BABEŞ-BOLYAI UNIVERSITY
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Fuzzy logic-based IoT enable thermostat

B.Sc. Thesis

PhD student: Candale Andrei
Scientific supervisor: Lect. Ph.D. Radu D. Găceanu

2015

Contents

List of Figures

List of Tables

Introduction

The evolution of modern technologies has lead to an increased number of Do-It-Yourself (DIY) possibilities in the field of electronics and microchips; thus it has become easy and affordable to incorporate such technologies in mundane, house-hold objects that can present themselves with some data that may be relevant. This new concept turned into a movement now called the Internet of Things (IoT).

We use the term Internet of Things to refer to the general idea of things, especially everyday objects, that are readable, recognizable, locatable, addressable, and/or controllable via the Internet, wireless LAN, wide-area network, or other means. Everyday objects includes not only the electronic devices we encounter everyday, and not only the products of higher technological development such as vehicles and equipment, but things that we do not ordinarily think of as electronic at all, such as food, clothing, and shelter; materials, arts, and sub-assemblies; commodities and luxury items; landmarks, boundaries, and monuments; and all the miscellany of commerce and culture. [6]

In order for this concept to grow, technologies had to be made available to the general public, meaning that the cost at which they came had to be affordable. Fortunately, the current technological context lead to the development of cheap and powerful components which can now be used to build the infrastructure of the IoT.

With the now given possibility to acquire powerful, yet cheap, means by which one can control and gather data from different contexts, various applications can be developed, like the one described by this paper.

The field of Internet of Things has known a major boom in the recent years, adding to its infrastructure all kinds of objects, appliances etc, giving the possibility to monitor and analyze data from just about anything. The broader affect of this, is the fact that data can be stored, analyzed and made sense of it in order to increase the comfort level and to grasp a better understanding of the environment we are living in.

Starting from this idea, this paper describes a new application that makes uses of the cheap technology we have at hand and the easy access to a world wide data transport infrastructure (the Internet). The result is a device that increases home comfort levels, gathers data that is analyzed and makes of it information that is digestible by a human being, enabling one to better understand and control the context one is living in.

The chosen hardware for the proposed thermostat consist of 3 micro-controllers, temperature and humidity sensors and radio transceivers, all coming to a price comparable to a normal thermostat. The main processing unit is an ESP8266 wireless module. An in depth description of the architecture will be presented in section.

Motivation

The motivation behind this proposal is three-fold: affordability, efficiency and connectivity.

The time we are living in presents itself with rapidly occurring technological breakthroughs that help human kind in setting up for themselves a better, more comfortable and efficient world. The greater problem concerning this is that it does not come cheap. Advanced electronics were not affordable to the general public so the vast majority of people were unable to get access to them, let alone develop things having them as a platform. This situation has changed in the past few years and now we have easy access to such devices enabling the contribution of everybody to the Internet of Things.

The recent years have marked an amazing rate at which new technologies are discovered and at which new technological processes of manufacturing are getting more efficient and more affordable. This created a self sustaining environment in that the newly invented electronics and machinery lead to more efficient development methods which in turn provided a more feature packed set of tools that is driving the observed rapid evolution.

Because of the above described advance, it is now possible to get access to powerful devices that are relatively cheap thus giving the opportunity to everybody to build gadgets that can be incorporated in the infrastructure of the Internet if Things.

Having access to diverse pieces of technology lets anybody create mechanisms through which data is gathered from the environment and shared on a wide network of other devices where is can be observed. The main idea behind the Internet of Things is having a large number of gadgets that collect, store and analyze data about everything and anything with the purpose of grasping a better understanding about them. The Internet of Things enables the world to research itself and find patterns that were otherwise invisible, thus leading to a greater understanding of the context we are living in, which in turn lets us control that context in a more efficient manner.

All these connected devices mean that data can be shared and used in a collaborative manner and the possibility to control processes from afar arises. For the application in discussion, this fact enables one to monitor and control one's house temperature setting from anywhere thus increasing the comfort level. For example, going away from home, one would turn the temperature down but when coming back, it is desirable to have the house already at a comfortable temperature level. This can be achieved by being connected to your home thermostat.

Chapter 1

Architecture

This chapter will describe the architecture of the thermostat: the components that make it up and how all of them are interacting with each other.

1.1 Overview

The thermostat architecture is made up of at least two modules: a main module, that is the central part of the setup and a second module. From now on we will refer to the main module as the central unit and the second module as reporter. When referring to the thermostat, the whole setup is considered, central unit and reporters.

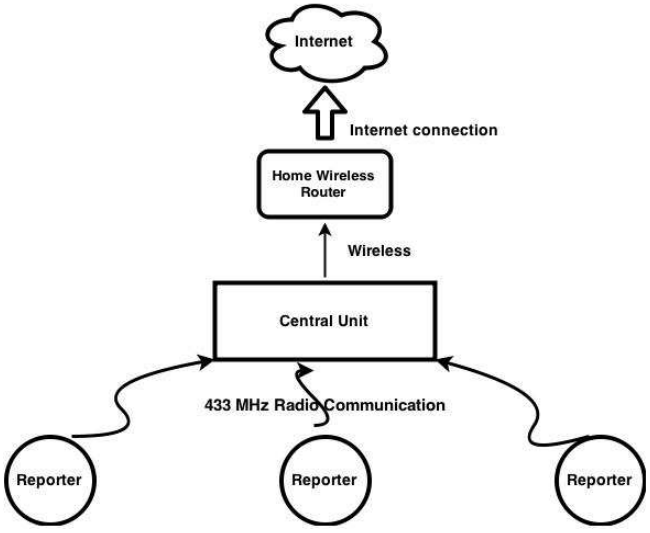

Figure 1.1: Agent based model architecture.

The thermostat setup follows an agent based model having one or multiple agents that are reactive to the environment, and that report their findings to a centralized module. In accordance to the agent based model, we have the central unit of the thermostat which acts as a central agent that receives, stores and analyses the data and one or multiple reporters that read temperature and humidity values from the environment they are in and send it to the central unit.

1.1.1 Central unit

The central unit is the module where the data is centralized, analyzed and where the actions of turning the heating on or off happen. It is home to an Attiny85 microcontroller, a 433 MHz radio receiver, a relay, an ESP8266 and a LF33CV voltage regulator. These components will be described in-depth in the next sections.

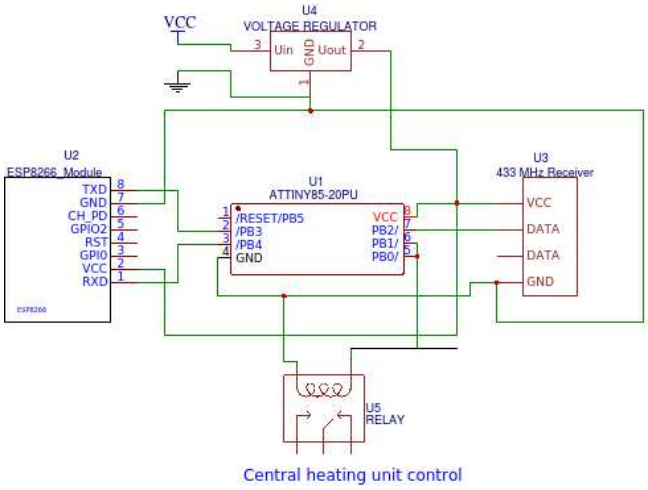

Figure 1.2: Central unit electrical diagram.

Most heating system in Northern Europe are based on a gas central heating unit that has two states: on or off. The control mechanism they employ is a standard one: there are two wires that need to make contact in order for the heating to be set to on. When the wires no longer make contact, the heating will be off. The two wires that provide the mean to control the central heating will be connected to the relay that's on board of the thermostat's central unit so it will be able to gain control over it.

The central unit of the thermostat fits on a small PCB and have a size of 7 by three point five centimeters because of it's low number of parts that come in a small package. It's size can be furthered reduced by using SMD components and by improving their layout.

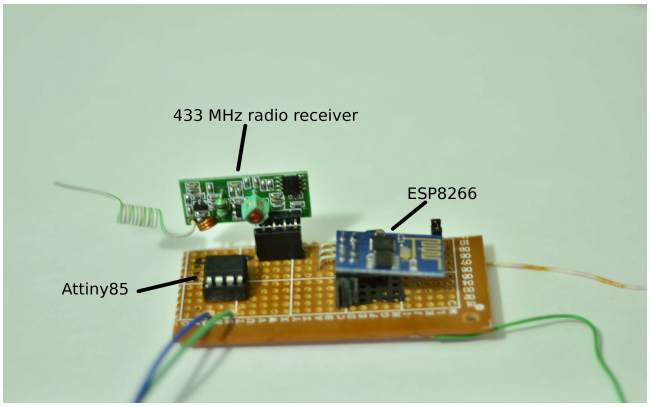

Figure 1.3: Central unit prototype on PCB.

The whole assembly of components can be powered with a voltage that ranges from 6 to 40 volts. That is due to on board LF33CV voltage regulator that limits the the number volts to 3.3. This voltage was chosen because the ESP8266 works at a maximum rating of 3.3 volts and the other parts were selected so they would also run at this rating. Given it's minimum and

maximum ratings, the central unit may be powered by batteries or an adapter that converts 110/220 volts alternative current to 6-40 volts direct current.

1.1.2 Reporter

The second module of the thermostat, the reporter, it's what collects that data from the environment and sends it to central unit. The reporter is an independent module that consists of an Attiny85 (See section 1.4) a 433 MHz radio transmitter, a DHT22 temperature and humidity sensors and an L7805CV voltage regulator.

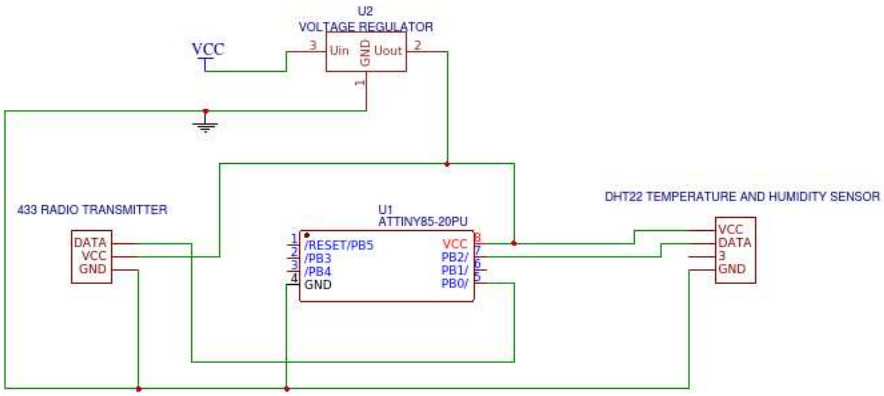

Figure 1.4: Central unit prototype on PCB.

The reporter is made up of components that could run on 3.3 volts but on them, the temperature and humidity sensors needs at least 5 volts for normal functioning. Also, the 433 MHz radio transmitter may work in a range of 5 to 12 volts. The greater the voltage, the greater the signal's strength will be, thus covering a greater area. The Attiny85, as stated above can work on 3.3 volts but given the ratings that the other components require, on board of the reporter there is an L7805CV voltage regulator that limits allow no more than 5 volts to pass. Given the above specifications, the following configuration was chosen: the reporter is powered by a 9 volt battery. The 433 MHz radio transmitter is powered directly from the battery while the other components (Attiny85 and DHT22) will be powered by 5 volts that is supplied through the voltage regulator.

The 433 MHz radio transmitter is well equipped to provide a range that is more than necessary for medium sized houses and it can even be placed outside, given that it is weather proof. The 9 volt battery that is powering the reporter give the transmitter enough power to have a range of about 300 meters with no obstacles in the way [18].

As stated above, the components of the reporter are in a small number thus giving the possibility to put everything in a small package. The dimensions of the PCB which holds the parts is five by four centimeters but it can be greatly reduced if SMD components are used and a better layout is chosen.

Because the communication between the reporter and the central unit is made by radio channels, one can place such a reporter outside giving chance to a new range of possibilities including having the outside temperature as factor by which the inside temperature can be tuned.

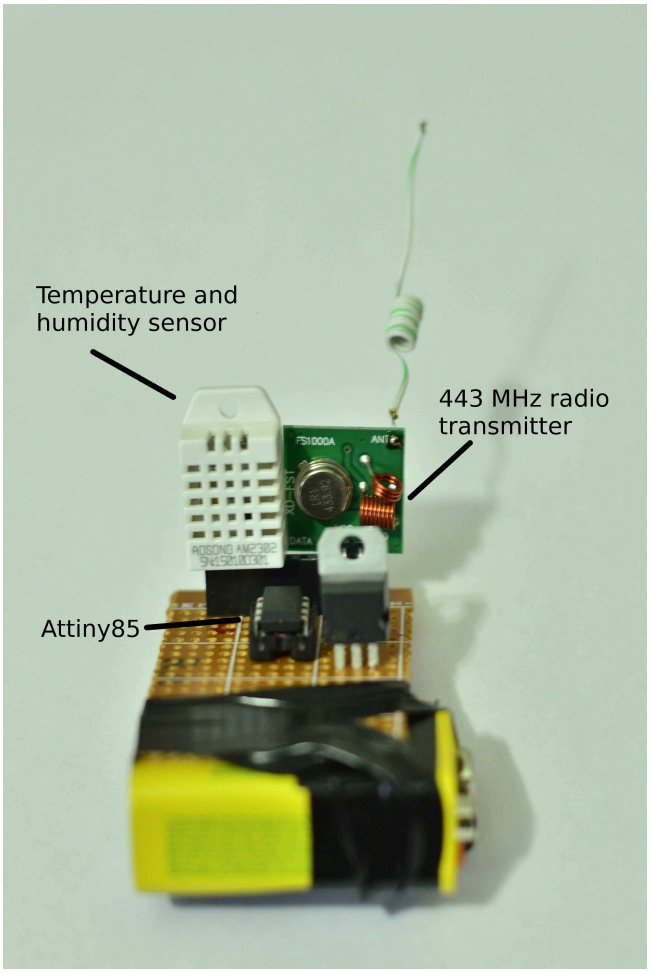

Figure 1.5: Reporter prototype on PCB.

1.2 Sensors

The thermostat in discussion uses a single sensors that reports temperature and humidity. The DHT22 is temperature compensated and calibrated in accurate calibration chamber and the calibration-coefficient is saved in OTP memory; when the sensor is detecting, it will cite coefficient from memory [7].

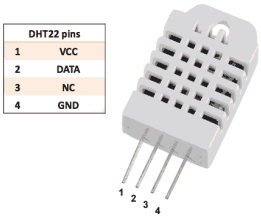

Figure 1.6: DHT22 Temperature and humidity sensor.

The DHT22 is a basic, low power sensors but precise enough to be reliable in the conditions the thermostat is designed for. The precision with which it reports the data is of 0.1 for both temperature an humidity having a rage of -40 C to 80 C for temperature while for humidity it has a range that starts from 0 and ends at 100, measured in percentage, representing the relative humidity of the environment.

The interval at which data can be collected from this sensor is of about two seconds, which

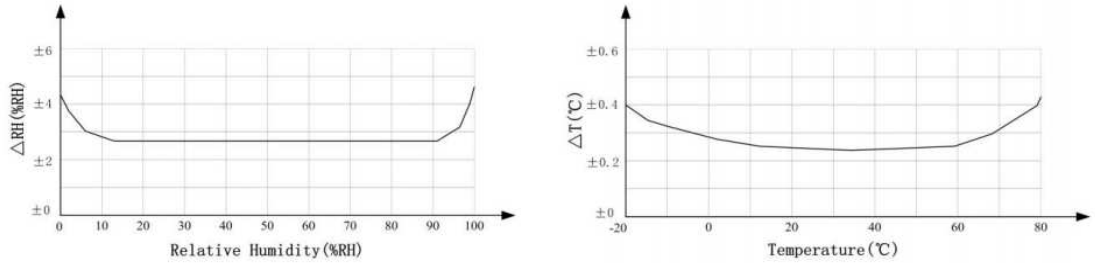

Figure 1.7: DHT22 Temperature and humidity sensor errors.

is more than enough for the purposes of this application. It uses a single-bus communication protocol that only needs one data-line. This represents an advantage since the number of I/O pins is small on the microcontroller responsible for reading the data.

1.3 Radio communication

The communication protocol and hardware was chosen based on two factors: low power and affordability. Given the fact that the agent-based architecture that was chosen for this applications has agents that only report data but do not need to receive any, the solution chosen comes in two packages: a transmitter and a receiver.

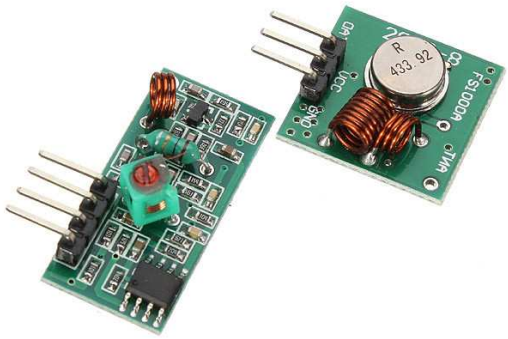

Figure 1.8: MX-05CV Radio transmitter and receiver. Image source [15]

The radio hardware is the MX-05CV receiver/transmitter pair 1.8 that works on 433 MHz frequency. It is a low power, cheap solution that is intended for application where data needs to be carried over short distances. The radio communication uses AM (Amplitude modulation) modulation technique, meaning that the amplitude of the radio signal is varied in order the represent the data one is transmitting.

The radio receiver works on 5 volts, direct current and takes up about 4 milliamperes of current when on stand-by which makes it suitable for low energy footprint application such as the one we're discussing. The receiver is fitted with a seventeen centimeter antenna that boosts its signal capturing capabilities.

The radio transmitter works in the range of 3.5 to 12 volts. The signal range varies proportionally to the voltage applied to the module having a transmitting power is of 10 milliwatts

and a maximum data transfer rate of 4 kilobytes per second.

The cost of such a pair of radio devices comes extremely cheap and in combination with the above mentioned specifications, it makes it the perfect communication solution.

1.4 Attiny85

The ATtiny25/45/85 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATtiny25/45/85 achieves throughputs approaching 1 MIPS (million instructions per second) per MHz allowing the system designer to optimize power consumption versus processing speed [9].

Figure 1.9: Attiny85 AVR microcontroller. Image source [10]

The Attiny85 is small microcontroller that has 8 kilobytes of In-System Programmable Flash, 512 bytes of EEPROM (non-volatile memory that holds across power-ups) and 256 bytes of SRAM and it work at a frequency of 8 MHz. It has 6 general purpose I/O lines that may be used for either output or input.

Given its low power consumption and the relatively high processing power, the Attiny85 can perform non-trivial tasks like generating pulses to drive a radio transmitter to send data, collect data from a radio receiver and read various information from sensors. The Attiny85 comes in a small package, it requires no external components like a crystal clock, having one internally that runs on 8 MHz, which makes it suitable for the central unit of the thermostat as well for the reporter due to its small PCB footprint.

The microcontroller comes with a sleep mode feature that enables it to go in a state where the power consumption is negligible. It features three sleeping modes, each of them disabling a further layer of its architecture thus lowering the power draw to a point where is almost non-existing. This is a nice feature that can be used in the thermostat's reporter since it runs on battery. The way if functions is that at power-up, after everything has been initialized in the memory, it tries and reads the data from the temperature and humidity sensors. After completing this task, it sends the data over radio to the central unit an it goes to deep-sleep thus preserving and prolonging the life of the battery by a significant amount. Tests [17] have shown that with normal operation on 8 MHz clock speed, the Attiny85 consumes about 10 to 12 milliamperes. Considering that a CR2032 [watch] battery contained about 200 milliamperes hour, that gives us ($200mAh/20mA$) ten hours of runtime. With the sleeping mode enabled, the microcontroller draws about 1 microamp of current, thus being able to reach four thousand hours of runtime. That is an amazing figure for the battery in discussion.

The Attiny85 can be programmed with the well known platform, Arduino by using the Arduino IDE [4]. As in all Arduino boards, the program for such a microconctroller has the following structure: a setup and a loop function. The setup function is called only once when the Atinny85 starts up and here is the place for code that initializes the runtime environment. After the function setup finishes, the function loop is called repeatedly for as long as the microcontroller has power. This pattern of programming follows the idea that your code has to do something periodically, like checking the data on some sensors or checking for the pressing of some buttons.

In the presented thermostat, the Attiny85 is used in both the central unit and reporter. In the central unit the, the microcontroller has the responsibility to read the data from the radio receiver, send that data to the ESP8266 (where data processing happens) and control the relay at the request of the ESP8266. The receptor has only two tasks, reading the temperature and humidity from the sensors and sending it via radio.

In order to accomplish all of this, the Attiny85 uses several libraries. For reading the data that the sensor sends, an Arduino secific library was used that implements the communication protocol that the DHT22 uses [1]. This library is easily can be easily ported so it can work with our setup.

For the radio communication, the Manchester encoding library [14] was used. It provides a reliable implementation of an encoding protocol that works regardless of the frequency of the microcontroller it operates on.

On the central unit of the thermostat, the Attiny85 is constantly communicating with the ESP8266 data that is receiving from the reporters and itself is receiving data from the ESP8266 regarding the state of the heating. Unlike other microcontrollers, as the one that lies on the Arduino board (Atmel Atmega328P-PU [8]), the Attiny85 does not have a hardware implementation of serial communication. In order to bring serial communication to this microcontroller, a SoftwareSerial library [5]. It enables our microchip to talk to the main processing module that lies on the central unit.

1.5 ESP8266

The ESP826 is a small, 32 bit wireless enabled microcontroller build onto ARM architecture. It offers a complete and self-contained Wi-Fi networking solution, allowing it to either host the application or to offload all Wi-Fi networking functions from another application processor [11]. The chip comes packed with 200 kB of ROM, 32 kB ofSRAM and 80 kB of DRAM which is more than enough to give it the role of the main processing unit of the thermostat. The module also provides the wireless connectivity through which it can be accessed and the means by which data can be published on a remote server. The chip supports 802.11 b,g and n standards and has TCP and UDP stack implemented on it.

Since its release, the open-source community has developed a number of programming languages support, including Lua [16] and JavaScript [13] but, while testing, it has been found that currently the C SDK that the producer has released is the most stable and it is the one used.

The SDK provided by Espressif, the maker of the ESP8266, exposes a low-level C interface to manipulate the various features of the wireless chip [12]. The way the code is structured in the case of this chip is similar to the one from the Attiny85. There is user_init function where the system is initialized and the environment prepared for runtime and the user can create a

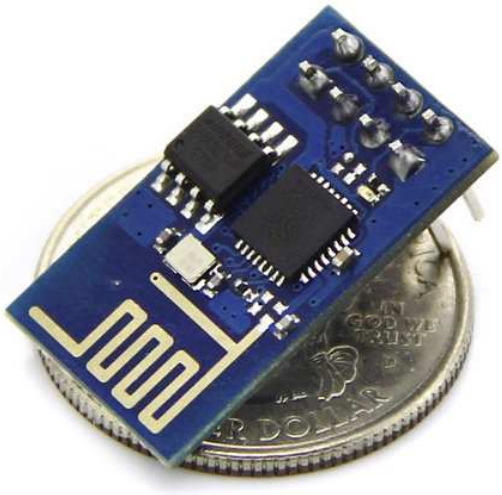

Figure 1.10: ESP8266 Wireless module. Image source [3]

loop for himself where the periodic tasks may take place. This is done my creating a system task that is fired in user_init and then it sustains itself by triggering the same task over and over.

```
static void ICACHE_FLASH_ATTR procTask(os_event_t *events)
{
    system_os_post(procTaskPrio, 0, 0 );
    printf( "Idle Task\n" );
}

void user_init(void)
{
    system_os_task(procTask, procTaskPrio, procTaskQueue,
                   procTaskQueueLen);
    system_os_post(procTaskPrio, 0, 0 );
}
```

In the following subsections the basic API of the ESP8266 wireless module will be described.

1.5.1 Operation Mode

As stated before, the ESP8266 wireless module has the sufficient features to either run a complete application on itself or to offload all the communication work of another microcontroller thus the chip can be configured in three ways: in STATION mode, SOFTAP (soft access point) mode or both STATION+SOFTAP mode. This setting can be modified by the API function $wifi\_set\_opmode(uint8 opmode)$ where, for STATION, opmode has the value 0x01, for SOFTAP 0x02 and for STATION+SOFTAP 0x03.

The STATION mode allows the module to connect itself to an existing access point (AP), host a webserver that can respond to requests which are made through the connected AP and make requests of its own. When using the chip in station mode, one can use its mDNS feature to publish the IP address that the router to which it connected to assigned to it. This is a nice feature that enables the user to acquire easy access to the device by finding it on the network he is already used to by a domain name.

```
// mDSN works only in STATION mode
wifi_set_opmode(0x1);
// free the informtion related to SOFTAP
wifi_softap_free_station_info();

// allocate memory for mDNS information
struct mdns_info *info = (struct mdns_info *)os_zalloc(
    sizeof(struct mdns_info));

// set parameters accordingly
info->host_name = "thermostat";
info->ipAddr = station_info.ip.addr;
info->server_name = MDNS_SERVER_NAME;
info->server_port = 80;
os_strcpy(info->txt_data, "version = now");

// initialize, enable and register the device
espconn_mdns_init(info);
espconn_mdns_enable();
espconn_mdns_server_register();
```

Now, the webserver that is running on the ESP8266 will be available from within the network to which it connected, by accessing the URL htpp://thermostat.local. This offers a great deal of flexibility because one does not have to change APs just because one wants to get access to the thermostat

By enabling the SOFTAP mode, the ESP8266 is able to create an AP to which anybody can connect. The AP can be secured using the protocols WPA / WPA2 or WEP. A webserver can listen to requests made from this mode as well.

```
struct softap_config ap_conf;
wifi_softap_get_config(&ap_conf);

os_memcpy(ap_conf.ssid, AP_SSID, os_strlen(AP_SSID));
os_memcpy(ap_conf.password, AP_PASS, os_strlen(AP_PASS));
ap_conf.authmode = AUTH_WPA_WPA2_PSK;
// set the configuration
wifi_softap_set_config(&ap_conf);

// start DHCP server
wifi_softap_dhcps_start();
```

In STATION+SOFTAP mode the wireless module can receive request through the access point (SOFTAP) it created and at the same time it is able to sent requests by also being in STATION mode. The module will no longer be accessible, from the network it connected to, through a domain name but only through its designated IP address.

1.5.2 Running a webserver

The ESP8266 wireless module API comes with a basic implementation of a webserver. The way it works is that first one sets the type of connection he wants (TCP, UDP) and the port on which the server will listen and then a callback is registered that is called whenever a connection is received. The said callback is responsible for setting other callbacks: for receive data event and for disconnect event.

```
void http_recieve(void *arg, char* data, unsigned short length);
void http_disconnect(void *arg);

void ICACHE_RAM_ATTR
webserver_listen(void *arg) {
    struct espconn *conn = (struct espconn *)arg;

    espconn_regist_recvcb(conn, http_recieve);
    espconn_regist_disconcb(conn, http_disconnect);
}

void setup_tcp_listener()
{
    // wipe clean memory
    os_memset(server, 0, sizeof(struct espconn));

    LOCAL esp_tcp esptcp;

    server.type = ESPCONN_TCP;
    server.proto.tcp->local_port = WIFI_SERVER_LISTEN_PORT;
    // register on connect callback`
    espconn_regist_connectcb(&server, webserver_listen);

    // create the TCP server
    espconn_accept(&server);
    // register connection timeout in seconds
    espconn_regist_time(&server, 15, 0);
}
```

The server will accept any TCP connection that comes on the specified port wither from the AP it setup or from the AP it connected to and it will call the corresponding callbacks that were registered.

1.5.3 Webserver wrapper

The server implementation provided by the API is a basic one and has functionality limited to fetching the request body and its size. In order facilitate the processing of requests, a wrapper around the API's webserver was developed that makes it easier to get and interpret a request's body. The wrapper has the following API:

```
/*
 * Returns the method of a HTTP request given its request body
*/
uint8 get_http_method(char* request);

/*
 * Returns the full URL of a request given it request body
*/
char* get_request_url(char* request);

/*
 * Returns a url that has not GET parameters in it
*/
char* get_url_without_params(char* url);

/*
 * Registers a calllback function that is to be called
 * when a request comes on the URL specified by the
 * first parameter.
 * process_request_func:
 *  typedef void (* process_request_func)(
 *      struct espconn * conn, char* data,
 *      unsigned short data_length);
/*
void register_url(char* url, process_request_func func);

/*
 * Returns the value of the request parameter whose key is passed as
 * the function parameter for the given method.
*/
char* get_param(char* name, uint8 method);
```

1.5.4 Making HTTP requests

The API that is provided for the ESP8266 comes with functions that perform HTTP requests, given the IP of a server, and that make DNS queries in order to resolve the IP of a domain name. The presented thermostat uses a library [2] that is written around the mentioned API to make it easier to perform requests and not have to deal with the rather complicated native process. The way a request is made is by simply calling a function that has as parameters the URL to which the request is made and a callback that will process the response.

While testing this library, it has been found that the DNS resolver cannot get the IP of the target server so modifications were brought to the library such that one could use an IP to make a request. In this configuration both the IP address of the server and the URL that would normally follow the domain name have to be supplied.

```
ip_addr_t addr;
IP4_ADDR(&addr, 184, 106, 153, 149);
http_get(url, &addr, http_callback_example);
```

1.6 Putting everything together

Every component that we have described so far plays a crucial role in the functioning of the thermostat and they are all necessary for this purpose. All the pieces of hardware that were chosen were done so considering most of all affordability, processing power and connectivity.

In the following, this section will describe how all of the above parts are combined together to make up a functioning thermostat and how the user is interacting with it.

1.6.1 Thermostat high level components interaction

As stated before, the thermostat is composed of a central unit and one or more reporters. The central unit's purpose is to collect data from reporters, process the data and make a decision about whether or not to turn the heating on or off and it is also responsible for running the necessary software that provides the user interaction with the system.

There are two types of connections that need to be alive for the thermostat be operating as intended. First of all, the radio connection through which data is send from the reporters to the central unit. This is a key aspect that the system cannot do without. The second connection, though not mandatory, is with the home wireless router. The central unit needs to connect to an access point through which the Internet is accessible so that data can be sent to remote servers. Even without having an Internet connection, if the central unit is linked with an AP, the access to it is much more easy to achieve given the mDNS feature that the ESP8266 has 1.5.1. As implied, these are not necessary conditions for the thermostat to work but having them present greatly increases the user experience.

1.6.2 User interaction

After first starting up the thermostat, the user needs to set the SSID and password of his/shes home wireless router in order for the ESP8266 to connect to it. In order to do that one must connect to the AP that the central unit has created an go to the address 192.168.4.1/$wifi\_setup$ to set the credentials. After doing so, the thermostat will connect to the router, the AP it created will disappear and the ESP8266 will be accessible through the network to which it connected by going to the URL http://thermostat.local.

The thermostat provides means by which one can monitor the temperature that the reporters are recording and set a desired temperature. This is done by going the URL http://thermostat.local/control
A webpage will appear where one can configure the temperature that one desires.

Chapter 2

Conclusions

Conclusions here...

Bibliography

[1] Dht-sensor-library, https://github.com/adafruit/dht-sensor-library.

[2] Esp8266 esphttpcliennt, https://github.com/caerbannog/esphttpclient.

[3] Esp8266 wifi serial module, http://is.gd/jiv6jk.

[4] How to program an attiny85, http://highlowtech.org/?p=1695.

[5] Softwareserial library, http://www.arduino.cc/en/reference/softwareserial.

[6] Six technologies with potential impacts on us interests out to 2025. In *Disruptive Civil Technologies*. National Intelligence Council, 2008.

[7] AOSONG, http://akizukidenshi.com/download/ds/aosong/AM2302.pdf. *DHT22 Temperature and Humidity sensor*.

[8] Atmel, http://is.gd/ef7Fyq. *Atmega328P-PU microcontroller*.

[9] Atmel, http://is.gd/wV8aIA. *Attiny85 microcontroller*.

[10] Attiny85. http://thomastesla.com/mcu_class/.

[11] ESP8266. http://espressif.com/en/products/esp8266/.

[12] Espressif, http://is.gd/H0otjK. *ESP8266 Wireless module API*.

[13] Espruino. http://www.espruino.com/.

[14] Manchester. http://mchr3k.github.io/arduino-libs-manchester/.

[15] MX-05CV. http://is.gd/eubb0h.

[16] NodeMCU. http://www.nodemcu.com/.

[17] Nathan Seidle. Low power attiny85, https://learn.sparkfun.com/tutorials/h2ohno/low-power-attiny.

[18] Texas Instruments, http://www.ti.com/lit/ds/swrs037b/swrs037b.pdf. *Low Power Sub-1 GHz RF Transmitter*.