TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Thomas Johann Seebeck Department of Electronics

Aivar Koodi 153756IVEM

AUTOMATIC CONFIGURING SOLUTION FOR INTERNET OF THINGS APPLICATIONS

Master's thesis

Supervisor: Mairo Leier
PhD
Co-supervisior: Olev Märtens
Professor

Tallinn 2017

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Thomas Johann Seebecki elektroonikainstituut

Aivar Koodi 153756IVEM

VÄRKVÕRGU SEADMETE AUTOMAATSE KONFIGUREERIMISE LAHENDUS

Magistritöö

Juhendaja: Mairo Leier
Doktorikraad
Kaasjuhendaja: Olev Märtens
Professor

Tallinn 2017

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Aivar Koodi

19.05.2017

Abstract

The number of Internet-of-Things (IoT) devices is increasing and it creates a need to configure them in a convenient and fast manner. This master's thesis is focusing on developing a solution for automatic configuring to provide these features for IoT devices. Although there are many solutions developed in the past, the one proposed in this thesis uses new technological possibilities enabled by Bluetooth Low Energy (BLE) and Wireless Fidelity (Wi-Fi) in System-on-a-Chips (SoC).

Thesis consists of choosing the suitable chip for the task, stating the system requirements and developing the method for automatic configuration system. In addition, a software is developed based on the method and tested to verify its' workability. The work ends with the presentation of the results, conclusion and future work.

This thesis is written in English and is 86 pages long, including 7 chapters, 45 figures and 1 table.

Annotatsioon

Värkvõrgu seadmete automaatse konfigureerimise lahendus

Värkvõrgu seadmete kasvutrendiga tekib vajadus nende kiireks ja mugavaks konfigureerimiseks. Käesolev magistritöö käsitleb värkvõrgu seadmete automaatse konfigureerimislahenduse välja töötamist, et neid nõudmisi rahuldada. Kuigi konfigureerimiseks on arendatud ka teisi süsteeme, keskendub see töö automaatse konfigureerimise lahenduse arendusel uusimate mikrokontrollerite kasutamisele, mis võimaldavad kasutada Bluetooth Low Energy (BLE) ja Wireless Fidelity (Wi-Fi) võimekust.

Magistritöö koosneb mikrokontrolleri valikust, konfigureerimise süsteemi nõuete seadmisest ja metoodika välja töötamisest. Lisaks luuakse metoodikal baseeruv tarkvara, mida ka testitakse, et kinnitada selle kontseptsiooni töötavust. Töö võtavad kokku tulemuste analüüs, järeldused ning arendusvõimalused.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 86 leheküljel, 7 peatükki, 45 joonist, 1 tabelit.

List of abbreviations and terms

IoT	Internet of Things
SoC	System-on-a-Chip
USD	United Stated Dollar
BLE	Bluetooth Low Energy
CPU	Central Processing Unit
DMIPS	Dhrystone millions of instructions per second
KB	Kilobyte
ROM	Read Only Memory
SRAM	Static Random Access Memory
RTC	Real-Time Clock
MB	Megabyte
QSPI	Quad serial peripheral interface
SAR	Successive Approximation Register
ADC	Analog to Digital Converter
DAC	Digital to Analog Converter
SPI	Serial Peripheral Interface
I²S	Inter-IC Sound
I²C	Inter-Integrated Circuit
UART	Universal Asynchronous Receiver/Transmitter
SD	Secure Digital
SDIO	Secure Digital Input Output
MMC	Multi-media Card
MAC	Media Access Control
DMA	Direct Memory Access
IEEE	Institute of Electrical and Electronics Engineers
CAN	Controller Area Network
IR	Infrared
TX	Transmission

RX	Receive
PWM	Pulse-Width Modulation
LED	Light-Emitting Diode
OTP	One Time Programmable
MIMO	Multiple-Input and Multiple-Output
SISO	Single-Input and Single-Output
MSC	Modulation and Coding Scheme
UDP	User Datagram Protocol
NFC	Near Field Communication
API	Application Programming Interface
Wi-Fi	Wireless Fidelity
SSID	Service Set Identifier
AP	Access Point
AES-CCM	Advanced Encryption Standard- Counter with CBC-MAC
CBC-MAC	Cipher Block Chaining- Message Authentication Code
WPA	Wi-Fi Protected Access
WFA	Wi-Fi Alliance
WLAN	Wireless Local Area Network
WAPI	WLAN Authentication and Privacy Infrastructure
RNG	Random Number Generator
RSA	Rivest-Shamir-Adleman public-key cryptosystems
ECC	Elliptic Curve Cryptography
SHA-2	Secure Hash Algorithm 2
mDNS	multicast Domain Name System
LOS	Line of sight
Optiverter	Inverter with optimized energy production algorithms
RSSI	Received signal strength indication
Mbps	Megabits per second
TCP	Transmission Control Protocol
IIoT	Industrial Internet of Things
M2M	Machine-to-Machine
RTOS	Real-Time Operating System
CRC	Cycle redundancy checks
MHz	Megahertz

Table of contents

List of figures

List of tables

1 Introduction

This document is the master's thesis written in the autumn of 2016 and spring of 2017 at Tallinn University of Technology under School of Information Technologies, Thomas Johann Seebeck Department of Electronics during IVEM11/15-Communicative Electronics Master's program with specialization to Cognitive Electronics. The subject of this thesis is Automatic Configuration System for Internet of Things Applications.

1.1 Motivation

The topic was proposed by Ubik Solutions OÜ who needs a solution to configure communication units of Optiverters easily. Optiverter is a patented technology to increase the efficiency of solar panel inverter. Right now, the configuration is conducted one at the time. Since there might be tens of devices in one installation, the current solution is not user-friendly nor time efficient.

The innovation of this work is using new technologies enabling Wireless Fidelity (Wi-Fi) and Bluetooth coexistence. Conducting automatic configuration for several devices using Bluetooth Low Energy (BLE) has not been done before according to findings presented in this thesis.

Furthermore, with the emerge of Bluetooth 5.0 core specification, there is a chance that promised flooding and routing mesh capability will be included in next version. This would establish a basis for more reliable and energy efficient data transmission, since only one device needs to be connected to the Wi-Fi Access Point (AP) and the data of other devices will be sent to it via Bluetooth mesh. In this sense, the developed automatic configuration solution adds a great functionality for future communication systems.

The trend of Internet of Things (IoT) is changing how users communicate with the rest of the world. According to Business Insider Intelligence in-depth research report it is projected that businesses are going to spend approximately 5 billion dollars on the IoT solutions in the next five years raising total number of IoT devices from 6,6 to 22,5 billion by the end of 2021 [1].

The issue, how to configure Wi-Fi easily, is currently actual because the user satisfaction is closely related with the industry drivers. The IoT industry is currently propelled by the high adoption of smartphones and tablets, expanded internet connectivity, decreasing cost of internet-connected sensors and huge investments in the sector [1].

Easy device deployment and solved reliability issues for highly beneficial devices propagate consumers, businesses and governments to invest into them and trust their workability to ease everyday life, therefore giving a boost to last two mentioned drivers of the IoT field.

Therefore, a system that could provide mentioned benefits, is an interesting and necessary research topic for mentioned company and many others.

1.2 Problem statement

Currently used solution to configure Optiverter communication units is not time efficient nor convenient. This creates a need to replace the solution with the one that could solve these issues. The problems introduced and solved in this master's thesis are presented as follows:

- how to configure IoT devices without reprogramming;
- how to decrease time spent on configuring;
- how to automate the configuring process
- how to make configuring process convenient to users.

1.3 Objectives and thesis organization

The objective of this thesis is to develop a solution to automatically configure IoT devices in order to increase the user-friendliness and decrease time spent on configuration processes. In addition, the software communication unit must be selected, software developed and tested to prove the workability of the concept.

This thesis follows the engineering method structure, where sections are organized as follows:

- section II presents the State-of-the-Art overview of the configuring systems;

- section III is stating the requirements for the automatic configuration system;
- section IV describes the reasoned selection of communication device for this thesis;
- section V introduces the developed method for automatic configuring system;
- section VI presents the experimental evaluation of the developed system;
- section VII presents the results, conclusions and future work of the thesis.

2 Configuration solutions

Analysis of the State-of-the-Art is necessary to take account the benefits and flaws of previously developed configuration solutions. The data gathered about the automatic configuration solutions is presented from oldest to newest.

2.1 State-of-the-Art

2.1.1 Texas Instruments CC3000 SmartConfig

SmartConfig™ technology was developed by Texas Instruments in 2014 to connect a CC3000-enabled device to the home wireless network using a one-step and one-time process. Main functionality of this system is to configure multiple CC3000 devices to the same AP at the same time. In addition, it is possible to configure multiple CC3000 devices to different APs serially. Following sections describe the procedure overview and the limitations of this technology [2].

2.1.1.1 Procedure overview

SmartConfig procedure is initiated after CC3000 enters SmartConfig mode. The mode is triggered externally on the CC3000 device when a button is pressed on the processor's board [2].

Once the SmartConfig state is initiated, the CC3000 starts probing UDP (user datagram protocol) packets. These packets contain the information regarding the SSID (Service Set Identifier) and the password of the selected Wi-Fi AP. CC3000 will connect to the selected AP [2].

After CC3000 receives the configuration data, it creates a SIMPLE_CONFIG_DONE asynchronous event. In addition, the CC3000 will save the SmartConfig association information received in shared memory [2].

Although the SmartConfig process can be performed without Advanced Encryption Standard 128 (AES-128) encryption, it is not reasonable to use this possibility. It will create a security issues because the association information is not secured during broadcast process and while holding the data in shared memory [2].

If AES-128 encryption is enabled, the configuring device will encrypt the association information broadcast. A specific key chosen by both sides is used to conduct this operation. The CC3000 will place the encrypted association information received to shared memory, which is then decrypted and stored as a profile [2].

In the final stage of the SmartConfig, the profile is stored, CC3000 resets and connects to the selected Wi-Fi AP. To announce the completion of the configuration process and stop it on the configuring device's side, the mDNS packet is sent by the CC3000 [2].

Basic principle is described on Figure 1. CC3000 SmartConfig basic principle Figure 1 and process flow on Figure 2.

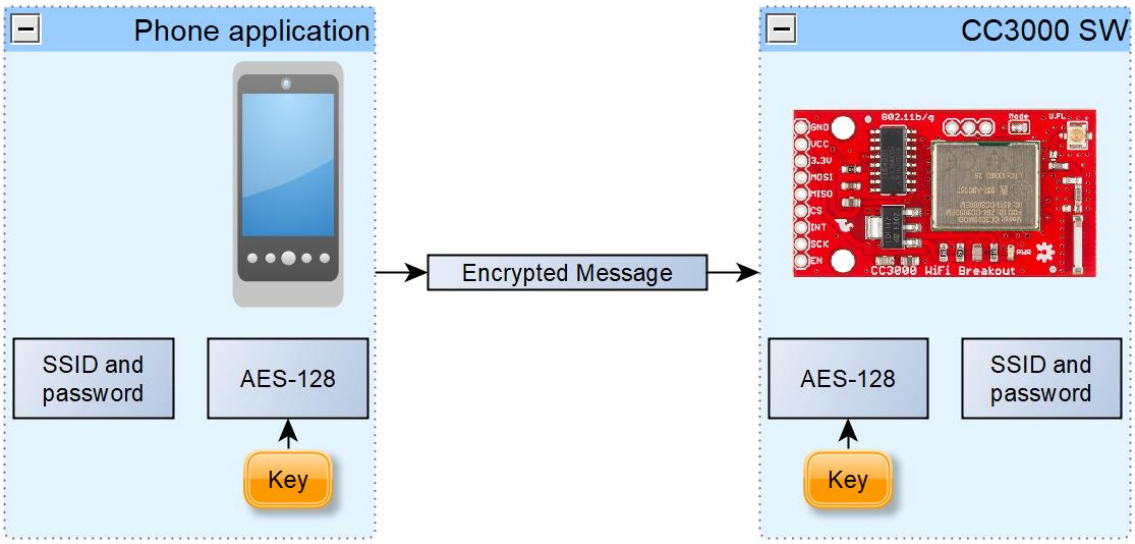


Figure 1. CC3000 SmartConfig basic principle [2]

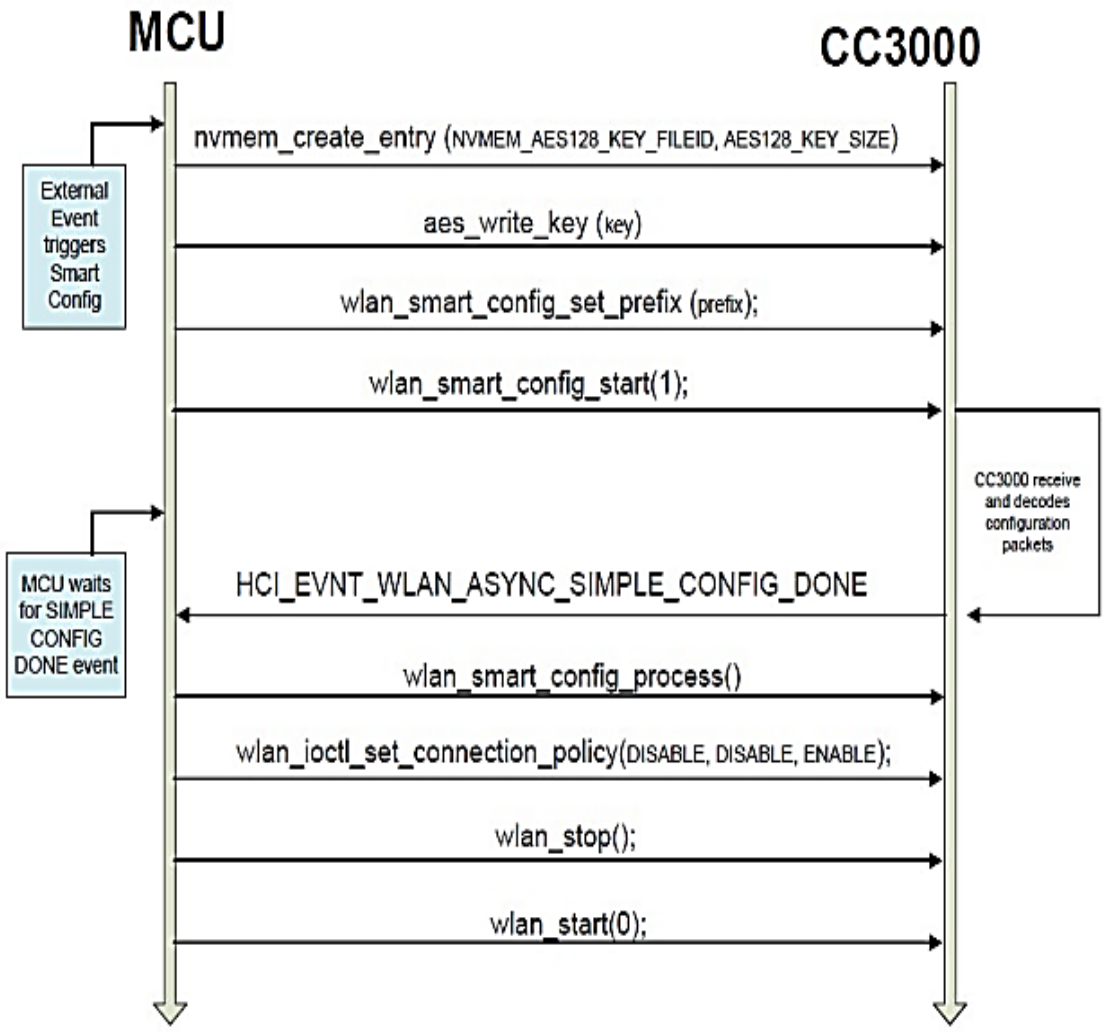


Figure 2. SmartConfig flow - AES encryption enabled [2]

2.1.1.2 Advantages

Advantages of TI CC3000 Smart Config are listed as follows:

- user-friendly;
- allows to configure multiple devices at once;
- configuration process can be done far from the device;
- SSID and password data encrypted.

2.1.1.3 Limitations

Limitations of TI CC3000 SmartConfig are listed as follows:

- “must use CC3000 devices for system setup;
- configuration process uses Wi-Fi for data transmission;
- new devices in the system require additional configuration process;
- it is not possible to configure multiple CC3000 devices to different APs simultaneously;
- key length is limited to 31 characters only when working with AES encryption;
- WEP security without SmartConfig AES encryption doesn't work;
- MTU in the configurable side should be set to 1500;
- it does not work with configuring devices that are MIMO devices;
- it does not work with configuring SISO devices at a channel width of 40MHz;
- it works at 11n rates only up to MCS7 [2].”

2.1.2 Espressif ESP-TOUCH

ESP-TOUCH communication protocol technology was developed by Espressif Inc. in 2015. The main function of this technology is to connect ESP devices to selected Wi-Fi AP, since it is not connected to network at the beginning. Following sections describe the procedure overview and the limitations of this technology [3].

2.1.2.1 Procedure overview

ESP-TOUCH Smart Config process is initiated after the Smart Config function is enabled in a device that supports ESP-TOUCH. Once this condition is fulfilled, the configuration using Wi-Fi enabled device could begin [3].

Wi-Fi enabled device such as tablet or smartphone connects and sends UDP packets to the Wi-Fi AP via specially dedicate mobile application. In addition, it will encode the SSID and password into the length field of a sequence of UDP packets. From there the ESP device reaches and decodes the information to connect with AP. After the

completion of the configuration process, the device will return the IP of the transmitter side and the transmitter side will get the IP of the device. The basic concept of the process is described on Figure 3 [3].

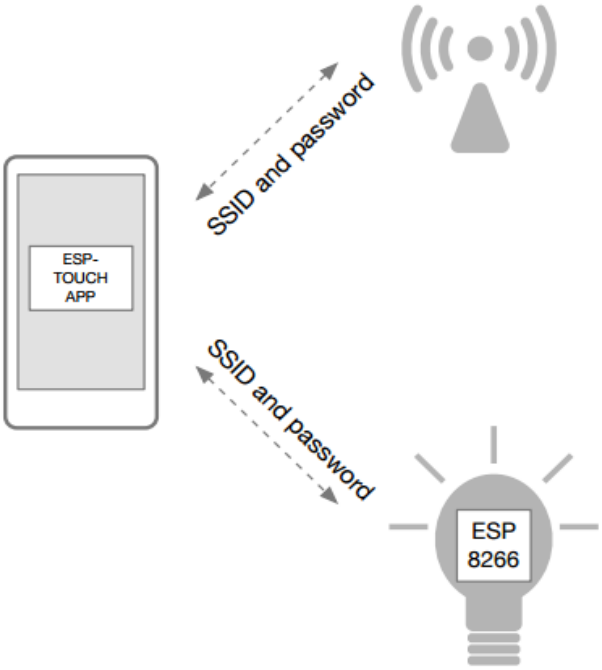


Figure 3. ESP-TOUCH basic principle [3]

If the device is unable to connect to the router within a specified time, the application returns the error message. Error message and the next round of SmartConfig process will start if the device is unable to obtain SSID and password within a certain time [3].

2.1.2.2 Advantages

Advantages of ESP-TOUCH are listed as follows:

- user-friendly;
- relatively fast configuring;
- configuration process can be done far from the device;
- allows to configure multiple devices at once.

2.1.2.3 Limitations

Limitations of ESP-TOUCH are listed as follows:

- must use ESP devices for system setup;
- configuration process uses Wi-Fi for data transmission;
- new devices in the system require additional configuration process.

2.1.3 Wi-Fi Alliance NFC “tap-to-connect”

NFC configuration technology is being developed by many different application software manufacturers. Some of the notable applications in this area are Wi-Fi Alliance, NFC Quick Actions and NFC Task Launcher. The main functionality of using Wi-Fi Alliance “tap-to-connect” software is to write Wi-Fi AP’s SSID and password on a Near Field Communication (NFC) tag, which is then used to give the credentials to device to access correct Wi-Fi AP. Following sections describe the procedure overview and the limitations of this technology [4].

2.1.3.1 Procedure overview

NFC configuration process is initiated when device is ready to scan the NFC tag. Software capable of writing data on the mentioned tag is used to write the association information on the tag. Information contains the selected Wi-Fi AP’s SSID and password. Once the tag is scanned by the device, the device automatically connects to selected Wi-Fi AP [4].

2.1.3.2 Advantages

Advantages of NFC configuration are listed as follows:

- relatively fast configuring;
- simple to use.

2.1.3.3 Limitations

Limitations of NFC configuration are listed as follows:

- configuration process must be done near the device;
- requires a NFC tag for configuring;
- each device must be configured separately;

- new devices in the system require additional configuration process.

2.1.4 Optiverter configuring solution

The development of Optiverter communication unit configuring solution began in 2015 by Ubik Solutions OÜ. The main functionality of using this solution is to send Wi-Fi AP's SSID and password via special mobile application to access correct Wi-Fi AP. Following sections describe the procedure overview and the limitations of this technology.

2.1.4.1 Procedure overview

Configuration process is initiated when device is in configuration mode. By using mobile application, the Wi-Fi SSID and password are sent to the device via Wi-Fi. Once the device is configured it will automatically connect to selected Wi-Fi AP.

2.1.4.2 Advantages

Advantages of Optiverter configuring solution are listed as follows:

- simple to use.
- configuration process can be done far from the device;

2.1.4.3 Limitations

Limitations of Optiverter configuring solution are listed as follows:

- relatively slow configuring;
- each device must be configured separately;
- new devices in the system require additional configuration process.

2.2 Summary of the State-of-the-Art

This chapter showed that there are different ways to carry out configuration processes. Found technologies include CC3000 SmartConfig, ESP-TOUCH, NFC "tap-to-connect" and Ubik Solutions OÜ configuring solution.

CC3000 SmartConfig, ESP-TOUCH and Optiverter configuring solution introduced a solution to configure the devices without close connection by using Wi-Fi for configuring. This is needed to improve comfort while configuring devices from a distance. For example, when devices are mounted on solar panels which are installed on the roof.

CC3000 SmartConfig and ESP-TOUCH provide the minimal time for configuration process by enabling the possibility to configure several devices at the same time.

NFC “tap-to-connect” is the most energy efficient by using NFC technology. The limitation of the technique is that it needs to conduct configuration process for each device separately and cannot configure from distance.

In addition, previously mentioned systems can not automatically configure a device which is added to the system later.

To overcome these flaws, a system is proposed ensuring minimal time on configuration process and user-friendly experience.

3 System requirements

To establish connection between communication devices and server, it is necessary to configure communication devices by giving them Wi-Fi AP's SSID and password, and server's Application Programming Interface (API) keys. Since configuring all devices one at the time is very inefficient, a system is needed to decrease the time and increase the user-friendliness of the configuration process while implementing new devices to the local device cluster.

The system requirements chapter covers the main features that must be provided to the device automatic configuration system taking account the demands stated in the Problem statement. These requirements are divided in 3 main subtopics: environmental, configuration and security.

3.1 Requirements

3.1.1 Environmental requirements

1. Units must be able to withstand temperature range -40..80 degrees Celsius.

2. Units must be able to transmit and receive data within line of sight (LOS) distance of 3 meters via BLE.

3. Units must be able to transmit and receive data within LOS distance of 40 meters via Wi-Fi.

4. Units must be functional for 25 years.

3.1.2 Configuration requirements

1. Communication between cluster devices must be done using BLE. Necessary for future to develop a mesh network for data transmission purposes with the server. More information about Bluetooth 5.0/Bluetooth Low Energy is presented in Appendix 1 – Bluetooth Core Specification versions. In addition, it could be

used to decrease the power consumption of the configuration process [6]. More information about BLE is presented in Appendix 2 – Bluetooth 4.2 / Bluetooth Low Energy.

2. Communication between cluster device and server must be done using Wi-Fi. Necessary to provide the long-range communication capability.

3. User must be able to connect to every unit via BLE using dedicated mobile application.

4. Units must be able to store SSID and password of the main Wi-Fi AP provided via BLE by user using dedicated mobile application or other units in the cluster.

5. Units must be able to detect all nearby BLE devices and store their media access control (MAC) addresses and Received signal strength indication (RSSIs).

6. Recently configured unit must be able to connect to server database in order to check whether the nearby devices belong to the cluster or not. Connection with server is established via Wi-Fi.

7. Units must be able to verify other units belonging in the cluster using data from server.

8. Units which has the SSID and password of the main Wi-Fi AP must be able to distribute it to other units belonging in the cluster via BLE.

3.1.3 Security requirements

1. All Bluetooth connections must be initialized using pairing processes, which is encrypted by P-265 elliptic curve cryptography.

2. Data transmission via Bluetooth must be encrypted using AES-CCM cryptography.

3. Units must be able to remove MAC addresses and RSSI data from cluster list of all nearby Bluetooth devices which does not belong to the cluster.

4. Connection and data transmission with Wi-Fi AP must be protected using Wi-Fi Protected Access 2 – Enterprise (WPA2-Enterprise).

5. Connection with server must be established through https and using unit specific API-keys.

3.2 Summary of the requirements

This chapter presented the requirements needed for the automatic configuration system operability. The requirements were divided into environmental, configuration and security requirements.

Optiverters are installed permanently outdoors. Due to harsh climate condition and guarantee time of 25 years, the devices must be robust. In addition, the solution must be able to conduct all communication activities within the LOS range of 3 meters for BLE and 40 meters for Wi-Fi. This needed because the configuring solution is a part of their Optiverter technology which could be installed everywhere. For example, on a roof.

Initialization requirements include the following features:

- configuration process must use BLE for communication;
- user must be able to connect to device at any time moment:
- devices must store Wi-Fi AP's SSID and password;
- devices must be able to get information about cluster devices from the server;
- communication with server must be established via Wi-Fi;
- devices must be able to scan the surrounding devices;
- devices must be able to configure near-by cluster devices automatically.

To ensure the data integrity, the security measures must be used. For this part, there are only usage suggestions, because the capability to offer different security protocol is dependent on the SoC selected for this thesis.

In conclusion, the solution needs to use robust devices to conduct reliable, user-friendly and, and safe, in the aspect of data security, configuration processes. Furthermore, it must conduct all previously mentioned functions automatically when new device is included.

4 Selection of communication device

Following chapter describes different wireless modules and selection made from these to use in current master thesis work. The selection is made based on the requirements stated in chapter System requirements.

4.1 Overview of communication devices

This section gives a broad overview of the background, technical capabilities and application areas of the different SoCs that seemed to be suitable for current master's thesis.

4.1.1 Espressif ESP32

ESP32 was developed by Espressif Inc and released on the market in September of 2016. The company states in their hardware design guidelines, that the chip can provide good power and communication performance in a wide variety of applications thanks to integrated Bluetooth 4.2 + BLE and 2,4 GHz Wi-Fi dual mode [6]. ESP32 costs approximately 8 USD [7]. The SoC is presented in Figure 4.

Figure 4. ESP32 SoC [6]

ESP32 is a SOC with Xtensa Dual-Core 32-bit LX6 microprocessor CPU performing at up to 600 Dhrystone millions of instructions per second (DMIPS) and operating at 240 MHz. The chip is designed with TSMC ultra-low power 40 nm technology, operating at temperatures from -40°C to 125°C. SOC is equipped with 448 KB ROM, 520 KB SRAM, 16 KB SRAM in RTC and up to 4x16 MB QSPI flash/SRAM. ESP32 possesses following wireless communication capabilities: Wi-Fi 802.11b/g/n/e/i and Bluetooth

v4.2 BR/EDR and BLE. Furthermore, it has On-board PCB antenna and IPEX connector for external antenna [6].

In addition, it has following peripheral interfaces:

- up to 18 channels of 12-bit SAR ADC;
- 2 × 8-bit DACs;
- 10 × touch sensors;
- temperature sensor;
- 4 × SPI;
- 2 × I²S;
- 2 × I²C;
- 3 × UART;
- 1 SD/SDIO/MMC host,
- 1 slave (SDIO/SPI);
- Ethernet MAC interface with dedicated DMA (direct memory access) and IEEE (Institute of Electrical and Electronics Engineers) 1588 support
- CAN (controller area network) bus 2.0;
- IR (infrared) TX/RX (transmit/receive);
- Motor PWMs (pulse width modulation)
- up to 16 channels LED (light emitting diode) PWMs;
- hall effect sensor;
- ultra-low power analogue pre-amplifier. [6]”

To provide communication security the ESP32 uses IEEE 802.11 standard security features, including WFA (Wi-Fi Alliance), WPA/WPA2 and WAPI (WLAN Authentication and Privacy Infrastructure). Furthermore, it uses secure boot, flash encryption, 1024-bit OTP (One Time Programmable) and cryptographic hardware acceleration: AES, SHA-2 (Secure Hash Algorithm), RSA (Rivest-Shamir-Adleman public-key cryptosystems), ECC (elliptic curve cryptography), Random Number Generator (RNG) [6].

The function block diagram is described on Figure 5.

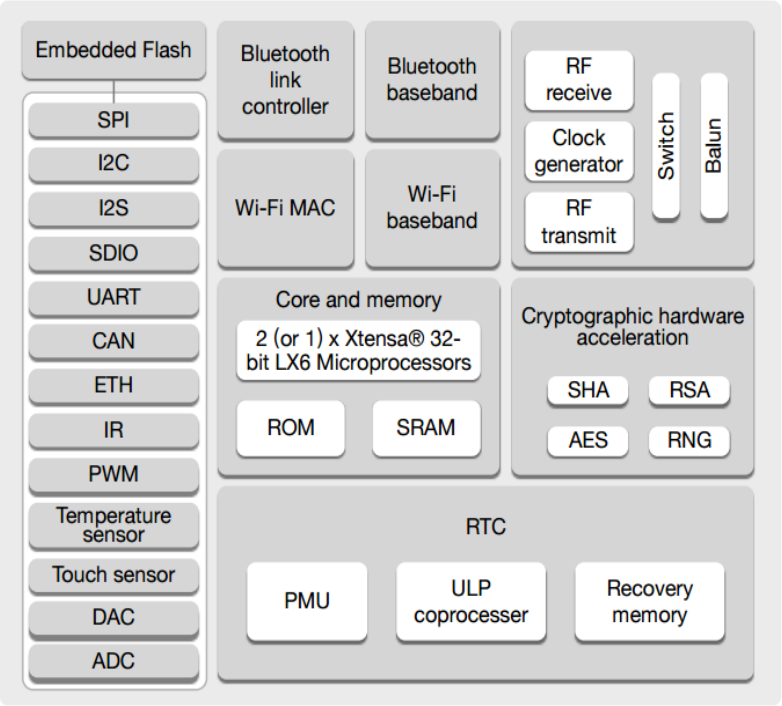


Figure 5. ESP32 function block diagram [6]

As mentioned, the ESP32 has many built in features which makes it very versatile. For example it can be used as a generic low-power IoT sensor hub, generic low-power IoT logger, video streamer from camera, Over The Top (OTT) device, internet music player, audio streaming device, communication unit of Wi-Fi enabled toy, Wi-Fi enabled speech recognition device, communication unit of audio headset, communication unit of smart power plug, communication unit of home automation device, communication unit of mesh network node, communication unit of industrial wireless control, communication unit of baby monitor, communication unit of wearable electronics, Wi-

Fi location-aware device, security ID tag, proximity and movement monitoring trigger device or temperature sensing logger [6].

4.1.2 Atmel ATWINC3400

ATWINC3400 was developed by Atmel and released to the market in July of 2015. The company states in their datasheet, that the chip is an add-on to existing MCU solutions optimized for low-power mobile applications. Wi-Fi and BLE 4.0 capabilities are used through UART or SPI to Wi-Fi interface [8]. In addition, the chip costs approximately 32 USD. The SoC is presented in Figure 6 [9].

Figure 6. ATWINC3400 SoC [8]

ATWINC3400 is designed using CMOS 65 nm technology, operating at temperatures from -40°C to 85°C. It supports single stream 802.11n mode in 2.4GHz ISM band which provides the PHY rate up to 72 Mbps (megabits per second). Chip offers a fully integrated power amplifier, LNA, integrated PA and T/R switch, and power management capability. In addition, the chip features an on-chip microcontroller, integrated flash memory and peripherals such as 1xUART, 1xSPI, 1xI^2C, and SDIO. Furthermore, ATWINC3000 provides variety of network options such as TCP, UDP, DHCP, ARP, HTTP, SSL, and DNS and support IEEE 802.11 WEP, WPA, WPA2 and China WAPI security. The function block diagram of ATWINC3400 is described on Figure 7 [8].

ATWINC3400 is used in IoT applications [8].

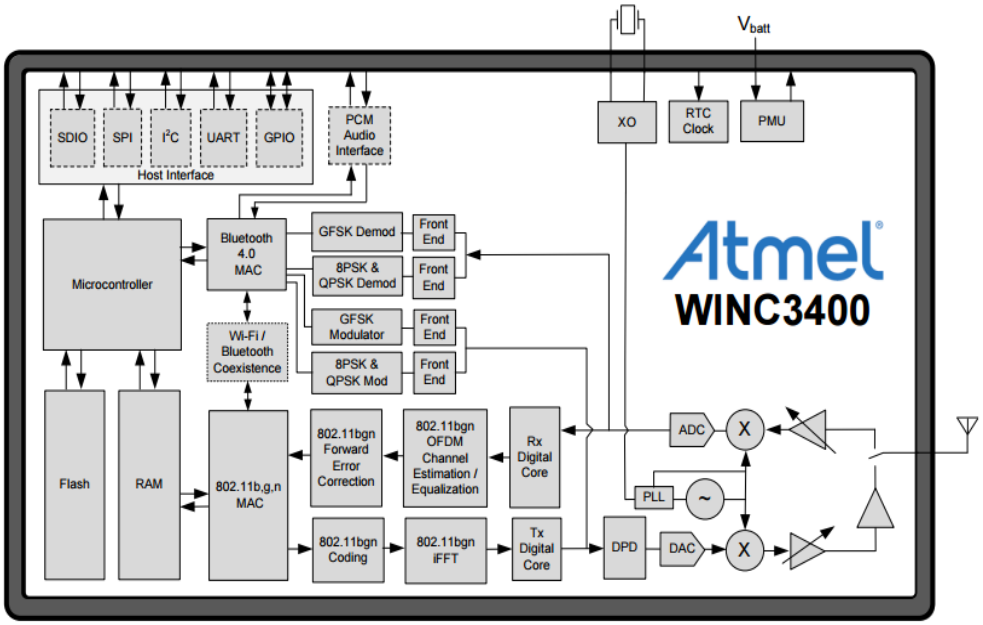


Figure 7. ATWINC3400 block diagram [8]

4.1.3 Texas Instruments WL1835MOD

WL1835MOD was developed by Texas Instruments Incorporated and released to the market in August of 2014. The company states in their datasheet, that the add-on chip provides high throughput and extended range using Wi-Fi and Bluetooth 4.1 [10]. Chip costs approximately 23 USD. The chip is presented in Figure 8 [11].

Figure 8. WL1835MOD chip [10]

WL1835MOD is a 2.4-GHz module, two antenna solution with drivers for high-level operating systems such as Linux® and Android™. The chip operates in temperature range between –20°C to 70°C. It features following Wi-Fi capabilities: 20- and 40-MHz SISO and 20-MHz 2 × 2 MIMO at 2.4 GHz for throughput of 80 Mbps for TCP and 100 Mbps for UDP. In addition, it has Bluetooth 4.1 compliance and CSA2 support and Host Controller Interface (HCI) for transport over UART [10].

The function block diagram of W2CBW0015 is described on Figure 9 [10].

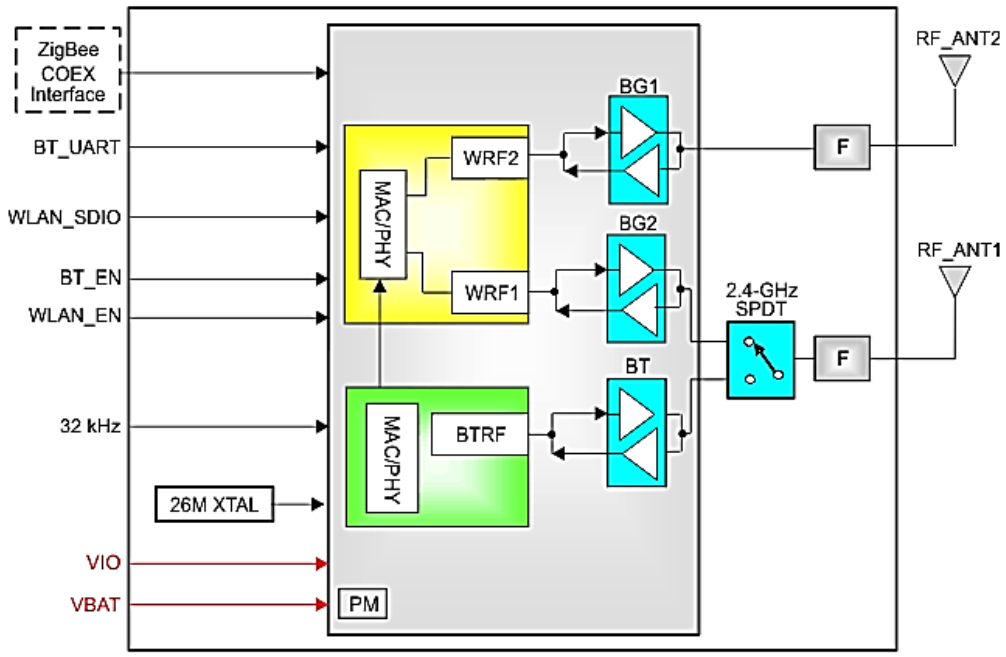


Figure 9. WL1835MOD functional diagram [10]

The application areas of WL1835MOD are listed as follows: Internet of Things (IoT), industrial and home automation, multimedia, smart gateway and metering, home electronics, video conferencing, home appliances and white goods, video camera and security [10].

4.1.4 Wi2Wi W2CBW0015

W2CBW0015 was developed by Wi2Wi Inc. and released to the market in May of 2013. The company states in their datasheet, that the chip provides an easy and flexible way to add Wi-Fi and Bluetooth 3.0+HS capabilities to devices and appliances [12]. Chip costs approximately 25 USD [13]. The chip is presented in Figure 10.

Figure 10. W2CBW0015 chip [12]

W2CBW0015 is a single-band (2.4GHz) Wi-Fi and Bluetooth 3.0+HS multi-band module with an operating temperature range from -30°C to 85°C. Chip includes RF front-end, PA, crystal, switch, integrated MAC, baseband, filter and EEPROM for calibration data and MAC address storage. In addition, it has SDIO 2.0 host interface for 32 bit processors running Linux, Android and Windows operating systems [12].

The function block diagram of W2CBW0015 is described on Figure 11 [12].

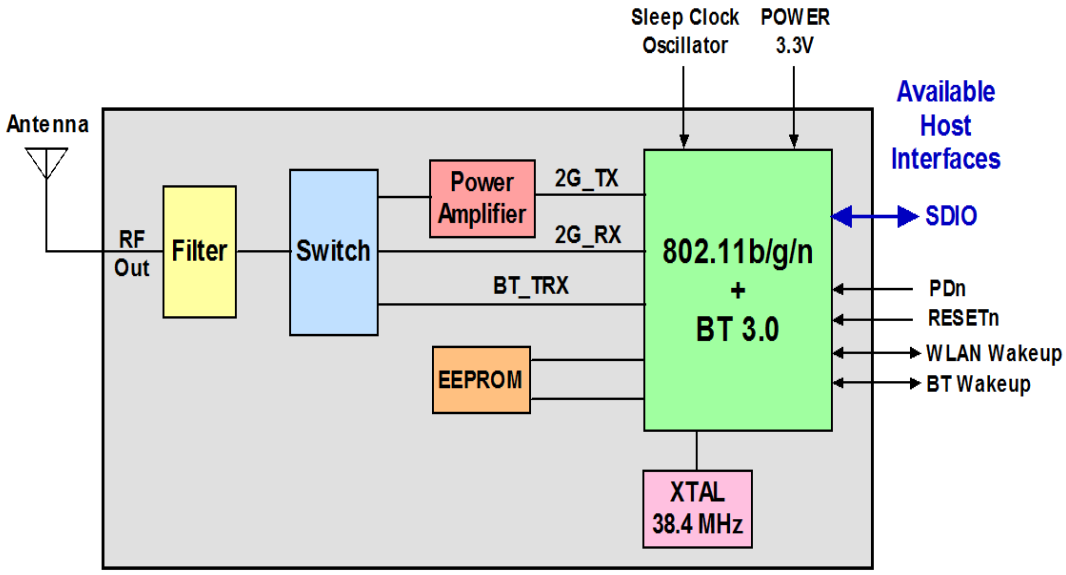


Figure 11. W2CBW0015 block diagram [12]

The application areas of W2CBW0015 are listed as follows: IoT (Internet of Things), IIoT (Industrial IoT) / M2M (Machine-to-Machine), imaging platforms (printer, digital camera), internet enabled consumer devices, video streaming (DTV, DVD/Blu-ray players), video conferencing, Vo-Fi (Voice over Wi-Fi), Wi-Fi enabled security cameras, home audio/video systems, hands-free audios, automotive aftermarket, warehousing and logistics handhelds, medical imaging and monitoring equipment, gaming platforms, mobile routers (Mi-Fi) or mobile hotspot, smart homes, and smart energy grid [12].

4.2 Selection

This section explains the motivation for selecting a System-on-a-Chip (SoC) to use it in device automatic configuration system. The data gathered from chapter Selection of

communication device section Overview of communication device is presented in Table 1.

Table 1. System-on-a-Chip selection

Chip	ESP32	ATWINC3400	WL1835MOD	W2CBW0015
Add-on chip	No	Yes	Yes	Yes
Cost (USD)	8	32	25	23
Bluetooth version	4.2	4.0	3.0 + HS	4.1
Https network option	Yes	No	No information	No information
Communication security	WFA, WPA/WPA2, WAPI, secure boot, flash encryption, 1024-bit OTP, AES, SHA-2, RSA, ECC, Random Number Generator (RNG).	WEP, WPA, WPA2 China WAPI	No information	WEP, TKIP, WPA/WPA2, AES
Temperature range (-40 .. 80)	-40 .. 125	-40 .. 85	-20 .. 70	-30 .. 85
Peripherals	32	No	No	No

The evaluation process begins with the estimation of the total price of one communication unit which is important because the automatic configuration system must be a low cost IoT solution.

Before the comparison of device prices, there is a necessity to evaluate whether the SoC can operate by itself without needing additional MCU for the main operability. The information from the datasheets suggest that only ESP32 is capable to run the processes needed for the operable system. The price of other SoCs will increase because they need

additional processing capability. In this light, the add-on chips are not as good as fully integrated chips.

By comparing the approximate cost, we can see that ESP32 is significantly cheaper than other SoCs which makes it the most usable.

Secondly, the system needs to use BLE which is implemented in Bluetooth versions 4.0, 4.1, 4.2 and 5.0. ESP32 ATWINC3400 and W2CBW0015 are suitable for the task. Because ESP32 has Bluetooth 4.2 makes it more beneficial for the thesis work in the sense of better data transmission capability, energy consumption and data security. More information available in Appendix 1 – Bluetooth Core Specification versions.

Thirdly, the data security measures must be evaluated because it is a critical part in IoT solutions. ESP32, ATWINC3400 and W2CBW0015 provides communication security. Since ESP32 uses newer security protocols it is reasonable to use it.

Fourthly, the chips have environmental temperature requirements because they are used outdoors for a long period of time. Approximately 10-20 years. The temperature requirements for this thesis are set in the range of -40°C to 80°C due to the harsh environmental condition in northern hemisphere. From this we can see that ESP32 and ATWINC3400 are both suitable for this task. ESP32 has an advantage over ATWINC3400 because it operates in larger temperature range.

Finally, the peripherals must be considered to ease the implementation of the automatic configuring system in different application areas. In this field, the ESP32 offers to most versatile selection of peripherals.

In addition, unlike others, ESP32 offers FreeRTOS (free real-time operating system) capability, which makes it possible to transport already developed functionality easily.

As the analysis shows the ESP32 is the most suitable SoC for the thesis work. It provides the necessary processing power and robustness with a minimal cost. In addition, it has many peripherals which could be used in further development if needed.

4.3 Summary of the device selection

This chapter introduced several platforms on which the system could be developed. ESP32, ATWIN3400, WL1835MOD and W2CBW0015 were the devices under consideration.

From the analysis, ESP32 had the best features at minimal cost. Therefore, the selection fell to this SOC.

In addition, Ubik Solutions used the Espressif ESP8266 as their previous platform which is a predecessor to ESP32. This feature makes it easy to transport already developed functionalities to ESP32.

5 Automatic configuration solution: developed method

This chapter is the bulk of this thesis work and covers the architecture of automatic configuration solution and reliability analysis, used programming language and IDE, mobile application, server, and the software developed for the solution.

5.1 System architecture and reliability analysis

Apart from other systems presented in chapter Configuration solutions, the proposed system uses BLE to establish connection for configuration processes.

The solution is visualized in Figure 12. A BLE device, such as smartphone, is used to establish the initial connection.

Connection is established with a cluster device which is a device belonging to one specific user. The devices are listed in server and their data can be viewed and added from computer interface or mobile device application.

Once configuration process is completed via BLE, the communication device will connect to selected Wi-Fi AP.

After connection with Wi-Fi AP is established, the configured device requests a list of cluster devices from the server over https.

It then uses the obtained list to configure nearby cluster devices.

Configuration process is ended when all the devices are capable of transmitting data to the server.

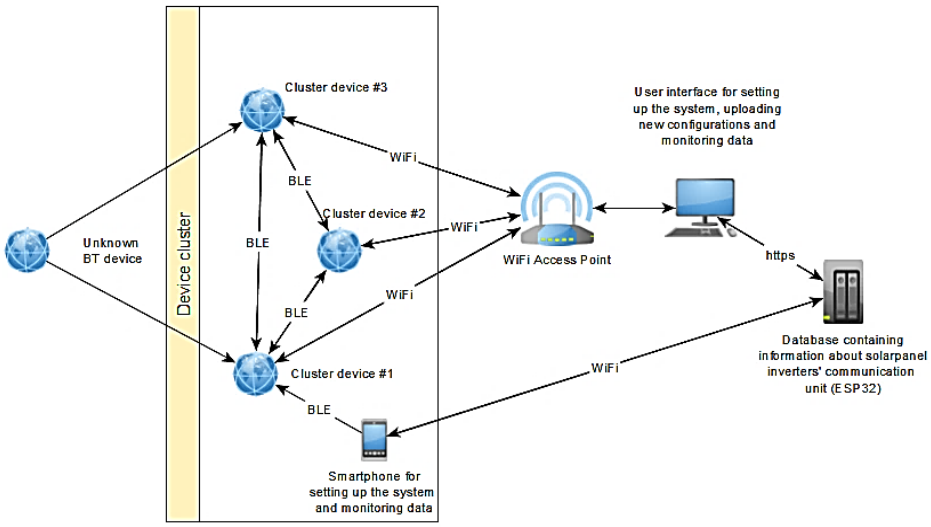


Figure 12. Main concept of automatic configuration solution

Following sections describes system architecture, analysis of reliability issues, advantages and limitations of this technology.

5.1.1 System architecture

As mentioned the proposed software system is intended to automatically configure all devices within predefined cluster via BLE. Objective is to connect all cluster devices to the server via Wi-Fi through the Wi-Fi AP. Maximum number of devices for thesis work is limited to 30 units to prevent Wi-Fi router malfunctions due to large demand.

To give a better overview of the solution, the process description is presented in Figure 13.

The configuration processes are described as follows:

1. User connects to her/his cluster device via BLE device and configures it to access the correct Wi-Fi AP.

2. Cluster device connects to Wi-Fi AP via Wi-Fi.

3. Cluster device sends a request to receive cluster devices from the server.

4. The server returns cluster list data to the cluster device.

5. Cluster device scans for near-by devices. Once found, it checks whether the device is a cluster device. If it is, the connection is established.

6. Unconfigured or not updated cluster device sends a confirmation information.

7. Found device is configured if the confirmation information states that the device is not configured or updated.

Connection with the unknown Bluetooth device is not established because it does not belong to cluster list received from the server.

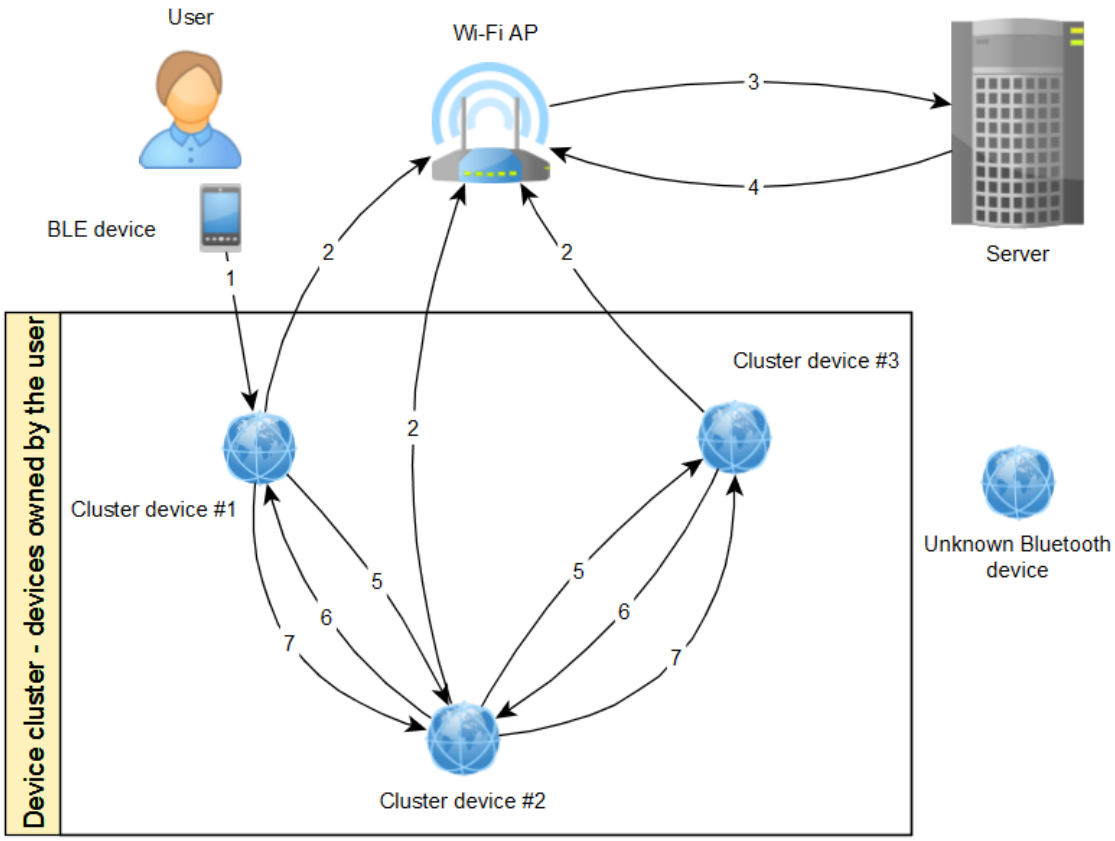
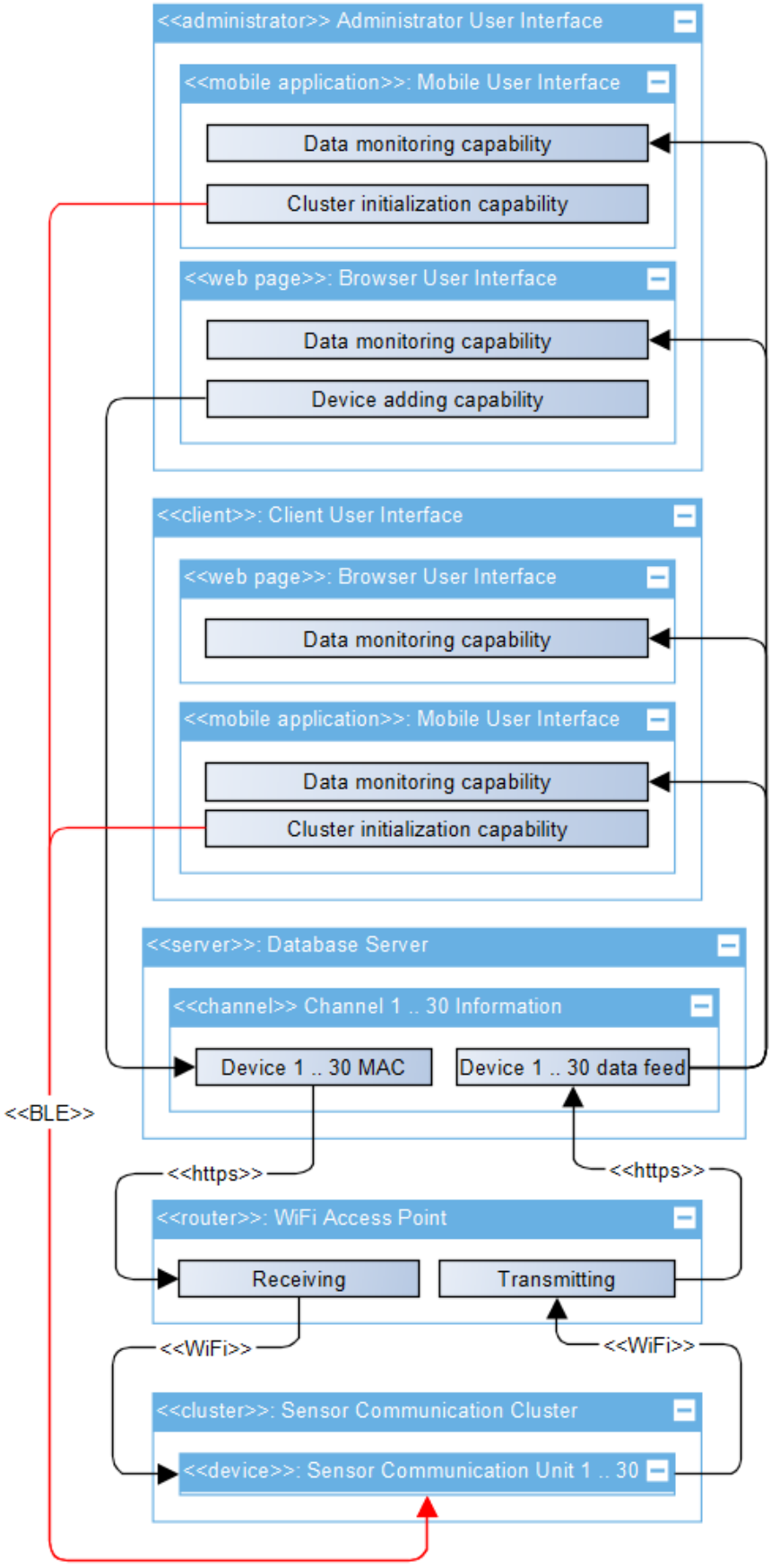


Figure 13. Process description

To go more specific, the deployment diagram to visualize the solution, can be found in Figure 14.


<<administrator>> Administrator User Interface
<<mobile application>>: Mobile User Interface
Data monitoring capability
Cluster initialization capability
<<web page>>: Browser User Interface
Data monitoring capability
Device adding capability
<<client>>: Client User Interface
<<web page>>: Browser User Interface
Data monitoring capability
<<mobile application>>: Mobile User Interface
Data monitoring capability
Cluster initialization capability
<<server>>: Database Server
<<channel>> Channel 1 .. 30 Information
Device 1 .. 30 MAC
Device 1 .. 30 data feed
<<BLE>>
<<https>>
<<https>>
<<router>>: WiFi Access Point
Receiving
Transmitting
<<WiFi>>
<<WiFi>>
<<cluster>>: Sensor Communication Cluster
<<device>>: Sensor Communication Unit 1 .. 30

Starting from the bottom to the top, the Figure 14 demonstrates that the sensor communication cluster may contain up to 30 sensors. Every communication unit advertises and scans information to and from near-by devices. Also, devices receive MAC addresses of other devices, which are in the cluster from server and transmit information feed. The feed contains vital information about devices operability and data from input/output pins. Sensor communication with server is conducted through the Wi-Fi AP via Wi-Fi.

Server provides storage for incoming data feed and MAC address of the corresponding device of each channel. Since the front-end development is not part of the thesis, the MathWorks Thingspeak server capabilities is used for initial configuration. One channel describes the data of one device and holds all feed information. Administrator can add and remove channels from the cluster through admin user interface. Apart from that feature the administrator and client interfaces are rather similar. Both have a capability to monitor the feed sent from devices and conduct initial configuration for the cluster. Although last functionality is possible only when using BLE enabled device such as smartphone or tablet. [14]

Operations of device software can be divided into two groups: continuous and discontinuous. Continuous operations are described on activity diagrams Figure 15 and discontinuous on Figure 15 and Figure 16.

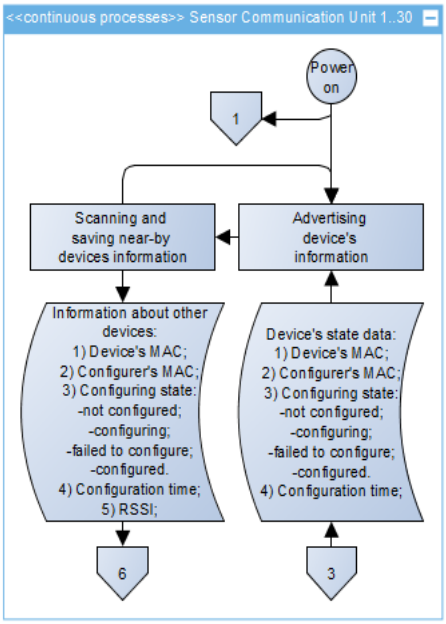


Figure 15. Activity diagram for continuous operations

Continuous processes, seen on Figure 15, are active the entire time when power is on. In this case, the device advertises its' configuration and configurer's information via BLE in order to notify other devices about the advertiser's state. Besides advertising, the device is also scanning via BLE to find and save MAC addresses, RSSI-s (received signal strength indication), configuration and configurer's information of all devices in the proximity. This data is later used in initialization processes.

Discontinuous processes, seen on Figure 16 and Figure 17, are meant to be active only once per one work cycle. Cycle ends when power is turned off. The process could take place more than once when the SSID and password for the Wi-Fi AP needs to be updated.

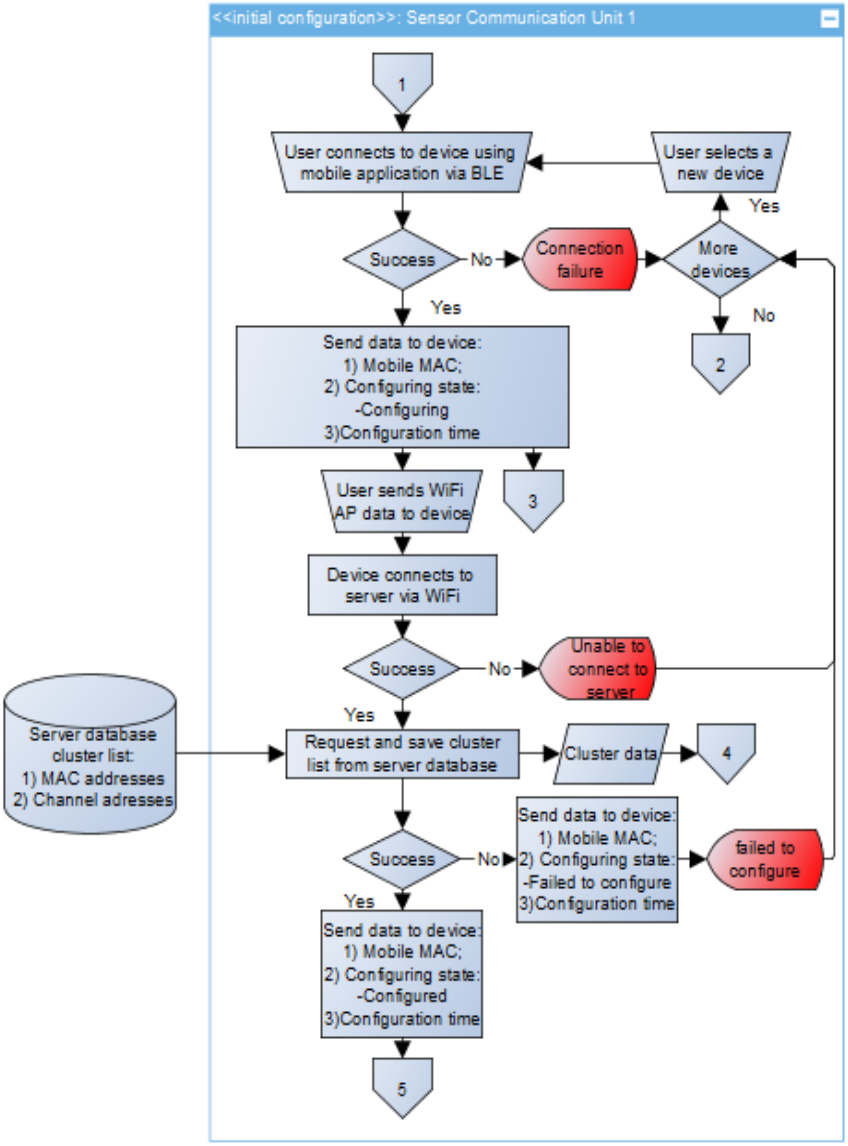


Figure 16. Activity diagram for discontinuous operations - initial configuration

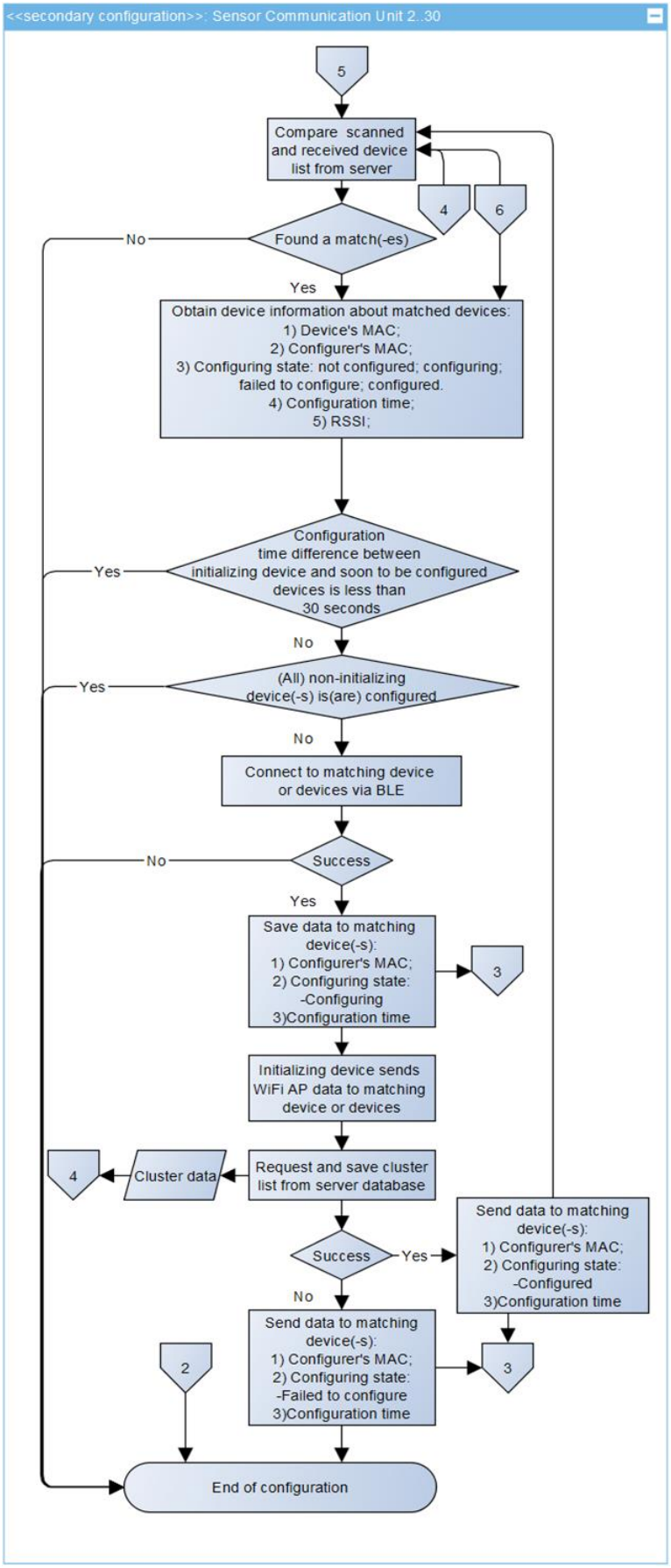


Figure 17. Activity diagram for discontinuous operations - secondary configuration

In case of initial configuration, one device is configured by client or administrator using smartphone or tablet via BLE. Wi-Fi AP's SSID and password is sent from the mobile application to device. Since the front-end development is not a part of the thesis, the EspBlufi mobile application is used for the initial configuration. The device changes its' configuration and configurer's information when it connects to the server via Wi-Fi successfully or fails to do so. If device is connected, then it requests a data packet from the server about friendly devices in the cluster. Secondary configuration starts after the cluster list is received and saved successfully.

Secondary configuration implies that the Wi-Fi AP data is sent between devices themselves and does not need human assistance any more. The cluster data saved in firstly configured device is compared with the ones captured from the near-by advertising devices. If a device recognizes another friendly unconfigured or not updated device, it automatically connects to it and sends Wi-Fi AP data.

In case of renewing the Wi-Fi AP data the process is carried out similarly to initial configuration. The decision, whether to update other devices in the cluster is determined by the configuration time. If configuration time between the configuring and under configuration device is more than 30 seconds the update configuration process is initiated.

Configuration process ends when all devices configuration information does not require reconfiguration and are able to access the Wi-Fi AP to receive and transmit data.

5.1.2 Analysis of reliability issues

According to IEEE Recommended Practice on Software Reliability [15], it is necessary to conduct analysis based on previously described fault tolerant system to determine where the defects probably lay and which are the weak areas of the software system. Reliability analysis provides an overview, that can be used to find possible solutions for reliability issues. The failure in discussed fault tolerant system is inability to connect devices automatically to the server.

Advertisement process is a continuously working activity, which is vital for other devices to find the advertiser. Automatic connection events might not take place, if the data in the advertisement packet is incorrect or memory location is not available.

System must be able to inform the user, if false or no data is sent. Also, the physical layer of BLE must be operable always to transmit data.

Similarly, to advertising, the scanning processes is also continuously working. This process is vital to find nearby sensor communication units in order to compare them with devices from cluster list, which is obtained from server. Automatic connection events might not take place, if memory location for scanned devices is not available. System must be able to inform when false or no data is received. Also, the physical layer of BLE must be operable always to receive data.

Running processes continuously is consuming a lot of power. In principle, the advertising and scanning events does not need to work all the time, because of two reasons. Advertising is only needed during initial configuration and update processes. Scanning is needed until all the cluster devices are configured. In future work, this flaw could be eliminated by sending the information from server side to start configuring or updating the devices. After devices receive the info, they will start advertising and scanning.

User accessing a device with smartphone or tablet via BLE is the feature, which is the basis of initiating the automatic configuration system. The connection creation between mobile device and sensor communication unit must be secured. System must be able to inform when pairing is successful or failed. The physical layer of BLE must be operable at all times to allow pairing.

User's ability to send Wi-Fi AP SSID and password to device via BLE through the mobile application is another important feature of the system. The Wi-Fi AP data must be encrypted, have memory holding SSID and password available at all times and physical layer of BLE operable. System must be able to inform when SSID and password are incorrect or not sent.

Wi-Fi module operability and system messages about current connection state must be granted when device connects to correct Wi-Fi AP.

After the connection is established, the device sends cluster data request to server. This request must be encrypted and correctly presented in order to get cluster data. System must be able to inform when request is sent.

If server accepts the request, the device receives cluster data from the server. The data received must be encrypted, received properly, and saved to a correct memory location. System must be able to inform which data was received.

It is important for a device to compare scanned and received device lists to find matches in order to start automatically configuring the cluster devices. For this operation, the memory locations must be available, data in them valid and information about the comparing process presented.

One device connecting to another device needs a secured connection and operational BLE physical layer. In addition, initializing device must have data about other device's previous configuration time for updating. System must be able to inform about the connection state and configuration time. Device must be able to change its' advertising data upon different activities. For this the data must be altered in correct memory location.

Information gathered provides the basis to find best ways to mitigate discussed reliability issues.

5.1.3 Applicable reliability techniques and methods

This section introduces different methods how to solve previously discussed problematic areas. Single-version techniques are used, because the project is not about safety critical system. Therefore, not needing the multi-version techniques, which uses at least two versions of the same software module to meet the requirements of design diversity. Single version techniques simply add functionalities that are unnecessary in a fault-free environment to a single software module [16].

By using single-version techniques, software is modified to be able to detect a fault, isolate it and prevent the propagation of its effect throughout the system. Firstly, the detecting process is viewed, which is the basis of recovery [16].

Single-version techniques use different acceptance tests to detect faults. The result of a program is subjected to a test and program will continue its execution only if the result passes the test. Otherwise it will indicate a fault. An effective test can be calculated in a simple manner and its' criteria are derived independently of the program application [16].

To increase reliability following techniques are used: timing checks, coding checks, reversal checks, reasonableness checks and structural checks. These are described briefly in following sections [16].

Timing checks can be applied to systems with timing constrains. These constrains are a foundation to checks that indicate a deviation from the correct behaviour [16].

Coding checks can be applied to systems whose data can be encoded using information redundancy techniques. Cyclic redundancy checks are applicable when the information is transported without changing its' content. Arithmetic codes are applicable to detect arithmetic operation errors. In addition, some systems allow reversal checks which compare the actual inputs of the system with the computed ones and gives a fault upon a disagreement [16].

Reasonableness checks use semantic properties of data to detect fault and structural checks are based on known properties of data structures [16].

Fault containment is a necessary procedure after fault detection. It is achieved in software altering the structure of the system and by putting a set of restrictions defining which actions are permissible within the system. There are four techniques for fault containment: modularization, partitioning, system closure and atomic actions. All of these are described in following sections [16].

Modularization is used to prevent the propagation of faults. It is done by limiting the amount of communication between modules to carefully monitored messages and by eliminating shared resources. "Before performing modularization, visibility and connectivity parameters are examined to determine which module possesses highest potential to cause system failure. Visibility of a module is characterized by the set of modules that may be invoked directly or indirectly by the module. Connectivity of a module is described by the set of modules that may be invoked directly or used by the module [16]."

Partitioning the modular hierarchy of a software architecture in horizontal or vertical dimensions creates the isolation between functionally independent modules. "Horizontal partitioning separates the major software functions into independent branches. The execution of the functions and the communication between them is done using control modules. Vertical partitioning distributes the control and processing function in a top-

down hierarchy. High level modules normally focus on control functions, while low-level modules perform processing [16].”

System closure bases on principle that no action is permissible unless explicitly authorized. The system environment has many restrictions and strict control which makes the processes highly visible. The purpose of this containment technique is to locate and remove any faults by viewing interactions between the elements of the system [16].

Atomic actions define interactions between system components. It is an activity in which the modules communicate exclusively with each other among a group and during that time there is no interaction with the rest of the system. The results are correct if atomic action terminates normally. Otherwise the detected fault affects only the participating components. This enables a possibility to define fault containment area and limit fault recovery to atomic action components [16].

Fault recovery from the faulty state to regain operational status is the next step after fault is detected and contained. Fault detection and containment mechanisms must be implemented properly because the information about fault containment region is crucial for the design of fault recovery mechanism. There are four techniques for fault recovery: exception handling, checkpoint and restart, process pairs, and data diversity. These are described in following sections [16].

“Exception handling is the interruption of normal operation to handle abnormal responses.” The exceptions which are triggered by events in a software module are classified into three groups [16].

1) Interface exceptions are initiated when an invalid service request is detected. This exception must be handled by the module that requested the service [16].
2) Local exceptions are initiated when fault within its internal operations is detected by fault detection mechanism. This exception must be handled by the faulty module [16].
3) Failure exceptions are initiated when fault recovery mechanism is unable to recover successfully. This exception must be handled by the system [16].

Checkpoint and restart, also known as backward error recovery technique bases on method where a snapshot of the system state is taken before fault occurs and used to recover the system after restart. Design faults, activated by some unexpected input sequence, are the most common software faults. These resemble hardware intermittent faults and as in hardware, a simple module restart is in many cases enough to complete its' execution [16].

The acceptance test block checks the correctness of the result and works simultaneously with the module's executing a program. Upon fault detection, the "retry" signal is sent to module to use checkpoint state stored in the memory for re-initialization. Checkpoints can be static or dynamic [16].

Static checkpoint stores a single snapshot of the system state in the memory at the beginning of the program execution. If a fault is detected at the output of the module, the system returns to previous state and starts the execution from the beginning. Dynamic checkpoints store snapshots of the system state in the memory at various points during the execution. Upon detecting a fault, the system goes back to the last checkpoint and continues the execution. Prior to creating checkpoints, fault detection checks need to be embedded in the code and executed [16].

"A number of factors influence the efficiency of check-pointing, including execution requirements, the interval between checkpoints, fault activation rate and overhead associated with creating fault detection checks, checkpoints, recovery, etc. In static approach, the expected time to complete the execution grows exponentially with the execution requirements. Therefore, static check-pointing is effective only if the processing requirement is relatively small. In dynamic approach, it is possible to achieve linear increase in execution time as the processing requirements grow. There are three strategies for dynamic placing of checkpoints: [16]"

1) "Equidistant, which places checkpoints at deterministic fixed time intervals. The time between checkpoints is chosen depending on the expected fault rate [16]."
2) "Modular, which places checkpoints at the end of the sub-modules in a module, after the fault detection checks for the sub-module are completed. The execution time depends on the distribution of the sub-modules and expected fault rate [16]."

3) “Random, placing checkpoints at random [16].”

Checkpoint and restart mechanism is used because it is conceptually simple, independent of the damage caused by a fault, applicable to unanticipated faults and general enough to be used at multiple levels in a system. Main disadvantage of restart recovery is that non-recoverable actions exist in some systems, which cannot be solved by reloading the state and restarting the system. Special treatment is needed to recover from such actions [16].

“Process pair technique runs two identical versions of the software on separate processors. First the primary processor, Processor 1, is active. It executes the program and sends the checkpoint information to the secondary processor, Processor 2. If a fault is detected, the primary processor is switched off. The secondary processor loads the last checkpoint as its starting state and continues the execution. The Processor 1 executes diagnostic checks off-line. If the fault is non-recoverable, the replacement is performed. After returning to service, the repaired processor becomes secondary processor [16].”

“The main advantage of process pair technique is that the delivery of service continues uninterrupted after the occurrence of the fault. It is therefore suitable for applications requiring high availability [16].”

“Data diversity is a technique aiming to improve the efficiency of checkpoint and restart by using different inputs re-expressions for each retry. Its is based on the observation that software faults are usually input sequence dependent. Therefore, if inputs are re-expressed in a diverse way, it is unlikely that different re-expressions activate the same fault.

There are three basic techniques for data diversity: [16]”

1) “Input data re-expression, where only the input is changed [16].”
2) “Input data re-expression with post-execution adjustment, where the output result also needs to be adjusted in accordance with a given set of rules. For example, if the inputs were re-expressed by encoding them in some code, then the output result is decoded following the decoding rules of the code [16].”

3) “Input data re-expression via decomposition and re-combination, where the input is decomposed into smaller parts and then re-combined after execution to obtain the output result [16].”

Methods to improve software reliability are described in following sections. Reliability considerations include protecting data, protecting system and reporting, protection from hardware failures, ensuring available memory, protecting system states, and protection from inadvertent operations, out-of-sequence commands or data, and environmental effects [15].

Data could be protected by using multiple data storage places, CRC (cycle redundancy checks), comparisons before use and self-checking of software data integrity [15].

System protection and reporting could be granted by sending commands/messages with CRC and handshaking for correctness, checking commands/data before use at destination, and checking of message size, content, stage of operation, etc. [15].

Protection from hardware failures could be solved by adding sensors, actuators and data lines to the system [15].

To ensure memory availability, the background memory checks for memory viability could be conducted [15].

For protecting system states, it is possible to use multiple bits (8, 16, and 32) to indicate any change in critical functions or for sensor and effector identification. Prior to activation, check address and instructions sequence against separate stored data and check if activation is “legal” under current conditions [15].

To grant protection from inadvertent operations, out-of-sequence commands or data, and environmental affects one or more listed processes could be carried out [15]:

1) Implement self-checking of software events and data integrity [15].
2) Implement watchdog timers that monitor the software’s operation and trigger a reset or alternate system [15].
3) Implement fault/failure detection, isolation, and recovery (FDIR) measures for the software [15].

5.1.4 Considered reliability techniques

This section describes the reliability measures which are considered for use in this thesis. It is based on previously discussed sections System architecture and Applicable reliability techniques and methods. Section covers finding suitable software fault recovery, fault containment and fault detection mechanisms.

The software does not need high reliability because it is used for initialization of devices which takes place only few times during the entire working time of the device. If the system fails to configure the devices, the outcome causes only drop in user satisfaction. Installation team spends extra time for configuring and customer access

The software should use checkpoint and restart fault recovery technique because it is the most simple and robust solution to solve occurring problems. Restart and reload of previous working system state is conducted upon system failures. Process pair and data diversity techniques are too complex for this project.

After the system is operable the device must send error report to the server to analyse the root cause of an error and improve the software based on the gathered information.

The checkpoint and restart technique should use dynamic checkpoints because they must be placed also after the beginning of the program execution. Needed when power failure occurs to remember the configuration, advertising and scanning information. Checkpoints must be placed modularly because recovery has to fall back to the last working state of the software module. Thus, excluding random placing. Equidistant placing is also not an option because the software working cycle is not constrained by time.

Software should use modularization fault containment method, because of the simplicity of the software. Modularization is needed to avoid a situation where one part of software creates a failure in the entire system. Since the software uses real-time operating system (RTOS), the atomic actions are used within the functionality. Partitioning is not reasonable to use because all software modules are needed to provide the operability of the system therefore making it hard to sort modules in hierarchy. System closure and atomic actions are not reasonable to use because they add unnecessary complexity to the system and restrict its' usability.

All processes listed in architectural overview need a reporting capability for user to view the exchanged data, software state and information about memory availability and BLE/Wi-Fi physical link operability. Additional measures like multiple data storage places, CRC outside BLE controller, comparisons before use, self-checking of software data integrity and sensors for hardware check are not needed due to the low reliability requirements for the system [15].

5.2 Software development language and tools

5.2.1 Process planning

- yEd Graph Editor Version 3.17 was used to create deployment and activity diagrams for software development process.

5.2.2 Software development

- Software was written in C language using Eclipse Neon.3 version 4.6.3. It was also used to build the developed software and flash it on the ESP32.

- CP2102 USB to TTL connector was used for uploading the code to ESP32 and observing its' activity via UART. The pin connection is presented in Figure 18.

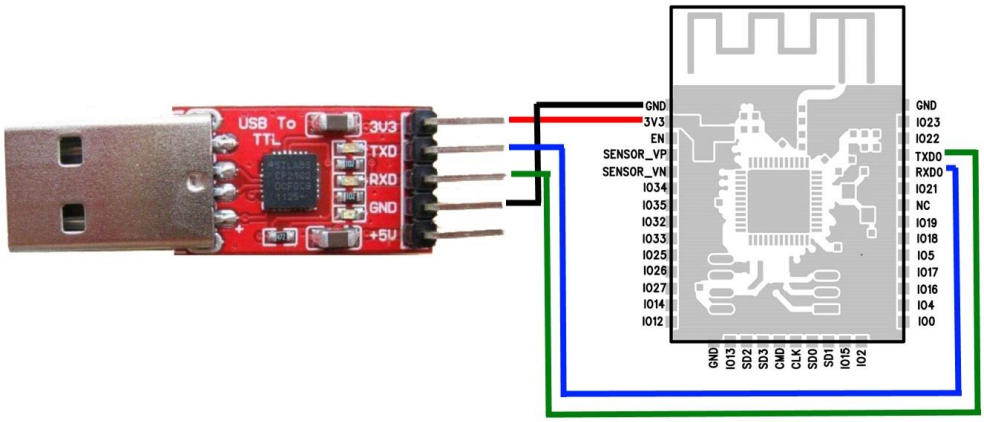


Figure 18. CP2102 USB to TTL connection [17] with ESP32 [6]

- xtensa-esp32-elf cross-compiler binary toolchain GCC version 5.2.0 was used to compile the code going into the ESP32 unit.

- ESP-IDF (The Espressif IoT Development Framework) was used for developing applications based on ESP32.

- Ubuntu 16.04 terminal was used to configure the IDF with the command “make menuconfig” visible in Figure 19

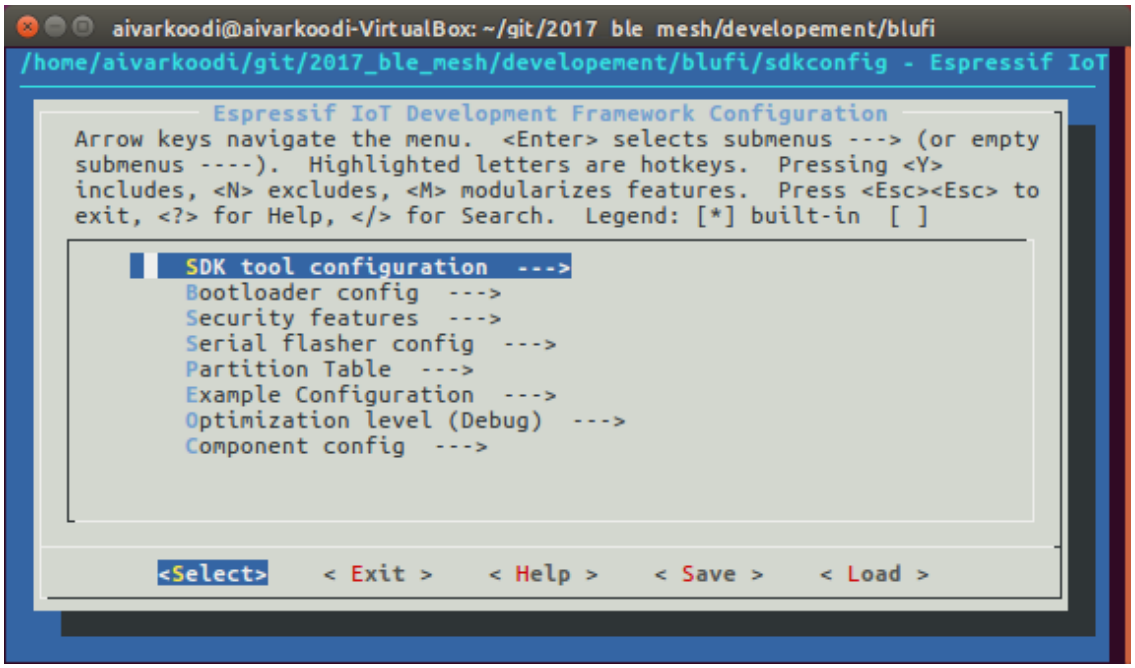


Figure 19. Project menuconfig

5.2.3 Software evaluation

For workability evaluation following tools were used:

- CuteCom 0.22.0 to view the UART debug information;

- EspBlufi mobile application version 1.0.1 to check the functionality of initialization process;

- Wireless Network Watcher v2.05 to check whether the device is connected to correct Wi-Fi AP;

- Postman version 4.10.7 to check the workability of the cloud server;

5.2.4 Server

MathWorks Incorporated ThingSpeak cloud server for IoT data is used to store cluster device information which is used later for verification of cluster devices near of an ESP32. [14]

Once the account in ThingSpeak is created, there is a possibility to create channels for users. As seen in Figure 20, there is one user.

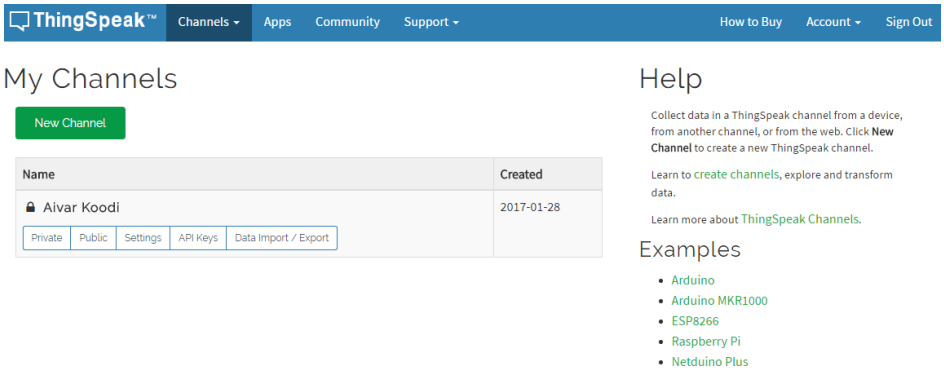


Figure 20. User channel [14]

Under this user's profile, there are several graphs. Each graph represents a device in her or his device cluster. The view is visible in Figure 21.

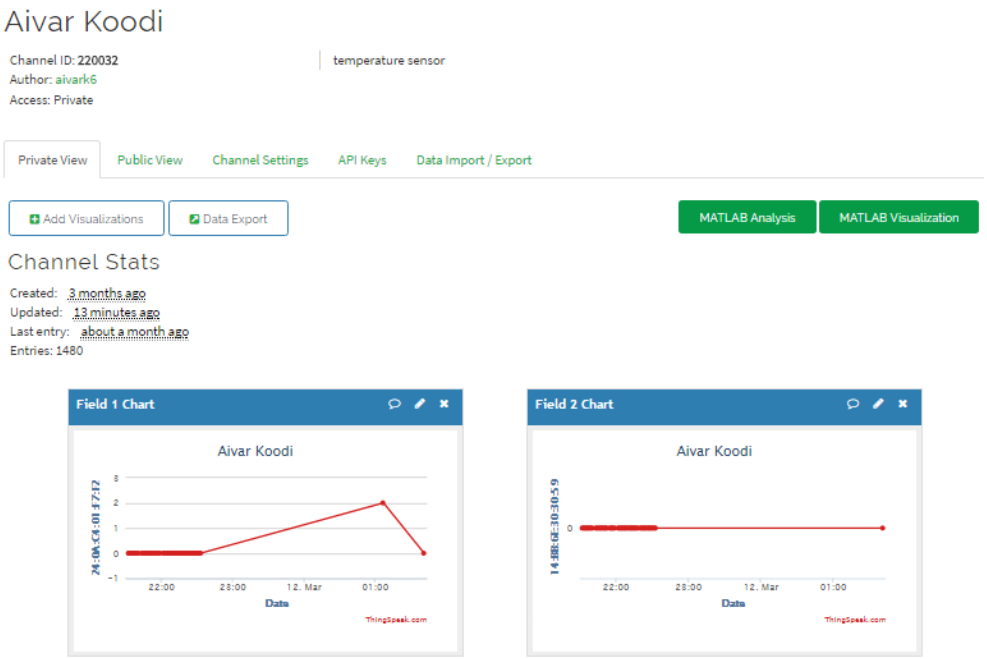


Figure 21. Data feed view [14]

The cluster devices can be added and changed from the Channel Settings tab, visible in Figure 22. The number of cluster devices from the server side is limited to 10, because free version is used in this thesis.

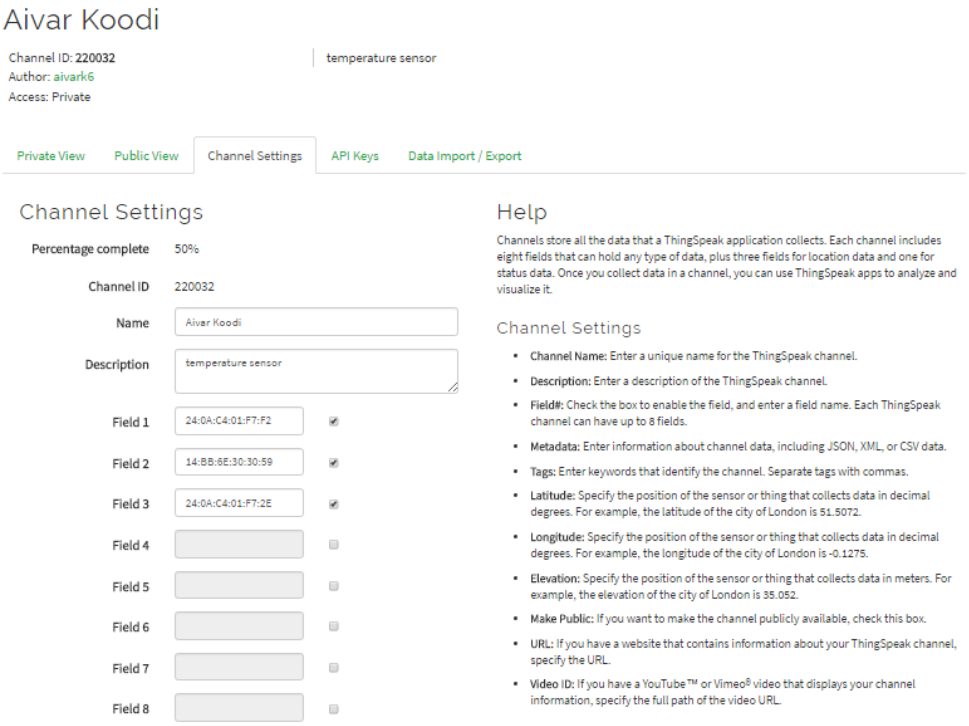


Figure 22. Cluster devices [14]

Cluster data is requested with a following GET command in JSON (JavaScript Object Notation) format seen in Figure 23.

```
https://thingspeak.com/channels/220032/feeds.json?results=0&api_key=WBWA6NBT6EXFS1ZF
```

Figure 23. Https request

The part of the request "*api_key=WBWA6NBT6EXFS1ZF*" indicates to the read API key needed for secure. Write and read keys can be found under API keys tab. Write key is used to send information feed to the server. The tab is visible in Figure 24.

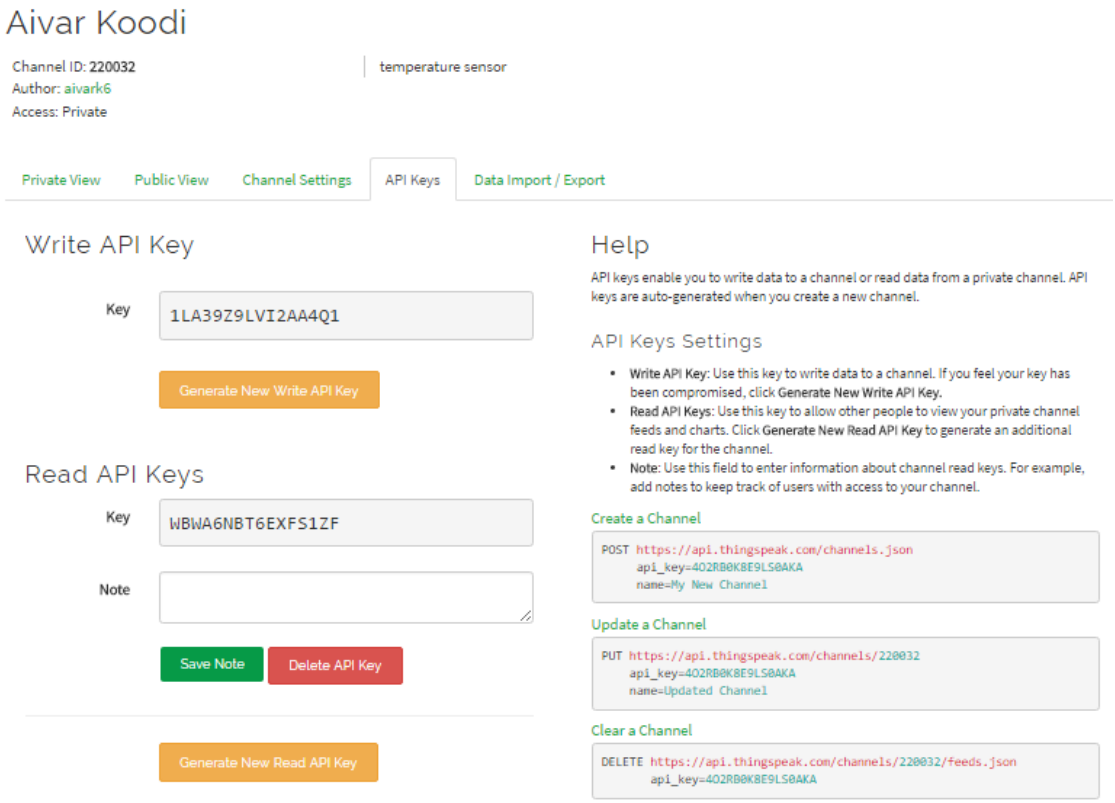


Figure 24. API keys [14]

Received cluster data structure is presented in Figure 25:

```
{
  "channel": {
    "id": unique user identifier,
    "name": "name of the device, company etc.",
    "description": "purpose of the device",
    "latitude": "cluster latitude location",
    "longitude": "cluster longitude location",
    "field1": "MAC of the first device",
    "field2": "MAC of the second device",
    "created_at": "date and time of first appearance",
    "updated_at": "data and time of last update",
    "last_entry_id": last update identifier
  },
  "feeds": [optional data about previous measurements]
}
```

Figure 25. Received cluster data structure

Received cluster data example is presenter in Figure 26.

```
{
  "channel": {
    "id": 220032,
    "name": "ESP32_temperature_sensor",
    "description": "temperature measurement",
    "latitude": "0.0",
    "longitude": "0.0",
    "field1": "24:0A:C4:01:F7:F2",
    "field2": "14:BB:6E:30:30:59",
    "created_at": "2017-01-28T12:07:26Z",
    "updated_at": "2017-03-11T22:41:06Z",
    "last_entry_id": 1480
  },
  "feeds": []
}
```


Figure 26. Received cluster data example

5.3 Software architecture

This section covers the architecture of developed software, including the overview of system components and modularized software, and detailed description of different software modules and their connections. The code can be accessed via a git repository link in Appendix 3 – Developed software.

5.3.1 System components

The system components described in Figure 27, are the basis for the system operability. There are five system components: ESP32 device hardware, ESP32 device drivers, real-time operating system, ESP32 generic middleware and user application code.

User application code holds the developed software. For this master thesis, it contains a software for automatic configuration system.

ESP32 generic middleware provide API structures and functions to application code. It is the link between the user application and ESP32 device drivers.

Real-time operating system kernel provides API structures and functions to application code. It executes the functionality on specific features on ESP32 device hardware. In this thesis, the FreeRTOS is used.

ESP32 device drivers are necessary to execute functions specific features on ESP32 device hardware.

ESP32 device hardware includes all physical features available on the ESP32 System-on-a-Chip.

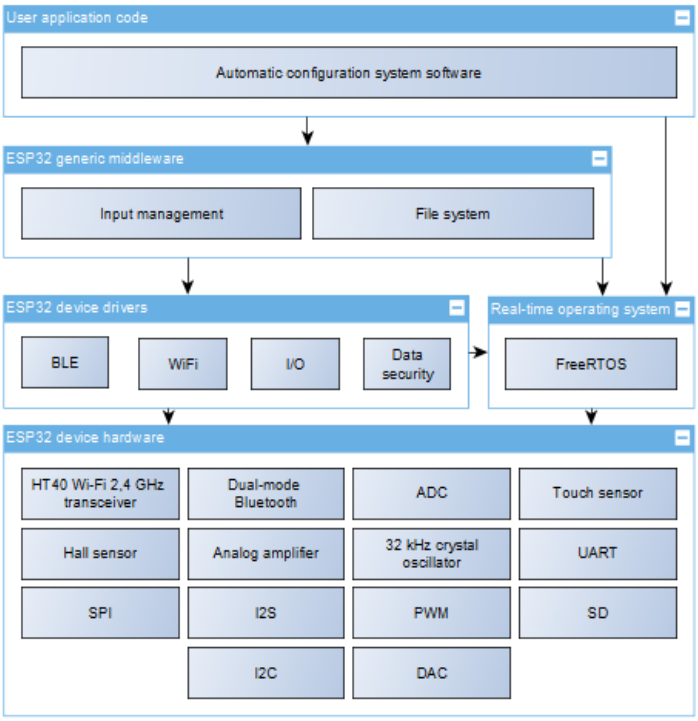


Figure 27. System components

5.3.2 Software flow

The software is modularized and divided in seven modules: initialization, advertise, scan, device configured, connection with Wi-Fi AP, connection with server, and device configuring.

Initialization of Wi-Fi and BLE controller is the first processes that takes place after the circuit is powered. Starting Wi-Fi functionality is necessary to establish a connection with selected Wi-Fi AP. BLE controller is needed to advertise and scan viable information for configuration process. In addition, it is used to create a connection and transmit data between BLE devices.

As soon as the system is initialised, the advertisement event begins. Off-page reference 1 indicates the places where the advertisement process gets its' advertisement data. It

runs for 5 seconds and then proceeds to scan event. During this time, there is a possibility for other devices to establish a connection. Off-page reference 2 directs to operations conducted by ESP32 after connection takes place.

Scan event starts after advertisement event and lasts for 5 seconds. During this time, the device saves all found nearby devices. Off-page reference 3 directs to device comparison functionality which is necessary for automatic configuration.

Initialization, advertisement and scan events are presented in Figure 28.

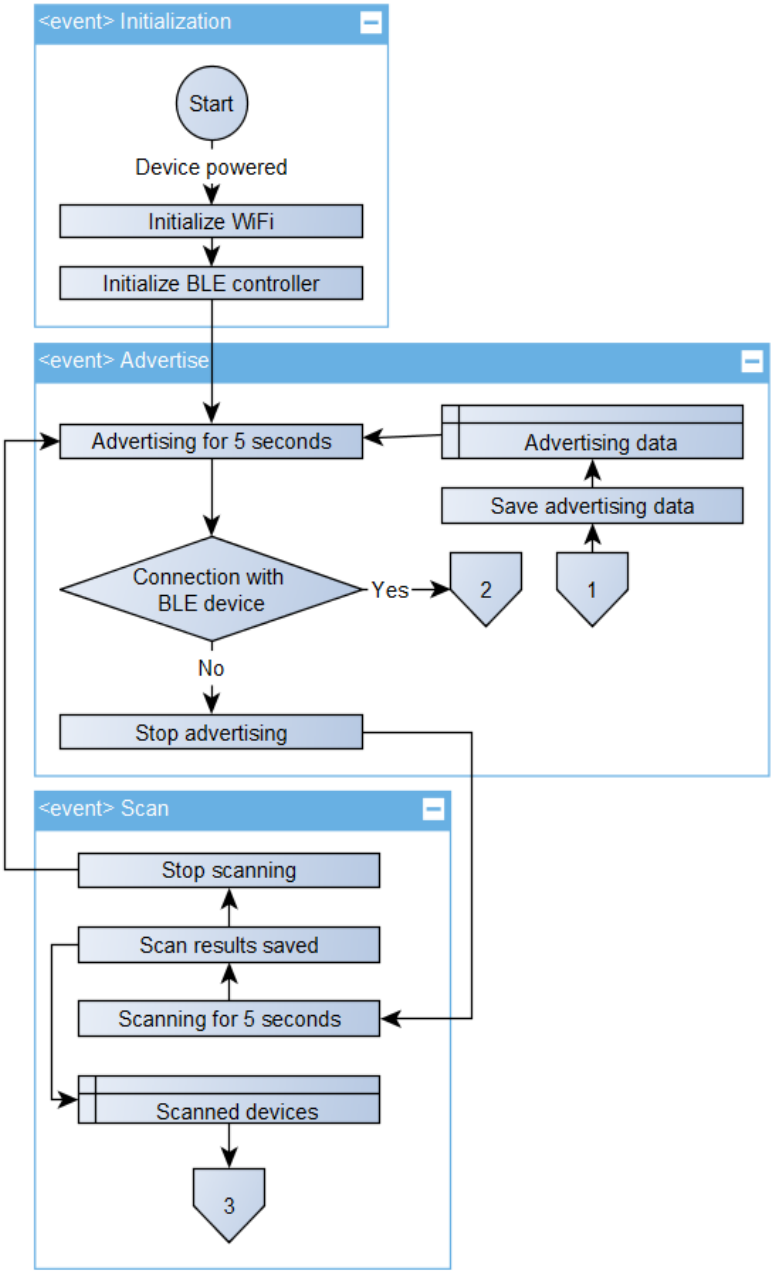


Figure 28. Initialization, advertisement and scan operations

The system may enter device configured event from advertising event. Off-page reference 2 comes from advertising event. The time of last configuration is checked before the main functionality is initiated to prevent endless configuration loops within the cluster while updating the devices. The time is obtained from update timer, which is presented with off-page reference 4 in Figure 29. If last configuration process took place less than 30 seconds ago, the device will return to advertising event. If not, the device configured event will turn off advertising and proceed to send update confirmation to the configuring device. Update configuration is a value which represent the sum of all characters of SSID and password in hexadecimal. Wi-Fi AP SSID and password is obtained if the update confirmation is suitable for configuring device. This data is saved and used in device configuring event. Off-page reference 5 directs to "send Wi-Fi data" function in device configuring event. After data is saved the system goes to advertising event and connection with Wi-Fi AP event.

Using saved Wi-Fi AP data, the device tries to connect to selected Wi-Fi AP. If this is not successful, the device returns to advertising event and waits for the next time it will be configured. If it is successful, the update timer is reset and the software proceeds to connection with server event. Off-page reference 6 directs to connection with server event.

Device configured and connection with Wi-Fi AP events are presented in Figure 29.

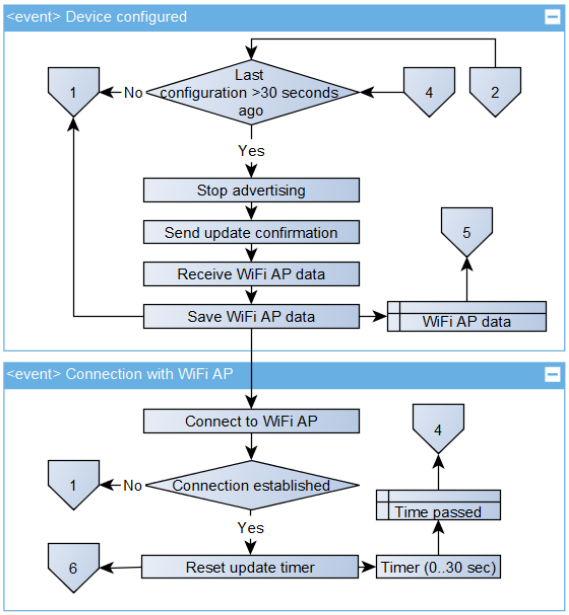


Figure 29. Device configured and connection with Wi-Fi AP operations

Connection with server begins with the check that ensures that the device is connected with Wi-Fi AP. If it is not, the system returns to advertise event. If it is, it will try to connect with server. Upon failed try it will go back to checking the connection with Wi-Fi AP within connection with server event. If the connection with server is established, the https request is written to obtain cluster device list from server. The server replies to that request by sending the data introduced in Software development language and tools. The data is read and parsed to obtain the cluster devices. To find near-by cluster devices a comparison operation is conducted. Previously scanned devices and device list obtained from server is compared. If no unconfigured or not updated devices are found, the system will fall back to advertise event. If there is, the system continues to device configuration event.

Device configuration event starts with the update timer check. If the last configuration took place more than 30 seconds ago, the system will go back to checking the connection with Wi-Fi AP within connection with server event. Otherwise it will stop advertising to establish a connection with unconfigured or not updated device. The device being configured sends the update configuration. This way there is a possibility to estimate whether the configuring device and the device configured share the same password. If they do the connection is immediately closed. Otherwise, the configuring device sends the Wi-Fi AP data to device which under configuring process. Either way the event ends with the termination of connection and falling back to checking the connection with Wi-Fi AP within connection with server event.

Connection with server and device configuring events are presented in Figure 30.

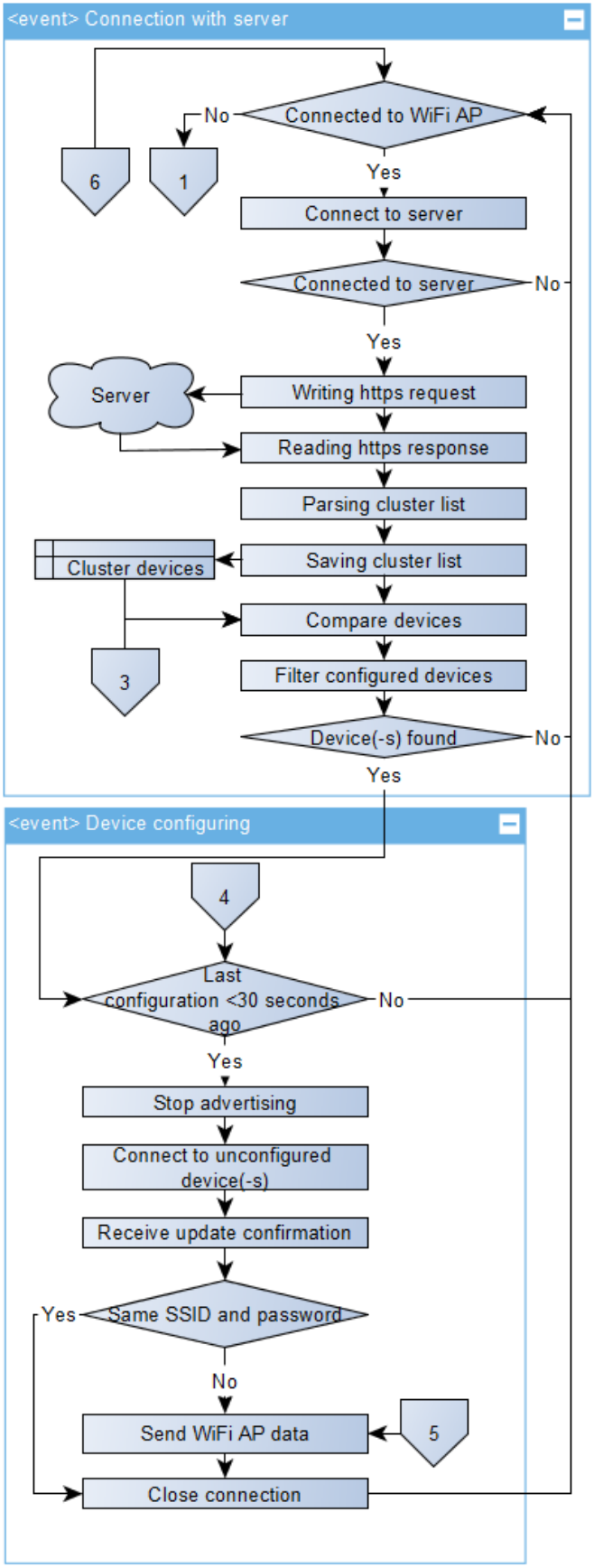


Figure 30. Connection with server and device configuring operations

Some of the events are run continuously such as advertise and scan events. In addition, connection with server event when the device is configured.

Other events are initiated only when the device is being configured or starting to configure others.

The automatic configuration process ends when all found cluster devices are configured successfully to the Wi-Fi AP.

5.4 Summary of the developed method

This chapter contains the system architecture and reliability analysis, and software architecture.

System architecture stated the outline of the workability, that is needed for software development. The architecture considers of all requirements stated in chapter Configuration requirements. The system is designed to be accessed and configured via user's BLE device. After configuration process by user, the system will automatically configure all other cluster devices in the proximity. Configuration process ends when all cluster devices are configured to access selected Wi-Fi AP.

Reliability analysis was conducted to find potential hazards within the system which could lead to the system failure. In addition, different methods and techniques how to mitigate them, were introduced. Furthermore, the solutions were considered based on previously collected information.

For the software fault detection, it is proposed to use reporting capability for user to view the exchanged data, software state and information about memory availability and BLE/Wi-Fi physical link operability.

For the software fault containment modularization is needed to avoid a situation where one part of software creates a failure in the entire system.

For the software fault recovery, the software should use checkpoint and restart fault recovery technique, because it is the simplest technique to regain the operability of the system. After system restart the error report must be sent to server.

Software architecture implemented the main concept into the code. The code can be accessed via a git repository link in Appendix 3 – Developed software.

Although the automatic configuration system itself is operational, there are some features left for the future work. Due to strict time constraints, following addition will be added in the future:

- data security features;
- reliability techniques considered to avoid system failures;
- the solution to avoid continuously working processes, such as advertising and scanning, to decrease power consumption is also added in the future.

The chapter about developed method and code gives a chance to evaluate the operability of the system.

6 Experimental evaluation

This chapter covers the tested functionalities of the software and is divided sections describing initial configuration, device data analysis, secondary configuration and Wi-Fi AP data update. In addition, a section is added to describe the potential benefit of the proposed system taking account the power and time used for one configuration action.

6.1 Initial Configuration

6.1.1 Connection via mobile application

Figure 31 displays the EspBlufi mobile application which scans two communication devices called BLUFI_DEVICE. These devices are advertising to start the configuration or updating process. One with a MAC address 24:0A:C4:01:F7:F2 is selected for initial configuration.

The information from the ESP is visible by using CuteCom interface seen on Figure 32. Text "*BLUFI_DEMO: BLUFI_ble_connect*" indicates that the connection process was successful and the configuration process is ready to take place.

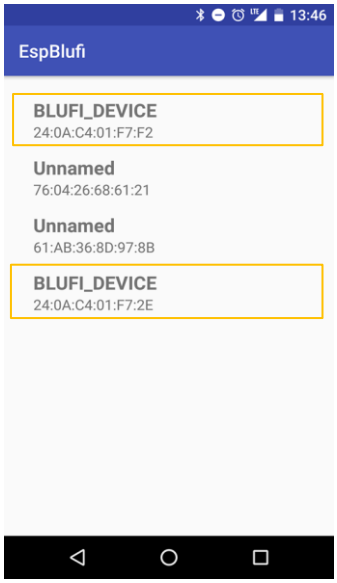


Figure 31. Advertising devices

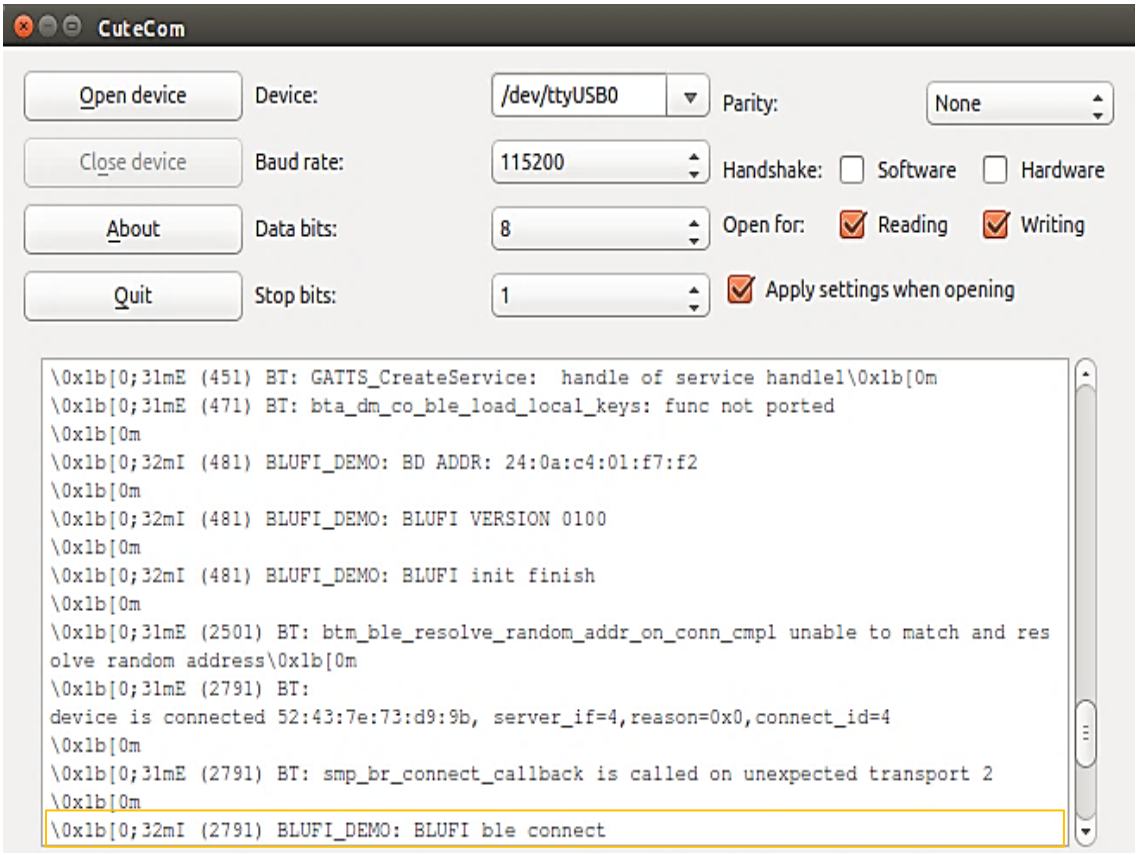


Figure 32. Initial connection

6.1.2 Configuration via mobile application

ESP32 is configured by sending the Wi-Fi SSID and password to the device. Configuring process is presented in Figure 33.

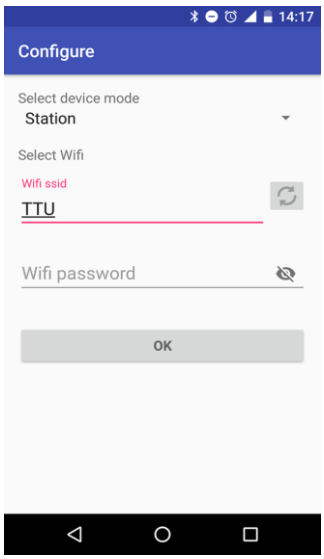


Figure 33. Initial or update configuration

The information from ESP32 is seen on Figure 34. Text "*HTTPS_REQUEST: Connected to AP*" indicates that the configuration process was successful and the ESP32 is ready to request cluster data from the server.

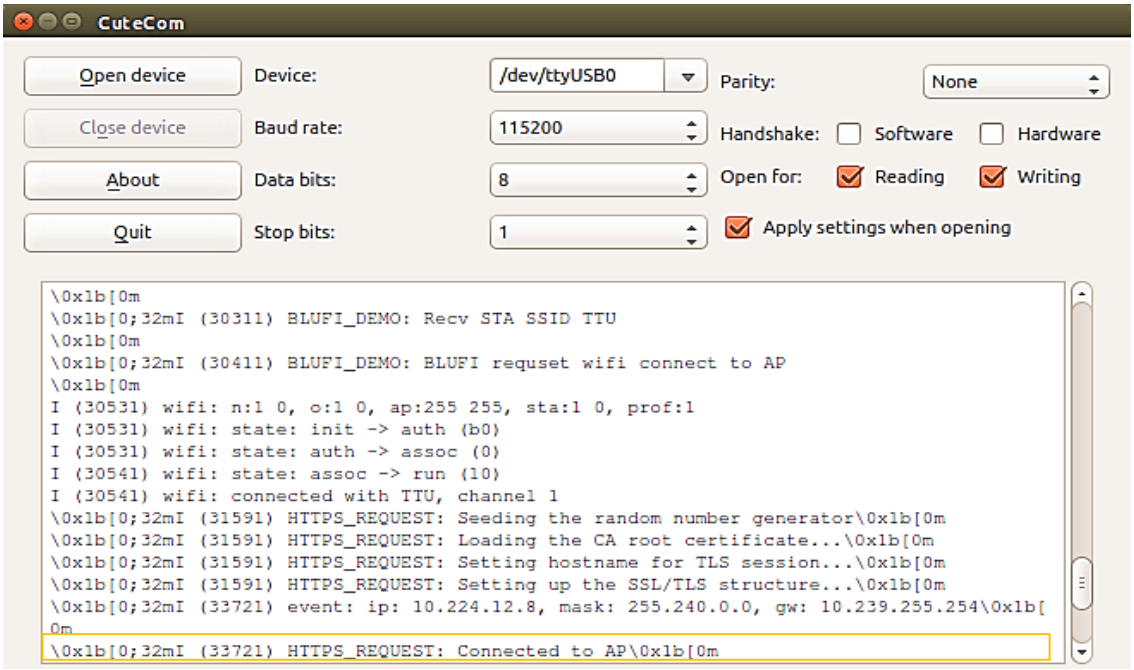


Figure 34. Initial configuration data from ESP32

Wireless Network Watcher verifies that the BLUFI_DEVICE with a MAC address 24:0A:C4:01:F7:F2 is connected to the correct Wi-Fi AP. Interface is presented on Figure 35.

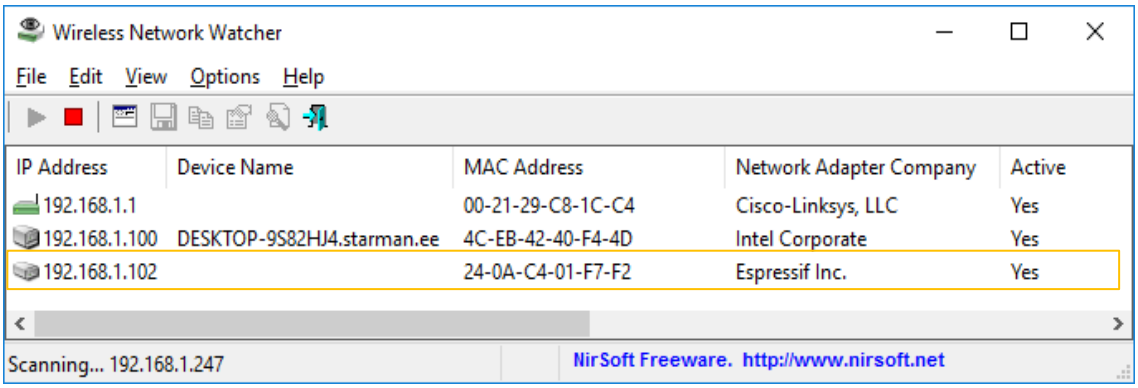


Figure 35. One device connected to Wi-Fi AP verification

6.2 Device data analysis

6.2.1 Cluster data request

After successful configuration process, the ESP32 sends a request to ThingSpeak IoT cloud server. As a response the cloud server sends the devices which belong to customers device cluster. The response is visible on Figure 36. Texts "*HTTPS_REQUEST: Cluster device 1: 24:0A:C4:01:F7:F2*", "*HTTPS_REQUEST: Cluster device 2: 14:BB:6E:30:30:59*" and "*HTTPS_REQUEST: Cluster device 3: 24:0A:C4:01:F7:2E*" indicates that there are three devices in the cluster according to the server data. This data is used in later stages of automatic configuration.

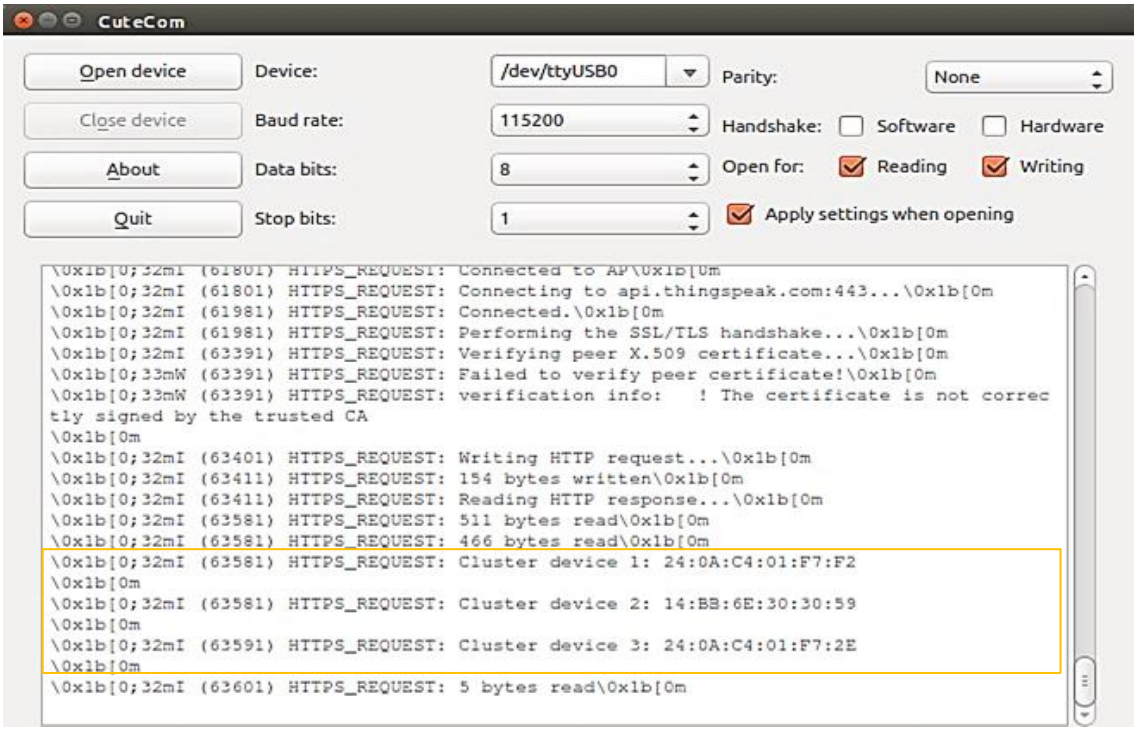


Figure 36. Cluster data

6.2.2 Scanned data observation

The scanning process runs continuously in ESP32. The scan results are visible on Figure 37. Texts "*BEACON_DEMO: Scanned device 0: 98:E0:D9:9B:46:4E*", "*BEACON_DEMO: Scanned device 1: 24:0A:C4:01:F7:2E*", "*BEACON_DEMO: Scanned device 2: 76:04:26:68:61:21*", and "*BEACON_DEMO: Scanned device 3: 6A:D9:A3:C7:0F:76*" indicates that the ESP32 found four devices in the close proximity. This data is used in later stages of automatic configuration.

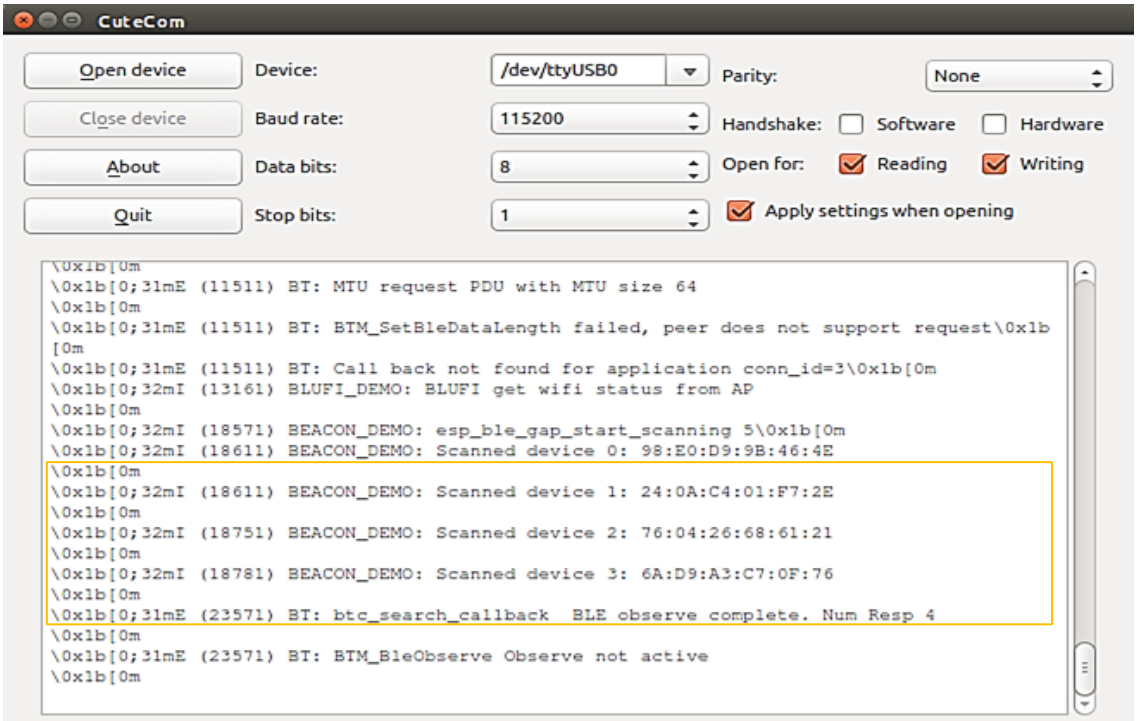


Figure 37. Scanned data

6.2.3 Unconfigured cluster device recognition

The data gathered from server and scanning process is compared to find devices belonging to the user's cluster. If these ESP32s are found, it is tested whether the devices were configured in the past 30 seconds to check if the Wi-Fi AP association information has been updated.

If Wi-Fi AP data update has taken place in during last 30 seconds, the configuration process will not take place because it was already updated within mentioned time. As we can see in Figure 38, ESP32 finds the cluster device but fails to reconfigure it. Text "*BEACON_DEMO: Cluster device 3 was recently updated less than 30 seconds ago.*" indicates that the connection process could not be initiated.

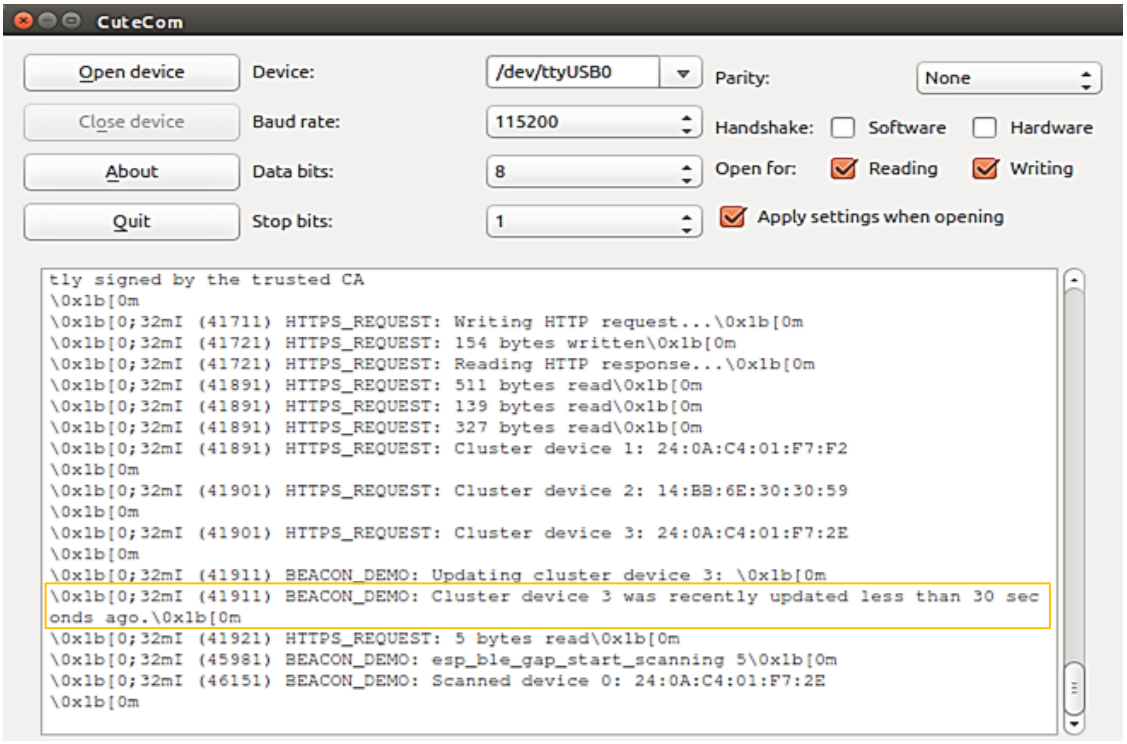


Figure 38. Configuration process failed

Otherwise, the secondary configuration phase starts as seen on Figure 39. Texts "*BEACON_DEMO: Updating cluster device 3:*" and "*BEACON_DEMO: Cluster device 3 was recently updated more than 30 seconds ago.*" indicates that the connection process is initiated.

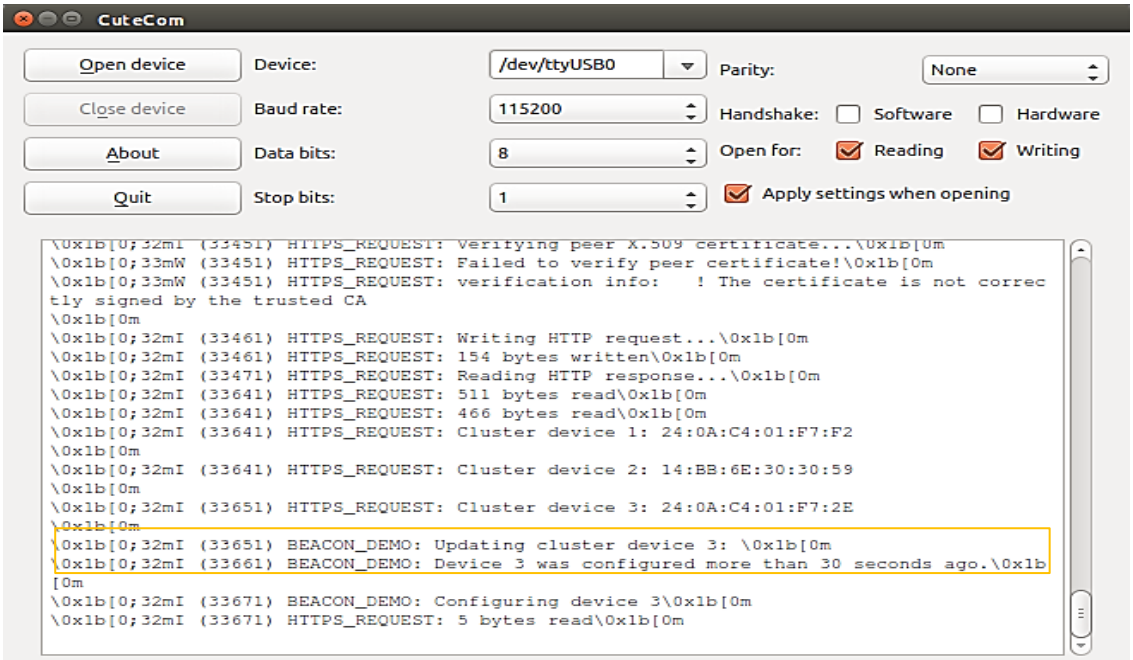


Figure 39. Configuration process initiated

6.3 Secondary configuration

Secondary configuration will take place after initial configuration. In this process ESP32s are the configuring devices. Process will continue until all cluster devices are configured successfully.

6.3.1 Connection via unconfigured ESP32

Differently from the initial configuration process, the connection with other ESP32 is created by configured ESP32. The connection establishment between two ESP32s can be viewed on Figure 40. Text "*GATTC_DEMO: REMOTE BDA: 24:0a:c4:01:f7:2e*" indicates that the connection is established with unconfigured ESP32. The MAC address of unconfigured ESP32 is 24:0A:C4:01:F7:2E.

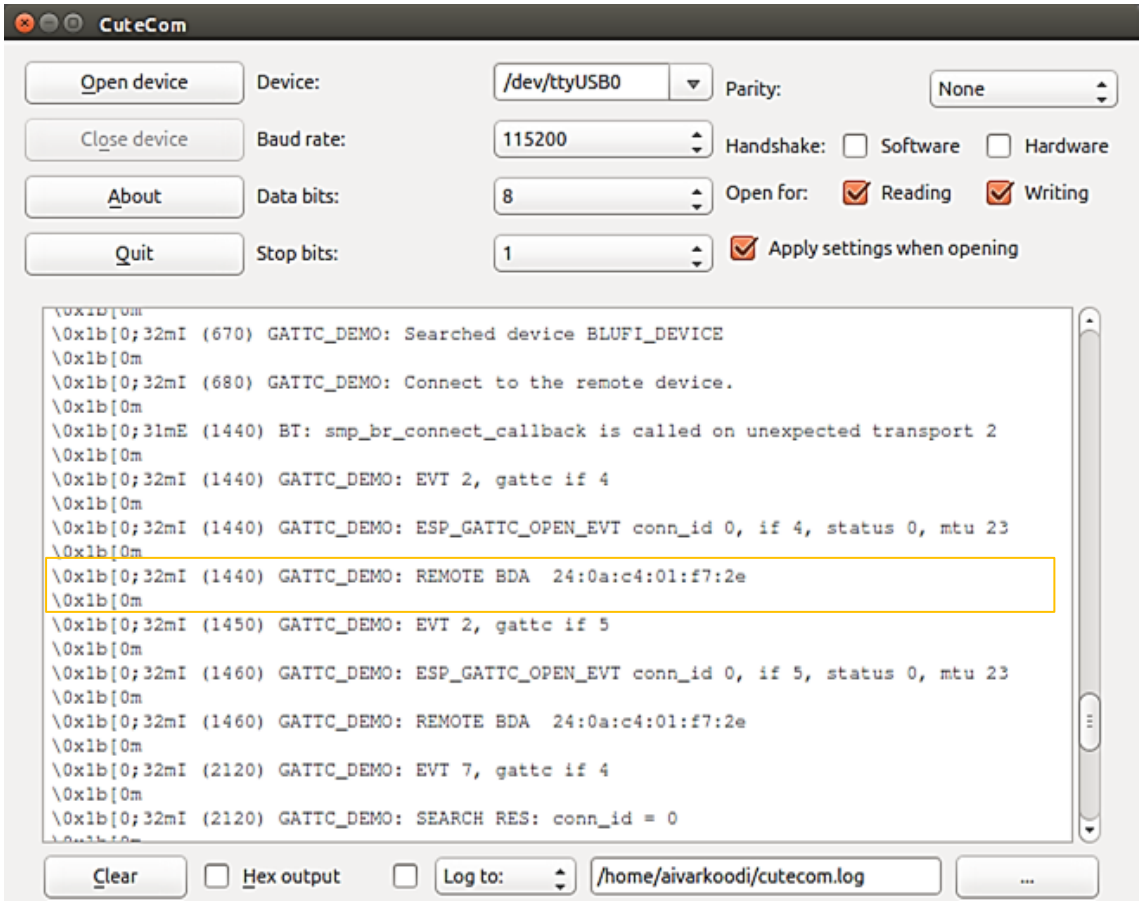


Figure 40. Configured ESP32 establishes connection

The connection with unconfigured ESP32 can be viewed in Figure 41. Text "*BT: 24:0A:C4:01:F7:f2, Server_if=4, reason=0x0, connect_id=4*" indicates that the connection is established with configured ESP32 and the MAC address of this is 24:0A:C4:01:F7:F2.

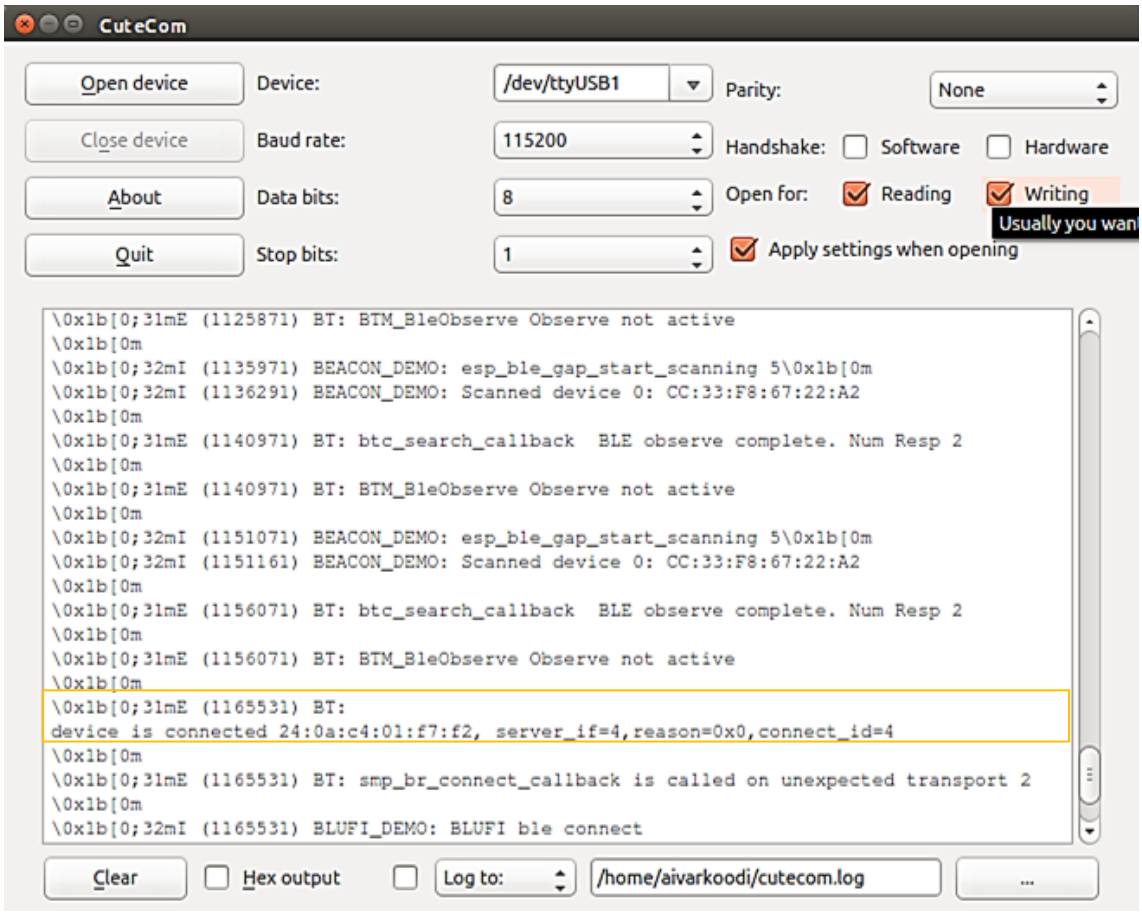


Figure 41. Unconfigured ESP32 receives connection

6.3.2 Configuration of unconfigured ESP32

Configured ESP32 sends Wi-Fi AP association information unconfigured ESP32. The unconfigured ESP32 uses this information to connect with Wi-Fi AP as seen in Figure 42. Text "*HTTPS_REQUEST: Connected to AP*" indicates that the configuration process was successful and the ESP32 is ready to request cluster data from the server.

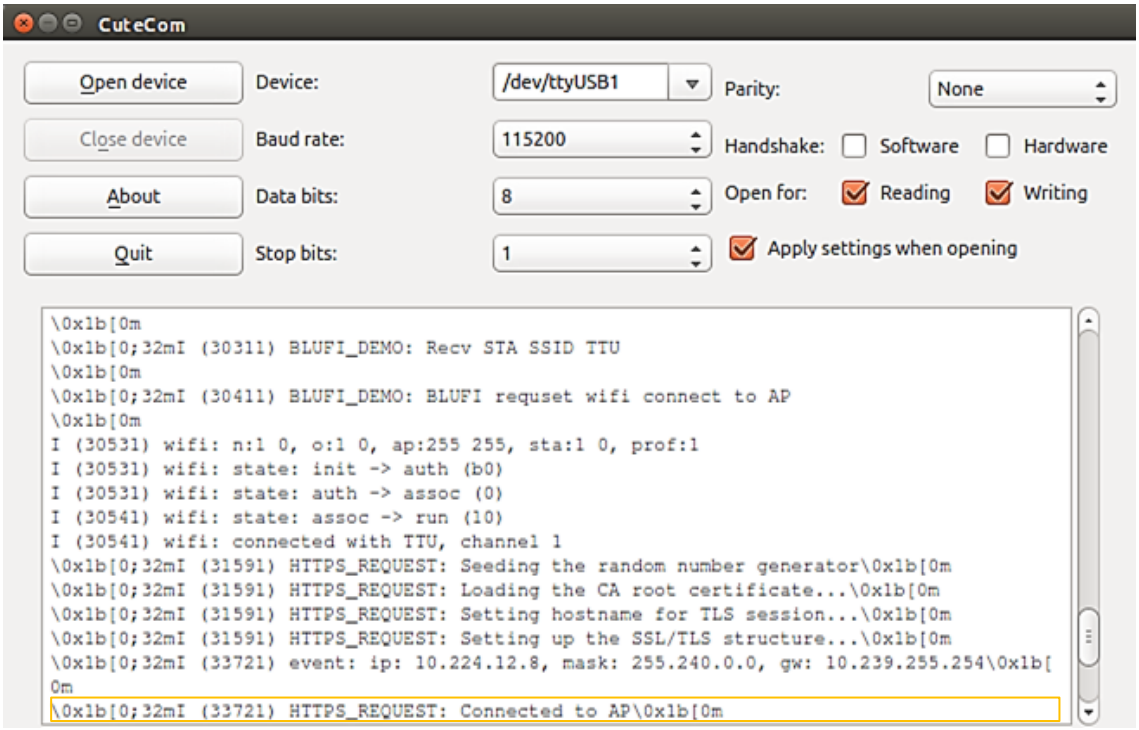


Figure 42. Configuring an unconfigured ESP32

Wireless Network Watcher verifies that ESP32s with a MAC addresses 24:0A:C4:01:F7:F2 and 24:0A:C4:01:F7:2E are connected to the correct Wi-Fi AP. Interface is presented on Figure 43.

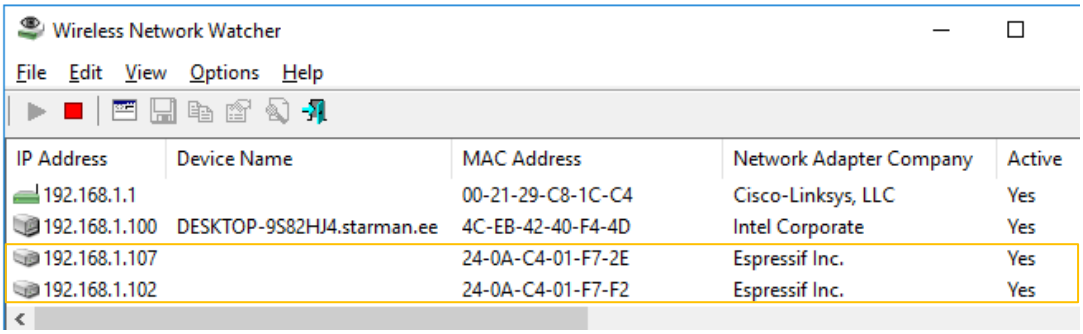


Figure 43. Two devices connected to Wi-Fi AP verification

In addition, we can see how the newly configured ESP32 makes a successful cluster device request in Figure 44. Texts "*HTTPS_REQUEST: Cluster device 1: 24:0A:C4:01:F7:F2*", "*HTTPS_REQUEST: Cluster device 2: 14:BB:6E:30:30:59*" and "*HTTPS_REQUEST: Cluster device 3: 24:0A:C4:01:F7:2E*" indicates that there are three devices in the cluster according to the server data.

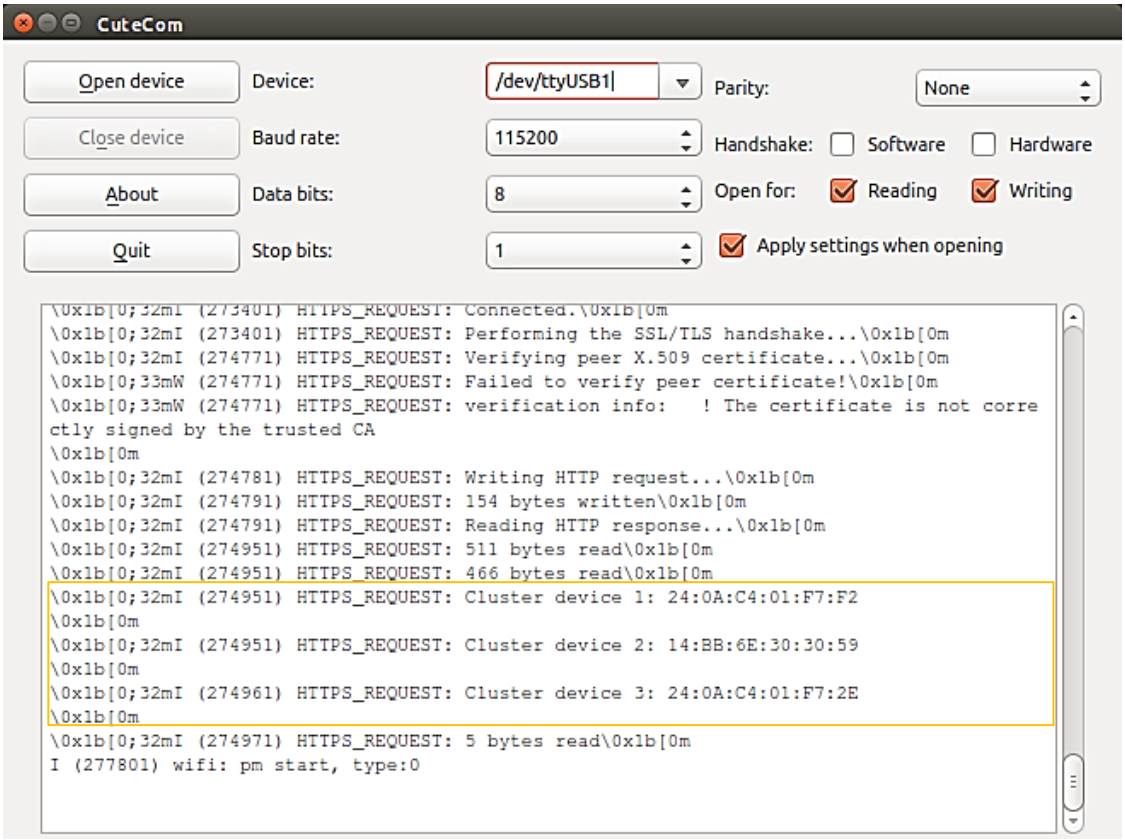


Figure 44. Server request from newly configured device

6.4 Summary of the evaluation

This chapter presented the results obtained while testing the developed system. The testing was divided in three sections: initial configuration, device data analysis and secondary configuration.

Initial configuration takes place when user accesses the ESP32 with a BLE device and configures it. The test conducted showed that the capability for these actions was possible and the ESP32 established a connection with Wi-Fi AP.

Device data analysis consists of operations in which ESP32 requests and receives data about cluster devices from the server, scans the near-by devices and recognizes the unconfigured or not updated cluster devices. These features were tested. The device could access server and receive cluster data, conduct a scan and find all near-by devices, and determine which device should be configured.

Secondary configuration takes place when recently configured ESP32 starts to configure or update another ESP32. The test conducted showed that the capability for these actions was possible and the ESP32 established a connection with Wi-Fi AP. Thanks to update timer, the system did not go to endless configuration loop. The fact that the operation was conducted automatically indicates the possibility to configure bigger clusters than the one limited by 30 devices in this thesis.

In conclusion, the software worked as it supposed to. The fact that security and reliability methods were not used makes the system vulnerable to any kind of cyber-attacks and causes significant problems upon failure. These two topics must be addressed in the future work.

7 Results, conclusion and future work

This chapter of the thesis covers the collected results, conclusion and future work with this project.

7.1 Results

This master's thesis presents a work done to propose and develop a solution for Ubik Solution OÜ for Optiverter communication unit automatic configuration. The automatic configuration solutions developed enables to configure IoT devices without reprogramming, decrease time spent on configuring, automate the configuring process and make the process user-friendly.

The State-of-the-Art research of found configuring systems, such as Texas Instruments CC3000 SmartConfig, Espressif ESP-TOUCH, Wi-Fi Alliance NFC "tap-to-connect", and Ubik Solutions OÜ configuring solution, showed that there is no technology available which would guarantee a user-friendly experience and low configuring time. In addition, none of the systems can automatically configure a device which is added to the system later on. To overcome these flaws, a system was proposed ensuring the minimal time on configuration process and user-friendly usage.

Since Ubik Solutions OÜ needs this system to work outdoors, within the time period of 10-20 years, the devices must be low-cost, robust and still be able to conduct all communication activities. Configuration process must use BLE for communication within the cluster. In addition, user must be able to connect to device at any time moment and store Wi-Fi AP's SSID and password to the device. Furthermore, device must be able to connect and receive cluster information from the server to compare it with scanned devices. This is needed to connect and configure with unconfigured or not updated devices. Behind the functionality, the data transmitted and received must be secured with encryption methods available to selected communication device.

ESP32, ATWIN3400, WL1835MOD and W2CBW0015 were the devices under consideration to be selected for the thesis. Conducted analysis showed that ESP32 had

the best features for minimal cost and therefore selected. In addition, Ubik Solutions used the Espressif ESP8266 as their previous communication device, which is a predecessor to ESP32. Both chips use RTOS which makes it easy to transport already developed functionalities to ESP32.

System architecture stated the outline of the workability, that is needed for software development. The system is designed to be accessed and configured via user's BLE device. After configuration process by user, the system will automatically configure all other cluster devices in the proximity. Configuration process ends when all cluster devices are configured to access selected Wi-Fi AP.

Reliability analysis was conducted to find potential hazards within the system which could lead to system failure. For the software fault detection, it is proposed to use reporting capability for user to view the exchanged data, software state and information about memory availability and BLE/Wi-Fi physical link operability. For the software fault containment modularization is needed to avoid a situation where one part of software creates a failure in the entire system. For the software fault recovery, the software should use checkpoint and restart fault recovery technique, because it is the simplest technique to regain the operability of the system. After system restart the error report must be sent to the server.

Software architecture implemented the main concept presented in system architecture into the code. Successful software tests proved that the system works. The code can be accessed via a git repository link in Appendix 3 – Developed software.

7.2 Future work

In the future, three issues need to be solved for this solution. The current problems are lack of data security, low system reliability and high energy consumption.

Firstly, security measures should be implemented in the software architecture and software to make it less vulnerable to all cyber-attacks. Fortunately, ESP32 offers many features to improve.

Secondly, reliability considerations should also be taken into account to prevent significant problems upon system failure. Since these issues are mapped and considerations proposed, there is a good basis to add this functionality.

Thirdly, running processes continuously is consuming a lot of power. In principle, the advertising and scanning events does not need to work all the time, because of two reasons. Advertising is only needed during initial configuration and update processes. Scanning is needed until all the cluster devices are configured. This flaw could be eliminated by sending the information from server side to start configuring or updating the devices.

In addition, with the release of Bluetooth 5.0 core specification, there is a chance that promised flooding and routing mesh capability will be included in next version. This would establish a basis for more energy efficient data transmission thanks to possibility that only one device needs to be connected with Wi-Fi AP and data of other devices will be sent to it via Bluetooth mesh. Since the automatic configuration system already uses BLE for communication with each other, the mesh networking would be easily implemented. The basic principle of the system using mesh capability is presented in Figure 45.

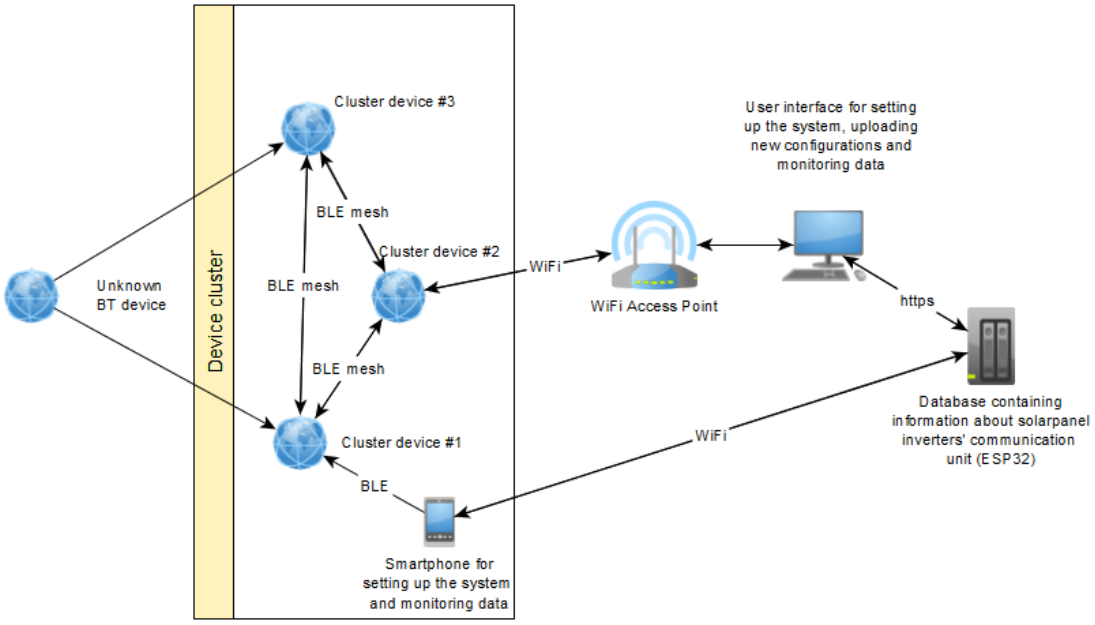


Figure 45. Basic principle using mesh networking

7.3 Conclusions

This master thesis solved the problems stated by Ubik Solution OÜ, by selecting suitable communication device for work, establishing base requirements and creating a methodology to conduct the functions needed. Furthermore, a software was created based on the system architecture and tested. Since the test results showed that the concept of automatic configuration solution works, it can be concluded that there is a possibility to develop a system based on set requirements.

References

[1] P. Newman, “The Internet of Things 2017 Report,” BI Intelligence, 2017.

[2] Texas Instruments Incorporated, “CC3000 SmartConfig Getting Starter,” Texas Instruments Incorporated, 24 September 2013. [Online]. Available: http://processors.wiki.ti.com/index.php/CC3000_SmartConfig_Getting_Started. [Accessed 19 November 2016].

[3] Espressif Incorporated, “ESP-TOUCH User Guide,” 12 April 2016. [Online]. Available: http://www.espressif.com/sites/default/files/30b-esp-touch_user_guide_en_v1.1_20160412_0.pdf. [Accessed 27 November 2016].

[4] Wi-Fi Alliance, “Wi-Fi CERTIFIED Wi-Fi Protected Setup™ adds NFC "tap-to-connect" for simple set up of security-protected Wi-Fi® devices and networks,” 9 April 2014. [Online]. Available: http://www.wi-fi.org/news-events/newsroom/wi-fi-certified-wi-fi-protected-setup-adds-nfc-tap-to-connect-for-simple-set-up. [Accessed 27 November 2016].

[5] D. Anupam and D. Biswajit, “A Study on Energy Consumption of Different Wireless Devices,” 28 December 2012. [Online]. Available: http://www.ijert.org/view-pdf/1847/a-study-on-energy-consumption-of-different-wireless-devices. [Accessed 7 January 2017].

[6] Espressif Incorporated, “ESP32 Datasheet,” 4 April 2017. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf. [Accessed 12 April 2017].

[7] Seeed Development Limited, “ESP-32S Wifi Bluetooth Combo Module,” Espressif Incorporated, 23 September 2016. [Online]. Available: https://www.seeedstudio.com/ESP-32S-Wifi-Bluetooth-Combo-Module-p-2706.html. [Accessed 2 November 2016].

[8] Atmel Corporation, “ATWINC3400 PRELIMINARY DATASHEET,” July 2015. [Online]. Available: http://www.atmel.com/images/Atmel-42396-SmartConnect-ATWINC3400_Datasheet.pdf. [Accessed 10 January 2017].

[9] Digi-Key Electronics, “Microchip Technology ATWINC3400-XPRO,” 5 January 2017. [Online]. Available: https://www.digikey.com/products/en/rf-if-and-rfid/rf-evaluation-and-development-kits-boards/859?k=atwinc3400. [Accessed 16 January 2017].

[10] Texas Instruments Incorporated, “WL1835MOD,” Texas Instruments Incorporated, December 2015. [Online]. Available: http://www.ti.com/product/WL1835MOD. [Accessed 2 November 2016].

[11] Mouser Electronics Incorporated, “WL1835MOD,” Texas Instruments Incorporated, December 2015. [Online]. Available: http://eu.mouser.com/Search/Refine.aspx?Keyword=WL1835MOD. [Accessed 4 November 2016].

[12] Wi2Wi Incorporated, “W2CBW0015 datasheet,” 17 May 2013. [Online]. Available:

http://wi2wi.com/pdf/W2CBW0015.pdf. [Accessed 17 November 2016].

[13] Mouser Electronics Incorporated, “W2CBW0015,” Wi2Wi Incorporated, June 2012. [Online]. Available: http://eu.mouser.com/ProductDetail/Wi2Wi/W2CBW0015-T/?qs=1QLBgCcymxVWwsJhYLtuuA%3D%3D. [Accessed 3 November 2016].

[14] The MathWorks Incorporated, “ThingSpeak,” The MathWorks Incorporated, 2017. [Online]. Available: https://thingspeak.com. [Accessed 12 November 2016].

[15] IEEE, “1633-2016 - IEEE Recommended Practice on Software Reliability,” 2016. [Online]. Available: https://standards.ieee.org/findstds/standard/1633-2016.html. [Accessed 11 February 2017].

[16] E. Dubrova, “Software Redundancy,” in *FAULT TOLERANT DESIGN*, Stockholm, Springer, 2013, pp. 114-119.

[17] Stak Trading, “CP2102 USB to TTL Serial Adapter,” Silicon Laboratories, January 2017. [Online]. Available: https://stak.com/Silicon_Labs_CP2102_USB_to_TTL_Serial_Adapter. [Accessed January 2017].

Appendix 1 – Bluetooth Core Specification versions

The document containing Bluetooth Core Specification versions is uploaded in GitLab environment and can be accessed by using Uni-ID provided by Tallinn Univeristy of Technology.

Following link directing to the pdf document was last available in 19.05.2017.
http://gitlab.pld.ttu.ee/thesis/2017_ble_mesh/blob/master/documents/finalthesis_appendix_1.pdf

Appendix 2 – Bluetooth 4.2 / Bluetooth Low Energy

The document containing Bluetooth 4.2 / Bluetooth Low Energy overview is uploaded in GitLab environment and can be accessed by using Uni-ID provided by Tallinn Univeristy of Technology.

Following link directing to the pdf document was last available in 19.05.2017.
http://gitlab.pld.ttu.ee/thesis/2017_ble_mesh/blob/master/documents/finalthesis_appendix_2.pdf

Appendix 3 – Developed software

The developed software is uploaded in GitLab environment and can be accessed by using Uni-ID provided by Tallinn Univeristy of Technology.

Following link directing to developed C project was last available in 19.05.2017.
http://gitlab.pld.ttu.ee/thesis/2017_ble_mesh/tree/master/developement/blufi