IoT Sensor and Actuator using NanoESP Boards

MICO-B Assignment

Florian Fritz
florian.fritz@st.oth-regensburg.de

Abstract

Our modern world is increasingly connected and automated through software systems. To allow these applications to interact with the real, physical world, hardware components have to be accessible over the network. This trend of internet connected hardware is generally referred to as the Internet of Things (IoT).
In this project we explore the space of internet connected devices by building a WiFi equipped sensor and actuator based on a NanoESP board. We implement a temperature and button sensor that communicates its status over the MQ telemetry transport (MQTT) protocol to both the actuator board as well as to an web interface.

Contents

1

Introduction

Our world is changing rapidly and technology is influencing more and more parts of our society. Instant communication over always connected smartphones, intelligent assistants in smart speakers, automated digital workflows, smart factories and much more make our modern economy possible. At the heart of all this are connected, intelligent software systems. However, none of these systems can work to its full potential without data about the real, physical world. A smart factory needs to monitor the condition of production plants, home automation software needs to read the living room temperature and switch lights on, logistic companies need to register the movement of goods.

The key observation is, that for a connected world, we not only need software, but also interfaces to interact with the physical world. More formally, we need to connect sensor and actuator hardware to our software components in the cloud. These physical *things* connected to the internet are often referred to as Internet of Things (IoT) [1]. With the *things* being accessible over the network, software can be easily used to orchestrate them to form more complex systems.

In our project we explore IoT applications by building a network connected sensor reporting temperature readings and button presses, as well as an actuator visualizing the collected data. To do so, we use two WiFi enabled NanoESP boards that communicate over MQTT, a common network protocol for IoT projects. We also integrate the home automation software openHAB into this sensor and actuator network to demonstrate the interaction between software and hardware components. Overall, the project demonstrates all essential parts needed for an IoT project, enabling us to use network connected hardware in future work.

2

Implementation

An IoT project inherently consists of multiple components communicating with each other over a network, making it important to understand the system as a whole before focusing on individual components. We therefore first introduce our project from a bird's-eyes view, followed by a detailed description of the network communication and the NanoESP boards.

2.1 System Overview

The overall goal of the project is to implement an internet-connected sensor, reporting temperature data and button presses, as well as an actuator, visualizing the current temperature and button status. Figure 2.1 shows the various components we choose to achieve this.

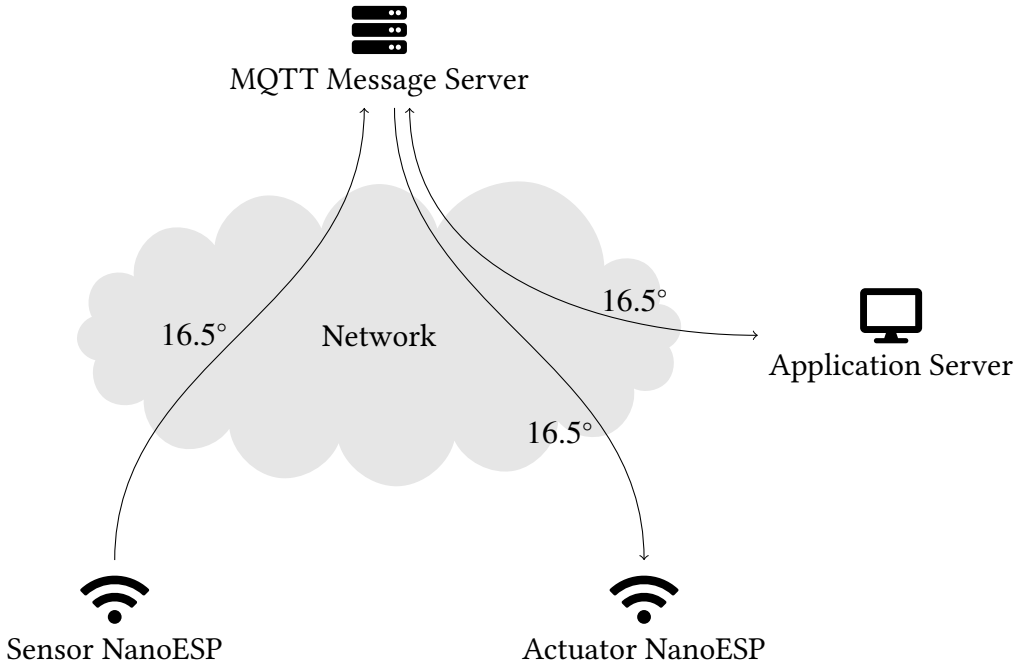


Figure 2.1: System Overview

On the bottom of Figure 2.1, we can see the sensor and actuator hardware, implemented using two NanoESP boards. These microcontrollers are equipped with a WiFi module, enabling them to connect to a local network. The sensor NanoESP uses this network to report its

temperature readings and button presses to a central message server, which in turn distributes these messages to the actuator as well as an application server. When receiving messages, the actuator NanoESP changes its status LEDs according to the last temperature reading and button event. The application server shown on the right is demonstrating the ability of our IoT devices to interact with other software components, enabling more complex applications in the future.

2.2 MQTT and OpenHAB

The communication between sensor, actuator and external application servers makes use of the MQ telemetry transport (MQTT) standard[2]–[4]. This lightweight messaging protocol is based on a server-client architecture implementing the publish/subscribe pattern [5], [6]. All devices connect to a central MQTT server, shown at the top of Figure 2.1, which is in charge of receiving and distributing messages among the communication partners.

Each MQTT message sent is associated with a topic describing its content, allowing the server to distinguish messages semantically. For example `sensors/temperature/garden` could describe a message containing the current temperature measured in the garden. As it makes no sense to forward each message to all connected devices, clients have to first subscribe to messages they are interested in. This subscription is done using a topic filter, e.g. `sensors/temperature/+` to subscribe to all messages related temperature sensors. This publish/subscribe patter based on a standard protocol decouples communication partners, making it simple to add new functions to our IoT project later on.

Topic	Description	Values
`rfmq/nanoesp/in`	Button Event from sensor to openHAB	ON, OFF
`rfmq/nanoesp/out`	Button Event from openHAB to LED	ON, OFF
`rfmq/nanoesp/temp`	Temperature Sensor	floating-point value

Table 2.1: MQTT message topics used in this project

Table 2.1 lists the topics used in our project. The sensor NanoESP periodically publishes its temperature readings at `rfmq/nanoesp/temp` and reports button presses on the `rfmq/nanoesp/in` topic. The actuator NanoESP subscribes to the `rfmq/nanoesp/temp` and `rfmq/nanoesp/out` topic and updates its status LEDs according to these messages.

Currently, we still miss the link between the `rfmq/nanoesp/in` and `rfmq/nanoesp/out` topic, as the actuator has to be notified on button presses. For this we use an application server, depicted on the right side of Figure 2.1. In our case, the application is the openHAB [7]

home automation software, that allows the user to integrate various IoT devices in automated workflows and online dashboards[8]. OpenHAB connects to the MQTT server and uses the sensor data to display a graph of the temperature readings and button status as shown in Figure 2.2. Additionally, the button state is forwarded to `rfmq/nanoesp/out` and the user can also trigger button changes in the web-interface.

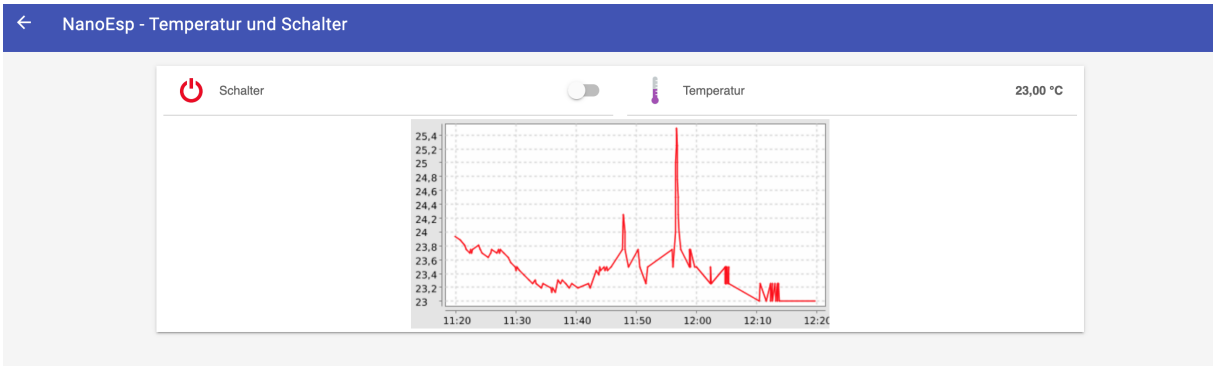


Figure 2.2: openHAB web interface

During our test setup both the MQTT severer and openHAB run on a local raspberry pi and all devices are connected to the same WiFi network. In a real world application the devices could reside in different networks, as long as they are connected to the internet, as MQTT works transparently over any network connection.
Lastly, we use an additional MQTT client on our smartphone [9] during development to manually inspect and send messages. This allows us to test individual components without the need to first integrate all devices.

2.3 NanoESP Sensor and Actuator

With the general setup and network communication covered, we now focus on the *things* in our IoT project. We first describe the used hardware components, followed by a discussion on the software implementation.

2.3.1 Hardware Components

The sensor and actuator are built using arduino compatible NanoESP boards[10], housing an ATmega328p and a ESP8266 microcontroller. The ATmega328 is programmable using the arduino IDE over the built in micro USB port, essentially resembling an arduino nano. The ESP8266 is solely used for WiFi functionality and controlled by the ATmega328 over a simple serial protocol. The board manifacturor provides a library [11] to send the correct

serial commands for TCP, HTTP and MQTT communication. The NanoESP connects these serial pins directly on the PCB and also has a small antenna etched onto it, thus requiring no further hardware to connect to a local WiFi network. As noted in the system overview, this direct WiFi connection allows us to directly send messages to the MQTT server without any additional gateway or base station hardware, making it simple to deploy the NanoESP board as a standalone hardware component.

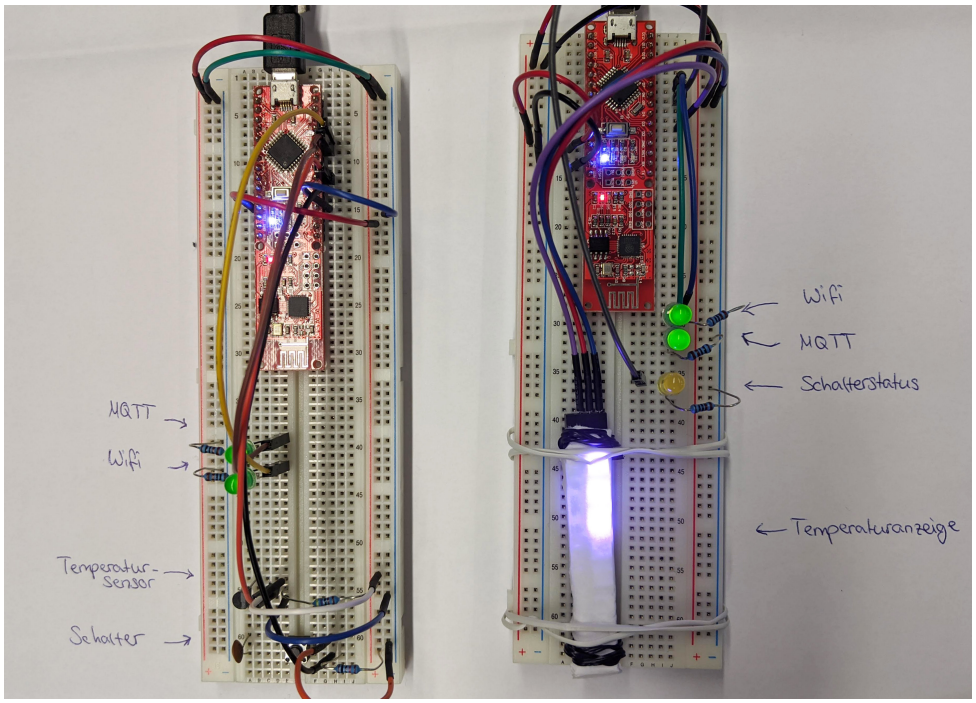


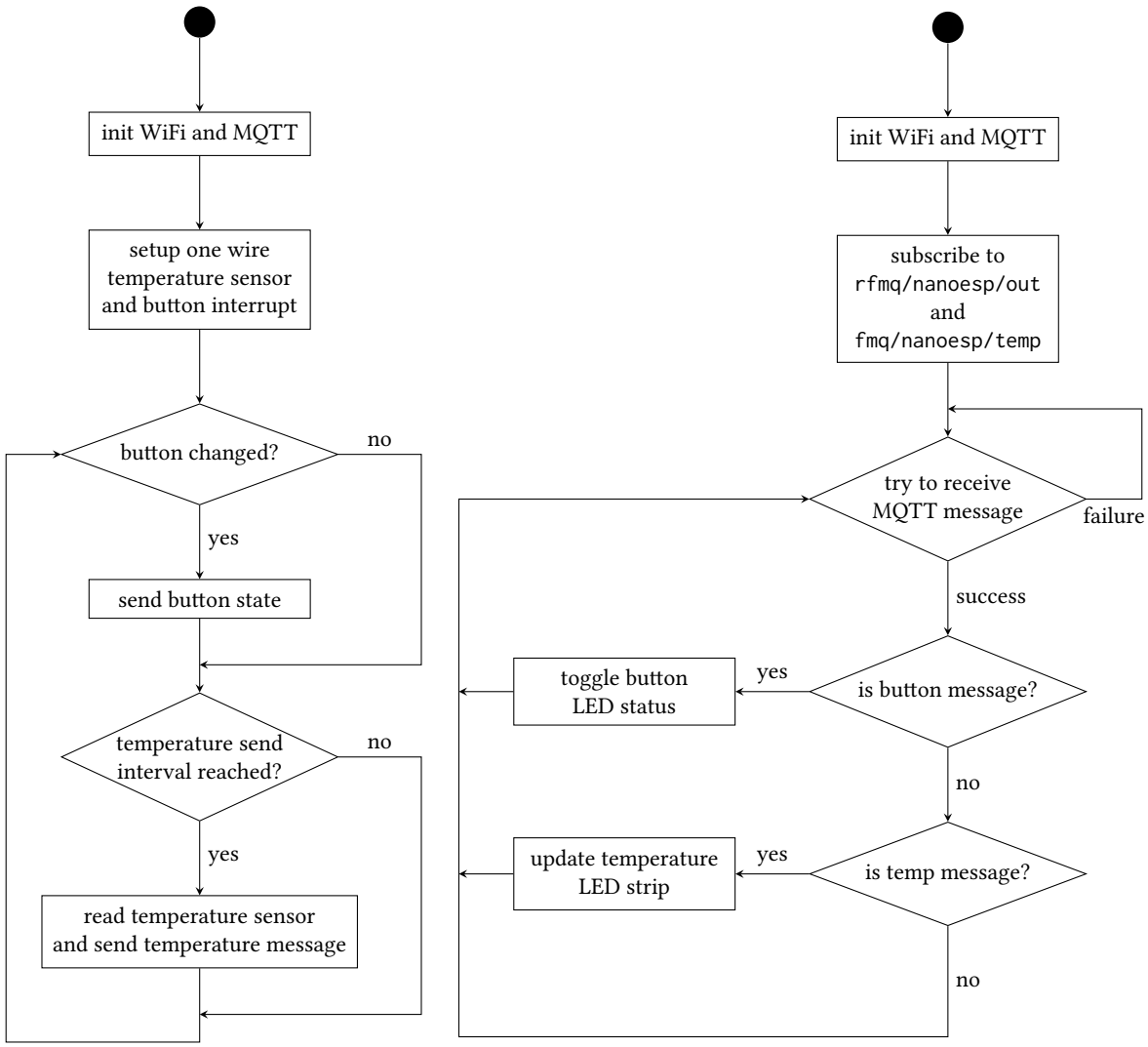
Figure 2.3: Hardware components: sensor NanoESP (left) and actuator NanoESP (right)

Figure 2.3 shows both the sensor NanoESP (left) and the actuator NanoESP (right) with all hardware components attached on a breadboard. Both have green status LEDs showing if the WiFi and MQTT connection setup was successful. The sensor board is connected to a DS18B20 [12] temperature sensor via an one wire bus, as well as to a simple button. The actuator board visualizes the current state of the button using a single yellow LED and the temperature using a WS2812 [13], [14] RGB LED strip. The temperature is visualized by changing the LED color from blue to red and by enabling more LEDs with increasing temperatures. Overall, the hardware components are rather simple, as the main focus lies on the communication between both controllers.

2.3.2 Software Implementation

To ease the software development process, we first implement each functionality (reading sensors, sending network messages, ...) in isolation and upon successful testing move the

functionality into a separate function. After all functions are implemented and tested we combine them to the final sensor and actuator software. To run different programs on the sensor and actuator NanoESP we use preprocessor macros to selectively activate features in the two program variants.



(a) Sensor **(b)** Actuator

Figure 2.4: Software Flowchart

Figure 2.4 shows flowcharts of the sensor (left) and the actuator (right). The actuator begins by connecting to the WiFi network and MQTT server using the NanoESP library[11]. After that it subscribes to all relevant message topics. Both tasks are done using simple library calls with the appropriate network and MQTT settings. With the initialization done, the actuator actively polls for incoming MQTT messages. Once a message is received, it checks whether its topic is related to an button event or temperature event and issues an update of the appropriate LEDs. The temperature display is done using a WS2812 [13], [14] RGB LED strip, which can be

easily controlled using a library provided by adafruit [15].

The sensor NanoESP also initializes the WiFi and MQTT connection first. After this the DS18B20 [12] temperature sensor is configured over a one wire bus as taught in the course. The last setup step is to register an interrupt handler for the button. The interrupt is registered on a falling signal, software debounced and sets a status flag whenever toggled. In the main loop the software checks if the button state has changed and sends a MQTT message to the `rfmq/nanoesp/in` topic using the vendor provided library[11] if necessary. We actively send the message in the main loop, as this potentially longer running action should not be performed in an interrupt handler. The temperature is sent in a fixed interval by checking in the main loop whether enough time has passed since the last temperature reading. If so, the sensor is read and the value is published using a MQTT message with topic `rfmq/nanoesp/temp`. We decided to use this polling time interval instead of an interrupts, as it is easier to implement and a slight shift in temperature reading times is not important for this application.

Overall, the software is rather simple, as it only forwards events from sensors or to actuators. The main focus during development was therefore on modularity and readability by splitting up logic into functions and by using a central place to define pinouts and other settings. For further details on the software we refer to the commented source code.

3

Conclusion

The project showed us how surprisingly simple it is to create internet connected devices. The process of connecting to the WiFi network and getting started with sending first MQTT is trivial using the vendor provided library and example. We also found the networking component to be very reliable, with loose cables being far more problematic than the WiFi connection. Overall the NanoESP is a great platform to get started with IoT development and allows for a variety of interesting projects.

For future work we would like to explore lower cost WiFi enabled boards. The NanoESP costs about 25€, making it very expensive for a simple sensor deployment. Part of the cost result form it combining both a programmable ATmega328p and a ESP8266 solely for WiFi communication. However, the ESP8266 can also be found cheaply on standalone boards for under 5€ and is very popular for cheap IoT projects. We plan to explore the MQTT functionality on the ESP8266 [16] in combination with its deep sleep mode [17] to build a battery powered sensor in the future.

Bibliography

[1] *Internet of things*, Page Version ID: 930565800, Dec. 2019. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Internet_of_things&oldid=930565800 (visited on 12/13/2019).

[2] *MQTT Version 3.1.1.* [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html (visited on 12/10/2019).

[3] *Publish & Subscribe - MQTT Essentials: Part 2*, Jan. 2015. [Online]. Available: https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe/ (visited on 12/11/2019).

[4] *Client, Broker / Server and Connection Establishment - MQTT Essentials: Part 3*, Jul. 2019. [Online]. Available: https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/ (visited on 12/11/2019).

[5] *Publish–subscribe pattern*, Dec. 2019. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Publish%E2%80%93subscribe_pattern&oldid=929177580 (visited on 12/10/2019).

[6] *What is Pub/Sub Messaging? How Pub/Sub Works (with Examples)*, Jul. 2019. [Online]. Available: https://blog.stackpath.com/pub-sub/ (visited on 12/10/2019).

[7] *openHAB.* [Online]. Available: https://www.openhab.org/ (visited on 12/11/2019).

[8] *openHAB Documentation.* [Online]. Available: https://www.openhab.org/docs/ (visited on 12/11/2019).

[9] instant:solutions OG, *MyMQTT – Apps on Google Play.* [Online]. Available: https://play.google.com/store/apps/details?id=at.tripwire.mqtt.client&hl=en_AU (visited on 12/11/2019).

[10] *The NanoESP Page.* [Online]. Available: https://iot.fkainka.de/en/nanoesp (visited on 12/10/2019).

[11] fk, *Die NanoESP Library*, Sep. 2016. [Online]. Available: https://iot.fkainka.de/library (visited on 12/10/2019).

[12] *DS18b20 Datasheet.* [Online]. Available: https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf (visited on 09/12/2019).

[13] *WS2812 Datasheet*. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/WS2812.pdf (visited on 09/12/2019).

[14] *Adafruit NeoPixel überguide*. [Online]. Available: https://learn.adafruit.com/adafruit-neopixel-uberguide/the-magic-of-neopixels (visited on 12/11/2019).

[15] *Adafruit/Adafruit_neopixel*. [Online]. Available: https://github.com/adafruit/Adafruit_NeoPixel (visited on 12/11/2019).

[16] Nuno Santos, *ESP8266: Connecting to MQTT broker*, Apr. 2017. [Online]. Available: https://techtutorialsx.com/2017/04/09/esp8266-connecting-to-mqtt-broker/ (visited on 12/13/2019).

[17] S. Santos, *ESP8266 Deep Sleep with Arduino IDE (NodeMCU)*, Jul. 2019. [Online]. Available: https://randomnerdtutorials.com/esp8266-deep-sleep-with-arduino-ide/ (visited on 12/13/2019).