SAPIENZA
UNIVERSITÀ DI ROMA

Home Net System: Design and implementation of a home automation system for energy consumption control

Facoltà di Ingegneria dell'informazione, Informatica e Statistica

Corso di Laurea in Ingegneria Informatica e Automatica

Candidate

Gioele Migno

ID number 1795826

Thesis Advisor

Prof. Alessandro Di Giorgio

Co-Advisor

Prof. Vincenzo Suraci

Academic Year 2019/2020

Thesis not yet defended

Home Net System: Design and implementation of a home automation system for energy consumption control

Bachelor's thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Version: October 5, 2020

Author's email: gioele.migno@gmail.com

Abstract

Il progetto svolto si colloca all'interno dello sviluppo di sistemi di automazione per il controllo del consumo energetico in ambito domestico e per piccole imprese, al fine di ridurre i costi di approvvigionamento energetico limitando i tempi di sovraccarico, effettuando il distacco mirato dei carichi a bassa priorità. Tale tecnica è detta load management e richiede l'installazione di un impianto domotico dotato di prese intelligenti (smart socket), ovvero in grado di condividere l'attuale consumo energetico e consentire il distacco del carico tramite comando remoto. A tal fine mi sono occupato della progettazione e realizzazione di un sistema domotico (Home Net System) pensato per essere installato in impianti elettrici preesistenti, sviluppando dei dispositivi installabili direttamente all'interno delle prese a muro. Tali dispositivi si basano sul microcontrollore Atmega328P e comunicano all'interno di ciascuna stanza mediante un bus RS485. All'interno di ogni stanza è presente un supervisore basato su microprocessore ARM che consente la gestione dei dispositivi da parte di un controllore locale mediante rete WiFi, in particolare tramite il protocollo MQTT. Durante i test effettuati mediante la realizzazione di un prototipo, si sono ottenuti tempi di risposta nell'ordine delle centinaia di millisecondi, tramite controllo locale, e tempi poco al di sotto di un secondo con controllo remoto gestito da un server. Le tempistiche ottenute consentono la gestione efficace del carico in modo da ridurre la durata dei sovraccarichi ed evitare che il contatore energetico effettui il completo distacco dell'impianto.

Il seguente testo è scritto in lingua inglese in modo da essere facilmente accessibile.

The project developed is part of the development of automation systems for energy consumption control in home and small business environments, to cut energy supply costs by reducing the duration of overloads, carrying out a targeted disconnection of low priority loads. This technique is called load management and requires the installation of a home automation system equipped with smart sockets, that can share real-time energy consumption and allow the detachment of the load via remote control. In order to do that, I dealt with the design and implementation of a home automation system (Home Net System) designed to be installed in an existing electrical system, developing devices that can be mounted directly inside wall outlets. These devices are based on the Atmega328P microcontroller and communicate within a room via a RS485 bus. Inside each room there is a supervisor based on an ARM microprocessor that allows the management of the devices by a local controller through WiFi network, in particular by MQTT protocol. During the tests carried out through the realization of a prototype, the response times obtained are in order of hundreds of milliseconds, via local control, and times of about a second with remote control managed by a server. The timing obtained allows effective load management so as to reduce the duration of overloads and prevent the disconnection of the overall plant by energy meter.

The following text is written in English to be easily accessible.

Contents

Introduction **vii**

1 Commercial Solutions **1**
1.1 IoT Protocols . . . 1
1.1.1 Wireless Protocol . . . 1
1.1.1.1 MQTT . . . 1
1.1.1.2 REST . . . 1
1.1.1.3 Zigbee . . . 2
1.1.2 Wired Protocol . . . 2
1.1.2.1 Industrial Ethernet . . . 2
1.1.2.2 Modbus . . . 2
1.1.2.3 CAN-bus . . . 3
1.2 Smart Socket . . . 3
1.3 Home Automation System . . . 3

2 Home Net System **5**
2.1 Network Structure . . . 5
2.2 Network Types . . . 6

3 Implementation **7**
3.1 Introduction . . . 7
3.1.1 TTL - RS485 . . . 7
3.1.2 Arduino . . . 8
3.1.3 Raspberry Pi . . . 9
3.2 Build Devices . . . 9
3.2.1 HNSBroker . . . 9
3.2.2 HNSHub . . . 9
3.2.3 HNSDevice . . . 10
3.2.3.1 Circuit Schematic . . . 10
3.2.3.2 Mounting Support Structure . . . 12
3.2.3.3 Mounting Example . . . 13
3.2.4 HNSSocket . . . 14
3.2.4.1 Open Energy Monitor . . . 14
3.2.4.2 Power Measurement Circuit . . . 14
3.2.4.3 Relay Case . . . 17
3.2.4.4 External Module . . . 17

3.2.4.5 Overall Scheme 18
3.3 Communication Protocols 18
3.3.1 HNSProtocol 19
3.3.1.1 Address Assignment 19
3.3.1.2 HNSSocket Application Protocol 20
3.3.2 MQTT HNSSocket 24
3.3.2.1 Announce Message 25
3.3.2.2 Switch Subdevice 25
3.3.2.3 Power Subdevice 26
3.4 Software 27
3.4.1 HNSDevice 27
3.4.1.1 Developer Interface 27
3.4.1.2 Ready Packet Interrupt (RPI) 28
3.4.1.3 HNSSocket 29
3.4.2 HNSHub 33
3.4.3 HNSBroker 34

4 Proof of Concept 37
4.1 Components 37
4.1.1 Testbed 37
4.1.2 Web User Interface 40

5 Results 41
5.1 Local Test 41
5.1.1 Refresh Time 41
5.1.2 Command Time 42
5.2 Remote Control Test 43
5.2.1 FWA Network 44
5.2.2 4G Network 45
5.2.3 Comparison 4G-FWA 46

6 Conclusions 47

Introduction

The world of Internet Of Things (IoT) is in continuous evolution thanks to the development of smaller and more powerful devices, their applications are many and find wide space within the domestic environment in which they can manage the power consumption and increase the comfort of users. One of the main problems with the energy supplies is the aleatory of a single customer demand, currently, when the electricity meter detects a power consumption above the limit, it performs the detachment of the entire electrical plant hence causing an inconvenience to users. Moreover, the excess of power consumption has a high cost for the energy supplier especially if the customer is an industry. IoT can resolve this problem by performing a targeted detachment of the load that causes the exceeding the limit, this method is called load shedding. To allow a capillary control of the single loads connected to the plant, each socket must be able to control the load and measure the power consumption, it must also allow the access of that data to an external controller, this kind of socket is called smart socket. Load shedding can be handled by a remote server managed directly by the energy provider, therefore the communication network used plays a significant impact on the reaction time that must be as small as possible. The new generation of mobile network 5G provides a great opportunity to develop this kind of system in areas without a high speed wired internet connection.

The following project aims to design and develop a home automation system installable within a pre-existing electrical plant, in order to allow the energy monitoring and control by a remote server. The overall system (Figure 0.1) has been developed together with a working group and consists of three components:

- *Local Wired Network:* Wired network of smart sockets
- *Local Controller:* This controller collects data from the local wired network and sends it to the remote controller.
- *Remote Controller:* The server that manages the load shedding

The remote controller and the local controller have been developed respectively by Giuseppe Spina and Alessandro Lamin. I instead deal with the development of the smart sockets and the local wired network.

The overall project is part of the European 5G Solutions challenge which aims to demonstrate the possible applications in which the 5G mobile network can improve the performance compared to the 4G network [1]. The possible fields of use are divided into three categories: *Factory of the Future*, *Smart City*, and *Smart Energy*. Our project deals with this last, in particular with *Use Case 2.1: Industrial*

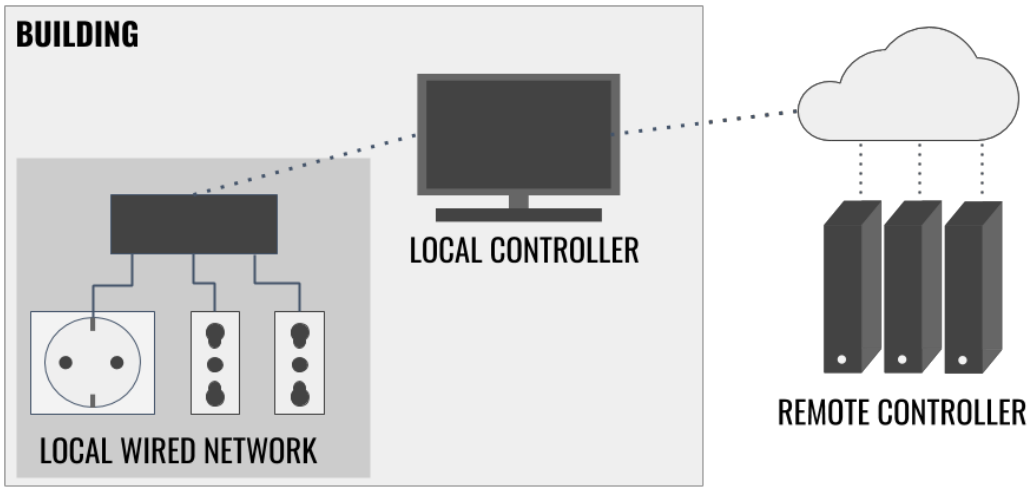


Figure 0.1. The overall system for remote control to act demand side management

Demand Side Management [2]. "The term Demand Side Management (DSM) is used to refer to a group of actions designed to efficiently manage a site's energy consumption with the aim of cutting the costs incurred for the supply of electrical energy, from grid charges and general system charges, including taxes".[1]

This document is organized in several chapters. In Chapter 1: *Commercial Solutions* are exposed the current devices on the market that can be used for energy consumption control and management. In Chapter 2: *Home Net System* is described the home automation system developed from a conceptual point of view and Chapter 3: *Implementation* shows the implementation choices and how to build the devices of the system. In Home Net System introduction, the local controller is called HNSBroker, this device is a simple MQTT Broker that provides a user interface to show the system operation and manage the loads connected to the proof of concept described in Chapter 4: *Proof of Concept*. The tests results of local control by HNSBroker and control by the remote controller are reported in Chapter 5: *Results*. Finally in Chapter 6: *Conclusions* are described the limitations of the system and how to improve it.

[1]Source: https://www.enelx.com/en/questions-and-answers/eindustry/what-is-demand-side-management

Chapter 1

Commercial Solutions

In this chapter are exposed several commercial solutions by analyzing the communication protocol used for IoT at first, then the smart sockets that can be purchased and the domotic systems available on the market.

1.1 IoT Protocols

The analysis of current IoT protocols on the market is divided into two parts: protocols based on wireless channels and those based on wired connections.

1.1.1 Wireless Protocol

1.1.1.1 MQTT

MQTT (Message Queue Telemetry Transport) usually runs over TCP/IP stack as an application protocol by using TCP as transport layer and in most cases is used with a wireless physical layer (WiFi). An MQTT message consists of two fields: *topic* and *payload*. The topic of a message indicates the object or the quantity to which the payload refers. For example, a temperature sensor can publish a message like {**topic**: home/bedroom/temperature, **payload**: 22.0}, the slash in the topic indicates the hierarchy of the topics [3].

In an MQTT network there are two types of devices:

- *MQTT Broker:* In the network, there can be only one MQTT Broker, it deals with forwarding the messages from an MQTT Client to other clients subscribed to the topic of the message.
- *MQTT Client:* A client can publish messages and receive the messages with the topics to which it is subscribed.

1.1.1.2 REST

REST (REpresentational State Transfer) is an architecture style for distributed hypermedia systems, a web service that applies this style is called RESTful. REST architecture is client-server based and stateless (i.e. each request from client to server

must contain all of the information necessary to understand the request). Usually, a RESTful web service is based on HTTP application protocol over TCP/IP stack, and information is exchanged using a payload in JSON, XML, or HTML format [4].

1.1.1.3 Zigbee

Zigbee is a wireless ad hoc network based on 2.4GHz band (Industrial, Scientific and Medical ISM band), it has been designed to be simpler and less expensive than other wireless networks such as WiFi. The most important characteristic of this system is the cost-effectiveness of communication chips and the low energy consumption, the max data rate is 250 Kbits/sec [5].

1.1.2 Wired Protocol

Currently in the market there are no wired protocols as widespread as wireless ones and designed specifically for IoT, most of them have been developed for PLCs communication in industrial scenarios.

1.1.2.1 Industrial Ethernet

Industrial Ethernet term refers to the use of Ethernet (link layer in TCP/IP stack) in an industrial setting which requires better determinism, in order to achieve this last, industrial Ethernet uses specialized protocols over Ethernet, the more popular ones are PROFINET, EtherNet/IP, EtherCAT, SERCOS III, POWERLINK, and Modbus-TCP. Data transmission rates range from 10 Mbps to 1 Gbps, however, 100 Mbps is the most popular speed used in industrial Ethernet applications [6].

1.1.2.2 Modbus

Modbus is a protocol based on master-slave model and designed by Modicon in 1979 for use with its PLCs, it is often used with RS485 physical layer or with Ethernet link layer (Modbus-TCP). Modbus frame consists of an Application Data Unit (ADU), which encapsulates a Protocol Data Unit (PDU):

ADU = [Address | **PDU** | Error check]

PDU = [Function code | Data]

Master requests information using *Function code* that indicates an action that needs to be performed such as read or write of a slave's register.

Modbus is open-source and development costs are low because it requires minimal hardware design, moreover is one of the most popular protocols used in industrial environments. However, since the protocol was designed in the late 1970s, the number of data types is limited to those understood by PLCs at the time, so large binary objects are not supported. Moreover, since it is strongly focused on PLCs logic, it is not appropriate to other contexts such as IoT where data exchanged are extremely heterogeneous [7].

1.1.2.3 CAN-bus

CAN (Controller Area Network) is a robust serial communication bus found mostly in automotive and industrial environments. It is a multi-master protocol and a CSMA/CD protocol, meaning each node on the bus can detect collisions and back off for a certain amount of time before trying to retransmit. In order to be robust to noise, the physical layer uses a differential signal such as RS485, since a device must detect a collision the channel used is full-duplex (i.e. a device can read the bus while it is transmitting on it). Currently, CAN-bus is a standard for device communication in automobiles [8].

1.2 Smart Socket

Commercial solutions that allow to manage a load and measure its power consumption are divided into two categories: external smart socket and smart relay. The first is a device designed to be plug-and-play, it is an external adapter to be connected to an electrical outlet. The second requires more competence to be installed since it is usually mounted inside an electrical panel or directly inside the wall socket.

Most external smart sockets use WiFi to control the devices directly by smartphone using a proprietary communication protocol, however, some brands also allow managing devices using a RESTful web service (e.g. Sonoff S20 Wi-Fi Smart Plug). Other brands use a Zigbee network for their home automatic ecosystem, for example, Ikea. The smart relay is most widespread in the hobby sector, Shelly produces a wide range of mini smart relays with several functionalities and performance, their devices use MQTT protocol and also provide a local web service to configure and manage them.

(a) **(b)**

Figure 1.1. An external smart socket by Sonoff (a) and a Shelly 1 smart relay (b)

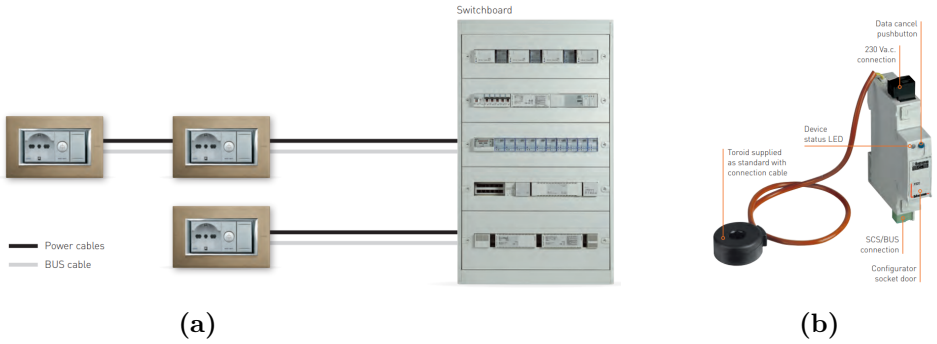
1.3 Home Automation System

Home automation systems currently on the market are divided into two categories according to the paradigm followed. The first typology applies industrial automation methods in a domestic environment, by using one or more PLCs that perform a centralized control by managing the relays installed in a single electrical panel. Since

a system of this kind requires a lot of work for design and installation, it is usually installed during the construction or renovation of a building.

The second type is designed to be easy to install and scalable, it usually uses plug-and-play WiFi devices by several brands, such as those shown in Figure 1.1, that provide a web interface or an app to control only their own devices. Alternatively, there are several services that allow controlling all devices, also by several brands, to build an integrated system (e.g. Home Assistant and OpenHAB).

An example of an industrial oriented home automation system is MyHOME Up[1] by BTicino, the devices communicate through a bus and they can be controlled remotely using a gateway called MyHOMEserver1. A smart socket is composed of two modules, one is a switch with a LED indicator for local control and the other is a relay with power consumption meter. The first is installed with the power socket, the second on a DIN-rail then in an electrical panel, however, BTicino also provides a device with only a relay, without a power meter, that can be installed directly in the power socket. MyHOME Up can also perform load shedding by disconnecting the least important appliances in case of overload, it is possible to manage up to 63 priority levels. In Figure 1.2a the power sockets wiring is shown, and the device in Figure 1.2b is the power measurement and relay module that is installed inside the electrical panel.



(a) **(b)**

Figure 1.2. Power sockets wiring (a). Power measurement and relay module (b)

Samsung has instead developed a home automation system by managing both WiFi and Zigbee wireless devices. The system is called SmartThings[2] and it is compatible with several brands and types of devices, from power sockets and lights to TVs and robot cleaners.

The current commercial solutions hence include a type of system fully wired that is difficult to scale, because the number of devices and their configurations are decided at the time of plant design, and a type based completely on plug-and-play wireless devices, that is more scalable but usually has low performance in response time. Moreover, there is not a standard protocol for wired master-slaves communication that is not focused on PLCs logic.

[1] https://www.download.bticino.it/bticino/assetfile/72488.pdf
[2] https://www.samsung.com/it/apps/smartthings/

Chapter 2

Home Net System

In this chapter, Home Net System is introduced by a conceptual point of view.

Home Net System is a home automation system designed for updating existing domestic electrical plants; open-source, efficiency and cost-effectiveness are the key features.

2.1 Network Structure

The network is based on a pyramidal structure like CIM (Computer Integrated Manufacturing) model, on the field level, represented by rooms, there are devices called HNSDevice with the task of collecting measures and execute commands, these devices are low power hence with lack computing capacity, such as a microcontroller. In each room or area of the house is present a controller called HNSHub with the aim of managing all devices and make them accessible outside through LAN. Thus room controller has two communication channels one to HNSDevice and the other to LAN as shown in the following figure: (Figure 2.1)

Figure 2.1. HNSHub has two channels, one for serial communication with HNSDevices and the other to connect to the local network.(LAN)

HNSHub requires more computation power so it needs a microprocessor (e.g ARM). Within the LAN there is a house monitor, called HNSBroker, connected to all HNSHubs installed. HNSBroker provides a user interface for monitoring of the whole house.

The hierarchical structure of the network is summarized in Figure 2.2 In the bottom block (ROOMs) there are rooms of the house, in each of them are present sensors and actuators, such as smart socket, light source, and temperature sensor, in the figure the rooms are represented by different greyscale. On top of the pyramid, there is supervision and control block of the whole house.

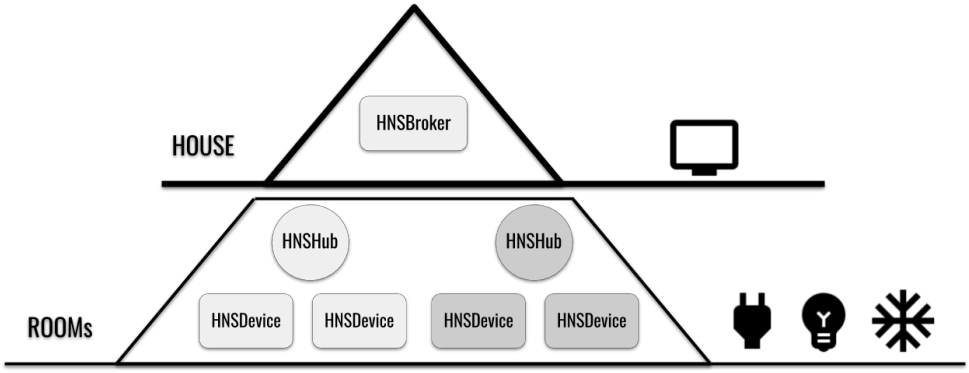


Figure 2.2. Hierarchical structure of the system..

2.2 Network Types

Inside the system, there are two types of networks: a master-slave architecture, used by HNSDevices and HNSHubs, and a connection over LAN used by HNSHubs to communicate with HNSBroker. In a master-slave architecture, there is only one device, called master, that controls the communication, the others are called slaves because they can only communicate following a request from the master. This model allows us to avoid transmission collision and it simplifies communication protocol. In each room, there is an independent network in which HNSHub communicates with HNSDevices, the latter are slaves and HNSHub is the master of the area. Connection over LAN is based instead on the TCP/IP model using an application protocol designed for Internet of Things (IoT) such as MQTT.

In Figure 2.3 an example of connection is shown. There are two areas, in each of them there is a separate master-slaves network, their controller (HNSHub) is connected to HNSBroker through LAN.

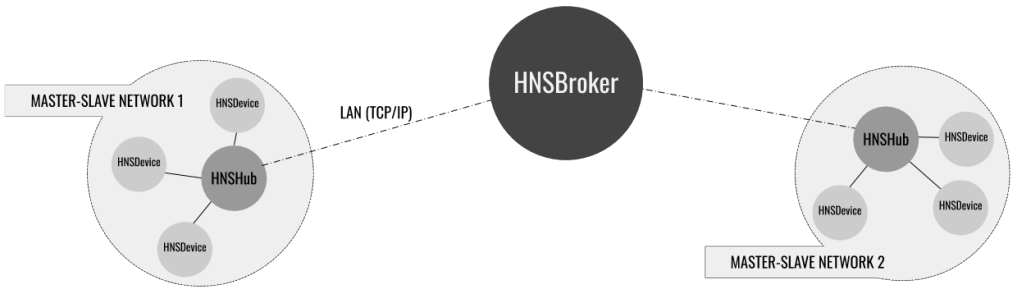


Figure 2.3. Hierarchical structure of the system..

Chapter 3

Implementation

In this chapter are described the technical choices for the realization of Home Net system and how to build the devices.

3.1 Introduction

Home Net System structure can be implemented using various types of master-slave networks, it can be wired or not, however, to obtain a higher speed and reliability it is necessary to use a wired network such as RS485. Furthermore using a cable network in a small area (e.g room) it is possible to power HNSDevices directly via the network, thus reducing the size of devices since they do not require a power supply module to convert mains voltage (e.g 240 VAC).

The different types of devices in the system (HNSBroker, HNSHub, and HNSDevice) can be manufactured in several ways. Any Linux device can be used for HNSBroker and HNSHub, but HNSBroker requires a higher computing capacity. HNSDevice can be based on any type of microcontroller equipped with a UART, however, in order to reduce energy consumption and cost, it should be used a microcontroller with low clock frequency such as ATMEGA328P 16MHz.

As previously mentioned the protocol for communication between HNSBroker and HNSHub is based on TCP/IP model, here too can be used any kind of application protocol, however, it is important to use an IoT standard protocol to make it compatible with other commercial products.

As a result of the above considerations, the following implementation used Arduino Pro mini as HNSDevice and RaspberryPi as HNSBroker and HNSHub. For communication, master-slave network and application protocol are based respectively on RS485 standard and MQTT protocol. The components are described in the following.

3.1.1 TTL - RS485

Most microcontrollers have built-in UARTs (Universally Asynchronous Receiver - Transmitter) that can be used to receive and transmit data serially. UARTs transmit one bit at a time at a specified data rate (i.e. 9600bps, 115200bps, etc.). This method

of serial communication is sometimes referred to as TTL serial (Transistor-Transistor Logic). Serial communication at a TTL level will always remain between the limits of 0V and Vcc, which is often 5V or 3.3V. A logic high ('1') is represented by Vcc, while a logic low ('0') is 0V.[1]

RS485 is a serial digital data transmission standard mainly used in the industrial environment due to its reliability and ability to reject noises. These characteristics are guaranteed by differential signal line, standard uses two wires (A B), the RS485 receiver compares the voltage difference between both lines, instead of the absolute voltage level on a signal line, as described in Figure 3.1

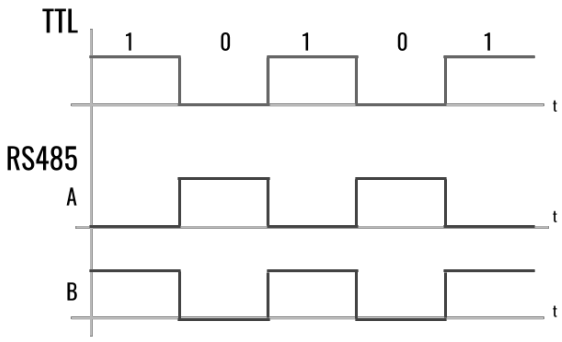


Figure 3.1. TTL-RS485 comparison

This works well and prevents the existence of ground loops, a common source of communication problems, moreover allows transmission across large distance (1200m) and more speed than other serial standards as RS232. The best results are achieved if the lines are twisted.

A RS485 Network can be half-duplex or full-duplex, both allow communication in two directions, but using a half-duplex network only one at a time it is possible. For this reason, TTL-RS485 converter chips, as MAX485, have two pins to set reception or transmission mode.

Since in Home Net System the RS485 standard is used for a master-slave network, it is sufficient a half-duplex network. The communication models used are based on the MAX485 chip [9] that supports up to 32 devices on the same connection. The transmission speed is set to 1 Mbits/s.

3.1.2 Arduino

Arduino is the most widespread microcontroller platform for rapid prototyping. There are several boards, the most famous is Arduino UNO based on Atmega328P microcontroller, a smaller equivalent board is Arduino Pro Mini that uses the same microcontroller but does not include USB converter for loading firmware, so it is necessary an external adapter.

[1]TTL definition: "Sparkfun RS-232 vs. TTL Serial Communication" https://www.sparkfun.com/tutorials/215

3.1.3 Raspberry Pi

Raspberry Pi is a series of small single-board computer based on ARM architecture become famous thanks to the low cost and portability. The main boards are Raspberry Pi and Raspberry Pi Zero, the first is the flagship model, it has received several updates over time, version 4 is currently available (Raspberry Pi 4), Raspberry Pi Zero is the smallest and cheapest board with reduced computing power, version W is equipped with WiFi and Bluetooth connectivity.

3.2 Build Devices

3.2.1 HNSBroker

HNSBroker is implemented using a RaspberryPi 3, this model is equipped with a WiFi module so no additional components are needed.

3.2.2 HNSHub

Since every room needs a controller, a RaspberryPi Zero W (RPiZ) is used for the implementation of HNSHub to reduce energy consumption and cost. Version W is necessary for MQTT communication with HNSBroker over LAN.

A RS485 module is necessary to communicate with HNSDevices in the room. Although the RPiZ is bundled with UART, it is not possible to use it for RS485 half-duplex communication since a real-time GPIO is needed for reception or transmission enabling, using a standard Linux distro such control is not feasible. Real-time management is required to reception enabling immediately after the end of transmission, a delay causes a loss of data transmitted by HNSDevice.

In order to resolve this problem and ensure the compatibility of the software with any Linux device, HNSHub uses an RS485-USB converter that automatically manages transmission and reception mode. The connection scheme is shown in Figure 3.2

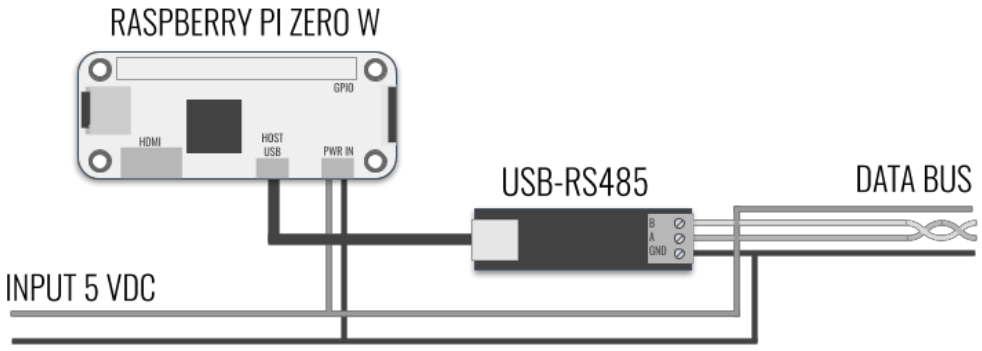


Figure 3.2. Connection scheme of HNSHub

3.2.3 HNSDevice

HNSDevice is based on Arduino Pro Mini, which uses an Atmega328P microcontroller, and a RS485-TTL Converter for communication with HNSHub.

The following components are needed for building:

- Arduino Pro Mini 5v 16MHz
- RS485-TTL Converter based on MAX485 Chip
- (x2) 10 Pin 2.54mm single row straight female pin header
- 6 Pin 2.54mm single row straight male pin header

For a simple human interface it uses:

- WS2812B RGB Led
- Touch button

3.2.3.1 Circuit Schematic

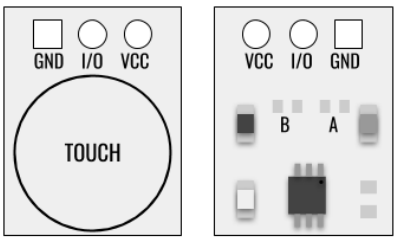
The circuit schematic of the device is composed of several modules, connected as follows.

RS485 Module is based on MAX485 Chip, it has four pins for communication and control:

- *Driver Input (DI):* It is used as transmission input, so the TX pin of Arduino is connected to it.
- *Receiver Output (RO):* Since it is the output comes from the network, it is wired to RX pin.
- *Receiver Output Enable (RE) Driver Output Enable (DE):* RO is enabled when RE is low, instead DI is activated with DE at the high logic level. So these pins can be wired together and connected to a single pin of Arduino (pin 8), when this pin is high DI is enabled, otherwise, RO is.

RGB Led Module is based on WS2812B chip [10] that is a smart control LED in which control circuit and RGB LED are integrated into a package of 5050 components. The chip has four pins: DIN, DOUT, VCC, and GND, DIN and DOUT are respectively input and output of control data signal, since in this application only one LED is used, DOUT is left disconnected. DIN is wired to pin 6 of Arduino. WS2812B has a great advantage that requires only one pin for control of color and brightness, on the contrary, a normal RGB LED it would need three PWM pins, each with a different resistor depending on color.

Touch Button Module is a capacitive touch sensor based on the dedicated TTP223 [11] touch sensor IC. The module provides a single integrated touch sensing area of 11 x 10.5mm with a sensor range of 5mm. An on-board LED will give a visual indication of when the sensor is triggered. When triggered, the module's output will switch from its idle low state to high (default operation). Solder jumpers (A B) allow for reconfiguring its mode of operation as described in Figure 3.3 [12]. Pin I/O of the module is wired to pin 2 of Arduino.



(a)

Effect	A	B
Jog High Output	No Weld	No Weld
Jog Low Output	Weld	No Weld
Self Lock High Output	No Weld	Weld
Self Lock Low Output	Weld	Weld

(b)

Figure 3.3. Front and back of touch module button (a). Trigger setting modes(b)

Arduino Pro Mini's pins (RX, TX, 2, 6, 8) are wired as described above, pin 4 is used for communication protocol (ready packet software interrupt). Since the other pins can be employed to plug an external module designed for a specific application, two female 10 pins header are connected to the Arduino board. Also, a male 6 pins header is used to connect the programmer for loading firmware (see Figure 3.4).

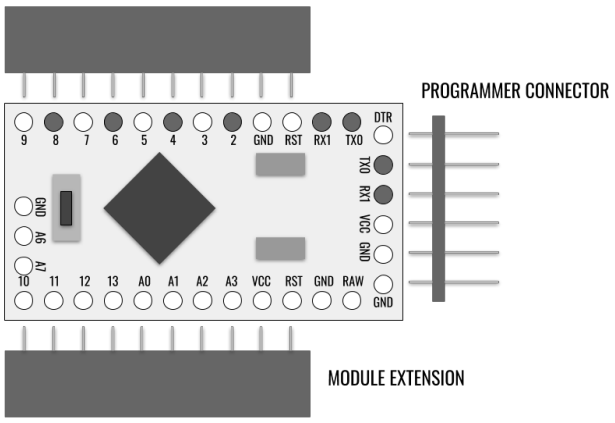


Figure 3.4. Header connection to Arduino. The gray I/O pins are not available for module extension

The overall circuit schematic is presented in the following image (Figure 3.5):

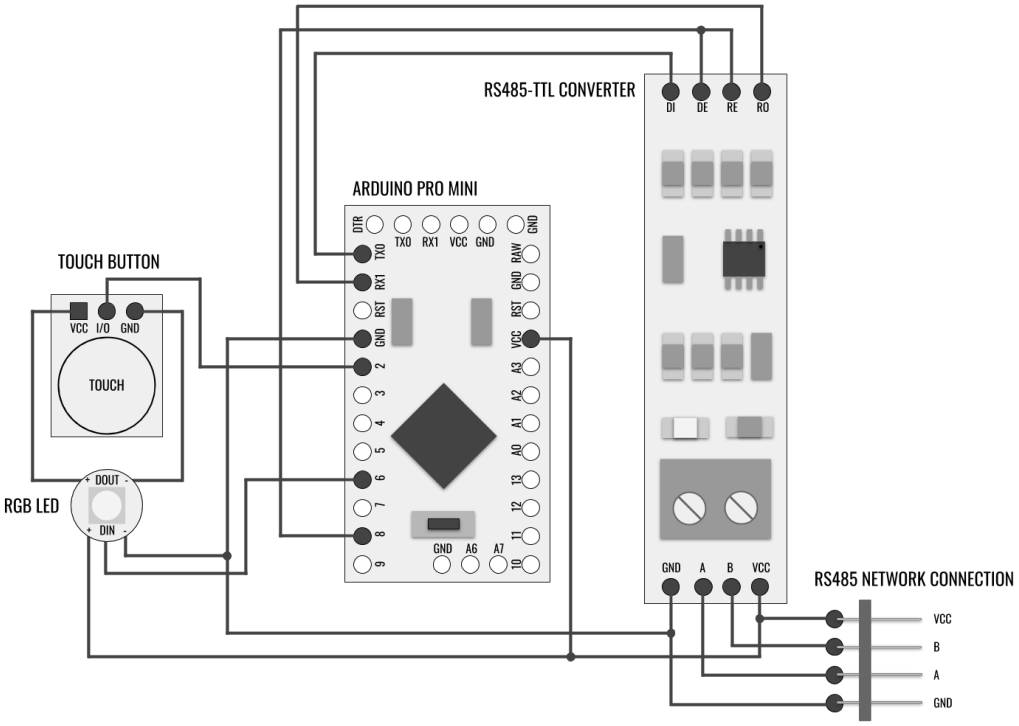
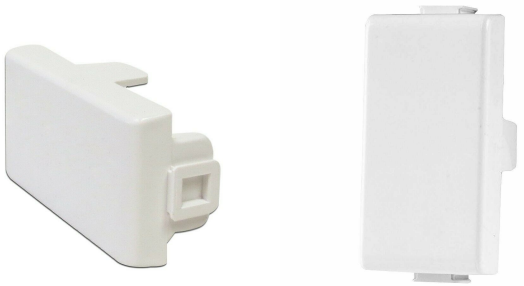


Figure 3.5. HNSDevice's circuit schematic

3.2.3.2 Mounting Support Structure

HNSDevice is designed to be installed in a false pole of a standard Italian electrical system, to improve ease of installation can be used a mounting support structure developed specifically for the type of plant.
The following support is designed for BTicino Matix Series (Figure 3.6)[2]



(a) **(b)**

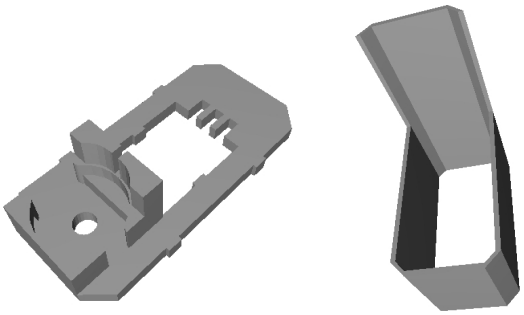
Figure 3.6. False pole of BTicino Matix Series

[2]Image source: https://picclick.it/FALSO-POLO-40Pz-Tipo-AM5000-BTicino-MATIX-COMPATIBILE-402107516465.html

The mount is made of two parts:

- *Holder of touch button and led* (Figure 3.7a): This component is used to assemble RGB led and touch button in a way that they are in contact with the plastic of false pole.

- *Shell* (Figure 3.7b): This part allows us to fix all the components and protect the device.



(a) **(b)**

Figure 3.7. Holder of touch button and RGB led (a). HNSDevice's shell (b)

3.2.3.3 Mounting Example

The assembly procedure of HNSDevice is composed of the following steps:

- **Step 1:** Desolder male header of communication control of RS485 converter module (Header of DI DE RE RO pins) and terminal of A B lines. Optionally to reduce energy consumption, it is advisable to remove the SMD led as illustrated in the following picture. (Figure 3.8a)

- **Step 2:** Weld headers on Arduino Pro Mini (Figure 3.8b)

- **Step 3:** Glue together RS485 module and Arduino as shown in Figure 3.8c

- **Step 4:** Setting touch button by welding jump A. In this way, the module simulates a normal open button with a pull-up resistor. Optionally: remove the SMD led

- **Step 5:** Assemble RGB led and touch button on holder (Figure 3.8d)

- **Step 6:** Connect the modules according to circuit schematic (Figure 3.5)

- **Step 7:** Join all with shell (Figure 3.8e) (Figure 3.8f))

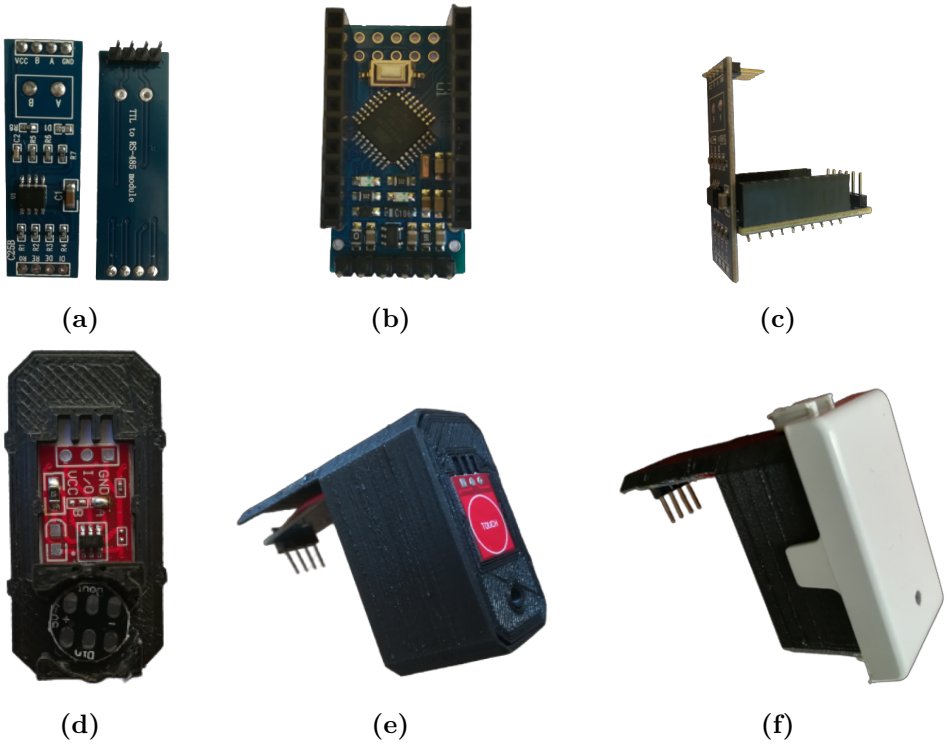


Figure 3.8. Assembly steps of HNSDevice

3.2.4 HNSSocket

HNSSocket is a smart socket able to measure the electrical power consumed by a load and detach it using a double relay module. Usage of a double relay allows us to disconnect both phase and neutral of 10A load (bipolar switch), otherwise only phase of 20A load (single pole switch). This device is based on HNSDevice with the addition of an external module to connect it to the relay module and power measurement circuit. Since the use of a switching power supply causes measurement disturbance, the measurement circuit integrates a linear power supply for the device.

3.2.4.1 Open Energy Monitor

Open Energy Monitor is a project with the aim to provide an open source tool for energy monitoring. HNSSocket uses, for power measurement, the method described by this project, it employs EmonLib library for software sensors readings and the acquisition circuit is based on their measuring instruments.

3.2.4.2 Power Measurement Circuit

In order to measure the electrical power, it is necessary to use two different sensors, one for the current [13] and the other for the voltage [14]. For the latter a 230VAC - 6VAC transformer is used with series resistors (R3 R4 RV1) to obtain 1VAC, since the transformer has two secondary windings is possible to use one of these to provide energy supply for HNSDevice, to adjust the output voltage an LM317 chip [15] is

used. This integrated circuit allows to get 5VDC using two resistors R1 and R2; the value of which has been calculated using the following equation:

$$V_o = V_{REF}(1 + \frac{R2}{R1}) + i_{ADJ}R2 \tag{3.1}$$

Where:

$$V_O = 5V \qquad V_{REF} = 1.25V \qquad R1 = 330\Omega \qquad i_{ADJ} = 50uA$$

Therefore obtaining:

$$R1 \simeq 977\Omega$$

The nearest commercial value is

$$R1 \simeq 1k\Omega$$

The current is measured instead by using a current transformer with a burden resistor. This type of sensors is installed around phase cable, when alternative current passes through the conductor, a magnetic field is generated, which in turn induces a current inside the transformer. Using the burden resistor, a voltage proportional to the alternating current is obtained. The specific sensor used is SCT-013-030 [16] that measures current from 0A to 30A, with an alternative voltage output with a maximum value of 1VAC in correspondence of 30A. This sensor is connected by a 3.5mm jack plug, represented in the circuit by J4 connector (Figure 3.9).

Therefore both sensors provide 1VAC output signal [-1, +1], to bring it to Arduino's analog input voltage range [0, 5], the mean value of signals is shifted to 2.5VDC, in this way their value is between 1.5v and 3.5v. The power measurement circuit is connected to Arduino through a 6 pins male header (J1) with the following pinout:

- Voltage sensor output (Pins 1, 2)
- Arduino 5v power supply
  - Vcc (Pin 3)
  - GND (Pin 4)
- Current sensor output (Pins 5, 6)

As connection cable, an USB cable can be used, preferably shielded to reduce external interference. The overall circuit is shown in Figure 3.9

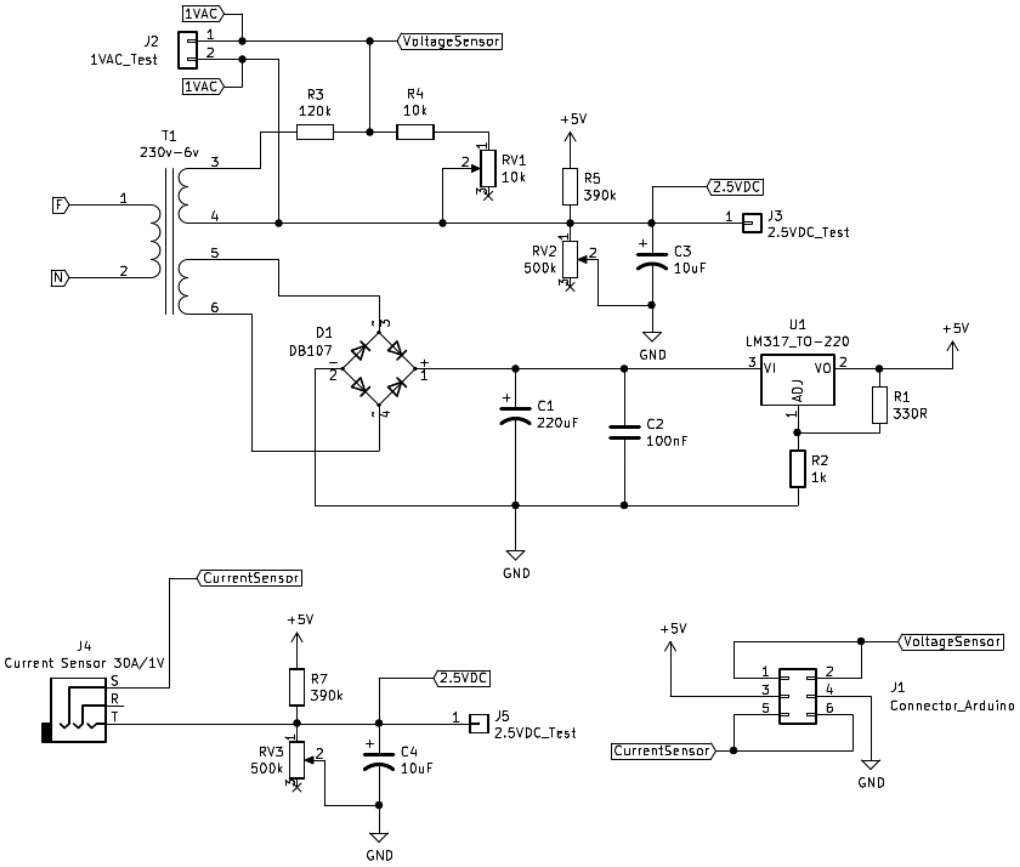
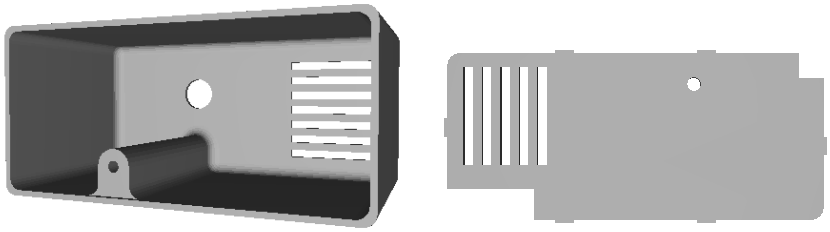


Figure 3.9. Power measurement circuit

The circuit is assembled on a stripboard with a size of 60x35 mm (Figure 3.10a) and enclosed within a plastic box printed with a 3D printer (Figure 3.10b). Figure 3.11 shows 3D CAD model of the case.

(a) **(b)**

Figure 3.10. Realization power measurement circuit

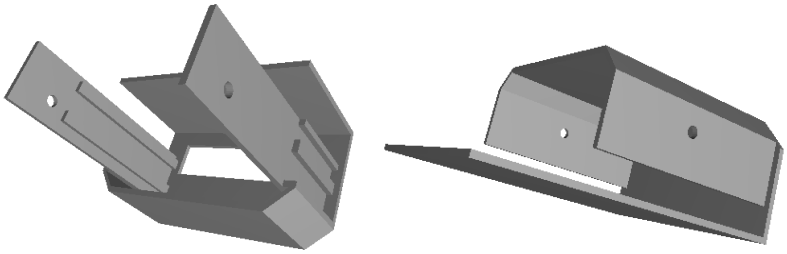


(a) (b)

Figure 3.11. Box of power measurement circuit

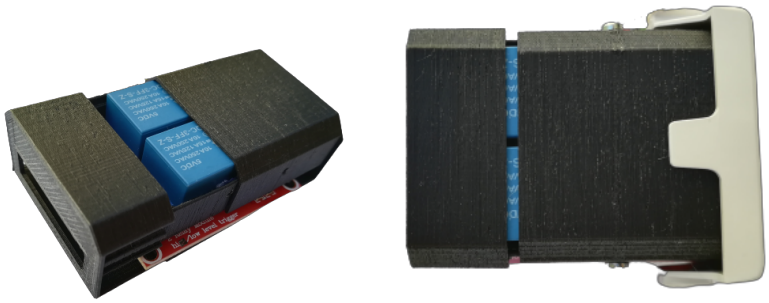
3.2.4.3 Relay Case

The relay module can be installed into a false pole like HNSDevice using a case printed with a 3D printer, in Figure 3.12 is shown 3D CAD models and a show of assembly in Figure 3.13.



(a) (b)

Figure 3.12. Relay case. On the left the terminal cover, on the right instead, relay shell



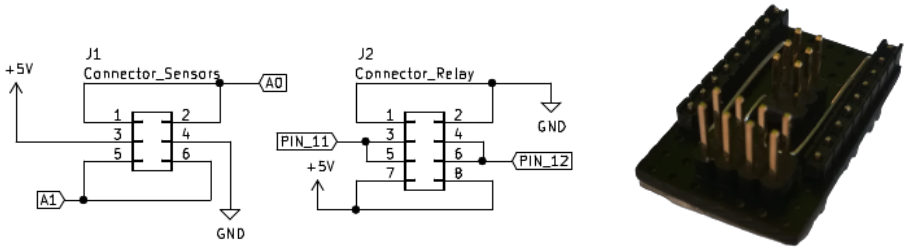
(a) (b)

Figure 3.13. Installation of relay module inside a false pole

3.2.4.4 External Module

In order to connect HNSSocket to the relay module and the power measurement circuit, an external module is used. The voltage sensor output is wired to A0 analog

pin input of Arduino, and the current sensor to A1, the relay module is connected to 11 and 12 digital I/O pins. The schematic circuit is reported in Figure 3.14a and Figure 3.14b shows a physical realization.



(a) **(b)**

Figure 3.14. Schematic circuit of the external module (a). External module used to connect relay module and sensors (b)

3.2.4.5 Overall Scheme

Since the transformer in the power measurement circuit, used as power supply for HNSDevice, has a limited power, the relay module is powered by data bus (i.e. RS485 Network)

The overall circuit is shown in Figure 3.15.

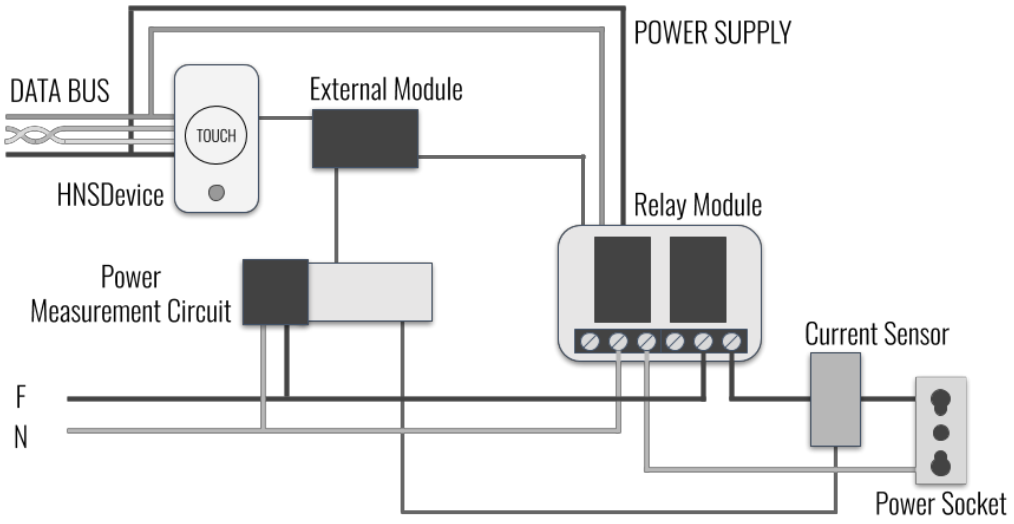


Figure 3.15. Overall circuit of HNSSocket

3.3 Communication Protocols

In the system, two different types of protocols are needed, within the RS485 network is necessary a light protocol due to the low computing power of HNSDevices. Over

the LAN, MQTT provides an application layer based on topic-payload message model that must be specialized for the purpose, so it is necessary an agreement about the structure of topic and type of payload exchanged.

3.3.1 HNSProtocol

Communication between HNSDevices and HNSHub is based on a custom protocol, over RS485 network, called HNSProtocol. The packet structure is composed of bytes; there are two bytes as a starter, two bytes for addressing, one byte for payload size, one byte for message type, and two bytes for error-detecting. Each device has an address, since an RS485 network can support up to 32 devices, 8 bits for address are more than enough, so a message contains both addresses of recipient and sender. Error detecting is based on a 16 bits Cyclic Redundancy Check (CRC 16bit), Figure 3.16 shows packet structure.

Figure 3.16. Packet structure of HNSProtocol

Regarding to Figure 3.16, the individual fields are described as follows:

- **START**: Two bytes, with specific values, allow to define the beginning of a packet
- **DST** (destination): Address of packet recipient
- **SRC** (source): Address of packet sender
- **DATA SIZE**: Indicates size, in bytes, of payload (DATA)
- **TYPE**: Used by the sender to indicate how the recipient should interpret the payload
- **DATA**: Payload of message
- **CRC 16 Bit**: A code for transmission errors detection

3.3.1.1 Address Assignment

HNSHub is the master of a room's network, and it deals with assigning addresses to the HNSDevices. Since addresses in [250-255] range are reserved for specific purposes such as address assignment, not all address space is available Initially, HNSDevices without address, have a special default address called NO_ADDRESS, HNSHub has a static address HUB_ADDRESS, it too belonging to reserved space. Since HNSHub is the master of the network, a new HNSDevice can not directly request an address to the controller, but it must wait for a special message from HNSHub.

The assignment procedure is as follows. Master periodically search for a new device by sending a message with SYNC type directed to devices with NO_ADDRESS address, when one of them responds, the assignment starts.

The new device responses with a packet that contains information about the device such as HNSModel and HNSID, first indicates device type (e.g. smart socket or temperature sensor), the other one is a unique code associated with the chip of the device. Then master reply by sending the address of the new device, finally, HNSDevice sends an acknowledgment.

In Figure 3.17 is represented the address assignment procedure.

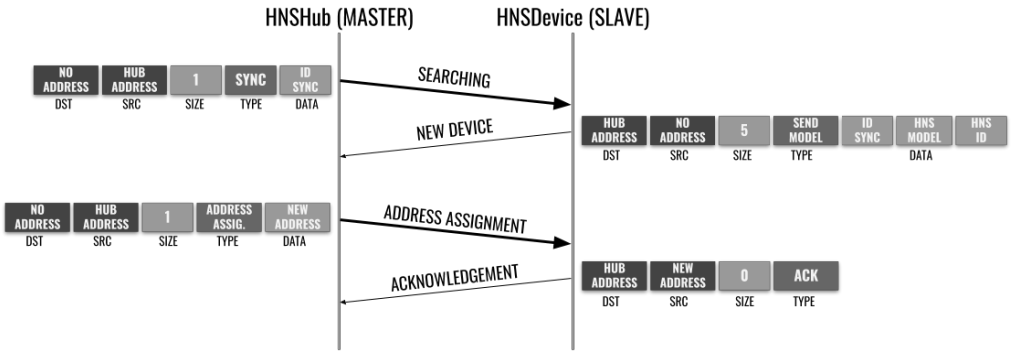


Figure 3.17. Address assignment procedure

Data in the SYNC message contains a code (ID_SYNC) for the identification of the synchronization session.

Since at the beginning every device without an address, have the same NO_ADDRESS address, more than one of them could respond to SYNC message during address assignment, thus leading to collisions, moreover, a collision cannot be detected because the channel is half-duplex (i.e. a device can not notice if another device is transmitting during the transmission). In order to avoid that every device replies to the same SYNC message, each device waits over a random period of time, if no one responds during this period, then it replies. Using a pseudo-random generator, it is necessary to choose a random seed in a way that it is different for each device, it can be obtained by reading noisy analog inputs, (i.e. analog input without any load).

3.3.1.2 HNSSocket Application Protocol

HNSProtocol provides a logic level for master-multislave communication, so an application protocol can be developed over it by using TYPE and PAYLOAD of the packet. In Home Net System, each type of HNSDevice (e.g. HNSSocket, HNSTempSensor) uses a different application protocol to communicate with HNSHub in order to obtain high efficiency for different type of data exchanged.

The application protocol used by HNSSocket can handle a device with up to eight power sockets, each of them is called subsocket. A subsocket is composed of three elements:

- *Switch*: Actuator that allows load shedding

- *Auto Switch*: Switch programmable for automatic load shedding if the power limit is exceeded
- *Sensor*: It is divided into three types:
  - *Real Power*: It provides only real power
  - *Real Power-Apparent Power*: It gives both real and apparent power
  - *Real Power-Voltage and Current*: It provides real power and apparent power expressed implicitly by voltage and current.

Since the important fields of packet are DATA_SIZE, TYPE, and DATA (payload), the protocol is described by referencing only to these.

Reading Device

The protocol supports two methods for reading the status of subsockets: in the first, called REFRESH_ALL, HNSSocket always sends data of all components (switch, auto-switch, sensor), in the second (REFRESH) instead, only data of changed parts from the previous reading. When a device has several subsockets, this last method is more efficient and faster.

In REFRESH_ALL method, the exchanged packets are as follows (Figure 3.18)

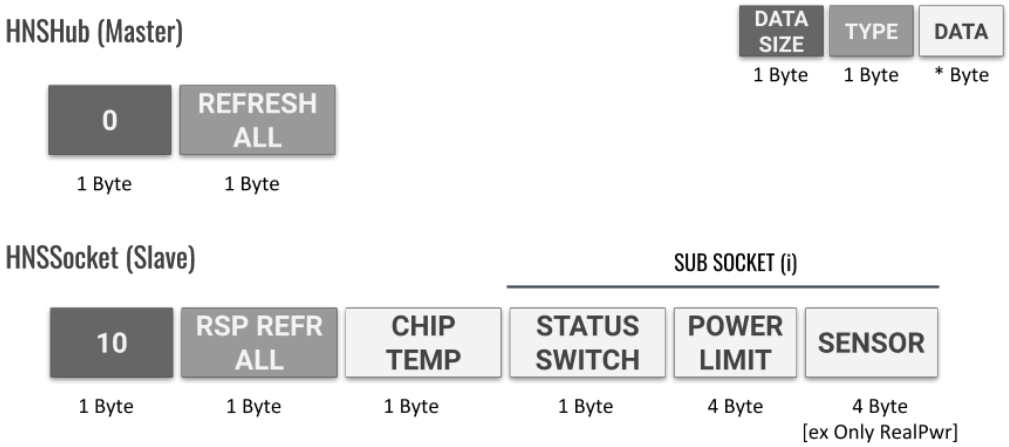


Figure 3.18. Reading HNSSocket by REFRESH_ALL method. DATA field contains information about every subsocket handle by the device. Payload contains also chip temperature CHIP_TEMP

HNSHub sends a packet with REFRESH_ALL type, device responses with RSP_REFRESH_ALL packet type that contains the temperature of the chip and information about all components of each subsocket. Chip temperature can be used to detect overheating that could cause a fire, since accuracy is not required, this temperature is expressed by 1 byte in Celsius degrees.

Possibles values for STATUS SWITCH are:

- *STATUS_POWER_ON*: Switch ON
- *STATUS_POWER_OFF*: Switch OFF
- *STATUS_POWER_ALLARM*: Switch OFF due to power limit exceeding

POWER_LIMIT contains the power limit set and SENSOR the reading from sensors, (e.g. real power), both are expressed by float number (4 Bytes) in big-endian (i.e start the transmission with most significant byte MSB). In Figure 3.19 is shown a RSP_REFRESH_ALL packet example sent by an HNSSocket equipped with two subsockets with real power sensor.

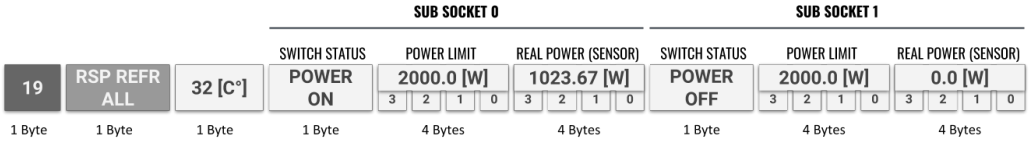


Figure 3.19. RSP_REFRESH_ALL packet example. HNSSocket has two subsockest with real power sensor

In REFRESH method instead, the exchanged packets are as follows (Figure 3.20)

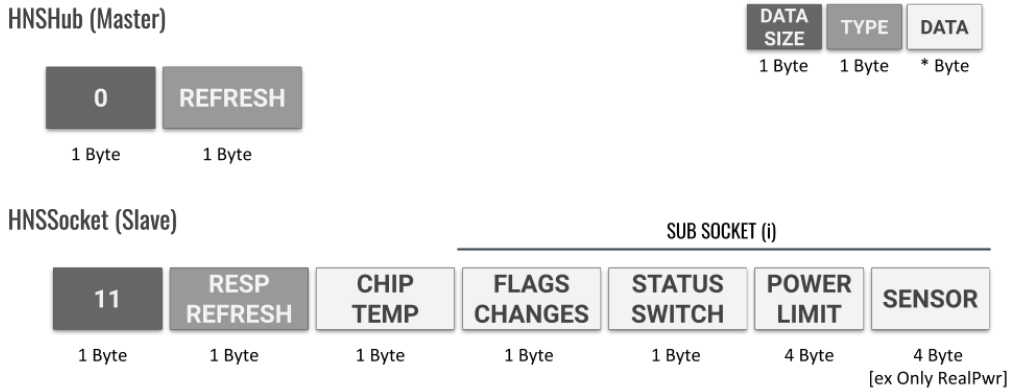


Figure 3.20. Reading HNSSocket by REFRESH method. FLAGS CHANGES indicates subsocket and which components are changed

The main difference with REFRESH_ALL method is that it is not necessary to send all subsocket and all elements, but only the changed components, in this way speeds up the transmission and avoids waste of bandwidth. Consider for example an HNSSocket with eight subsockets in which six of these are often off, in every REFRESH_ALL message, most of the information is useless.
FLAGS CHANGES byte indicates the changes from the previous update and has the following structure shown in Figure 3.21.

The first three most significant bits (MSb) indicate which subsocket FLAGS CHANGED refers to. The last three less significant bits (LSb) are divided into three flags (STATUS_SWICTH, POWER_LIMIT, SENSOR), a flag is set if its

associated component is changed. The changed elements follow FLAGS CHANGED in DATA field respecting the order shown in Figure 3.21 (STATUS_SWICTH, POWER_LIMIT, SENSOR)

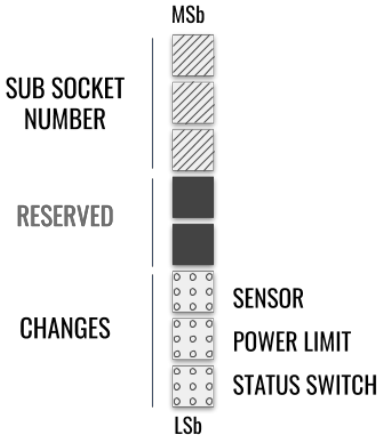


Figure 3.21. FLAGS CHANGED structure.Two bits are RESERVED (not used)

Taking the above example, assuming only the power of subsocket_0 is changed from a previous message, the packet shown in Figure 3.19 becomes the one in Figure 3.22 using REFRESH method where payload size has been reduced from 19 to 6 bytes.

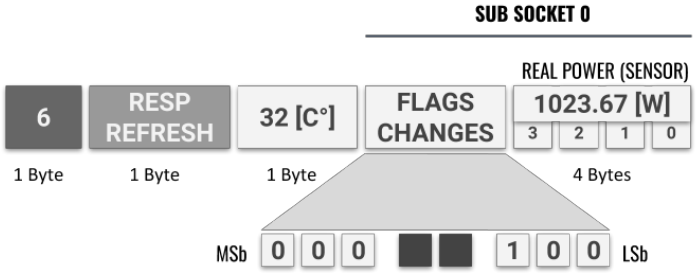


Figure 3.22. RSP_REFRESH packet example. HNSSocket has two subsockest with real power sensor. Only real power changed from the previous message

Control Device

Application protocol also uses FLAGS CHANGES to send commands. HNSHub sends to HNSSocket a message in which the type indicates new status to set (ON/OFF), and FLAGS CHANGES what to apply, then the device replies with an acknowledgment. Possibles message TYPEs are:

- *SET_STATUS_ON / SET_STATUS_OFF*: It used by the master to switch ON/OFF the components indicated by FLAGS CHANGES
- *RSP_SET_STATUS_ON / RSP_SET_STATUS_OFF*: It used by the slave as an acknowledgment

The protocol allows us to set the status of a Switch and power limit of an Auto Switch (it can also be disabled using SET_STATUS_OFF type). The general structure of the packet is shown in Figure 3.23.

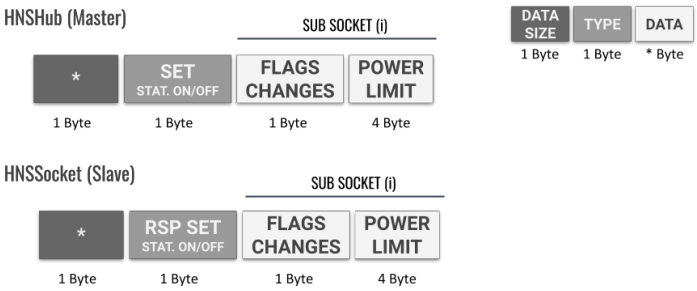


Figure 3.23. Sending command to HNSSocket. DATA contains POWER_LIMIT to set if AUTO_SWITCH bit of FLAGS CHANGES is set

Two command examples are shown in Figure 3.24, HNSSocket has two subsockets, in Figure 3.24a HNSHub turn on the switch of subsocket_1, in Figure 3.24b it sets the power limit of subsocket_1 to 1500W. HNSSocket replies with an acknowledgment containing the same FLAGS CHANGES sent by master in both cases.

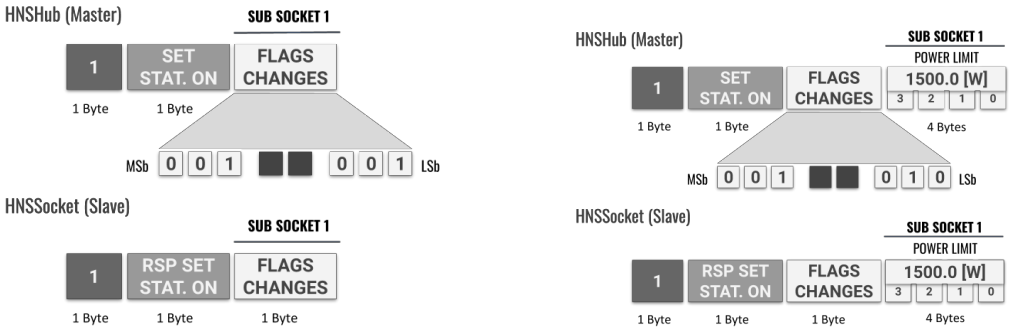


Figure 3.24. HNSHub power on subsocket_1 switch of HNSSocket (a). HNSHub sets power limit of subsocket_1 to 1500W (b)

3.3.2 MQTT HNSSocket

The purpose of the controller inside each room is to collect data from HNSDevice through HNSProtocol and make these available to HNSBroker over MQTT protocol. Since only the room's controller is connected to LAN, all devices in a room are regarded as a single device represented by the HNSHub, then all HNSDevices are managed as virtual devices. Each virtual device (HNSDevice) has a unique name inside the LAN called MQTTDeviceID; this code is based on the device model (e.g. HNSSocket) followed by HNSID and four less significant digits of HNSHub's MAC address, both of these are in hexadecimal representation. The use of MAC addresses allows us to quickly identify controllers of devices then classify the HNSDevices by room/area.

The following protocol was developed together with Alessandro Lamin. In order to make Shelly and HNS products coexist in the same home system, messages exchanged are based on the structure used by Shelly products, so some technical choices depend on this goal, such as prefix in message topic, HNS uses *custom* prefix, Shelly instead *Shelly*.

3.3.2.1 Announce Message

When HNSHub finds a new HNSDevice through address assignment procedure over HNSProtocol, it is necessary to notify the new device to HNSBroker, in order to do that, HNSHub sends an MQTT message to HNSBroker, this message is called *announce message*. Announce message topic is *custom/announce* and its payload is an array of JSON objects for each subdevice of the new HNSDevice. A subdevice is a basic component such as a sensor or an actuator, for example, in the case of HNSSocket, each element of a subsocket (Switch, Auto Switch, Sensor) is a subdevice. Each JSON object has the structure shown in Figure 3.25.

Key	Value	Description
ip_address	ex: 192.168.1.20	IP Address of device. In HNS system it is IP of HNSHub
device_id	*MQTTDeviceID*	Name of device
sub_type	switch|power	Indicates sub type: - swicth: actuator - power: real power sensor
sub_id	<Integer>	ID of subsocket
topics_prefix	custom	Prefix of message topic
update_topic	<topics_prefix>/<device_id>/ <sub_id>/<sub_type>	Topic used by device to publish updates
command_topic	<update_topic>/command	Topic used by HNSBroker to send commands to device

Figure 3.25. Components of JSON object of announce message payload

3.3.2.2 Switch Subdevice

A subdevice with *switch* sub_type is a switch that allows load shedding, it can be read and controlled to turn on and off the load.

Status Update

HNSHub publishes a status update every five seconds by default, but if status changes, it sends the update instantly. The message structure is the following; the payload can be on or off.

Topic: custom/<device_id>/<sub_id>/switch

Payload: on

off

Status Control

In order to control a switch, HNSBroker publishes a message in which the topic is the same as a status update but with the addition of */control*. The payload indicates status to set.

Topic: custom/<device_id>/<sub_id>/switch/control

Payload: on

off

When the command is received, HNSHub forwards it to HNSDevice over HNSProtocol, once executed, the device sends an acknowledgment to HNSHub that publishes the following MQTT message as acknowledgment for HNSBroker.

Topic: custom/<device_id>/<sub_id>/switch/control/ACK

Payload: on

off

Figure 3.26 shows the sequence of messages for status control (ON -> OFF). The thick line on the straight line that represents the HNSBroker, indicates the overall time required for command execution.

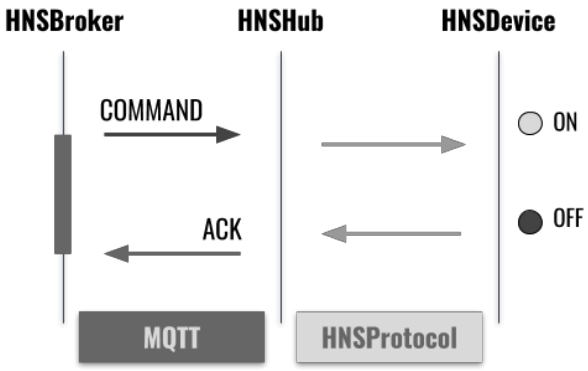


Figure 3.26. Example of the sequence of messages exchanged between HNSBroker, HNSHub and HNSDevice in order to turn off a switch.

3.3.2.3 Power Subdevice

Power subdevice is a sensor that measures real power consumed by a load. HNSHub publishes sensor update every five seconds by default, but if a change occurs, it immediately sends an update. The real power measured is contained in the payload; it is expressed in Watts to two decimal places.

Topic: custom/<device_id>/<sub_id>/power

Payload: <real_power[0.2f][W]>

HNSSocket could also provide measurements of apparent power, voltage, and current, but these are not used for compatibility reasons with Shelly products.

3.4 Software

3.4.1 HNSDevice

The software of HNSDevice is written in C/C++ programming language and supports Arduino language (Wiring) in order to simplify library development using third-party libraries. The project structure is as follows:

- **HNSCore**: It is a folder that contains all files for communication over HNSProtocol. Below the most important ones.
  - *HNSRouting*: It contains the functions for reading and writing packets on the channel.
  - *HNSMacro*: The only acceptable values of TYPE field of packet are defined in this file.
  - *HNSError*: It contains tools useful for debugging
- **HNSCommand**: Each HNSModel uses a specific application protocol (e.g. HNSSocket Application Protocol described above), in the HNSCommand folder there is a file for each application protocol supported, that contains keywords for communication.
- **HNSDevice**: It contains specific files for HNSDevices.
  - *HNSDevice*: The main function is defined in this file
  - *HNSSync*: It incorporates the functions for address assignment procedure.
- **HNSDeviceLib**: It includes libraries for each HNSModel supported
- **HNSArduino**: To optimize the communication over RS485, HNSDevice uses an interrupt-based transmission incompatible with interrupts and UART management of Arduino, so HNSArduino contains Arduino library modified by removing Serial() family function and interrupt handlers.

Also HNSHub code is written in C/C++, it uses the same structure just described for HNSCore folder and it shares the HNSCommand folder.

3.4.1.1 Developer Interface

All of the previous folders are included in the HNSLibrary, the user can customize the code of a HNSDevice by modifying the main file that contains an easy developer interface inspired by Wiring. Main function is handler by library, so the interface provides two functions:

- *setup()*: Function performed only once at the beginning. (Like setup() in Wiring)
- *routine()*: Function loops consecutively. (Like loop() in Wiring)

In the main file is also necessary to define the HNSModel of HNSDevice (myModel) and the function to be executed when a new packet is received (netHandler()) In Figure 3.27 is shown the main file of the HNSSocket device.

```
1  //define HNSModel
2  //macro defined in HNSSocket library
3  HNSModel myModel = HNSModel_HNSSocketPower;
4
5  //function pointer of communication handler
6  void (*netHandler)(void) = HNSSocketPower_netHandler;
7
8  //function performed only once at the beginning
9  void setup(){
10    HNSSocketPower_begin(); //function defined in HNSSocket library
11 }
12
13 //loop function
14 void routine(){
15    HNSSocketPower_routine(); //function defined in HNSSocket library
16 }
17
```

Figure 3.27. Main file of HNSSocket

3.4.1.2 Ready Packet Interrupt (RPI)

The microcontroller used by HNSDevices is Atmega328P and allows us to manage several hardware interrupts including the arrival of a character from the UART (UART0_RX_vect). The handler of this interrupt is a function that builds the packet and, when it is ready, throws a software interrupt called Ready Packet Interrupt. RPI exploits a type of hardware interrupt called PCINT (Pin Change Interrupt), which is thrown when a pin changes its status. Typically this type of interrupt is used to monitor the status of an input pin, however, it can also be used on a pin configured as output. When a packet is ready, the RPI is thrown by changing the status of an output pin with PCINT enabled. Finally, the handler of PCINT performs netHandler() function defined in the main file. The pin used for this purpose is bit 4 of register D (pin 4 of Arduino). Atmega328P does not allow us to define a PCINT handler (Interrupt Service Routine ISR) for a single pin but only for a group of them, pin 4 belongs to Pin Change Interrupt Request 2 and the interrupt code is PCINT2_vect.

The ISR of Ready Packet Interrupt is shown in Figure 3.28.

```
ISR(PCINT2_vect){
  //get which pin of PCINT2 group is changed
  unsigned char pins_changed = oldPORTD ^ PORTD;
  oldPORTD = PORTD;

  /*
  * HANDLER PACKET READY INTERRUPT
  */
  if(rbi(pins_changed, PD4)){  //execute if only pin 4 changed state
    disable_packet_ready_interrupt();
    sei();

    netHandler();

    enable_packet_ready_interrupt();
  }
}
```

Figure 3.28. ISR of packet ready interrupt

Figure 3.29 recaps the packet management procedure.

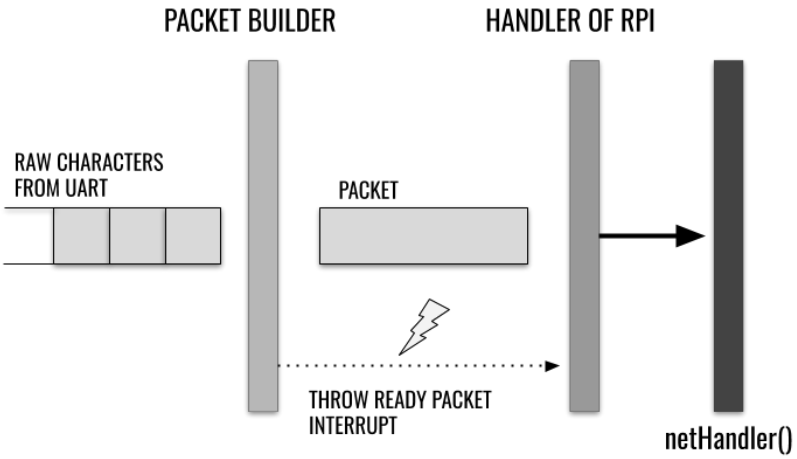


Figure 3.29. Read and management of a packet. Packet builder builds a packet from raw UART input, when a packet is ready, throws a RPI, then the handler of RPI performs netHandler() function

3.4.1.3 HNSSocket

One of the main advantages of the previous framework (HNSLib) is that it allows us to separate the control logic from the communication logic necessary to connect HNSSocket with HNSHub, making code development easier. With reference to the main file of HNSSocket, shown in Figure 3.27, the communication is managed by HNSSocketPower_netHandler function that applies the HNSSocket Application Protocol described in section 3.3.1.2. Instead, HNSSocketPower_begin and

HNSSocketPower_routine are responsible for control logic, they deal respectively with setting pins mode and managing the actuators-sensors.

The code of HNSSocketPower_routine is based on a finite automation by the use of function pointers in which each status corresponds to a function. The current status is stored in *currentStatusRoutine* function pointer, HNSSocketPower_routine performs this function continuously.
The possible states of a single subsocket are as follows:

- *routineStatusOFF*: The switch is powered off and the device does not measure the power consumption.

- *routineStatusON*: The switch is powered on and the reading of power occurs periodically.

- *routineStatusALARM*: When power consumption exceeds the limit, the device enters in the alarm status in which switch is turned off and the LED indicator blinks with red light. After 5 seconds the status is restored to routineStatusON.

Figure 3.30 shows the finite automation of HNSSocket control logic. In routineStatusOFF (OFF) and routineStatusON (ON) status, the device can be turned on/off through the touch button or remote control (command from HNSHub). In routineStatusON, when the value of power consumption is above the limit, the status changes to ALARM, then after a timeout (five seconds), the status is restored to ON. If the load shedding is managed via remote control, the power limit is set to maximum power (e.g. 1500W), in this case, the status alarm is needed to avoid damaging the device.

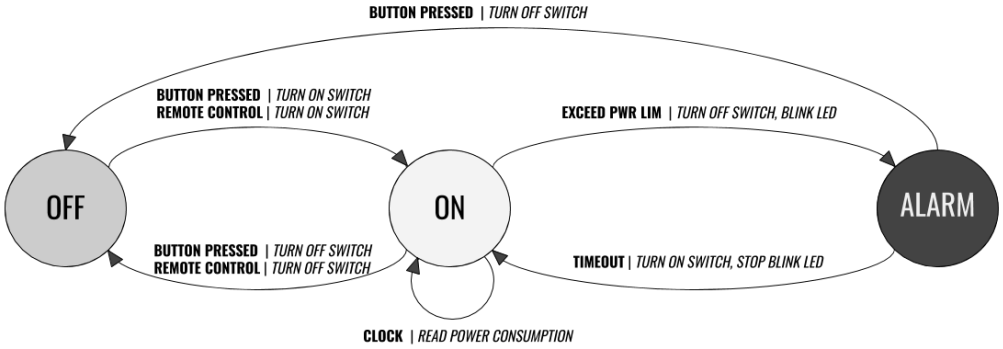


Figure 3.30. Finite automation control logic of HNSSocket. (**EVENT** | ACTION)

Since the communication with HNSHub takes place through interrupts, HNSDevice can communicate in every status, however since to measure the power consumption HNSDevice samples voltage and current sinusoidal wave, an interrupt causes an incorrect measurement, so it is necessary to disable the communication during the reading. The reading takes about 120 milliseconds and in this time interval the device is unreachable, therefore a synchronization method with HNSHub is required.

In order to synchronize the requests from the HNSHub and the HNSSocket's availability windows, HNSHub sends requests with a period equal to the reading time (~150ms), and the HNSSocket waits for a request after every measurement. In this way, once the synchronization has taken place, HNSHub's requests are always successful and, since the response time is reduced, the communication channel is used efficiently. When the device changes status from OFF to ON, it is not synchronized with HNSHub, so the first request often fails.

Measure power consumption

HNSSocket measures power consumption by using EmonLib, a third part library provided by the Open Energy Monitor project introduced in section 3.2.5.1. Using the hardware described in section 3.2.5.2, this library allows us to measure: real power, apparent power, power factor, RMS voltage, and RMS current, however, HNSSocket uses only the real power.

In order to obtain an accurate measurement of the voltage and the current, it is necessary to set some parameters of EmonLib. The voltage sensor required to fix VCAL and PHASECAL constants, the first indicates the main voltage when the analog input of the device reads 1VAC, the second is necessary to compensate for the phase shift. This last is not easy to calculate so it is set to the default value (PHASECAL=1.7). During the calibration of the power measurement circuit of voltage sensor, done by RV1 potentiometer, VCAL can be obtained by measuring the main voltage at that moment. The current sensor needs instead only one parameter called ICAL, it indicates the current that flows in the load when the device reads 1VAC. Using a current transformer with a built-in burden resistor (like SCT-013-030), ICAL is easy to obtain and in this case is 30.0 (maximum measurable value) [17].

EmonLib measures the power consumption by sampling the current and voltage sinusoidal wave, to get an acceptable measurement it is necessary to sample a sufficient number of waves, however increasing the number of samples, the measuring time rises too, so find a compromise between number of waves and time required is crucial.

Analysis measurement error according to semi-waves

Library allows us to choose the number of semi-waves to sample, in order to find the right number of semi-waves, several tests were performed by comparing the voltage and current measured by EmonLib and by a multimeter.

The multimeter model used is a TACKLIFE CM02A [18], it can measure AC voltage, DC voltage, and AC current through a current clamp. The technical specifications of interest are reported in Figure 3.31.

Measuring Range		Resolution	Accuracy
0-6 A	[AC 45 ~65hZ]	0.001 A	± 2.5 %
6-60 A	[AC 45 ~65hZ]	0.01 A	± 2.5 %
0-6 V	[AC 45 ~65hZ]	0.001 V	± 0.8 %
60-600 V	[AC 45 ~65hZ]	0.1 V	± 0.8 %
0-6 V	[DC]	0.001 V	± 0.5 %

Figure 3.31. Multimeter TACKLIFE CM02A technical specifications)

The testbed consists of an Arduino UNO that uses EmonLib and reports the measurement via serial monitor at the end of the test, and the multimeter that has been fixed so that the current clamp does not move during the current reading. To collect data read by multimeter instead, several photographs have been taken, then the read values have been reported to a CSV file. The values of semi-waves tested are 20, 14, 12, 10, 8. The data has been analyzed using Pandas, a python library, in Jupyter Notebook.

To calculate the voltage error, it is necessary to calibrate the sensor setting the VCAL parameter of EmonLib, the main voltage when the multimeter reads 1VAC is 242.7 VAC, however, it is not possible to obtain a correct measurement using this value due to low accuracy of multimeter (± 0.8 %). Following several tests, VCAL has been set to 236.5. The error has been calculated as the difference between the means of values read by EmonLib and by the multimeter, then the error has been divided by the multimeter mean in order to obtain a percent value.

$$VoltageError = \left| \frac{meanMultimeter - meanEmonLib}{meanMultimeter} \right| \tag{3.2}$$

Since the current has a higher variance than the voltage (0-16A), for each semi-waves value, the current error has been calculated using several values to consider the different error magnitude according to the current intensity. In order to adjust the current that flows in the load, it has been used a hairdryer that allows us to tweak the temperature and the fan speed with three degrees of freedom (LOW, MEDIUM, HIGH).

Figure 3.32 shows the several combinations that have been tested, and reports, for each of them, the mean of value read by the multimeter.

		TEMPERATURE		
		LOW	MEDIUM	HIGH
FAN SPEED	LOW	NOT TESTED	NOT TESTED	NOT TESTED
	MEDIUM	0.66 A	2.55 A	5.91 A
	HIGH	1.13 A	4.70 A	8.16 A

Figure 3.32. Combinations of hair dryer tested with relative current measured by the multimeter

The overall current error has been calculated as follows:

$$CurrentError = \frac{1}{n}\sum_{i=0}^{n} \mid \frac{meanMultimeter_i - meanEmonLib_i}{meanMultimeter_i} \mid \tag{3.3}$$

Where n indicates the number of different options combinations of the hairdryer tested. (n = 6)

In Figure 3.33 are reported current and voltage error for each semi-waves values. In every row there is the time required to sample the semi-waves, this value corresponds to the period of the main voltage (240 VAC 50Hz) multiplied by number of waves (e.g. READ_TIME(20) = 0.02 * (20 / 2) = 200ms). In order to compare the error of EmonLib with the accuracy of the multimeter (ERR. MULT.), this latter is also reported in the figure.

SEMI-WAVES	READ TIME	VOLTAGE ERROR	ERR. MULT.	CURRENT ERROR	ERR. MULT.
20	200 ms	0.728 %	± 0.8 %	1.341 %	± 2.5 %
14	140 ms	0.924 %		1.697 %	
12	120 ms	0.701 %		1.051 %	
10	100 ms	0.811 %		1.109 %	
08	80 ms	0.819 %		1.128 %	

Figure 3.33. Comparison of the different numbers of semi-waves.

From the data collected, it is clear that the best performance is not obtained using too high semi-waves value, the reason why can be the loss of the instantaneous value by sampling too many waves and calculating the mean of them. Since the accuracy of the multimeter is similar or even lower than the errors of EmonLib, it is not possible to reach a higher accuracy using this multimeter.

Following the data collected, HNSSocket uses, for measuring the real power, 12 semi-waves.

3.4.2 HNSHub

Also, the software of HNSHub is written in C/C++, the project structure is similar to that of HNSDevice, in addition to folders HNSCore and HNSCommand shared

with HNSDevice, the code of HNSHub also consists of the following folders:

- **HNSHub**
  - *HNSSync*: This file incorporates the functions for address assignment procedure.
  - *HNSDevice*: It contains the definition of HNSDevice class that represents a device. It stores several information such as address in RS485 network, deviceID for MQTT communication, and timestamp of the last connection of the device with the HNSHub.
  - *HNSDatabase*: It manages the list of all HNSDevice connected to the HNSHub, an item is an HNSDevice instance.
  - *HNSMQTT*: It includes the MQTTSubDevice base class that represents a subdevice in MQTT communication, (e.g. MQTTSubDevice_socket_swicth class indicates the switch of a subsocket in a smart socket). This class contains information such as the timestamp of the last status update via MQTT, the default, and the minimum update period.
  - *HNSAddressMap*: It manages a hashmap with the aim to obtain efficiently the address of a HNSDevice from its deviceID. This conversion is necessary when HNSHub receives a command from HNSBroker that refers to a device by using its deviceID (MQTT protocol).
- **HNSHubLib**: This folder contains libraries for managing the different HNS-Model supported.

The control flow of HNSHub consists of querying cyclically every registered HNS-Device in the list of HNSDatabase, by using refresh() function defined for each HNSModel. This function performs the reading of data from devices and executes commands coming from the HNSBroker, if they are, via HNSProtocol. Following to refresh data, HNSHub publishes an update to HNSBroker via MQTT protocol if data changed from the last update or a timeout occurred. In order to find new devices in the RS485 network, HNSHub periodically performs HNSSync() function that applies the address assignment procedure described in 3.3.1.1. section.

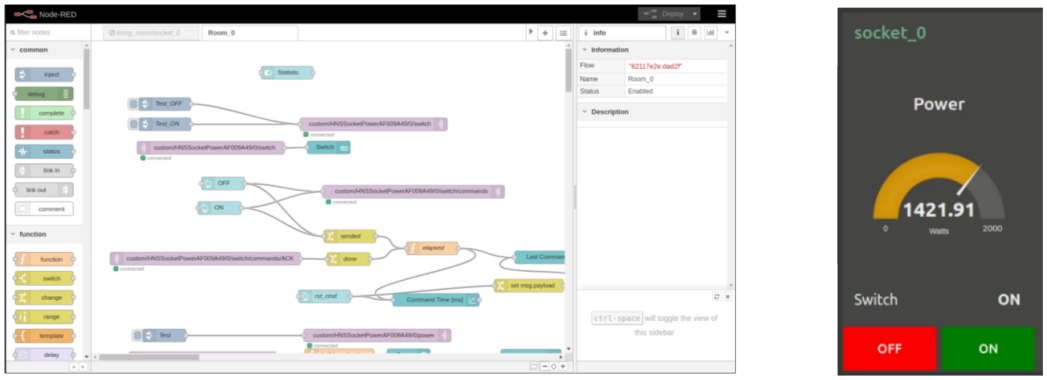
The presence of the HNSHub in the system, between the HNSDevices and the HNSBroker, makes efficient use of the WiFi channel by avoiding the transmission of the same data (i.e. it does not publish the same data of the last update). However, in order to prove the proper functioning of devices, HNSHub sends an update periodically (five seconds) although the data is the same as the last communication. In some cases, when an HNSDevice measures a physical quantity that changes continuously over time, like real power measured by HNSSocket, it is necessary to set a maximum frequency of MQTT data publication to HNSBroker in order to avoid the congestion of the LAN.

3.4.3 HNSBroker

The HNSBroker within the HNS system is used to monitor and manage manually the devices, the code is based on the Node-RED platform.

Node-RED is a flow-based development tool for visual programming, it provides a web browser-based flow editor, which can be used to create JavaScript functions. It supports natively MQTT protocol and allows us to build easily a web user interface by using a specific module called node-red-dashboard.

An example of NodeRED interface is shown in Figure 3.34a. Figure 3.34b shows the user interface of a smart socket developed through node-red-dashboard module.



(a) **(b)**

Figure 3.34. Example of a NodeRED flow (a). HNSSocket user interface (b)

Chapter 4

Proof of Concept

In this chapter is shown a prototype of Home Net System that uses three HNSSocket in order to show the feasibility of load management using this system.

4.1 Components

The prototype consists of the realization of Home Net System by considering only one room which contains three power sockets, each of them is managed by an HNSSocket. The devices tested as HNSHub are a Raspberry Pi Zero W and a Linux PC, during the tests this last was used so that the system could be monitored easily. The HNSBroker is implemented by using a Raspberry Pi 3 and provides a web user interface for local control and management. HNSHub and HNSBroker communicate through a dedicated WiFi network provided by the Linux PC configured as a hotspot by using Create-ap[1] program.

4.1.1 Testbed

The testbed is enclosed in a wood case, on the front panel there are three standard power sockets, each of them is mounted on a support by BTicino Matix series composed of three slots. More specifically, two standard Italian sockets (CEI 23-50 P 17) and one Unel socket (CEI 23-50 P 40) are used. This last typology is usually used to supply appliances such as electric ovens or washing machines. The mounting supports with Italian socket have two free slots, that are used to install the HNSDevice and the relay module, in case of Unel socket, only one slot is available so it is used for the HNSDevice, hence the relay module is installed outside. To avoid instability of the supply voltage of HNSDevices when switching occurs, the relay modules are powered by an external dedicated power supply connected to the network bus. The power supply has been specifically developed using "linear" technology since the common switching power supply introduces noise during power consumption measurement and control of the LED indicator of HNSDevices. The external power supply is based on the LM317 chip, the same integrated circuit used in the power measurement circuit, the circuit schematic is reported in Figure 4.1. The testbed also includes a 16A circuit breaker connected upstream of the power

[1] https://github.com/oblique/create_ap

sockets, overall wiring diagram is shown in Figure 4.2, and Figure 4.3 reports the realization of the testbed by using the wood case.

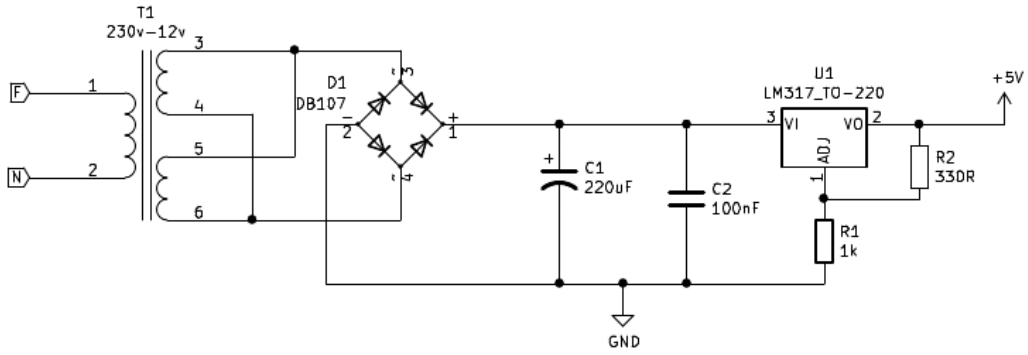


Figure 4.1. Circuit schematic of relay modules power supply

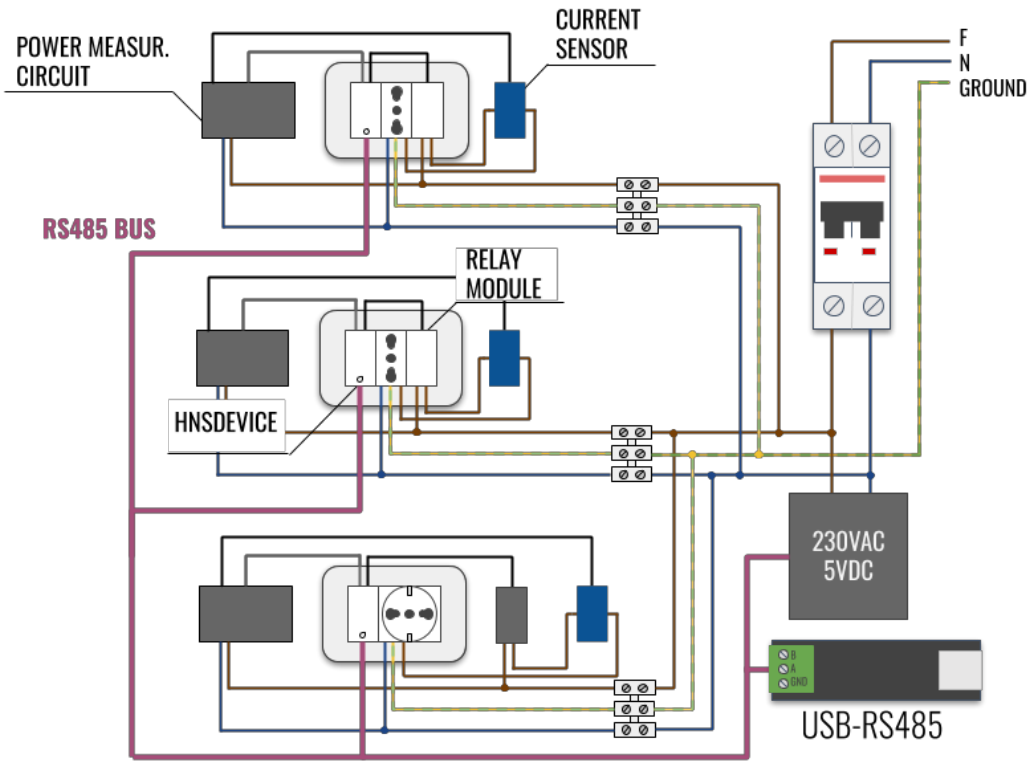


Figure 4.2. Overall wiring diagram of the testbed

(a)

(b)

Figure 4.3. Realization of testbed. Front panel (a), back panel (b)

4.1.2 Web User Interface

The web user interface provided by HNSBroker contains a panel for each power socket that includes information about real-time power consumption, the status of the switch, and two buttons for remote control. The interface also includes information about the response time of the system, in particular, two temporal charts show Refresh Time and Command Time. Refresh Time represents the time between two power consumption updates of a socket sending by HNSHub. Command Time instead, indicates the time required to perform a command, this time is calculated by considering the time between sending by HNSBroker and receiving an acknowledgment from HNSHub that confirms the execution of the command (see Figure 4.5). Figure 4.4 shows the web user interface developed using Node Red.

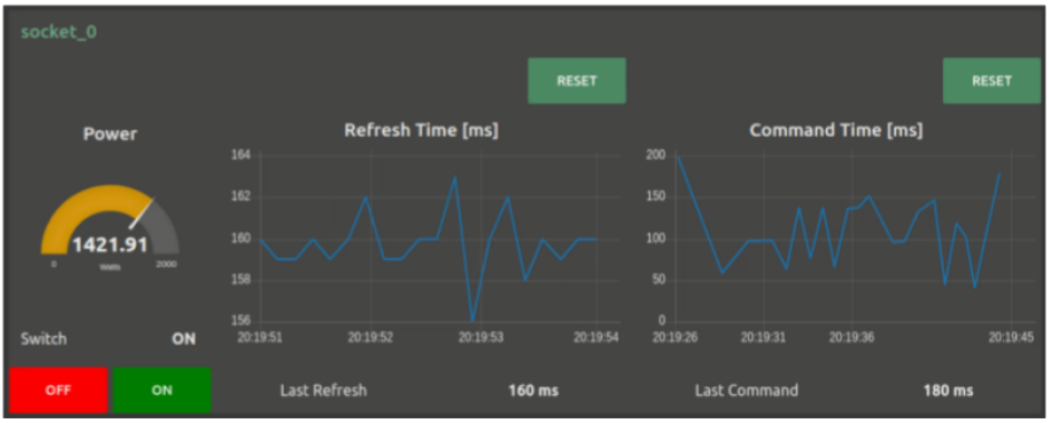


Figure 4.4. Web user interface of an HNSSocket

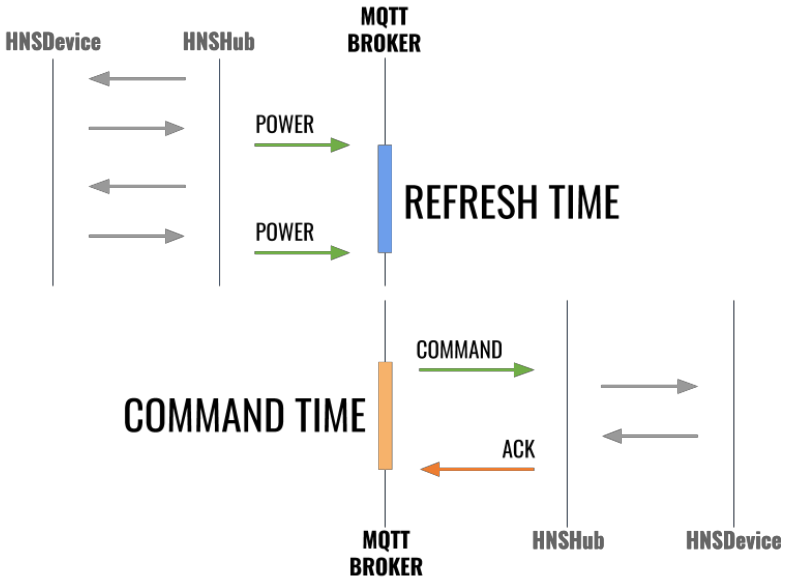


Figure 4.5. Refresh Time and Command Time

Chapter 5

Results

Results of several tests, that have been performed by using the testbed described in chapter 4, are provided in this chapter. The results include both response times of the system managed only by the local controller (HNSBroker) introduced in the previous chapter, and those of the overall system following an overload, described in Introduction, in which the local controller used is the one developed by Alessandro Lamin and the load management is handler by the remote controller developed by Giuseppe Spina.

5.1 Local Test

As previously mentioned in the section 4.1.2, Home Net System response times of interest are Refresh Time and Command Time, these times are collected by HNSBroker.

5.1.1 Refresh Time

Refresh Time corresponds to the time between two power consumption updates from an HNSSocket. This time depends on the variability over time of power consumption, indeed, HNSHub immediately sends an update if only the power measured is different from the previous value communicated, however, HNSHub always sends an update every five seconds even if power consumption remains constant.

In Figure 5.1 is shown measurements of Refresh Time when there is no load attached to the power socket, so the power consumption remains constant to zero. As can be seen, the power update is done regularly with a small variance of frequency.

In Figure 5.2 is instead shown Refresh Time when the power socket is on and a load is connected to. The load used is a hairdryer that allows us to obtain a different value for every power consumption measurement. Refresh Time depends on the time required to perform the measurement (~120 ms) and the period of 150ms required to synchronize HNSHub with HNSSocket, indeed a new measurement can be obtained only every 150 ms (see section 3.4.1.3). The orange line on the figure indicates the mean value, the difference between this mean and period of synchronization (150ms) is an estimate of the impact of WiFi communication between HNSHub and HNSBroker.

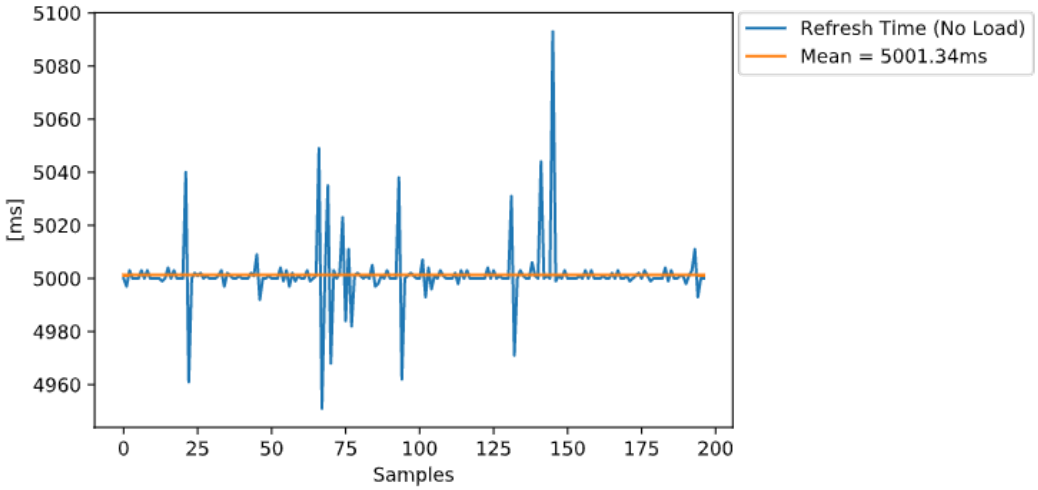


Figure 5.1. Samples of Refresh Time when there is no load attached to the power socket. Default update period set to 5000ms

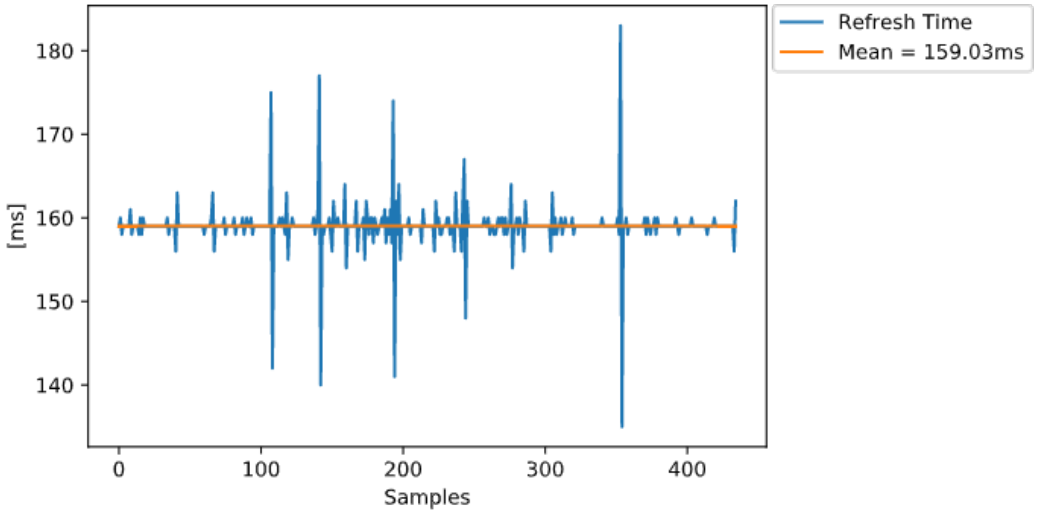


Figure 5.2. Samples of Refresh Time when there is a load connected to the socket

5.1.2 Command Time

Command Time is the time required to perform a command (i.e. toggle relay of an HNSSocket). Also, Command Time depends on the synchronization method, since HNSHub can send a message to HNSDevice only every 150ms. Figure 5.3 shows Command Time collected when the command is sent through the web user interface on HNSBroker. The commands have been sent randomly then time depends heavily on the instant of sending indeed, HNSSocket can be immediately ready to accept a request or it is necessary to wait up to 150ms.

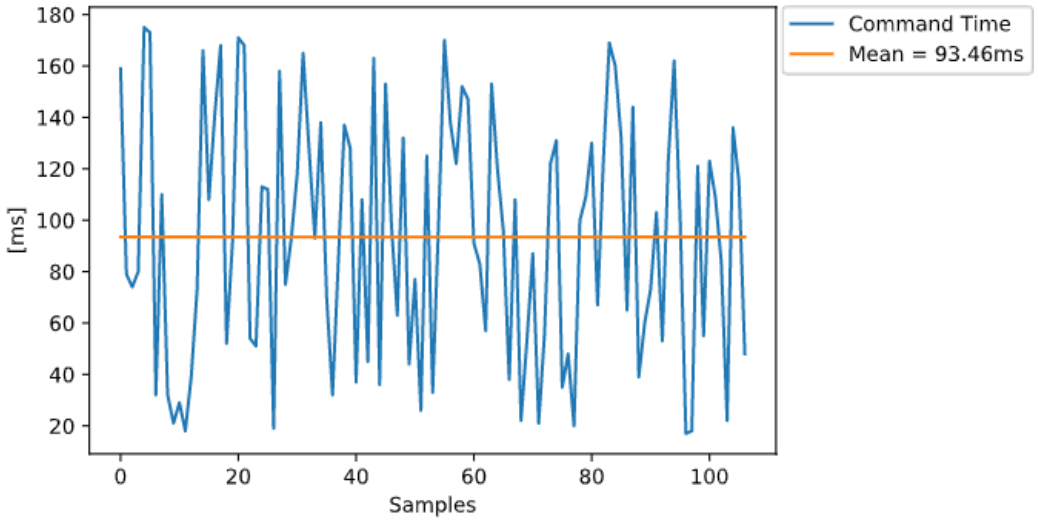


Figure 5.3. Samples of Command Time when the commands are sent by web user interface of HNSBroker

5.2 Remote Control Test

Home Net System can also be used to perform load management by remote control, i.e. using the overall system described in Introduction. To prove that, several tests have been performed in collaboration with Alessandro Lamin and Giuseppe Spina, the following data have been collected by the local controller developed by Alessandro Lamin.

In the final project, HNSHubs and the local controller must be in the same building in order to communicate through a LAN network, however, due to the restrictions imposed following the Covid-19 pandemic, the following tests have been done remotely by connecting HNSHub and the local controller through Internet network.

Two tests have been performed, the first was done by connecting HNSHub to Internet using a FWA Eolo network based on radio links, the second instead, using a 4G TIM network, in both cases, the local controller was connected by FTTC (Fiber To The Cabinet). Figure 5.4 shows the overall connection diagram used during tests.

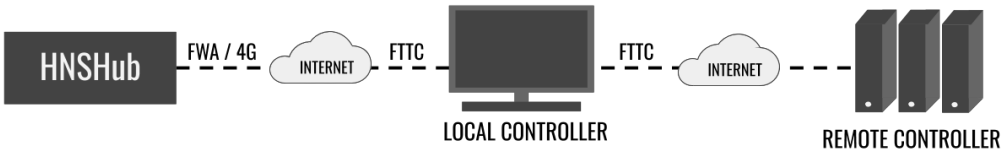


Figure 5.4. Overall connection diagram used during the remote control tests

Times of interest are Disconnection Time and Command Time, this last corresponds to the previous one introduced in the last section. Disconnection Time instead, represents the time required by the remote controller to detect overload and perform the load detachment. This time depends on connection speed and on the processing

time of each node. Figure 5.5 shows Disconnection Time and Command Time definitions.

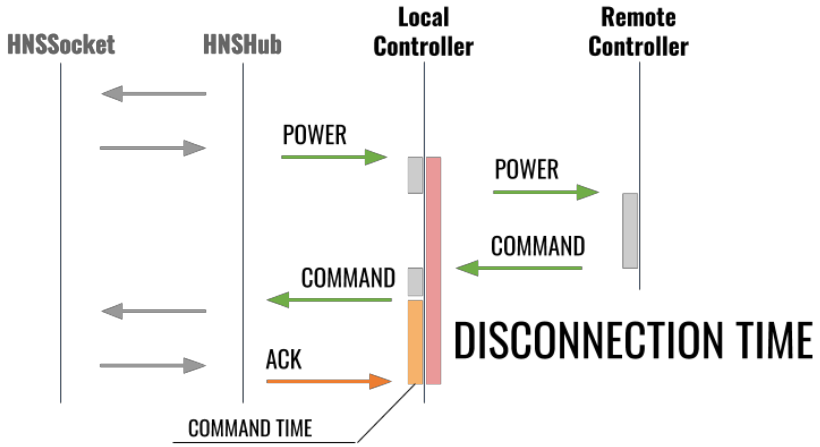


Figure 5.5. Definition of Disconnection Time

5.2.1 FWA Network

In Figure 5.6 are shown samples of Disconnection Time obtained using FWA connection between HNSHub and local controller, in the charts, there is also the mean value (~879.6ms).

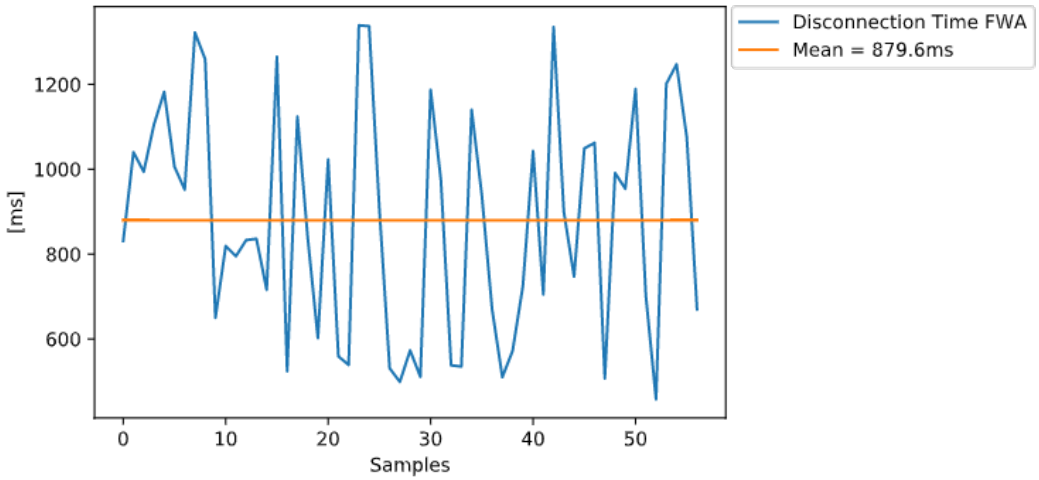


Figure 5.6. Samples of Disconnection Time during remote control by using FWA network for connection of HNSHub and the local controller

Figure 5.7 instead, shows Command Time. It is higher than one measured in local tests (mean=93.46ms) due to the remote connection between HNSHub and the local controller.

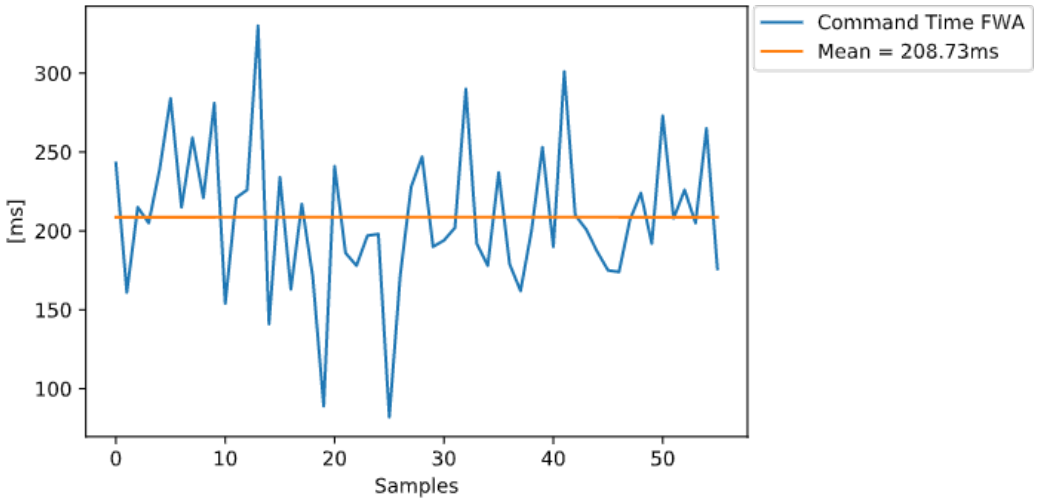


Figure 5.7. Samples of Command Time during remote control by using FWA network for connection of HNSHub and the local controller

5.2.2 4G Network

The same data has been collected by using 4G network. Figure 5.8 shows Disconnection Time, Figure 5.9 instead Command Time. In this case, the mean values are lower than FWA tests since 4G network has a better performance.

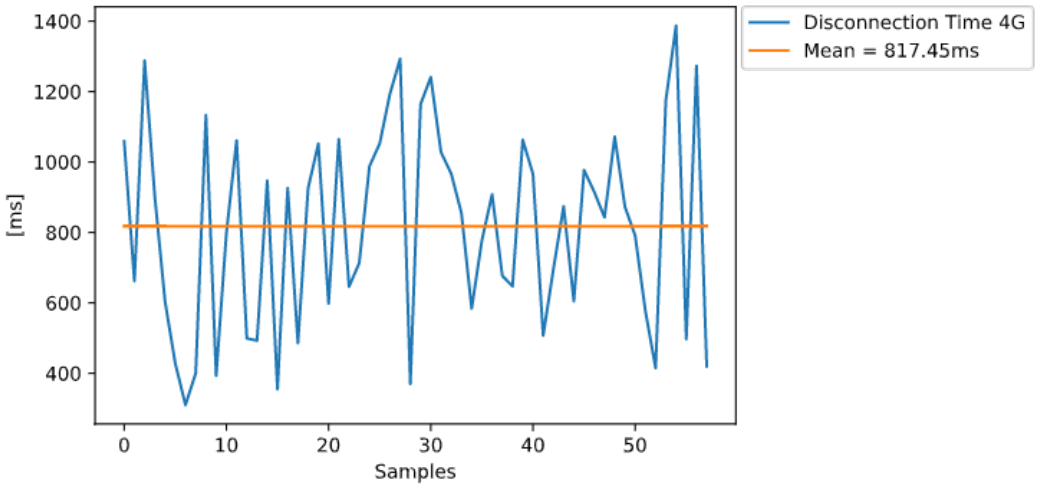


Figure 5.8. Samples of Disconnection Time during remote control by using 4G network for connection of HNSHub and the local controller

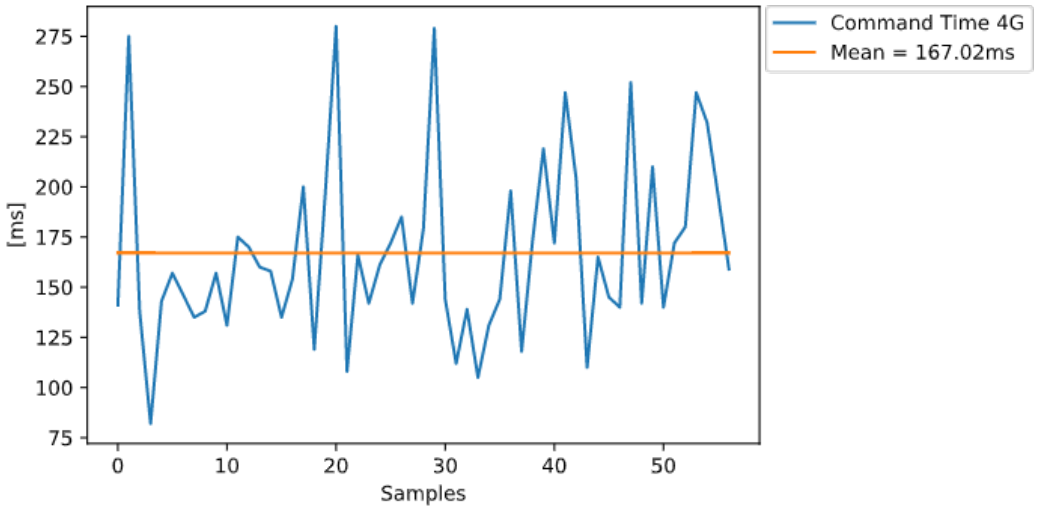
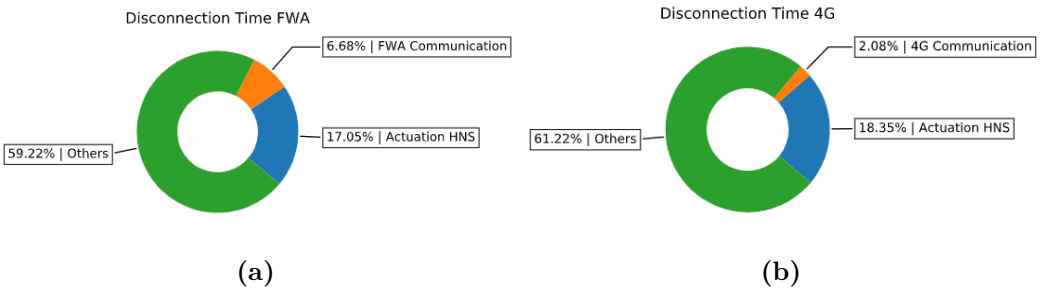


Figure 5.9. Samples of Command Time during remote control by using 4G network for connection of HNSHub and the local controller

5.2.3 Comparison 4G-FWA

In order to figure out how the Internet connection can affect the performance of the system, it is useful to compare the impact of the network used to connect HNSHub and Local Controller, on Disconnection Time.

Figure 5.10a and Figure 5.10b show the components of Disconnection Time respectively in the case of FWA network and 4G network. Actuation HNS is fixed to 150ms and indicates the maximum waiting time for the execution of a command in HNS system, FWA Communication and 4G Communication instead, refer to an estimate of network impact on Command Time. These values have been obtained by subtracting Actuation HNS from Command Time. The component called Others includes the processing time of Remote Controller and impact of the connection between it and Local Controller. It can be seen that in the case of FWA, that has lower performance, the impact on Disconnection Time is higher than 4G.



(a) **(b)**

Figure 5.10. Components of Disconnection Time using FWA network (a), 4G network (b)

Chapter 6

Conclusions

The project developed consists of the design and implementation of a home automation system for energy consumption control that is a compromise between the two typologies of the system currently on the market (industrial oriented and plug-and-play devices based). Home Net System allows us to take advantage of both types by using an efficient wired network in the field level (room or area), and a flexible WiFi network in the control level (i.e. for communication between the local controller and the area controllers). The usage of wired subnetworks permits us to use the shared WiFi channel efficiently, since the useless information, like identical measurement, are suitably filtered. It also allows to considerably cut down costs and energy consumption of field level devices, since the protocol used is basic and does not require the implementation of a heavy stack like TCP/IP that is necessary for plug-and-play WiFi devices.

The main objective of a system for load management is to minimize the overload duration to reduce energy costs and above all to prevent the complete disconnection of the plant by energy meter. During tests, Home Net System showed response time in the order of hundreds of milliseconds, well below the current timing of action of a common electric meter that occurs in the order of minutes/hours. Moreover, through remote control testing, the feasibility of remote management was demonstrated with an overload management times (Disconnection Time) of about 900ms.

The current Disconnection Time can be reduced by connection of the local controller and HNS system in the same LAN of a building, indeed there is a difference of about 74ms between 4G remote Command Time (167ms) and local Command Time (93ms). To further reduce Command Time it is necessary to include an other microcontroller inside HNSSocket in order to have two core, one used to power measurement and the other to communicate with HNSHub, in this way it is not required a synchronization method since HNSSocket is always available to receive a request from HNSHub. The second chip can be added easily by using an external adapter. This improvement can potentially bring the local command time in the order of tens of milliseconds.

Home Net System has been designed to be installed in an existing electrical plant, in particular in order to allow the mount of the devices directly in the wall outlets and make them accessible for manual human control through false poles. However, the current components used by HNSSocket are bulky and they cannot be installed inside a common three slots wall outlet (503E box), in particular, it is necessary to reduce the size of the power measurement circuit by design a new board that uses SMD (Surface Mount Device) components rather than those THT (Through-Hole Technology). There is also a need to replace the current clamp with an equivalent model but with a small size (e.g. WCS1800).

In the future, home automation systems such as HNS could be provided to users directly by energy providers so that they can manage their energy consumption in order to lower the energy supply costs. The system could be sold as a low-price subscription service, like modem in an Internet connection contract, so as to encourage the installation, in this way then, the user would get a home automation system at low cost and the energy provider can make demand side management.

Bibliography

[1] "Explore 5G solutions." [Online]. Available: https://www.5gsolutionsproject.eu/explore/

[2] "5G Solutions: LL2 smart energy." [Online]. Available: https://www.5gsolutionsproject.eu//living-labs/smart-energy/

[3] "Mqtt: The standard for iot messaging." [Online]. Available: https://mqtt.org/

[4] "What is rest." [Online]. Available: https://restfulapi.net/

[5] "Zigbee." [Online]. Available: https://zigbeealliance.org/solution/zigbee/

[6] "Difference between ethernet and industrial ethernet." [Online]. Available: https://www.analog.com/en/technical-articles/what-is-the-difference-between-ethernet-and-industrial-ethernet.html#

[7] "Sensor protocols for iot project." [Online]. Available: https://www.embedded-computing.com/guest-blogs/8-sensor-protocols-for-your-next-iot-project

[8] S. S. Michael, "Introduction to can." [Online]. Available: https://www.allaboutcircuits.com/technical-articles/introduction-to-can-controller-area-network/

[9] "Maxim Integrated MAX485 Datasheet." [Online]. Available: https://datasheets.maximintegrated.com/en/ds/MAX1487-MAX491.pdf

[10] "Worldsemi WS2812B Datasheet." [Online]. Available: https://cdn-shop.adafruit.com/datasheets/WS2812B.pdf

[11] "TTP223 Datasheet." [Online]. Available: https://datasheet.lcsc.com/szlcsc/TTP223-BA6_C80757.pdf

[12] "TTP223 Capacitive Touch Switch." [Online]. Available: https://alltopnotch.co.uk/product/ttp223-mini-red-capacitive-touch-switch-button-self-lock-no-lock-module-arduino/

[13] "CT Sensors - Interfacing with Arduino." [Online]. Available: https://learn.openenergymonitor.org/electricity-monitoring/ct-sensors/interface-with-arduino

[14] “Measuring AC Voltage with an AC to AC power adapter.” [Online]. Available: https://learn.openenergymonitor.org/electricity-monitoring/voltage-sensing/measuring-voltage-with-an-acac-power-adapter

[15] “Texas Instruments LM317 Datasheet.” [Online]. Available: https://www.ti.com/lit/ds/symlink/lm317.pdf

[16] “YHDC SCT-013-030 Datasheet.” [Online]. Available: http://www.ebmstore.it/ebay/download/SCT013-030V.pdf

[17] “CT AC-Power Adapter: Installation and Calibration Theory.” [Online]. Available: https://learn.openenergymonitor.org/electricity-monitoring/ctac/ct-and-ac-power-adaptor-installation-and-calibration-theory?redirected=true

[18] “TACKLIFE CM02A User Manual.” [Online]. Available: https://www.manualslib.com/manual/1540012/Tacklife-Cm02a.html#manual