Farzad Mohebi

MANTRANIC MULTIPURPOSE IOT GATEWAY

Design and Implementation

Thesis
CENTRIA UNIVERSITY OF APPLIED SCIENCES
Information Technology Engineering
October 2020

ABSTRACT

Centria University of Applied Sciences	**Date** October 2020	**Author** Farzad Mohebi
Degree programme Information Technology Engineering		
Name of thesis MANTRANIC MULTIPURPOSE IOT GATEWAY DESIGN AND IMPLEMENTATION		
Instructor Jari Isohanni		**Pages** 34+5
Supervisor Jari Isohanni		

This thesis aimed to create a multipurpose gateway for the Internet of Things that can handle various forms of wired and wireless connections with easy configuration setup by using simple and inexpensive components.

IoT is one of today's most evolving technologies, and the number of IoT devices in the world is going to increase. In implementing IoT projects, there are different connections, e.g., WiFi, Bluetooth, LoRa, Sigfox, Ethernet, and USB. Each device may use a particular connection, requiring to use several gateways for different devices, and also an exclusive multi-connection gateway is expensive to use in small projects. This thesis offers a solution to the problem. There are two main sections in this thesis. The first part is about hardware design and implementation, and the second part is about firmware and gateway software design.

This thesis work designed a PCB and implemented a low-cost multi-connection gateway circuit board which has connection ability through Bluetooth 4.2, Bluetooth Low energy 4.2, WiFi 2.4 GHz, 433 MHz LoRa Long-range radio frequency and also the physical connection of Ethernet10/100, Micro USB V2, I2C, SPI, and general-purpose input/outputs.

This project used C, C++, and HTML programming languages to build the firmware and configuration web service in Arduino IDE in the software section. The server services are configured over MQTT protocol, which is most in use in IoT, and it uses AES 256-bit encryption method over broadcasting messages. Thesis work designed a Restful API built-in for further development and customization over third party services.

In this thesis work, an open-source, cost-effective, and easily configurable gateway has been developed to reduce the cost of time and money in implementing IoT Smart solutions.

Key words
IoT, Gateway, PCB, SoC, BLE, WiFi, LoRa, Arduino, C, C++, HTML, SoC, MQTT, I2C, SPI, API

CONCEPT DEFINITIONS

Gateway	a device that communicates with legacy or non-internet connected devices
IoT	Internet of Things
PCB	Printed circuit board
SoC	System on a Chip
MCU	Micro Controller Unit
BLE	Bluetooth Low Energy
WiFi	Wireless Fidelity
LoRa	Long Range Radio Frequency
LPWAN	Low-Power Wide-Area Network
IDE	Integrated Development Environment
C	C Programming Language
C++	C ++ Programming Language
HTML	Hypertext Markup Language
UART	Universal Asynchronous Receiver-Transmitter
MQTT	light-weight messaging protocol for small sensors and mobile devices
RTC	Real-Time Clock
OLED	Organic Light-Emitting Diode
USB	Universal Serial Bus
SPI	Serial Peripheral Interface
I/O	Input/Output
NFC	Near Field Communication
JSON	JavaScript Object Notation
XML	Extensible Markup Language
CSV	Comma-Separated Values
UDP	User Datagram Protocol
MIPS	Microprocessor without Interlocked Pipelined Stages
FPU	Floating-Point Unit
I2C	Inter-Integrated Circuit
OTA	On the Air
GPIO	General-Purpose Input/Output
EEPROM	Electrically Erasable Programmable Read-Only Memory

A/D	Analog-to-Digital Converter
FSK	Frequency-shift keying
GFSK	Gaussian frequency-shift keying
MSK	Minimum-Shift Keying
GMASK	Gaussian Minimum-Shift Keying
OOK	On-Off Keying
FPGA	Field-Programmable Gate Array
LED	Light-Emitting Diode
SPIFFS	Serial Peripheral Interface Flash File System
UI	User Interface
API	Application Programming Interface
KNX	an open standard for commercial and domestic building automation
CAN bus	Controller Area Network
Zigbee	a short-range telecommunication network according to IEEE 802.15.4 standard
Z-Wave	a wireless communications protocol used primarily for home automation
FOG	an architecture that uses edge devices to carry out a substantial amount of computation, storage, and communication locally and routed over the internet backbone.

ABSTRACT
CONCEPT DEFINITIONS
CONTENTS

PICTURES

1 INTRODUCTION

The human desire to make a comfortable existence is due to their technological ambition. The world has undergone the industrial revolution over the last century with the inventions of modern technology frontiers. Technology products communicate with human beings and do any possible task with far higher precision in a shorter time. From the moment people started using 'smart concepts,' the world is now more connected. Smart concepts include smart devices, smart cities, and smartphones. It forms an ecosystem of devices whose primary role is to connect different sending and receiving data methods. The Internet of Things (IoT) is a new technology in charge of connected, intelligent devices. IoT let humankind to make their imaginative visions come alive in the last decades. It should be considered as a part of the fourth industrial revolution. It has socially, economically, and technologically affected every aspect of human life. Humanity now foresees the moment for things that are sensing, learning, and acting by themselves. IoT also integrates other principles such as FOG computing, edge computing, communication protocols, mobile devices, sensors, and geo-location. (Adebayo, Chaubey, Numba. 2019). Internet of things gateway is the bridge for on-field IoT devices, cloud services, and smartphone-like client equipment; the IoT gateway provides a link between field and cloud, even offline services, and real-time system control in the zone. The IoT gateway aggregates all data, translates the protocols of sensors, and pre-processes the data before sending them, providing IoT data protection by adding a defensive line, enhancing energy efficiency on IoT devices, and converting all forms of mixed IoT interactions into a single standard protocol.

Manufacturers of smart things currently only have their gateways for their things in any application, leading the end-user to confusion when using various manufacturers' products and having multiple gateways to handle a project. Even at the software level, it is difficult to synchronize different applications to work with all other parts. Versatile IoT connections and protocols trigger this problem. Also, users may need to have access to various firmware parts to set up their intended configuration. The user can then set up a core controller based on a development board or MCU, which also needs to connect to all those gateways to handle. Manufacturers of IoT Systems currently produce multiple gateways. Each of these gateways is configured to perform a specific function. The thesis work aims to combine general functionalities of gateways in use and control MCUs in a universal product, which must handle the much-needed functionalities. This project aims to design and implement PCB hardware with hand-picked components and develop suitable firmware drivers and configuration web application. The hardware must be a prototype of the final product according to the industrial requirements.

An open-source solution must have the ability to change each component so that the hardware must handle any scenario by the end-user. A gateway is responsible for connecting the things in a local area, collecting the data, interpreting it to a standard dataset format, and transmitting it to third-party services. This attitude requires the gateway to have frequently used connection components and easily configurable MCU with appropriate libraries for driving and modifying. The device should be built from scratch, and the MCU gateways should be selected from types that could provide sufficient processing power and adequate internal flash memory size and provide adequate embedded facilities to complement existing gateways functionalities. The gateway is the device that drives IoT. It links unknown devices to customized services and devices provided by third parties. The designer should consider how to design each part so the design can interact in the real environment.

Lastly, the main challenges to be tackled in the current study are designing gateway hardware with a multi-connection method over different connection protocols and external MCU GPIO's for inter-application use, reliability, and robustness of the Gateway circuit board, which is divided into four categories. The second challenge can be further divided into more parts; isolation (physical and signal), according to the requirements, the resistance between two test points should be at least 1MOhm in 60V; power supply robustness, which should tolerate the reverse polarity and short circuits and 48V injection for 2 minutes. Temperature, which should be in a way that the gateway can perform flawlessly in the temperature boundaries, which are + 85C and -20C at their highest and lowest respectively; and lastly, power consumption, which should be in a way that the current consumption of the gateway would not exceed 150mA.

2 THEORY

This chapter will introduce some specialized knowledge about IoT, IoT gateways, communication protocols, MCU's, and general information and a comparison of three high-performance multi-connection gateways in the market. This chapter also goes through the specific information and concepts of most in use protocols and their structure. The definition of the Internet of Things and the general architecture of it, and the purpose of IoT gateways existence and construction are discussed in this chapter. This chapter contains brief information on hardware concepts of specific electronic components used in gateways and the selection of proper components and tools for implementing this thesis work.

2.1 Internet of Things

In 1999, British technology pioneer Kevin Ashton first used the term 'Internet of Things' (IoT) to characterize a system where sensors could connect artifacts in the real world to the internet. Ashton invented the term to demonstrate the ability to monitor and control items without the need for human interaction by connecting Radio-Frequency Identification (RFID) used in industrial supply chains to the Internet. Nowadays, the Internet of Things has become a common term that describes contexts in which a range of objects, computers, sensors, and everyday products are enabled by Internet connectivity and computing power. However, the concept of "Internet of Things" is comparatively modern; the idea has been around for years to incorporate computers and networks to track and manage devices. By the late 1970s, systems were already in commercial use for remotely monitoring meters on the electrical grid via phone lines. In the 1990s, wireless technology developments enabled business and industrial solutions for equipment monitoring and operation to become widespread in "machine to machine" (M2M). Many of these early M2M solutions, rather than Internet Protocol (IP)-based networks and Internet standards, were based on closed purpose-built networks and proprietary or industry-specific standards. It is not an innovative concept to use IP to link devices other than computers to the internet. At an internet conference in 1990, the first internet 'device,' an IP operated toaster that could change the state of power switch over the internet, was featured. Over the next few years, other IP-enabled 'things' became available, including a soda machine at Carnegie Mellon University in the United States and a coffee pot in the Trojan room at Cambridge University in the United Kingdom (which remained linked to the internet until 2001). A robust line of research and development through "smart object networking" helped create the basis for today's Internet of Things from these origins (Rose, Eldridge & Chapin 2015, 12.) An IoT ecosystem consists of web-enabled smart devices that acquire, transmit, and react on data they obtain from their environments utilizing embedded systems, such as processors, sensors, and communication hardware. IoT devices exchange the sensors data they receive where data is either transmitted to the cloud for local processing or analysis by linking to an IoT gateway or other edge node. These devices often connect with other similar devices and operate on the data they get from each other. Without human interaction, the machines do most of the work, while humans may communicate with the devices, such as setting up, sending instructions, or accessing the data. The protocols for connectivity, networking, and communications used by these web-enabled devices rely mainly on IoT applications. Artificial intelligence (AI) and machine learning are also being used by IoT to help make data processing procedures faster and more flexible (Rouse 2015.) The presentation of four stages of IoT architecture can be found in the diagram below.(Figure 1.)

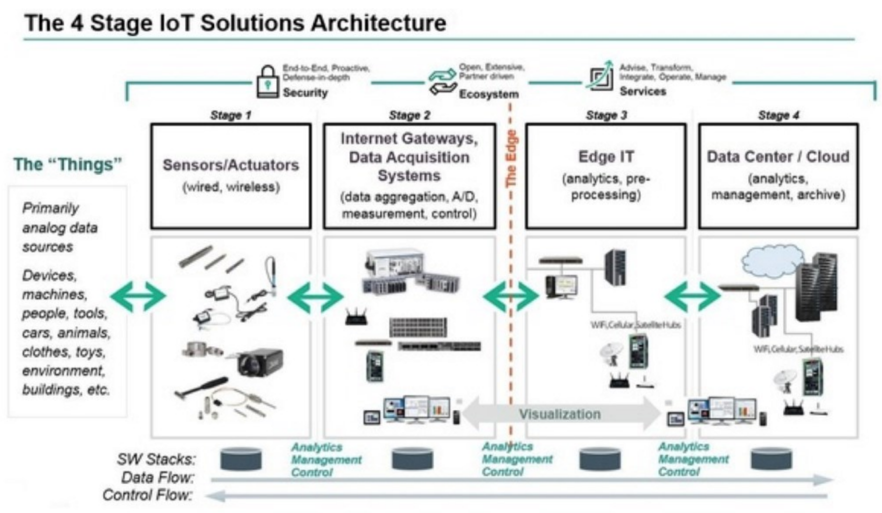

The 4 Stage IoT Solutions Architecture
End-to-End, Proactive, Defense-in-depth
Security
Open, Extensive, Partner driven
Ecosystem
Advise, Transform, Integrate, Operate, Manage
Services
Stage 1
Stage 2
Stage 3
Stage 4
The "Things"
Primarily analog data sources
Devices, machines, people, tools, cars, animals, clothes, toys, environment, buildings, etc.
Sensors/Actuators (wired, wireless)
Internet Gateways, Data Acquisition Systems (data aggregation, A/D, measurement, control)
The Edge
Edge IT (analytics, pre-processing)
Data Center / Cloud (analytics, management, archive)
WiFi, Cellular, Satellite Hubs
Visualization
SW Stacks:
Data Flow:
Control Flow:
Analytics Management Control

FIGURE 1. Four Stages of IoT architecture (adapted from Stokes 2018)

2.2 IoT Gateway

An IoT gateway bridges the connectivity distance between IoT devices, sensors, networks, and the cloud. IoT gateway systems provide local processing and storage solutions by consistently integrating the field data to the cloud and the capacity to monitor field devices autonomously based on data input by sensors (Desai 2016.) When the capacities and specifications of devices mutate, it is not feasible to directly interact with networks. Most sensors and devices do not support protocols such as WiFi or Bluetooth because they are energy-intensive. In their raw form, specific systems aggregate data where it is daunting and essential, and they all link to several public and private networks (Desai 2016.) IoT gateway per-forms many critical processes, from interpreting protocols to encrypting, encoding, handling, and filtering data. In an IoT ecosystem, a gateway sits between devices and sensors to communicate with the cloud (Desai 2016.) Basic IoT devices usually serve a single operation such as measuring temperature, wind, force, or other variables. The hardware requirements are relatively low for this type of device. The components used for this type of devices are inexpensive, usually using an SoC inside, containing at least two microprocessors, and several wireless connectivity stacks. (Cherkasova 2018.)

A gateway's general hardware block diagram combines a microcontroller, some discrete hardware circuits, power, indication and instrumentation modules, interfaces to connect, and URAT switch. Figure 2 illustrates a gateway's potential general hardware blocks. In specific gateways, all blocks are not implemented because of the unique gateway uses. The UART block is primarily found in open-source implementations where the user may change the firmware or log data to external memory, such as SD cards. The overall structure of an IoT solution is close to what can be seen in Figure 2. All peripherals are connected to the gateway, and the gateway collects, interprets, and sends the data to the internet or a third-party service (Application or Web server) through a broker method.

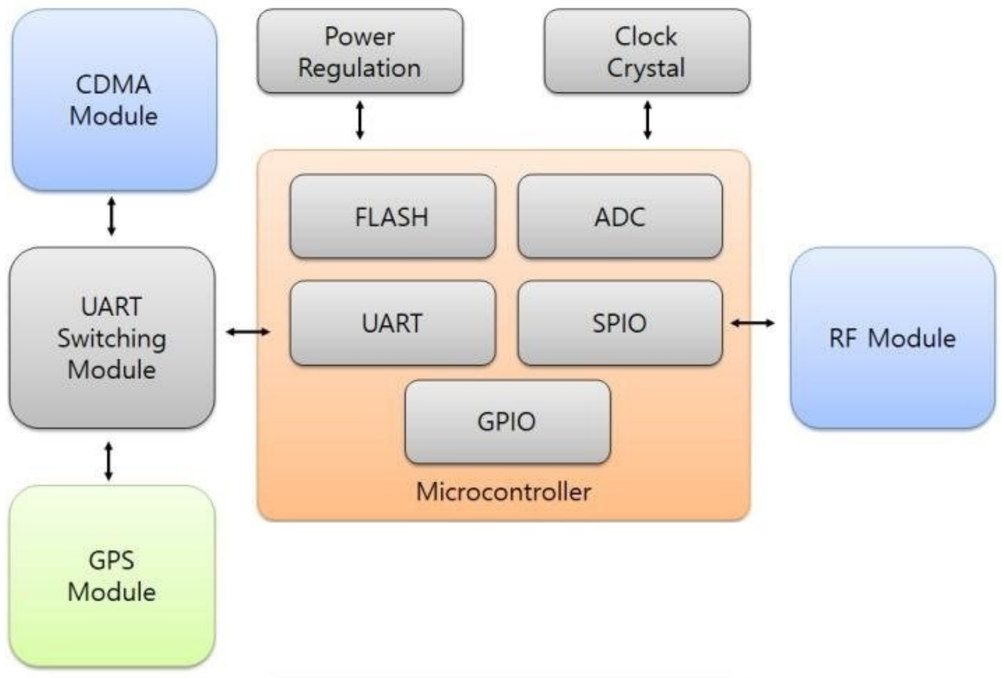


FIGURE 2. The general block diagram of a gateway (adopted from jo, lim 2019)

As an interpretation node, Gateway transforms data into a dataset in various formats, such as JSON, XML, CSV. Due to the massive volumes of data in the IoT, it needs to use a lightweight data set and fast data transfer method to process, store, and distribute data. Besides, gateways for controlling the trigger actions need to be reasonably fast enough to get data and act. (Youngjun Jo1 & Sangsoon Lim 2019.)

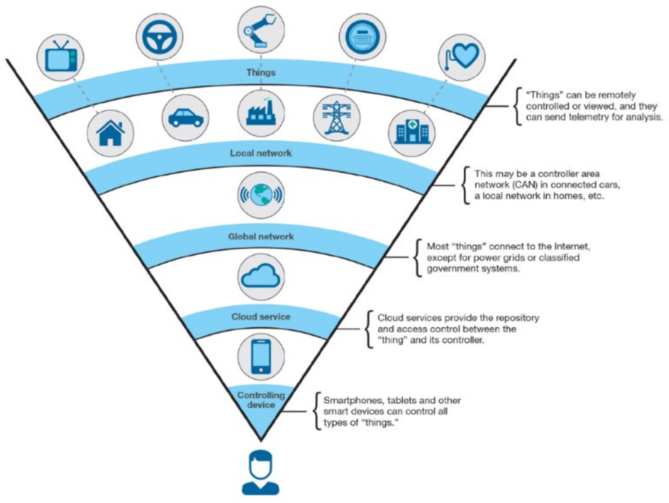

Things
"Things" can be remotely controlled or viewed, and they can send telemetry for analysis.
Local network
This may be a controller area network (CAN) in connected cars, a local network in homes, etc.
Global network
Most "things" connect to the Internet, except for power grids or classified government systems.
Cloud service
Cloud services provide the repository and access control between the "thing" and its controller.
Controlling device
Smartphones, tablets and other smart devices can control all types of "things."

FIGURE 3. The general structure of an IoT solution (adopted from Gerber, Kansal 2017)

2.3 Communication Technologies and Protocols

This part will cover basic descriptions of several communication protocols used in this thesis work and some other protocols that are not used in this thesis work. Gateways can communicate with things and clouds over different connections. e.g., GPIO, Serial communication, WiFi, BLE, LoRa, and Local area network over Ethernet10/100 in M2M connections and MQTT for M2C connections.

2.3.1 GPIO

All microcontrollers have an interface called GPIO, making it easier to reach the chip's internal properties. The pins are usually arranged on a microcontroller in groups of 8 that can be defined as input data where data from external devices can be accepted and an output where data can be sent to external devices. Microcontrollers can be operated by utilizing GPIO to do things like blinking LEDs by different external devices and can even control external devices operation in a comparable pattern (Vermesan & Friess 2013.) In general, the data is transmitted through the voltage level, such as the high level, the microcontroller supply voltage, and the low level, the 0V ground voltage. However, the logic level is usually used to judge the voltage level. For, e.g., if the logic level high is specified above 66 percent of the supply voltage, some lower voltage devices allow high voltage devices to interact as long as the voltage exceeds a logic high level. Besides, for ADC transitions or as an interrupt line for the Central Processing Unit, GPIO may also be programmed. (Liu 2017.)

2.3.2 Serial Communication

Serial communication is the method of streaming data sequentially using one or more, but not more than four wires for one single bit at a time. Parallel communication, on the opposite, transmits several bits along different wires concurrently. Two groups, synchronous and asynchronous, can be categorized through serial communication. Synchronous correspondence requires the sender to give the recipient a clock signal, but asynchronous does not (Blom 2017.) UART (Universal Asynchronous Receiver / Transmitter), a physical circuit block in a microcontroller, should be added to accomplish the transmission and reception of data serially and asynchronously. UART will convert the incoming data into sequential order in a parallel fashion and vice versa within the system; UART uses two wires to relay data between devices. Pin Tx is responsible for transferring data to other devices, and pin RX is responsible for the reception of data from other devices. Once UART gets the parallel data from the data bus, as seen in Figure 4 (Liu 2017.) it can add a start bit, a parity bit, and stop bits to the data to build a new data packet and send it out via pin Tx bit by bit. The Least Significant Bit (LSB) will be sent last, and the Most Significant Bit (MSB) will be sent first. If the UART identifies the start bit, it will notice that a data message is about to be transmitted. After getting all the data bits, it will wait for parity and stop bits.

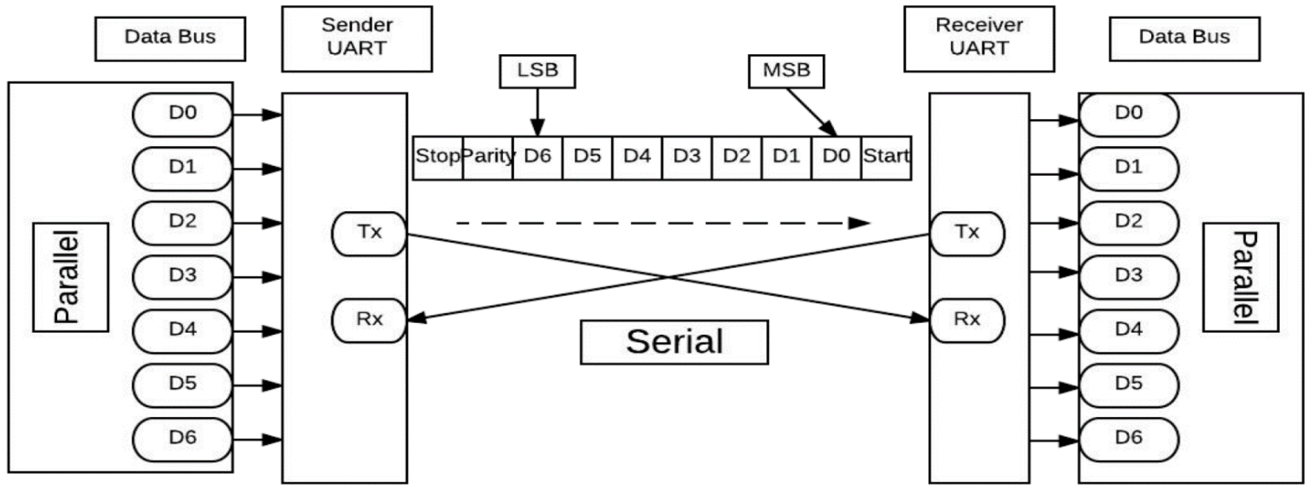


FIGURE 4. Overview of asynchronous serial communication (adopted from Yilin Liu 2017,7)

2.3.3 WiFi

WiFi is a wireless local area network (WLAN) that uses 2.4Ghz UHF and 5Ghz ISM frequencies following the IEEE 802.11 protocol. WiFi grants internet connectivity to users inside the range (approximately 66 feet from the point of access). For many IoT networks, WiFi is helpful, but such communications usually link to an external cloud service and are not connected to the smartphone directly. Owing to its comparatively high-power consumption, it is also not suggested for battery-powered applications. (Mehl 2018.)

2.3.4 BLE

Bluetooth Low Energy (Bluetooth LE, BLE) is similar to Bluetooth, a wireless, private area network technology. BLE is designed to operate on low power usage and is affordable. Like Bluetooth, BLE still has Bluetooth SIG-managed requirements. The Bluetooth SIG developed BLE for low-power devices that use less data. BLE goes to sleep while it is not in service and wakes up when the data transition occurs. Sleep mode makes it suitable for IoT systems operating on batteries and with low power consumption. The BLE modules enable a feature called "Dual Mode," making the device work with classic Bluetooth and a BLE device. (Latif 2018.)

2.3.5 LoRa

LoRa is a proprietary spread-spectrum radio modulation (EP2763321 of 2013 and US7791415 of 2008) initially developed by Cycleo (Acquired by Semtech in 2012) (Latif 2018.) Semtech's LoRa chips communicate in the sub-gigahertz spectrum (109MHz, 433MHz, 866MHz, 915MHz), which is an unlicensed band and has less disturbance than others (like the 2.4 GHz range used by WiFi, Bluetooth, and other protocols). LoRa Signals pass barriers at specific frequencies and travel long distances while drawing relatively little power, which is ideal for many IoT devices, where they are limited to battery life. LoRa devices use a spread-spectrum technique within the sub-GHz spectrum to transmit at various frequencies and data rates. LoRa enables the gateway to adjust to changing circumstances and refine how each machine shares data (Harwood 2018.) An end to end LoRa network protocol architecture is demonstrated in Figure 5.

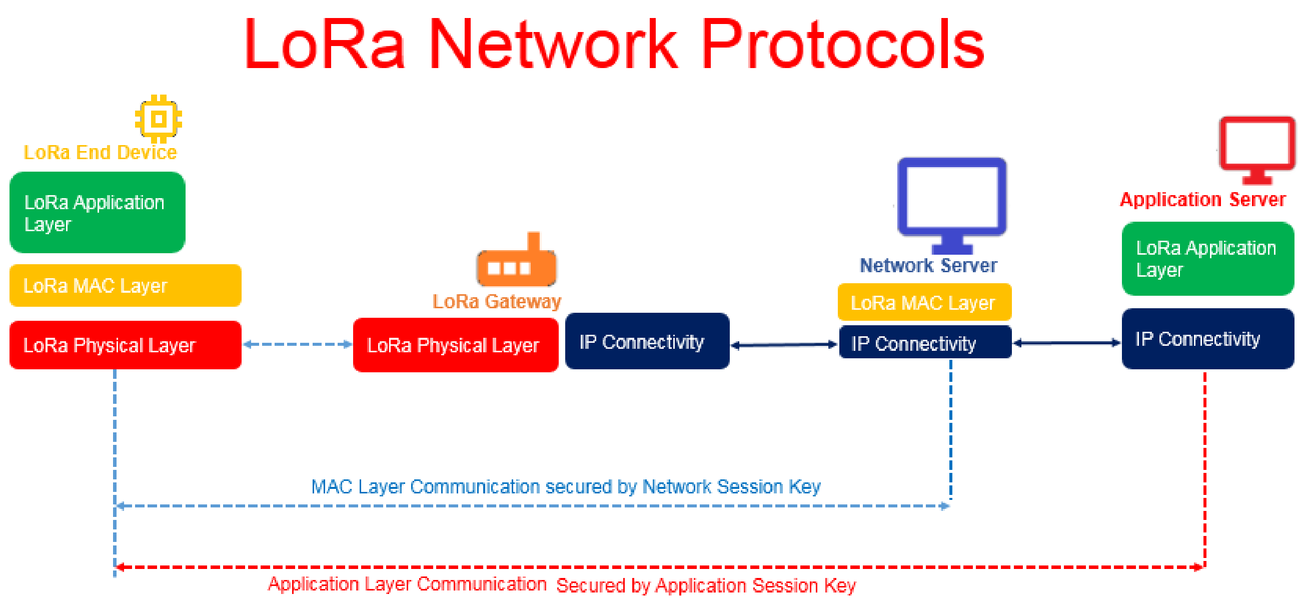


FIGURE 5. The end to end LoRa network protocol architecture (adapted from LoRa. Techplayon 2018, 13)

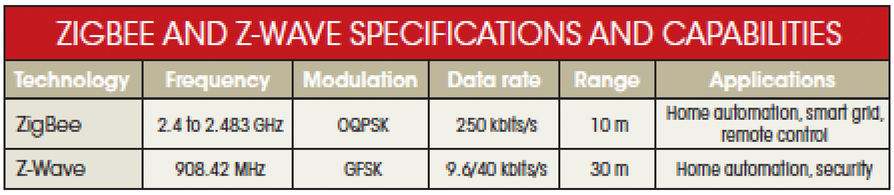
2.3.6 Ethernet

The most used Local Area Network (LAN) technology is Ethernet. To link the devices to the internet, it offers a wired connection. The TCP / IP stack is a physical and link layer protocol that defines how networked devices share their data via the physical medium to the computer or other network devices. It is based on the 802.3 IEEE standard. Inside an IoT system, it is possible to connect stationary or fixed IoT devices using Ethernet. Ethernet cables, for example, are used to link computers to routers to provide Internet access. As a basic example of LAN technology with Ethernet, the Ethernet cable acts as a wired connecting medium for the router and the device. Ethernet data speed is dependent on the type of cable and can be restricted by the network administrator. For Ethernet networking, there might be fiber optic, co-axial, or twisted pair cables used. The Ethernet has very low latency, making it ideal for mission-critical IoT applications where computers may be placed in a local area network or co-located. Unlike other networks, a regular ethernet connection is different. Phones should provide an Ethernet port to link the phones to the router so that the Ethernet cable can be plugged into to provide Internet access or disconnected from that port. The Ethernet is ideal for setting up IoT networks within a range of 100 meters where the devices can be located. It offers a networking solution for IoT devices with low latency and maximum data speed with Cat5e cables.(Priya 2018.)

2.3.7 Other Communication Technologies and Protocols

Other communication technologies are frequently used on the Internet of Things like Zigbee, Z-wave, CAN bus, KNX, and some application-specific others. Users can choose between different technologies due to specific requirements of applications. The ZigBee and Z-Wave short-range wireless technologies are used for remote monitoring and control. However, their specifications and applications are different. Both technologies are ideal for home-area networks (HANS), becoming more widespread in the U.S (Frenzel 2012). TABLE1 shows a comparison between Zigbee and Z-Wave technologies. ZigBee technology's operational frequency is between 2.4 to 2.483 GHz, where Z-Wave operates on 908.42 MHz. The modulation method on ZigBee is OQPSK, wherein Z-Wave has the GFSK modulation method. ZigBee's data transfer rate is 250 Kbits/S, where Z-Wave is in the range of 9.6 to 40 Kbits/S. The ZigBee data transfer range is up to 10 meters, but the Z-Wave has a data transfer range of up to 30 meters.

TABLE1. The comparison between Zigbee and Z-Wave technologies.



ZIGBEE AND Z-WAVE SPECIFICATIONS AND CAPABILITIES

Technology	Frequency	Modulation	Data rate	Range	Applications
ZigBee	2.4 to 2.483 GHz	OQPSK	250 kbits/s	10 m	Home automation, smart grid, remote control
Z-Wave	908.42 MHz	GFSK	9.6/40 kbits/s	30 m	Home automation, security

A controller area network (CAN bus) is a robust vehicle bus standard that allows microcontrollers and devices to interact with each other's interface without a host controller. It is a message-based protocol, initially developed for copper-saving multiplex electrical wiring in vehicles, but can also be used in many other scenarios. The data is transmitted sequentially for each unit in a frame, but in a way that if more than one unit transmits at the same time, the highest priority unit will proceed while the others back off (Corrigan 2016.) KNX is an open-source standard for commercial and urban building automation. KNX devices can manage lighting, blinds, shutters, security systems, energy management, audio-video, white goods, displays, and remote control. It can use twisted pair (in a tree, line, or star topology), powerline, RF, or IP links. On this network, the devices form distributed applications, and tight interaction can work.(knx 2019.)

2.3.8 MQTT

The basic protocol for exchanging messages between devices is MQTT (Message Queuing Telemetry Transport), used regularly in IoT applications. MQTT is designed for large, low data traffic networks to reduce the volume of data. The data transfer takes place over TCP using MQTT, and it could be SSL encrypted. A data transfer model of a "publisher-subscriber" is used in MQTT. The Publisher-subscriber model implies that one central core (an MQTT broker) is used to exchange messages. An MQTT broker is a central hub that links MQTT publishers with MQTT subscribers (typically in a cloud on the public Internet). MQTT publishers send messages, and MQTT subscribers subscribe to receive messages. Messages are split into "topics," and the system can either "publish" to the topic or "subscribe" to it. A specific "topic" can have multiple MQTT subscribers. Messages are shared within a topic as they are received by the MQTT broker and then sent to the subscribed devices. A device (electrical socket) may be a publisher for some subjects at the same time (publishing the computed values) and a subscriber for other subjects at the same time (responding to output control commands). The MQTT protocol categorizes messages according to the topic. Each message must contain a subject that the MQTT broker may use to pass the message on to the subscribed MQTT recipients. In this way, every message has a payload transmitted to the subscribers (MQTT. Nieto-Products 2017.) A scenario of controlling a lamp based on the MQTT protocol is shown in Figure 6.

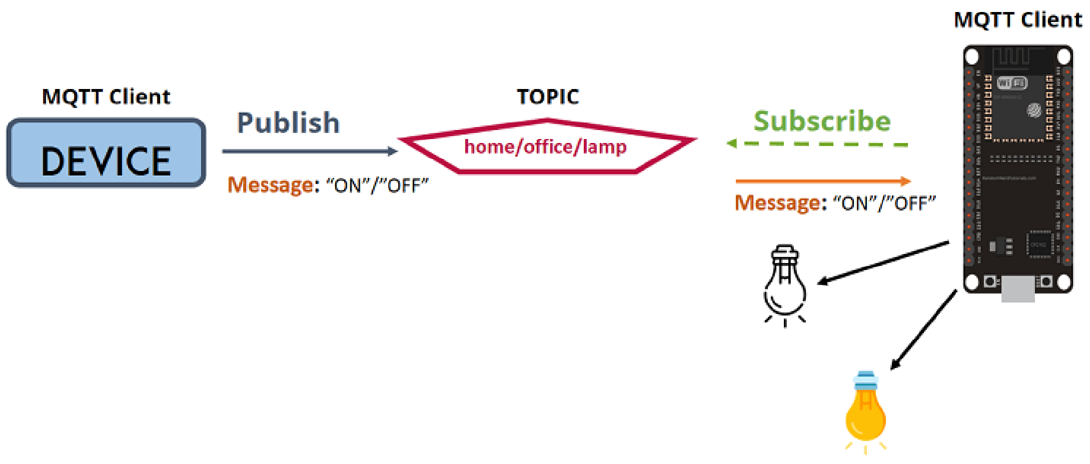


FIGURE 6. A scenario of controlling a lamp based on the MQTT protocol (adopted from Santos 2017,16)

2.4 Description and Comparison of Three Multi connection Gateways

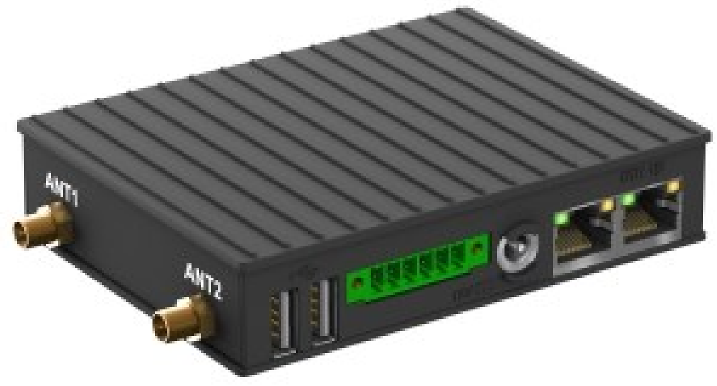
The things gateway from the things network LoRaWAN community, which costs 300 Euros. The Things Gateway (Picture 1) is an open-source LoRaWAN compliant gateway that offers a network coverage range up to 10KM with WiFi and Bluetooth 4.2 modem, Ethernet connection port embedded for potential communication protocols or DIY add-ons into the LoRaWAN protocol, and an XBEE slot. It could support as many as 10,000 nodes.(The Things Gateway 2019.)

Picture1. The Things Gateway (The Things Gateway 2019)

NXP IoT gateway from NXP semiconductors N.V, which costs 499 Dollars. The NXP IoT gateway (Picture 2) is a WiFi, Zigbee, Bluetooth low-energy 4.2 modem multi-link gateway, and an Ethernet communication port. It also hosts NFC and USB type A for further dongle access and add on connectivity. It can connect to versatile cloud services and has a cloud-based system management mobile software reference design. (IoT Gateway Solution.)

Picture 2. The NXP IoT Gateway (IoT Gateway Solution)

Compulab IOT-GATE-iMX8 from Compulab Starting from 114 Dollars. Compulab IOT-GATE-iMX8 (Picture 3) is powered by NXP I MX8M-Mini Processor, Cortex-A53 Quad-Core. It is provided with up to 4 GB RAM, and 128 GB eMMC with LTE modem, WiFi 802.11ac, Bluetooth 5.0, Ethernet, USB2, RS485/RS232, CAN-FD with custom I / O. It works in a range of temperatures from -40C to 80C and input voltage from 8V to 36V. It also has a custom operating system for Debian Linux and support for Docker and Microsoft Azure IoT.(Compulab IOT-GATE-iMX8.)



Picture 3. Compulab IOT-GATE-iMX8 (Compulab IOT-GATE-iMX8)

FIGURE 7 demonstrates a comparison between connections of these three gateways. The Things gateway has WiFi, Bluetooth, LoRaWAN, ZigBee, and a physical Ethernet port where the NXP IoT gateway use WiFi, BLE, ZigBee, NFC, and physical Ethernet and USB type A connections, and Compulab IoT gateway iMX8 has WiFi, BLE, and physical connections of Ethernet, USB V2, RS485, RS232, and a CAN-FD port.

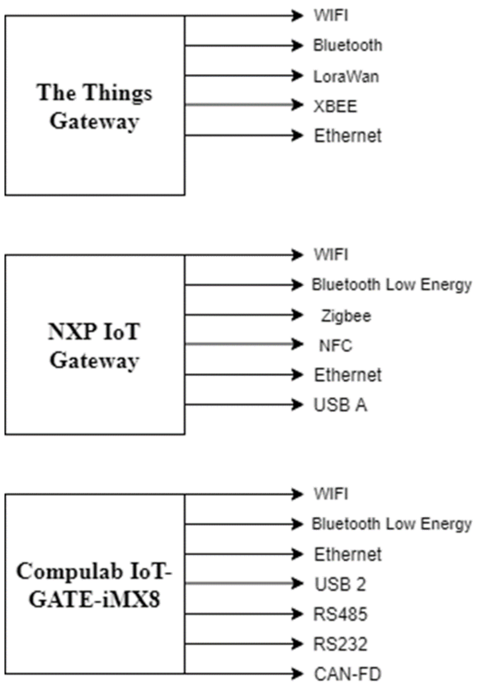


FIGURE 7. Overview diagram of gateways connectivity in comparison

2.5 MCU's Feature Comparison in Gateways

Microcontroller units are the most crucial hardware part of the gateway. Gateway is very dependent on MCU's computational power, where MCU is the central brain of the device and process all communication and ordering services and tasks. Communication speed and multi-threading abilities are the leading performance factor of MCU. (Wu, Qiu & Zhang 2020.)

TABLE2. MCU specification comparison table.

Name	Type	Frequency	Core	RAM	Connections	Interfaces
PIC32-MZ2048EFM	PIC	200 MHz	single	512 KB	Ethernet	-
NXP i.MX6UL	ARM Cortex-A7	900 MHz	single	128 KB	Ethernet, USBA	MIPI
NXP i.MX8M Mini	ARM Cortex A53	1.8 GHz	Quad	256 KB	Ethernet, USB2, RS485, RS232	MIPI PCIe

The Things Gateway use PIC32-MZ2048EFM (MIPS M-class core) that costs 12 Dollar, with the following features: 200 MHz/330 DMIPS, MIPS M-class core; four 64-bit accumulators; single-cycle MAC, saturating and fractional math; IEEE 754-compliant; dual Panel Flash for live update support; and FPU for fast single.(The Things Gateway 2019.)

NXP IoT Gateway Solution use i.MX6UL (ARM Cortex A7) which costs 24 Dollar, with the following features: single-core ARM Cortex-A7; multilevel memory system; smart speed technology; dynamic voltage and frequency scaling; multimedia powerhouse; Ethernet interfaces; human-machine interface; advanced security; and integrated power management.(IoT Gateway Solution.)

IOT-GATE-iMX8 uses NXP i.MX8M Mini (ARM Cortex A53) (67 Dollar) with the following features: quad symmetric Cortex-A53 processors; PCI Express (PCIe); USB 2.0 OTG controllers with integrated PHY interfaces; Ultra Secure Digital Host Controller; Gigabit Ethernet controller with support for Energy Efficient Ethernet; UART, I2C; on-chip RAM (256 KB + 32 KB); temperature sensor with programmable trip points; 32/16-bit DRAM interfaces; Video Processing Unit; and MIPI Interface.(Compulab IOT-GATE-iMX8.)

2.6 Modular Open-Source Implementation Background

Due to the gateway contrast in 2.4 and 2.5, an overall view of a gateway's essential parts can be obtained. Mantranic gateway must use clock frequency adjustable Multicore MCU. Mantranic gateway will use wireless connections of WiFi, Bluetooth, LoRaWAN, and physical connections of Ethernet, USB V2, and GPIO. Mantranic gateway needs to use a highly tolerated switching power, Ensuring high performance of computing and response time. The data storage memory module ensures that the user can have an application tailored data log. USB to UART Bridge and status indication modules are needed to make the platform more user friendly and easy to modify.

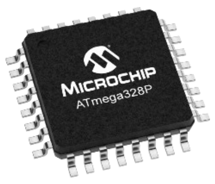
2.7 Selected Hardware Modules

This study intends to develop a cost-effective open source solution, so it is necessary to pick the open-source and moderately inexpensive components with low implementation costs and try to get the most out of it. The following sections describe chosen microcontroller units for thesis work. ESP32 is a 32-

bit LX6 dual-core SoC Xtensa microprocessor up to 600 MIPS with a single 2.4 GHz WiFi and Bluetooth hybrid chip equipped with the 40 nm ultra-low-power semiconductor, which costs 6 Euros. It aimed to achieve the best performance in power and RF, demonstrating robustness, flexibility, and reliability in various applications and control scenarios.(espressif 2020.)

Picture 4. ESP32 SoC (WiFi, Bluetooth, Ethernet, SD card) from Espressif systems (espressif 2020)

ESP32 features all of the states of the art low-power chip characteristics, including fine-grained clock gating, multiple power modes, and dynamic power scaling. For example, in an application scenario for the low-power IoT sensor hub, ESP32 is automatically woken up only when a predefined condition is noticed. A low-duty cycle is used to minimize the amount of energy that the chip expends. The power amplifier output is also adjustable, leading to an ideal trade-off between a range of communication, data rate, and power consumption. It allows updating firmware with OTA, Ethernet, and SD card drive. ESP32 also has all the libraries available to build firmware in Arduino IDE by the C and C++ programming languages (espressif 2020.) Due to the limited number of GPIO's on ESP32, it was needed to expand GPIO's, so that it is needed to add an Atmega 328 microcontroller to the project to support ESP32 as the essential client-server operation part and extend GPIO's, which cost 1.14 Euro.(microchip 2020.)


Picture5. Atmega 328 microcontrollers from microchip technologies (microchip 2020)

The high-performance microchip Pico power 8-bit AVR RISC-based microcontroller, which costs 8 Euro, integrates 32 KB ISP flash memory with read-write capabilities, 1024B EEPROM, 2 KB SRAM, 23 general-purpose I / O lines, 32 general-purpose working registers, three dynamic timer/counters with comparable modes, internal and external interrupts, a byte-oriented 2-wire serial interface USART, SPI serial interface. The system works in the range from 1,8 - 5,5 volts.(microchip 2020.)

Picture 6. RA-02 LoRa module from the AIthinker company (ai-thinker 2020)

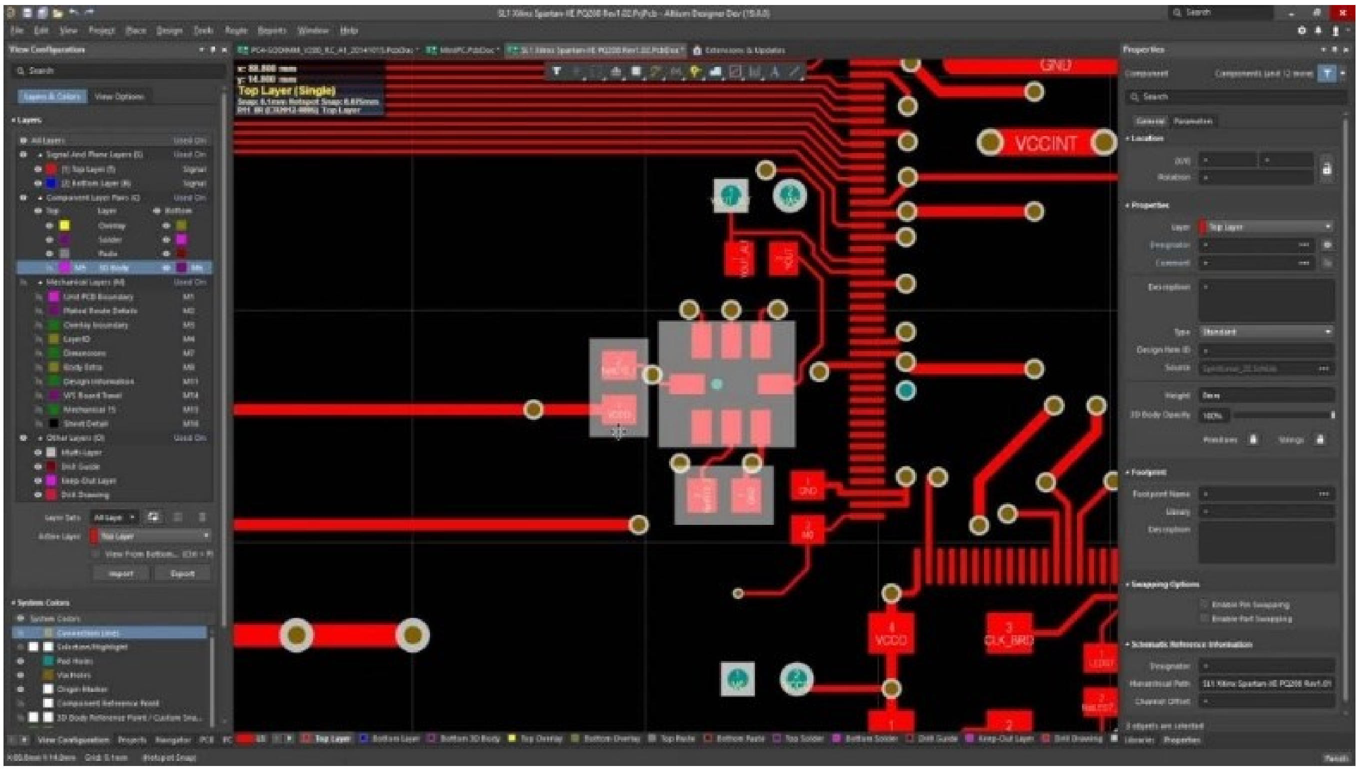
RA-02 LoRa 433MHz module developed by AIthinker, based on the SX1278 chip. The SX1278 RF module is used basically for communication with a long-range spread spectrum. It can tolerate the minimization of current usage. The SX1278 has a high precision of -148 dBm with a power output of + 20 dBm, a long transmission distance, and high-reliability thanks to the proprietary LoRa modulation technology from Semtech. At the same time, LoRa modulation technology in anti-blocking and selection also has distinct advantages compared to traditional modulation technology, the size, interference, and power consumption cannot be considered to solve the conventional design. It supports the automatic detection of RF signals, CAD model, and ultra-high-speed AFC. The AIthinker LoRa module also supports modulation modes on FSK, GFSK, MSK, GMSK, LoRa, and OOK (ai-thinker 2020.) The CP2102 USB to UART Bridge offers a complete solution with the plug and play interface, including royalty-free drivers. This USB 2.0 compliant computer has 20 digital I / O connectors and is available in a QFN28 package of 5x5 mm, which costs 1Euro.(silabs 2020.)

Picture7. CP2102 UART to USB chip from SiliconLab (silabs 2020)

2.8 Software and IDE for Implementation

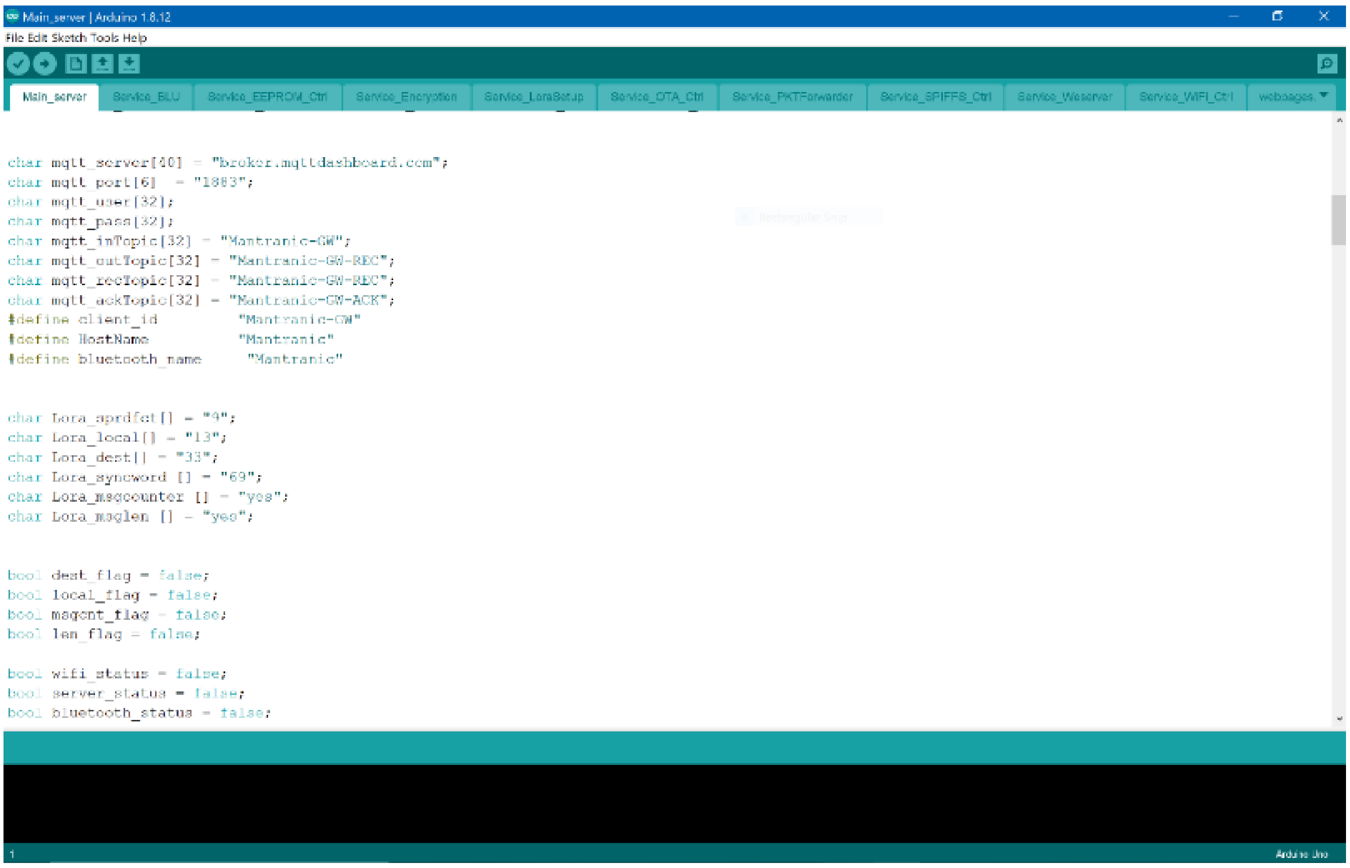
The thesis work uses the Altium designer for hardware design and Arduino IDE with C and C++ programming for firmware programming. Altium Designer is a software kit for printed circuit boards and electronic design automation. The Altium designer suite covers four key functional areas: schematic capture, 3D PCB design, field-programmable gate array (FPGA) creation, and release/data administration. Notable features listed in the reviews include: integration with several component distributors allows search for components and access to manufacturer's data; interactive 3D editing of the board and

MCAD export to STEP; cloud publishing of design and manufacturing data; lastly, simulation and debugging of the FPGA can achieve using the VHDL language and checking that for a given set of input signals. The expected output signals would be generated. FPGA soft processor software development tools (compiler, debugger, profiler) are available for selected embedded processors within an FPGA. (altium 2020.)



Picture 8. Altium Designer software (altium 2020)

The Arduino Integrated Programming Environment (IDE) is a cross-platform software developed by C and C++ functions (for Windows, macOS, Linux). It is used to code and upload programs to compatible Arduino boards and other vendor development boards (Arduino IDE) to aid third-party cores. The IDE source code is released under version 2 of the GNU General Public License. The Arduino IDE supports the C and C++ languages using particular code structuring guidelines. A software library from the Wiring project is provided by the Arduino IDE, which provides many standard input and output procedures. User-written code only requires two primary functions, to start the sketch and the main loop compiled and joined with a program stub main() into an executable cyclic program with the toolchain of GNU included with the IDE distribution. The Arduino IDE uses the Avrdude software to transform the executable code into a hexadecimal encoding text file, loaded into the Arduino board by a firmware loader program. By default, Avrdude is used to flash the user code to official Arduino boards as the uploading tool. Picture 9 demonstrates a view of the Arduino IDE.(Arduino 2020.)

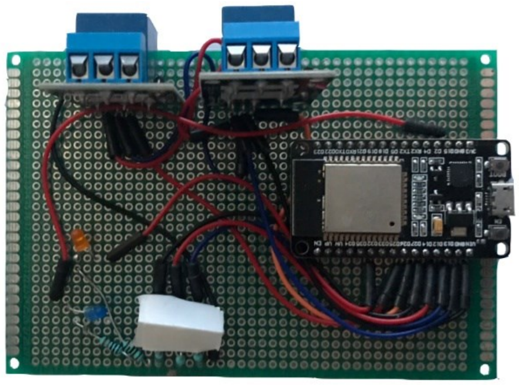


```
char mqtt_server[40] = "broker.mqttdashboard.com";
char mqtt_port[6]  = "1883";
char mqtt_user[32];
char mqtt_pass[32];
char mqtt_inTopic[32] = "Mantranic-GW";
char mqtt_outTopic[32] = "Mantranic-GW-REC";
char mqtt_recTopic[32] = "Mantranic-GW-REC";
char mqtt_ackTopic[32] = "Mantranic-GW-ACK";
#define client_id          "Mantranic-GW"
#define HostName           "Mantranic"
#define bluetooth_name      "Mantranic"


char Lora_sprdfct[] = "9";
char Lora_local[] = "13";
char Lora_dest[] = "33";
char Lora_syncword [] = "69";
char Lora_msgcounter [] = "yes";
char Lora_msglen [] = "yes";


bool dest_flag = false;
bool local_flag = false;
bool msgcnt_flag = false;
bool len_flag = false;

bool wifi_status = false;
bool server_status = false;
bool bluetooth_status = false;
```

Picture 9. Arduino IDE (Arduino 2020)

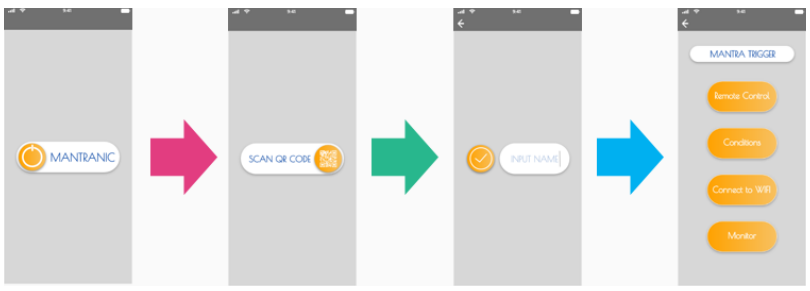
3 TESTING AND IMPLEMENTATION

To test the ESP32 capabilities, a test prototype gateway has been made (Picture 10) with the ESP32 development board to test some trigger actions and connection stability. The test prototype was consisting of two relay modules, two single-color LEDs, and one RGB line as well as ESP32 DevBoard and connected to a ruuvi sensor tag to read the sensor data.



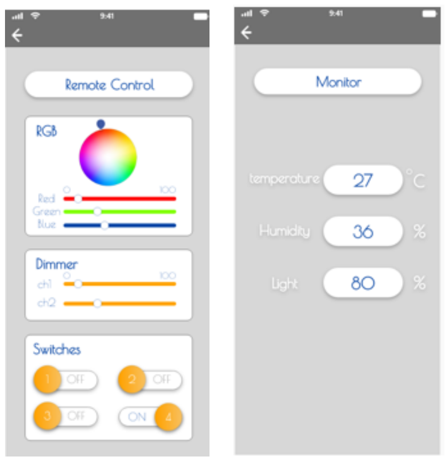
Picture 10. Test prototype gateway

This module was controlled with an android app through WiFi and Bluetooth protocol, which was able to control various outputs, show the status and fetch the sensor data and show in the monitor section of the mobile app. Also, it helped to measure the response latency and actions in light and massive workload tests.



Picture 11. Start and main page of the test application.

This test abled to trigger an action node and fetch sensor tag data through WiFi and BLE protocols, and the test board was working seamlessly in the range of 25-meter radius on WiFi and 52-meter radius on BLE. Because of enormous request amounts through the RGB control value stream, the main ESP32 module was getting heated up to 62 degrees centigrade, a struggle to handle such an issue for the actual gateway board.



Picture 12. Remote control and monitor pages of the test application.

The other main problem was because of nonregulated wiring. Some package was lost while triggering the action nodes, which was handled by acknowledged messages to ensure that the end nodes received the package. To test the regulated radio connections on the ESP32 Module, the chip producer provided test software to check the chip's performance through different methods.

3.1 Hardware Design

Hardware design consists of two parts schematic and PCB design, both based on various standards. In this project, four basic standards should be considered through the design process: signaling standard, routing standard, power management standard, and heat conduction standard. Some spaces must be given to various RF and power components to prevent electromagnetic field effect and radiofrequency blocking points due to the mentioned concerns. The design could be a compact form of PCB in 5*8 CM, but because of a reliable approach for alignment with standards, it has been designed in 10*10 CM. In the schematic diagram, due to modular components, mostly schematics are officially recommended by manufacturers, and the only thing to design was switching power and two voltage conversion circuits alongside pin diagram for the specific design of Mantranic gateway.

3.1.1 Schematics

All schematics were designed and simulated by the Altium Designer, and the simulation output of signal integrity and radio blocking points were passed. In this section, the main parts with schematics will be reviewed, and the rest can be found in appendixes. FIGURES 8 to 11 are demonstrating the main parts of the Mantranic gateway schematic. ESP32 is the central controlling unit in this project, where all necessary components are connected. Due to standardized dependent components that may affect the performance, the ESP32 module prevents such influences in this project.

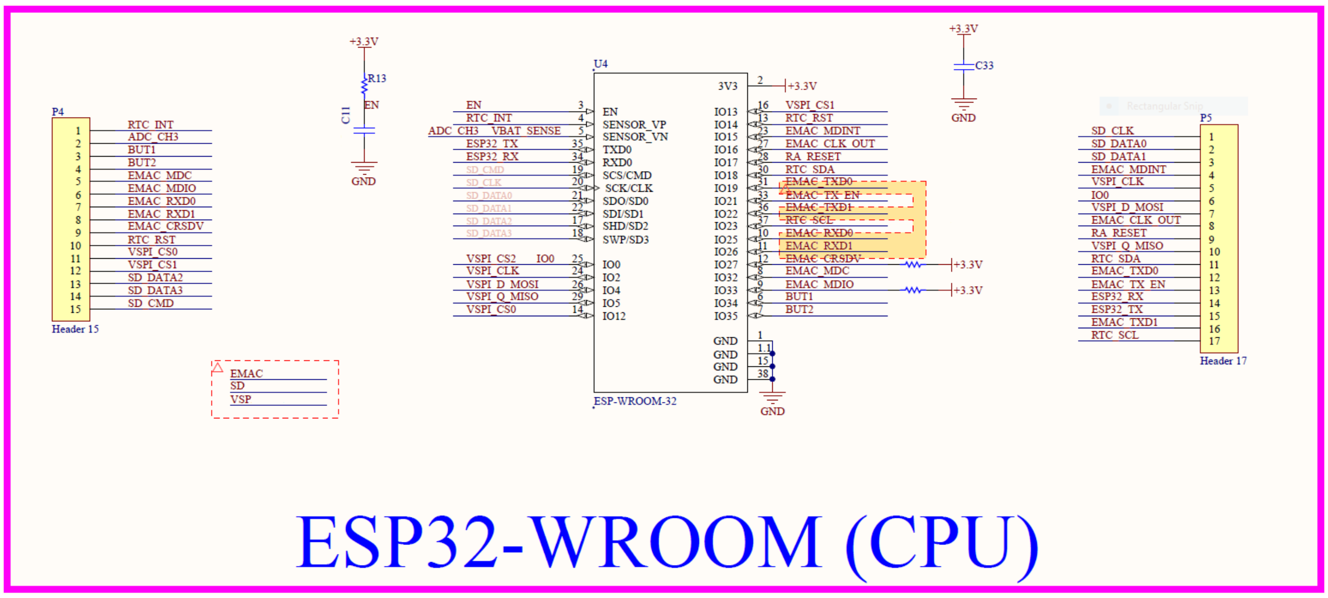


FIGURE 8. ESP32 schematic.

AI-thinker RA-02 is the LoRa connection unit that provides long-range connectivity (1KM-12 KM depends on topographical blockers) over 433 MHz radiofrequency. It also gives the opportunity of getting connected to LoRa pico-satellites to have global internet independent connectivity.

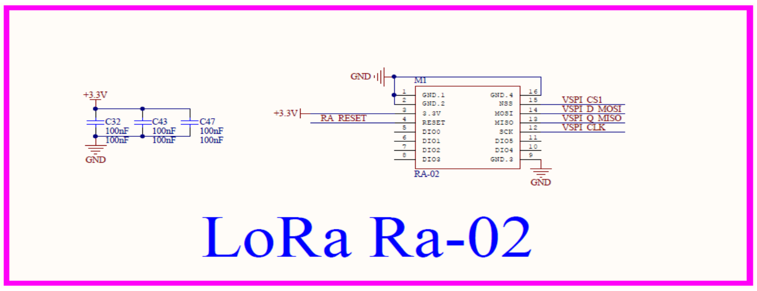


FIGURE 9. LoRa RA-02 Schematic.

The ESP32 was made to work as a server and has limited IO pins; ATmega32 acts as a client to handle micro-processes and work as an I/O Expander. AVR microcontroller needs an external clock crystal component as an oscillation unit in which a 16000MHz crystal is used in this project to meet the maximum computational power.

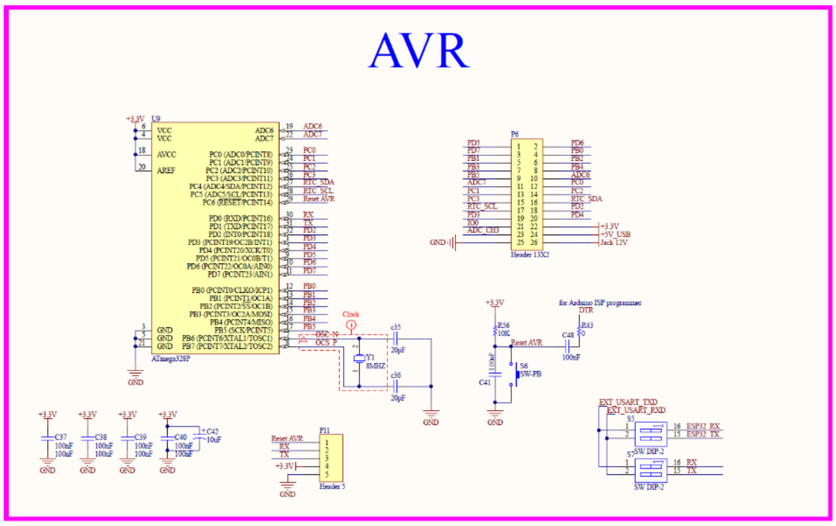


FIGURE 10. AVR ATmega32 Schematic.

A micro SD slot was added for data storage of sensors data and performance logs in the JSON format file for later data visualization, or system report for dashboard and inter-application uses. MicroSD slot also helps the user load the firmware for updates through a micro SD memory card if users prefer local updates.

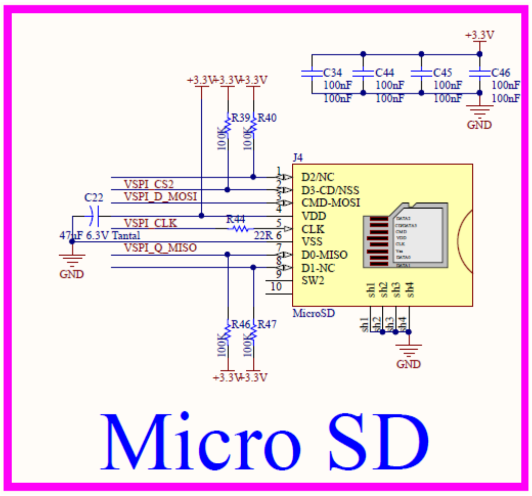
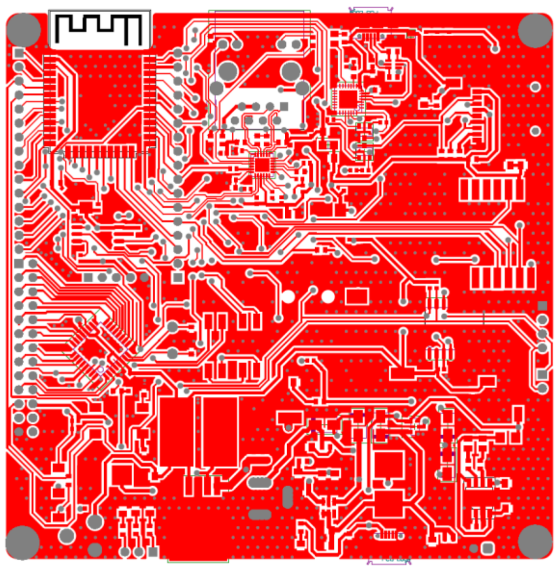
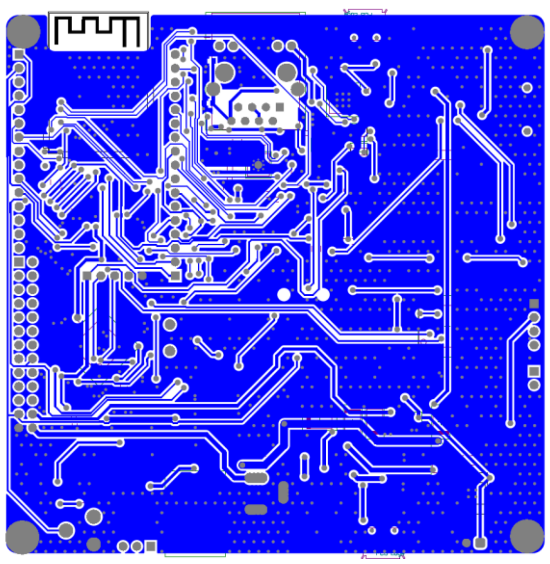


FIGURE 11.SD Card Slot Schematic.

3.1.2 PCB Layouts

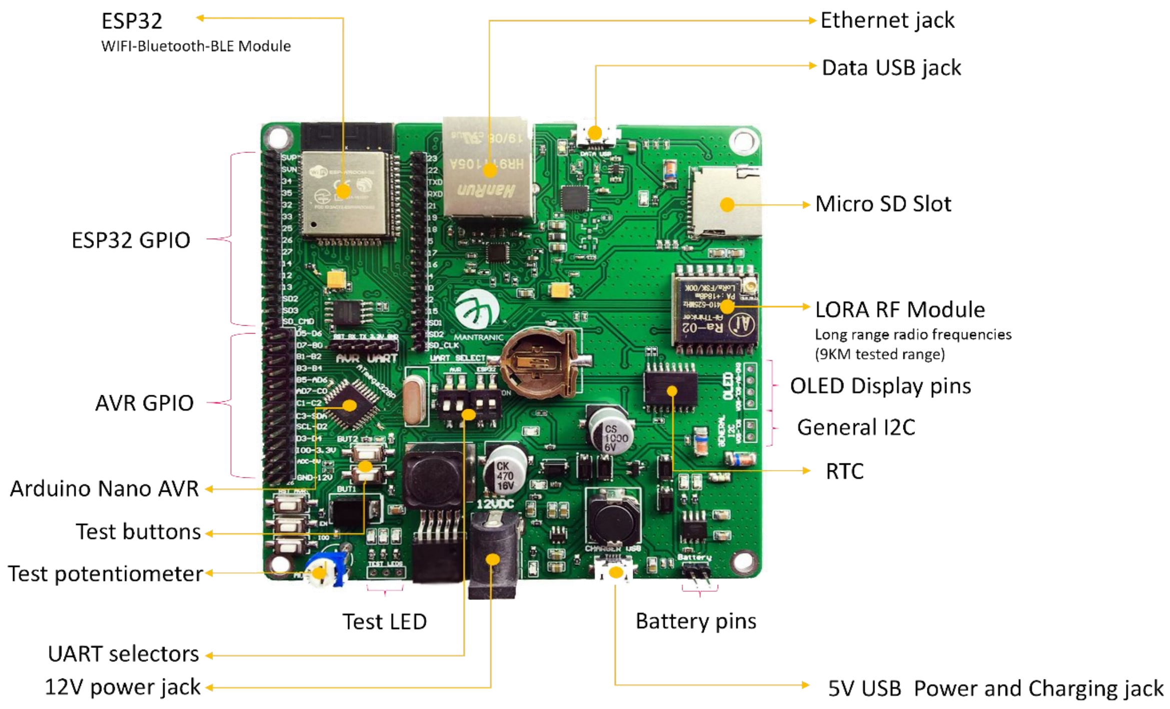
Mantranic gateway is a double layer and isolation coated 10*10 CM PCB with 2 mm thickness, and its design is just the conversion of the schematic to PCB in Altium Designer. All polygons and routings are based on PCB design standards. Picture 13 depicts the back and front sides of Mantranic gateway PCB.

Picture 13. The back and the front side of the PCB layout.

3.1.3 Mantranic Gateway

The final product of hardware designed by this thesis work is demonstrated in Picture 14. As in Picture 14, the Mantranic gateway contains all features mentioned before in a modular format where all specific components and outputs are accessible to the user for different uses. The primary and secondary MCU GPIO sets are in standard development board format to keep the similarity of decoration for users with prior experience with famous ESP32 V2 and Arduino nano development boards. In the final product, Mantranic gateway enables the user to power the gateway with three different sources for different project types. Mantranic contains test modules on board, enabling the user to test the simple operations' physical inputs and outputs. Mantranic also has OLED display pinouts where users can mount an OLED display and an I2C connection pinout in case of project need.



Picture14. The Mantranic Gateway circuit, which costs 40 Euros.

3.2 Software Design

Mantranic gateway firmware consists of two essential parts, driver and web server service interface. Driver part coded by C and C++ language in Arduino IDE and contains all pin configuration and peripherals connection codes where all variables of connection and packet forwarding can get modified by web service part to configure the gateway and other peripherals. Web service contains all wizard UI elements to configure the gateway by the user over a web browser.

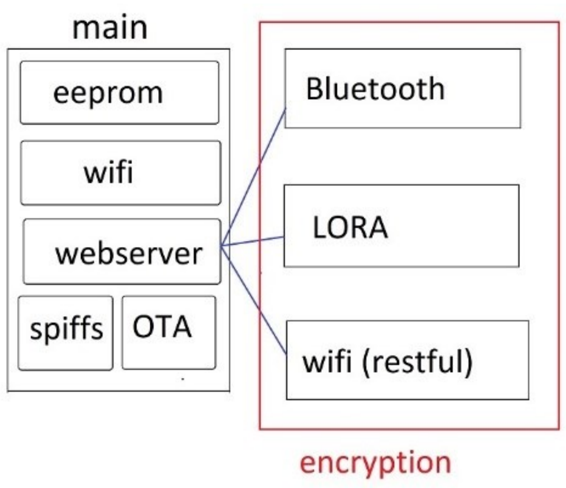


FIGURE 12. Mantranic gateway firmware flow Diagram.

3.2.1 Driver

Driver codes are written in C and C++ language combinations in Arduino IDE, and it consists of 9 parts that are bonded together in an Arduino package where it is available in Arduino libraries, which enables to use or modify them in the different use case. Mantranic also provided libraries for the MicroPython environment, which widens the technology variety for users with different programming skills. Due to API based firmware of Mantranic gateway, it would be easy for users to implement any kind of tailored software.

3.2.2 Main Server

The main server part is to set up all connection methods consist of WiFi, Bluetooth, LoRa, and MQTT server configuration. In these matters, the server part does event handling for all error possibilities, e.g., wrong or incomplete configuration or connection lost. Also, it handles the security of connection and resource allocation for each part from each ESP32 core. The central server includes 17 header files to handle specific configurations. The first section handles the configuration of WiFi, MQTT server, Bluetooth and LoRa connections, the local server web service's login page credentials, and the central server's time/date setting. The second section handles the events during each connection attempt and task running process in the first part. In the third part, it tries to test the connections in message forwarding mode and print the result in the serial monitor in the Arduino IDE. In case of acknowledging failure, the call back function triggers a retry connection attempt after 5 seconds. In the fourth part, it set up the timekeeper on a real-time clock chip. In the fifth part, after successful connection attempts, it saves the credentials that user-provided on EEPROM memory and run a test task to test the MQTT server performance for handling user-defined request (APPENDIX 1).

3.2.3 Bluetooth Service

Bluetooth service is to handle Bluetooth connection to register different Bluetooth devices and handle those devices' messages. As ESP32 provides two different types of Bluetooth connection, Bluetooth 4.2 and BLE (Bluetooth Low Energy), it switches between these modes by the node's type, and for power consumption optimization reason, it stays preferably on BLE where the power consumption is in its minimum level (APPENDIX 2).

3.2.4 EEPROM Control Service

EEPROM control service provides memory management and bit commit rules for ESP32 EEPROM. The serial run time method of this project's memory management chip process goes through incremental register selection on memory. EEPROM control service also manages external memory drive and data buffer stream. The memory management model in Mantranic is module-based, which makes an efficient registry allocation, high response, and reliable performance (APPENDIX 3).

3.2.5 Encryption Service

Encryption service focuses on data encryption for all communications through each connection type. As a matter of security and data protection to prevent unauthorized access and penetration attempts, two 128-bit data encryption layers were used to align with advanced encryption standards. Also, Mantranic includes a hardware encryption chip all bounded in one complicated numeric inflation mathematical method. Mantranic uses a randomized handshake exchange timer in 12 seconds cycle, which is off due to increased power consumption. Still, the user can set it on where the use case needs to ensure concrete security (APPENDIX 4).

3.2.6 LoRa Connection Setup

The LoRa connection setup, the LoRa connection authentication and spread factor, and event handler were set for packet forwarder mode. Mantranic handles packets in JSON format; here, JSON parsing was set and, in case of acknowledgment, adds a new object which includes payload, received signal strength indication, and signal to noise ratio. Due to different valid LoRa frequencies in different countries, also users can set the valid radio frequency range to operate on LoRa connection (APPENDIX 5).

3.2.7 On The Air Firmware Update

ESP32 module contains the air firmware update feature. In this part, for ease of use, the firmware update was provided over WiFi and Bluetooth connection where Mantranic gateway was set to check the firmware version index and in case of mismatch download's new firmware file over an async web server request from the user device, setup and reboot the gateway (APPENDIX 6).

3.2.8 Packet Forwarder Service

Packet forwarder service is the message handling part of the Mantranic, where all kinds of message handling methods were set over different connection types. In every connection type, there are three main factors that the gateway must handle. The first one is the gateway address for each connection type. The second one is the sender's address, which must be identified and recorded. The third part is the message-id, which increments every new message that each connection receives or broadcasts. Also, the event handler of any possible event during packet forwarding was set up (APPENDIX 7).

3.2.9 SPIFFS Control Service

The serial peripheral interface flash file system or SPIFFS is a lightweight file system for microcontrollers with an SPI flash chip. SPIFFS makes it possible to access the flash memory as if it was a regular file system on a computer with a much lighter weight and a more straightforward method. SPIFFS can compress files to make a better memory managed system where there is a restriction to a specific memory space on the ESP32 chip. In this part, configuration files and log files management over SPIFFS and event handlers for the SPIFFS were set up (APPENDIX 8).

3.2.10 Web Server Service

Local web server is a service on Mantranic gateway for easily set up all the necessary configurations over a local or online web application for all connection types, e.g., WiFi, Bluetooth, LoRa, MQTT alongside the OTA firmware update, system status, rebooting, and hard reset features. Web service also has all kinds of event alerts for every occurrence on the system during configuration, ensuring that the user understands unsuccessful or successful configuration events (APPENDIX 9).

3.2.11 Web Server Service User Interface

This part contains HTML codes bounded in C language in Arduino IDE to make web application visuals run on Mantranic local configuration web application (APPENDIX 10). UI consists of 6 main sections for configurations, which shall be described briefly here. The first main section is the config panel. With this section's help, users can go through all available configurations such as WiFi, server, LoRa. The user can also see the gateway's real-time status and see the firmware version and install the newer version if it exists. This pane lets the user reset or hard reboot the device. The second main section is the WiFi config. This section can be used to see all available WiFi connections in range and connect to the one they prefer. The third is the server config, which can set up the MQTT server configuration and input the preferred credentials. The fourth section is the LoRa config, allowing users to set the LoRa connection configuration and the message length. The fifth is the status used to scout the system status like uptime, network, and server status. The sixth and last main section is the firmware update. Using this section, users can get the newer version of the firmware setup file and upload it to the device. All the screenshots are available in appendixes (APPENDIX 11).

4 CONCLUSION

Currently, doing an IoT project is mostly based on the gateway that is used and its capabilities. Most of the gateways in the market are single-purpose gateways with limited connection functionality. In some projects, the gateways with various connections cost more than the whole project's total budget when the maker's community is considered. In projects with a vast area or many nodes to serve, multiple gateways need to be used; synchronizing those gateways and operating multiple gateways costs much time, effort, and money. In this thesis work, the aim was to make a simple solution with affordable hardware for makers projects and industrial projects. In this thesis work, the hardware design's main challenges were isolation, power supply robustness, and operating temperature. During the project implementation, all those milestones have been met accordingly. Now Mantranic gateway can perform point to point connection in 10 KM in the urban area and have the capacity of approximately 500 nodes to serve over each connection type while all of them can be operated simultaneously. Mantranic can log data locally on an SD card and broadcast the data over asynchronized restful API and MQTT brokers. It is operable over a 3.7V battery pack for more than two weeks and also can get connected to the LoRa picosatellites to broadcast data over LoRa without any other connections globally. Further work on this system can be a vertical IoT dashboard in company with the Mantranic gateway for data analysis and nodes setup and configurations for data log and trigger actions.

REFERENCES

Adebayo, A.Chaubey, M.& Numba, L. 2019. The Fourth Industrial Revolution and How It Relates to The Application of Internet of Things (IoT). Berlin. Journal of Multidisciplinary Engineering Science Studies (JMESS).

microchip. Article on the website of Microchip Technologies. https://www.microchip.com/wwwproducts/en/ATmega328p. Accessed 6 September 2020.

altium. Uncompromising PCB Design Experience. https://www.altium.com/altium-designer. Accessed 6 September 2020.

Ai-thinker. RA-02 Product specification v1. http://wiki.ai-thinker.com/_media/lora/docs/c048ps01a1_ra-02_product_specification_v1.1.pdf. Accessed 6 September 2020.

Arduino. Article on the website of Arduino. https://www.arduino.cc/en/main/software. Accessed 6 September 2020.

Wu, Z. Qui, K. Zhang, J. 2020. A Smart Microcontroller Architecture for the Internet of Things. Basel. MDPI Sensors (Basel).

Gerber, A. Kansal, S. Simplify the development of your IoT solutions with IoT architectures. https://developer.ibm.com/articles/iot-lp201-iot-architectures/. Accessed 25 October 2020.

Blom, J. 2017. Serial Communication. Sparkfun Electronics. Available: https://learn.sparkfun.com/tutorials/serial-communication/all. Accessed 6 September 2020.

Cherkasova, V. 2018. IoT devices and gateways. Student Circuit. Available: https://www.student-circuit.com/learning/year3/iot/iot-devices-and-gateways. Accessed 6 September 2020.

Compulab IOT-GATE-iMX8. Article on the website of CompuLab Ltd. https://www.compulab.com/products/iot-gateways/iot-gate-imx8-industrial-arm-iot-gateway/. Accessed 6 September 2020.

Silabs. Article on the website of SiliconLabs. https://www.silabs.com/. Accessed 6 September 2020.

Durda, F. 2014. Serial and UART Tutorial. The FreeBSD project. Available: https://www.freebsd.org/doc/en/articles/serial-uart/. Accessed 6 September 2020.

Desai, N. 2016. What is an IoT Gateway, and How Do I Keep it Secure. GlobalSign GMO internet group. Available: https://www.globalsign.com/en/blog/what-is-an-iot-gateway-device. Accessed 6 September 2020.

espressif. Article on the website of Espressif Systems. https://www.espressif.com/sites/default/files/documentation/esp32-wrover-e_esp32-wrover-ie_datasheet_en.pdf. Accessed 6 September 2020.

Franzel, L. 2012. What is the Difference Between ZigBee and Z-Wave?. Endeavor Business media. Available: https://www.electronicdesign.com/technologies/communications/article/21796052/whats-the-difference-between-zigbee-and-zwave. Accessed 6 September 2020.

Harwood, T .2018. LoRa Network Protocol and Long-Range Wireless IoT. postscapes. Available: https://www.postscapes.com/long-range-wireless-iot-protocol-lora/. Accessed 6 September 2020.

IoT Gateway Solution. Article on the website of NXP Semiconductors N.V. https://www.nxp.com/design/designs/iot-gateway-solution:IOT-GATEWAY-SOLUTION. Accessed 6 September 2020.

Liu, Y. 2017. Evaluation and Measurement of IoT Gateways: Mid Sweden University. Bachelor thesis.

Latif, Z. 2018. Wireless Protocols for the Internet of Things. BlueEast. Available: https://medium.com/blueeast/wireless-protocols-for-internet-of-things-7ed0bd860c66. Accessed 6 September 2020.

LoRa- (Long Range) Network and Protocol Architecture with Its Frame Structure. Article on the website of techplayon. http://www.techplayon.com/lora-long-range-network-architecture-protocol-architecture-and-frame-formats/. Accessed 6 September 2020.

Mehl, B. 2018. 6 Communication Protocols Used by IoT. Kisi. Available: https://www.getkisi.com/blog/internet-of-things-communication-protocols. Accessed 6 September 2020.

MQTT. Article on the website of Nieto-Products. https://www.netio-products.com/en/glossary/mqtt. Accessed 6 September 2020.

Priya.2018. Ethernet Technology: IOT Part 23. EE World Online Network. Available: https://www.engineersgarage.com/tutorials/ethernet-technology-iot-part-23/. Accessed 6 September 2020.

Rose,k. Eldridge, S. Chapin, L. 2015.THE INTERNET OF THINGS: AN OVERVIEW
Understanding the Issues and Challenges of a More Connected World. Geneva. The Internet Society (ISOC).

Rouse, M. 2020. internet of things (IoT). IoTAgenda. Available: https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT. Accessed 6 September 2020.

AI-thinker. Article on the website of AIthinker Ltd. http://wiki.ai-thinker.com/_media/lora/docs/c048ps01a1_ra-02_product_specification_v1.1.pdf. Accessed 6 September 2020.

Stokes, P. 2018. Data-Driven Investor. 4 Stages of IoT architecture explained in simple words. Available:
https://medium.com/datadriveninvestor/4-stages-of-iot-architecture-explained-in-simple-words-b2ea8b4f777f. Accessed 6 September 2020.

Santos, R. 2017. What is MQTT, and How It Works? randomnerdtutorials. Available: https://randomnerdtutorials.com/what-is-mqtt-and-how-it-works/. Accessed 6 September 2020.

The Things Gateway. Article on the website of The Things Industries. https://www.thethingsnetwork.org/marketplace/product/the-things-gateway. Accessed 6 September 2020.

Vermesan, O. Friess, P. 2013. Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems. Aalborg: River Publishers.

Corrigan, S. Introduction to the Controller Area Network (CAN). https://www.ti.com/lit/an/sloa101b/sloa101b.pdf. Accessed 25 October 2020.

Knx. A brief introduction to KNX. https://www.knx.org/knx-en/for-professionals/What-is-KNX/A-brief-introduction. Accessed 25 October 2020.

Youngjun, J. Sangsoon, L. 2018. Design and implementation of heterogeneous surface gateway for underwater acoustic sensor networks. Indonesia. International Journal of Electrical and Computer Engineering (IJECE).

Main server code

```
#include <Mantranic.h>
#include <MantraLoRa.h>
#include <MantraETH.h>
#include <MantraTimer.h>
#include <MantraCrypto.h>
#include "webpages.h"
#include <ESPmDNS.h>
#include <Update.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include <DNSServer.h>
#include <EEPROM.h>
#include <ESP32Ping.h>
#include <WiFiUdp.h>
#include <DS3231.h>

WiFiUDP udp;
DS3231 Clock;
MantraTimer timer;


String version_num = "1.0.a";

TaskHandle_t Task1;
TaskHandle_t Task2;

String macAdd = WiFi.macAddress();
const byte DNS_PORT = 53;
IPAddress apIP(192, 168, 4, 1);
String st; String content; int statusCode;
DNSServer dnsServer;

String esid, epass, emqtt_server, emqtt_port, emqtt_user, emqtt_pass , emqtt_topic;
String qsid, qpass, qmqtt_server, qmqtt_port, qmqtt_user, qmqtt_pass , qmqtt_topic;

String qlora_sprdfct , qlora_syncword , qlora_localadd , qlora_destadd, qlora_addlength;


char mqtt_server[40] = "broker.mqttdashboard.com";
char mqtt_port[6]  = "1883";
char mqtt_user[32];
char mqtt_pass[32];
char mqtt_inTopic[32] = "Mantranic-GW";
char mqtt_outTopic[32] = "Mantranic-GW-REC";
char mqtt_recTopic[32] = "Mantranic-GW-REC";
char mqtt_ackTopic[32] = "Mantranic-GW-ACK";
#define client_id          "Mantranic-GW"
#define HostName           "Mantranic"
#define bluetooth_name      "Mantranic"


char Lora_sprdfct[] = "9";
char Lora_local[] = "13";
char Lora_dest[] = "33";
char Lora_syncword [] = "69";
char Lora_msgcounter [] = "yes";
char Lora_msglen [] = "yes";


bool dest_flag = false;
bool local_flag = false;
bool msgcnt_flag = false;
bool len_flag = false;

bool WiFi_status = false;
bool server_status = false;
bool bluetooth_status = false;
const char* ssid = "Mantranic";
const char* password ;

const char* update_username = "admin";
const char* update_password = "admin";


long Day = 0;
int Hour = 0;
int Minute = 0;
int Second = 0;
```

Main server code

```
int HighMillis = 0;
int Rollover = 0;
char timestring[25];
char sdstring[30];
char datetimestring[30];
bool Century = false;
String outgoing;              // outgoing message
byte msgCount = 0;            // count of outgoing messages

WiFiClient espClient;
PubSubClient client(espClient);
AsyncWebServer server(80);

static bool eth_connected = false;

void WiFiEvent(WiFiEvent_t event)
{
  switch (event) {
    case SYSTEM_EVENT_ETH_START:
      Serial.println("ETH Started");
      //set eth hostname here
      ETH.setHostname("esp32-ethernet");
      break;
    case SYSTEM_EVENT_ETH_CONNECTED:
      Serial.println("ETH Connected");
      break;
    case SYSTEM_EVENT_ETH_GOT_IP:
      Serial.print("ETH MAC: ");
      Serial.print(ETH.macAddress());
      Serial.print(", IPv4: ");
      Serial.print(ETH.localIP());
      if (ETH.fullDuplex()) {
        Serial.print(", FULL_DUPLEX");
      }
      Serial.print(", ");
      Serial.print(ETH.linkSpeed());
      Serial.println("Mbps");
      eth_connected = true;
      break;
    case SYSTEM_EVENT_ETH_DISCONNECTED:
      Serial.println("ETH Disconnected");
      eth_connected = false;
      break;
    case SYSTEM_EVENT_ETH_STOP:
      Serial.println("ETH Stopped");
      eth_connected = false;
      break;
    default:
      break;
  }
}

bool testClient()
{

  bool ret = Ping.ping("www.google.com");

  if (ret) {
    //Serial.print("SUCCESS!");
    return true;
  } else {
    // Serial.print("FAILED! Error code: ");
    return false;
  }
}


String parsing_msg(String msg) {
  DynamicJsonBuffer jBuffer;
  JsonObject& jObject = jBuffer.parseObject(msg);
  String Mode = jObject["Mode"];
  String Via = jObject["Via"];
  String msg_payload = jObject["payload"];
  String msg_id = jObject["clientid"];
  String msg_time = jObject["timestamp"];

    Serial.print("Mode is: ");
    Serial.println(Mode);
    Serial.print("Via is: ");
    Serial.println(Via);
```

Main server code

```
    Serial.print("client id: ");
    Serial.println(msg_id);
    Serial.print("payload: ");
    Serial.println(msg_payload);
    Serial.print("incoming time: ");
    Serial.println(msg_time);

  if (Mode == "PKTFWD") {
    if (Via == "WIFI_TCP") {
      TCP_Sender(msg_payload);
    }
    else if (Via == "WIFI_UDP") {
      UDP_Sender(msg_payload);
    }
    else if (Via == "HTTP_POST") {
      POST_Sender ();
    }
    else if (Via == "LORA") {
      LORA_Sender (msg_payload);
    }
    else if (Via == "I2C") {
      I2C_Sender (msg_payload);
    }
    else if (Via == "MQTT") {
      MQTT_Sender ();
    }
    else if (Via == "BLU") {
      BLUETOOTH_Sender ();
    }
    else {
      Serial.println("nothing defined");
    }
    return msg_payload;
  }
}


void callback(char* topic, byte * payload, unsigned int length) {
  String msg = "";
  for (int i = 0; i < length; i++) {
    msg += ((char)payload[i]);
  }
  Serial.println("=================================");
  Serial.print("Message arrived: ");
  Serial.println(topic);
  Serial.print("entire message:");
  Serial.println(msg);
  String msg_payload = parsing_msg(msg);
  Serial.println("=================================");
  String ack = ("recieved: " + msg_payload);
  char ack_c[255];
  ack.toCharArray(ack_c, (ack.length() + 1));
  client.publish(mqtt_recTopic, ack_c);
  delay(100);
  yield();
}

void reconnect() {
  if (!client.connected()) {
    if (WiFi.status() == WL_CONNECTED && testClient()) {
      Serial.print("Attempting MQTT connection...");
      // Attempt to connect
      if (client.connect(client_id)) {
        Serial.println("connected");
        client.subscribe(mqtt_inTopic);
        server_status = true;
        return;
      } else {
        Serial.print("failed, rc=");
        Serial.print(client.state());
        Serial.println(" try again in 5 seconds");
        server_status = false;
      }
      client.disconnect();
    }
  }
}

```

Main server code

```
  if (millis() >= 3000000000) {
    HighMillis = 1;

  }
  //** Making note of actual rollover **//
  if (millis() <= 100000 && HighMillis == 1) {
    Rollover++;
    HighMillis = 0;
  }

  long secsUp = millis() / 1000;

  Second = secsUp % 60;

  Minute = (secsUp / 60) % 60;

  Hour = (secsUp / (60 * 60)) % 24;

  Day = (Rollover * 50) + (secsUp / (60 * 60 * 24)); //First portion takes care of a rollover [around 50 days]

};


void Task2code( void * pvParameters ) {
  Serial.print("Task2 running on core ");
  Serial.println(xPortGetCoreID());

  for (;;) {
    timer.run();
    time_keeper();
  }
}

void setup() {
  Mantra.begin();
  EEPROM.begin(512);
  reading_eeprom();
  WiFi.onEvent(WiFiEvent);
  ETH.begin();
  setup_WiFi();
  setup_config();
  setupSpiffs();
  delay(10);
  Serial.println();
  Serial.println();


  Serial.println("====================================");
  Serial.println("|        M2M and Lora Server       |");
  Serial.println("|        Mantranic DevBoard        |");
  Serial.println("|        Author: Farzad Mohebi     |");
  Serial.println("|        version: 1.1.alfa         |");
  Serial.println("====================================");
  Serial.println("***      MQTTServer starting     ***");
  Serial.print("***        SpreadFactor ");
  Serial.print(Lora_sprdfct);
  Serial.println("        ***");

  client.setServer(mqtt_server, atoi(mqtt_port));
  client.setCallback(callback);

  Setup_LoRa();
  timer.setInterval(30000, reconnect);

  xTaskCreatePinnedToCore(
    Task2code,
    "Task2",
    10000,
    NULL,
    1,
    &Task2,
    1);
  delay(500);


    if (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect(client_id)) {
      Serial.println("connected");
```

Main server code

```

      client.subscribe(mqtt_inTopic);

      server_status = true;
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      server_status = false;
    }

    }




}

void loop() {
  //dnsServer.processNextRequest();
  onReceive(LoRa.parsePacket());
  time_keeper();
  client.loop();
}
```

APPENDIX 2

Bluetooth Service code

```cpp
String Bluetooth_Address() {
  String bluetooth_add = "";
  const uint8_t* point = esp_bt_dev_get_address();

  for (int i = 0; i < 6; i++) {
    char str[3];
    sprintf(str, "%02X", (int)point[i]);
    bluetooth_add += str;
    if (i < 5) {
      bluetooth_add += ":";
    }

  }
  return bluetooth_add;
}
void bluetooth_msgctrl() {
  if (ESP_BT.available()) //Check if we receive anything from Bluetooth
  {
    int incoming;
    incoming = ESP_BT.read(); //Read what we recevive
    Serial.print("Received:"); Serial.println(incoming);
  }
}

```

APPENDIX 3

EEPROM Control Service code

```
 String Bluetooth_Address() {
  String bluetooth_add = "";
  const uint8_t* point = esp_bt_dev_get_address();

  for (int i = 0; i < 6; i++) {
    char str[3];
    sprintf(str, "%02X", (int)point[i]);
    bluetooth_add += str;
    if (i < 5) {
      bluetooth_add += ":";
    }

  }
  return bluetooth_add;
}
void bluetooth_msgctrl() {
  if (ESP_BT.available()) //Check if we receive anything from Bluetooth
  {
    int incoming;
    incoming = ESP_BT.read(); //Read what we recevive
    Serial.print("Received:"); Serial.println(incoming);
  }
}

```

APPENDIX 4

Encryption Service code (simple version provided due to security reasons)

```
String key = "xxxxxxxxxxxxxxxxxxx";
String plain_0 = "Add NodeAdd NodeAdd NodeAdd NodeAdd Node";
String cipher_0, check_0;

/* Now iv has been added */
unsigned long long int my_iv = xxxxxxxx;

/* AES global instance (comment out to recreate problem)*/
AES aes ;

String AES_encrypt(String plain) {
  /* Local AES instance (uncomment to recreate problem) */
  //AES aes ;
  aes.set_IV(my_iv);
  byte * plain_buf = (unsigned char*)plain.c_str();
  byte * key_buf = (unsigned char*)key.c_str();
  // add padding where appropriate
  int cipher_length = (plain.length() + 1 < 16) ? 16 : (plain.length() + 1) + (16 - (plain.length() + 1) / 16);
  byte cipher_buf[cipher_length];
  aes_encrypt(plain_buf, plain.length() + 1, cipher_buf ,256);
  String cipher = aes.printToString(cipher_buf, false);
  return cipher;
}

String AES_decrypt(String cipher) {
  /* Local AES instance (uncomment to recreate problem) */
  //AES aes ;
  aes.set_IV(my_iv);
  byte * cipher_buf = (unsigned char*)cipher.c_str();
  byte * key_buf = (unsigned char*)key.c_str();
  int plain_length = cipher.length();
  byte plain_buf[plain_length];
  aes.do_aes_decrypt(cipher_buf, cipher.length(), 256);
  String plain = aes.printToString(plain_buf, false);
  return plain;
}
```

LoRa Connection Setup code

```
void Setup_LoRa() {
  while (!Serial);
  Serial.println("");
  Serial.print(" ## LoRa Server ##");
  LoRa.setPins(13, -1, -1);

  //433E6 for Asia
  //866E6 for Europe
  //915E6 for North America
  if (!LoRa.begin(433E6)) {
    Serial.println("Starting LoRa failed!");
    while (1);
  }
  LoRa.setSyncWord(atoi(Lora_syncword));           // ranges from 0-0xFF, default 0x34, see API docs
  LoRa.setSpreadingFactor(atoi(Lora_sprdfct));
}

void onReceive(int packetSize) {
  if (packetSize == 0) return;          // if there's no packet, return
  int recipient = LoRa.read();          // recipient address
  byte sender = LoRa.read();            // sender address
  String incoming = "";
  while (LoRa.available()) {
    incoming += (char)LoRa.read();
  }

  DynamicJsonBuffer jBuffer;
  JsonObject& jObject = jBuffer.parseObject(incoming);
  String Mode = jObject["Mode"];
  String payload = jObject["payload"];
  if (client.connected()) {
    String ack = "";
    ack += "{ payload:'";
    ack += payload;
    ack += "',RSSI:'" ;
    ack += String(LoRa.packetRssi());
    ack += "',Snr:' " ;
    ack += String(LoRa.packetSnr());
    ack += "'}" ;
    if (Mode == "ACK") {
      client.publish(mqtt_ackTopic, ack.c_str());
    }
    else if (Mode == "REC") {
      client.publish(mqtt_recTopic, ack.c_str());
    }
    else {
      Serial.println("Mode not defined");
    }

  }
}
```

On The Air Firmware Update code

```
size_t content_len;
void handleDoUpdate(AsyncWebServerRequest *request, const String& filename, size_t index, uint8_t *data, size_t len, bool final) {
  if (!index) {
    Serial.println("Update");
    content_len = request->contentLength();
    // if filename includes spiffs, update the spiffs partition
    int cmd = (filename.indexOf("spiffs") > -1) ? U_SPIFFS : U_FLASH;
    if (!Update.begin(UPDATE_SIZE_UNKNOWN, cmd)) {
      Update.printError(Serial);
    }
  }

  if (Update.write(data, len) != len) {
    Update.printError(Serial);
  }

  if (final) {
    AsyncWebServerResponse *response = request->beginResponse(302, "text/plain", "Please wait while the device reboots");
    response->addHeader("Refresh", "20");
    response->addHeader("Location", "/");
    request->send(response);
    if (!Update.end(true)) {
      Update.printError(Serial);
    } else {
      Serial.println("Update complete");
      Serial.flush();
      ESP.restart();
    }
  }
}
```

APPENDIX7

Packet Forwarder Service code

```
void LORA_Sender (String outgoing) {
  LoRa.beginPacket();                   // start packet
  if (dest_flag) {
    LoRa.write(atoi(Lora_local));              // add destination address
  }
  if (local_flag) {
    LoRa.write(atoi(Lora_dest));             // add sender address
  }

  if (msgcnt_flag) {
    LoRa.write(msgCount);                 // add message ID
  }
  }

  if (len_flag) {

    LoRa.write(outgoing.length());        // add payload length
  }


  LoRa.print(AES_encrypt(outgoing));
  LoRa.endPacket();                     // finish packet and send it
  msgCount++;                           // increment message ID


void UDP_Sender (String msg) {
  DynamicJsonBuffer jBuffer;
  JsonObject& jObject = jBuffer.parseObject(msg);
  const char * udpHost = jObject["host"];
  const int udpPort = jObject["port"];
  const char *  msg_payload = jObject["msg"];
  udp.beginPacket(udpHost, udpPort);
  udp.printf(msg_payload);
  udp.endPacket();

  Serial.println("sent by UDP");
}

void TCP_Sender (String msg) {
  DynamicJsonBuffer jBuffer;
  JsonObject& jObject = jBuffer.parseObject(msg);
  const char * tcpHost = jObject["host"];
  const uint16_t tcpPort = jObject["port"];
  const char *  msg_payload = jObject["msg"];

  if (!espClient.connect(tcpHost, tcpPort)) {
    Serial.println("Connection to host failed");
    delay(10);
    return;
  }
  Serial.println("Connected to server successful!");
  espClient.print(msg_payload);
  espClient.stop();
  Serial.println("sent by TCP");
}

void POST_Sender () {
  Serial.println("sent by POST");
}

void BLUETOOTH_Sender () {
  Serial.println("sent by BLU");
}

void MQTT_Sender() {
  Serial.println("sent by MQTT");
}

void I2C_Sender(String msg) {
  DynamicJsonBuffer jBuffer;
  JsonObject& jObject = jBuffer.parseObject(msg);
  const int device_id = jObject["device_id"];
  const char *  msg_payload = jObject["msg"];
  Wire.beginTransmission(device_id);
  Wire.write(msg_payload);
  Serial.println("sent by I2C");
}
```

SPIFFS Control Service code

```
bool read_Server_config() {
  if (SPIFFS.exists("/Server_config.json")) {
    //file exists, reading and loading
    File configFile = SPIFFS.open("/Server_config.json");
    if (configFile) {
      size_t size = configFile.size();
      std::unique_ptrr<char[]> buf(new char[size12]);

      configFile.readBytes(buf.get(), size);
      DynamicJsonBuffer jsonBuffer;
      JsonObject& json = jsonBuffer.parseObject(buf.get());
      if (json.success()) {
        strcpy(mqtt_server, json["mqtt_server"]);
        strcpy(mqtt_port, json["mqtt_port"]);
        strcpy(mqtt_user, json["mqtt_user"]);
        strcpy(mqtt_pass, json["mqtt_pass"]);
        strcpy(mqtt_inTopic, json["mqtt_inTopic"]);
        strcpy(mqtt_recTopic, mqtt_inTopic) ;
        strcat(mqtt_recTopic, "-REC");
        strcpy(mqtt_ackTopic, mqtt_inTopic) ;
        strcat(mqtt_ackTopic, "-ACK");
        return true;
      } else {
        Serial.println("failed to load  json Server_config");
        return false;
      }
    }
  }
}
bool read_Lora_config() {
  if (SPIFFS.exists("/Lora_config.json")) {
    //file exists, reading and loading
    File configFile = SPIFFS.open("/Lora_config.json");
    if (configFile) {

      size_t size = configFile.size();
      std::unique_ptrr<char[]> buf(new char[size12]);

      configFile.readBytes(buf.get(), size);
      DynamicJsonBuffer jsonBuffer;
      JsonObject& json = jsonBuffer.parseObject(buf.get());
      if (json.success()) {
        strcpy(Lora_sprdfct, json["Lora_sprdfct"]);
        strcpy(Lora_local, json["Lora_local"]);
        strcpy(Lora_dest, json["Lora_dest"]);
        strcpy(Lora_syncword, json["Lora_syncword"]);
        strcpy(Lora_msgcounter, json["Lora_msgcounter"]);
        strcpy(Lora_msglen, json["Lora_msglen"]);
        if (strlen(Lora_sprdfct) == 0) {
          strcpy(Lora_sprdfct, "9");
        }
        if (strlen(Lora_syncword) == 0) {
          strcpy(Lora_syncword, "69");
        }
        if (strlen(Lora_local) == 0) {
          local_flag = false;
        } else {
          local_flag = true;
        }
        if (strlen(Lora_dest) == 0) {
          dest_flag = false;
        } else {
          dest_flag = true;
        }
        if (strcmp(Lora_msgcounter, "yes") == 0) {
          msgcnt_flag = true;
        } else {
          msgcnt_flag = false;
        }
        if (strcmp(Lora_msglen, "yes") == 0) {
          len_flag = true;
        } else {
          len_flag = false;
        }
        return true;
      } else {
        Serial.println("failed to load  json Lora_config");
        return false;
      }
```

SPIFFS Control Service code

```
int HighMillis = 0;    }
  }
}
void setupSpiffs() {
  //clean FS, for testing
  if (SPIFFS.begin()) {
    Serial.println("mounted file system");

  } else {
    Serial.println("failed to mount FS");
  }
  if (!(read_Server_config())) {
    Serial.println("reading server config failed");
  }
  if (!(read_Lora_config())) {
    Serial.println("reading server config failed");
  }
  //read configuration from FS json
  Serial.println("mounting FS...");

}

bool saveServerConfig(String mqtt_server, String mqtt_port, String mqtt_inTopic, String mqtt_user = "", String
mqtt_pass = "") {
  String mqtt_recTopic = mqtt_inTopic + "-REC";
  String mqtt_ackTopic = mqtt_inTopic + "-ACK";
  Serial.println("saving config");
  DynamicJsonBuffer jsonBuffer;
  JsonObject& json = jsonBuffer.createObject();
  json["mqtt_server"] = mqtt_server;
  json["mqtt_port"]   = mqtt_port;
  json["mqtt_user"] = mqtt_user;
  json["mqtt_pass"] = mqtt_pass;
  json["mqtt_inTopic"] = mqtt_inTopic;
  json["mqtt_recTopic"] = mqtt_recTopic;
  json["mqtt_ackTopic"] = mqtt_ackTopic;
  File configFile = SPIFFS.open("/Server_config.json", FILE_WRITE);
  if (!configFile) {
    Serial.println("failed to open config file ");
    return false;
  }

  json.prettyPrintTo(Serial);
  json.printTo(configFile);
  configFile.close();
  return true;
}
bool saveLoraConfig(String Lora_sprdfct, String Lora_syncword, String Lora_local = "", String Lora_dest = "",
String Lora_msgcounter = "", String Lora_msglen = "") {
  Serial.println("saving Lora config");
  DynamicJsonBuffer jsonBuffer;
  JsonObject& json = jsonBuffer.createObject();
  json["Lora_sprdfct"] = Lora_sprdfct;
  json["Lora_syncword"] = Lora_syncword;
  json["Lora_local"]   = Lora_local;
  json["Lora_dest"] = Lora_dest;
  json["Lora_msgcounter"] = Lora_msgcounter;
  json["Lora_msglen"] = Lora_msglen;

  File configFile = SPIFFS.open("/Lora_config.json", FILE_WRITE);
  if (!configFile) {
    Serial.println("failed to open config file ");
    return false;
  }
  json.prettyPrintTo(Serial);
  json.printTo(configFile);
  configFile.close();
  return true;
}
```

Web Server Service code

```
bool is_authentified(AsyncWebServerRequest * request) {
  if (request->hasHeader("Cookie")) {
    if (cookie.indexOf("ESPSESSIONID=1") != -1) {
      return true;
    }
  }
  return false;
}

bool already_loggedin(AsyncWebServerRequest * request) {
  String header;
  if (!is_authentified(request)) {
    AsyncWebServerResponse *response = request->beginResponse(301);
    response->addHeader("Location", "/login");
    response->addHeader("Cache-Control", "no-cache");
    request->send(response);
    return false;
  } else {
    return true;
  }
}

void setup_config() {

  class CaptiveRequestHandler : public AsyncWebHandler {
    public:
      CaptiveRequestHandler() {}
      virtual ~CaptiveRequestHandler() {}

      bool canHandle(AsyncWebServerRequest *request) {
        return true;
      }

      void handleRequest(AsyncWebServerRequest *request) {
        AsyncResponseStream *response = request->beginResponseStream("text/html");
        response->print(temp_page(config_panel, false));
        request->send(response);
      }
  };


  /*use mdns for host name resolution*/
  if (!MDNS.begin(client_id)) { //http://esp32.local
    Serial.println("Error setting up MDNS responder!");
    while (1) {
      delay(1000);
    }
  }
  Serial.println("mDNS responder started");
  server.onNotFound([](AsyncWebServerRequest * request) {
    request->send(404, "text/html", temp_page("Page Not Found : 404"));
  });


  server.on("/", HTTP_GET, [](AsyncWebServerRequest * request) {

    if (already_loggedin(request)) {
      request->send(200, "text/html", temp_page(config_panel, false));
    }
  });

  server.on("/reboot", HTTP_GET, [](AsyncWebServerRequest * request) {
    if (already_loggedin(request)) {
      request->send(200, "text/html", temp_page("<p>Rebooting Mantranic ...</p>"));
      delay(1000);
      ESP.restart();
    }
  });

  server.on("/WiFi", HTTP_GET, [](AsyncWebServerRequest * request) {
    if (already_loggedin(request)) {
      String WiFi_panel = "<script>function c(l){document.getElementById('ssid').value=l.inner-Text||l.textContent;document.getElementById('pass').focus();}</script>";
      WiFi_panel += "<h1>Nearby Wireless Networks</h1>";
      WiFi_panel += search_network();
      WiFi_panel += "<form method='get' action='/WiFi_config'> <input id='ssid' name='ssid' length='32' place-holder='SSID' required='required'> <input id='pass' name='pass' length='32' type='password' placeholder='Pass-word' required='required'> <button type='submit'>save</button> <button class='b' onclick='document.location.re-load(); return false;'>scan</button></form>";
```

Web Server Service code

```
      request->send(200, "text/html",  temp_page(WiFi_panel));
    }
  });
  server.on("/WiFi_config", HTTP_GET, [](AsyncWebServerRequest * request) {
    if (already_loggedin(request)) {
      if (request->hasArg("ssid"))
        qsid = request->arg("ssid");
      if (request->hasArg("ssid"))
        qpass = request->arg("pass");
      int c = 0;
      Serial.println("Waiting for WiFi to connect");
      if ( qsid.length() > 1 ) {
        WiFi.begin(qsid.c_str(), qpass.c_str());
        while ( c < 20 ) {
          delay(500);
          c++;
          if (WiFi.status() == WL_CONNECTED) {
            Serial.println("clearing eeprom");
            for (int i = 0; i < 64; ++i) {
              EEPROM.write(i, 0);
            }
            writing_WiFi_eeprom();
            Serial.println("successfully connect to WiFi");
            content = "{\"Success\":\"saved to eeprom... reset to boot into new WiFi\"}";
            statusCode = 200;
            request->send(statusCode, "text/html", temp_page( content));
            c = 0;
            delay(2000);
            //WiFi.mode(WIFI_STA);

            ESP.restart();
            break;
          }

          else if ((WiFi.status() != WL_CONNECTED && c > 19)) {
            content = "{\"Error\":\"WiFi not found\"}";
            statusCode = 404;
            Serial.println("Sending WiFi 404");
            request->send(statusCode, "text/html", temp_page( content));
            c = 0;
          }
        }
      } else {
        content = "{\"Error\":\"404 not found\"}";
        statusCode = 404;
        Serial.println("Sending 404");
        request->send(statusCode, "text/html", temp_page( content));
      }
    }
  });

  server.on("/firmware", HTTP_GET, [](AsyncWebServerRequest * request) {
    if (already_loggedin(request)) {
      request->send(200, "text/html", temp_page(update_panel));
    }
  });
  server.on("/update", HTTP_POST,
  [](AsyncWebServerRequest * request) {},
  [](AsyncWebServerRequest * request, const String & filename, size_t index, uint8_t *data,
   size_t len, bool final) {
    handleDoUpdate(request, filename, index, data, len, final);
  }
           );
  server.on("/status", HTTP_GET, [](AsyncWebServerRequest * request) {
    if (already_loggedin(request)) {
      String status_panel = "";
      status_panel += "<script>\n";
      status_panel += "setInterval(loadDoc,1000);\n";
      status_panel += "function loadDoc() {\n";
      status_panel += "var xhttp = new XMLHttpRequest();\n";
      status_panel += "xhttp.onreadystatechange = function() {\n";
      status_panel += "if (this.readyState == 4 && this.status == 200) {\n";
      status_panel += "document.getElementsByTagName(\"body\")[0].innerHTML =this.responseText}\n";
      status_panel += "};\n";
      status_panel += "xhttp.open(\"GET\", \"/status\", true);\n";
      status_panel += "xhttp.send();\n";
      status_panel += "return false;}\n";
      status_panel += "</script>\n";
      status_panel += "<script>";
      status_panel += "function send_status(status){\n ";
```

Web Server Service code

```
    status_panel += "xhttp.send();\n";
    status_panel += "\n}";
    status_panel += "</script>\n";
    status_panel += "<h1>Core status</h1> <hr>";
    status_panel += "<p>Datetime : <span style='float:right'>";

    sprintf(datetimestring, "20%02d-%02d-%02d", Clock.getYear(), Clock.getMonth(Century), Clock.getDate());
    status_panel += datetimestring;
    status_panel += "<p>Uptime: <span style='float:right'>";
    sprintf(timestring, "%d days %02d:%02d:%02d", Day, Hour, Minute, Second);
    status_panel += String(timestring);
    status_panel += "</span></p>";
    status_panel += "<p>Used SPIFFS:  <span style='float:right'>";
    status_panel += double(SPIFFS.usedBytes()) / 1024;
    status_panel += "/190KB</span></p>";
    status_panel += "<p>Heap free size:  <span style='float:right'>";
    status_panel += ESP.getFreeHeap();
    status_panel += "</span></p>";
    status_panel += "<p>SD card: ";
    if (Mantra.SDINIT) {
      status_panel += "<span style='color:green;float:right'>connected</span></p>";
      status_panel += "<p>SD Card Size: <span style='float:right'>";
      uint64_t cardSize = SD.cardSize() / (1024 * 1024);
      sprintf(sdstring, "%lluMB\n", cardSize);
      status_panel += String(sdstring);
      status_panel += "</span></p>";
    }
    else {
      status_panel += "<span style='color:red;float:right'>disconnected</span></p>";
    }
    status_panel += "<p> Temprature: <span style='float:right'>";
    status_panel += Clock.getTemperature();
    status_panel += "°C</span></p>";
    status_panel += "<h1>Network</h1> <hr><p>WiFi status: ";
    if (WiFi_status == true) {
      status_panel += "<span style='color:green;float:right'>connected</span></p>";
      status_panel += "<p>Connected to:  <span style='float:right'>";
      status_panel += WiFi.SSID();
      status_panel += "</span></p>";
      status_panel += "<p>RSSI: <span style='float:right'>";
      status_panel += WiFi.RSSI();
      status_panel += "</span></p>";
      status_panel += "<p>IP : <span style='float:right'>";
      status_panel += get_ip_str(WiFi.localIP());
      status_panel += "</span></p>";
      status_panel += "<p>Subnet: <span style='float:right'>";
      status_panel += get_ip_str(WiFi.subnetMask());
      status_panel += "</span></p>";
      status_panel += "<p> Gateway: <span style='float:right'>";
      status_panel += get_ip_str( WiFi.gatewayIP());
      status_panel += "</span></p>";
      status_panel += "<p>Mac Address: <span style='float:right'>";
      status_panel += WiFi.macAddress();
      status_panel += "</span></p>";
    }
    else {
      status_panel += "<span style='color:red;float:right'>disconnected</span></p>";
    }
    status_panel += "<hr><p>Ethernet status: ";
    if (eth_connected) {
      status_panel += "<span style='color:green;float:right'>connected</span></p>";
      status_panel += "<p>IP : <span style='float:right'>";
      status_panel += get_ip_str(ETH.localIP());
      status_panel += "</span></p>";
      status_panel += "<p>Subnet : <span style='float:right'>";
      status_panel += get_ip_str(ETH.subnetMask());
      status_panel += "</span></p>";
      status_panel += "<p>Gateway : <span style='float:right'>";
      status_panel += get_ip_str(ETH.gatewayIP());
      status_panel += "</span></p>";
      status_panel += "<p>MAC Address: <span style='float:right'>";
      status_panel += ETH.macAddress();
      status_panel += "</span></p>";
      status_panel += "<p>Duplex : <span style='float:right'>";
      if (ETH.fullDuplex()) {
        status_panel += "FULL DUPLEX</span></p>";
      } else {
        status_panel += "HALF DUPLEX</span></p>";
      }
      status_panel += "<p>Link Speed : <span style='float:right'>";
```

Web Server Service code

```
        }
        status_panel += "<p>Link Speed : <span style='float:right'>";
        status_panel += ETH.linkSpeed();
        status_panel += "Mbps</span></p>";
      } else {
        status_panel += "<span style='color:red;float:right'>disconnected</span></p>";
      }
      status_panel += "<h1>Bluetooth</h1> <hr><p>Bluetooth status: ";
      if (bluetooth_status == true) {
        status_panel += "<span style='color:green;float:right'>Active</span></p>";
        status_panel += "<p>Broker:  <span style='float:right'>";
        status_panel += "</span></p>";
      } else {
        status_panel += "<span style='color:red;float:right'>Disabled</span></p>";
      }

      status_panel += "<h1>Server</h1> <hr><p>Server status: ";
      if (server_status == true) {
        status_panel += "<span style='color:green;float:right'>connected</span></p>";
        status_panel += "<p>Broker:  <span style='float:right'>";
        status_panel += mqtt_server;
        status_panel += "</span></p>";
        status_panel += "<p>Port:  <span style='float:right'>";
        status_panel += mqtt_port;
        status_panel += "</span></p>";
        status_panel += "<p>Incoming Topic:  <span style='float:right'>";
        status_panel += mqtt_inTopic;
        status_panel += "</span></p>";
        status_panel += "<p>Recieving Topic:  <span style='float:right'>";
        status_panel += mqtt_recTopic;
        status_panel += "</span></p>";
        status_panel += "<p>acknowledge Topic:  <span style='float:right'>";
        status_panel += mqtt_ackTopic;
        status_panel += "</span></p>";
        status_panel += "<p>ClientID :  <span style='float:right'>";
        status_panel += client_id;
        status_panel += "</span></p>";
      } else {
        status_panel += "<span style='color:red;float:right'>disconnected</span></p>";
      }
      request->send(200, "text/html", temp_page(status_panel));
    }
  });

  server.on("/about", HTTP_GET, [](AsyncWebServerRequest * request) {
    if (already_loggedin(request)) {
      request->send(200, "text/html", temp_page(about_panel));
    }
  });

  server.on("/system", HTTP_GET, [](AsyncWebServerRequest * request) {
    if (already_loggedin(request)) {
      request->send(200, "text/html", temp_page(system_panel));
    }
  });

  server.on("/login", HTTP_GET, [](AsyncWebServerRequest * request) {
    String msg;
    if ( request->hasHeader("Cookie")) {
      Serial.print("Found cookie: ");
      String cookie =  request->header("Cookie");
      Serial.println(cookie);
    }
    if ( request->hasArg("DISCONNECT")) {
      Serial.println("Disconnection");
      AsyncWebServerResponse *response = request->beginResponse(301);
      response->addHeader("Location", "/login");
      response->addHeader("Cache-Control", "no-cache");
      response->addHeader("Set-Cookie", "ESPSESSIONID=0");
      request->send(response);
      return;
    }
    if ( request->hasArg("USERNAME") &&  request->hasArg("PASSWORD")) {
      if ( request->arg("USERNAME") == update_username &&   request->arg("PASSWORD") == update_password ) {
        AsyncWebServerResponse *response = request->beginResponse(301);
        response->addHeader("Location", "/");
        response->addHeader("Cache-Control", "no-cache");
        response->addHeader("Set-Cookie", "ESPSESSIONID=1");
        request->send(response);
```

Web Server Service code

```
        }      Serial.println("Log in Failed");
    }
    request->send(200, "text/html", temp_page(login_panel, false));
  });



  server.on("/server_config", HTTP_GET, [](AsyncWebServerRequest * request) {
    if (already_loggedin(request)) {
      if ( ! request->hasArg("server") || ! request->hasArg("port") || ! request->hasArg("topic") || request->arg("server") == NULL || request->arg("port") == NULL || request->arg("topic") == NULL)  {
        request->send(400, "text/html", temp_page("400: Invalid Request"));         // The request is invalid, so send HTTP status 400
        return;
      }
      else {
        qmqtt_server = request->arg("server");
        qmqtt_port = request->arg("port");
        qmqtt_topic = request->arg("topic");
        if (!( ! request->hasArg("username") || ! request->hasArg("password") ||  request->arg("username") == NULL || request->arg("password") == NULL  )) {
          qmqtt_user = request->arg("username");
          qmqtt_pass = request->arg("password");
        }
      }
      if (saveServerConfig(qmqtt_server, qmqtt_port, qmqtt_topic, qmqtt_user, qmqtt_pass)) {
        Serial.println("saved configuration successfully");
        request->send(200, "text/html", temp_page("configuration has been set"));
        delay(1000);
        ESP.restart();
      }
      else {
        Serial.println("saving configuration failed");
      }
    }
  });



  server.on("/server", HTTP_GET, [](AsyncWebServerRequest * request) {
    if (already_loggedin(request)) {
      String server_panel = "<script>function c(l){document.getElementById('server').value=l.innerText||l.textContent; document.getElementById('port').value='1883' ;document.getElementById('topic').focus();}</script>";
      server_panel += "<h1>MQTT Server selection</h1>";
      server_panel += "<span> HiveMQ - <a href='#p' onclick='c(this)'>broker.mqttdashboard.com</a></span><br>";
      server_panel += "<span>Eclipse - <a href='#p' onclick='c(this)'>iot.eclipse.org</a></span><br>";
      server_panel += "<span> Mosquitto - <a href='#p' onclick='c(this)'>test.mosquitto.org</a></span><br>";
      server_panel += "<form method='get' action='/server_config'> <input id='server' name='server' length='32' placeholder='MQTT Server' required='required'> <input id='port' name='port' length='4' placeholder='port' required='required'> <input id='username' name='username' length='32' placeholder='username'> <input id='password' name='password' length='32' placeholder='password'> <input id='topic' name='topic' length='32' placeholder='Topic' required='required'><button type='submit'>save</button> <input type='reset' value='clean' class ></input></form>";
      request->send(200, "text/html",  temp_page(server_panel));
    }
  });

  server.on("/lora_config", HTTP_GET, [](AsyncWebServerRequest * request) {
    if (already_loggedin(request)) {
      String qlora_sprdfct , qlora_syncword , qlora_localadd , qlora_destadd, qlora_addlength, qlora_addid;
      if ( ! request->hasArg("sprdfct") || ! request->hasArg("syncword") || request->arg("sprdfct") == NULL || request->arg("syncword") == NULL )  {
        request->send(400, "text/html", temp_page( "400: Invalid Request"));
        return;
      }
      else {
        qlora_sprdfct = request->arg("sprdfct");
        qlora_syncword = request->arg("syncword");
      }
      if ( ! request->hasArg("localadd") || request->arg("localadd") == NULL) {
        qlora_localadd = "";
      } else {
        qlora_localadd = request->arg("localadd");
      }
      if (! request->hasArg("destadd") || request->arg("destadd") == NULL ) {
        qlora_destadd = "";
      } else {
        qlora_destadd = request->arg("destadd");
      }
```

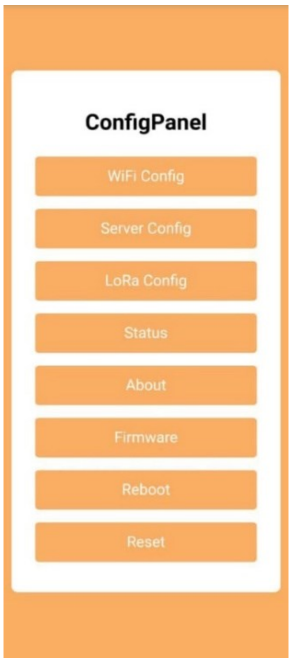
Web Server Service code

```
              } else {
        qlora_addlength = request->arg("addlength");
      }
      if (! request->hasArg("addid")   || request->arg("addid") == NULL ) {
        qlora_addid = "";
      } else {
        qlora_addid = request->arg("addid");
      }
      saveLoraConfig(qlora_sprdfct, qlora_syncword, qlora_localadd, qlora_destadd, qlora_addid, qlora_addlength);
      request->send(200, "text/html", temp_page( "configuration has been set"));
      delay(1000);
      ESP.restart();
    }
  });

  server.on("/lora", HTTP_GET, [](AsyncWebServerRequest * request) {
    if (already_loggedin(request)) {
      String lora_panel = "<script>function c(l){document.getElementById('sprdfct').value = ";
      lora_panel += Lora_sprdfct;
      lora_panel += ";document.getElementById('syncword').value = ";
      lora_panel += Lora_syncword;
      lora_panel += ";document.getElementById('localadd').value = ";
      lora_panel += Lora_local;
      lora_panel += ";document.getElementById('destadd').value =";
      lora_panel += Lora_dest;
      lora_panel += ";document.getElementById('addid').value ='";
      lora_panel += Lora_msgcounter;
      lora_panel += "';document.getElementById('addlength').value ='";
      lora_panel += Lora_msglen;
      lora_panel += "';}</script><h1>Lora Configuration</h1> <form method='get' action='/lora_config'> <input id='sprdfct' name='sprdfct' length='5' placeholder='spread factor from 6-12' required='required'> <input id='syncword' name='syncword' length='5' placeholder='sync word from 0-255' required='required'> <input id='localadd' name='localadd' length='5' placeholder='add local address from 0-255'> <input id='destadd' name='destadd' length='5' placeholder='add destination address from 0-255' ><select id='addid' name='addid'><option value='no'>Without message ID</option><option value='yes'>Add message ID</option></select> <select id='addlength' name='addlength'><option value='no'>Without message length</option><option value='yes'>Add message length</option></select> <input value='auto fill' type='button' onclick= 'c(1)'></input> <button type='submit'>save</button> <input type='reset' value='clean' ></input></form>";
      request->send(200, "text/html",  temp_page(lora_panel));
    }
  });

  server.on("/reset", HTTP_GET, [](AsyncWebServerRequest * request) {
    if (already_loggedin(request)) {
      clear_eeprom();
      SPIFFS.format();
      request->send(200, "text/html",  temp_page("<p>reseting all configurations...</p><p>wait for the Mantranic to restart</p>"));
      delay(1000);
      ESP.restart();
    }
  });


  server.on("/plain", HTTP_POST, [](AsyncWebServerRequest * request) {
    Serial.println(request->arg("plain"));
    parsing_msg(request->arg("plain"));
    request->send ( 200, "text/plain", "{success:true}" );
  });

  server.on(
    "/post",
    HTTP_POST,
  [](AsyncWebServerRequest * request) {},
  NULL,
  [](AsyncWebServerRequest * request, uint8_t *data, size_t len, size_t index, size_t total) {
    String body = "";
    body =  (char*)data;
    Serial.println(body);
    parsing_msg(body);
    request->send ( 200, "text/plain", "{success:true}" );
  });
  server.begin();
}

```

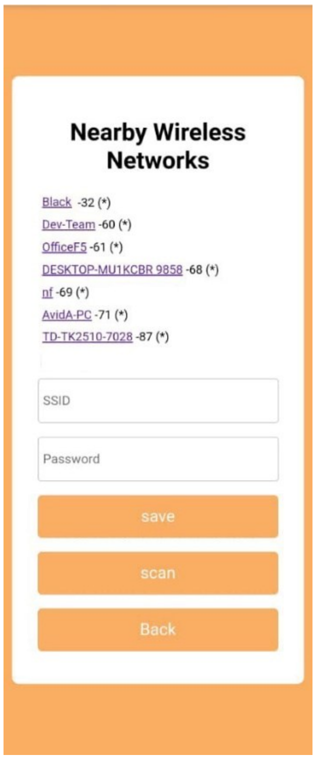
Web Service UI code

```
String temp_page(String body, bool back = true) {
  String html = "<!DOCTYPE HTML> <head> <meta http-equiv='content-type' content='text/html'; charset='UTF-8'>
<meta name='viewport' content='width=device-width, initial-scale=1, user-scalable=no'> <style> div, input {
padding: 5px 5px 5px 5px; background: #ffffff; } input,select { width: 96%; height: 40px; margin: 8px 0;
display: inline-block; border: 1px solid #ccc; border-radius: 4px; font-size: 15px } select { width: 100%;}
button:hover, input[type=submit]:hover { background-color: #d17824; } h1 { text-align: center; } body { text-
align: center; font-family: verdana; background: #f78d23b5; font-size: 14px; } input[type=submit],in-
put[type=button],input[type=reset], button { height: 50px; border: 0; border-radius: 0.3rem; background-color:
#f78d23b5; color: #ffffff; line-height: 2.4rem; font-size: 1.2rem; width: 100%; cursor: pointer; margin: 8px
0; } .panel { background: #fff; max-height: auto; width: 280px; margin: 75px auto; padding: 30px; border-
radius: 8px; text-align:left; display:inline-block; } </style> </head> <body> <div class='panel'>";
  html += body;
  if (back == true) {
    html += "<form action='/' method='get'><button>Back</button></form>";
  }
  html += "</div> </body> </html>";
  return html;
}
String config_panel = "<h1>ConfigPanel</h1> <form action='/WiFi' method='get'><button>WiFi Config</but-
ton></form> <form action='/server' method='get' ><button>Server Config</button></form> <form action='/lora'
method='get' ><button>LoRa Config</button></form><form action='/status' method='get' ><button>Status</but-
ton></form><form action='/system' method='get' ><button>System</button></form> <form action='/about'
method='get' ><button>About</button></form>";

String login_panel = " <form name=loginForm method='get' action='/login'> <h1>Mantranic Settings</h1> <input
name=USERNAME placeholder='Username' > <input name=PASSWORD placeholder=Password type=Password> <input
type=submit onclick=check(this.form) value=Login> </form>";

String update_panel = "<script src='https://ajax.goog-
leapis.com/ajax/libs/jquery/3.2.1/jquery.min.js'></script><form method='POST' action='#' enctype='multi-
part/form-data' id='upload_form'><input type='file' name='update' id='file' onchange='sub(this)' style=dis-
play:none><label id='file-input' for='file'> Choose file...</label><input type='submit' class=btn value='Up-
date'><br><br><div id='prg'></div><br><div id='prgbar'><div id='bar'></div></div><br></form><script>function
sub(obj){var fileName = obj.value.split('\\\\');document.getElementById('file-input').innerHTML = ' '+ file-
Name[fileName.length-1];};$('form').submit(function(e){e.preventDefault();var form = $('#upload_form')[0];var
data = new FormData(form);$.ajax({url: '/update',type: 'POST',data: data,contentType: false,pro-
cessData:false,xhr: function() {var xhr = new window.XMLHttpRequest();xhr.upload.addEventListener('progress',
function(evt) {if (evt.lengthComputable) {var per = evt.loaded / evt.total;$('#prg').html('progress: ' +
Math.round(per*100) + '%');$('#bar').css('width',Math.round(per*100) + '%');}}, false);return xhr;},suc-
cess:function(d, s) {console.log('success!') },error: function (a, b, c) {}});});</script>";

String lora_panel = "<script>function c(l){document.getElementById('sprdfct').value = "+(int)Lora_sprd-
fct+";document.getElementById('syncword').value = 69;document.getElementById('localadd').value = 13;docu-
ment.getElementById('destadd').value = 33;}</script><h1>Lora Configuration</h1> <form method='get' ac-
tion='/lora_config'> <input id='sprdfct' name='sprdfct' length='5' placeholder='spread factor from 6-12' re-
quired='required'> <input id='syncword' name='syncword' length='5' placeholder='sync word from 0-255' re-
quired='required'> <input id='localadd' name='localadd' length='5' placeholder='add local address from 0-255'>
<input id='destadd' name='destadd' length='5' placeholder='add destination address from 0-255' ><select
id='addid' name='addid'><option value='no'>Without message ID</option><option value='yes'>Add message ID</op-
tion></select> <select id='addlength' name='addlength'><option value='no'>Without message length</option><op-
tion value='yes'>Add message length</option></select> <input value='auto fill' type='button' onclick=
'c(1)'></input> <button type='submit'>save</button> <input type='reset' value='clean' ></input></form>";

String system_panel = "<script>function myconfirmation(mode) {if (mode == 1){var ans = confirm('Are you sure
about rebooting ?');if (ans){console.log('going to reboot');window.location.href = '/reboot';return
false;}}else if (mode == 0){var ans = confirm('Are you sure about reseting all configurations ?');if (ans ==
true){window.location.href = '/reset';return false;}}}</script><h1>System </h1> <form action='/firmware'
method='get'><button>Firmware</button></form><button onclick='return myconfirmation(1)'>Reboot</button><button
onclick='return myconfirmation(0)'>Reset</button>";

String about_panel = "<h1>WE ARE FUCKING BEST!!!</h1><p>1.0.a</p>";
```

Config Panel

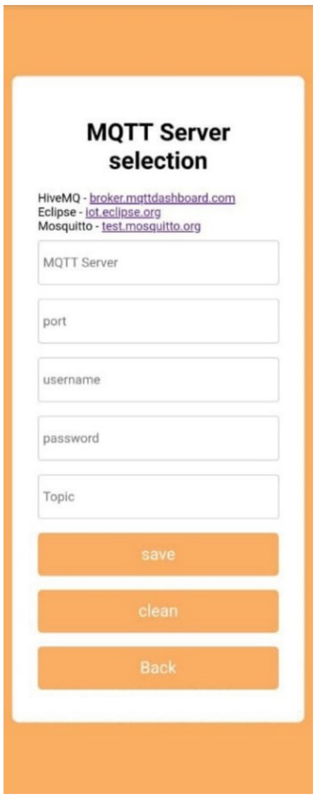


WiFi Config

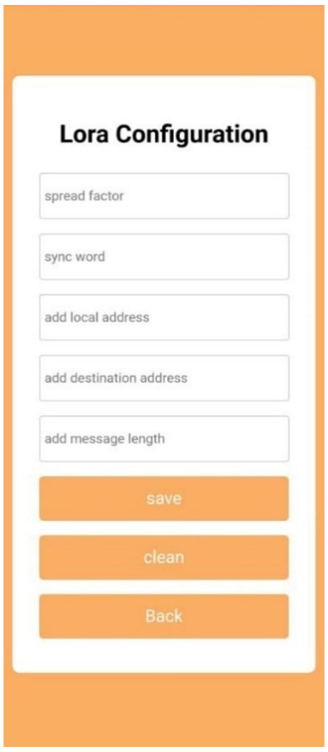


Black -32 (*)
Dev-Team -60 (*)
OfficeF5 -61 (*)
DESKTOP-MU1KCBR 9858 -68 (*)
nf -69 (*)
AvidA-PC -71 (*)
TD-TK2510-7028 -87 (*)

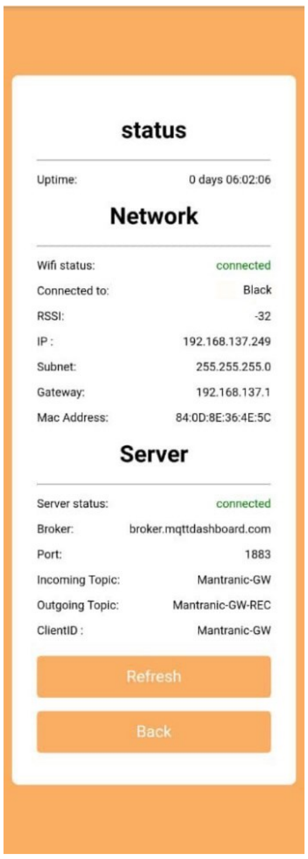
Server Config


MQTT Server selection
HiveMQ - broker.mqttdashboard.com
Eclipse - iot.eclipse.org
Mosquitto - test.mosquitto.org
MQTT Server
port
username
password
Topic
save
clean
Back

LoRa Config



Status

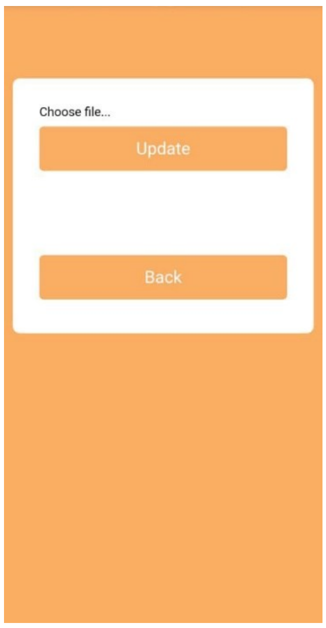


Uptime:	0 days 06:02:06

Wifi status:	connected
Connected to:	Black
RSSI:	-32
IP :	192.168.137.249
Subnet:	255.255.255.0
Gateway:	192.168.137.1
Mac Address:	84:0D:8E:36:4E:5C

Server status:	connected
Broker:	broker.mqttdashboard.com
Port:	1883
Incoming Topic:	Mantranic-GW
Outgoing Topic:	Mantranic-GW-REC
ClientID :	Mantranic-GW

Firmware Update


Choose file...
Update
Back