2596 is used instead of the LM7805.



(a) The LM2596S buck converter



(b) The XL6009 boost converter

The RFID scanner requires an input voltage of 24 V. This can be done by using a step-up converter [10]. A possible step-converter is the XLSEMI XL6009 (depicted in figure 7.3a). This step-up converter is able to

raise its input voltage of more than 5 V to an output voltage at least 1.25 V higher than it's input. The maximal output current is 4 A, according to the datasheet [47]. The efficiency is approximately 90 %. Because the XL6009 was able to meet all the necessary specification, it was used in the system.

7.2. Internal battery

The PicoUPS has a 13.5 V (S)LA battery charger. It is not equipped with a battery management system (BMS) or DoD control. (S)LA batteries can handle up to 100% depth of discharge (DoD) [46]. This, however, strongly affects the cycle life, with about 4000 cycles at 20% DoD, and 150 cycles at 100% DoD.

Since the internal battery is used as an back up power source, not many (deep) cycles will be made. So this should not be a problem for the system.

The C-rate (or nominal capacity) needed by the battery can be estimated using equation 7.5.

$$C-rate_{battery} \approx \frac{P_{system} * t_{standalone}}{DoD} Wh \tag{7.4}$$

$$C-rate_{battery} \approx \frac{P_{system} * t_{standalone}}{V_{battery} * DoD} Ah \tag{7.5}$$

This leads to capacity required displayed in table 7.2.

Table 7.2: The estimation based on equation 7.5 of minimal required battery capacity for a off grid run time of 8, 24 hours, and one week, rounded by the nearest integer number

	8h operation		24h operation		1 week operation	
DoD	$E_{battery}$ (Wh)	$E_{battery}$ (Ah)	$E_{battery}$ (Wh)	$E_{battery}$ (Ah)	$E_{battery}$ (Wh)	$E_{battery}$ (Ah)
20%	340	33	1200	100	8400	700
40%	200	17	600	50	4200	350
60%	134	11	400	33	2800	233
80%	100	8	300	25	2100	175
100%	80	7	240	20	1680	140

P_{system} = 10 W, $V_{battery}$ = 12 V

When using a (S)LA battery and take a DoD of 60% for calculated losses, a battery with a capacity of minimal 11 Ah is needed for 8 hour operation. For 24 h operation, the minimal estimated required capacity is 50 Ah.

7.3. External power supply

To charge the internal battery, a voltage between 14 and 18 V is needed. This can be accomplished by using a generic 230 VAC to 12 VDC voltage converters. According to Yuasa [49], a big manufacturer of 12 V batteries, the charging current should not exceed the capacity of the battery divided by 10.

Since an 11 Ah battery is necessary when powering the node for 8 hours, the charging current is limited to 1.2 A (according to equation 7.6, since the PicoUPS only allows for a current limiter of 0.6 A, 1.2 A, or 2.4 A).

$$I_{charge} = \frac{E_{battery}}{10} = 1.1\,A \approx 1.2\,A \tag{7.6}$$

Since the battery charges with a maximum of 1.2 A, the voltage supply has to be able to supply at least 26.5 W to the circuit, according to equation 7.7.

$$P_{powersupply} = P_{charge} + P_{system} = U_{charge} * I_{charge} + 10\,W = 13.5\,V * 1.2\,A + 10\,W = 26.2\,W \approx 26.5\,W. \tag{7.7}$$

This corresponds to a 12 V power supply that has be able to deliver at least 2.2 A to the system.

7.4. Prototype implementation

According to specifications, as stated in chapter 2, the communication subsystem has to be able to run a minimum of 8 hours and ideally 24 hours off grid. According to Vutetakis et al. [46], a (S)LA battery has a life span of about 500 cycles when being discharged to ≈ 70% DoD, the percentage the used battery will approximately be if the system is running for 8 hours on battery power. Since the PicoUPS does not have DoD control, the system - with an estimated power consumption of 10 W for both the central- and the sensor node - and a connected battery of 9 Ah is estimated to last approximately 9 hours until the battery is empty (100% DoD), according to equation 7.8.

$$t_{standalone} \approx \frac{V_{battery} * C - rate_{battery} * DoD}{P_{system}} = \frac{12\,V * 9\,Ah * 100\%}{10\,W} = 9.6\,hours \tag{7.8}$$

For the proof of concept, a Ultracell UL9-12 [42] 12 V 9 Ah battery is used. According to table 7.2 this is not nearly enough to provide backup power for a period of 24 hours (see section 8), but it demonstrates that the system is working and run for about 8 hours.

According to the data sheet of the battery, the battery lasts approximately 10 hours with a constant discharge current of 0.83 A (approximately 10 W). Therefore, approximation 7.8 is a valid approximation for this battery.

8

Prototype implementation verification

Three experiments are conducted, to prove the (wireless) functioning of the prototype: Since transmission distance is very important, a range test was conducted. Measured are the range:

- between 2 XBee modules
- between 2 XBee modules, with a third module in the middle, functioning as a message relay
- between 2 XBee modules, with a metal building with a height of 25 meter between the modules

Furthermore, a short experiment was done, to verify the uninterruptable power supply (UPS) functions, to verify the implementation of the components used.

8.1. XBee Range test

The first test, the range between 2 XBee modules is tested. After this range was determined, the ZigBee protocol was tested by separating 2 XBee modules further than the range determined in the first test - with a third XBee module in the middle.

8.1.1. Experiment setup

XCTU, the standard software suite for XBee modules, features a built-in range test module (see appendix B.3 for an overview of the user interface). The test starts with the two modules placed directly next to each other. One module is separated further away and stays one minute at each 50 meters - measured with runkeeper[1] - extra distance, to get an overview of the mean signal strength and packet loss. The experiment is conducted with the antenna's of both modules placed at 1.5 meters from the ground, pointing in the upwards direction.

The second experiment is conducted to prove that the ZigBee protocol lets devices in the network relay traffic, so nodes can communicate even when they are not directly in range of each other. Since the range of two XBee devices was determined in the first range experiment, the XBee modules were placed separated beyond their range, with a third relay module in between. Measured are the ability to communicate between the outer two nodes and the network connections between the nodes. After a while the relay node was removed from the network, and the differences in ability and network connections were measured. Expected is that the two outer nodes are able to communicate, as long as the third relay node was enabled and placed in between these two nodes. The experiment is conducted with the antenna's of both modules placed at 1.5 meters from the ground, pointing in the upwards direction.

[1] Runkeeper is an app to measure walking or running distance with GPS

The third experiment is conducted with a big, metal building[2] with a height of 25 meters between the XBee modules. The modules are placed approximately 150 meters apart - measured with Google Maps. Expected is that the RSSI will be lower, but the percentage of successful transmitted packets will be approximately equal, compared to the first part of the experiment. The experiment is conducted with the antenna's of both modules placed at 1.5 meters from the ground, pointing in the upwards direction.

8.1.2. Results

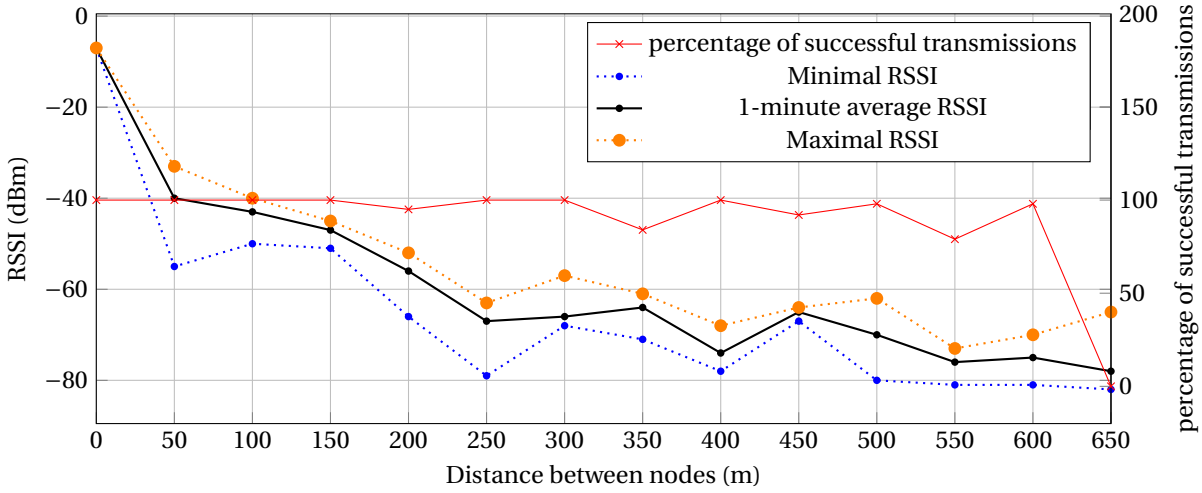


Figure 8.1: A figure of the maximal, 1-minute average, and minimal received signal strength indicator (RSSI) and the percentage of successful transmission with a varying transmission distance

The first experiment is conducted at the Mekelpark, the campus of the Delft University of Technology. One XBee module, connected to the XCTU range test application, was placed stationary, in front of the sports building at the south end of the campus. A second XBee module was placed directly near the first module, and moved 50 meters away from the first node. The mean RSSI and percentage of successful packets was measured at this distance for a period of one minute. This was repeated each 50 meters, until the two nodes lost their connection. Figure 8.1 shows the 1-minute average received signal strength indicator (RSSI) and the percentage of successful transmissions over a varying distance from 0 to 650 meters. The raw data is depicted in appendix C.2.

Table 8.1: The network connections in experiment 2, with and without a third 'relaying' node placed between the two outer nodes.

Address	**Role**	**Connections**	**Location**
Middle 'relay' node enabled			
0000	Coordinator	Router [7D9A]	Local, connected to XCTU
7D9A	Router	Router [86EA] Coordinator [0000]	Middle 'relay' node
86EA	Router	Router [7D9A]	Outer node, receiving data
Middle 'relay' node disabled			
0000	Coordinator	-	Local, connected to XCTU
86EA	Router	-	Outer node, not receiving data

Since the first test concludes that the maximum range of two XBee S2C PRO devices is approximately 670 meters (see section 8.1, two modules were separated 750 meters. A third module was placed in between the two modules. Table 8.1 shows the network situation with and the middle relay module enabled. Shown is that the nodes were able to communicate before removal of the third 'relay' module. With the middle 'relay' node

[2] The building is the 'Hoogspanningshal', located at the EEMCS building at the TU Delft

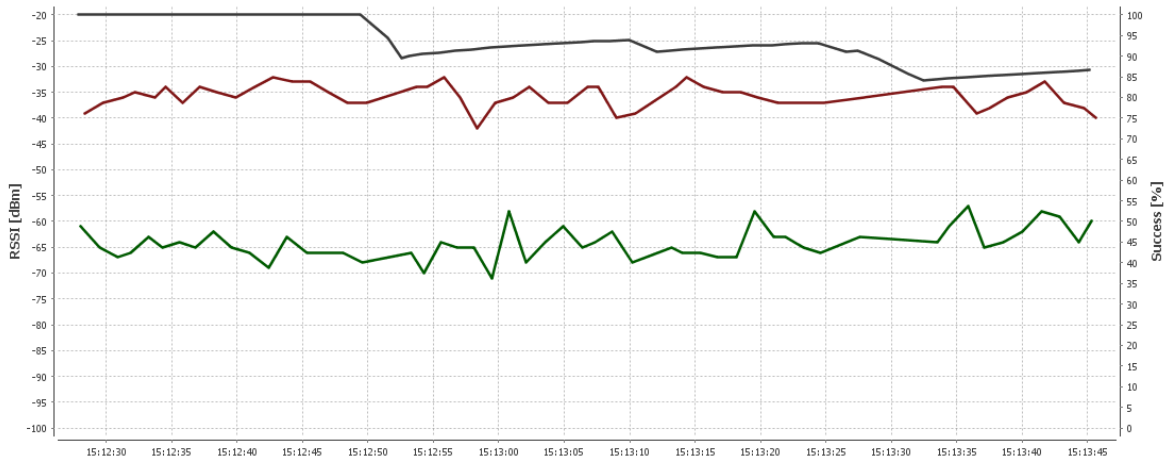


Figure 8.2: The local RSSI (the local XBee module, green), the remote RSSI (the remote XBee module, black) and the percentage successful transmissions when standing separated in a 45 degrees 'pizza slice' around the 'Hoogspanningshal'.

enabled, the outer two nodes had an active connection to the center node. The center node has a connection to both router nodes. After the relay module was removed, no connections were available.

The third experiment was conducted at the 'Hoogspanningshal'. This building models a metal vessel, that is in between two gangways. The setup is comparable to the setup used in the first experiment; one node was connected to the XCTU range test application and the other was placed behind the 'Hoogspanningshal', to measure the RSSI and the percentage of successful package transmissions. Since no data could be transferred through or over the 'Hoogspanningshal', the experiment was altered slightly. One XBee module was not moved, but instead of standing at the other side of the 'Hoogspanningshal', the second node was placed next to the building, in order to test if an optional third (relaying) XBee module could be used to transfer information around the vessel. This results are given in figure 8.2. Shown are the local RSSI (the RSSI on the local XBee module), the remote RSSI (RSSI on the remote XBee module, black) and the percentage successful transmissions when standing separated in a 90 degrees 'pizza slice'[3] around the 'Hoogspanningshal'.

8.1.3. Discussion

The first experiment - as depicted in figure 8.1 - concluded that when a distance of less than 550 no structural packet loss was observed and a stable connection could be maintained. If the modules are placed more than 550 meters apart, the connection is possible, but unstable and prone to errors or transmission defects, effectively uncontrollably reducing the reliability of the network. The measurements were conducted at the 'Mekelpark', where a lot of noise sources were active at the time of measurement. Clearly visible in the (not averaged) test results were buses, bikers and even people taking pictures. An example is given in figure C.2 in appendix C.2. However, these noise sources could also be active at a shipyard. Also, the greater the distance, the greater the deviation in the RSSI; this is probably due to reflections or other 2.4 GHz noise sources, such as wireless internet routers, Bluetooth, and many other devices. However, these noise sources are most likely also found on a shipyard. The experiment concludes that the range of two individual XBee modules is large enough.

The second experiment was conducted to show that the ZigBee protocol is able to 'relay' information for the network. Conclusive evidence is given in table 8.1; before the third 'relay' node is removed from the network the two XBee modules are able to communicate. When the 'relay' node was removed, the XBee modules were not able to communicate anymore. This experiment presents conclusive evidence that the ZigBee protocol is able to reliably function as a middle, 'relaying' node.

A big metal obstruction between the nodes is a problem. The nodes were almost instantly not able to communicate while the building was in between the nodes. This experiment shows that it is not possible to

[3] see figure C.3 in appendix C.3 for a picture of the measurement locations

communicate from one side of the vessel to the other, if the vessel is approximately 25 meters or higher. However, the ZigBee protocol can be a solution. Shown in the second part of the experiment is that a third module can be placed at a strategic location, to relay messages from one side to another. If a third module is placed next to the vessel (in a 'pizza slice'[4] of approximately 90 degrees around the vessel), the communication will take place around the vessel, instead of through or over.

8.2. UPS function test

The second experiment is an UPS function test. This test was conducted, to verify the setup of the UPS system.

8.2.1. Experiment setup

The outline of this experiment is very simple.

1. Wire up the electrical system as shown in figure 8.3 with a fully charged battery connected.
2. Measure the voltages at the output of the step-up converter, the step-down converter, the battery, and the PicoUPS.
3. Disconnect the 16 V input voltage
4. Again, measure the voltages at the output of the step-up converter, the step-down converter, the battery, and the PicoUPS.

The voltage at the output of the PicoUPS is expected drop from 16 V to battery level (approximately 12.8 V), since the output of the PicoUPS is not regulated. The voltage at the output of the step-up converter is expected to remain unchanged, since this converter features a regulated output. The voltage at the input of the step-down converter is expected to show similar behaviour.

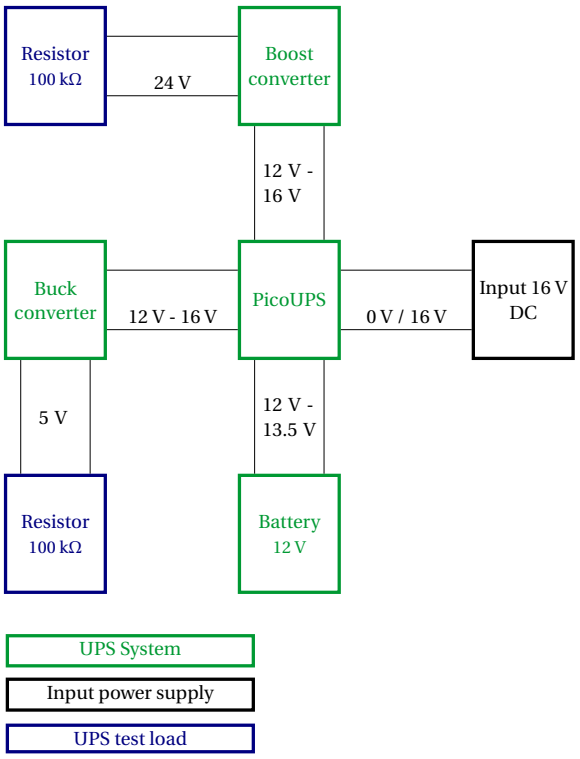


Figure 8.3: A schematic overview of the experiment setup.

8.2.2. Results

The setup as in figure 8.3 was build. The input voltage for the PicoUPS was generated by a power supply, set to 16 V. The output voltages of the PicoUPS, the step-up converter and the step-down converter were measured using a Tenma digital multimeter [38], with the power supply enabled, as well as disabled.

The experiment results are shown in table 8.2. Shown is a voltage drop on the PicoUPS (unregulated) output, and a stable voltage on the output of both the step-up and the step-down converter.

8.2.3. Discussion

This experiment proved the UPS system is working as it should; the output voltages are regulated, and therefore remain the same voltage, while the input can vary and switch from battery tot an external power source.

[4] see figure C.3 in appendix C.3 for a picture of the measurement locations

Table 8.2: Voltages of the step-up and step-down converter, as depicted in figure 7.2, with and without an input voltage of 16 V connected to the PicoUPS.

Device (output)	Voltage with input voltage PicoUPS (V)	Voltage without input voltage PicoUPS (V)
PicoUPS	16.1[1]	12.9[2]
step-up converter[3]	5.0	5.0
step-down converter[4]	24.1	24.1

[1]This voltage is equal to the measured power supply voltage.

[2]This voltage is equal to the measured the battery voltage.

[3]connected to the RFID Reader.

[4]connected to the μC.

9

Conclusion

The goal of the project is, according to section 1:

"*design a detection system based on wireless detection and categorization of people, which communicates to a central place where it gives the safety officer of a ship a direct and reliable overview of who is on board the vessel at all times.*" [21].

Section 2 proposes a number of requirements in various levels of 'importantness'. This chapter will discuss if, and which, the criteria are met.

The system is able to reliably transmit data, originating from a RFID reader, from a sensor node to a central node. This is done by using XBee modules with a sufficient range and dedicated, self-designed application protocol. The system remains stable within a range of 550 meter. The system supports multiple sensor nodes. Ones every 900 seconds the central node broadcasts a discover packet to all sensors nodes. The answers on this broadcast are written in a database to detect faulty sensor nodes.

The system shows the number of people on-board. This is done by using a local web-server embedded on the central node's Raspberry Pi. This web page also shows the active number and the current error status of nodes.

Both the sensor node and the central node are protected against outages due to a persistent memory storage. If, for example, an error occurs and the system must be rebooted, no boarding data is lost.

All sensor data is written to a database on the sensor node. If the transmission is confirmed by the central node, the entry of that sensor data is removed from the sensor node database. If the data from the sensor node isn't confirmed by the central node, the data is send again and a error is raised. This error is logged locally and (if transmission succeeds) on the central node. The data that is send from the XBee module is encrypted with 128 AES pre shared key encryption.

A secondary protection against outages is the implementation of an uninterruptable power supply (UPS) system was implemented to keep the system functioning when a power outage occurs. A schematic overview of this system is available in figure 7.2. A battery of 9 Ah was used. According to equation 7.8 battery can power a 10 W system for approximately 10 hours. This statement is supported by the datasheet of the manufacturer [42].

The maximum packet rate per second was calculated to be 651 (as seen in equation 5.1). Every packet equals an action in the system. This means that it's theoretical possible to have 651 actions per second, which is much higher then the required 10 actions per second.

The central node is not proven to consume less than 3 W. The maximum power use of a Raspberry Pi

is in the range of 3 W. This, however is under full load operation and several power consuming peripherals connected; a situation that will never occur on our system. The central node does not have an RFID reader system connected; its power budget is 10 W. The used display, did not specify what it's power usage is. This could be further investigated. The XBee module uses a maximum of 0.39 W, when transmitting data, and 0.1 W when receiving. Therefore, it is safe to conclude that the central node consumes approximately 3 W on average, but no conclusive evidence can be given. The C.H.I.P. uses approximately 2 W. Therefore, the sensor node is consuming less than 3 W. Peripherals, such as the GPS- and PIR sensor are not included in this calculation.

The system is not fitted with a reset button or status LED's . However, it is able to initiate a reboot when it senses an error. It also is able to track and report its location. This is reported in the thesis of A. Lengyel and K.P. van der Mark [21].

Since all *must haves* and *should haves* from chapter 2 are successfully included in the system, the project can be called a succes. The implementation of the requirements has led to a reliable and sustainable communication system.

10

Future recommendations

This report poses an excellent solution for the given assignment. A system that is able to reliably transport information from several sensor nodes to a single central node was developed and tested. Chapter 9 provides an overview of the functionality and security of the developed system.

There are, however, several opportunities to gain extra functionality. It is advisable to conduct more research on the internal battery type. According to Stan et al. [36] there are more suitable batteries in uninterruptable power supply (UPS) application. These batteries, however, require more sophisticated chargers and were not feasible to implement during this project.

The influence of weather on the XBee transmission range was not tested. Rain and possibly fog can have a great impact on transmission range. Therefore, an experiment should be conducted to quantitatively determine the influence of rainfall on the XBee transmission distance. Upon that, field tests at several shipyards should be conducted, in order to precisely determine the system's performance.

More comparing research has to be conducted to choose an optimal embedded system. The system hast to be able to It can be very expensive to create a grid connection at a shipyard. Therefore, an 12 VDC adapter is used, instead of a 230 VAC, so it is easy to exchange a 230 VAC to 12 V adapter with a large 12 V battery pack. The estimated capacity required for such a battery is - according to table 7.2 - 233 Ah (at 12 V) for a depth-of-discharge (DoD) of 60%. More research about the used battery type and size is recommended - a lead-acid battery can become quite heavy. An 12 V, 250 Ah battery weights approximately 73 kg [31]. Upon that, more losses and environmental constraints should probably be considered when sizing the battery.

Furthermore, a custom UPS system could be developed. This will probably be more efficient and extra functionality could be added; the step-up and step-down converters could be replaced by a more efficient solution, the internal battery could be saved by cutting power at (for example) 80% DoD, a solar power system could be added to the system as a third power source, and advanced battery measurements - such as coulomb-counting - could report the current state of the internal battery to its micro processor, or even to the central node.

The Raspberry Pi 3B is not an optimal solution for the central node, due to a badly designed internal power supply (see chapter 4.1). This embedded system should be replaced by another micro processor, with a lower power usage and features depending on the future implementations.

The display should be further investigated. Since the available display was an unknown, unbranded 7 inch display, the precise power usage is not known. Thorough testing has to conclude if the approximation made in chapter 6.6 are actually true.

Network safety can be implemented tuning the ZigBee parameters; the 'Node Join' parameter allows nodes to join a ZigBee network for a specific duration. A system, that involves pressing a button at the central

node could allow a sensor node to be 'paired' with another node, whilst denying access to other devices that are not paired with the network. This will prevent hackers from joining the network and send false data. This possibility is considered not necessary for this project, since pre-shared key encryption is already enabled on the ZigBee network. Attackers first need to know the secret password, before they can join and transmit data. Furthermore, at this moment no benefits could be obtained by transmitting false data. However, if for example attendance is checked using the entrance and exit times of people, it could be of value to further secure the network.

If a the number of scanned tags of a person is never equal to three, it is most likely that a tag is broken. This could be intercepted by an active monitoring system, that monitors both the age and errors of the tags.

A follow-up project can implement more functions to improve user friendliness. If a new set of sensor nodes and a central node have to be deployed, simple modifications have to be done by hand to install the system. These settings are the network settings[1] of the ZigBee devices. Also error LED's could be implemented on every node, to locally display its current status - red for a severe error, yellow for a slight issue and green for status OK. This will make the error report procedure more robust; if someone sees a red light, he can inform his supervisor.

A cloud connection was not implemented. An internet connection brings advanced security and performance related issues. The used Raspberry Pi 3B can be used as a web server, but it is not protected against hackers. However, since a web server was used to display information, not much time and effort is needed to ensure availability of this central information on other devices, such as the supervisors' computer; only a network connection with the central node is required. If a lot of supervisors or other applications use the status display, the web server will be unable to function and will possibly stop receiving data from the sensor nodes. It is advisable to separate web server functionality from the central node. An option is to use two embedded systems in the central node, one to act as web server and one for the central node functionality, with a shared database. Another option is to store the database in the 'cloud', and provide access over the internet. Here, more advanced functionality could be implemented, such as a combined view of multiple vessels or docks, every stored User ID could be connected to a name, and an attendance check could be performed. More time effort could be invested to enhance the stability and functionality of the central display, when multiple people are using it simultaneously.

[1] These settings PAN ID, Encryption Enable, and a pre-shared key on the ZigBee module

A

Project Description

Bachelorproject: Life Counter

During the dry docking/repair period of vessels the owner is responsible for the safety of its crew and its subcontractors onboard. Therefore a system is required for identification and logging of the crew and visitors. The entrance of a vessel during repair is called a gangway. Aim of the project is to develop a system that will detect people leaving or entering the vessel through one of two gangways. In addition the number of people on the vessel has to be displayed physically somewhere at the shipyard. It's important that this system is highly reliable due to the importance of accurate data.

The system should be able to:

- Count people entering or leaving the vessel (without obstructing people or requiring a certain action) through one of the two gangways (width of gangway < 1.5 meters)
- Physically display the number of people currently on the vessel
- Distinguish crew, shipyard personnel and subcontractors

And should physically exist of:

- Two systems, each to be placed near a gangway.
- A counter-display to be placed at some location within 100 meters from the vessel

The responsibility for the solution of these problems can easily be divided over the subgroups, while tight cooperation is necessary for the successful realization of this project. A suggestion of the division is:

Communication subsystem

- Design a communication system. The communication should be able to bridge distances of up to 150 metres.
- Choose a fitting communication protocol for a total of 3 parts (2 ports at gangways, display). Equip the communications system with appropriate error correction methods.
- Create a log of the various states of the communication system and the communication itself.
- At some docking facilities there is an internet connection available. Find a way how this could be used to relay the live number of people inside the vessel.
- (optional) Figure out a solution for adding more communication parts to the system (e.g. thrid or fourth gate, additional displays etc)
- (optional) Determine the effects of weather conditions on the communication channel
- (optional) Add/develop location tracker of the three system parts
- (optional) Add a security layer to the communication

Registration/detection subsystem

- Develop a system that determines the number of people inside the vessel
- Determine people leaving and entering the vessel at a gangway
- Find out a way for distinguishing crew, shipyard personnel and subcontractors
- Create a log of the people onboard
- Create a system that displays the number of people onboard the vessel
- (optional) Implement a way to assess the quality of the systemcount per individual crossing
- (optional) Blackout protection (develop a backup energy system in case of blackout)
- (optional) Create an error log and system status log of various operating states

The suggestion is made to first create a small version of the implementation (proof of concept) and if successful, designing a large scale prototype.

Resources:

A couple of raspberry pi's, arduino's are available. In addition a budget is available of 1500 euro's. This amount can be spend in consultation with project proposer.

Project proposer:
Menno Gravemaker (06-13057060 mennogravemaker@gmail.com)

B

Python Source Code

B.1. Central Node

B.1.1. File: Central_node.py

```
#!/usr/bin/env python
#THIS FILE CONTAINS THE ACTIVATION OF THE CENTRAL SYSTEM
import serial
from settings import *
from modules.base import *
from xbee import ZigBee

#ACTIVATE SERIAL
serial_port = serial.Serial(serialport, baudrate)

init(serial_port, rpi)
from modules.base import xbee

#DISCOVER
discover()
#INIT MASTER
init_master()

#LET IT RUN UNTIL KEYBOARD INTERUPT
while True:
        try:
                time.sleep(0.001)
        except KeyboardInterrupt:
                break
xbee.halt()
serial_port.close()
```

B.1.2. File: Settings_central_node.py

```
#THIS FILE CONTAINS ALL THE SETTINGS FOR THE SENSOR NODE

rpi = 'node'
serialport = '/dev/ttyUSB0' #connection type
baudrate = 9600
twaitdiscovery = 10 #time to let all poortjes respond
node_identifier = 1
## address for unmarced
address = '\x00\x00\x00\x00\x00\x00\x00\x00'
addressrange = '\xff\xfe'
coor = 0 #COORDINATOR = FALSE
```

B.2. Sensor Node

B.2.1. File: Sensor_node.py

```
#THIS FILE CONTAINS THE FUNCTION TO START THE SENSOR NODE

#!/usr/bin/env python
import serial
import time
from settings import *
from modules.base import *
from modules.error import *
from xbee import ZigBee
#open serial
serial_port = serial.Serial(serialport, baudrate)
#init system
init(serial_port,rpi)
try:
        from modules.base import xbee,timer_act, timer_disc
except:
        print "Modules laoded"
rxaddress = '\x00\x00\x00\x00\x00\x00\x00\x00'
addressrange = '\xff\xfe'

#continue until keyboard interupt
while True:
        try:
                time.sleep(10.001)
#               init_random_db()
        except KeyboardInterrupt:
                break
try:
        xbee.halt()
        serial_port.close()
        timer_disc.cancel()
        timer_act.cancel()
except:
        print "Modules shutdown
```

B.2.2. File: Settings_sensor_node.py

```
rpi = 'node'
serialport = '/dev/ttyUSB0'
baudrate = 9600
twaitdiscovery = 10 #time to let all poortjes respond
node_identifier = 1
## address voor ongemarkeerd
address = '\x00\x00\x00\x00\x00\x00\x00\x00'
addressrange = '\xff\xfe'
coor = 0
```

B.3. Python Modules

B.3.1. File: Modules/base.py

```
#!/usr/bin/env python
#THIS FILE CONTAINS ALL THE XBEE FUNCTIONALITY AND CONNECTIONS TO THE OTHER FILES.
import serial
from threading import Timer
import time
import hexdump
import os
import re
from settings import *
from database import *
from xbee import ZigBee
```

```
#SET SYSTEM TIME
def setTime(timestamp):
        command = 'sudo date -s \"$(date --date=\'@'+timestamp+'\')\"'
        os.system(command)
        return 0

#INIT SYSTEM
def init(serial_port, rpi):
        global xbee
        global timer_disc
        global timer_act
        xbee = ZigBee(serial_port, callback=receiveData, escaped=True)
        init_actions()                  #init database
        if rpi == 'central':
                # configureer dit apparaat als een centrale node
                timer_disc = Timer(20,discover)
                timer_disc.start()
                print "xbee geconfigureerd als central rPi"
        elif rpi == 'node':
                # configureer dit apparaat als een secundaire node (bij de poortjes)
                timer_act = Timer(10,send_act)
                timer_act.start()
                print "xbee geconfigureerd als sensor rPi"
        else:
                print "er is een error opgetreden bij het aanmaken van de poort"
                quit()

#SEND DATA TO CENTRAL
def sendData(address, datatosend, addressrange):
        # stuur de data die doorgegeven wordt via de argumenten.
        xbee.send('tx',dest_addr_long = address,dest_addr = addressrange,data =
            datatosend)
        #ZigBee.send('rx')
        print ''
        print ''
        print 'Verzonden : '+datatosend
        return 0

#GET TIME
def getTimestamp():
        return str(time.time())

#RECEIVE STATUSCODE AND TRANSLATE TO HUMAN LANGUAGE
def recieveData_tx_statuscode(rxdata):
        if rxdata == '00':
                rxdata = 'Transmission succes: Pakketje succesvol verzonden'
                # Haal nu deze dinges uit de database.
        elif rxdata == '01':
                rxdata = 'Transmission error: MAC ACK Failure'
        elif rxdata == '02':
                rxdata = 'Transmission error: CCA Failure'
        elif rxdata == '15':
                rxdata = 'Transmission error: Invalid destination port'
        elif rxdata == '21':
                rxdata = 'Transmission error: Network ACK failure'
        elif rxdata == '22':
                rxdata = 'Transmission error: Not Joined to network'
        elif rxdata == '23':
                rxdata = 'Transmission error: Self-addressed'
        elif rxdata == '24':
                rxdata = 'Transmission error: Address not found'
        elif rxdata == '25':
                rxdata = 'Transmission error: Route not found'
        elif rxdata == '26':
                rxdata = 'Transmission error: Broadcast source failed to hear a
                    neighbour relay the message'
        elif rxdata == '2b':
                rxdata = 'Transmission error: Invalid binding table index'
        elif rxdata == '2c':
                rxdata = 'Transmission error: Resource error lack of free buffers,
                    timers, etc.'
```

```
        elif rxdata == '2d':
                rxdata = 'Transmission error: Attempted broadcast with APS
                    transmission'
        elif rxdata == '2e':
                rxdata = 'Transmission error: Attempted multicast with APS
                    transmission, but EE=0'
        elif rxdata == '32':
                rxdata = 'Transmission error: Resource error lack of free buffers,
                    timers, etc.'
        elif rxdata == '74':
                rxdata = 'Transmission error: Data payload too large'
        elif rxdata == '75':
                rxdata = 'Transmission error: Indirect message unrequested'
        else:
                rxdata = 'Transmission error: Unknown error'
        return rxdata

#RECEIVE DATA COMMAND AND activate actions
def receiveData_command(data):
        # Commando's zijn:
        # TMS = timestamp
        #       DATA = TMS[timestamp]
        # ACA = action acknoledge
        #       DATA = ACA[transmission_id],[timestamp]
        # ACT = action
        #       DATA = ACT[node_id],[transmission_id],[user_group],[user_id],[
            richting],[timestamp]
        # DIS = discover node command
        #       DATA = DIS
        # DIR = discover response
        #       DATA = DIR[node_id]
        # ERR = ERRor
        #       DATA = ERR[node_id],[error_type],[timestamp]

        rfdata = data['rf_data']
        address_long = data['source_addr_long']
        address = data['source_addr']
        result = re.match( '^...',rfdata, flags=0)
        command = result.group(0)
        payload = re.sub( '^...','',rfdata)

        if command == 'TMS':
                setTime(payload)
        elif command == 'ACT':
                # is received by master and send by node
                receive_act(address_long, address, payload)
        elif command == 'ACA':
                # master sends back and is received by the sensor
                # if this succeeds, the entry is deleted from the database
                receive_aca(address_long, payload)
        elif command == 'DIA':
                # is send by the node as an answer from the master
                receive_dia(address_long, payload)
        elif command == 'DIS':
                # DISCOVER commando from the master
                # node sends data back to get discovered
                txdata = 'DIA'+str(get_node_identifier())
                sendData(str(data["source_addr_long"]), txdata, str(data["
                    source_addr"]))
        elif command == 'ERR':
                # an error is received by the master from the node. this is written
                    in de error table.
                receive_err(payload)
        else:
                print "We received something that wasn't defined:"
                print str(data)
        return 0

#RECEIVED DATASTRING AND SPLIT IT
def receiveData(data):
        if data['id'] == 'rx_explicit':
```

```
                receiveData_command(data)
        elif data['id'] == 'tx_status':
                recieveData_tx_statuscode(str.lower(hexdump.dump(data['
                    deliver_status'])))
        else:
                print 'We received something else: '+str(data)
#       print rxdata
        return 0

#HANDLE DISCOVER COMMAND
def rxdiscover(data):
        #print str(data) #debug mode
        #if data id = tx
        naam = str(data['parameter']['node_identifier'])
        adres = str(hexdump.dump(data['parameter']['source_addr_long']))
        naamlist.append(naam)
        adreslist.append(adres)
        return 0

#Send acknowlegdement
def send_act():
        #read database and transmit the package
        # read db to list of variables
        addressrange = '\xff\xfe'
        address = '\x00\x00\x00\x00\x00\x00\x00\x00'
        result = ''
        node_id = get_node_identifier()
        message_id = ''
        try:
                result = r_all()
                for row in result:
                        print row
                        message_id,user_group,user_id,direction,timestamp = row
                        print ""
                        print ""
                        print 'er is iets om te verzenden:'
                        print 'node_id = '+str(node_id)
                        print 'message_id = '+str(message_id)
                        print 'user_group = '+str(user_group)
                        print 'user_id = %i' %(int(user_id))
                        print 'direction = '+str(direction)
                        print 'timestamp = '+str(timestamp)
                        data = 'ACT'+str(node_id)+','+str(message_id)+','+str(
                            user_group)+','+str(int(user_id))+','+str(direction)+','+
                            str(timestamp)
                        print str(data)
                        sendData(str(address), data, str(addressrange))
        except:
                print "We received an error during the sending and receiving."

        print "Klaar met versturen!"
        timer_act = Timer(10,send_act)
        timer_act.start()

#RECEIVE ACA
def receive_aca(address_long, payload):
        transmission_id,timestamp = payload.split(',')
        setTime(timestamp)
        d_entry(transmission_id) #DELETE ENTRY FROM DB
        print ""
        print ""
        print "New action verified, can delete from database"
        print 'transmission id = '+transmission_id
        print 'timestamp = '+timestamp
        return 0
#DISCOVER FUNCTION
def discover():
        data = 'DIS'
        addressrange = '\xff\xfe'
        address = '\x00\x00\x00\x00\x00\x00\xff\xff'
        sendData(str(address), str(data),str(addressrange))
```

```
        timer_disc = Timer(20,discover)
        timer_disc.start()
        return 0

#DIA FUNCTION RECEIVED
def receive_dia(adres, node_identifier):
        print 'Node '+node_identifier+' is active!'
        ## WRITE ID AND TIME IN DATABASE
        ## CHECK FOR NODES
        ## ACTIVATE ERROR LIGHT
        w_val_n(node_identifier, adres, str(getTimestamp() ))
        print "diafunction done"
        return 0

#ACT FUNCTION RECEIVED
def receive_act(address_long, address_range, payload):
        # Write it in the database.
        # data is being received [node_id],[transmission_id],[tag_id],[richting],[
            timestamp]
        node_id,message_id,user_group,user_id,direction,timestamp = payload.split(',
            ')
        try:
                w_val_m(user_group, user_id, direction, timestamp)
        except:
                print "it broke down"
        print ""
        print ""
        print "New action detected!"
        print 'node_id = %i' %(int(node_id))
        print 'message_id = %i' %(int(message_id))
        print 'user_group = %i' %(int(user_group))
        print 'user_id = %i' %(int(user_id))
        print 'direction = %i' %(int(direction))
        print 'timestamp = %i' %(int(timestamp))
        txdata = 'ACA'+message_id+','+str(getTimestamp())
        sendData(str(address_long),txdata,str(address_range))
        raise Send_Error ##New
        return 0

#RECEIVED ERROR
def receive_err(payload):
        node_id,etype,timestamp = payload.slit(',') #Split payload
        w_val_e(node_id,etype,timestamp)                        #write to database
        return 0

#GET NODE IDENTIFIER
def get_node_identifier():
        return node_identifier

#SEND TEST DATA
def send_example():
        addressrange = '\xff\xfe'
        address = '\x00\x00\x00\x00\x00\x00\x00\x00'
        tag_id = '0123456789ABCDE012345678'
        data = str('ACT1,1433,'+tag_id+',in,1481811096.02')
        sendData(address, data, addressrange)


#INSERT RANDOM DATA IN THE DATABASE
def init_random_db():
        w_val(1,'3378402378042804312',-1)
        w_val(-1,'34213421432143214423',-1)
        w_val(2,'4321432143214321',-1)
        w_val(1,'47280437281043281094',1)
        w_val(1,'3378402378042804312',-2)
```

B.3.2. File: Modules/database.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
```

```
#THIS FILE CONTAINS ALL DATABASE FUNCTIONS.
#THIS FILE CONTAINS:
# -init  || database openen
# -create_table || tabel make if it does not excist
# -writeval()
# -readquery
# -readall
# -readoldest
# -readnewest
# -close
import hexdump
import sqlite3 as sqlite
import sys, time
from error import *

##Read query executer
def r_executequery(query):
        returnvalues=[]
        try:
                cur.execute(query)
        except sqlite.Error, e:
                raise Read_Error_Database #raise error
                print "Error %s:" % e.args[0]


        finally:
                for row in cur:
                        returnvalues.append(row)

                return returnvalues

##Write query executer
def w_executequery(query):
        try:
                cur.execute(query)
        except sqlite.Error, e:
                raise Write_Error_Database

                print "Error %s:" % e.args[0]

        finally:
                con.commit() #WRITE TO DATABASE. ESSENTIAL FUNCTION
                return 0

##CREATE ACTIONS
def create_table_actions():
        query="CREATE TABLE " +  tablename + "(message_id INTEGER PRIMARY KEY,
            user_group INTEGER(1), user_id TEXT, direction INTEGER(1), time timestamp
             default (strftime(\'%s\', \'now\')));"
        w_executequery(query)
        return 0

##CREATE SERVICE
def create_table_service():
        query="CREATE TABLE " +  tablename_s + "(id TEXT PRIMARY KEY, time timestamp
             default (strftime('%s', 'now')));"
        w_executequery(query)
        return 0

##CREATE MASTER
def create_table_master():
        query="CREATE TABLE " +  tablename_m + "(user_id TEXT NOT NULL, user_group
            INTEGER(1) NOT NULL, inout INTEGER(1), first_time INTEGER, last_time
            INTEGER, scanned_tags INTEGER(1), PRIMARY KEY (user_id, user_group));"
        w_executequery(query)
        return 0

##CREATE ERROR
def create_table_error():
        query="CREATE TABLE " +  tablename_e + "(node_id INTEGER(1) NOT NULL, etype
```

```
            TEXT NOT NULL, timestamp INTEGER);"
        w_executequery(query)
        return 0

##CREATE NODES
def create_table_nodes():
        query = "CREATE TABLE "+ tablename_n + "(node_id INTEGER(1) NOT NULL,
            node_address TEXT NOT NULL, last_active INTEGER, PRIMARY KEY (node_id,
            node_address))"
        w_executequery(query)
        return 0

##ACTIONS: DELETE ENTRY FROM ACTIONS
def d_entry(message_id):
        query = "DELETE FROM "+tablename+" WHERE message_id="+str(message_id)
        w_executequery(query)
        return 0

##ACTIONS: read first value from database
def r_first():
        query = "SELECT * FROM %s ORDER BY message_id DESC LIMIT 1" %(tablename)
        return(r_executequery(query))

##ACTIONS: read last value from database
def r_last():
        query = "SELECT * FROM %s ORDER BY message_id ASC LIMIT 1" %(tablename)
        return(r_executequery(query))

##ACTIONS: read all from database
def r_all():
        query = "SELECT * FROM %s" %(tablename)
        return(r_executequery(query))

##ACTIONS: write to database
def w_val(user_group, user_id, direction):
        query="INSERT INTO " + tablename +  " (user_group, user_id, direction)
            VALUES (%i, '%s', %i);" %(int(user_group), str(int(user_id)), int(
            direction))
        print query
        w_executequery(query)
        return 0

##MASTER: write to database
def w_val_m(user_group, user_id, direction, timestamp):
        # CHECK IF ENTRY IS ALREADY THERE
        query="SELECT COUNT(*) FROM " + tablename_m + " WHERE user_id=\'" + str(int(
            user_id)) + "\' AND user_group = \'" + str(int(user_group)) + "\';"
        ans=r_executequery(query)[0]    #HOW MANY ID in DATABASE
        dir=int(direction)/abs(int(direction))  #SEND + of -1
        if int(ans[0]) == int(0):
                query="INSERT INTO "+tablename_m+" (user_id, user_group, inout,
                    scanned_tags, first_time, last_time) VALUES ('%s', %i, %i, %i, %i
                    , %i);" %(str(int(user_id)), int(user_group), int(dir), int(
                    direction), int(timestamp), int(timestamp))
                w_executequery(query)
        else:
                # CHECK IF TIMESTAMP IS OLDER THEN THE LAST ONE
                query = "SELECT last_time FROM " + tablename_m + " where user_id =
                    \'" +str(int(user_id))+"\' AND user_group = \'" + str(int(
                    user_group)) + "\';"
                result = r_executequery(query)[0]
                if int(timestamp) > int(result[0]):
                        # if timestamp is bigger --> write
                        print "timestamp is bigger then the last one. write to
                            database"
                        query="UPDATE "+ tablename_m + " SET inout=\'" + str(int(dir
                            ))+ "\', scanned_tags=\'"+str(abs(int(direction)))+"\',
                            last_time=\'"+str(int(timestamp))+"\' WHERE user_id=\'"+
                            str(int(user_id))+"\';"
                        w_executequery(query)
                else:
```

```
125                         print "timestamp is not bigger then the last one. it's not
                                used"
            return 0

    ##NODES: UPDATES ADRES to DATABASE
    def w_val_n(node_id, node_address, timestamp):
130             query = "SELECT COUNT(*) FROM " + str(tablename_n) + " WHERE node_id = \'" +
                     str(node_id) + "\' AND node_address = \'" + str.lower(hexdump.dump(
                    node_address)) + "\';"
                result = r_executequery(query)[0]
                print result[0]
                print str(timestamp)
                if int(result[0]) == int(0):
135                     query = "INSERT INTO " + str(tablename_n) + " (node_id, node_address
                            , last_active) VALUES (\'"+str(int(node_id))+"\',\'"+str.lower(
                            hexdump.dump(node_address))+"\',\'"+str(timestamp)+"\');"
                        print query
                else:
                        query = "UPDATE " + str(tablename_n) +" SET last_active = \'" + str(
                            timestamp) + "\' WHERE node_id = \'" + str(int(node_id)) +"\' AND
                             node_address = \'" + str.lower(hexdump.dump(node_address)) + "
                            \';"
                print query
140             w_executequery(query)
                return 0

    ##NODES: UPDATES ADRES to DATABASE
    def w_val_e(node_id, etype, timestamp):
145             etype = int(etype)

                #NUMBER TO TEXT
                if(etype==1):
                        et = "Regular undifined error"
150             elif(etype==2):
                        et = "Critical error, restart"
                elif(etype==3):
                        et = "Bad Critical error, reboot"
                elif(etype==4):
155                     et = "Write error database"
                elif(etype==5):
                        et = "Read error database"
                elif(etype==6):
                        et = "Connect error"
160             elif(etype==7):
                        et = "Send error"
                else:
                        et = "Undefined error. Might be critical"

165             query = "INSERT INTO " + str(tablename_e) + " (node_id, etype, timestamp)
                    VALUES (\'"+str(int(node_id))+"\',\'"+str(et)+"\',\'"+str(timestamp)+"\')
                    ;"
                w_executequery(query)
                return 0



170
    ##SERVICE: read last value from database
    def r_last_s():
            query = "SELECT * FROM %s ASC LIMIT 1" %(tablename_s)
            return(r_executequery(query))
175
    ##SERVICE: clear service db records for the day
    def clear_service_cache():
            tijd=int(time.time())
            dag = time_to_store
180         deltatijd = tijd-dag
            query="DELETE FROM " + tablename_s + " WHERE time<"+deltatijd
            w_executequery(query)
            return 0

185 ##INIT NODE
```

```
def init_actions():
        global dbname
        global tablename
        global tablename_s
        global tablename_n
        global tablename_m
        global cur
        global con
        global query
        global time_to_store


        time_to_store = 2*24*3600 #TIME FOR LOCAL DATABASE

        dbname =  'lifecounter'
        tablename = 'actions'
        tablename_s = 'service'
        tablename_n = 'nodes'
        tablename_m = 'master'
        con = None


        try:
                con = sqlite.connect(dbname+'.db', check_same_thread=False)
                cur = con.cursor()


        except sqlite.Error, e:
                raise Connect_Error_Database
                print "Error %s:" % e.args[0]
                sys.exit(1)

        create_table_actions()
        create_table_master()
        create_table_nodes()
        return 0

##INIT MASTER
def init_master():
        global dbname
        global tablename_n
        global tablename_m
        global tablename_e
        global con
        global cur

        dbname =  'lifecounter'
        tablename_n = 'nodes'
        tablename_m = 'master'
        tablename_e = 'error'

        con = None
        cur = None

        try:
                con = sqlite.connect(dbname+'.db', check_same_thread=False)
                cur = con.cursor()


        except sqlite.Error, e:
                raise Connect_Error_Database
                print "Error %s:" % e.args[0]
                sys.exit(1)

        create_table_master()
        create_table_nodes()
        create_table_error()
        return 0

##DB CLOSE
def close():
```

```
        cur.close()
        con.close()
        return 0
```

B.3.3. File: Modules/error.py

```
#!/usr/bin/env python
#THIS FILE CONTAINS ALL THE ERROR CASES AND HANDLERS
import os, time
from xbee import ZigBee
from settings import *

##GENERAL ERRORS
class Error(Exception):
        def __init__(self):
                print "Regular undefined error. E1"
                etype = "RegularError"
                write_file(etype + "\t\t" + getTimestamp())
                send_err(1)


##CRITICAL ERROR --> RESTART SOFTWARE
class Critical_Error(Error):
        def __init__(self):
                print "Critical error, please contact your system administrator. E2"
                etype = "CriticalError"
                write_file(etype + "\t\t" + getTimestamp())
                write_file("Restart_All_Service \t\t" + getTimestamp())
                send_err(2)
                close() ##Close database
                os.system("/code/sh run_party.sh") #Start restart script
                sys.exit() #Stop everything



##BAD CRITICAL --> REBOOT COMPLETE SYSTEM
class Bad_Critical_Error(Error):
        def __init__(self):
                print "Critical error, please contact your system administrator. E3"
                etype = "Bad_CriticalError"
                write_file(etype + "\t\t" + getTimestamp())
                write_file("REBOOTSYSTEM\t\t" + getTimestamp())
                send_err(3)
                os.system("sudo init 6") ##Ghehe, reboot, maw ik weet niet wat er
                    stuk is

##DATABASE ERRORS --> DO NOTHING
class Write_Error_Database(Error):
        def __init__(self):
                print "Write error, database is probably busy. Please try again. E4"
                etype = "Write_Error"
                write_file(etype + "\t\t" + getTimestamp())
                send_err(int(4))
                print(str(getTimestamp()))
                pass


##DATABASE ERRORS --> DO NOTHING
class Read_Error_Database(Error):
        def __init__(self):
                print "REad error, database is probably busy. Please try again. E5"
                send_err(5)
                etype = "Read_Error"
                write_file(etype + "\t\t" + getTimestamp())

##DATABASE ERROR CRITICAL --> STOP
class Connect_Error_Database(Error):
        def __init__(self):
                print "Connection error. Please try again. E6"
                etype = "Connect_Error"
```

```
                write_file(etype + "\t\t" + getTimestamp())
                send_err(6)



##COMMUNICATION ERROR --> DO NOTHING
class Send_Error(Error):
        def __init__(self):
                print "Send error. Please try again. E7"
                etype = "Send_Error"
                write_file(etype + "\t\t" + getTimestamp())
                send_err(7)


##COMMUNICATION ERROR --> DO NOTHING
class Settings_Error(Error):
        def __init__(self):
                print "Settings did load properly. Please try again. E8"
                etype = "Settings_Error"
                write_file(etype + "\t\t" + getTimestamp())

###ESSESTIAL FUNCTIONS
#WRITE TO LOCAL LOG
def write_file(error_string):

        errorlog = open("error.log","a")
        if(os.path.getsize("error.log") > 0):
           errorlog.write("\n"+error_string)
        else:
           errorlog.write(error_string)

        errorlog.close()
        return 0

#GET TIME
def getTimestamp():
        return str(time.ctime())

#SEND TO CENTRAL
def send_err(etype):
        if(coor==0):
                etype = str(etype)
                node_id = str(get_node_identifier())            #get node_id
                addressrange = '\xff\xfe'                                       #ad.
                     of master Pi
                address = '\x00\x00\x00\x00\x00\x00\x00\x00'#ad. of master Pi
                data = str('ERR,'+node_id+','+etype+','+getTimestamp())#data
                sendData(address, data, addressrange)           #sendData
        elif(coor==1):
                print "Error received"
        else:
                print "Critical error. Settings did not load properly"
                raise Settings_Error
        return 0

def sendData(address, datatosend, addressrange):
        from base import xbee
        xbee.send('tx',dest_addr_long = address,dest_addr = addressrange,data =
            datatosend)
    #ZigBee.send('rx')
        print ''
        print ''
        print 'Verzonden : '+datatosend
        return 0


def get_node_identifier():
        return node_identifier
```

C

ZigBee range experiments

C.1. XCTU Range test

C.2. Experiment 1: range test

Table C.1: Raw data of figure 8.1. Shown are the minimal, maximal and 1-minute average receive

Distance (m)	mean RSSI (dBm)	minimum	maximum	percentage of successful transmitted packets (%)
50	-40	-55	-33	100
100	-43	-50	-40	100
150	-47	-51	-45	100
200	-56	-66	-52	100
250	-67	-79	-63	95
300	-66	-68	-57	100
350	-64	-71	-61	100
400	-74	-78	-68	84
450	-65	-67	-64	100
500	-70	-80	-62	92
550	-76	-81	-73	98
600	-75	-81	-70	79
650	-78	-82	-65	98
700	-100	-	-	0

C.3. Experiment 3:Measurement locations

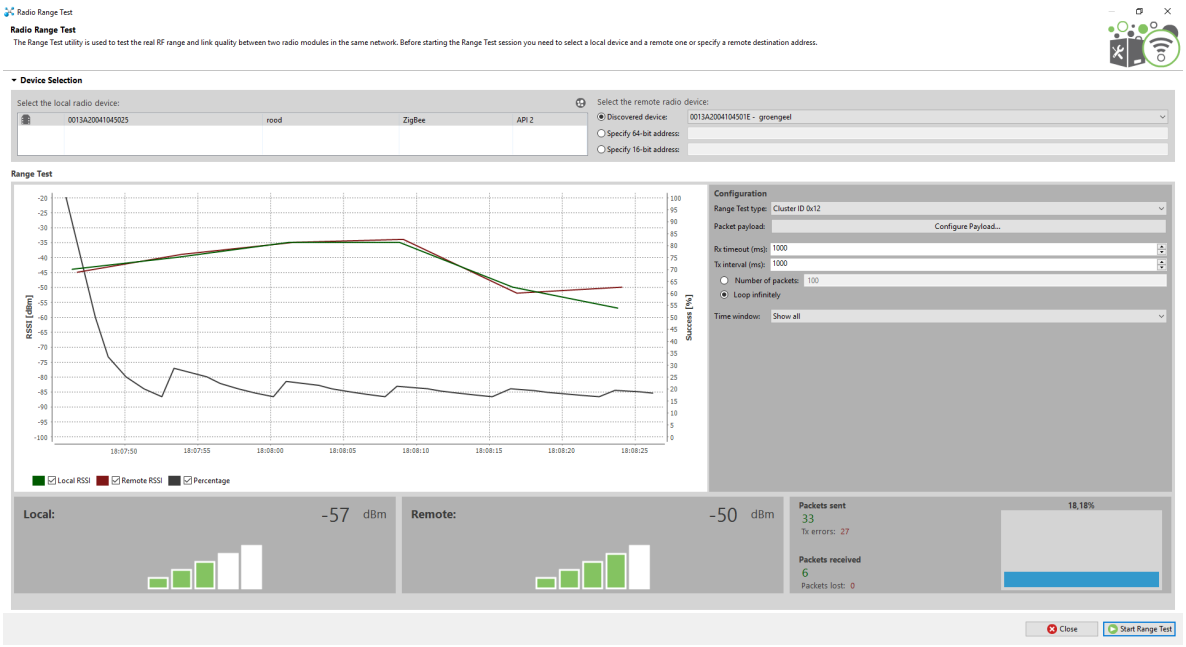


Figure C.1: A screenshot of the XCTU range test panel

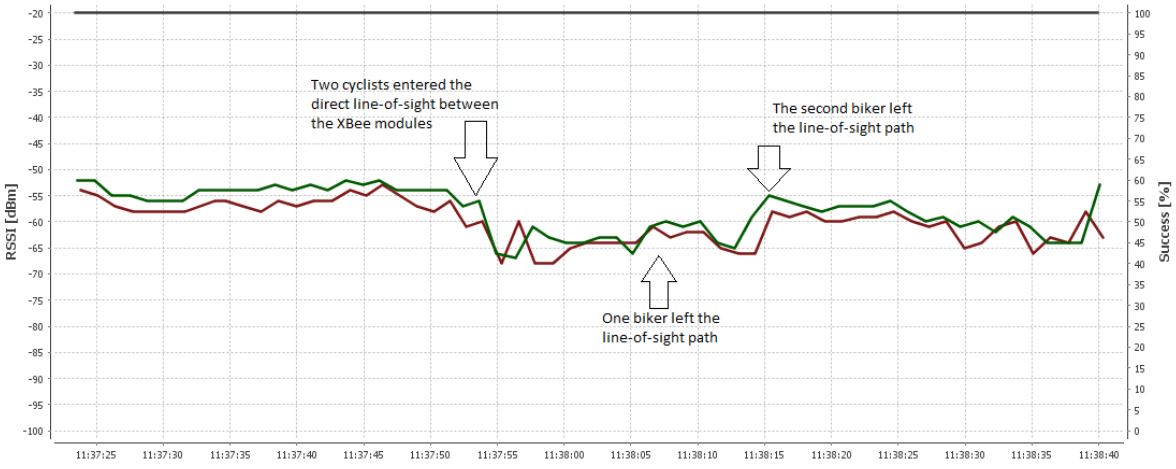


Figure C.2: One minute of measuring with the XBee modules separated 200 meters. Clearly visible are the changes in local received signal strength index (RSSI) (the local XBee node, green) and the remote RSSI (the remote XBee node, black). Shown is that all packets are transferred correctly; the percentage of successful transmitted packets is 100.

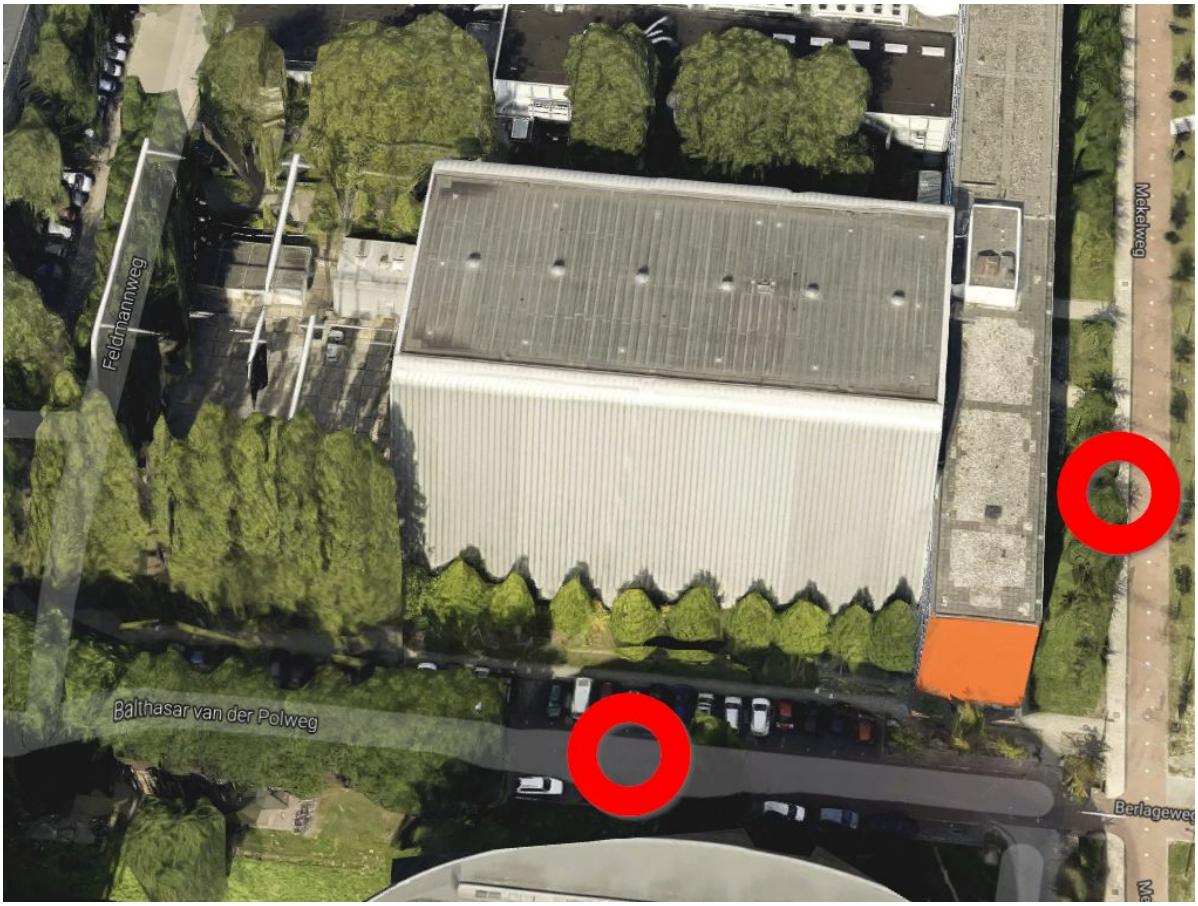


Figure C.3: The location of measurement of the third experiment

Bibliography

[1] Russell Barnes. Raspberry pi 3 benchmark. *MagPi*, 43:13, March 2016. URL https://www.raspberrypi.org/magpi-issues/MagPi43.pdf.

[2] SIG Bluetooth. Bluetooth specification, 2010.

[3] Vongsagon Boonsawat, Jurarat Ekchamanonta, Kulwadee Bumrungkhet, and Somsak Kittipiyakul. Xbee wireless sensor networks for temperature monitoring. In *the second conference on application research and development (ECTI-CARD 2010), Chon Buri, Thailand*, 2010.

[4] G. Byrd. Tracking cows wirelessly. *Computer*, 48(6):60–63, June 2015. ISSN 0018-9162. doi: 10.1109/MC.2015.154.

[5] *picoUPS-120-ATV - Quick installation guide*. CarTFT.com, 2010. URL https://www.cartft.com/support_db/support_files/PicoUPS-120-ATV_Manual_EN.pdf.

[6] J. Chen, J. Zeng, and Y. Zhou. Packet error rate of zigbee under the interference of rfid. In *Advanced Computer Control, 2009. ICACC '09. International Conference on*, pages 581–585, Jan 2009. doi: 10.1109/ICACC.2009.82.

[7] *GUIDANCE REGARDING THE PROVISIONS OF CHAPTER XI-2 OF THE ANNEX TO THE INTERNATIONAL CONVENTION FOR THE SAFETY OF LIFE AT SEA*. ClassNK, 1976. URL https://www.classnk.or.jp/hp/pdf/activities/statutory/isps/ISPS-Code-B-e.pdf.

[8] *XBee®/XBee-PRO S2C ZigBee® RF Module*. Digi International Inc., 2016. URL http://www.digi.com/resources/documentation/digidocs/pdfs/90002002.pdf.

[9] Electronic Communications Committee (ECC). *THE EUROPEAN TABLE OF FREQUENCY ALLOCATIONS AND APPLICATIONS IN THE FREQUENCY RANGE 8.3 kHz to 3000 GHz (ECA TABLE)*, 2016. URL http://www.erodocdb.dk/docs/doc98/official/pdf/ERCRep025.pdf.

[10] Braham Ferreira and Wim van der Merwe. *The principles of electronic and electromechanic power conversion*, pages 169–171. John Wiley & Sons, Inc., Hoboken, New Jersey, 2013. ISBN ISBN 978-1-118-79885-0.

[11] Braham Ferreira and Wim van der Merwe. *The principles of electronic and electromechanic power conversion*, pages 152–169. John Wiley & Sons, Inc., Hoboken, New Jersey, 2013. ISBN ISBN 978-1-118-79885-0.

[12] Braham Ferreira and Wim van der Merwe. *The principles of electronic and electromechanic power conversion*, page 142. John Wiley & Sons, Inc., Hoboken, New Jersey, 2013. ISBN ISBN 978-1-118-79885-0.

[13] Elinux Foundation. Raspberry pi hardware, 2016. URL http://elinux.org/RPi_Hardware. Accessed: 2016-12-16.

[14] F. R. Gfeller and U. Bapst. Wireless in-house data communication via diffuse infrared radiation. *Proceedings of the IEEE*, 67(11):1474–1486, Nov 1979. ISSN 0018-9219. doi: 10.1109/PROC.1979.11508.

[15] *User manual Odroid C2*. Hardkernel, 2016. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4762946.

[16] Dave Hugher. Raspberry pi 3 benchmark. *HCC EMBEDDED*, May 2015. URL http://rtcmagazine.com/articles/view/110540.

[17] Digi international. *DIA Difference Between ZigBee and DigiMesh*, 2013. URL http://www.digi.com/wiki/developer/index.php/DIA_Difference_Between_ZigBee_and_DigiMesh.

[18] L. Jiang, L. D. Xu, H. Cai, Z. Jiang, F. Bu, and B. Xu. An iot-oriented data storage framework in cloud computing platform. *IEEE Transactions on Industrial Informatics*, 10(2):1443–1451, May 2014. ISSN 1551-3203. doi: 10.1109/TII.2014.2306384.

[19] F. Leens. An introduction to i2c and spi protocols. *IEEE Instrumentation Measurement Magazine*, 12(1): 8–13, February 2009. ISSN 1094-6969. doi: 10.1109/MIM.2009.4762946.

[20] William Lehr and Lee W McKnight. Wireless internet access: 3g vs. wifi? *Telecommunications Policy*, 27(5–6):351 – 370, 2003. ISSN 0308-5961. doi: http://dx.doi.org/10.1016/S0308-5961(03)00004-1. URL http://www.sciencedirect.com/science/article/pii/S0308596103000041. Compeitition in Wireless: Spectrum, Service and Technology Wars.

[21] A. Lengyel and K.P. van der Mark. Bachelor of Science Thesis - Non-intrusive person detection. Technical report, Delft University of Technology, Feb 2017.

[22] Yehui Liu. *Study on Smart Home System Based on Internet of Things Technology*, pages 73–81. Springer London, London, 2013. ISBN 978-1-4471-4793-0. doi: 10.1007/978-1-4471-4793-0_9. URL http://dx.doi.org/10.1007/978-1-4471-4793-0_9.

[23] *LoRaWAN Specifications*. LoRA Alliance, 2015. URL https://www.lora-alliance.org/portals/0/specs/LoRaWAN%20Specification%201R0.pdf.

[24] The MagPi. The making of raspberry pi zero, 2016. URL https://www.raspberrypi.org/magpi/making-pi-zero/. Accessed: 2016-12-16.

[25] Paul Malmsten. python-xbee documentation, 2013.

[26] *PIC16F627A/628A/648A Data Sheet*. Microchip, 2009. URL http://ww1.microchip.com/downloads/en/DeviceDoc/40044G.pdf.

[27] Piet van Mieghem. *Data Communications Networking*, pages 9–18. Purdue University Press, West Lafayette, IN, USA, 2006. ISBN 9085940087.

[28] Piet van Mieghem. *Data Communications Networking*, page 142. Purdue University Press, West Lafayette, IN, USA, 2006. ISBN 9085940087.

[29] *OPENUPS 6-30V Intelligent Uninterruptible Power Supply*. mini-box.com, 2014. URL http://resources.mini-box.com/online/PWR-OpenUPS/PWR-OpenUPS-hardware-manual.pdf.

[30] *C.H.I.P. Manual*. Next Thing Co, 2015. URL https://docs.getchip.com/chip.html.

[31] Accu Servie Holland NL. *LP series-General Purpose LP12250*, 2016. URL https://smetsersbeheer.nl/files/pdf/VRLA-LP-12-250-ah-AGM-Deep-cycle-Accu-Service-Holland.pdf.

[32] *UPS PIco Uninterruptible Power Supply with Peripherals and I2C control Interface for use with Raspberry Pi® B+, A+, B, and A*. PiModules, ModMyPi, 2014. URL http://www.pimodules.com/_pdf/UPS_PIco_Final_PiMMP-R6.pdf.

[33] J Call Quer. *Arduino Uno*, 2014. URL http://digital.csic.es/bitstream/10261/127788/7/D-c-%20Arduino%20uno.pdf.

[34] Gustavus J. Simmons. Symmetric and asymmetric encryption. *ACM Comput. Surv.*, 11(4):305–330, December 1979. ISSN 0360-0300. doi: 10.1145/356789.356793. URL http://doi.acm.org/10.1145/356789.356793.

[35] *XBEE AT Commands*. Sparkfun Electronics, 2011. URL https://cdn.sparkfun.com/learn/materials/29/22AT%20Commands.pdf.

[36] A. I. Stan, M. Swierczynski, D. I. Stroe, R. Teodorescu, S. J. Andreasen, and K. Moth. A comparative study of lithium ion to lead acid batteries for use in ups applications. In *2014 IEEE 36th International Telecommunications Energy Conference (INTELEC)*, pages 1–8, Sept 2014. doi: 10.1109/INTLEC.2014.6972152.

[37] M. Sun and Y. Qian. Study and application of security based on zigbee standard. In *2011 Third International Conference on Multimedia Information Networking and Security*, pages 508–511, Nov 2011. doi: 10.1109/MINES.2011.79.

[38] Tenma^{TM}. *Digital Multimeter: 72-7925*, January 15th 2017. URL http://www.farnell.com/datasheets/1955372.pdf?_ga=1.66499374.1092890716.1479808085.

[39] *LM2596 SIMPLE SWITCHER® Power Converter 150-kHz 3-A Step-Down Voltage Regulator*. Texas Instruments, 2016. URL http://www.ti.com/lit/ds/symlink/lm2596.pdf.

[40] *LM340, LM340A and LM7805 Family Wide VIN 1.5-A Fixed Voltage Regulators*. Texas Instruments, 2016. URL http://www.ti.com/lit/ds/symlink/lm7805.pdf.

[41] THE EUROPEAN PARLIAMENT AND THE COUNCIL. *REGULATION (EC) No 725/2004 on enhancing ship and port facility security*, March, 31 2004. URL http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2004:129:0006:0091:en:PDF.

[42] *UL9-12*. Ultracell, 2014. URL http://ultracell.net/datasheets/UL9-12.pdf.

[43] International Telecommunication Union. Radio regulations. *ITU Radio Regulations*, 2012.

[44] Eben Upton. *Raspberry Pi User Guide*, 2012. URL http://www.cs.unca.edu/~bruce/Fall14/360/RPiUsersGuide.pdf.

[45] D. Vassis, G. Kormentzas, A. Rouskas, and I. Maglogiannis. The ieee 802.11g standard for high data rate wlans. *IEEE Network*, 19(3):21–26, May 2005. ISSN 0890-8044. doi: 10.1109/MNET.2005.1453395.

[46] D. G. Vutetakis and H. Wu. The effect of charge rate and depth of discharge on the cycle life of sealed lead-acid aircraft batteries. In *IEEE 35th International Power Sources Symposium*, pages 103–105, Jun 1992. doi: 10.1109/IPSS.1992.282019.

[47] *400KHz 60V 4A Switching Current Boost / Buck-Boost / Inverting DC/DC Converter*. XLSEMI, 2016. URL www.sunrom.com/get/283300.

[48] Huanjia Yang, Lili Yang, and Shuang-Hua Yang. Hybrid zigbee {RFID} sensor network for humanitarian logistics centre management. *Journal of Network and Computer Applications*, 34(3):938 – 948, 2011. ISSN 1084-8045. doi: http://dx.doi.org/10.1016/j.jnca.2010.04.017. URL http://www.sciencedirect.com/science/article/pii/S1084804510000913. {RFID} Technology, Systems, and Applications.

[49] *Technical Manual - Powersports Batteries*. Yuasa, 2014. URL http://www.yuasabatteries.com/pdfs/TechManual_2014.pdf.