1 Introduction ESP8266

ESP8266 is a microcontroller designed by Espressif System in 2014 as a simple WiFi to serial converter with full TCP/IP stack. Originally, the device was shipped with preloaded firmware that provide a set of AT commands that enabled operations such as connection to WiFi network, creating AP, providing DHCP, creating TCP server, etc. ESP8266 was usually connected with Arduino or any other similar device thru the UART interface.

Since the beginning, a lot of effort has been centered to load programs directly onto the chip, the biggest problem was the absence of any English written documentation from the manufacturer.

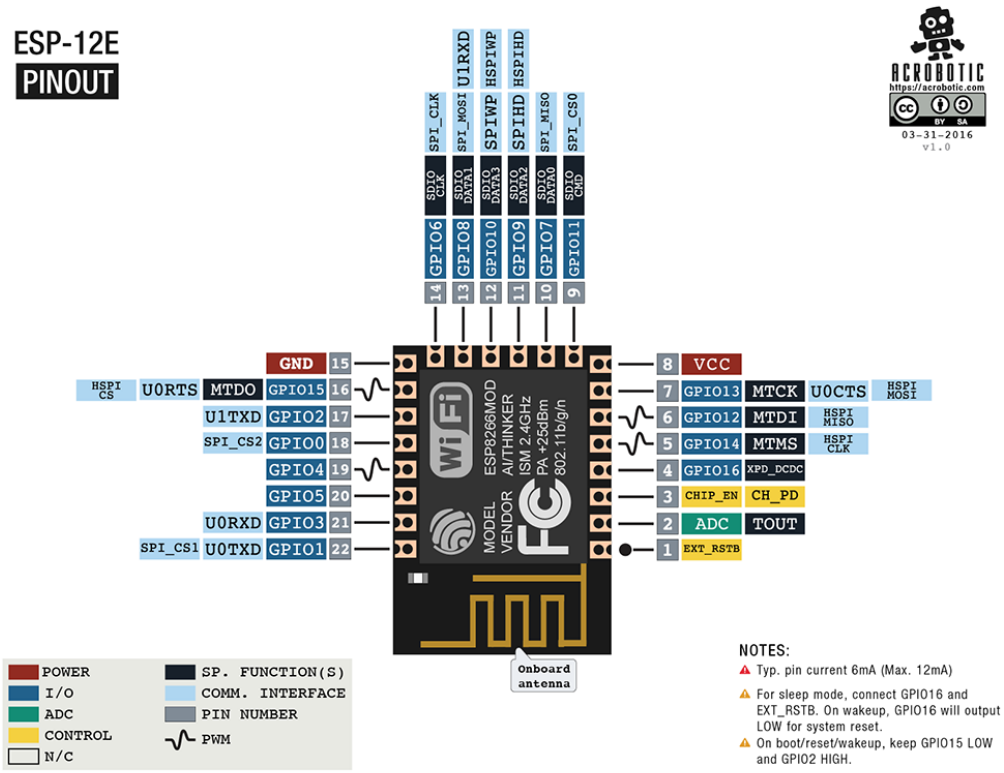

Figure 1: ESP8266 pinout

1.1 Introduction EspHub

With low cost and affordability, the ESP8266 is ideal home automation solutions. Modules with a wide range of sensors can motitor for example temperature, humidity, CO2 level, aquarium water temperature and many more. The measured data needs to be collected, stored and visualized, preferably on the display connected to the ESP or in the nice looking web interface.

Imagine buying new ESP and wanting to build an awsome WiFi driven aquarium in which you will monitor the temperature, water level and remotely turn on and off lighting. The hardest part of this project is how to send and receive data from your aquarium to your home server or cloud. A tough question is also how to enter your WiFi network credentials, hardcoding can be little bit clumsy. EspHub aims is significantly simplify this procedure.

After you write the Arduino like code for your ESP, that scans temperature, water level and switches light, you can simply import the Arduino EspHubLib library and declare that your "aquar-

ium" will provide temperature, humidity and light switch ability. For full functionality, you will also need your own home automation server EspHubServer and Mosquito MQTT broker, which you can run on Raspberry Pi or other computer from fruit familly.

After you run the aquarium device, EspHubLib create a captive portal when you simply enter the credentials to your WiFi network and then the auto discrovery procedure handled by EspHubServer finds all new divices and notifies you in the web interface. Once you approve your device and adjust its parameters, like name, name of provided values and units of provided values, you can monitor status of the device in a web interface.

For better control of your devices, you can connect the display screen to ESP and create display module (currently only ILI9341 240x320 display is supported). All you have to do is properly connect the display to ESP and announce the display ability in Arduino EspHubLib. In EspHubServer, you can then set which values from which sensor device will be displayed on screen. The server will periodically generate nice-looking charts and send them over a WiFi connection to the display.

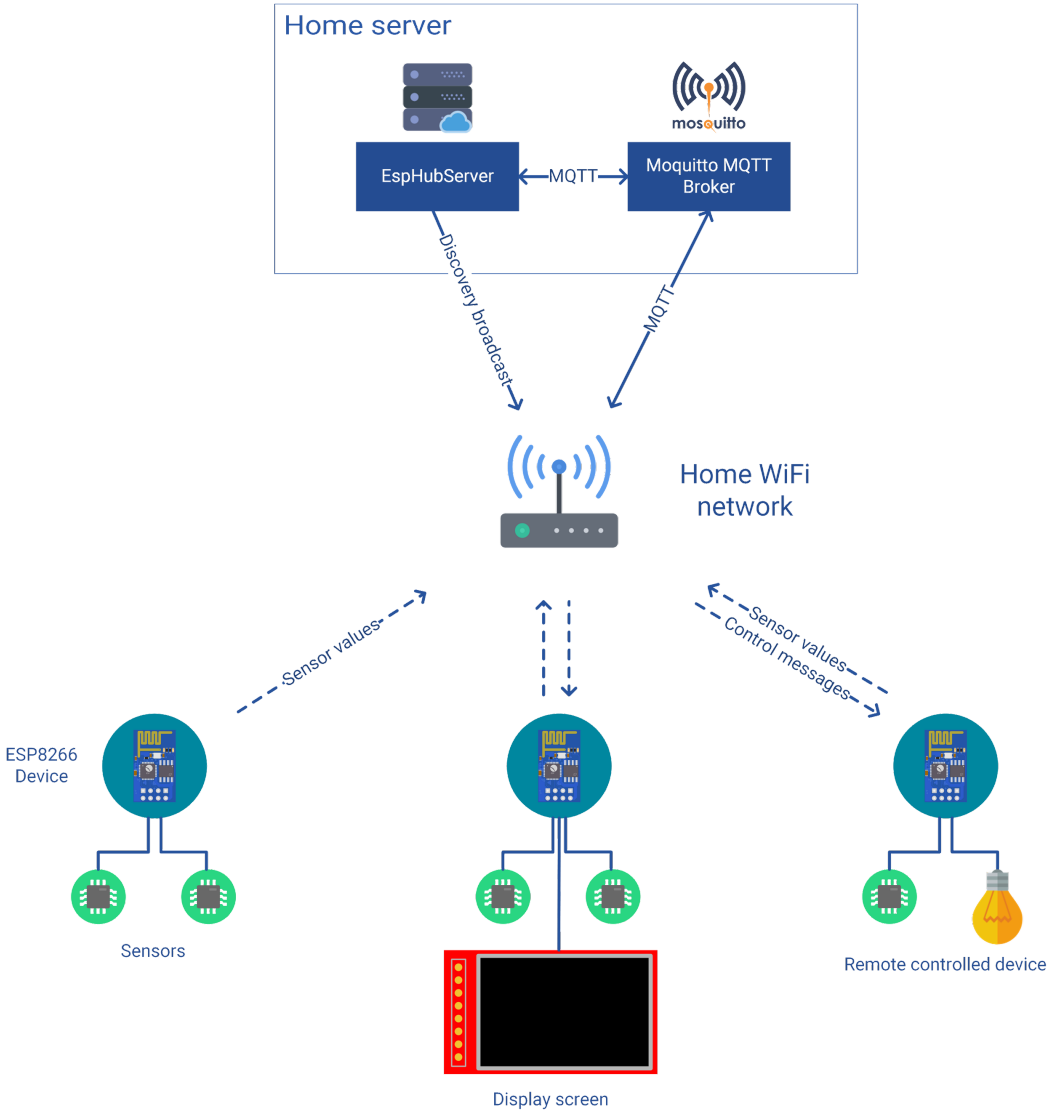

Figure 2: EspHub function overview[1]

2 ESP8266 development options

From manufacturer are ESP8266 modules usually shipped with pre uploaded AT firmware, this firmware provide a set of basic command which allows you to configure WiFi connection, create AP, manipulate with IP address, get network status, create TCP server or client, etc.. Since the beginning, a lot of effort has been centered to load programs directly onto the chip, the biggest problem was the absence of any English written documentation from the manufacturer.

Fortunately, after the growing interest about this module, manufacturer released official SDK tool, for development in C language. There are two version of SDK, non OS version and OS version based on FreeRTOS realtime operation system. None of them provides an easy way to develop application, so the community has decided to create tools that make development easier for a wider group of users. Probably the most popular is currently the ESP8266 Core for Arduino platform.

2.1 Official SDK

This SDK is developed and maintained directly by the Espressif System manufacturer. It is provided in two versions of Non-OS SDK and RTOS SDK which is based on FreeRTOS system. Non-OS SDK uses timers and callbacks as the main way to perform various functions, such as nested events and functions triggered by certain conditions. At this point, the Non-OS version is much better supported by the manufacturer that publishes more frequent releases than for RTOS version.

RTOS provide environment with preemptive multithreading. You can use standard interfaces from FreeRTOS to realize resource management, recycling operations, execution delays, inter-task messaging and synchronization, and other task-oriented process design approaches. there is also option to use standard BSD socket API to develop applications[4]. RTOS version include cJSON library to easy parse JSON packets and many other libraries for access to integrated peripheries. on the other hand preemptive multithreading could be little bit dangerous especially due to hardware limitation on ESP8266 and third-party libraries that may not be thread safe.

Espressif also provide complete development toolkit including hardware development modules.

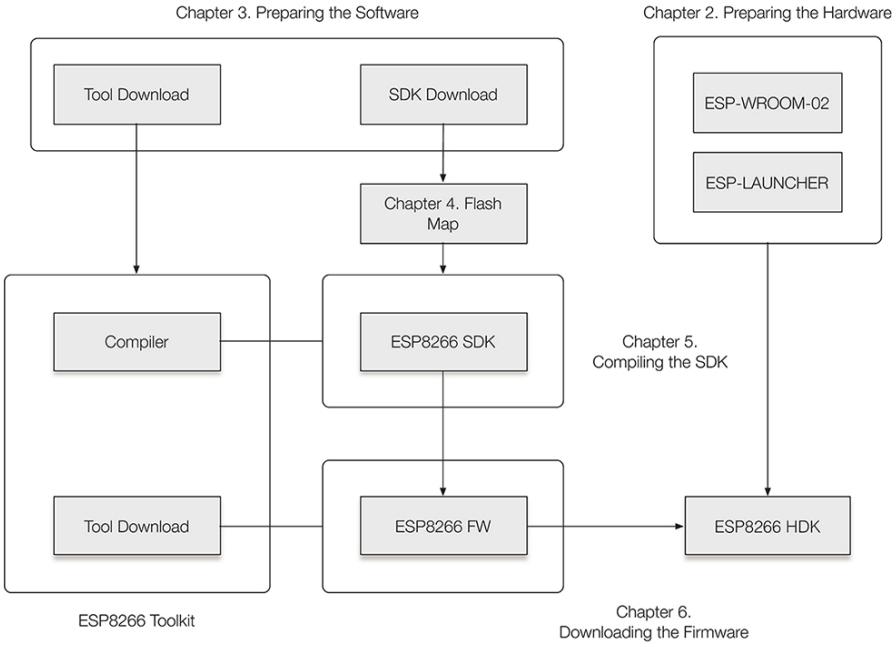

Figure 3: Official SDK compilation process overview

2.2 Community SDK

An official SDK is not released under an open source license, and contains binary blobs with unknown internal implementation, due to this and other reasons community developed few open source alternatives.

First and one of the most popular is **Esp-open-sdk** based on GCC toolchain and currently developed by Paul Sokolovsky[5]. Another way is using open source toolchain for Xtensa processors **gcc-xtensa**, maintened by Max Filippov[6]. Also worth mentioning is the unofficial development kit **esp8266-devkit** from Mikhail Grigorev[7].

2.3 Arduino Core

Probably most favourite development options is using Arduino Core for ESP8266. This project is maintained by community and brings ESP8266 SDK into Arduino IDE, this make possible to compile Arduino like code to ESP platform. It was quite a challenge at begining, because the AVR architecture (used by most of Arduino boards) is incompatible in some ways with Xtensa processors, but thanks to the great effort of the developer and the intense development, most of the arduino libraries are compatible with ESP8266 at this time[**?**].

ESP8266 Core is available from Arduino IDE board manager and also PlatformIO development platform is supported[9].

2.4 NodeMCU Lua firmware

NodeMCU is Lua implementation for ESP8266, originally was developed with custom board, but hardware development is already discontinued. Lua firmware is still maintained by community and support wide range of sensors and libraries. Firmware image could by build in nice web interface[10]. This embedded version of Lua is even based driven and thanks to the use of spiffs file system, it allows easy access to the ESP flash memory.

Unfortunately there is some problems with software reliability and insufition of memory on ESP8266, due to this reasons is this platform targeted mainly for experimental purpose[11].

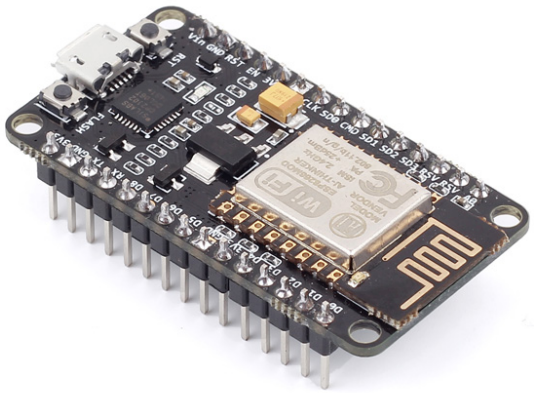

Figure 4: NodeMCU module with preloaded Lua firmware[12]

2.5 Other

MicroPython Lightweight implementation of python for embedded devices. Build for ESP8266 is relatively stable at this time and support most of ESP8266 features[13].

Espruino Development platform based on JavaScript. A very useful feature is that you can program ESP directly from an internet browser using JavaScript code[14].

Mongoose OS Platform with strong integration with Amazon AWS IoT services, coded in C or JavaScript, also support coding directly from web browser[15].

Sming C++ framework based on Non-OS SDK, which integrate many useful libraries and make programing in SDK much more easier[16].

EspEasy High level firmware for ESP which allow configuration of ESP and many sensors thru the GUI web interface[17].

3 Communication protocol

3.1 Introduction of MQTT protocol

EspHub uses a well-known MQTT protocol for communication with devices. MQTT (Message Queue Telemetry Transport) is lightweight TCP/IP protocol based on publisher-subscriber model.[2] This protocol is currently widely used in many IoT solutions, thanks to easy implementation and straightforward usability. As a basic communication protocol is used, for example in AWS IoT, Azure IoT HUB, Facebook Messagner, Domoticz and many other platforms.

Message and topic

Each MQTT message consists of a message topic and message content. Topics are based on logical hierarchical structure, for example bulding-1/office-A154/light-sensor. The message is any string and both are encoded by UTF-8. The topic is not required to registre in advance, MQTT broker will register topic automatically when first message from Publisher arrives.

Each device may request a subscription of any topic, in which case it becomes a Subscriber. The device can subscribe a specific topic, or it can use wildcard. For example device want to get all messages from sensor in office-A154, then it can use topic bulding-1/office-A154/+.

Character **+** represents one level in topic hierarchy, for example bulding-1/+/light-sensor means light sensors from all offices in bulding-1. Character **#** represents one and more level in topic hierarchy and must be placed as the last character, e.g. /building-1/# for subscribtion all sensors in all offices in bulding-1.

Message QoS

QoS is a major feature of MQTT, it makes communication in unreliable networks a lot easier because the protocol handles retransmission and guarantees the delivery of the message, regardless how unreliable the underlying transport is. QoS system i MQTT protocol is divided into 3 levels of reliability.

QoS level 0 The minimal level is zero and it guarantees a best effort delivery. A message won't be acknowledged by the receiver or stored and redelivered by the sender. Guarantee same level of reliability as underlying TCP connection. Is advantageous when sending large amounts of data at short intervals and we can afford message losses.

QoS level 1 It is guaranteed that a message will be delivered at least once to the receiver. But the message can also be delivered more than once.

QoS level 2 it guarantees that each message is received only once by the counterpart. It is the safest and also the slowest quality of service level. The guarantee is provided by two flows there and back between sender and receiver.

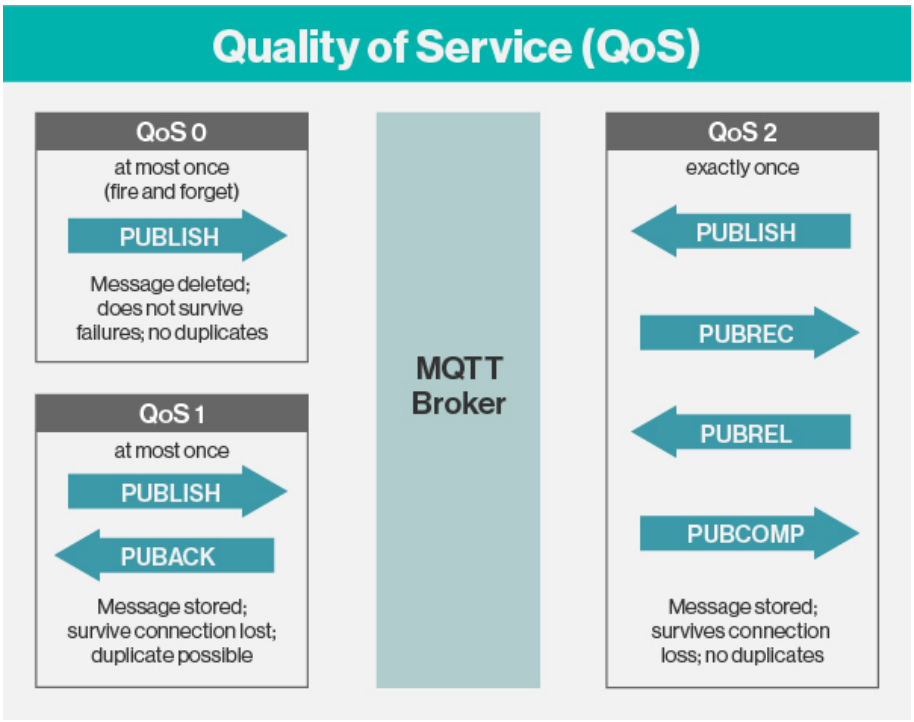

Figure 5: MQTT QoS model[3]

3.2 EspHub implementation

EspHub uses two protocols for communication with the devices, prefered protocol is MQTT. Devices send over MQTT verification, telemetry and data, the EspHubServer send validation and commands. Second protocol is UDP, server periodically send UDP packet on network broadcast address. The purpose of these packets is to find new devices added to the network.

When it connects to WiFi network, first verifies if it has stored information about parent server in flash memory. If the device has no information about server or if server is unavailable the device start listening on UDP broadcast address on port **11114**. Server distributes its name, address and port of MQTT server he listens to. When device receive UDP discovery packet, it try to connect to MQTT broker on given IP address and port and send Hello message in case of success, if MQTT broker is unavailable it go back to waiting mode.

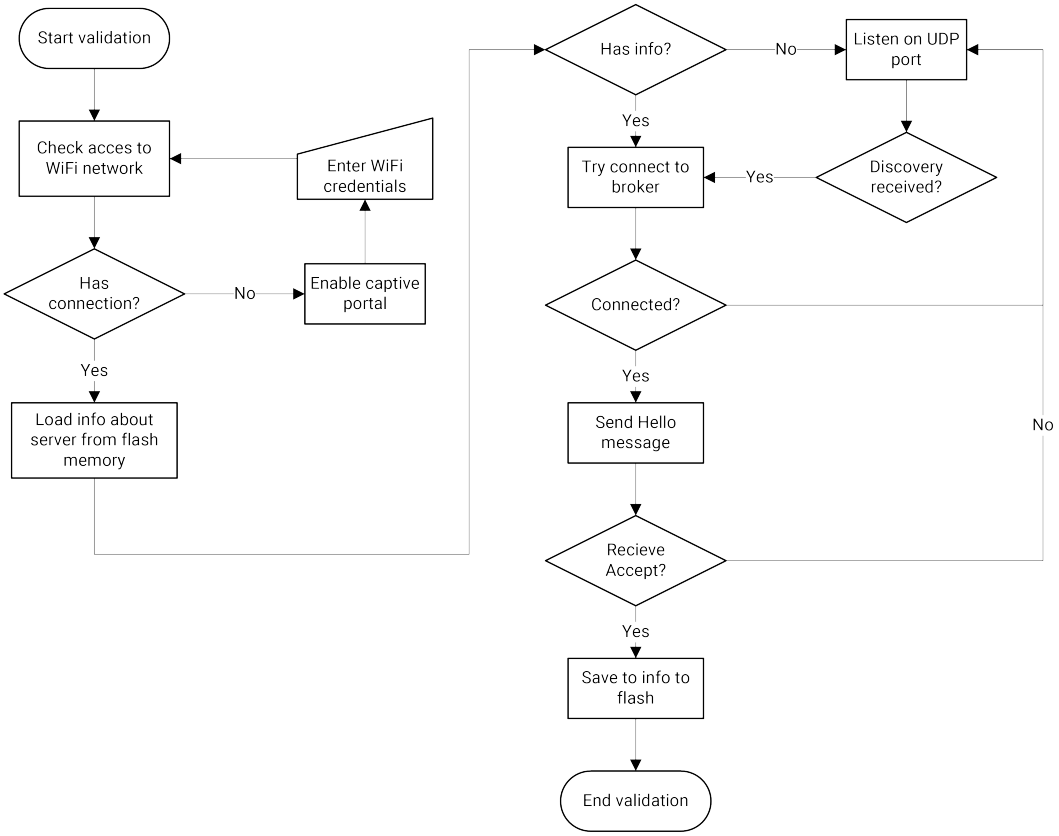

Figure 6: Validation process

4 Installation

4.1 Virtualenv

The best way to run EspHubServer is **virtualenv** it keeps order in installed python libraries and prevents collision with other python packages.

Step 0 Install virtualenv on your computer. The simplest way is to use PIP package manager. On Debian:

```
$ sudo apt-get install python-pip python3-dev
$ sudo pip install --upgrade virtualenv
```

Step 1 Download EspHub source.

```
$ cd ~   # change t oyour favourite working directory
$ git clone https://github.com/TanasVlachopulos/EspHubServer.git EspHubServer
$ cd EspHubServer
```

Step 2 Create virtualenv.

```
$ virtualenv -p python3 venv
# ...
# Installing setuptools, pip, wheel...done.
```

Step 3 Switch to virtualenv and install dependencies. EspHub use python settuptools to satisfy all dependencies, for instalation you can use PIP tool. It is necessary to use Python 3.4 or higher, you can check it with `python --version`.

```
$ source venv/bin/activate
(venv) $ pip install .
# ...
#   Running setup.py install for EspHubServer ... done
# Successfully installed Click-6.7 Django-1.11 EspHubServer-0.1
Pillow-4.1.0 configparser-3.5.0 olefile-0.44 paho-mqtt-1.2.3
pygal-2.3.1 pytz-2017.2 schedule-0.4.2
```

> Optionally you can install package **cairosvg**, which is necessary for use remote display feature. However, there are problems with cairosvg package on some computers, especially those with Windows system.
>
> ```
> (venv) $ pip install cairosvg
> ```

Step 4 EspHubServer provide simple CLI script **esphub**, to handle simple tasks.

```
(venv) $ esphub --help
(venv) $ esphub start --help
(venv) $ esphub start
```

Step 5 After all work you cant exit virtualenv context.

```
(venv) $ deactivate
```

References

[1] http://learn.acrobotic.com/tutorials/post/esp8266-getting-started

[2] https://mosquitto.org/man/mqtt-7.html

[3] http://internetofthingsagenda.techtarget.com/definition/MQTT-MQ-Telemetry-Transport

[4] https://espressif.com/en/support/explore/get-started/esp8266/getting-started-guide

[5] https://github.com/pfalcon/esp-open-sdk

[6] https://github.com/jcmvbkbc/gcc-xtensa

[7] https://github.com/CHERTS/esp8266-devkit

[8] https://github.com/esp8266/Arduino

[9] http://docs.platformio.org/en/latest/what-is-platformio.html

[10] https://nodemcu-build.com/index.php

[11] https://internetofhomethings.com/homethings/?p=424

[12] http://zdenekberan.eu/internet-veci-pro-zacatecniky-postavte-si-bezdratovy-teplomer-za-par-korun/

[13] https://learn.adafruit.com/building-and-running-micropython-on-the-esp8266/overview

[14] https://www.espruino.com/

[15] https://mongoose-os.com/

[16] https://github.com/SmingHub/Sming

[17] https://github.com/letscontrolit/ESPEasy