2016•2017
FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN
master in de industriële wetenschappen: elektronica-ICT

Masterproef

Secure and flexible sleep tracking device communications for integration in Smart Home networks

Promotor :
Prof. dr. ir. Ronald THOELEN

Copromotor :
dr. Kris AERTS

Promotor :
B Eng WINSLOW MIMNAGH

Copromotor :
ir. COEN LAUWERIJSSEN

Robbe Bloemen

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

2016•2017
Faculteit Industriële ingenieurswetenschappen
master in de industriële wetenschappen: elektronica-ICT

Masterproef

Secure and flexible sleep tracking device communications for integration in Smart Home networks

Promotor :
Prof. dr. ir. Ronald THOELEN

Copromotor :
dr. Kris AERTS

Promotor :
B Eng WINSLOW MIMNAGH

Copromotor :
ir. COEN LAUWERIJSSEN

Robbe Bloemen
Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

Preface

The last step before graduating as a master in Electronics and ICT Engineering technology, is writing a master's thesis. By doing my master's thesis at Zenseri, I have not only learned a lot at the technical level, but I have also gained my first experiences in the workplace. This would not have been possible without the help and support of others during the master's thesis. I like to thank my supervisors, family and friends.

In particular, I want to thank my external supervisor and CEO of Zenseri, Winslow Mimnagh, for his excellent guidance during the project. He gave me the chance to be part of his company and to participate in this interesting project. His door was always open for an enlightening conversation or a push in the right direction.

For technical assistance, I could always rely on the support of both my internal supervisors from the faculty as the employees of Zenseri and 2M Engineering. Prof. Dr. ir. Ronald Thoelen and Prof. Dr. Kris Aerts were assigned as internal supervisors. I am thankful for the useful feedback I received from them. I especially thank Henk Schär and Michel Sperling for their assistance during the project.

Last but not least, I would like to thank all other employees and other students at 2M Engineering for the enjoyable time that I have had throughout the year.

Robbe Bloemen

Table of Contents

List of tables

List of figures

List of abbreviations

AFE	Analog Front End
AP	Access Point
API	Application Programming Interface
DHCP	Dynamic Host Configuration Protocol
DDoS	Distributed Denial of Service
DNS	Domain Name System
DSP	Digital Signal Processing
HTTP	HyperText Transfer Protocol
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
OS	Operating System
OTA	Over-The-Air
PBC	Push-Button-Connect
PIN	Personal Identification Number
PMMA	PolyMethyl Methacrylate
REST	REpresentational State Transfer
SSDP	Simple Service Discovery Protocol
SSID	Service Set IDentifier
UDP	User Datagram Protocol
UI	User Interface
UPnP	Universal Plug and Play
URL	Uniform Resource Locator
UUID	Universally Unique IDentifier
WPS	Wi-Fi Protected Setup
XML	Extensible Markup Language

Abstract

"BEDsense" is a sleep sensor system under development at Zenseri BV. It is designed to integrate into Smart Homes and control lighting, heating and alarms when home residents are in bed. Data from the BEDsense will also be used for sleep analysis providing sleep profiles, quantitative metrics and recommendations to improve sleep quality of users. The objective of this master's thesis is to develop and in part implement a universal plug and play strategy in terms of connecting to the home Wi-Fi network, discovery and data communications. A second objective is to study a system solution whereby BEDsense data can be collected and used for sleep analysis. These objectives take into account the limitations of memory, computing power and data communication bandwidth/availability.

This thesis focuses on defining the communication specifications, implementation of a prototype system, development of a user interface via a Windows and Android app and finally definition of a sleep analysis system.

The result of this thesis is an agreement on the communication protocols, provisioning means (WPS- and AP-mode), SSDP protocol definitions making the BEDsense visible within the network, a user interface app and test tool for provisioning, configuration and diagnostic services in order to test performance. Subsequently, a system is created where raw sensor data is transformed into a format that is suitable for storage and transmission. These results form a first step for Smart Home integration and sleep analysis.

Abstract in het Nederlands

"BEDsense" is een slaapsensor onder ontwikkeling bij Zenseri BV. De sensor is ontworpen voor integratie in Smart Homes om het licht, verwarming en alarmen te controleren wanneer de bewoners in bed liggen. Daarnaast zal de BEDsense data ook dienen voor slaapanalyse waar het gebruikersprofielen en aanbevelingen voorziet om de slaapkwaliteit te verbeteren. Het doel van deze masterproef is het ontwerp en de implementatie van een universele strategie met betrekking tot het opzetten van Wi-Fi verbinding, het ontdekt worden en datacommunicatie. Daarnaast wordt een systeem onderzocht waar BEDsense data gebruikt kan worden voor slaapanalyse, rekening houdend met de beperkingen qua geheugen, rekenkracht en datatransmissie.

De initiële focus lag op het bepalen van de communicatiespecificaties. Vervolgens werd een prototype ontwikkeld. Aanvullend werd er een gebruikersinterface voorzien a.d.h.v. een Windows- en een Android app. Tot slot werd een systeem voor de slaapanalyse bedacht.

Om de Wi-Fi-verbinding op te zetten is er gekozen om WPS- en AP-mode te implementeren. Verder worden delen van het SSDP-protocol gebruikt om zichzelf zichtbaar te maken binnen het netwerk. De ontwikkelde test tool en app maken het mogelijk om de prestaties van het prototype te testen en te zorgen voor de gebruikersinterface. Aansluitend is er een systeem gecreëerd dat ruwe sensor data omzet in een formaat dat bruikbaar is voor opslag en transmissie. Deze resultaten vormen een eerste stap voor de Smart Home-integratie en de slaapanalyse.

1 Introduction

1.1 Project description

Zenseri B.V. is a Dutch startup company set up in May 2016 to develop and manufacture an innovative sleep sensor system, called BEDsense. Figure 1.1 shows the design of the BEDsense. BEDsense is the world's first noncontact sleep sensor that works underneath most mattresses. It combines a bed sensor, smartphone app and web-based app to help you track and better understand your sleeping patterns, and then creates personalized feedback and suggestions to help improve your sleep.

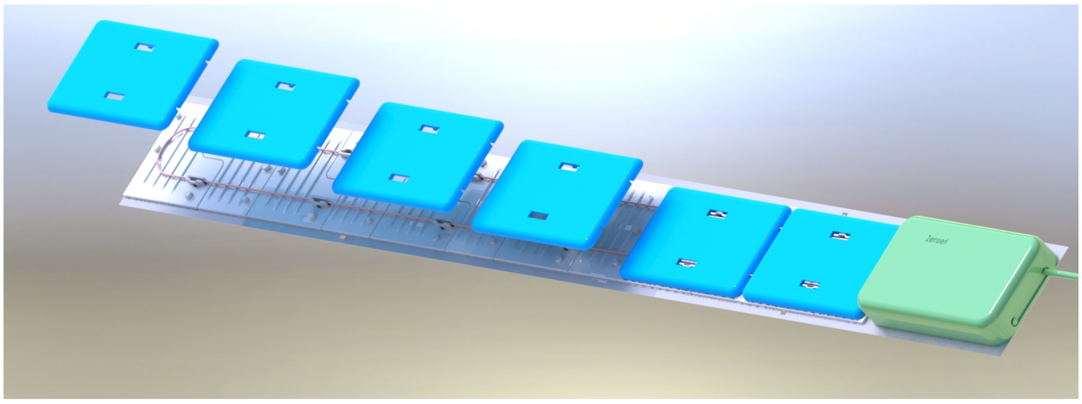
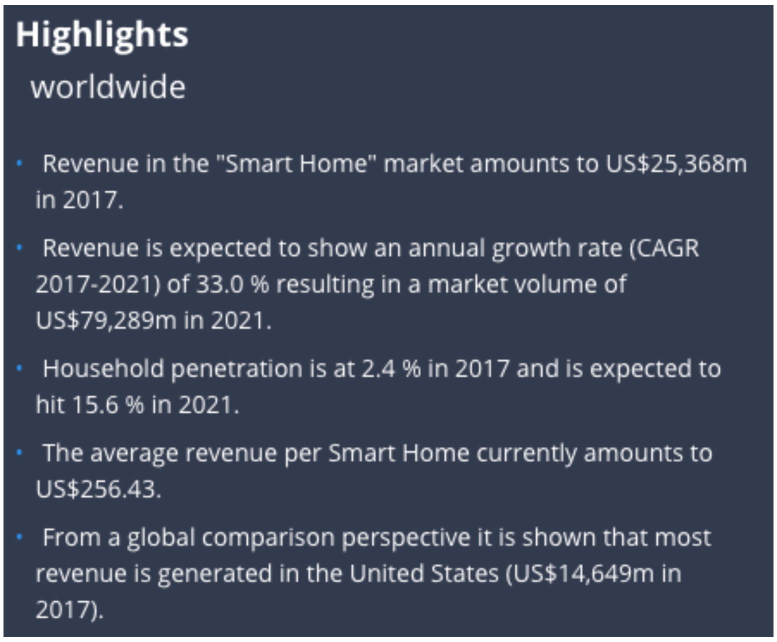


Figure 1.1: Sketch of a BEDsense prototype

The BEDsense has been developed to respond to a specific demand of Managed Smart Home providers who want to apply BEDsense into their platform in order to control lighting, heating, and alarms when residents go to bed. The benefits are:

- ease of use: most home owners do not want to be bothered turning the burglar alarm on/off, locking doors, reducing the temperature or turning on/off lights while they are in the house at night. If the Smart Home can detect if persons are in bed and/or asleep, the system itself can make these choices;
- cost effective systems allow energy efficiencies. When residents go to bed lights can be turned off, heating system can be set to night sleep mode and smart/connected appliances can be set on stand-by. When residents wake-up in the morning, alarms are switched off, heating and cooling systems are adjusted, as are the lights;
- unobtrusive placement: most home owners do not like sensors in bedrooms which can give discomfort (while sleeping), limit privacy or require repositioning (while changing bed cloths) the BEDsense solves these problems with sleek mechanical design, uses no camera's and once installed does not need to be moved;
- low cost: Smart home providers have an existing base of customers who pay a monthly subscription between 40 – 60 USD. For this they get equipment, installation and 24/7 monitoring. Smart home providers must offer the BEDsense to customers for this same subscription price, therefore they have asked for a price of 20 USD landed.

In order to make this project successful, Zenseri needs to be looking at the right marketplace that can truly benefit from the BEDsense and is large enough to sustain the business goals. Zenseri has therefore chosen to launch BEDsense initially in the North American market where, according to Statista [1], Smart Home household penetration is at 13.6 % (as opposed to Europe which is only 2.2%) in 2017 and is expected to hit 57.5 % in 2021. In 2016 there were 125 million households in the USA so we can estimate the number of Smart Homes is 17 million and the total number of beds is 44 million. The first market sector will be the managed Smart Home market, which is about 40% of the total Smart Home market, meaning that Zenseri's first target market size is over 17 million units. Zenseri is currently in negotiations with a large US customer in the managed Smart Home market and has over 1.2 million customers with some 3 million beds.


Highlights
worldwide

- Revenue in the "Smart Home" market amounts to US$25,368m in 2017.
- Revenue is expected to show an annual growth rate (CAGR 2017-2021) of 33.0 % resulting in a market volume of US$79,289m in 2021.
- Household penetration is at 2.4 % in 2017 and is expected to hit 15.6 % in 2021.
- The average revenue per Smart Home currently amounts to US$256.43.
- From a global comparison perspective it is shown that most revenue is generated in the United States (US$14,649m in 2017).

Figure 1.2: Wordwide Smart Home market data highlights [1]

The managed Smart Home provider combines security and home automation solutions with 24/7 monitoring. Security and trust are important conditions for proper functioning of a society. Unfortunately, the feeling of security has decreased in recent years. Managed Smart Home providers, that combine security and home automation, respond to this need for security. Nowadays the Smart Home system has a good overview of what the residents of the house are doing throughout the day. Though at night, it knows very little about the residents, which causes a big missing link in their platform.

Managed Smart Home providers in the USA see the BEDsense as a solution to their ''missing link'' problem that is how can a Smart Home provide feedback and benefits through the night when residents are asleep. The addition of the BEDsense makes it possible to drive other connected devices when the residents go to bed and fall asleep. Think of activating the alarm, lock doors, and turn off lights, lower the temperature. Moreover, if the system can detect when the residents wake up and get out of bed, it can deactivate the alarms, turn on the lights and temperature again, etc.

The next important goal of Zenseri is to use the BEDsense data for sleep analysis. There is an increasing interest in sleep analysis. A good night's rest is one of the most important parameters for the wellbeing of people, even for healthy people. In the context of a healthy lifestyle, a growing number of people want to know the quality of their sleeping and have the desire to have feedback, advice and recommendations on how to improve their sleep. Mothers want to understand if their children are getting enough sleep and a restful night, if not they want to know what they can do to create a healthy sleep environment. Within this trend, people increasingly integrate technology into their lives, with the purpose of collecting information about themselves in order to learn from it.

The BEDsense project aims to create a low-cost but reliable solution that makes it possible to generate and collect sleep information in an efficient way. It will then no longer function only as a sensor to expand the Smart Home system, but it will also be used to monitor the sleep quality of an individual. BEDsense data will be obtained from the Smart Home networks and collected on an external server where algorithms will analyze the user profile against big data. This results in sleep quality metrics and recommendations that will be sent back to the user in order to improve their sleep performance.

For both the above applications the BEDsense must be equipped with communication means and a user interface (UI) to enable seamless integration in a Smart Home network. A mobile app will provide the user interface for this integration. Users can configure and control the BEDsenses using the app. The resulting system will allow Zenseri to make a 'Smart-Bedroom'. This means that it must be able to monitor bed occupancy and sleep quality, upload data automatically to a server and use this information of the bedroom environment to optimize the users sleep experience.

1.2 Problem statement

The project description above discussed the BEDsense project from the point of view of two applications:

1. at bedtime, connected devices in my home respond when I get in bed and/or fall asleep (e.g., arm security system, lock doors, turn off lights, adjust thermostat, etc.) and when I wake up or get out of bed (e.g., disarm security system, turn on lights, adjust thermostat, etc.),
2. my BEDsense provides information about me and my families sleep quality, which helps to improve our sleep experience and allows us to live healthier and happier lives.

For each application we can define separate problem statements. The first problem deals with the integration of the BEDsense into any Smart Home environment. Ideally it is universally plug and play (UPnP) and seamlessly communicating with the Smart Home system, however the device lacks a physical interface and has very limited resources in memory, computing power and communication bandwidth hence the problem statement is:

BEDsense has to be equipped with communication means for provisioning, networking and configuration that would allow near UPnP, optimum device performance for integration into Smart Home environments.

The second problem arises when the BEDsense is used for sleep analysis. The raw data derived from the sensor is not suitable for efficient storage and transmission. Hence, the data must be transformed to a format that is, unlike the raw data, compatible with the Smart Home controller and useful for further analysis.

Define a strategy for the data formats needed to communicate, store and analyze data from the BEDsense for local network use within the Smart Home and outside local networks to allow sleep analytics for deriving sleep quality.

1.2.1 Objectives

The first goal of this master's thesis is to investigate and compare different possibilities available on the market to integrate a Wi-Fi based sensor into Smart Home networks. Based on the results of this investigation a strategy to enable secure, flexible and robust integration of the BEDsense must be designed and in part implemented while taking the limited resources of the device into account. More specifically, a prototype will be developed that meets the following requirements:

- able to connect to set up Wi-Fi connection with the Smart Home network;
- automatic discovery of the device and it's embedded service(s);
- future proof, it must be possible to add functionality through Firmware Over-The-Air updates (FOTA);
- configurable and controllable through a complementary mobile application.

A second goal is to create a strategy whereby BEDsense data is transformed into sleep profiles and recommendations to improve their sleep. This involves the design of a system on how data is transmitted and stored in order to do analysis and eventually return recommendations to the user.

1.2.2 Approach

The project is divided into two major parts. The development of a prototype for Smart Home integration is worked out in the first part. Based on this prototype, a strategy for the sleep analysis application is described in the second part of this master's thesis.

Firstly, a research of all different methods for secure and reliable connectivity is completed. Based on the results of this research, the development and implementation of the chosen methods has started. Different types of customers may want to use different methods. Multiple methods must therefore be implemented into the system prototype to reach as much customers as possible.

When the specification and implementation of the prototype is finished, the development of the UI can start. The development of the UI tools facilitates testing and further development.

Subsequently, to reach the second goal, a strategy is designed and in part implemented to show the potential opportunities in the field of sleep analysis.

1.3 Outline

Chapter 2: System description
This first chapter, gives an overview of the BEDsense device. It contains a description of the three main parts of the BEDsense, namely: the external fiber optics, the microcontroller and the Wi-Fi module.

Chapter 3: Smart Home Integration
This chapter covers the specifications of the systems communication means needed for integration into Smart Home environments. Every aspect of the connectivity will be addressed in detail in the next subchapters. A comparison of different methods or approaches is made. At the end of each comparison, the decision of each aspect will be explained. Subsequently, the implementation of the chosen methodology is illustrated. Following parts are reviewed: Wi-Fi provisioning methods, Service discovery, FOTA and the mobile app. This section finally results in a complete description of the BEDsense prototype.

Chapter 4: Sleep analysis
The second application of the BEDsense - the sleep analysis - is divided in three phases. This chapter gives an overview of the three phases and contains a strategy to enable the first phase. This strategy is partially implemented to demonstrate the opportunities in this phase.

Chapter 5: Security
This chapter focuses on the security challenges of IoT-like devices. Firstly, the IoT security concerns and challenges are explained. Afterwards, it contains an overview of the security measures already implemented and the ones still to be taken in the future.

Chapter 6: Conclusion
This last chapter summarizes what has been achieved during this master's thesis. Additionally, this chapter discusses the results. Also the limitations and future work are reviewed in a critical way.

2 System description

The BEDsense electronics consist of two major modules, the MSP module (hereinafter called the optical engine) which runs the detection algorithms and the ESP-WROOM-02 Wi-Fi SOC module which provides the connectivity. The Digital Signal Processing (DSP), the detection algorithms, and the external fiber optics are in development at 2M Engineering. The BEDsense will communicate primarily with the Smart Home control system and a mobile application for provisioning, configuration and potential usage of the sensor data. Figure 1.1 shows the overview of the BEDsense system. The changes needed for the control panel will be made by the Smart Home providers. The development of the mobile application is part of this project.

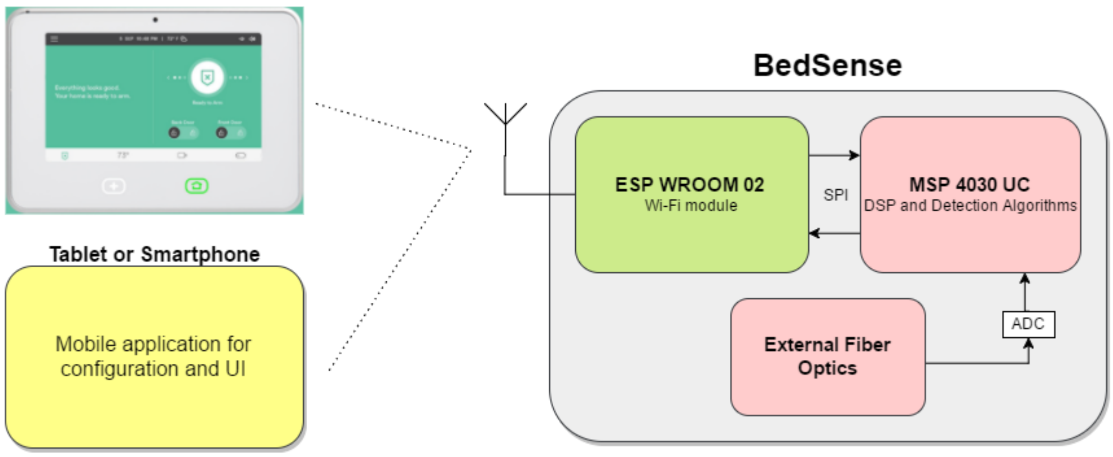


Figure 2.1: Overview of system

The BEDsense PCB is still under development, the version (4.08) used in this project is shown in Figure 2.2.

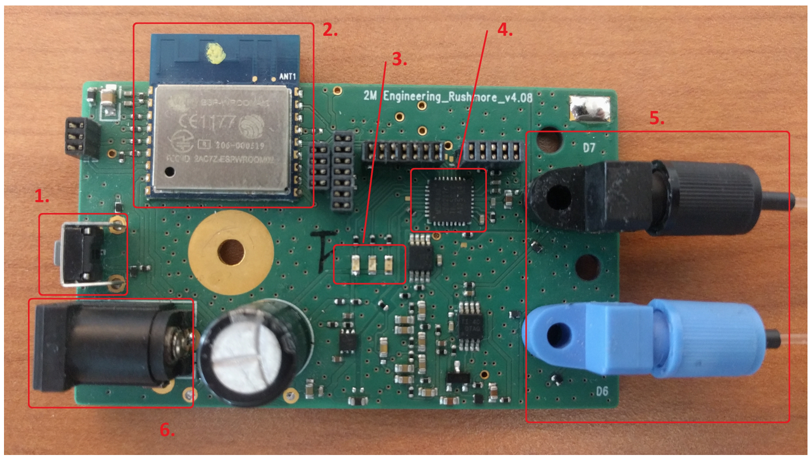


Figure 2.2: Top view of the BEDsense PCB

The highlighted components are the following:

1. push button,
2. ESP Wi-Fi Module,

3. LEDs,
4. MSP microcontroller,
5. optical fiber connectors,
6. power connector.

The next subchapters cover the different parts in more detail.

2.1.1 External fiber optics

A plastic optical fiber is composed of a thin plastic rod made from Polymethyl Methacrylate (PMMA), which is surrounded by a plastic protective cladding that has a different refractive index. The plastic rod contains two parts: the inner portion of the rod and the surrounding cladding. Light rays enter the fiber at different angles and do not follow the same paths. Light rays entering the center of the fiber core at very low angle will take a shorter path through the center of the fiber. Light rays entering the core at a high angle of incidence or near the outer edge of the fiber core will take longer paths. Each path resulting from difference of incidence will give rise to a mode [2]. Figure 2.3 shows the light propagation through the fiber.

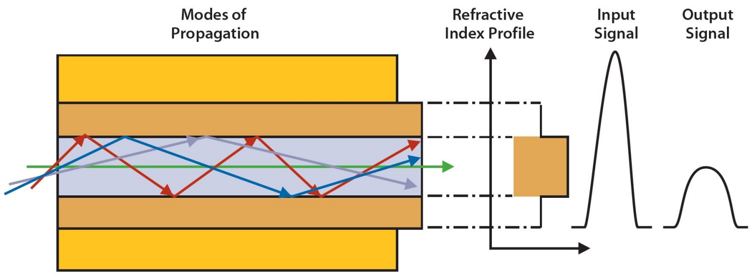


Figure 2.3: Light propagation through a step-index multimode fiber [2: p. 11]

Step-index multimode fiber guides light through total reflection on the boundary between the core and cladding. The refractive index is uniform in the core. Due to modal dispersion, step-index multimode fibers are suitable for sensing applications.

Microbending loss involves the accumulated effect of periodic displacements in an otherwise straight fiber. As the name may suggest, such displacements occur in the transverse direction of the fiber. Figure 2.4 illustrates the microbending effects due to pressure. The undesirable effects of microbending in communication systems are of particular interest for sensing perturbations close to the environment of the fiber. The periodic deformation of the fiber provides a mechanism by which power from guided modes of the fiber is coupled to adjacent modes or radiation modes, leading to power loss [2].

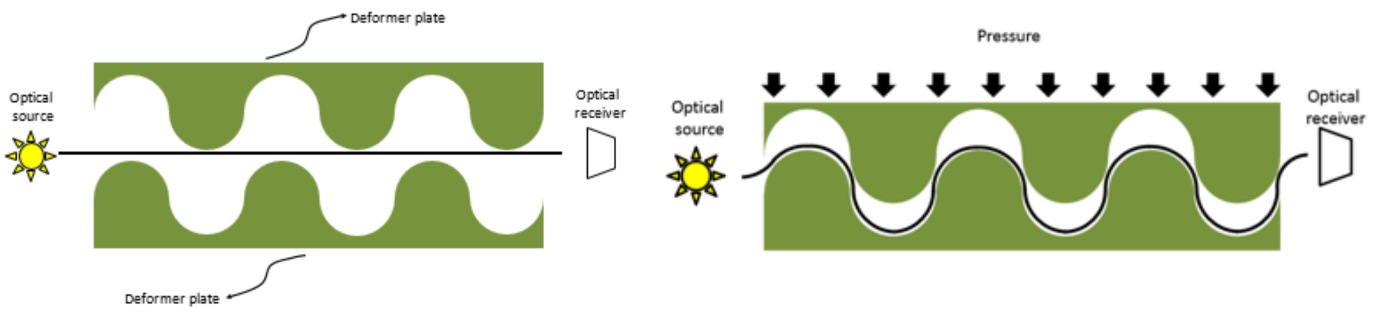


Figure 2.4: Unloaded and loaded configuration of the fiber optics

As described above the microbending effects can be used for the development of an intensity modulated pressure sensor. Lagakos et al. proposed a generic model based on microbending for the measurement of several environmental perturbations [3].

This mechanism seems to be an excellent low cost sensor solution for the contactless detection of vital signs. The combination of this sensor and the right algorithms results in a promising, low-cost system for sleep monitoring. The analog output signals of the fiber will be converted to a digital signal and be processed in the BEDsense.

2.1.2 BEDsense board

The BEDsense board contains the intelligence of the sensor system. It is connected to the optical fiber and controls the light that is sent through the fiber. The light travels through the fiber and returns to the BEDsense board where an optical receiver measures the amount of light. The output of the optical receiver is an analog signal. The analog input signal has to undergo a lot of steps before it can detect sleep. The Block diagram below gives an overview of how the signal is processed on the BEDsense board.

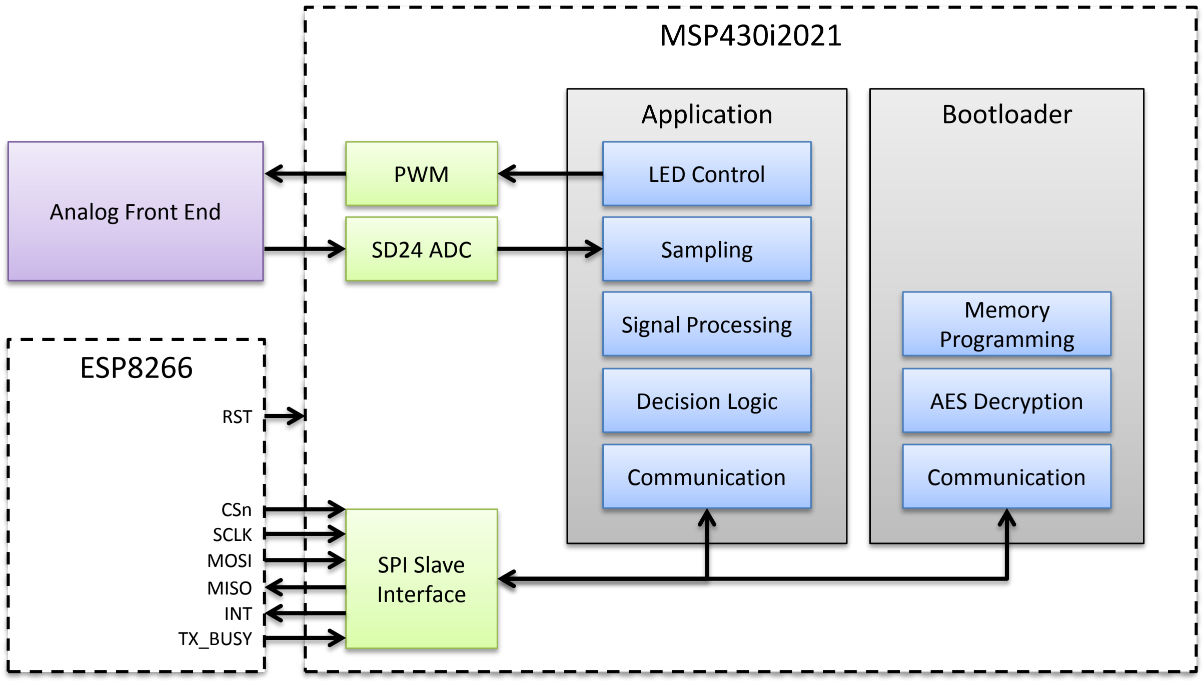


Figure 2.5: Block diagram of the BEDsense board

The analog output from the optical receiver requires amplification and conditioning to provide the best possible signal to the ADC. The Analog Front End (AFE) does this. The optical engine controls the AFE. The output of the AFE is converted to a digital signal and is sent to the embedded signal processing functions on the optical engine. The algorithms are able to convert the digital signals into a bed-occupancy status.

The output is sent through an SPI interface to the Wi-Fi module. The Wi-Fi module is an ESP-WROOM-02 low-power 32-bit MCU Wi-Fi module, based on the ESP8266 chip. The bed-occupancy status will be made accessible through the Wi-Fi communication.

3 Smart Home Integration

3.1 Wi-Fi provisioning

Wi-Fi provisioning is the process of connecting a new Wi-Fi device to a Wi-Fi network. It involves loading the device with the Service Set IDentifier (SSID) and its security credentials. Wi-Fi was created to allow nomadic devices such as laptops, smartphones and tablets, to wirelessly connect to the Internet. These personal computing devices naturally include a touch screen or a display and keyboard to provide the UI. The usual procedure for provisioning a smartphone, for instance on a Wi-Fi network, is done via the phone's Wi-Fi setting page. The phone scans for Wi-Fi networks and presents a list of available networks to the user. After choosing the network, the user is prompted for a password. If the password is typed correctly, the provisioning is successful, and often indicated by a Wi-Fi symbol in a status bar [4].

The challenge for a BEDsense device is that it does not have a display or keyboard (although there is a very minimal interface scheme via a button and three LED's). The near ''headless device'' needs an alternate method to obtain the network name and password from the user. Various alternate methods are available and no provisioning method is perfect because they all come with their merits and challenges.

The predominant Wi-Fi provisioning methods for headless devices in the market are compared to each other. The following aspects are important in this comparison: ease of use, security, robustness and flexibility.

The next few paragraphs provide a detailed overview of the most used provisioning methods in the market. Later the key considerations are discussed for choosing the right provisioning methods for the BEDsense.

3.1.1 Wi-Fi Protected Setup

Wi-Fi Protected Setup (WPS) was introduced by the Wi-Fi Alliance in 2006 as an easy and secure method to provision devices without knowing the network name and without having to type long passwords. The standard defines two mandatory methods for WPS-enabled Acces Points (APs): Personal Identification Number (PIN) method and Push-Button-Connect (PBC) method. In both PIN and PBC methods, the AP and the provisioned device exchange a series of messages to establish a temporary connection that is used to deliver the credentials from the AP to the device.

In the PIN method, an 8-digit PIN is printed on a sticker on either the Access Point or the provisioned device (Figure 3.1). The user needs to read the PIN from one device and type it using a keypad on the other device. Since APs do not have keypads, the PIN is usually printed on the AP, and typed by the user at the provisioned device. The obvious drawback is that it doesn't work for headless devices – it requires at least a numerical keypad to type the PIN.

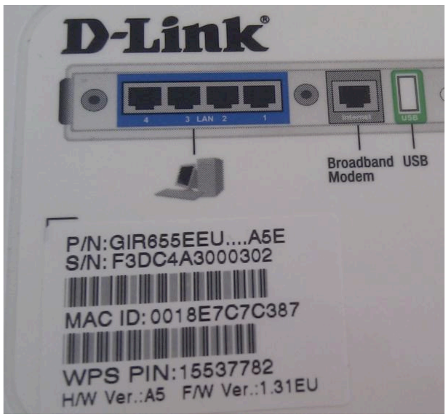


Figure 3.1: Label with WPS PIN on the back of a D-Link router [5: p.4]

In the PBC method, the user pushes a button on both the AP and the unprovisioned device. Once the button on the AP is pushed, WPS-enabled devices can openly join the network within two minutes. The drawback of this method, beyond the lack of security during the two-minute period, is that the user must have physical access to both the AP and the unprovisioned device.

A major problem in WPS was unveiled in 2011 [5]. A design flaw was found in the PIN method that allows brute-force attack to expose the network password in less than four hours. Since the PIN method is mandatory to achieve WPS certification, all new APs released to the market starting 2007 supported this method by default.

Right after the security breach was discovered, most AP vendors came out with recommendations to disable WPS, and although most of them released product upgrades that prevented hacking, WPS received a bad reputation in the industry and is still avoided in some countries.

To conclude, WPS only needs one push on a button, but a design flaw allows brute-force attack to expose the network password. The extra point in ease of use takes down the security score.

3.1.2 Access-Point mode

Another common provisioning method for headless devices is the Access Point (AP) Mode. Before trying to connect to the home network, the unprovisioned device wakes up initially as an AP, allowing a PC or smartphone to connect to it directly. The device also includes an embedded web server. After a smartphone or pc is connected the device's AP, the user opens a web browser and browses into the device's website via a predefined local URL or IP address. On this website, the user can enter the home network's SSID and password. The device stores the SSID and password in nonvolatile memory and then switches from AP mode to station mode. It will try to connect to the network with the stored credentials.

The primary benefit of AP mode provisioning is that it uses standard capabilities that exist in any smartphone, tablet and PC. Another benefit is that the vendors can add additional parameters to the embedded website to configure other device functions at the same time when the device is provisioned on the Wi-Fi network.

A disadvantage of the AP mode is that the phone gets disconnected from the home network when connecting to the configuration AP network of the unprovisioned device. Another drawback, when using a PC that both has ethernet and Wi-Fi internet connection, the PC may prioritize the ethernet connection and may not connect to the device's AP through Wi-Fi.

Furthermore, recently released smartphones often check whether the Wi-Fi network is actually connected to the Internet. If not (like it would when the phone is connected to the AP of the unprovisioned device), the smartphone disconnects from the Wi-Fi network and then forces a cellular data connection.

3.1.3 ESP-touch

Espressif's ESP-TOUCH protocol implements SmartConfig technology to help users connect ESP8266EX-embedded devices to a Wi-Fi network through simple configuration on a smartphone [6]. Figure 3.2 shows how the Wi-Fi credentials of the AP are sent to the embedded device through the ESP-TOUCH app running on a smartphone.

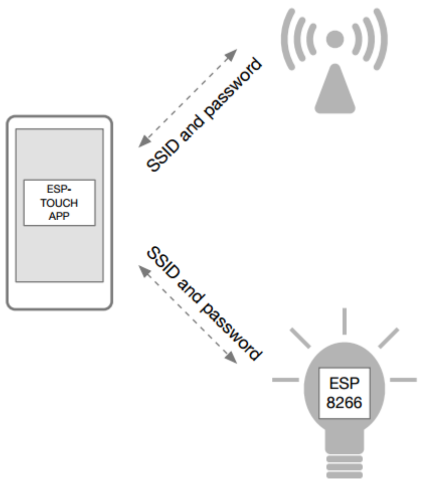


Figure 3.2: Typical ESP-TOUCH application [6: p.1]

Since the BEDsense (which is an ESP8266 embedded device) is not connected to the network at the beginning, the ESP-TOUCH application cannot send the information to the device directly. With the ESP-TOUCH communication protocol, a Wi-Fi enabled device such as a smartphone sends User Datagram Protocol (UDP) packets to the Wi-Fi Access Point (AP), and encodes the SSID and password into the Length field of a sequence of UDP packets where the ESP8266 device can reach and decode the information.

The two key benefits of ESP-TOUCH are ease of use and the potential to integrate seamlessly into the phone app of the product. Another unique capability of the SmartConfig technology is the ability to provision multiple devices simultaneously. If multiple Wi-Fi devices are in SmartConfig mode at the same time, one smartphone can provision all of them at the same time.

Its main drawback is the fact that the phone needs to be connected to a network using a frequency band and data rate that is supported by the unprovisioned device. For example, if the unprovisioned device supports only the 2.4GHz band, and the phone is using the 5GHz band to communicate with a dual-band network, then SmartConfig will not work, simply because the unprovisioned device is not listening on the 5GHz band. Some new routers and phones are using proprietary data rates to increase throughput, which makes ESP-TOUCH not always possible.

3.1.4 Out-of-band provisioning

The provisioning methods mentioned so far use the Wi-Fi radio to deliver the network credentials to the unprovisioned device. Thus, they don't require additional network interfaces to perform provisioning.

However, it is also possible to use a non-Wi-Fi medium to deliver the network credentials to the un-provisioned device, the so-called out-of-band provisioning methods. They can be wireless, for example using near field communication or Bluetooth; or wired, using a USB interface for instance.

Adding an out-of-band provisioning method improves robustness and flexibility, but increases the total cost of the device.

3.1.5 Comparison and decision

So far, different Wi-Fi provisioning methods are reviewed and discussed with their advantages and weaknesses. Table 1 shows the main pros and cons of the leading Wi-Fi provisioning methods.

Table 1: Comparison of the leading Wi-Fi provisioning methods

	WPS	**AP-mode**	**ESP-TOUCH**	**Out-of-band**
Advantages	● two modes: PIN and PBC ● only needs two pushes on a button	● uses standard capabilities ● can be nicely integrated into the control framework	● seamless integration into the mobile app ● capable of provisioning multiple devices at the same time	● many different kinds possible ● both wireless and wired possible
Disadvantages	• security flaws in PIN mode -> bad reputation • device must be reachable in PBC mode	● phone/pc gets disconnected from home network ● phone/pc may prioritize other network	● phone needs to be connected to the network using the same frequency band ● may not work on some routers/phones	● increases the total cost ● is not widely used

Based on the forgoing comparison, a design consideration was made. It has been decided to implement both WPS and AP-mode as provisioning methods for the BEDsense.

Especially ease of use is a critical aspect because provisioning is the first thing users do when they open the product's box. The provisioning procedure can shape the entire opinion about the product. In considering ease of use, WPS PBC is preferred because it requires no networking knowledge or tools and takes the least number of steps (only press two buttons).

WPS does require physical access to both the Wi-Fi router and the BEDsense to push the WPS button, which is not always obvious. Also, many of the APs have disabled WPS because of the security breach in the PIN method that was discussed earlier. Since the necessity of physical access and the limitation that not all APs support WPS, it might not always be possible to provision the BEDsense using WPS. Therefore, a second method is chosen in order to have a fallback method.

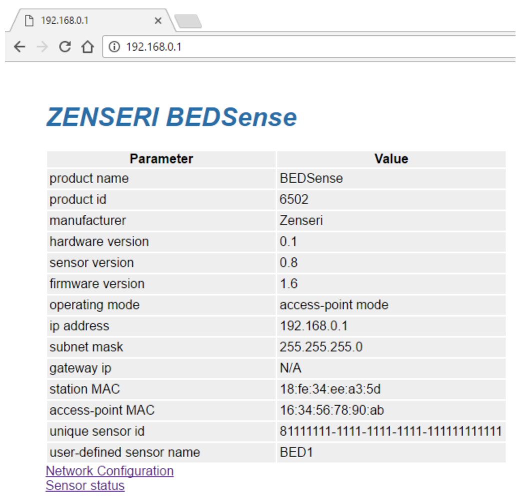
Because AP mode is probably the most ubiquitous Wi-Fi provisioning method and works in most cases, it is added as an alternative option. The standard capabilities that are used in AP modes make it possible to integrate this mode into a mobile application. This can be considered as a third provisioning method, but essentially the mobile application uses the AP-mode. By doing this, a uniform user experience is provided in the mobile application.

3.1.6 Implementation

AP-mode

When a new BEDsense device comes out of the box, it will start up as an AP. The red LED on the BEDsense is continuously on. The default SSID is "ZENSERI-BEDS-XXXX" where X represents a digit. The password will be made available for the user inside the box.

The user can use a computer, tablet or smartphone to connect to the BEDsense AP. The BEDsense includes an embedded web server. The user opens a web browser and browses into the device's website through the local IP-address (192.168.0.1). The website contains a list of information about the BEDsense (Figure 3.3 left). Under this list is a link to go to the network configuration page. All network parameters are listed on this page. The user can change some of those parameters. The interesting parameters to change are the station password, station SSID and operating mode.

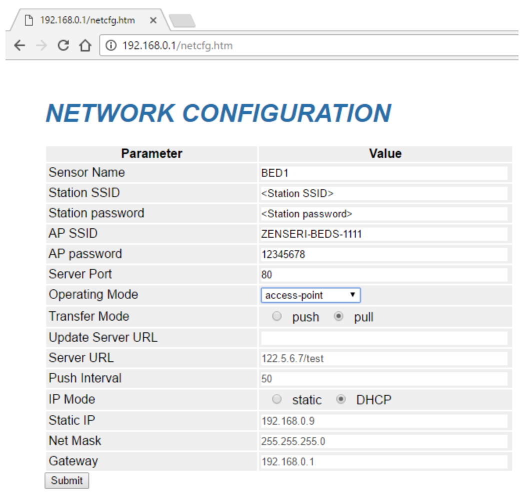


192.168.0.1

ZENSERI BEDSense

Parameter	Value
product name	BEDSense
product id	6502
manufacturer	Zenseri
hardware version	0.1
sensor version	0.8
firmware version	1.6
operating mode	access-point mode
ip address	192.168.0.1
subnet mask	255.255.255.0
gateway ip	N/A
station MAC	18:fe:34:ee:a3:5d
access-point MAC	16:34:56:78:90:ab
unique sensor id	81111111-1111-1111-1111-111111111111
user-defined sensor name	BED1

Network Configuration
Sensor status



192.168.0.1/netcfg.htm

NETWORK CONFIGURATION

Parameter	Value
Sensor Name	BED1
Station SSID	<Station SSID>
Station password	<Station password>
AP SSID	ZENSERI-BEDS-1111
AP password	12345678
Server Port	80
Operating Mode	access-point
Transfer Mode	push [✓ pull]
Update Server URL	
Server URL	122.5.6.7/test
Push Interval	50
IP Mode	static [✓ DHCP]
Static IP	192.168.0.9
Net Mask	255.255.255.0
Gateway	192.168.0.1

Figure 3.3: BEDsense's website: start page (left) and network settings (right)

When the correct SSID and password are entered and the operating mode is changed to 'station mode', the user pushes the Submit button. The BEDsense will reboot with the new parameters. If the parameters were filled in correctly, the provisioning is finished.

The AP-method can also be integrated in a mobile application; this is explained in more depth in section 3.5 where the development of the app is discussed.

WPS-mode

No matter in what operating mode the BEDsense is (station mode or AP-mode), the user can start the WPS provisioning with a short press on the button. The device will start seeking to join a network. A blinking blue LED shows this. The user has to push the WPS button on the wireless access point. When both the AP and the BEDsense recognize each other, the secure setup is initiated. The user is no longer involved in setting a password.

When two minutes are passed without having found an AP, the BEDsense ends the search.

3.2 Service Discovery

The device has a service to offer (the BEDsense data). An application will collect this data from the device. The application and the device have to discover eachother through some mechanism. Devices need to make themselves visible for others, but not too visible since this can cause serious security problems. As a result, the BEDsense needs to be seen and secure.

Smart Home networks often use the UPnP standards for communication between network devices. Smart Home systems already have an extensive number of integrated devices like IP-cameras, wireless sensors, Amazon Echo [7] or Nest Thermostat [8]. A major requirement of the Smart Home market is that the devices must be universally Plug and Play in being discovered. Universal Plug and Play (UPnP) described by the UPnP Forum as follows [9: p. 1]:

> UPnP™ technology defines an architecture for pervasive peer-to-peer network connectivity of intelligent appliances, wireless devices, and PCs of all form factors. It is designed to bring easy-to-use, flexible, standards-based connectivity to ad-hoc or unmanaged networks whether in the home, in a small business, public spaces, or attached to the Internet. UPnP technology provides a distributed, open networking architecture that leverages TCP/IP and Web technologies to enable seamless proximity networking in addition to control and data transfer among networked devices.

The BEDsense device will be developed according to the UPnP device architecture [9]. By following this architecture, the user will be able to access the BEDsense services with minimum manual configuration. The next paragraphs focus on different aspects of UPnP networking and how it is applied to the BEDsense.

Because of memory limitations, it is impossible to implement the full UPnP architecture. Only a part of the protocols can be implemented. Therefore the SSDP protocol is adjusted in order to get sort of a lightweight version. The two main parts that are implemented deal with the discovery and the description of the BEDsense. Both of them are explained below.

3.2.1 Discovery

One part of the UPnP networking is the discovery of services. When a UPnP device appears in the network, it advertises its presence to control points via the Simple Service Discovery Protocol (SSDP). Basic information about the device and its services is thus made known throughout the network. Similarly, when a control point is added to the network, SSDP allows the control point to search for devices of interest on the network [10]. Hence two types of SSDP messages will be sent. On the one hand search messages, when an SSDP client is looking for SSDP services. On the other hand, advertisement messages, when an SSDP service announces its presence. These two types of discovery messages are the fundamental exchange. They contain few, essential specifics about the device or its services, e.g. its type, Universal Unique IDentifier (UUID), a Uniform Resource Locator (URL) to more detailed information. Figure 3.4 shows this discovery architecture in a schematic way.

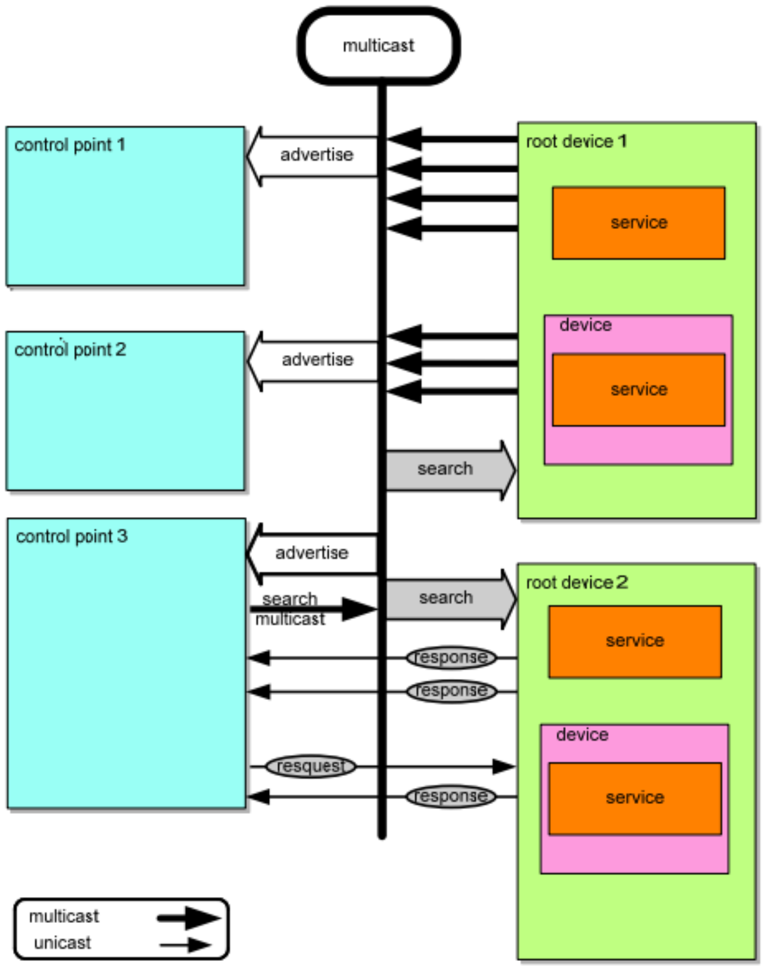


Figure 3.4: Discovery architecture [9: p. 13]

Advertisement messages

When a device is newly added to the network, it must multicast several discovery messages to a standard address and port (239.255.255.250:1900), advertising itself and its services. Control points listen to this port to detect new capabilities on the network. When we apply this architecture, the BEDsense device acts like a root device. Each BEDsense device embeds a service from which the BEDsense status can be queried. The control points are the devices that are able to use the BEDsense. These can be different types of devices such as a smartphone or a control panel attached to a wall in a Smart Home.

There are three types of advertisement messages:

1. SSDP: alive, when a device is newly added to a network;
2. SSDP: update, sent for periodic updates;
3. SSDP: byebye, when a device is removed from a network.

The BEDsense must use these three types of messages as the UPnP architecture prescribes.

Search messages

The SSDP message formats must be implemented as defined in the UPnP device architecture [9: pp. 12-36]. The pattern of a multicast M-SEARCH looks like this:

```
M-SEARCH * HTTP/1.1
```

```
HOST: 239.255.255.250:1900
MAN: "ssdp:discover"
MX: seconds to delay response
ST: search target
USER-AGENT: OS/version UPnP/1.1 product/version
```

Where the values in italics are placeholders for actual values. A response to the M-SEARCH request has the following pattern:

```
HTTP/1.1 200 OK
CACHE-CONTROL: max-age = seconds until advertisement expires
DATE: when response was generated
EXT:
LOCATION: URL for UPnP description for root device
SERVER: OS/version UPnP/1.1 product/version
ST: search target
composite identifier for the advertisement
USN: BOOTID.UPNP.ORG: number increased each time device sends an initial
   announce or an update message
CONFIGID.UPNP.ORG: number used for caching description information
   SEARCHPORT.UPNP.ORG: number identifies port on which device responds to unicast M-SEARCH
```

The discovery enables the next step in UPnP networking, i.e. description.

3.2.2 Description

After a control point has discovered BEDsense devices, it still knows very little about the device. The next step is to retrieve the device's description from the URL provided by the SSDP discovery message.

The description is partitioned into two logical parts: a device description and a service description. A device description describes the physical and logical containers including vendor-specific information. Each service description includes a list of actions, to which the service responds, and parameters for each command. The description process is visualized in Figure 3.5.

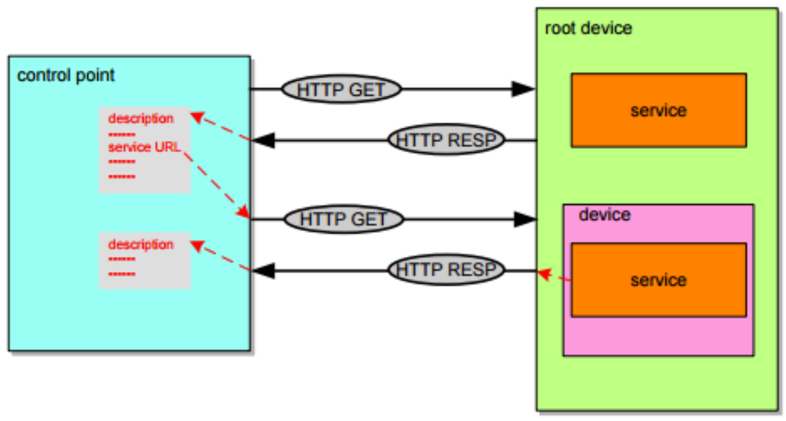


Figure 3.5: SSDP description architecture [9: p. 37]

In a Smart Home environment, there can be many control points such as a controller attached to a wall, a tablet or a smartphone. The BEDsense will act as a root device and offers one service, a sensor status service. Control points can query this service. The responses contain the status of the sensor.

The device and service descriptions are written in XML syntax based on the standard UPnP device template [9: pp. 43-55]. The service description includes a list of commands to which the service responds and a list of variables that model the state of the service.

3.2.3 Implementation

After provisioning, the BEDsense device joins the Wi-Fi network of the Smart Home. After obtaining the IP address, it will advertise itself according to the UPnP standards. It starts broadcasting several SSDP advertisement messages. Other devices on the network supporting UPnP listen to these broadcast messages to discover BEDsense devices and their associated URLs in the network.

The logical hierarchy for the BEDsense device and its embedded services is shown in Figure 3.6. Basically, each BEDsense device consists of a root device with a unique 128-bit UUID and a corresponding URL to obtain all the device information. Furthermore, each BEDsense root device embeds a service from which sensor status can be queried.

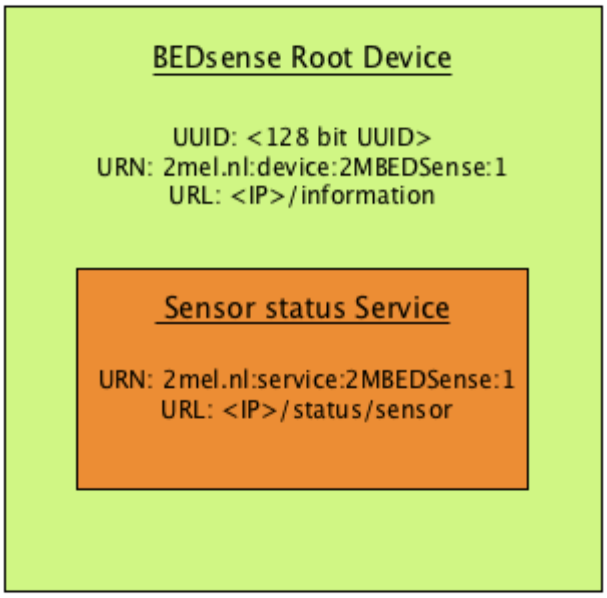


Figure 3.6: Hierarchy of the BEDsense as a UPnP root device

Discovery

As specified in section 3.2.1 the BEDsense needs to send three different types of advertisement messages as prescribed in the UPnP device architecture. The first type is the 'SSDP: alive' message. The BEDsense multicasts several discovery messages when it is newly added to a network. Such a message has the following format:

```
NOTIFY * HTTP/1.1
Host: 239.255.255.250:1900
CACHE-CONTROL: max-age=1800
LOCATION: http://192.168.1.2:80/description.xml
NT: upnp:rootdevice
NTS: ssdp:alive
SERVER: lwIP/1.40 UPnP/1.0 BEDSense/1.0
USN: uuid:11111111-1111-1111-1111-111111111111::upnp:rootdevice
BOOTID.UPNP.ORG: 44
NEXTBOOTID.UPNP.ORG: 45
```

```
CONFIGID.UPNP.ORG: 1
API.2mel.nl: 2MBEDSenseV1
```

The BEDsense will update its presence periodically using 'SSDP:update' messages. An example is given below:

```
NOTIFY * HTTP/1.1
Host: 239.255.255.250:1900
CACHE-CONTROL: max-age=1800
LOCATION: http://192.168.1.2:80/description.xml
NT: uuid:11111111-1111-1111-1111-111111111111
NTS: ssdp:update
SERVER: lwIP/1.40 UPnP/1.0 BEDSense/1.0
USN: uuid:11111111-1111-1111-1111-111111111111
BOOTID.UPNP.ORG: 44
NEXTBOOTID.UPNP.ORG: 45
CONFIGID.UPNP.ORG: 1
API.2mel.nl: 2MBEDSenseV1
```

Before leaving a network, the BEDsense sends an 'SSDP:byebye' message. If the device is removed abruptly, it might not be able to multicast the message. For this reason, discovery messages include an expiration value. The 'SSDP:byebye' message looks as follows:

```
NOTIFY * HTTP/1.1
Host: 239.255.255.250:1900
NT: upnp:rootdevice
NTS: ssdp:byebye
USN: uuid:11111111-1111-1111-1111-111111111111
BOOTID.UPNP.ORG: 44
CONFIGID.UPNP.ORG: 1
```

Description

When discovery is completed, the control point still knows little about the BEDsense device. Therefore, the next step is the device description. The discovery messages contain the location of a more detailed description. The UPnP device architecture prescribes the format of this description. The description is in XML syntax and contains several pieces of vendor-specific information, a URL for presentation of the device, etc. An example of a BEDsense device description is given below:

```xml
<root xmlns="urn:2mel.nl:device:2MBEDSense-1-0">
    <specVersion>
        <major>2</major>
        <minor>0</minor>
    </specVersion>
    <URLBase>http://10.3.221.107:80</URLBase>
    <device>
        <deviceType>urn:2mel.nl:device:2MBEDSense:device:1</deviceType>
        <presentationURL>http://10.3.221.107:80</presentationURL>
        <friendlyName>Bed Occupancy Sensor</friendlyName>
        <manufacturer>Zenseri</manufacturer>
        <manufacturerURL>https://www.2mel.nl/</manufacturerURL>
        <modelDescription>Bed Occupancy Sensor</modelDescription>
        <modelName>BEDSense</modelName>
        <modelNumber>0.1</modelNumber>
        <modelURL>https://www.2mel.nl/products/</modelURL>
        <serialNumber>6502</serialNumber>
        <UDN>uuid:22222222-2222-2222-2222-222222222222</UDN>
    </device>
</root>
```

By following the prescribed syntax, other UPnP devices will discover the BEDsense automatically and can read the description. A Windows PC, for example, will show the BEDsense in the list of network devices as shown in Figure 3.7.

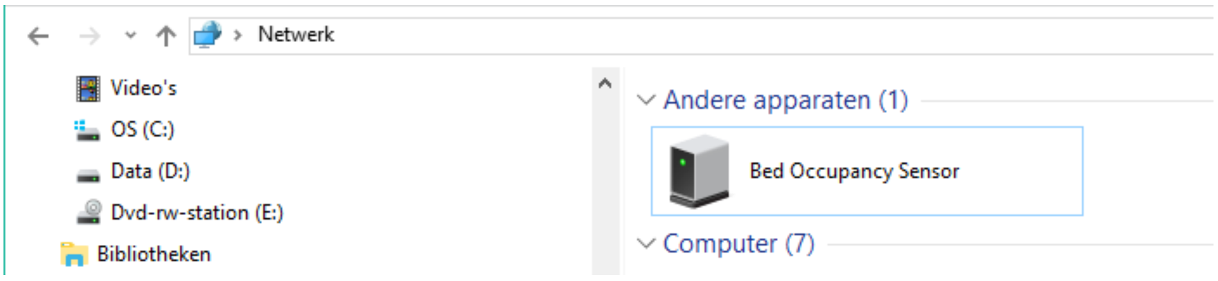


Figure 3.7: A discovered BEDsense in the list of network devices in Windows File Explorer

3.3 Communication protocol

Once a BEDsense device is registered in a Wi-Fi network (Provisioning), has obtained an IP address and has been discovered (Service Discovery), it is possible for other network devices to communicate with it.

A BEDsense device has a Wi-Fi interface and responds to commands via a REST API. Data is sent in JSON format using either HTTP GET or POST methods. Basic functionality is accessible with this mechanism for example

1. Query the BEDsense status

Sensor status is queried by requesting (GET) data from the URL http://<BEDsenseIP>/status/sensor. The JSON reply from the device looks like this:

```
{
  "occupancy status": "EMPTY",
  "occupancy-status timestamp": "1194",
  "motion status": "LARGE MOTION",
  "motion-status timestamp": "1192",
  "pressure value": "8421753",
  "temperature value": "22",
  "pressure status": "NO ERROR",
  "temperature status": "NO ERROR",
  "sensor-operating mode": ""
}
```

The important part is the occupancy status. It can have the following states:

- UNDEFINED: the system is unable to detect a definite state;
- EMPTY: no human is detected on the bed;
- AWAKE: a human is detected and is awake;
- ASLEEP: a human is detected and is asleep.

2. Query or change the network settings

To query the network settings of a BEDsense a HTTP GET request to "http://<BEDsense IPaddress>/configuration/network" will be answered with a JSON packet like:

```
{

  "operating mode": "station",
  "station ssid": "<SSID>",
  "station password": "<PASSWORD>",
  "access-point ssid": "ZENSERI-BEDS-2222",
  "access-point password": "<PASSWORD>",
  "ip method": "DHCP",
  "static ip": "192.168.0.9",
  "subnet mask": "255.255.255.0",
  "gateway ip": "192.168.0.1",
  "server port": "80",
  "sensor name": "BED2",
  "data-transfer method": "pull",
  "push interval": "50",
  "push-server url": "122.5.6.7/test",
  "update-server url": ""
}
```

It is also possible to send an HTTP POST with the above JSON data to change the network settings. The answer from the device will again be a JSON packet with all possible labels and a value indicating if the new parameter was accepted or not.

3. Reset the device

In order to reset the device a HTTP POST request can be done to http://<IPaddress>/action with JSON data of:

```
{
"action" : "reboot"
}
```

After changing the network parameters, a reboot must be performed.

3.4 Over the air firmware updates

Firmware Over-the-Air (FOTA) technology makes it possible to deliver updated firmware to mobile or embedded devices. In order to make the BEDsense robust and durable for the future, it is required to do updates from time to time. Also, there will always be inevitable issues with the BEDsense firmware such as: software defects, missing features, design issues and security issues. Being able to resolve these issues with a firmware update can save a lot of time and money. It is therefore an opportunity to enhance product functionality, operational features and to provide fixes for particular problems. This makes FOTA highly desirable for the BEDsense.

Since the updates will be performed over the air, the Wi-Fi module has to provide the mechanisms for doing this. The mechanisms should allow updates for his own firmware as well as updates for the optical engine as shown in Figure 3.8.

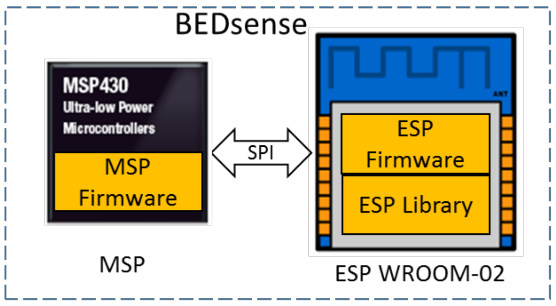


Figure 3.8: BEDsense firmware overview

The ESP module will receive the binary files for both modules. It will check the version of the firmware first. When the existing firmware has a lower version than the new firmware, it will start writing the new firmware images. The ESP module has to write the MSP firmware through the SPI interface

3.4.1 Push or pull updates

One can identify two different firmware update modes – externally pushed firmware and another one that is initiated by the remote device. Figure 3.9 and Figure 3.10 show a flow diagram of the procedures in both modes.

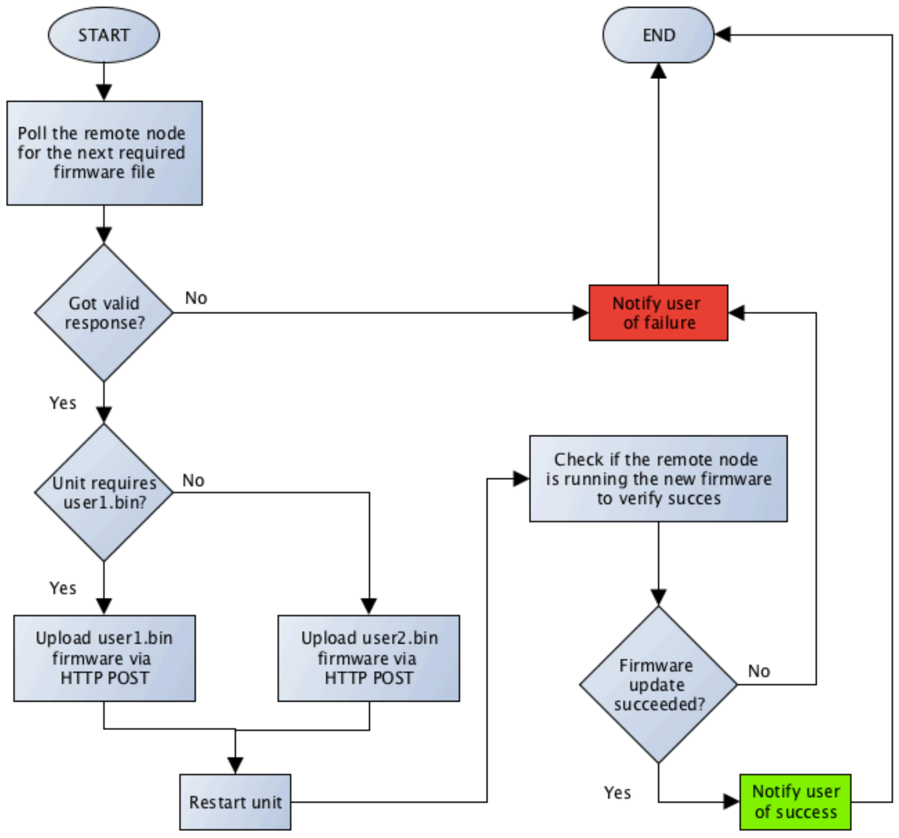

START
Poll the remote node for the next required firmware file
Got valid response?
No
Yes
Unit requires user1.bin?
No
Yes
Upload user1.bin firmware via HTTP POST
Upload user2.bin firmware via HTTP POST
Restart unit
Check if the remote node is running the new firmware to verify succes
Firmware update succeeded?
No
Yes
Notify user of failure
Notify user of success
END

Figure 3.9: FOTA push mode flowchart

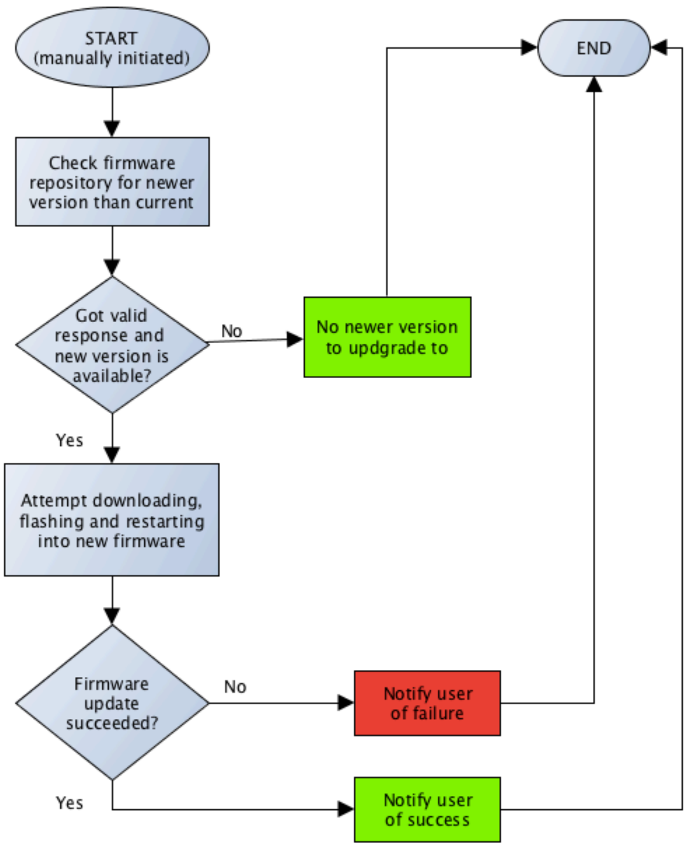

START (manually initiated)
Check firmware repository for newer version than current
Got valid response and new version is available?
No
No newer version to updgrade to
Yes
Attempt downloading, flashing and restarting into new firmware
Firmware update succeeded?
No
Notify user of failure
Yes
Notify user of success
END

Figure 3.10: FOTA pull mode flowchart

Externally pushed firmware update would be when the device receives the firmware file over a HTTP POST request, whereas in the pulled firmware update mode, the device checks the server for firmware updates.

In our case the BEDsense will be integrated into the Smart Home environment where it is connected to the local network. The firmware updates will be sent to the customer and the actual update will be completed using their system. The mode of upgrading will therefore be chosen in consultation with the Smart Home provider. In the development stage the updates will be done using the pull mode, so the FOTA will be initiated manually.

3.4.2 ESP firmware updates

The ESP Wi-Fi module is well positioned to allow FOTA. Espressif provides a guide on how to do FOTA updates on their Wi-Fi modules [11]. It is possible to update the firmware of the ESP module when the module is not connected to a PC. The available 2MB on the SPI flash memory is partitioned into two, what can be seen in Figure 3.11.

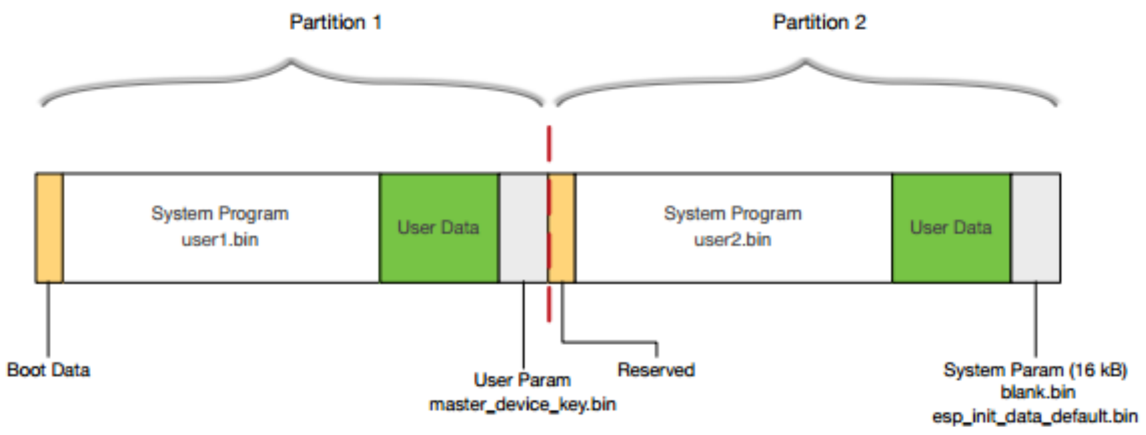


Figure 3.11: ESP8266 FOTA flash memory layout

The basic idea for the FOTA update procedure used in the ESP module is very simple: use only half the flash space with the firmware such that two partitions can be created. When the firmware is running out of one partition, the other partition can be loaded with a new version. A restart can make the ESP8266 run out of the new partition. At that point, the partition that is not used can be updated, and so forth. The user code runs on one or the other partition. This has the advantage that when something goes wrong during FOTA, the previous firmware can be recovered. The system can never be in a state where it is stuck or partially programmed. A drawback of this solution is that it limits the available memory space for the user code.

3.4.3 Optical engine updates

The optical engine has to be put in bootloader mode in order to upgrade the firmware. If the MSP is in bootloader mode, an upgrade register will be made available. This register represents the starting address for the upgrade image. All 15360 bytes (15Kb) of a new firmware image are written within a single SPI transaction, utilizing the auto address increment function.

After writing the last byte of the image, the BTL_STATUS register can be checked to read to verify a correct update procedure. When everything is ok, the value of this register reads 0xAB. When there is a wrong checksum the result will be 0xAA.

3.4.4 Implementation

When the BEDsense is officially released and used in Smart Home environments, the binaries are sent to the Smart Home providers when an update has to be performed. They will provide an interface that gives the user the possibility to initiate the FOTA update. The BEDsense is still in development, but it is also useful in the development phase to do FOTA updates. This allows us to add functionality to the prototype if needed. Hence, the tools that are needed to do FOTA are developed and can be sent to future customers along with the prototype.

The Windows desktop provides the tools to set up the FOTA procedure. Figure 3.12 illustrates the overview of the FOTA system used during the development of the BEDsense.

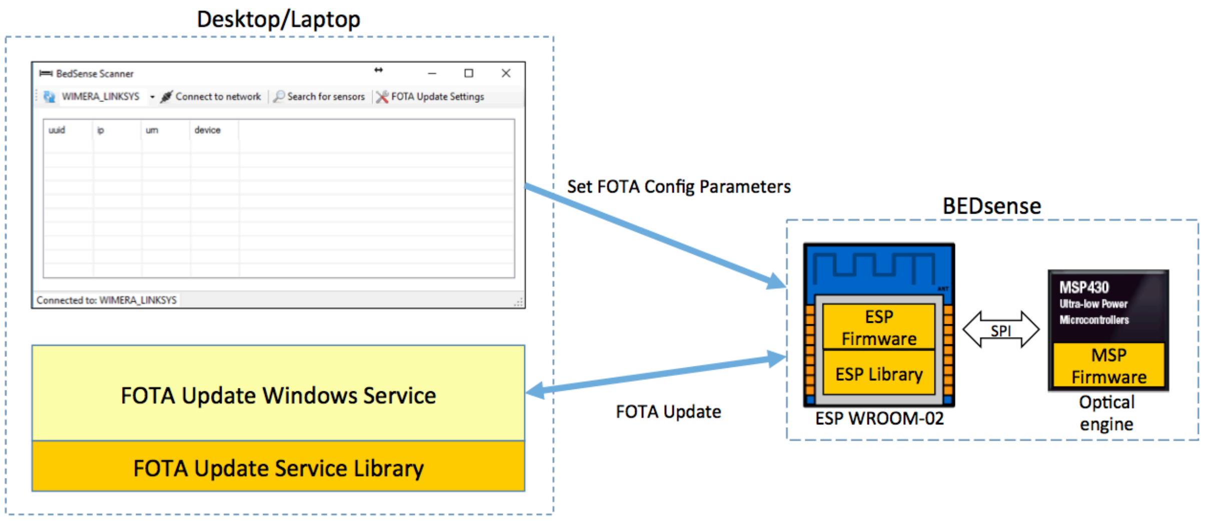


Figure 3.12: Overview of FOTA system used during development.

The mechanisms are partially included in the desktop tool. The desktop application will provide the UI to set the parameters for the FOTA such as selecting the binary file, setting the FOTA server address, etc. A Windows service will be used for the FOTA server. This service is installed together with the installation of the desktop tool.

Figure 3.13 shows the message flow of the FOTA procedure. It gives a good overview of how the FOTA works. There are clearly three main parts: the BEDsense device that has to be updated, the desktop tool provides the UI and the Windows service that acts like the FOTA update server.

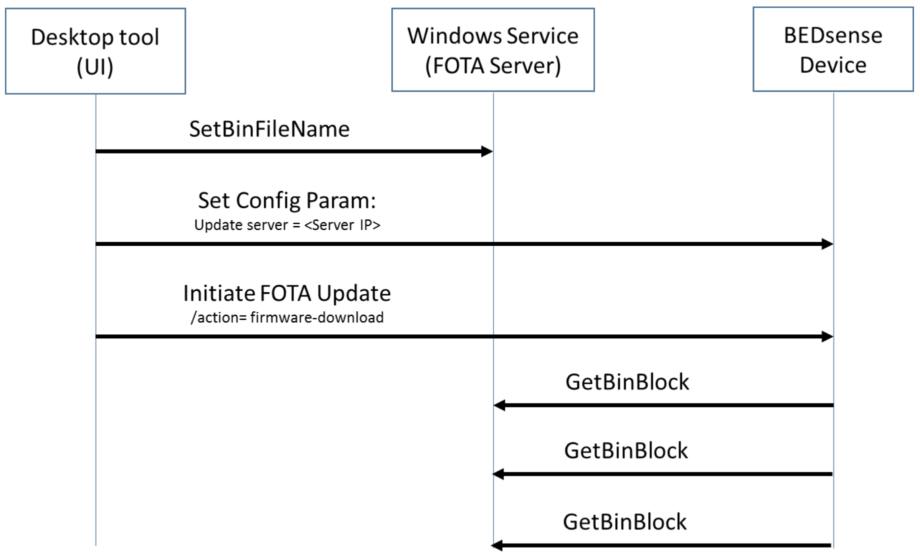


Figure 3.13: FOTA message flow

Desktop tool

In the specification is chosen for the pull-method to start the FOTA procedure. The initiation, therefore, must be done manually. A user who wants to initiate the FOTA update will only use the desktop tool as interface. The desktop tool will communicate with the FOTA server and the BEDsense. As shown in the message diagram in Figure 3.13, there are three calls from the desktop tool. The first one is a call to the FOTA server to tell the service where the firmware binary files are located and which block size will be used. The next step is to tell the BEDsense device the service URL. Finally, it gives the BEDsense the command to initiate the FOTA update.

FOTA server

The Windows service acts like a FOTA server. It houses the update files and exposes the API to access them. The URL is the following: http://<IPaddress>:8888/Zenseri/BedSense. The API calls are listed in Table 2.

Table 2: Description of the FOTA server API calls

Set binary filename	
URL	/binfile?filename1=<fullpath1>&filename2=<fullpath2>&blocksize=<size>
Method	POST
URL parameters	filename1=[path to binary for partition 1] filename2=[path to binary for partition 2] blocksize=[integer]
Description	Sets the binary files for partition 1 and 2 and the block size to be used
Get binary filename	
URL	/getbinfilename
Method	GET
URL parameters	
Description	Returns the name of the files set and the block size set. It also returns the size of the files in number of bytes

Get binary block	
URL	/binblock?blocknum=<bid>&partition=<pid
Method	GET

URL parameters	blocknum=[integer]
	partition=[integer] (1 or 2)

Description	Returns the block of data for the specified block number and partition
Download firmware	
URL	/action
Method	POST
Data parameters	"action":"firmware-download"
Description	Initiates the firmware download.

3.5 Mobile application

Because the BEDsense is a headless device, there must be alternative ways to make the BEDsense easy to use. One alternative way is to use a mobile application for Android smartphones, called "BEDsense App", to handle the communication between the user and the BEDsense device. There are several actions that the user can do: configuring a new BEDsense, changing the settings of an existing BEDsense, monitoring the status of his BEDsenses, etc.

3.5.1 Development strategy

First, a decision has to be made which operating system (OS) and development tools will be used for the development of the BEDsense App. Nowadays there are three types of strategies used in the development of applications for mobile devices [12].

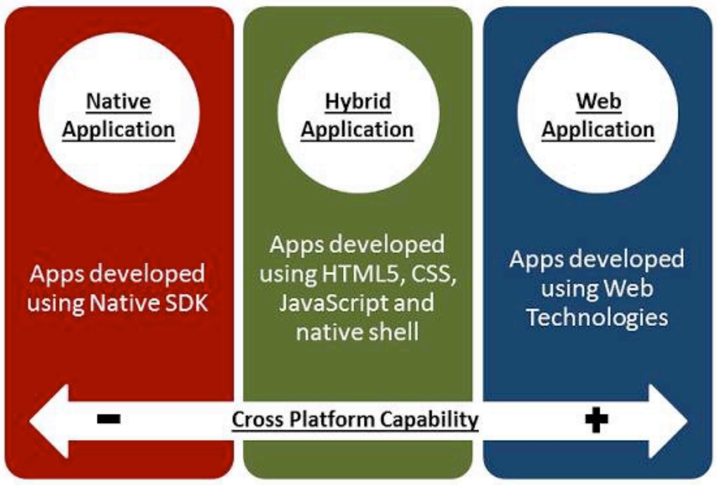

Native Application
Apps developed using Native SDK
Hybrid Application
Apps developed using HTML5, CSS, JavaScript and native shell
Web Application
Apps developed using Web Technologies
−
Cross Platform Capability
+

Figure 3.14: Strategies for mobile development [12: p. 33]

Based on different sources [12, 13, 14], the three strategies can be compared to each other as follows: native applications are developed using languages supported by the mobile OS technology stack. Web applications are in fact applications built on the web using the mobile device just as a front end. Hybrid mobile applications try to mix the best of both worlds; they use the power of server-side computing but don't treat the device only as a front end. They typically consist of a web portion that contains the elements that are shown to the user and a bridging mechanism that provides access to advanced features of the native platform.

It is soon decided not to use the strategy of the web applications, due to the many restrictions such as: no native API, no offline mode and no possibility to leverage the processing to the mobile device [12]. Therefore, only the native and the hybrid strategy are compared in more detail.

Native mobile application development

Native apps are built using only the tools and technologies (including programming languages) suggested by the mobile application stack vendors, such as Google (Android) [15] and Apple (iOS) [16]. Thus, it requires high level of specialized knowledge to develop a native application. Native apps are compiled into machine code, which gives the best performance you can get from the mobile phone. Best performance includes fluid animations, full access to the phone's hardware, multi touch support and the latest APIs [17].

The International Data Corporation (IDC) indicated the share of the smartphone market based on the number of devices sold [18]. The evolution of the market share is shown in Figure 3.15: Worldwide smartphone OS market share.

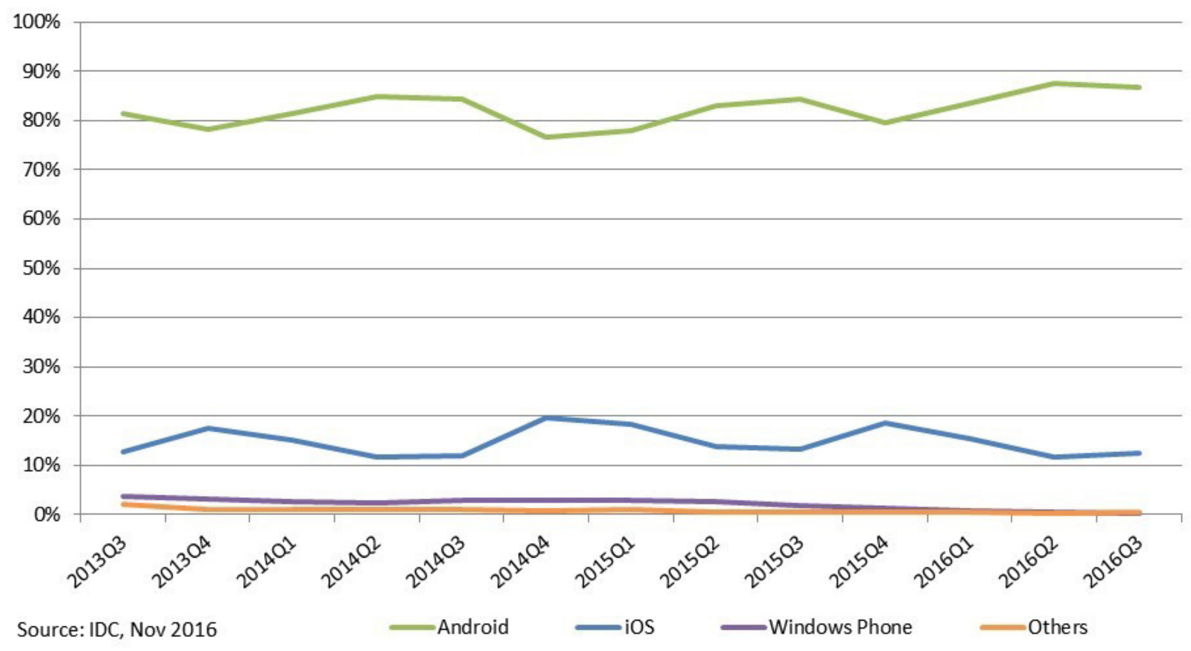


Figure 3.15: Worldwide smartphone OS market share

The worldwide smartphone OS market share has two major players. Android dominates the smartphone market with 86,8%, followed by iOS with 12,5% market share. The others continue to decline in market share, and will eventually almost disappear.

Hybrid mobile application development

Hybrid app development uses Web technologies such as HTML5, JavaScript and CSS that run inside the "Native Shell" of the mobile platform. Thanks to the increasing sophistication of multi-platform frameworks such as Apache Cordova/Phonegap [19], Appcelerator [20] or Xamarin [21], performance and user experience have improved greatly. According to Cygnet [22], all these frameworks solve the purpose of developing apps for multiple platforms. As a developer, you should choose the one that meets the requirements and purpose of your solution; one might be better than the other depending on the requirements.

Hybrid apps still depend on the native browser, which means that they are not as fast as native apps. On the other hand, development and maintenance is faster and therefore cheaper. The developer writes the code once and deploys it to different operational systems.

Comparison and decision

Having seen the differences between the native and hybrid development strategies, a decision has to be made for the development of the BEDsense app. Both strategies come with their strengths and weaknesses. Table 3 shows the main pros and cons of the two strategies.

Table 3: Pros and cons of native and hybrid mobile application development

	Native	**Hybrid**
Advantages	● best performance ● latest APIs	● platform independent development ● cost-effective
Disadvantages	● increased development time and costs ● content restrictions and guidelines based on the ecosystem	● limited device-specific feature-related APIs ● not suited for very high performance requirement

Based on this comparison and the fact that the smartphone market practically consists of two major platforms, i.e. Android and iOS, it has been decided to develop the BEDsense app natively. More precisely, an Android app will provide the user interface in this early development phase of the BEDsense. The app reaches 86% of the smartphone market. However, this might be an overestimation of the Smart Home market. Probably iPhone has a higher market share in that submarket. Therefore, In the future a native iOS app will be developed to reach the remaining smartphone users.

3.5.2 Provisioning a BEDsense

When determining the specifications of the connectivity (section 3.1), it was chosen to integrate AP-mode into the mobile application. Installers can choose to use WPS or the mobile app to connect unprovisioned BEDsenses to the home network. This section describes how the provisioning is integrated into the BEDsense App.

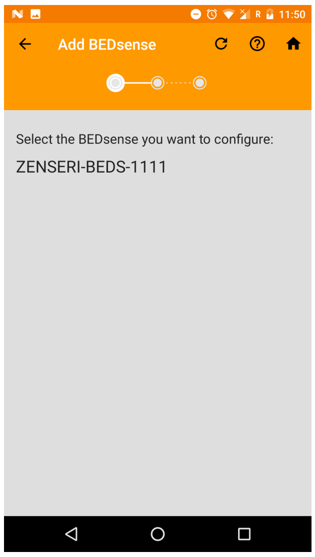
An unprovisioned BEDsense boots in AP-mode. The smartphone must first be connected to the BEDsense network. Secondly, the user must in some way be able to fill in the home network credentials and send it to the BEDsense. Finally, the BEDsense app should give the instruction to change his operating mode to station mode. These three steps are similarly integrated into the app.

1. Connect to the BEDsense AP

This part has the purpose of finding an unprovisioned BEDsense and setup a connection between the smartphone and the BEDsense.

Not only in this first step, but also in the next steps, the smartphone's Wi-Fi resources need to be addressed. So the first thing the app will do is check whether Wi-Fi is enabled. If not, it will make it enabled right away.

The app will then automatically search for unprovisioned BEDsenses in his neighborhood. It is possible to filter the BEDsenses out of all the Wi-Fi APs due to the hardcoded SSID of an unprovisioned BEDsense. The SSID is determined in the factory settings and has following format: "ZENSERI-BEDS-*XXXX*" (where *X* represents a digit). All available Wi-Fi APs containing "ZENSERI-BEDS" in their SSID will be listed on the screen, as seen in Figure 3.16. The user can continue to the next step by pushing on the SSID.



Select the BEDsense you want to configure:

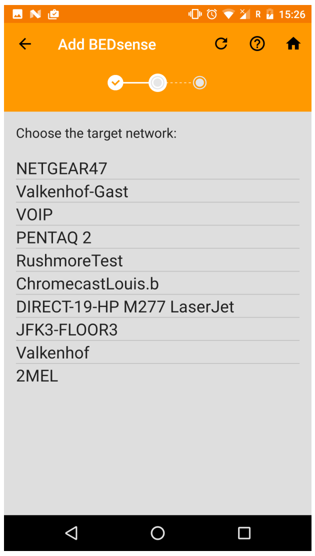
ZENSERI-BEDS-1111

Figure 3.16: List of available BEDsenses

2. Select the home network

The second step is to select the home Wi-Fi network. The app will automatically search for Wi-Fi networks and list them on the screen (Figure 3.17). The user only has to indicate the one that will be used to connect the BEDsense to. The major advantage of this manner is that the user doesn't have to type the SSID manually. When the desired network is not in the list, the user can search again. If the home network was found, this step only takes one push on the SSID to complete.

After this step, the app will make a connection to the BEDsense automatically and save the home network SSID so that it can be used for the next step.



Choose the target network:

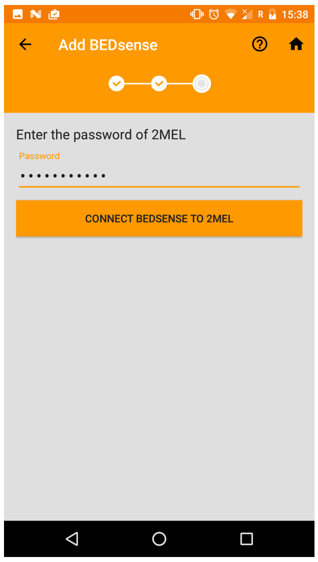
NETGEAR47
Valkenhof-Gast
VOIP
PENTAQ 2
RushmoreTest
ChromecastLouis.b
DIRECT-19-HP M277 LaserJet
JFK3-FLOOR3
Valkenhof
2MEL

Figure 3.17: List of networks

3. Enter the password of the home network

The only thing that has to be done to finish the provisioning is to enter the password of the home network and give the instruction to reboot. The user is simply prompted to type the password in a text field and push the button that says: “Connect BEDsense to <SSID of home network>” (Figure 3.18).

After pushing the button, the app will send the credentials of the home network to the BEDsense in the background. Finally, it will send the instruction to restart in station mode.



Enter the password of 2MEL

Figure 3.18: Enter the password

3.5.3 Control BEDsense(s)

After a BEDsense is provisioned successfully it is connected to a local network. Everyone that is connected to the same network can discover the presence of the BEDsense and communicate with each other. It is to say, if both devices communicate according to the rules of the SSDP protocol. The BEDsense has the SSDP Implemented, so the BEDsense app must have this implementation as well.

The first step to use a BEDsense is to make sure that the mobile phone is connected to the same Wi-Fi network as the BEDsense(s). This is prompted by the user via a dialog box. If this is not the case, the app sends the user to the Wi-Fi settings where he can make connection to the correct network.

While the phone is connected to the correct network, it can start the discovery of BEDsenses according to the SSDP rules. It multicasts an SSDP M-SEARCH request.

```
InetAddress group = InetAddress.getByName("239.255.255.250");
int port = 1900;
String query =
        "M-SEARCH * HTTP/1.1\r\n" +
                "Host: 239.255.255.250:1900\r\n"+
                "MAN: \"ssdp:discover\"\r\n"+
                "MX: 1\r\n"+
                "ST: urn:2mel-nl:device:2MBEDSense:1\r\n"+
                "\r\n";

socket = new DatagramSocket(port);
socket.setReuseAddress(true);
socket.setSoTimeout(2000);

DatagramPacket dgram = new DatagramPacket(query.getBytes(), query.length(),
        group, port);
socket.send(dgram);
```

BEDsenses that receive this message respond to this message with an SSDP response as discussed in section 3.2. The response includes a link to the description of the BEDsense in question. From this point, the BEDsenses are known for the app and it knows how to use them.

By using the HTTP GET and POST request as discussed in section 3.3, it can query the BEDsense status and change the network settings.

4 Sleep analysis

As mentioned in the project description, Zenseri does not want to limit the application of the BEDsense to the managed Smart Home market. Zenseri will sell BEDsenses directly to consumers in the future. The ultimate goal is to develop a system to do sleep diagnostics by combining all data from a large number of BEDsenses. This is not a short-term objective, Zenseri wants to take small steps to eventually reach the goal.

There will be three different phases in the evolution of the BEDsense with an increasing amount of complexity. In the first phase, only bed occupancy is measured. Secondly, also the movements of a human body will be measured. From this stage, doing sleep analysis is possible. The third and last phase will collect all data from many BEDsense devices and use it together to do holistic sleep diagnostics. More info needs to be derived and combined. In this stage, we are in the era of big data. Note that the goal is not to develop sleep diagnostics for medical purposes, but it will be used to give people advise for a better sleep lifestyle.

Phase 1: bed occupancy

The BEDsense will only be able to measure bed occupancy. This means that it can recognize when the bed is occupied and that it can distinguish whether it is a dead object or a living human being. Note that the BEDsense for the Smart Home customer is in this phase. It has three different values for bed occupancy: empty, present awake and present asleep. It is impossible to measure real sleep quality in this stage. Only the duration of sleep can be derived with this information. Although this is an important factor in a person's night quality, the BEDsense will not be used for sleep analysis.

Phase 2: body movements

Besides bed occupancy algorithms, new complex algorithms will be added to the BEDsense to come up with more information. This information will include the precise motion of the body parts. Different values of motion will be possible: small motion, medium motion and large motion. Small motion refers to breathing, medium motion is derived from movements of limbs, and large motion is caused by tossing and turning of the whole body.

Body movements are an important behavioral aspect of sleep because they can be associated to sleep states [23]. Both frequency and duration of body movements are important characteristics for sleep analysis [24]. During the last decades, actigraphy (activity-based monitoring) has become an important measuring tool in sleep research.

Different sensors are available on the market today that claim to be able to measure sleep quality. Typically, the actigraphy is obtained from a wrist device. However, the wrist device may affect the sleep quality due to the discomfort. Moreover, the measured movements are movements of the wrists only and don't always give enough information of body movements. It is therefore very challenging to measure movements of specific body parts.

An alternative way to measure body movements is to use a camera. The sleeping subjects are not attached to any equipment. Near infrared cameras can detect movements of the body very accurately, without making contact to the body. The disadvantages of this approach are the high cost of the camera and the privacy issues. People will feel watched when entering their bedroom.

BEDsense uses a totally new way of measuring body movements. It has the advantages of being non-contact based and being able to measure body movement in a very sensitive way. On top of that, it is practically invisible for the user because it is placed underneath the mattress.

Phase 3: big data

When phase 2 is evaluated, the next step is to combine all data and set up sleep profiles. Sleep profiles are created by combining the sleep quality measurements with extra personal information derived from the user via a mobile application. The data is analyzed and compared with the data from other users. Zenseri investigates possible improvements and shares them with the user via an application or website. By doing this, Zenseri teaches people how they can improve their sleep lifestyle.

Despite the fact that this last phase will not be developed in the near future, it is useful to keep the future plans in mind during the development of the first prototypes. This will reduce development time and costs in the future.

4.1 Transform raw data

The fiber-optics pressure sensor produces an analog signal. This signal is processed in the analog domain through the analog front end, converted to digital and further processed in the digital domain. A 24-bit ADC takes samples at 60Hz.

It may be clear that these samples are not suitable for storage or transmission. The BEDsense would run out of memory in a few hours and it would take a lot of power-consuming data transfer if it was transmitted.

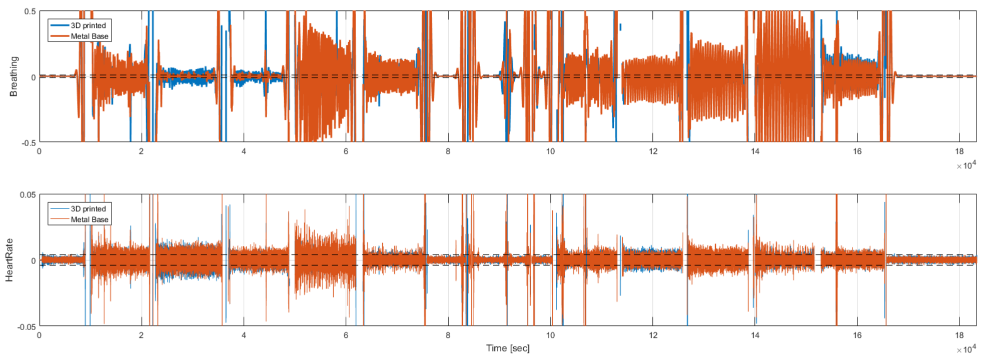


Figure 4.1: Visualization of raw data from the fiber-optics using Matlab

The optical engine therefore has some embedded algorithms to do real-time signal processing. The sensor system provides one main signal: the real-time status of the bed occupancy. The occupancy status can reflect four possible states:

- UNDEFINED,
- EMPTY,
- AWAKE,
- ASLEEP.

The real-time status of the bed occupancy can directly be applied in the Smart Home network to control heating, alarms, lighting or other devices. This is not the case for the second application, the sleep analysis. In order to enable reliable sleep analysis for phase 1, the occupancy status should be collected for long periods of time. Even more signals are needed to reach the second phase. This requires a good strategy for the storage and transmission of this data.

A first strategy to limit the amount of data is to track only changes in occupancy status. It is very likely that during the day, when people usually are not in bed, no changes occur in the occupancy status. It will be "EMPTY" for hours. But also at night it is much more efficient when only the changes are tracked and saved. It can be assumed that in between two changes the status is equal to the first status. This results in an "event-based" strategy, where an event reflects a change in occupancy status.

One important requirement is that an event should not be lost. This would lead to a misinterpretation of the data, which would mislead the sleep analysis. For this reason a failover buffering system is developed.

4.1.1 Failover system

The BEDsense relies on the wireless network to send the measured data to the server. No matter how stable the Wi-Fi network is, the Wi-fi connection might drop unexpectedly. When the BEDsense has no Wi-Fi connection, the measurements cannot be sent. Short interruptions are not yet a big problem, especially not in the first phase. As long as there is no event during the interruption, no data will be lost and the interruption will remain unseen for the user. On the other hand, even short times of no connection can cause problems in the next two phases of measuring sleep quality. When the duration of interrupted connection increases, the amount of data loss increases proportionally. This will result in unreliable data and a bad user experience.

It is impossible to prevent connection drops from occurring, so there has to be a failover system. In our case failover is switching to an alternative way of saving the measurement data upon the failure of the network connection. When the network connection is restored, it will send all saved data at once. This will prevent the BEDsense from losing data.

A protocol is designed to save the events in a practical way. Essentially, the BEDsense now embeds a logfile for events. The smallest structure that will be used in the logfile is called a tuple with following format:

{4-byte Event, 4-byte Event ID, 4-byte Boot ID, 4-byte Tick stamp}

Every event will create a tuple and will be saved in flash memory. 20 blocks of the memory are reserved for saving tuples. Every block contains 64 tuples, so a maximum of 1280 tuples can be stored on the device. This amount may seem low, but in normal use of the BEDsense, the number of events per day will be between several dozens to hundreds. Hence, the memory will be sufficient to prevent data losses for short connection drops.

When all blocks are full, the blocks with the oldest records will be removed and will be reused for the new records.

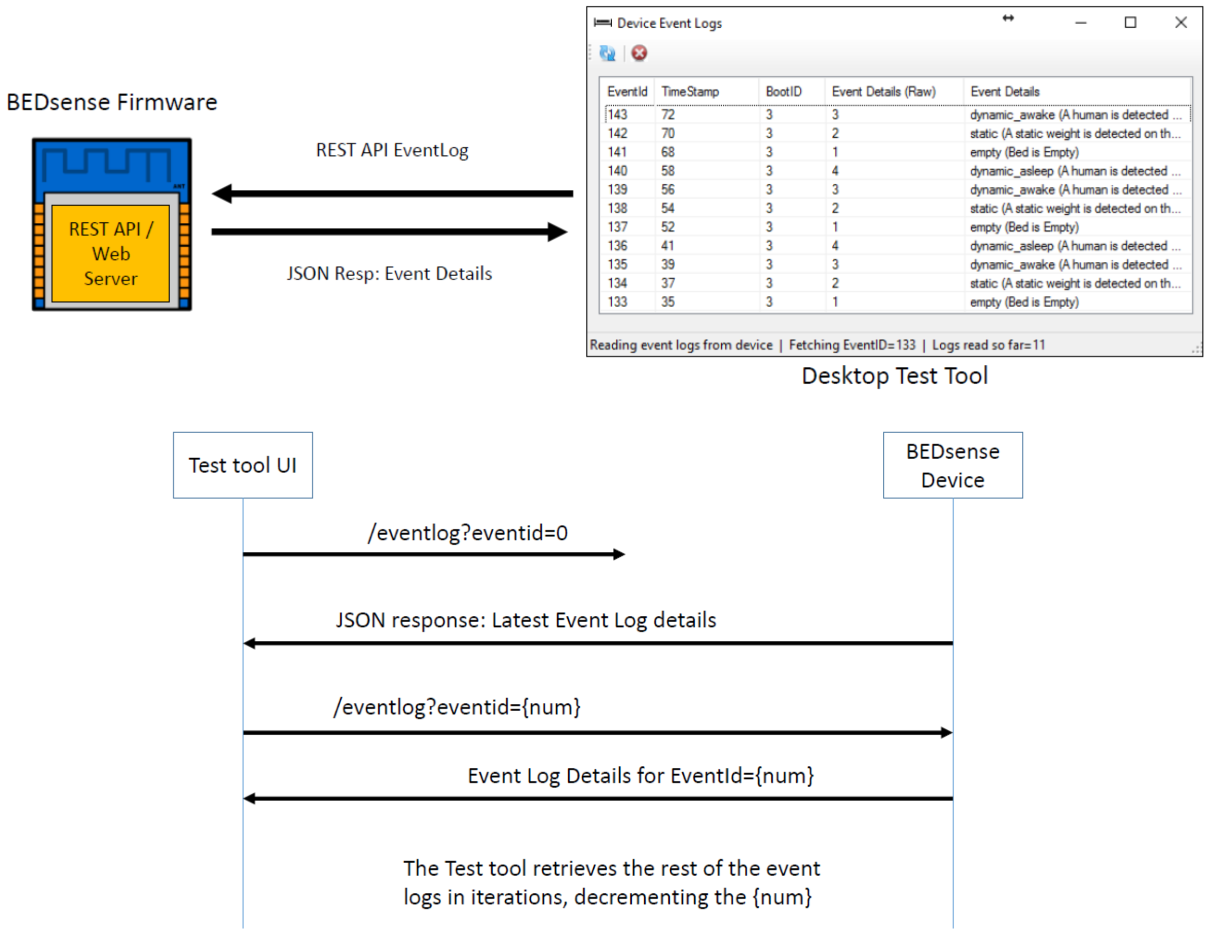
The BEDsense firmware exposes an API to retrieve the logs. The event log can be queried with this URL:

If the event ID equals 0, the response will be the most recent event. Any other number will give the actual event corresponding to that event ID (if it exists). An example of the JSON format is given below:

```
{
"event id":0,
"event records":[1,30,5,9]
}
```

Event ID is a sequential number given to each event, starting from 1. It is possible that events are not available anymore when they are removed to make space for new events. A strategy to get the logfile arranged from new events to old, is to request the last one. The event ID of the newest record is known, so you can request the previous record by decreasing this event ID. This can be repeated until the requested event ID doesn't exist.

This strategy is used in the desktop test tool (Figure 4.2).



EventId	TimeStamp	BootID	Event Details (Raw)	Event Details
143	72	3	3	dynamic_awake (A human is detected ...
142	70	3	2	static (A static weight is detected on th...
141	68	3	1	empty (Bed is Empty)
140	58	3	4	dynamic_asleep (A human is detected ...
139	56	3	3	dynamic_awake (A human is detected ...
138	54	3	2	static (A static weight is detected on th...
137	52	3	1	empty (Bed is Empty)
136	41	3	4	dynamic_asleep (A human is detected ...
135	39	3	3	dynamic_awake (A human is detected ...
134	37	3	2	static (A static weight is detected on th...
133	35	3	1	empty (Bed is Empty)

Reading event logs from device | Fetching EventID=133 | Logs read so far=11

Figure 4.2: Eventlog retrieval design using the desktop test tool

4.2 System design for phase 1

Although Zenseri is not in the sleep analysis market yet, the mechanisms to make these future steps possible can already be investigated. The first two phases in particular will have a similar basic structure. The first prototype already has a lot of things on board to start with the development of phase 1.

Sleep analysis in phase 1 is based on three possible events generated by the changes of occupancy status. Events contain time and status information of a BEDsense. By combining all events that are generated during a night, some important factors of sleep quality can already be deduced, think about the total duration of sleep, total duration in bed, number of wake ups, etc.

The ultimate system that allows sleep analysis to be done using the BEDsense events, is the following:

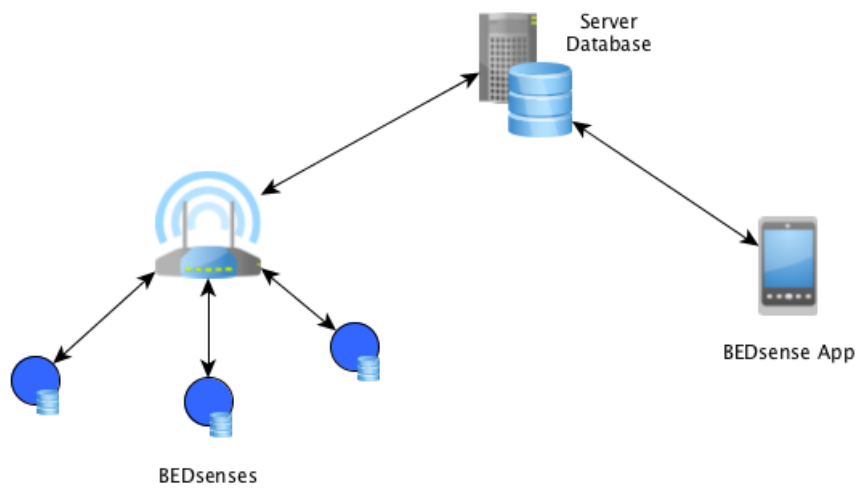


Figure 4.3: First phase sleep analysis system overview.

The BEDsenses are connected to the Internet via a Wi-Fi AP. Each BEDsense stores events using the failover system. The eventlog created by the failover system results in a list of tuples stored on the BEDsense flash memory. Each BEDsense can therefore be considered as a data source, which leads to a distributed storage of data. The concept of distributed database management is proved as one of the most energy-efficient data storage and query techniques for wireless sensor networks (WSN) [25]. The sensor network is seen as a distributed database where queries are injected from a base station [26]. The BEDsense is still a wired sensor, but will eventually become a wireless solution. Hence it is beneficial that a similar system is used as in the wired system.

It uploads events to an external server. The events are stored in a database on this server. Even if this Internet connection drops down for a while, the failover system ensures that all events are saved on the device and sent to the database when the Internet connection is restored.

Lastly, the BEDsense owner can look into his sleep history using an intended application. The application converts the events to numbers, graphs and percentages that are easily understandable for the user.

This architecture is a 3-tier architecture because there is a separate tier for:

- **the presentation:** what the end user gets to see;
- **the logic and data access:** controls application functionality by performing detailed processing. It also processes data;
- **data storage**: data is stored and retrieved from the database.

Due to the separation of the functionality, this architecture is highly manageable because almost all its components can be changed independently. We are not tied to these three layers. As the system grows, more layers can be added if necessary. For instance, when different database systems need to be approached in different ways, it is useful to add a database abstraction layer. This layer ensures that you can access the data independently of the database system [27].

4.2.1 Upload events

Events are stored on the BEDsense device as discussed in section 4.1.1. There are two ways of communicating the events to a central server; either push-based or pull-based. In the push mode, events are proactively forwarded from the BEDsense to the central server, whereas in pull-mode events are acquired on demand. The push mode is suitable for applications that require continuous monitoring. Events are forwarded to the central server periodically or when an interesting event occurs. The pull mode is suitable for applications where monitoring is not needed continuously [28].

Continuous monitoring of the BEDsense data is not needed for the sleep analysis application, so the pull-mode suffices.

4.2.2 Storage on central database

In the first place, it is important to know what the user wants to store and how he wants to use it before designing a strategy to implement the database. In this case – phase 1 of the sleep analysis – the user wants his sleep data for longer periods of time to be stored. This is needed because of the memory constraints of the BEDsense device. He will then want to be able to see his sleep data. He might want to see the evolution of sleep quality of last week or month. Or he wants to know exactly how many times he woke up last night, or how many hours he was effectively sleeping. Young parents might check the sleep behavior of their baby or child. With this user application in mind, a strategy for a database system is designed.

The first choice that must be made is whether to use SQL- or NoSQL-database – or relational and non-relational databases - to store the BEDsense sleep data. Relational databases are structured whereas non-relational databases are document-oriented and distributed. Because of the predefined structure of the occupancy events and the relation of the events to a BEDsense and thus to a user, it is obvious to use a relational database.

The design of a relational database is done using an entity-relationship diagram (ERD). For this first phase the ERD will remain very simplistic as shown below.

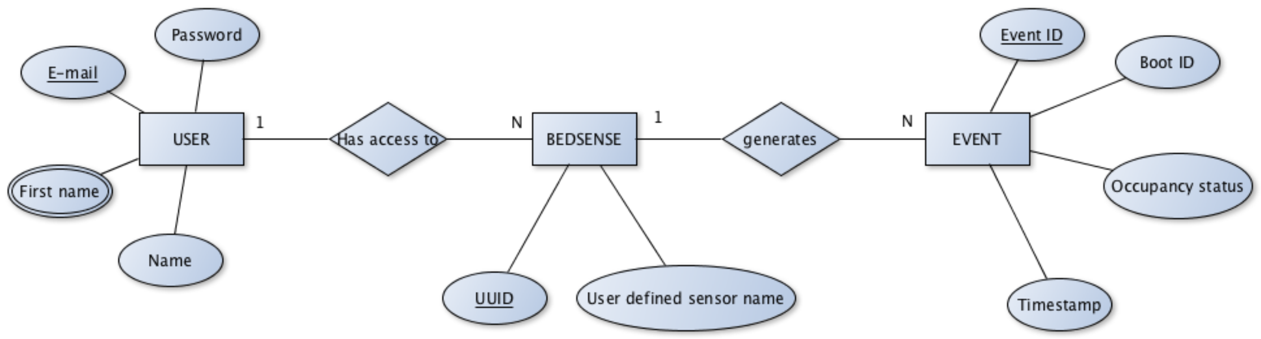


Figure 4.4: Entity-relationship diagram of phase 1 database

There are three entities: user, BEDsense and event. A user can have access to one or more BEDsenses. Each BEDsense generates events. The events correspond to the tuples from the failover system and contain the necessary information for the sleep analysis.

For demonstration purposes, based on the ERD in Figure 4.4, an SQL database is implemented. The popular open-source database MySQL is used for this implementation. The database will run on a local XAMPP server and managed with phpMyAdmin. The database will have to be accessible over the Internet in the future, which causes additional security measures. These will be discussed in chapter 5.

4.2.3 Visualization

Now that it is defined how the BEDsense data is transformed into events and how the events are stored in a database, we only have to convert it into useful representations. To demonstrate this, the mobile application is expanded.

Registration and login

The first thing a user must do in order to get access to the sleep analysis is to register himself. For this demonstration setup, only the e-mail address and password are used for registration and login. The e-mail address has to be unique because it's the key of the user table. After registration, the user can login using the same e-mail and password.

Sleep history

After being registered and logged in, the user has access to a list of BEDsenses. One BEDsense is typically linked to one person, and therefore the BEDsense can have the same name as the person it belongs to. From the list of BEDsenses, using these user defined names, one can select which device he wants to retrieve the sleep history.

There is only one step left to see the actual sleep analysis; select the date from which he wants to retrieve the measurements. This is done using a calendar, which is set by default on the current date. When selecting a date, a request is sent to the database to get all occurred events from 12:00 p.m. of the selected day, till 12:00 p.m. the next day. Using the retrieved events, the sleep analysis of one night can be done. How the app requests the events and transforms them into a visual representation is explained below.

Requesting the events is done asynchronously in the background. This asynchronous task sends an HTTP post request with the two timestamps (selected day and next day) and the BEDsense UUID. If the selected BEDsense has events in between the two dates, a JSON array of the events is returned. Every JSON object in the array reflects a status change of the BEDsense. The app transfers each JSON object to an object of the class OccupancyEvent.

When the user wants to see the sleep history of a certain period of time, the app queries the events from the database and creates an arraylist of occupancy events. This arraylist is used to create useful sleep information. A lot of interesting information can be derived from these events such as:

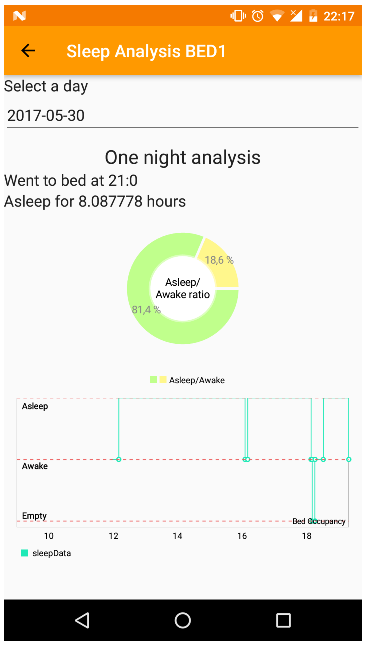
- time when you went to bed;
- time when you fell asleep;
- number of times you woke up;
- number of times you went out of bed;
- total duration in bed or asleep;
- ratio between the length of time spent in bed and the length of time you effectively slept;
- the length of time you were awake before you got out of bed.

To derive the above parameters, the processing is very similar for every calculation. The next fragment of code shows how the arraylist of events is used to calculate the total duration that the person was asleep:

```java
private float getDurationAsleep(ArrayList<OccupancyEvent> events){
    float asleep = 0;
    for(int i=0; i < events.size()-1;i++){
        OccupancyEvent event = events.get(i);
        OccupancyEvent next_event = events.get(i+1);
        Log.d("DEBUG" , "Occupancy.Status = " +
        event.getOccupancy_status());
        if(event.getOccupancy_status()==Occupancy_status.ASLEEP){
            float seconds  = (float) (next_event.getDateTime().getMillis()
                                 - event.getDateTime().getMillis())/1000;
            asleep = asleep + seconds;
        }
    }
    return asleep;
}
```

The above function returns the total number of seconds where the occupancy status was asleep. By selecting two consecutive events and checking the status, it is possible to derive the duration of one status, i.e. duration asleep. The result can be used in further calculations such as the in bed/asleep ratio.

The results of the calculations are still just numbers, which are not user-friendly. Users typically expect a clear graph or a simple score from 0-100, completed with some easy to understand parameters. To make this possible, a chart view/ graph library is used [29]. Figure 4.5 illustrates an example of how the events can be visualized.



22:17

Sleep Analysis BED1

Select a day

2017-05-30

One night analysis

Went to bed at 21:0

Asleep for 8.087778 hours

Figure 4.5: Visualization of sleep data from one night

The user can select a specific date using a calendar view. The app converts all occupancy events into useful information. The user can right away see the exact time that he went to bed and the total duration he was asleep. A pie chart shows the ratio that he effectively slept while he was in bed. Another graph shows the points of time when the occupancy events occurred.

It is worthwhile noticing that the visualizations of **Fout! Verwijzingsbron niet gevonden.** are just examples of how the events can be transformed into user readable views. The appearance of these views can always be changed according to a desired theme or personal taste.

This conversion of events into quality metrics can be repeated for multiple nights to come up with sleep history views for weeks or even months. Again, the way of visualizing the results is less important.

5 Security

BEDsense is in the era of Internet of Things, where digitally connected devices are changing our lives, including our homes, offices, cars, etc. IoT is growing at a fast pace and researches around the world estimate that by 2020, the number of active wireless connected devices will exceed 40 billion [30]. The upside is that we will be able to do things we never imagined before. But as with every good thing, there is a downside: IoT is becoming an increasingly attractive target for cybercriminals. More connected devices result in more attack vectors and more possibilities for hackers to target us [31].

Various attacks with IoT devices have already been demonstrated. For instance, IoT botnets were used in multiple Distributed Denial of Service (DDoS) attacks against DNS servers in October 2016 [32]. It was a unique type of attack because for the first time it was not launched from a PC, but using IoT devices such as security cameras and some network attached storage. The reason why such an attack was possible was because of the complete lack of security in those devices. In fact, they are connected to the Internet while having default access. They come out of the factory, consumers are buying them and they are exposed to the Internet. Hackers could take them over and use them to launch the DDoS attacks.

It is important to be aware of these IoT security concerns during these early development phases to make the BEDsense future proof. Also for the user and the Zenseri itself, it is important that the BEDsense system is designed while taking the security issues into account.

Building a completely secure system is a virtual impossibility so the objective is not developing an unhackable solution but finding a good compromise between security and performance with the available resources. In the end, we will have a reasonable product with standard forms of security and know the weaknesses and understand where the security problems are. The next subsections take a closer look at the different safety aspects.

5.1 Authentication

To prevent easy access to the BEDsense devices, no default passwords that are the same across all devices are used. Instead, the BEDsense provides decent authentication using unique passwords for each device that rolls of the factory. The password will be a strong one and is labeled on the packaging so that the owner can login but other people cannot.

5.2 Encryption

Algorithms in the optical engine convert the signal from the external optics into sleep data. It is extremely important that the algorithms remain secret. To make sure they are protected during firmware updates, the firmware of the optical engine - containing the algorithms – is encrypted using AES (Advanced Encryption Standard). The bootloader is able to decrypt the firmware before installation.

5.3 Database

The designed database for the phase 1 sleep analysis system already includes some security measures:

- all SQL queries are protected from SQL injection using prepared statements and parameterized queries. When preparing the SQL statement, it is parsed and compiled by the database server. By specifying parameters, the database engine is told where you want to filter on. When the execution is called, the prepared statement is combined with the specified parameter values. By sending the actual SQL separately from the parameters, the risk of ending up with something you did not intend is limited;
- hashed passwords are used to prevent the stored passwords to be stolen [31]. The hash algorithm is applied to the user's password before it is stored on the database making it implausible for attackers to determine the original password, while still being able to compare the resulting hash to the original password.

Additional security means will have to be taken when the BEDsense data is stored by transmitting it over the Internet to a central server. Instead of using the HTTP protocol for communication, the secure version HTTPS has to be used to send user information.

5.4 Firmware updates

The fact that the BEDsense is upgradable using the FOTA system is a very important security aspect on its own. When security issues are found, and they will, the BEDsense device can be upgraded and the security vulnerabilities can be closed.

There is however an important thing about the FOTA updates. When an upgrade is performed, the device needs to check whether it is actually performing a legitimate update. Hence, the update needs to be signed using a certificate and the device needs to validate whether the certificate has the right signature on it. This is not strictly necessary for updates during development, but is something that should definitely be considered in commercial use.

6 Conclusion

6.1 Results

Exploring in depth different aspects of integrating IoT-like devices into Smart Home environments, made it possible to determine what was appropriate for the BEDsense integration. In collaboration with other people, both inside Zenseri and 2M Engineering as well as outside, choices about the design are made. It was therefore not just making choices but a crucial part of this project was communicating the choices with cooperating parties. This collaboration has led to a BEDsense prototype. This prototype is successfully equipped with necessary communication means needed for integration into Smart Home environments:

- Wi-Fi provisioning means (WPS- and AP-mode),
- SSDP protocol definitions for discovery,
- user interface via Android application,
- over the air upgradable.

Subsequently, a system is created where raw sensor data is transformed into a format that is suitable for storage and transmission. This system enables the first phase for doing sleep analysis. The implementation of the database and communication with the mobile app show the possibilities of this first phase and form the basis for the next phases.

6.2 Discussion

The starting point of this master's thesis was a promising, innovative sleeping sensor able to detect bed occupancy from underneath mattresses, but unable to communicate the data so that it becomes useful information. The developed communication means enable integration in Smart Home networks, the efficient transmission of the BEDsense data and the possibility to add functionality due to the FOTA upgrades.

The designed prototype is not a final product that is suitable for commercial purposes, but it demonstrates that it is possible to create a low-cost solution with reasonable performance regarding ease of use, robustness and security. In the next steps of the development, new prototypes will need to be developed. The IoT market is growing at a stunning fast pace and therefore the BEDsense is forced to evolve as quickly. In the future, the BEDsense will become a wireless sensor and will therefore use different communication technologies such as Z-Wave or Zigbee instead of Wi-Fi. This doesn't take away the importance of this prototype, on the contrary, it is a major step forward and the results will still be useful for the next development phases.

When it comes to the results for the sleep analysis strategy for the first phase, we can determine that the possibilities should not be underestimated. Although only the bed occupancy status with three useful values is measured; a lot of significant factors that determine sleep quality can be derived. The implementation of the database and the visualization of sleep history in the mobile application are not suitable for commercial use. The local database needs to become an external database so that it is accessible from anywhere and anytime. This step involves many other measures, especially in terms of security and legislation. Furthermore, another type of database system is needed to make Big Data analysis possible. SQL databases do not scale well to very large sizes. These are points that need to be addressed in the future.

In order to achieve the ultimate goal where the BEDsense data is transformed into sleep profiles and recommendations to improve sleep, the BEDsense needs o transfer more data and the data management will have to be done in a totally different way.

6.3 Conclusion

The results of this thesis facilitate further development of the BEDsense system and are a major step to the ultimate goal of Zenseri, integration in Smart Home environments and collection of sleep data to improve the user's sleep lifestyle.

Bibliography

[1] Statista. (2017). *Worldwide Smart Home statistics*. Available: https://www.statista.com/outlook/279/100/smart-home/worldwide

[2] J. Laferrière, G. Lietaert , R. Taws , and S. Wolszczak *Reference Guide to Fiber Optic Testing* vol. 1: JDSU, 2007.

[3] R. P. DePaula, N. Lagakos, J. A. Bucaro, and E. Udd, "Optimizing Fiber Optic Microbend Sensor," vol. 0718, pp. 12-20, 1987.

[4] Gil Reiter. (2014, 16-10-2016). *A primer to Wi-Fi provisioning for IoT applications*. Available: http://www.ti.com/lit/wp/swry011/swry011.pdf

[5] Viehböck Stefan, "Brute forcing Wi-Fi Protected Setup," 2011.

[6] Espressif, "ESP-touch user guide," ed, 2016, p. 1.

[7] Amazon. (2017, 30 Apr). *Amazon Echo*. Available: https://www.amazon.com/Amazon-Echo-Bluetooth-Speaker-with-WiFi-Alexa/dp/B00X4WHP5E

[8] Nest. (2017, 30 Apr). *Programs itself. Then pays for itself.* Available: https://nest.com/thermostat/meet-nest-thermostat/

[9] "UPnP Device Architecture Version 1.1," ed. ISO/IEC 29341-1-1, 2011.

[10] W. Chen, S.-Y. Kuo, and H.-C. Chao, "Service integration with UPnP agent for an ubiquitous home environment," *Information Systems Frontiers,* vol. 11, p. 483, 2008.

[11] Espressif Systems IOT Team, "ESP8266 FOTA introduction," Espressif2016.

[12] P. R.M.de Andrade, A. B.Albuquerque, O. F. Frota, R. V Silveira, and F. A. da Silva, "Cross Platform App : A Comparative Study," *International Journal of Computer Science and Information Technology,* vol. 7, pp. 33-40, 2015.

[13] M. Panhale, *Beginning Hybrid Mobile Application Development*, 1 ed.: Apress, 2016.

[14] R. Smeets and K. Aerts, "Trends in Web Based Cross Platform Technologies," *International Journal of Computer Science and Mobile Computing,* vol. 5, pp. 190-199, 2016.

[15] Android. (2017, 20 Oct). *The world's most popular mobile OS*. Available: https://www.android.com/

[16] Apple. (2016, 20 Oct). *iOS 10*. Available: https://www.apple.com/ios/ios-10/

[17] Ziflaj Aldo. (2014, 20 Oct). *Native vs Hybrid App Development*. Available: https://www.sitepoint.com/native-vs-hybrid-app-development/

[18] IDC. (2017, 22 Apr). *Smartphone OS Market Share, 2016 Q3*. Available: http://www.idc.com/promo/smartphone-market-share/os

[19] Cordova. (2016, 20 Oct). *Mobile apps with HTML, CSS & JS*. Available: https://cordova.apache.org

[20] Appcelerator. (2016, 20 Oct). *Build great mobile experiences faster,* . Available: http://www.appcelerator.com/

[21] Xamarin. (2016, 20 Oct). *Everything you need to deliver great mobile apps*. Available: https://www.xamarin.com/

[22] Cygnet, "PhoneGap or Titanium or Xamarin - Which Cross-Platform Framework Should You Choose?," vol. 2016, ed, 2015.

[23] A. Heinrich, X. Aubert, and G. de Haan, "Body movement analysis during sleep based on video motion estimation," pp. 539-543, 2013.

[24] S. Gori, G. Ficca, F. Giganti, I. D. Nasso, L. Murri, and P. Salzarulo, "Body movements during night sleep in healthy elderly subjects and their relationships with sleep stages," *Brain Research Bulletin,* vol. 63, pp. 393-397, 6/30/ 2004.

[25] O. Diallo, J. J. P. C. Rodrigues, M. Sene, and J. Lloret, "Distributed Database Management Techniques for Wireless Sensor Networks," *IEEE Transactions on Parallel and Distributed Systems,* vol. 26, pp. 604-620, 2015.

[26] T. Hara, V. I. Zadorozhny, and E. Buchmann, *Wireless Sensor Network Technologies for the Information Explosion Era*, 2010.

[27] Kris Aerts, *Databaseprogrammatie met Java en C#*, 2014.

[28] A. Skordylis, N. Trigoni, and A. Guitton, "A Study of Approximate Data Management Techniques for Sensor Networks," pp. 1-12, 2006.
[29] Philipp Jahoda. (2017, 25-04). *MPAndroidChart*. Available: https://github.com/PhilJay/MPAndroidChart
[30] C. Drubin, "The Internet of Things will Drive Wireless Connected Devices to 40.9 Billion in 2020," *Microwave Journal,* vol. 57, p. 51, Oct 2014 2014.
[31] B. Dickson, "Why IoT Security Is So Critical," *TechCrunch,* 2015 Oct 24 2015.
[32] M. Smith, "IoT botnets used in unprecedented DDoS against Dyn DNS; FBI, DHS investigating," *Network World (Online),* 2016 Oct 22 2016.