Technical University of Munich
Department of Civil, Geo and Environmental Engineering
Chair of Computational Modeling and Simulation
Softwarelab Project

ESP32 WiFi Sniffer Manual

Students:

Hua-Ming **Huang**
huaming.huang.tw@gmail.com

András László **Aninger**
aninger.andras@gmail.com

Supervisor:

Felix **Eickeler**

January 2021

Contents

1 Introduction

1.1 The motivation for the application

The idea for the implemented application detailed in this document came from the need for validation of pedestrian movement simulations. The basic assumption was that by tracking the position of nearby Wi-Fi enabled devices one has a good estimate about how the people using them are moving around in an area.

1.2 Other possible use cases

The validation for simulations case aside there are two other possible scenarios where this or similar applications could be used.

One could utilize the large amount of data that potentially can be gathered to enhance the simulations instead of only validating them. This approach potentially would need some sort of learning algorithm to be included into the computational model which could then update itself based on the new evidence found via measurement. This application case, however, would need highly accurate position data for the learning process to be effective.

Another use case would be to apply the tool in a live manner for crowd control at large events. The organizers then could follow an estimation on a map how the crowd moves around in the area and intervene in time if parts of the venue start to get dangerously overcrowded. For this application less accurate position estimations would be already sufficient but the calculation and transmission of the data would have to happen really fast.

1.3 Hardware used

For the realization of the project Wi-Fi and LoRa capable microcontrollers with an ESP32 dual core chip were used. These were mostly devices from the company LilyGo and of the type of TTGO ESP32-Paxcounter and TTGO Meshtastic T-Beam.

The Paxcounter version has an on-board SD card slot therefore all the files needed by the application are stored on an external SD card and read in from there. In case of the T-beam only internal flash memory (so-called Serial Peripheral Interface Flash File System (SPIFFS)) is available, therefore this was used. In the code there are instructions stated how to upload files to the internal flash memory.

1.4 Software used

For the programming of the application and the microcontroller units the Visual Studio Code editor with the PlatformIO extension was used.

1.5 A word on accuracy

The position determination algorithm in this implementation is signal strength (RSSI) based [2] and is good enough for determining overall trends in the movement of pedestrians. During testing of the application (with 4 measuring devices) it was found that the estimated positions are within the 1 meter radius of the actual positions. Given how strongly measured RSSI values fluctuate even at the exact same position, this is not unusually bad and with increasing number of microcontrollers measuring the signal strength the accuracy would increase. One could also increase accuracy by using more sophisticated algorithms (F.Shang et al. [1]) like the one we used (M.BOTTA, M.SIMEK [2]) and will address here later on.

One most consider, however, that the use case does not really demand that the application pinpoints the location with high accuracy. It is only of interest, how larger crowds tend to move, not how each individual exactly walks around.

2 Workflow of the application

2.1 Collecting Wi-Fi signal data

For the position determination to work at least three microcontroller units collecting data are needed. These so-called slave units should be positioned in a way that they have a clear line of sight to the area in question. Each individual slave listens to Wi-Fi data packets sent by the devices within reach and collects the MAC addresses and RSSI values contained within. The content of the packet is not recorded.

2.2 Slave devices

To declare a device as slave one must define it as such in the JSON config file on the SD card. One core of the microcontroller uses Wi-Fi to obtain the needed information (MAC, RSSI) from the data packets sent by nearby devices. This is then placed on a queue (FIFO type buffer) where the other core has access to it. This core then logs the data to the SD card (*.dat file, binary to increase write/read speed) and transmits it to the master device using the LoRa antenna of the microcontroller.

2.3 LoRa

For communication and sending data between the devices the application uses the LoRa network protocol. The main purpose for using it is the desire for high level of parallelization. Since on the microcontrollers used only one of the cores can access the Wi-Fi hardware at the same time, using Wi-Fi for communication with only one of the cores and LoRa with the other eliminates waiting times for hardware access rights.

2.4 Master device

One of the microcontrollers in the system must be set as the master device, using the config JSON file. One of the cores of this device will be responsible for receiving the data packets acquired and sent by the slave devices. Upon receiving the different packets this core matches up the recordings about the same device and creates a combined data packet which gets put on a queue. From there the other core receives it and logs it into a *.csv file on the SD card. This core also hosts an asynchronous webserver which ensures a browser-based access to the system.

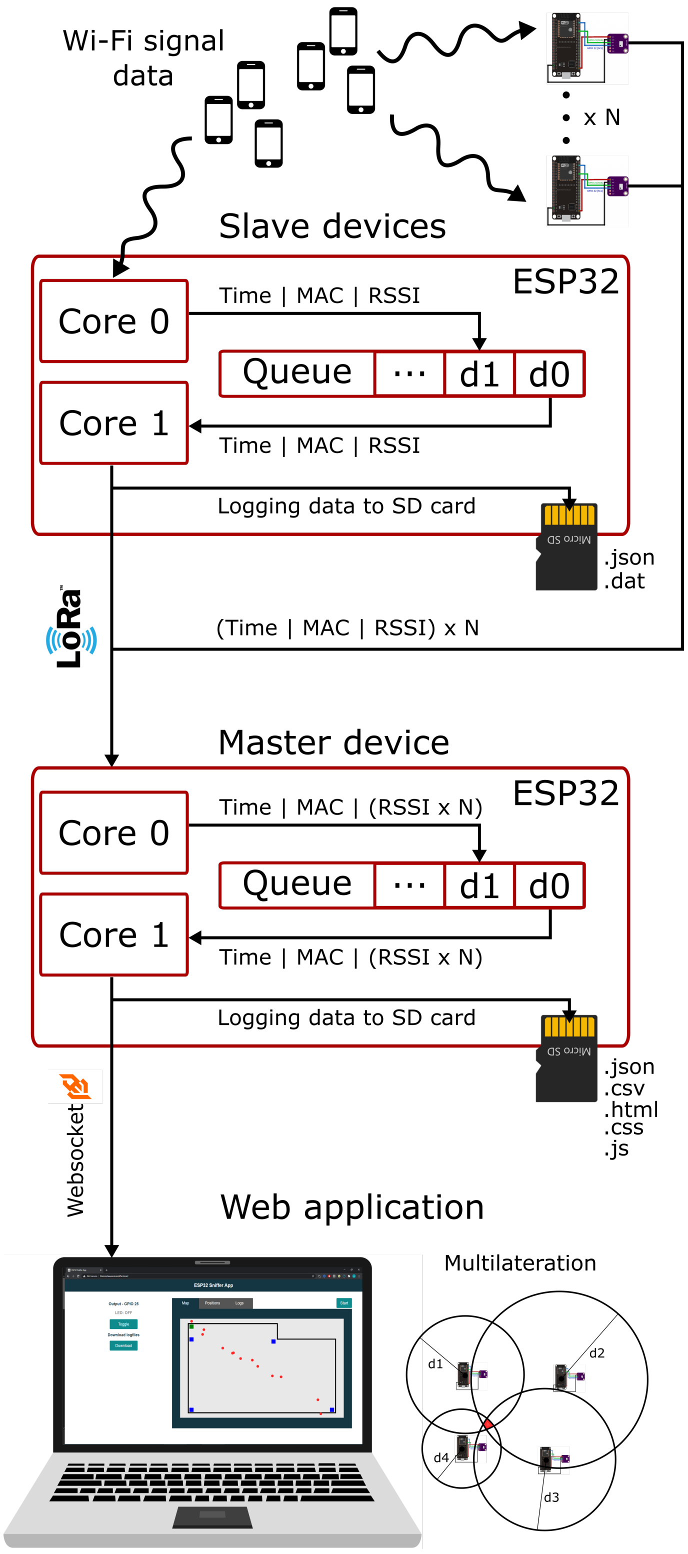


Figure 2.1: Workflow of the Sniffer application

2.5 Webserver connection

Multiple clients in the local network can access the webserver at the same time using the mDNS protocol (typing “**`http://themostawesomesniffer.local/`**” into the browser for mDNS access or typing the IP address of the master board directly). After the initial handshake via HTTP a websocket connection is established between server and client and further communication is channeled through it.

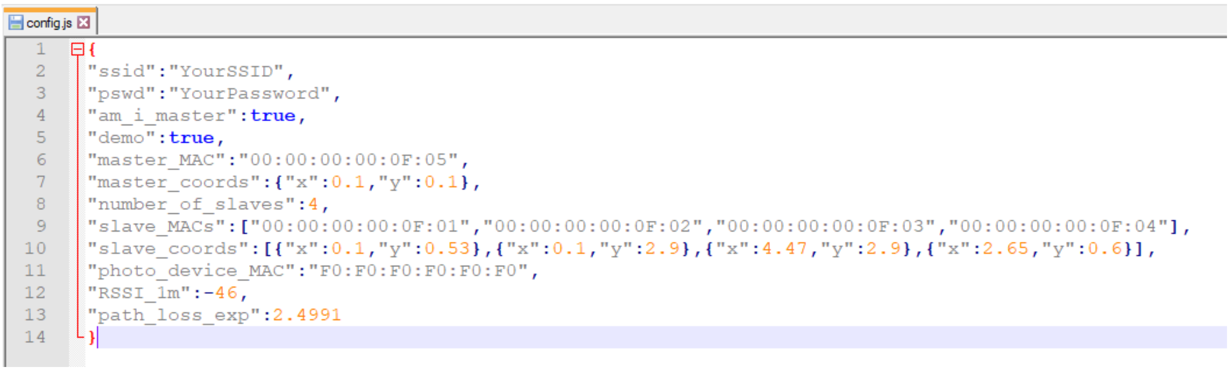
2.6 Web application

The necessary files (*.html, *.css, *.js) are loaded to the browser from the SD card of the master device when connecting to the webserver. After the websocket connection is established, downloading the logfile and controlling the onboard LED is possible (for multiple clients). Live streaming the measured data and position calculation and display is also available for a single client. The position determination is based on a multilateration algorithm using non-linear least squares method for iterating the final position estimate. For determining distances between the slave and tracked devices based on signal strength (RSSI) data, the log-normal shadowing radio propagation model is used. This position estimation method delivers good enough results when no major obstacles are present within the area.

3 Configuration of the devices

For the configuration of devices a JSON file (Figure 3.1) is used. This file (config.js) then is uploaded either to the SD card of the microcontroller or if it does not have such an option then to the internal SPI flash file system. The files to be uploaded to the SD card can be found in the project folder under the subfolder “sdcardcontent”. Files for the SPIFFS upload were placed under the subfolder “data”. The file uploading for the SD card happens in a straight forward manner, one has to connect the SD card to the notebook via a card reader and copy the folder into it. If the program has already been uploaded to the microcontroller one now only has to attach the SD card to it and it is ready to go.

In case of the SPIFFS uploading, one has to make sure that all files are placed into the subfolder “data” in the project folder as mentioned earlier and then one must run the “**`pio run -t uploadfs`**” command in the PlatformIO terminal. Alternatively one can use the PlatformIO IDE’s GUI and run the “Upload Filesystem Image” task. After that the program itself can be uploaded to the board too.



```
{
"ssid":"YourSSID",
"pswd":"YourPassword",
"am_i_master":true,
"demo":true,
"master_MAC":"00:00:00:00:0F:05",
"master_coords":{"x":0.1,"y":0.1},
"number_of_slaves":4,
"slave_MACs":["00:00:00:00:0F:01","00:00:00:00:0F:02","00:00:00:00:0F:03","00:00:00:00:0F:04"],
"slave_coords":[{"x":0.1,"y":0.53},{"x":0.1,"y":2.9},{"x":4.47,"y":2.9},{"x":2.65,"y":0.6}],
"photo_device_MAC":"F0:F0:F0:F0:F0:F0",
"RSSI_1m":-46,
"path_loss_exp":2.4991
}
```

Figure 3.1: The configuration JSON file

In the following the different configuration fields in the file (Figure 3.1) will be discussed.

- **ssid**: Character array. The Service Set Identifier of your local Wi-Fi network.

- **pswd**: Character array. Your Wi-Fi password.

- **am_i_master**: Boolean value. You can make the decision to use the specific microcontroller as a master (if it is set to true) or a slave device (if it is set to false).

- **demo**: Boolean value. You can decide to run the App in *normal* or *demo* mode. In demo mode (if it is set to true) the slaves only listen to a specific device (e.g., your mobile phone) which you have known its actual position in advance. (Since nowadays Apple/Android devices feature MAC address randomization when connecting to wireless networks for user privacy, you can use the tool Advanced IP Scanner to identify your phone's MAC address when connecting to the local Wi-Fi network.) Now this can be defined at the photo_device_MAC field. It was meant to be used for the photo device used to create the pictures for photogrammetry in case of creating a signal strength map, but this was not implemented in the end so it was miss-used for the purpose of the demo. But we kept the name to signal the initial intention, so that it can be used for extending the project with a signal strength map generating option. The position of the demo device in the room can be defined in the "general.js" file. With this demo mode one can check how accurate the position estimations are. On the other hand, one can choose to run in normal mode (if it is set to false), this will collect *all* MAC addresses of random devices passing by.

- **master_MAC**: Character array. The MAC address of the master device.

- **master_coords**: Json object containing two float fields. The Coordinate system is placed to the upper-left corner of the room, x pointing rightwards, y downwards. This was chosen because it is aligned with the HTML Canvas's coordinate system therefore making the plotting of positions easier.

- **number_of_slaves**: Integer value. One must provide the number of slave devices used.

- **slave_MACs**: Array of character arrays. It contains the MAC addresses of the slave devices.

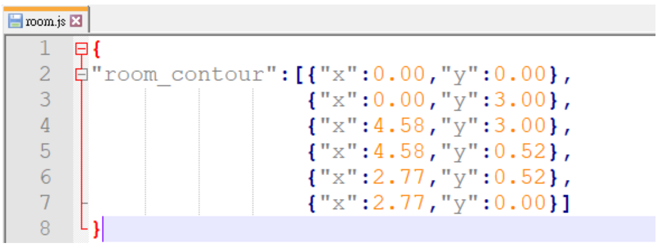
- **slave_coords**: Array of Json objects containing two float fields for the coordinates of the slave devices inside the room. Coordinate system is the same as was stated under the "master_coords" field.

- **photo_device_MAC**: Character array. Currently used for defining the MAC address of the demo device. But it is meant for containing the MAC address of the device that will be used to take the pictures for creating a signal strength map.

- **RSSI_1m**: Integer value. Measured reference signal strength. For details see the section on position determination.

- **path_loss_exp**: Float value. Measured exponent value for the Log-normal Shadowing Model used for position determination, details in the section about position determination.

For the plotting of the contour of the room, the corners of it must be defined in the "room.js" file in JSON format (see Figure 3.2). This also means that it can only handle a shape bounded by straight lines, no curved contours are possible. The function "drawSetup" defined in the "tabs.js" file is responsible for plotting the room contours and the slave, master and demo positions to the canvas under the "Map" tab in the web page.



```
{
"room_contour":[{"x":0.00,"y":0.00},
                {"x":0.00,"y":3.00},
                {"x":4.58,"y":3.00},
                {"x":4.58,"y":0.52},
                {"x":2.77,"y":0.52},
                {"x":2.77,"y":0.00}]
}
```

Figure 3.2: File containing the corners of the room to be plotted on the web page

4 Flashing of the devices

To structure the project with multiple boards associated with different (slave or master) tasks, the PlatformIO project configuration file (platformio.ini) in the root directory of the project is used. "platformio.ini" has sections and key/value pairs within the sections. A section with an `env:` prefix defines a **working environment**. Multiple `[env:NAME]` environments with different `NAME` (i.e., Master, Slave1, Slave2, Slave3, Slave4) are created for the current setup. More environment blocks must be established and customized when more devices are utilized in this project. Within each working environment block, one should configure at least 3 options: board, upload_port, build_flags. The procedure to setup these 3 options is addressed as follows respectively:

- **board:** You can find a valid board ID in Boards catalog supported by PlatformIO.
- **upload_port:** To find out the name of the serial port on Windows: Go to `Device Manager` → Click `Ports` tab → Here you will find the listed COM port devices, and their numbers: `COM1`, `COM2`, etc.
- **build_flags:** Define a perceptible name for each board in the format `-D name = definition` as a MACRO and use the conditional group (`#ifdef`, `#elif`, `#endif`) around any board specific code stuff. Note that `#ifdef MACRO` is precisely equivalent to `#if defined MACRO`. `defined` is useful when you wish to test more than one macro for existence at once. For example, `#if defined (Slave1) || defined (Slave2)` would succeed if either of the names `Slave1` or `Slave2` is defined as a macro. In this way, master-related code will only be compiled and uploaded to the Master, and slave-related code will only be compiled and uploaded to the Slave. One can simply set up the whole thing in only one project, rather than keeping two windows of VS Code open.

Please visit documentation for the other options and examples.

Next, the GPIO pins for each board need to be defined. This is done in the file "globalVariables.hpp". The conditional group (`#ifdef`, `#elif`, `#endif`) is again used here to achieve this purpose.

While finishing the configuration of "platformio.ini" and GPIO pins of each board in "globalVariables.hpp", it's a good practice to interact with the boards with PlatformIO IDE's project tasks GUI. One can choose to build, upload the sketch or upload files to SPIFFS with either all boards simultaneously under the **Default** task section or the individual board under the respective task section. (Figure 4.1)

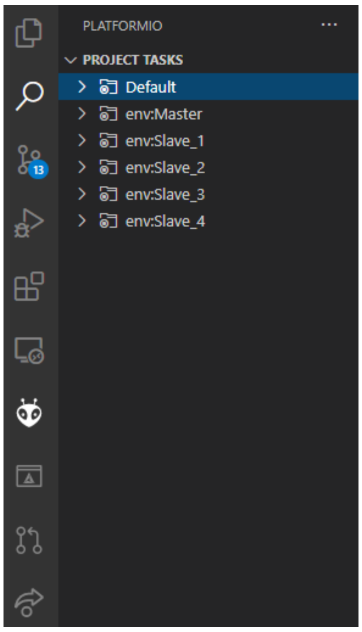


Figure 4.1: PlatformIO IDE's project tasks GUI

5 Position estimation algorithm

The RSSI-based position determination happens in two steps.

First, an estimation of the distance between the unknown device and a slave unit is calculated based on the so-called log-normal shadowing model (LNSM). The assumed connection between the distance and signal strength (RSSI) is depicted on the Figure 5.1. According to this relationship the distance can be determined [2] in terms of the measured signal strength ($RSSI$) a reference signal strength ($RSSI_{d_0}$) at a given reference distance (d_0) and the so called path loss exponent:

$$d = d_0 10^{\left(\frac{RSSI_{d_0} - RSSI}{10\eta}\right)} \quad [m] \tag{5.1}$$

The reference distance was taken as $1m$, the reference signal strength was measured at this distance with the help of one of the microcontrollers (measured at $1m$ distance at multiple points and then averaged). The path loss exponent was also measured. In this case also the RSSI values were measured at multiple positions, each further and further away from the source (mobile phone). The path loss exponent was determined from 5.1 at each point and then averaged. For this calculation the previously measured reference RSSI value was used.

$$\begin{gathered} d_0 = 1 \;\; [m] - Reference \;\; distance \\ RSSI_{d_0} \;\; [dBm] - Reference \;\; signal \;\; strength, \;\; measured \\ \eta \;\; [-] - Path \;\; loss \;\; exponent, \;\; measured \end{gathered}$$

The signal strength (RSSI) is defined in terms of the 10-based logarithm of the ratio of the measured power and the 1 mW reference power:

$$RSSI = 10lg\left(\frac{P}{1[mW]}\right) \quad [dBm]$$

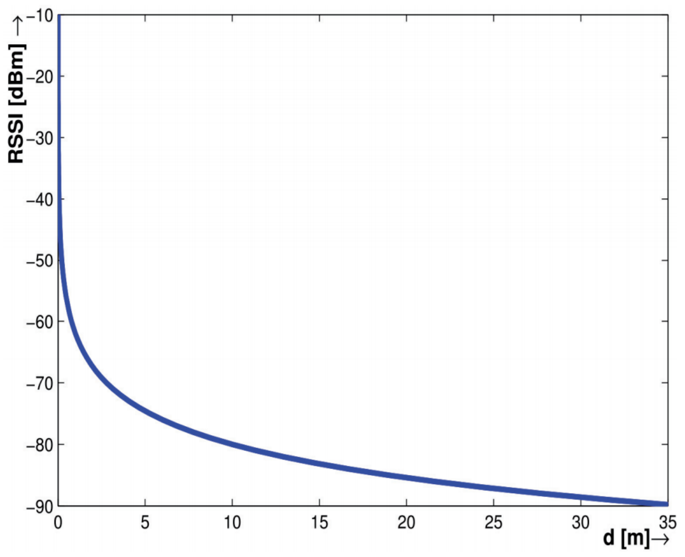


Figure 5.1: Relation between distance and RSSI according to the LNSM [2]

Knowing the distance of a device from a microcontroller means that it must lie on a circle with radius of this distance and the microcontroller as its midpoint. This scenario was depicted for 4 slave devices on Figure 5.2.

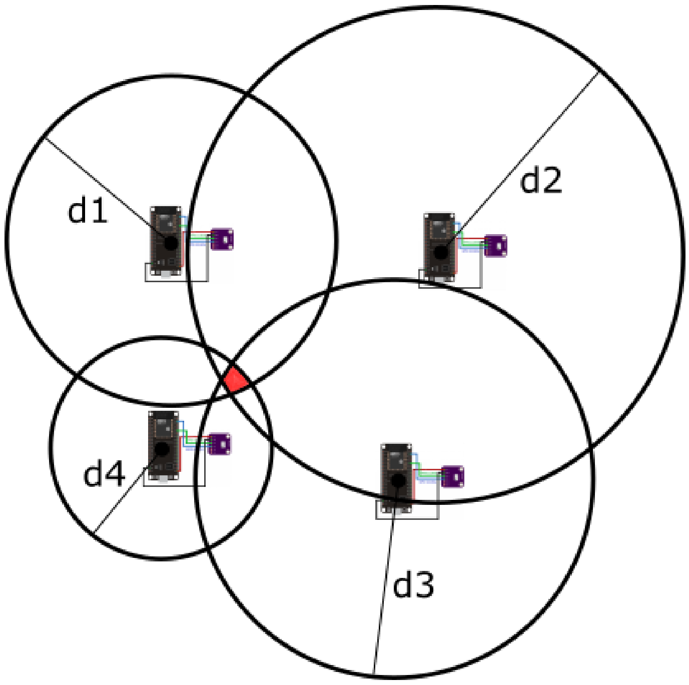


Figure 5.2: Multilateration scheme with 4 slave devices

Therefore an error measure can be defined using the measured distance and the equation of the circle centered at one of the microcontrollers end evaluated at a guessed position of the unknown device.

$$err_i = d_i^2 - \left[(x_p - x_i)^2 + (y_p - y_i)^2\right]$$

As a starting guess the position of the nearest slave device was used. The final estimated position is given as the coordinate pair which minimizes the sum of the squared errors:

$$S = \sum_{i=1}^{N} err_i^2(x_p, y_p) = min!$$

This minimization is done by using the non-linear least squares method.

6 Possible further development options

6.1 Improvement on the position determination accuracy

6.1.1 Better algorithms for RSSI-based methods

Based on the work of e.g. F.Shang et al. [1], one can realize that there are more sophisticated algorithms available than the one used in this project. These rely on the same measurements done with this application, only the evaluation part must be changed.

6.1.2 Signal strength map

One could program a mobile application which is capable of taking pictures from the venue and triggering an RSSI-measurement at the slave devices. This way one has as many RSSI values for each picture taken as many slave devices one uses. Using photogrammetry software a map from the pictures can be generated and the position of the mobile phone determined on it when the picture was taken. For these position we would then have all the RSSI values from the slave devices. Therefore a map was created with known RSSI values at given position.

When monitoring the area later on, one only has to compare the measured RSSI values with the ones defined on the map and determine the position of the unknown device based on this comparison. With this the reflections, diffractions of the radio waves on obstacles is taken into account to some level, therefore the the results are expected to be more accurate.

6.1.3 Time difference of arrival (TDOA)

Currently the timestamp is recorded when as many RSSI values as the number of slave devices have been collected/received by the master device. However, there indeed exists different arrival times from different slaves at the master, which is not taken into account in the current setup. Precise clock synchronization between slave devices is essential for achieving high-accuracy TDOA. By achieving accurate enough application layer clock synchronization one could use time difference measurements instead of RSSI values in the multilateration scheme. Since these are more reliable than RSSI measurements, the accuracy of the position determination would most probably increase.

6.2 Better accessibility and easier configuration

Currently the web application is only accessible for devices within the local network using an mDNS service or typing the IP address of the master device directly into the browser. It would be way more convenient if the application would be accessible from everywhere in the world where internet access is available. For this a VPN-based solution could be implemented.

Another inconvenience can be the configuration of the devices. If the microcontroller has an SD card slot, than copying the new file to the SD card an inserting it again is an acceptable effort for updating the configuration file, but still not ideal. However, in case when only the SPIFFS file system is available updating the configuration requires even more time. For this reason it would be nice to be able to select a device via the web application and upload the new configuration file “over the air”. This would not only speed up the later development process substantially when testing new settings or modifications, but would make the everyday use of the applications more convenient.

6.3 A more robust master device

Instead of using a basic esp32 one could use a more robust board like a Raspberry Pi as master device. It would be some extra effort to attach a LoRa module to it, but it would enable a more robust web application to be hosted on it. Python or PHP could be installed to do calculations on board. It would have all the benefits of a normal notebook but being more lightweight at the same time.

7 Useful links

In the following a decent amount of the online sources we have relied on during our work is collected, thematically ordered. This collection is meant to provide an entry point for the different topics we have touched on for those who might want to improve on this project later on.

7.1 Wi-Fi

7.1.1 General

http://www.sharetechnote.com/html/WLAN_FrameStructure.html

https://dot11ap.wordpress.com/a-mpdu-vs-a-msdu/

https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_wifi.html

https://www.esp32.com/viewtopic.php?t=3096

http://www.lucadentella.it/en/2017/01/16/esp32-6-collegamento-ad-una-rete-wifi/

7.1.2 Sniffing related

https://hackmag.com/security/esp32-sniffer/

https://metalfacesec.github.io/security/2020/01/09/esp32-wifi-sniffer-part-1.html

https://blog.podkalicki.com/esp32-wifi-sniffer/

https://www.youtube.com/watch?v=fmhjtzmLrg8

7.2 Webserver

7.2.1 General

https://www.theserverside.com/news/1363576/What-is-the-Asynchronous-Web-and-How-is-it-Revolutionary

https://github.com/me-no-dev/ESPAsyncWebServer#async-websocket-plugin

https://lastminuteengineers.com/creating-esp32-web-server-arduino-ide/

https://www.hackster.io/donowak/host-web-page-over-the-internet-on-esp32-using-sd-card-e4c72b

https://www.instructables.com/How-to-Serve-Html-Files-From-SD-Card-on-ESP32/

https://techtutorialsx.com/2018/08/24/esp32-web-server-serving-html-from-file-system/

https://techtutorialsx.com/2018/09/17/esp32-arduino-web-server-serving-external-css-file/

https://randomnerdtutorials.com/esp32-web-server-arduino-ide/

7.2.2 mDNS

https://www.megunolink.com/articles/wireless/find-esp32-esp8266-ip-address-with-mdns/

https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/protocols/mdns.html

https://tools.ietf.org/html/rfc6762

https://techtutorialsx.com/2020/04/18/esp32-advertise-service-with-mdns/

https://techtutorialsx.com/2020/04/17/esp32-mdns-address-resolution/

https://techtutorialsx.com/2020/04/27/esp32-query-mdns-service/

7.2.3 JSON

https://arduinojson.org/v6/doc/

7.2.4 Websocket

https://m1cr0lab-esp32.github.io/remote-control-with-websocket/

https://www.educba.com/what-is-websocket/

https://tools.ietf.org/html/rfc6455

https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_servers

https://randomnerdtutorials.com/esp32-websocket-server-arduino/

https://techtutorialsx.com/?s=websocket&orderby=relevance&order=DESC

7.2.5 Over-the-air programming

https://lastminuteengineers.com/esp32-ota-updates-arduino-ide/

https://lastminuteengineers.com/esp32-ota-web-updater-arduino-ide/

https://randomnerdtutorials.com/esp32-over-the-air-ota-programming/

https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/ota.html

7.3 LoRa

7.3.1 General

http://www.techplayon.com/lora-long-range-network-architecture-protocol-architecture-and-frame-formats/

https://randomnerdtutorials.com/esp32-lora-sensor-web-server/

https://randomnerdtutorials.com/esp32-lora-rfm95-transceiver-arduino-ide/

https://github.com/sandeepmistry/arduino-LoRa/blob/master/examples/LoRaDuplex/

7.3.2 Application layer clock synchronization

https://lora-alliance.org/resource_hub/lorawan-application-layer-clock-synchronization-specification-v1-0-0/

https://arxiv.org/pdf/1709.08296.pdf

7.4 NTP time

https://lastminuteengineers.com/esp32-ntp-server-date-time-tutorial/

https://www.youtube.com/watch?v=r2UAmBLBBRM

7.5 SD card

https://github.com/espressif/arduino-esp32/tree/master/libraries/SD#faq

https://randomnerdtutorials.com/esp32-data-logging-temperature-to-microsd-card/

https://majenko.co.uk/blog/fast-efficient-data-storage-arduino

7.6 SPIFFS

https://docs.platformio.org/en/latest/platforms/espressif32.html#uploading-files-to-file-system-spiffs

7.7 FreeRTOS

https://freertos.org/implementation/a00004.html

https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/freertos-smp.html

https://www.youtube.com/watch?v=jpVcCmh8sig&ab_channel=RalphSBacon

https://www.youtube.com/watch?v=ywbq1qR-fY0&ab_channel=RalphSBacon

https://randomnerdtutorials.com/esp32-dual-core-arduino-ide/

https://techtutorialsx.com/2017/09/13/esp32-arduino-communication-between-tasks-using-freertos-queues/

https://www.studytonight.com/operating-system/cpu-scheduling

http://www.iotsharing.com/2017/06/arduino-esp32-freertos-how-to-use-task-param-task-priority-task-handle.html

7.8 Position determination

https://core.ac.uk/download/pdf/30312076.pdf

https://journals.sagepub.com/doi/full/10.1155/2014/371350

https://www.researchgate.net/publication/256733582_Path_Loss_Exponent_Analysis_in_Wireless_Sensor_Networks_Experimental_Evaluation

https://www.rn.inf.tu-dresden.de/dargie/papers/icwcuca.pdf

http://mason.gmu.edu/~rtruong2/project2/

http://www.alanzucconi.com/2017/03/13/positioning-and-trilateration/

https://webpages.uncc.edu/~anasipur/pubs/Chuku-Southeastcon-2013.pdf

7.9 General ESP32 programming

https://www.circuitbasics.com/types-of-memory-on-the-arduino/

https://medium.com/the-esp-journal/esp32-programmers-memory-model-259444d89387

https://www.geeksforgeeks.org/structure-member-alignment-padding-and-data-packing/

https://www.youtube.com/watch?v=DoctWoxIaH8&list=PLmQ7GYcMY-2JV7afZ4hiekn8D6rRIgYfj&ab_channel=ControltheController

https://randomnerdtutorials.com/esp32-ssd1306-oled-display-arduino-ide/

https://docs.platformio.org/en/latest/boards/espressif32/ttgo-lora32-v2.html#hardware

https://youtu.be/C2xF3O6qkbg

https://leanpub.com/kolban-ESP32

https://docs.espressif.com/projects/esp-idf/en/latest/esp32/

https://lastminuteengineers.com/esp32-deep-sleep-wakeup-sources/

https://majenko.co.uk/blog/evils-arduino-strings

https://youtu.be/DoctWoxIaH8

https://majenko.co.uk/blog/splitting-text-c

References

[1] F.SHANG et al. *A Location Estimation Algorithm Based on RSSI Vector Similarity Degree*. International Journal of Distributed Sensor Networks, Hindawi Publishing Corporation, 2014.

[2] M.SIMEK M.BOTTA. *Adaptive Distance Estimation Based on RSSI in 802.15.4 Network*. RADIOENGINEERING, VOL.22,NO.4, 2013.