UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Oliver Ossip

A comparison of Over-the-Air Update approaches for the ESP32 and ESP8266 Development Boards

Bachelor's thesis (9 ECTS)

Supervisor:Jakob Mass, MsC

Tartu 2020

ESP32 ja ESP8266 arendusplaatide läbi õhu uuenduste võrdlus

Lühikokkuvõte:

Käesoleva bakalaureusetöö eesmärk on uurida ja katsetada erinevaid läbi õhu (ingl *over-the-air*) uuendamise võimalusi ESP32 ja ESP8266 põhinevatel Wi-Fi arendusplaatidel. Tänapäeval on üha rohkem esemevõrgu (ingl *Internet of Things*) seadmeid ning tekib paratamatult vajadus neid uuendada. Põhjus selleks võib olla kas turvaauk või viga koodis, mis tuleks parandada seadme korrektseks toimiseks. Läbi õhu uuendamine annab võimaluse seadme tarkvara kiireks parandamiseks. See lubab seadmeid uuendada kaugemalt ning kohtadest, kuhu inimesel puudub füüsiline ligipääs. Käesolevas töös katsetatakse erinevaid raamistikke ning kahe erineva generatsiooni arendusplaate.

Võtmesõnad:

Arduino, ESP8266, ESP32, OTA programming

CERCS: P170

A comparison of Over-the-Air Update approaches for the ESP32 and ESP8266 Development Boards

Abstract:

This bachelor thesis goal is to research and test different over-the-air update opportunities with ESP32 and ESP8266 based Wi-Fi development boards. Nowadays, there are more Internet of Things devices, so there is a need to update these. The reason for that can either be a security flaw or a bug that needs to be fixed for the device to work. Over-the-air update allows to make changes fast and in remote locations where humans have no physical access. This thesis tests different frameworks and two separate generation development boards.

Keywords:

Arduino, ESP8266, ESP32, OTA programming

CERCS: P170

Contents

1. Introduction

Internet of things (IoT) has become increasingly popular. In 2021, there are expected to be around 20 billion IoT smart devices in the world [1]. IoT is the physical devices connected to internet with sensors, functional software, which can be remotely controlled or monitored. With IoT we don't need human interaction, all communication is machine to machine. Without human intervention, machines can communicate faster and gather more data. The main benefits for IoT are to save both money and time [2]. By IoT the objects can sense around environment and be controlled remotely through existing network infrastructure [3]. These are some real-world IoT applications:

- Smart city
- Smart homes
- Smart cars
- Iot in farming
- Health & Fitness

IoT devices are usually deployed in large quantities and they are used in places that are difficult or impractical to access. For example there can be many sensors on the field measuring soil fertility [4]. This would give necessary information when would be the best time to cultivate.

It is hard to create perfect software, so there might be bugs that need fixing. Also, there might be new regulations like Data Protection Regulation (GDPR) [5]. In a smart home environment the sensors can collect personal information. According to new regulation this means that the device data collection should be limited. In such cases it is good to have over-the-air (OTA) update functionality. Over-the-air update is a process which allows devices to upgrade firmware or software wirelessly [6]. OTA updates are used for updating software, resolving bugs or adding features. Over-the-air update can save time and update the device without any physical human interaction. Without OTA it would be difficult to update devices that are geographically dispersed. Using over-the-air updates also bring more benefits. This gives the ability to quickly respond to the security vulnerabilities. OTA updates can reduce maintenance costs [7, 8]. This gives customers opportunity to use devices longer because they don't have to buy new device to get new features. Nowadays, probably no one would imagine that they are taking their device to shop for getting a simple software update.

The goal of this thesis is to compare over-the-air updates solutions on development boards. Three different frameworks are used for it. These are tested with two microcontrollers, ESP32 and ESP8266. Frameworks are compared by setup, user experience, coding and performance.

The thesis is structured as follows. Section 2 gives a simple overview of over-the-air update and introduces similar works. Section 3 introduces used technologies and frameworks. Section 4 gives an overview of the prototype implementation and comparison. Section 5 concludes the work and introduces future research options.

2. State of the Art

In this chapter the author gives a simple overview of over-the-air update benefits and problems. Also introduces different solutions for OTA. There are some similar articles on the topic of development board over-the-air update usage.

2.1 OTA overview

Over-the-air update is launching new software or firmware wirelessly. OTA update is used in many different places: smartphones, laptops, TVs and even cars. Launching new software through the Internet is faster and easier than to do it with physical connection. OTA update reaches bigger range of users or devices. It is possible to add new features after device is already set in place [9]. It is difficult to deploy perfect software. Usually there are bugs that affect in a smaller or bigger way. These bugs needs to be patched to keep the software's robustness and security. The solution for this are over-the-air updates. There can be three levels of IoT software updates:

- Full Firmware
- Platform
- Application

There is no need to update the full firmware every time. Sometimes it is only needed to update some driver or application. Dividing these into smaller pieces might reduce the update sending time or downtime. There are also some problems with over-the-air programming. Depending on the IoT device, updates should be sent that it will not interrupt device work. Before there should be checks if device has enough memory. In case of battery usage there should be also battery percentage check. One of the most important aspects is security. Updates must be sent securely to the device, so no other third party has a chance to send updates and take control of it. To avoid that, it is possible to use signed updates. IoT device confirms that the update is sent from the right party and then applies the update. There should be some solutions in case the update fails, so that the device can boot to the old firmware [10].

There are some different ways to complete the over-the-air update. A web server can be used, device is serving server for updates and user uploads a new update file to device. The effective approach is to check new update from server. This solution is best for updating multiple devices [11]. There are also some companies (for example Particle) that offer updating infrastructure [9].

2.2 Similar works

This part of thesis introduces similar works. These articles are using over-the-air update for development boards. Each article has a different solution and a board.

2.2.1 Remote Programming with TelosB Sensor Node

The article [12] uses TelosB Sensor Node[1] with TinyOS to program over-the-air. TelosB applications for OS is designed for sensor nodes and are programmed in C or in its derivatives. Used TelosB sensor node uses 16-bit microcontroller and is equipped with humidity, temperature and light sensor. TinyOS is a component-based operating system for low-power wireless sensor networks (WSN). TinyOS programs are written in nesC (network embedded systems C) [13]. For OTA update, patches are used to update the code. It runs diff script, which contains the update, code difference and patch operation for applying the update. Patching is performed using architecture-specific instructions. The article author brings out problem that diff- like approaches, generating update patch by comparing two software versions, ignore the application structure. So even a tiny change in the code may result in a large update patch. Osman uses software compression for reducing the size of software updates. Compression algorithms are used to encode the data such that encoded output is smaller than the source.

2.2.2 OTA Programming with Particle Core

The article [14] uses Particle Core[2] for OTA programming. Particle provides the web IDE and has OTA update function built in. Particle Core uses STM32 microcontroller with a Texas Instruments CC3000 Wi-Fi module. Particle microcontrollers are using Device OS [15]. From Particle web IDE is capable to update the program into the microcontroller.

[1] https://telosbsensors.wordpress.com/
[2] https://www.particle.io/iot-platform

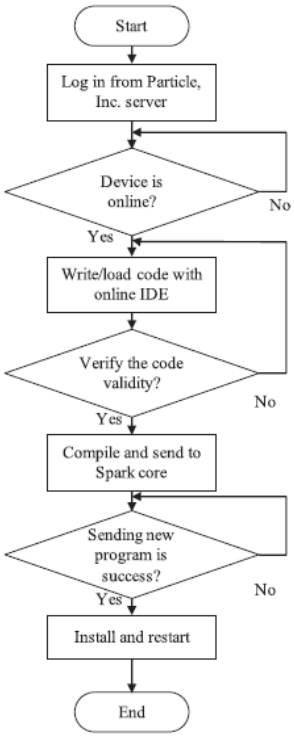

Figure 1: Flowchart for updating program [14]

Figure 1 shows the flowchart of the OTA function via the Particle server. OTA function has four conditions: compiling, transferring, installing, reboot.

2.2.3 Arduino update with Beagle Board

The article [16] uses Arduino, Beagle Board[3] and 3G modem to make OTA update. The purpose of the article is to make car diagnostics and update car software when there is an update. That would save a lot of time for mechanics, manufacturers and customers because many of the problems are already known before riding to a workshop and some problems can be solved with an update.

[3] https://beagleboard.org/beagleboard

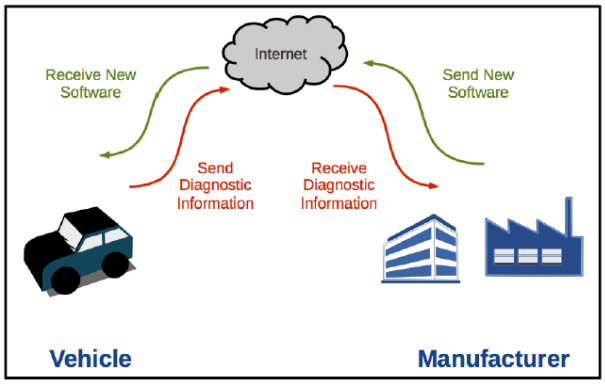

Figure 2: System work principle

Every time the system starts, there will be communication with the server. Figure 2 shows how the system works. The system sends current version number to server and server checks if it is a new version for update. The Beagle Board connected to 3G modem implements the client side. Ubuntu Linux is installed to the board since it has many open source applications. Software AVRDUDE is used for updating the AVR microcontrollers. AVRDUDE is a program for downloading and uploading the on-chip memories of Atmel's AVR microcontrollers [17]. The board is used for connecting the Internet, storing update files and updating the Arduino.

2.2.4 Update with ESP8266 and MQTT

The article [18] presents a home automation solution based on a MQTT[4] (Message Queue Telemetry Transport) with ESP8266-based sensors and actors. MQTT is simple and lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks [19]. It allows devices to listen to commands and publish current state to the controller and other interested parties. This allows administrators to find outdated bootloaders and failed updates. For over-the-air update has been used Sming framework [20], which has OTA support. The memory mapping mechanism of the MCU allows only a single page of 1MB of flash to be mapped at the same time. To avoid executing the same code while downloading the flash is split into half to contain two firmware ROM slots with different versions: the one being executed and the other which is being downloaded.

[4] https://mqtt.org/

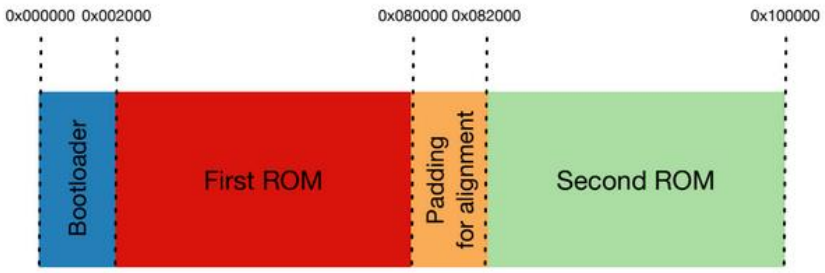

Figure 3: The flash layout for two ROM slots

For alignment and easy debugging, the blocks are in same size, flash layout shown on figure 3. First ROM acts like safety mechanism because when download fails, the previous version stays intact. The OTA update contains four main phrase: checking for update, reprogramming device, verifying crytographic signature and assuming that the update was successful. The update servers provides file with meta-information about the latest version. When the file is downloaded, the version is checked. If the versions differ, then the update is initialized. Reprogramming phrase the correct slot of ROM is selected and a new firmware update is downloaded.

2.2.5 Comparison

	TelosB	Particle	ESP8266	Arduino with Beagle Board
2 devices for OTA				X
Protocol	HTTPS	HTTPS	MQTT	HTTPS
Update infrastructure		X		
Server for OTA	X	X	X	X
Wireless tech	Wi-Fi	Wi-Fi	Wi-Fi	3G
Verification	X	X	X	X

Table 1: Similar works comparison

Table 1 is table to compare articles. This thesis uses two different development boards for testing. Both have Wi-Fi chips and OTA support. This thesis does not use any security protocol for testing frameworks. These articles are using server for the over-the-air update, but this thesis uses also development board web server for updating the board.

3. System design

In this chapter the author introduces technologies and frameworks used for this thesis. There will be introduced ESP development boards used for the thesis. Three OTA framework and some different solutions are also introduced.

3.1 Technologies

3.1.1Wemos D1 board

The author chose ESP8266 board because of its popularity and it is affordable price [21]. ESP8266 also has compatibility with other development environments. This chip also has a big community where different topics are shared and discussed.

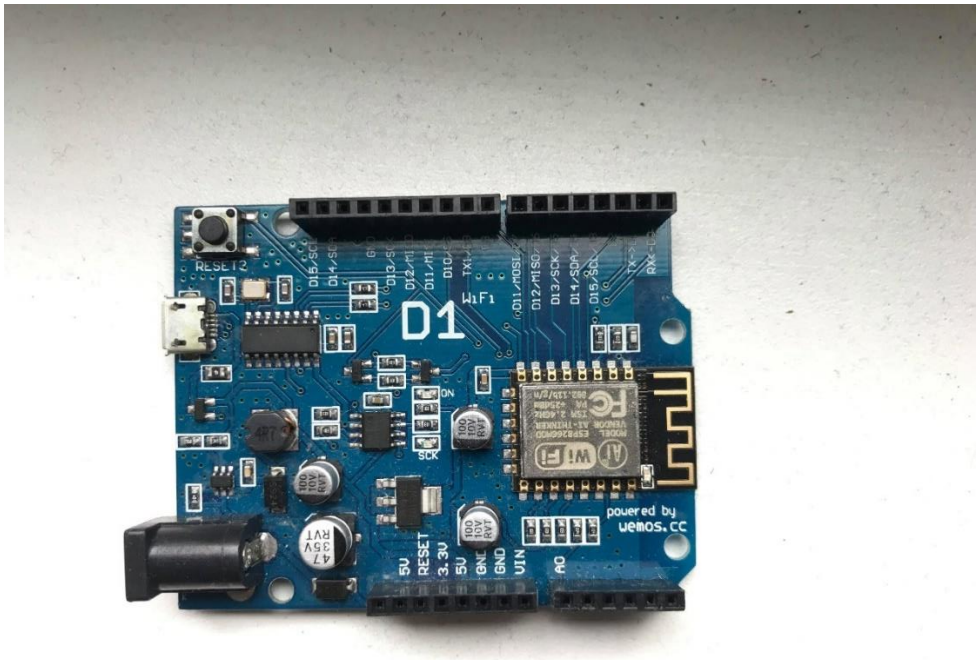

Figure 4: Wemos D1 board

The author was using Wemos D1 R1 ESP8266 Wi-Fi based board that uses the Arduino UNO layout, shown on figure 4. It gives the possibility to use Arduino UNO shields. Wemos D1 also has a mini version, which is almost the same size as Wi-Fi chip.

Wemos D1 specifications:

- 17 pins
- Wi-Fi 2.4GHz
- Micro USB connection
- Power jack(9-24V)
- Compatible with Arduino
- Compatible with NodeMCU
- Operating voltage: 3.3V
- Clock speed: 80MHz/160MHz
- Flash: 4MB

Wemos D1 uses ESP8266EX, it is integrated with a 32-bit Tensilica processor, antenna switches, power amplifier, filters and power management modules [22] . ADC(Analog to Digital Converter) is a feature that converts analog voltage on a pin to a digital number. This allows to use sensors and get values.

3.1.2 DOIT ESP32 DevKit v1

The author chose ESP32 because it is the next generation of ESP8266. This gives the opportunity to compare two different generation boards.

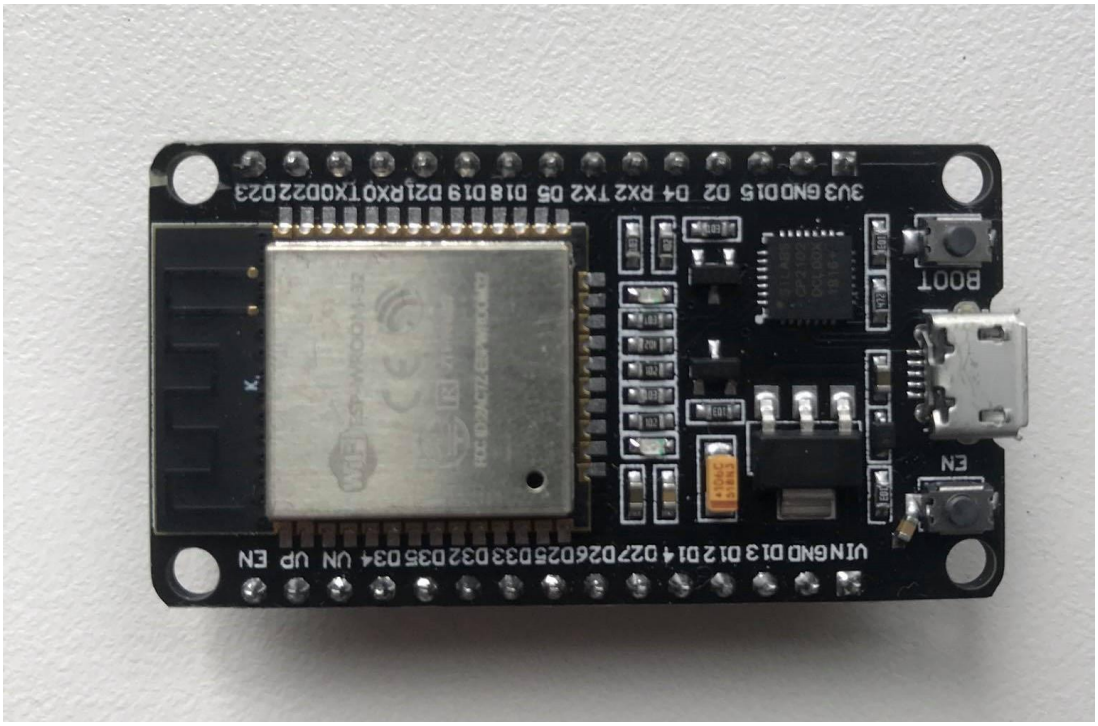

Figure 5: DOIT ESP32 DevKit v1 board

DOIT specifications:

- 30 pins
- Wi-Fi 2.4GHz
- Bluetooth
- Hall sensor
- Touch sensor
- Clock speed: 240MHz
- SRAM: 520KB
- Flash: 4MB
- Micro USB connection
- Operating voltage: 3.3V

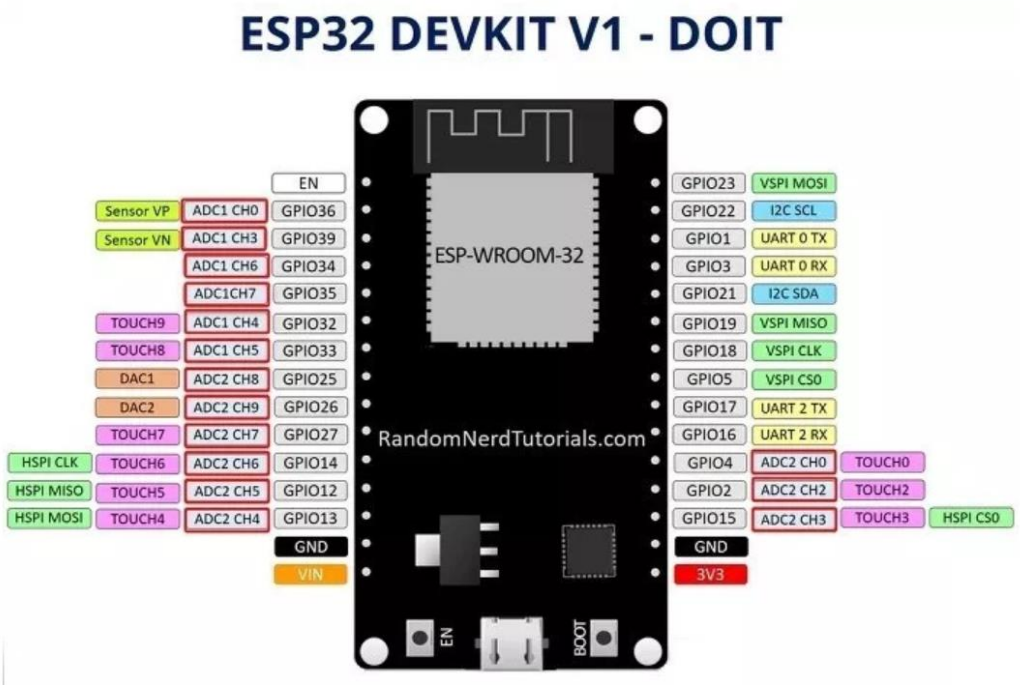

Figure 6: DOIT 30 pins pinout

DOIT board has a 30 and a 36 pin version. The author used the 30 pin layout board, which is shown on figure 5 and figure 6 with pinout reference [23]. ESP32 supports measurements in 18 different channels, but only 15 are used in this board. There are some limitations when using Wi-Fi. ADC2 pins cannot be used, because they are used by chip. Instead it is possible to use ADC1 pins.

3.1.3 Arduino IDE

The Arduino IDE(integrated development environment) is a cross-platform application for writing and uploading programs to Arduino compatible boards. It is written in Java. The software is open source and is available for extension. The language can be expanded through C++ libraries. The Arduino language is a set of C/C++ functions that can be called from your code. The Arduino sketch undergoes minor changes and goes to C/C++ compiler [24, 25].

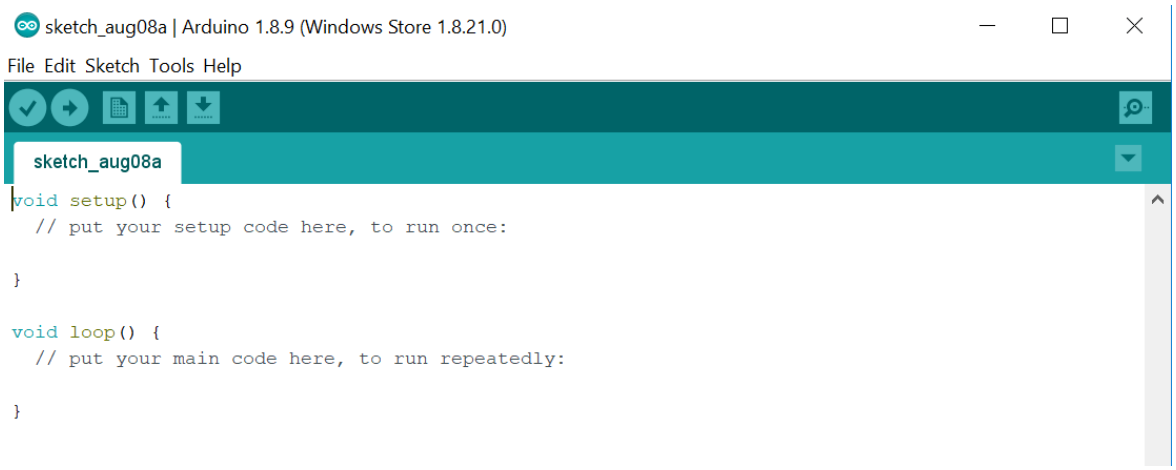

Figure 7: Arduino IDE default sketch

Arduino sketch contains mainly two functions loop() and setup(), as shown on figure 7. Setup function is called when sketch starts and it is called only once [26]. For example there are set Wi-Fi credentials. Loop function loops consecutively, allowing the program to change and respond [27]. This function is running program main code, for example, controlling leds and reading sensor values.

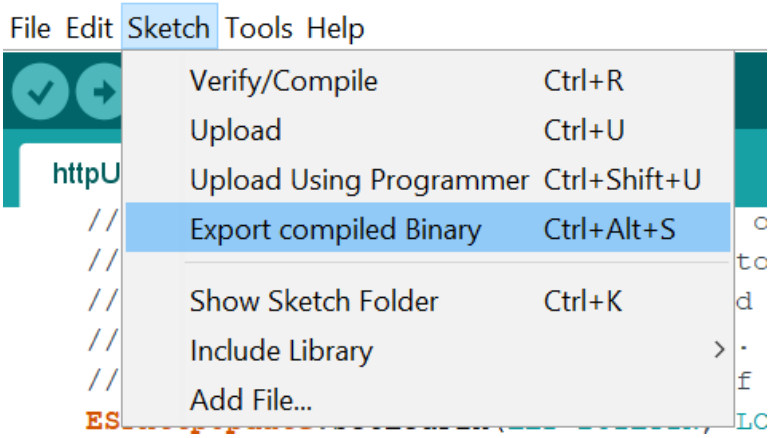

Figure 8: Shows how to export binary in Arduino IDE

In Arduino IDE it is possible to export compiled binary files, as shown on figure 8. This is an executable file, which can be sent to board. Binary files are used for making over-the-air updates.

3.1.6 Flask

Flask is a web framework in python. It is called also microframework, because Flask keeps the core simple but extensible. This framework allows to use different databases or other system parts. By default Flask doesn't contain any form validation, upload handling or open authentication technologies. But it is possible to add these extensions as if they were already implemented in Flask itself. Templates and static files are stored in subdirectories within the application's Python source tree [28]. Thesis uses Flask for updating the development board.

3.2 OTA Frameworks

In this section the author introduces frameworks that has been used in this thesis. Frameworks have been used with ESP32 and ESP8266 boards. There are different possibilities to update the board. This thesis uses Arduino, Sming and ESP-IDF frameworks. The author chose frameworks that has been kept up to date and have enough documentation for over-the-air update.

3.2.1 Arduino core

This framework is meant to use with Arduino IDE. This gives an opportunity to use the Arduino code layout. Coding in Arduino IDE requires adding framework to IDE. This framework is possible to use with PlatformIO, which is cross-platform for embedded products.

ESP32[5] and ESP8266[6] chips have separate frameworks for Arduino Core. Both contain many libraries with examples. These frameworks might differ by libraries and given examples.

ESP32 and ESP8266 board frameworks already contain three basic updates:

- Arduino/Basic OTA
- Web Server
- HTTP Update

[5] https://github.com/esp8266/Arduino
[6] https://github.com/espressif/arduino-esp32

By default these over-the-air update examples are not secure. It is possible to improve security adding certs, password or signing update. Initial flash must be done with cable. Compiling and flashing is done using Arduino IDE.

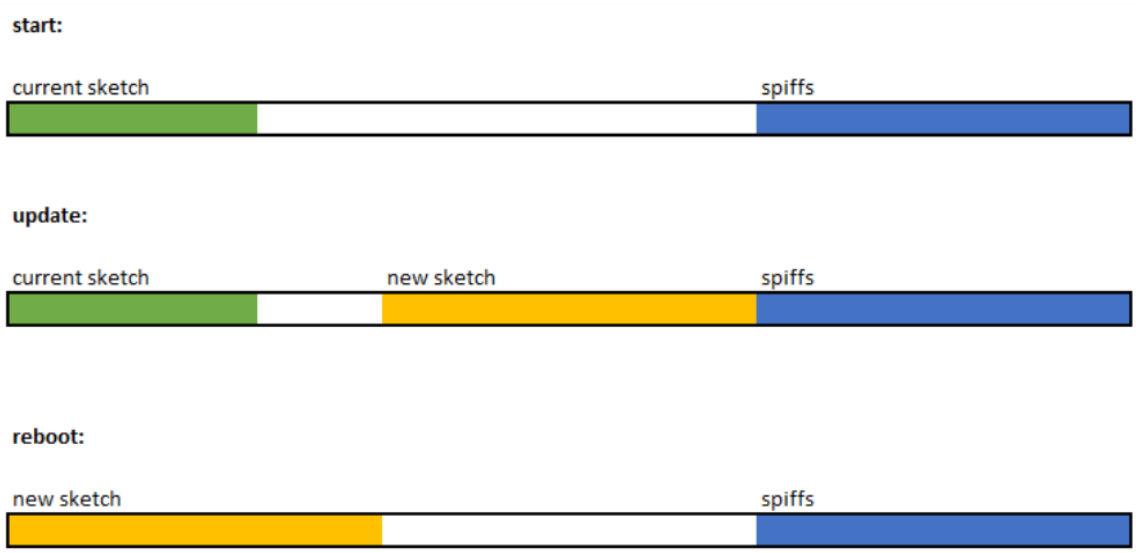

Figure 9: Shows update process memory view

Figure 9 shows the update process for ESP8266. The update sketch will be stored in the middle, between the current sketch and spiffs. With reboot current sketch will be replaced with new one. Spiffs is flash file system. It is designed for low ram usage and only bigger data blocks can be deleted [11].

Arduino OTA

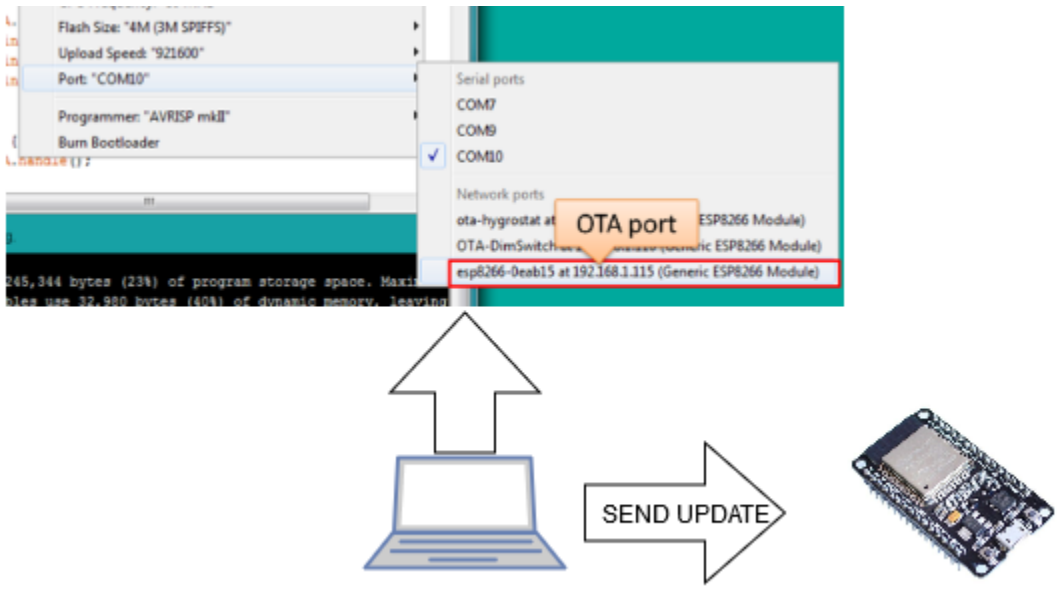

Figure 10: Arduino OTA update

Arduino OTA, also called Basic OTA is meant for using in local network. This OTA is meant to be used with Arduino IDE, but it is possible to send updates from command line. Process is shown on figure 10. After first update the board should be connected to Wi-Fi. Then IDE should automatically find new port option for OTA. Using the right port, it is possible to update the board the same way when using cable.

Web Browser

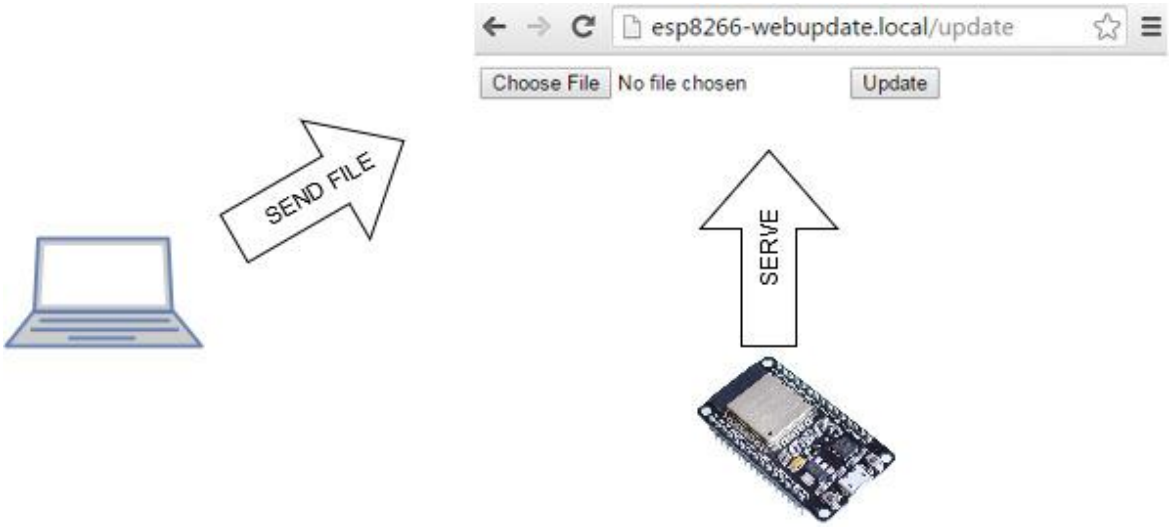

Figure 11: Web Browser update

Web server OTA option allows to send updates to development board through the web page, shown on figure 11. This update method is used when there are small quantity of modules or update from Arduino IDE is inconvenient. After the first upload, serial monitor prints web address to access the web server. Next updates are possible to complete by uploading the compiled binary file.

HTTP Server Update

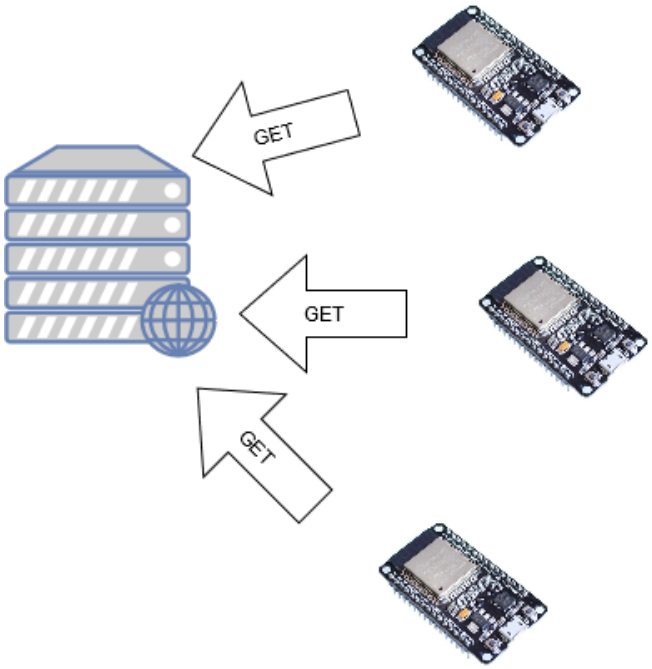

Figure 12: Http server update

Http server update OTA option can be used to update multiple development boards. As shown on figure 12 the board makes request to get update. Server keeps the compiled binary update.

```
X-Esp8266-Sta-Mac: 60:01:94:10:7F:2C
X-Esp8266-Ap-Mac: 62:01:94:10:7F:2C
X-Esp8266-Free-Space: 3874816
X-Esp8266-Sketch-Size: 295200
X-Esp8266-Sketch-Md5: 4edf7938c60af28b2057747a0623a4b9
X-Esp8266-Chip-Size: 4194304
X-Esp8266-Sdk-Version: 3.0.0-dev(c0f7b44)
X-Esp8266-Mode: sketch
```

Figure 13: Headers sent from ESP8266 board

Figure 13 shows headers development board sends sent to server. Now it depends on the server how this information is used. One of the values is a Sketch-Md5 value, which can be used to check if an update is needed. Md5 is a hash-function producing 128-bit value hash. It was designed for a cryptographic hash function, but it suffers with some vulnerabilities.

Still, it is suitable to use for verifying data [29]. Example sketch uses two response codes. Response code 304 is sent when an update is not needed. This indicates that the resource has not been changed. Response code 200 is used when update downloading was successful. After that, the development board boots with new firmware.

Async Elegant OTA

Async Elegant OTA[7] library is like the Web Browser OTA solution above, figure 11. This is not the default library in framework, this should be added IDE. As the name says its web page user interface (UI) is more elegant and nicer to use. This library uses asynchronous web server[8]. This gives an opportunity to handle more than one connection at the same time. Asynchronous execution gives a possibilty to run tasks on the background. In this case it gives an opportunity to run a task while the update is being downloaded.

ESP32 FOTA

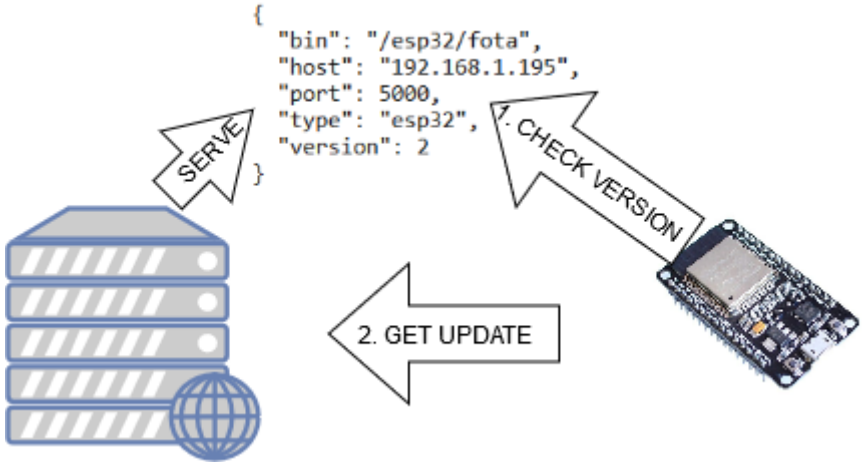

Figure 14: FOTA update

FOTA[9] library is meant only for ESP32 board. For updating, there must be server side. The server holds compiled binary file and json file. Json file should contain:

- A webserver with the firmware information in a JSON file
- Firmware version
- Firmware type
- Binary file path

Development board checks json to compare firmware version. If json is updated and there is a new version, update is downloaded from firmware bin location. This library must be imported separately.

[7] https://github.com/ayushsharma82/AsyncElegantOTA
[8] https://github.com/me-no-dev/ESPAsyncWebServer
[9] https://github.com/chrisjoyce911/esp32FOTA

3.2.2 Sming framework

Sming[10] is an asynchronous C/C++ framework with good performance and multiple network features. It is open source and made for embedded devices. Sming works with multiple architectures, one of these is ESP8266. Sming is described as following [20]:

- Superb performance and memory usage
- Simple and powerful APIs
- Built-in powerful wireless modules
- Powerful asynchronous network stack

Sming over-the-air update uses second-stage bootloader rBoot[11]. This allows ESP8266 board hold more than one application. There can be totally different applications or different versions same application.

This framework does not have UI. Compiling and flashing is made using make commands on the command line.

The framework has example that demonstrates the use of rBoot. Example uses serial monitor commands to make update. OTA update is hold on a server and is sent to the development board, when over-the-air update command is inserted on the serial monitor.

3.2.3 ESP-IDF

ESP-IDF[12] is the official framework for ESP32. The software development framework by Espressif is intended for the development of IoT. The framework is written in C. Over-the-air update can do upgrading at the runtime by downloading new image from specific server through Wi-Fi or Ethernet and then flash it into some partitions.

This framework has two demos: native OTA and HTTPS OTA, which adds an abstraction layer in order to upgrading with HTTPS protocol. It also supports versions of the application. This means each update has its own version and is checked before downloading. New version is downloaded when versions differ. Version checking is performed after the initial upload is made. Code compiling and flashing is made on the command line. There is a simple UI for configuring the development board [30, 31].

[10] https://github.com/SmingHub/Sming
[11] https://github.com/raburton/rboot
[12] https://github.com/espressif/esp-idf

4. Prototype Implementation and Performance Evaluation

This chapter introduces the prototype what is used for testing, setup for server and compares results.

The code used for comparing over-the-air updates is simple system to show how this can be used in smart city environment. The prototype uses small led light and light sensor, also known as a photoresistor. The goal is turn on the light when it is dark outside and turn off when there is enough daylight. The system can be used in city environment to control street lamps. Usually city lights are turned on at a certain time, but this solution can save money by not turning lights on too early. This gives citizens a safer feeling that the streets are illuminated at nights. Because of the shadows some places can be darker than others. Using sensors, it is possible to identify the darker parts of town and turn on only these lights.

The goal of this chapter is to compare different OTA frameworks and two different generation Wi-Fi chips. Each over-the-air update download and boot time is measured. These results are used for comparing results. Development boards are Wemos D1 with ESP8266 and DOIT DevKit, which uses the newer generation microcontroller ESP32.

All tests are made in a home environment. The author used regular internet customer router Inteno DG200A. Tests have been made about one-meter radius from router. Each OTA timings consists on the average of five tests.

4.1 Development board code

This subsection introduces code used for development board. Here the author describes the used code.

```
const int ledPin = 13; // The pin led is attached.
const int sensorPin = A0; // The pin sensor is attached.
const int lightValue = 350; // Value, when turn the light on.
int sensorValue; // Sensor value variable.
int counter = 0; //Set counter.

void setup() {
  Serial.begin(115200);
  pinMode(ledPin, OUTPUT); // Initialize led.
}

void loop() {
  // Reading analog every loop causes problems
  // with ESP8266 Wi-Fi connection.
  if (counter == 5000) {
    sensorValue = analogRead(sensorPin);

    // Turn on led, if it is dark enough.
    if (sensorValue < lightValue) {
      digitalWrite(ledPin, HIGH);
    } else { // Else keep or turn of light.
      digitalWrite(ledPin, LOW);
    }
    counter = 0;
  }
  counter += 1;
}
```

Figure 15: Code in Arduino IDE

Figure 15 shows code for controlling the led with light sensor with Arduino framework. Similar solution can be used also in Sming framework. As Sming does not use setup and loop functions, there should be similar task set as loop. Figure 15 code has set counter for analog reading, because of ESP8266 Wi-Fi problem[13]. The development board could not connect internet, when analog read is made too often, like every loop time. This code must be added upon to over-the-air update code.

[13] https://github.com/esp8266/Arduino/issues/1634

```
while(1) {
    uint32_t adc_reading = 0;
    // Read analog value
    adc_reading += adc1_get_raw((adc1_channel_t)channel);
    // Set led pin on or off
    if (adc_reading< value ){
        gpio_set_level(BLINK_GPIO, 1);
        vTaskDelay(1000 / portTICK_PERIOD_MS);
    }else{
        gpio_set_level(BLINK_GPIO, 0);
        vTaskDelay(1000 / portTICK_PERIOD_MS);}
}
```

Figure 16: ESP-IDF code

ESP-IDF framework is quite different from commands. Figure 16 shows the light control code in ESP-IDF framework. For looping it is just used while cycle.

```
xTaskCreate(&led_controller, "led_controller", 2048, NULL, 5, NULL);
xTaskCreate(&ota_example_task, "ota_example_tasks", 8192, NULL, 5, NULL);
```

Figure 17: ESP-IDF tasks

For using function, it is needed to create tasks in ESP-IDF and Sming. These are needed because there is no loop function as in Arduino core. As shown on figure 17 there are two tasks made, one is for making over-the-air update and second to control the led.

```
// wait for WiFi connection
// counter for not asking update every update
if ((WiFiMulti.run() == WL_CONNECTED)&& counter == 500000) {
  counter = 0;
  WiFiClient client;
  // calls update function
  t_httpUpdate_return ret = ESPhttpUpdate.update(client, "http://192.168.1.195:5000/esp8266");

  switch (ret) {
    case HTTP_UPDATE_FAILED:
      USE_SERIAL.printf("HTTP_UPDATE_FAILD Error (%d): %s\n", ESPhttpUpdate.getLastError(), ESPhttpUpdate.getLastErrorString().c_str());
      break;

    case HTTP_UPDATE_NO_UPDATES:
      USE_SERIAL.println("HTTP_UPDATE_NO_UPDATES");
      break;

    case HTTP_UPDATE_OK:
      USE_SERIAL.println("HTTP_UPDATE_OK");
      break;
  }
}
```

Figure 18: Code snippet from Http Update

Figure 18 shows code from Http update, where the update function is called. Wi-Fi credentials are set in setup(). For not asking every loop time there is a set counter. There is, if statement to check Wi-Fi connection and then update function is called from given path. There is a set counter to control update checking. The author chose a random time, but this should be configured depending on the built system.

4.2 Firmware server

Some over-the-air update solutions are using a server. For that, thesis author used a server written in python using Flask. Server also has a simple web page where it is possible to upload a compiled binary file.

Figure 19: Simple upload page

Figure 19 shows simple upload page, which can be used for uploading files to server. Files will be saved into project source tree. Files are named after selection made on web page. Uploading overwrites the old file. After successful uploading user will be directed to success page.

```
# Sends update to the board.
@app.route('/<updatetype>', methods=["GET"])
def send_update(updatetype):
    try:
        client_md5 = request.headers['X-Esp8266-Sketch-Md5']
    except:
        client_md5 = request.headers['X-Esp32-Sketch-Md5']
    local_md5 = hashlib.md5(open(updatetype + '.bin', 'r+b').read()).hexdigest()
    if local_md5 == client_md5:
        return "Update not needed", 304
    return send_file(updatetype + ".bin")
```

Figure 20: Http update server side

Http update over-the-air update server side is shown on figure 20. Function checks if there are sent sketch md5 in request headers. If file is found in file tree, then md5 is calculated. Results of md5 are compared. When these results differ, the update will be sent to the development board.

```
# ESP32 FOTA json
@app.route('/json')
def fota_json():
    return jsonify(
        type="esp32",
        version=2,
        host="192.168.1.195",
        port=5000,
        bin="/esp32/fota"
    )
```

Figure 21: FOTA json

```
@app.route('/esp32/fota', methods=["GET"])
def send_fota():
    file_path = "esp32_fota.bin"
    file_size = os.path.getsize(file_path)
    fh = open(file_path, 'rb')
    return Response(fh,
                    mimetype='application/octet-stream',
                    headers=[
                        ('Content-Length', str(file_size)),
                        ('Content-Type', 'application/octet-stream')
                    ])
```

Figure 22: Fota update

Figure 21 serves json data, what is needed for FOTA update. New firmware is looked from given host and port when version is changed. Bin field is used for the path in server. Figure 22 shows bin file sending to development board. For sending update content type must be application/octet-stream and header must contain content-length. For Sming and ESP-IDF framework binary files are set into static folder. Flask serves static folder, so the development board can download update. These files must be updated manually in server.

4.3 Framework Comparison

In this subsection is compared results, when using over-the-air update.
The goal of comparison was to measure download and boot time for each OTA solution. These results can be used to compare how much framework is spending for downloading and booting. Results can be used to compare two different development boards. The author also measured downtime, which is the time when the device is not responding. Since each over-the-air solution has own file, then these file sizes can be used for comparison.

For measuring time was used serial monitor. Depending on the framework there was message, which let know that download is started or completed. Validating results there was used server log and web browser network monitor.

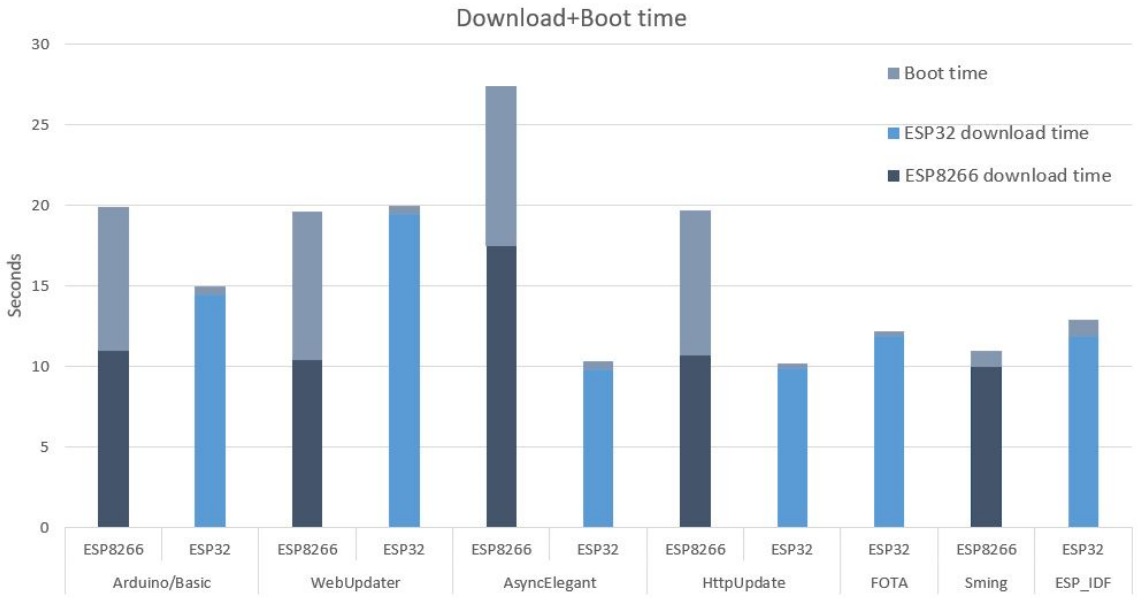

Figure 23: OTA update time

Figure 23 shows uploading firmware using different OTA solutions and two different ESP boards. As shown on chart ESP32 board is around 8 seconds faster on boot than ESP8266 board. ESP32 booting time is below one second and ESP8266 is around nine seconds. Fastest update timings has ESP32 using Http update. Booting time for ESP8266 with Sming framework is much faster. This is because of the rBoot bootloader used in Sming.

As it is possible to see Basic OTA and WebUpdater binary file downloading takes more time on ESP32. These solutions may not be good for using with ESP32. Http update download time is almost equal.

Device downtime depends on the framework and code that it is written in. Some solutions like web server wait for the Internet connection before starting the task. This can extend downtime for a couple of seconds. There are some frameworks and solutions that allow to run task background when update is downloaded. These are Sming, ESP-IDF and AsyncElegant solution.

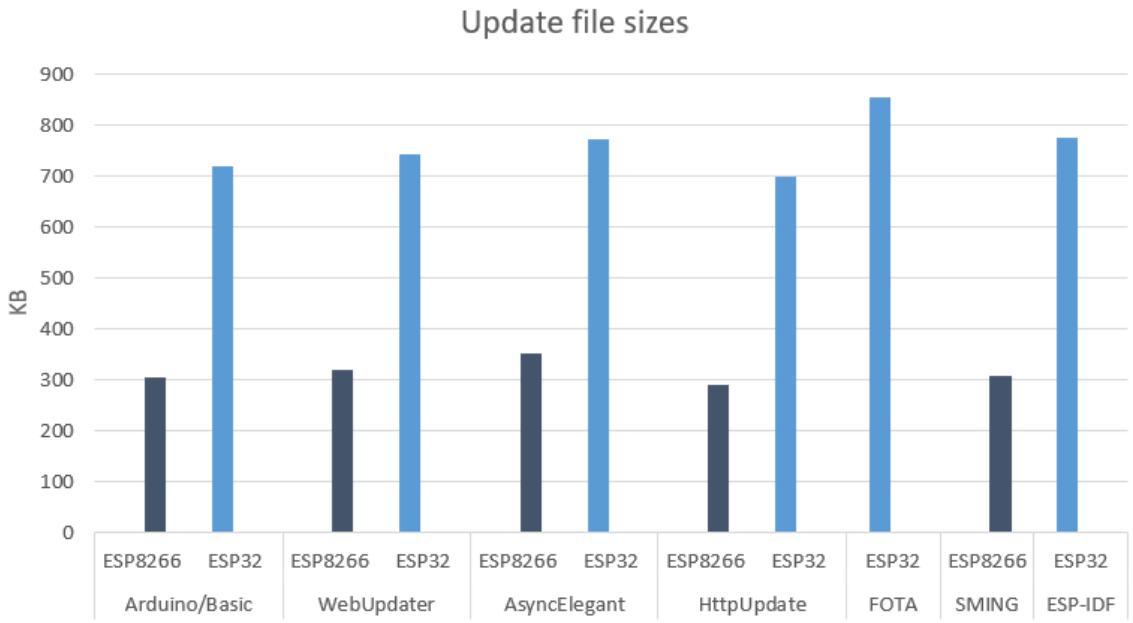

Figure 24: Update file sizes

Figure 24 shows file sizes in kilobytes. As it is possible to see from the charts ESP32 updates are twice the size as ESP8266. This might be because ESP32 library is bigger. It is possible to see that the smallest files for both boards are using Http update. This is because the update file handling is in the server and the development board's job is only asking for the update. Comparing ESP32 updates, it is possible to see that the biggest file belongs to FOTA update. The size cap between FOTA and Http update is over a hundred kilobytes. The difference between solutions is that FOTA checks version number on server. The author would have expected that the largest file would be for web server update, which is true for ESP8266.

4.4 Discussion and Limitations of the system

The author finds that the best framework to work with is Arduino. This allows to use Arduino IDE, which is easy to use. There are many examples and libraries for Arduino, which can be used to code ESP board. Probably this is the easiest framework to start with, since it has a better documentation and an easier code layout. Depending on the project it might be useful to use Sming or ESP-IDF. In the beginning, Sming seemed easy to use and have enough documentation, but it was more complicated than expected. For OTA updates right slots need to be chosen for board to continuously work. The author thinks Sming was more complicated than other two frameworks. Configuring the board was hard and seemed like there is some documentation missing. As for ESP-IDF framework the set up was easy because of the installer. Configuring device was better to understood because there was simple UI to set values.

Coding in Arduino IDE felt more convenient. Compared to the other frameworks there was no need to set separate tasks. Using Sming it was possible to use similar functions to Arduino, but for ESP-IDF analog reading and led configuring was more different. As Sming and Arduino set Wi-Fi credentials and endpoint from code, ESP-IDF used configuring files instead.

These over-the-air solutions are mainly designed to do update over Wi-Fi. Probably frameworks can also be used with different connections, but then they might need specific configuring. For example, ESP-IDF framework can be configured to using Ethernet.

Both boards have limitations for how many pins it is possible to use. ESP32 has more pins but using Wi-Fi chip, it adds some limitations for using analog pins.

These Wi-Fi chips only support 2.4GHz Wi-Fi, so it is not possible to use these in 5GHz network.

5. Conclusion and Future work

As a result of the thesis, there has been tested three over-the-air frameworks and two different generation ESP boards, ESP32 and ESP8266, have been tested. The ones that were tested are Sming, Arduino and ESP-IDF frameworks. There are multiple solutions for OTA, but each solution is little bit different. Some update solutions are using web server, IDE or server for sending updates. For comparing updates, there are made timings for the updating. This gives a chance to analyze which solution would be the fastest and provide better downtime. For longest download and boot time is Arduino framework AsyncElegant solution for ESP8266 with about 27 seconds. The fastest upload is Http update with about 10 seconds using ESP32. But this is not the best result for downtime. The best downtime depending on configuration can have AsyncElegant using ESP32 and Sming framework for ESP8266. There is also a downside using Sming as configuring framework can be quite time-consuming. The easiest framework to work with is Arduino Core using IDE. There is simple to add libraries, compile and flash code to board. Thesis results can be used for choosing the development board and framework.

5.1 Future work

Since systems are always improving, there might be new boards and frameworks to be tested.

Nowadays security is important and for production it is needed to add some security layer, as security can influence the over-the-air. It would be nice to know which solutions are faster and more optimized.

There are many different boards which can be used for OTA. In this thesis I used two ESP development boards. In general there are many boards that can be tested.

6. References

[1] “18 Most Popular IoT Devices in,” [Online]. Available: https://www.softwaretestinghelp.com/iot-devices/. [Accessed 3 May 2020].

[2] T.Quek, “The advantages and disadvantages of Internet Of Things,” [Online]. Available: https://www.linkedin.com/pulse/advantages-disadvantages-internet-things-iot-tommy-quek/. [Accessed 7 May 2020].

[3] M. B. A. a. E. V. Quesada, “Predictable Influence of IoT in the Higher Education,” *International Journal of Information and Education Technology,* vol. 7, 2017.

[4] F. G. F. X. P.-B. a. M. I. A. Andreas Kamilaris, “A Semantic Framework for Internet of Things-enabled Smart Farming Applications,” 2016.

[5] F. G. M. S. F. E.-M. Daniel Bastos, “GDPR Privacy Implications for the Internet of Things,” 2018.

[6] A. Bakolia, “Programming NodeMCU ESP8266 Over-the-Air using Arduino IDE,” 17 May 2019. [Online]. Available: https://circuitdigest.com/microcontroller-projects/esp8266-ota-update-programming-using-arduino-ide. [Accessed 7 May 2020].

[7] “Over the Air Updates,” [Online]. Available: https://www.freertos.org/ota/index.html. [Accessed 24 April 2020].

[8] A. R. Gonzales, “barbara,” 23 January 2019. [Online]. Available: https://barbaraiot.com/articles/importance-ota-updates-iot-devices/. [Accessed 30 April 2020].

[9] J. Eiden, “Why Intelligent OTA Firmware Updates Are Critical For IoT Products,” Particle, 9 May 2019. [Online]. Available: https://blog.particle.io/ota-firmware-updates/. [Accessed 30 April 2020].

[10] E. B. T. E. a. K. S. FJ Acosta Padilla, “The Future of IoT Software Must be Updated,” 21 September 2016.

[11] “Arduino ESP8266,” [Online]. Available: https://arduino-esp8266.readthedocs.io/en/latest/ota_updates/readme.html#arduino-ide. [Accessed 25 April 2020].

[12] O. Ugus, “Secure and Reliable Remote Programming in Wireless Sensor,” 2013.

[13] “TinyOS,” [Online]. Available: http://tinyos.stanford.edu/tinyos-wiki/index.php/FAQ. [Accessed 3 May 2020].

[14] A. Nugroho, “Development of a Field Environmental Monitoring Node with Over the Air Update Function,” 2015.

[15] “Particle,” Particle, [Online]. Available: https://docs.particle.io. [Accessed 3 May 2020].

[16] W. F. M. E. K. Mansour, “AiroDiag: A Sophisticated Tool that Diagnoses and Updates Vehicles Software Over Air”.

[17] “Introduction,” [Online]. Available: https://www.nongnu.org/avrdude/user-manual/avrdude_1.html#Introduction. [Accessed 3 May 2020].

[18] D. Frisch, “An Over the Air Update Mechanism for ESP8266 Microcontrollers,” 2017.

[19] “MQTT,” [Online]. Available: mqtt.org. [Accessed 3 May 2020].

[20] “Sming,” [Online]. Available: https://sminghub.github.io/Sming/about. [Accessed 7 May 2020].

[21] “8 Reasons Why ESP8266 has Been So Popular,” TechDesign, 6 February 2017. [Online]. Available: https://blog.techdesign.com/8-reasons-esp8266-popular/. [Accessed 7 May 2020].

[22] “D1,” WEMOS Electronics, [Online]. Available: https://wiki.wemos.cc/products:d1:d1. [Accessed 3 May 2020].

[23] “ESP32 pinout,” [Online]. Available: https://randomnerdtutorials.com/esp32-pinout-reference-gpios/. [Accessed 3 May 2020].

[24] “Introduction,” Arduino, [Online]. Available: https://www.arduino.cc/en/Guide/Introduction. [Accessed 3 May 2020].

[25] “Frequently Asked Questions,” Arduino, [Online]. Available: https://www.arduino.cc/en/Main/FAQ. [Accessed 3 May 2020].

[26] “setup,” Arduino, [Online]. Available: https://www.arduino.cc/reference/en/language/structure/sketch/setup/. [Accessed 3 May 2020].

[27] “loop,” Arduino, [Online]. Available: https://www.arduino.cc/reference/en/language/structure/sketch/loop/. [Accessed 3 May 2020].

[28] “Flask,” [Online]. Available: https://flask.palletsprojects.com/en/1.1.x/foreword/. [Accessed 4 May 2020].

[29] “MD5,” [Online]. Available: https://www.lifewire.com/what-is-md5-2625937. [Accessed 3 May 2020].

[30] “ESP-IDF Programming Guide,” Espressif, [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/index.html. [Accessed 3 May 2020].

[31] “ESP-IDF OTA,” Espressif, [Online]. Available: https://github.com/espressif/esp-idf/blob/master/examples/system/ota/README.md. [Accessed 3 May 2020].

Appendix

I. Code Repository

Development board codes and server for testing can be found from repository:

https://github.com/0lla/ESP32-and-ESP8226-OTA

II. License

Non-exclusive licence to reproduce thesis and make thesis public

I, Oliver Ossip,

(author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

A comparison of Over-the-Air Update approaches for the ESP32 and ESP8266 Development Boards,

(title of thesis)

supervised by Jakob Mass.

(supervisor's name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Oliver Ossip

08/05/2020