Brezžična senzorska omrežja

Uredila dr. Miha Janež in prof. dr. Nikolaj Zimic

april 2022

Contents

Chapter 1
Publishing sensor readings to AWS IoT using MQTT

Matjaž Zupančič Muc and Luka Boljević

Abstract In this seminar an embedded application is developed. We utilize a test board to get temperature and pressure readings in the room the board is situated in. This is periodically measured using a BMP280 digital pressure sensor, and published to AWS IoT Core using the MQTT protocol. The board is connected to the computer via a USB mini cable, and connected to the internet via a WEMOS D1 Mini WiFi module. The measured data is displayed on AWS in whichever format, using the MQTT test client feature of AWS Console.

1.1 Introduction and motivation

The goal of this seminar is to transfer temperature and pressure measurements from an embedded device (**ESP8266**) running **FreeRTOS** [1] to a remote server (**AWS IoT Core** [2]) using **MQTT protocol** [3].

MQTT is a lightweight publish-subscribe network protocol that was designed to reduce network bandwidth requirements and is commonly used in applications when system resources are limited. Likewise, MQTT clients are very small, require minimal resources so they can be used on small microcontrollers (like ours is).

In practice, the ability to process and act on large volumes of data comes with lots of challenges. For this reason, AWS introduces **IoT services** and solutions to connect and manage billions of devices. It allows users to collect, store, analyze and act on data collected by an arbitrary number of devices. In this seminar, we will keep it simple and use their services to send and show temperature and pressure measurements using the MQTT protocol ”under the hood”.

1.2 Used libraries and modules

The hardware used in this seminar is rather simple. We use a test board provided during the conduct of the subject BSO [4]. The board has two components of importance for us:

- The first is the core component - this is the **WEMOS W1 Mini** module. Besides providing a connection to a computer via a micro USB connection, it also contains an **ESP8266 microcontroller**. ESP8266 is a low cost single core microcontroller equipped with a Wi-Fi module.
- The second component is the **GY-91 sensor module**. The module uses a combination of a **motion tracking module MPU-9250** (gyroscope, accelerometer and compass), and a **BMP280 barometric pressure sensor**. Apart from providing pressure measurements, the BMP280 sensor is able to give us temperature readings. The entire GY-91 module communicates with ESP8266 via the **I2C interface**.

The ESP8266 microcontroller is running **FreeRTOS**, a real-time operating system for microcontrollers. FreeRTOS is in charge of scheduling tasks that are executing at the same time. It is written mostly in C programming language, and is thus designed to be small, simple, and fast.

Pressure and temperature is read using `bmp280` library, that serves as an extra for FreeRTOS. The `bmp280.c` file provides the implementation for getting the readings from the BMP280 sensor, and it likewise implements **compensation algorithms**. These compensation algorithms are a *correcting* step, as they attempt to get rid of any noise in the readings. The compensation algorithms are provided by Bosch themselves, in their data sheet for BMP280. The BMP280 sensor can give temperature readings anywhere in the range of -40 to +85 degrees Celsius, while being able to provide pressure readings on altitudes between -500 and +9000 below/above sea level.

The `paho_mqtt_c` library, likewise provided as an extra for FreeRTOS, is used for MQTT communication. Namely, this library provides the framework so that we can make an MQTT client (our board), properly structure MQTT messages, connect, publish (send) and subscribe to (receive) messages.

1.3 Solution

1.3.1 Architecture

For reference, we used an open source GitHub repo [5]. The application running on the microcontroller is composed of three tasks: `wifi_task`, `read_sensor_task`, and `mqtt_task`. All three tasks have priority set to 2 (max priority is set to 15).

`wifi_task` is concerned with providing a WiFi connection to the board, so the measurements can actually be sent to AWS. We only have to specify the name and password of the wireless connection our computer is currently connected to, in a configuration file `private_ssid_config.h`.

`read_temp_task` is the one that utilizes the `bmp280` library to give temperature and pressure readings. It also structures and pushes (or attempts to push) the message to the queue so that it can be published later. We created the queue so that it can hold up to 3 messages at any one time. When a message is successfully sent to the queue, the microcontroller waits for any given time (we put 5 seconds) before performing new readings and doing the process all over again, indefinitely.

`mqtt_task` establishes a connection to the MQTT server on AWS, under a given endpoint, which looks like `<prefix>.iot.<region>.amazonaws.com`. After a successful connection, a client (our board) is created and registered to the server, after which the message, made during `read_temp_task` can be taken from the queue, and then published to AWS under the `esp8266/temp` topic (can be named anything).

1.3.2 Amazon Web Services (AWS)

1.3.2.1 General information

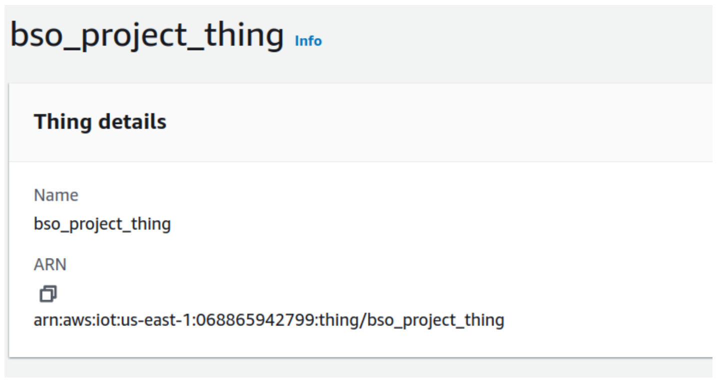
AWS IoT core allows for an easy connection of devices to the cloud. It supports HTTP, web-sockets and MQTT connections. It also provides mutual authentication and encryption at all points of connection. Therefore before data is exchanged, identity of a device has to be proven. Connections using MQTT use **certificate based authentication**, and any traffic from and to AWS IoT Core is encrypted over **TLS**.

1.3.2.2 Connecting a device

In order to connect a device to AWS IoT Core, there are some requirements that have to be met. The device has to first authenticate itself using a **certificate**. It also has to have the right **policy** attached to the certificate. Namely, the policy describes what operations the device (or so called *thing*, by AWS) can perform once it is authenticated. We had to take the following steps, before we successfully connected our device to AWS IoT Core.

1. Create a device (*thing*).
2. Create a private and public key, and certificate.
3. Create a policy.
4. Attach the policy to the certificate.
5. Attach the *thing* to the policy.

A device (*thing*) called `bso_project_thing` was created (1.1). Then, the following policy `bso_project_policy` was attached to it (1.2). Policy action *iot** (highlighted by a red square on 1.2) implies that this policy allows our device to perform all iot actions (*iot:Connect*, *iot:Subscribe*, *iot:Publish* , *iot:Receive*, ...). Highlighted policy resource * allows the device to access any AWS resource. An ECC based certificate was created locally and then signed by AWS. Certificate has `bso_project_policy` attached to it (1.3).

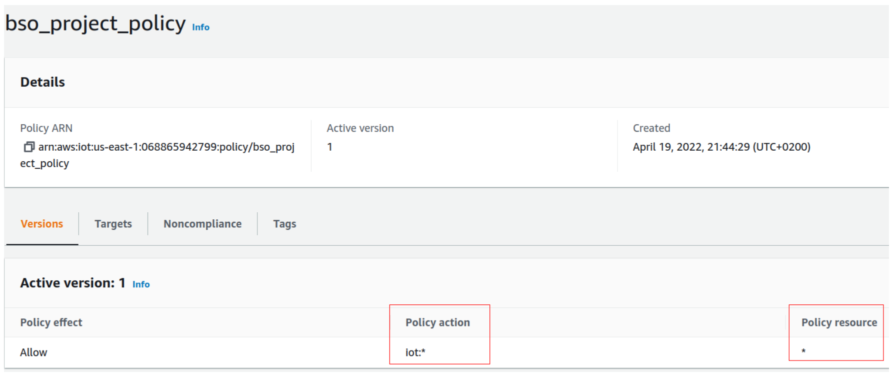

bso_project_thing Info

Thing details

Name
bso_project_thing

ARN
arn:aws:iot:us-east-1:068865942799:thing/bso_project_thing

Fig. 1.1: Created device (*thing*)


bso_project_policy Info

Details

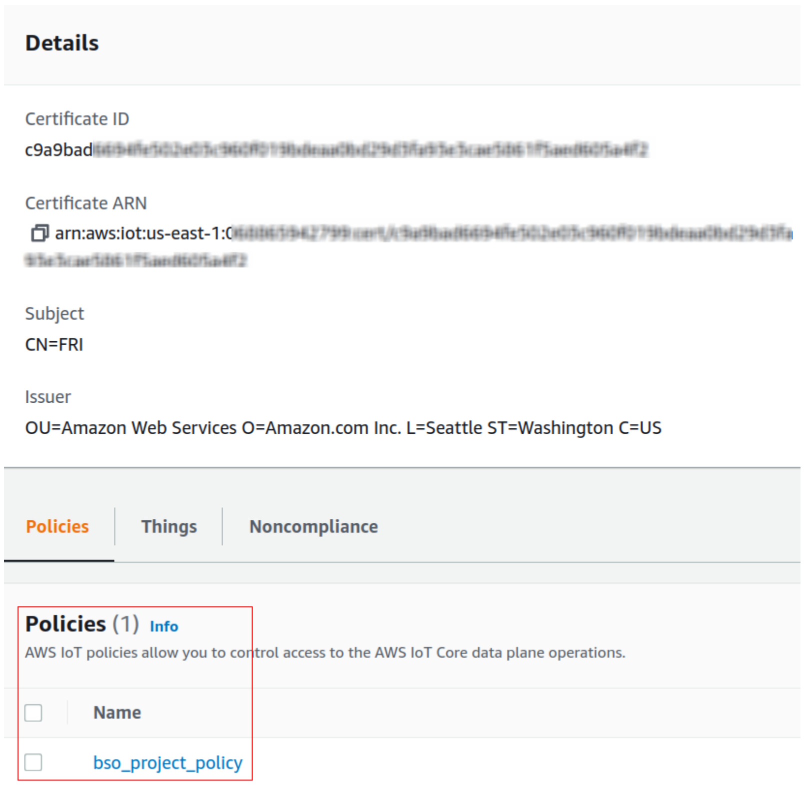
Policy ARN	Active version	Created
arn:aws:iot:us-east-1:068865942799:policy/bso_project_policy	1	April 19, 2022, 21:44:29 (UTC+0200)

Active version: 1 Info

Policy effect	Policy action	Policy resource
Allow	iot:*	*

Fig. 1.2: Created policy

Data is published to the `esp8266/temp` topic, as described before. Published data can be viewed through AWS Console using the MQTT test client feature.



Details

Certificate ID
c9a9bad[redacted]

Certificate ARN
arn:aws:iot:us-east-1:0[redacted]

Subject
CN=FRI

Issuer
OU=Amazon Web Services O=Amazon.com Inc. L=Seattle ST=Washington C=US

Policies | Things | Noncompliance

Policies (1) Info
AWS IoT policies allow you to control access to the AWS IoT Core data plane operations.

	Name
☐	bso_project_policy

Fig. 1.3: Certificate

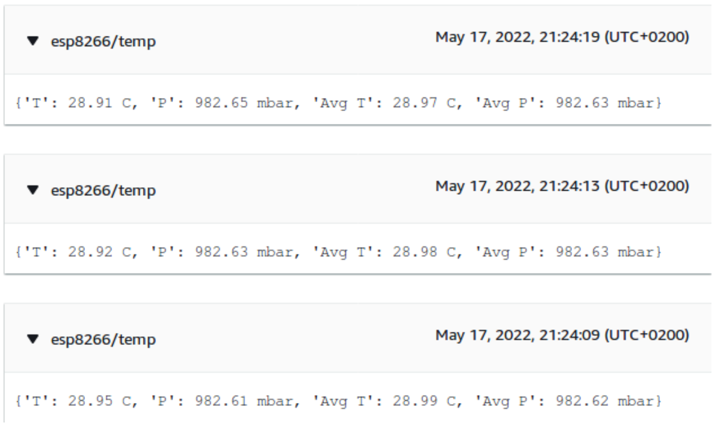
1.4 Results

We performed a few tests to check how sensitive the sensor is, and how quickly can it detect change in temperature and pressure. While temperature isn't a problem to test, measuring pressure presented is a slight challenge, as this seminar was done in the middle of May, when there is little to no rain or higher pressure outside. Still, we came up with a short one to check whether it can at least detect a slight change in altitude (and thus in pressure). The readings were sent to AWS every 5 seconds.

The tests were the following. First, we get baseline temperature and pressure readings. Then, we put the test board in the freezer to see how fast the temperatures drop. After that, the board is put inside an oven up to a relatively high temperature (so as to be 100% sure we have not damaged any parts). We take it out, let it stabilize a bit, and we take it one floor down (a difference of roughly 10 meters in altitude) to see whether it can detect the height difference.

1.4.1 The tests

Figure 1.4 shows the baseline reading inside a (apparently very warm) room inside the house. `'T'` denotes ”temperature”, `'P'` denotes ”pressure”, while `'Avg T'` and `'Avg P'` are the average readings (during the work time of the board) of those two metrics.



▼ esp8266/temp May 17, 2022, 21:24:19 (UTC+0200)

```
{'T': 28.91 C, 'P': 982.65 mbar, 'Avg T': 28.97 C, 'Avg P': 982.63 mbar}
```

▼ esp8266/temp May 17, 2022, 21:24:13 (UTC+0200)

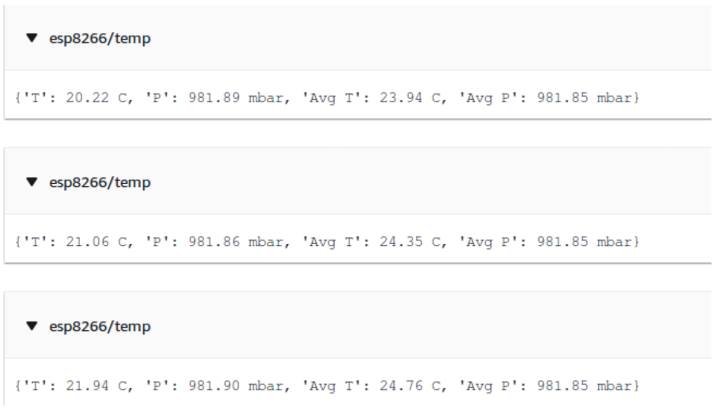
```
{'T': 28.92 C, 'P': 982.63 mbar, 'Avg T': 28.98 C, 'Avg P': 982.63 mbar}
```

▼ esp8266/temp May 17, 2022, 21:24:09 (UTC+0200)

```
{'T': 28.95 C, 'P': 982.61 mbar, 'Avg T': 28.99 C, 'Avg P': 982.62 mbar}
```

Fig. 1.4: Baseline readings

As aforementioned, the board is then put inside of the freezer. Notice how rather quickly the temperatures start to drop in figure 1.5.



▼ esp8266/temp

```
{'T': 20.22 C, 'P': 981.89 mbar, 'Avg T': 23.94 C, 'Avg P': 981.85 mbar}
```

▼ esp8266/temp

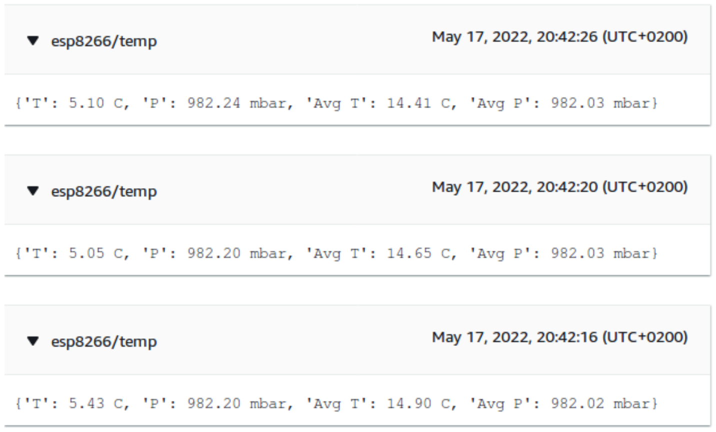
```
{'T': 21.06 C, 'P': 981.86 mbar, 'Avg T': 24.35 C, 'Avg P': 981.85 mbar}
```

▼ esp8266/temp

```
{'T': 21.94 C, 'P': 981.90 mbar, 'Avg T': 24.76 C, 'Avg P': 981.85 mbar}
```

Fig. 1.5: Dropping temperatures inside of the freezer

We decided to take out the board at around 5 degrees (figure 1.6), not to damage the board. It is quite evident that the temperatures would keep dropping the longer the board stayed inside.



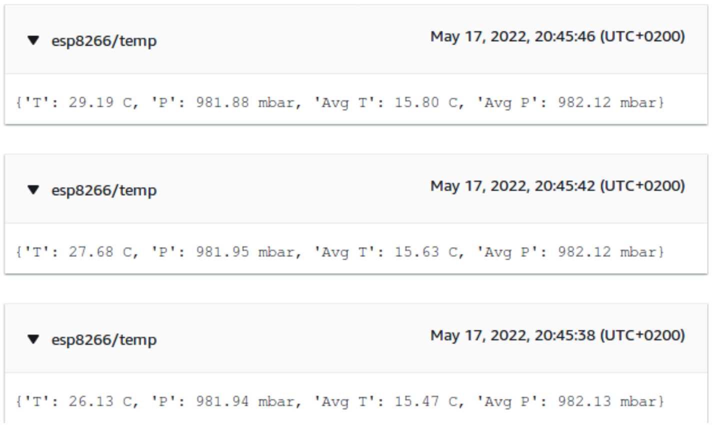
```
▼ esp8266/temp                                May 17, 2022, 20:42:26 (UTC+0200)
{'T': 5.10 C, 'P': 982.24 mbar, 'Avg T': 14.41 C, 'Avg P': 982.03 mbar}

▼ esp8266/temp                                May 17, 2022, 20:42:20 (UTC+0200)
{'T': 5.05 C, 'P': 982.20 mbar, 'Avg T': 14.65 C, 'Avg P': 982.03 mbar}

▼ esp8266/temp                                May 17, 2022, 20:42:16 (UTC+0200)
{'T': 5.43 C, 'P': 982.20 mbar, 'Avg T': 14.90 C, 'Avg P': 982.02 mbar}
```

Fig. 1.6: ”Lowest” temperature inside of the freezer

After that, the board was placed inside of the oven, and the oven was turned on shortly after that. The temperatures quickly started to rise as the oven was heating up (figure 1.7), and we decided to take it out at 40 degrees, as seen in figure 1.8, to be 100% sure we damaged any parts.



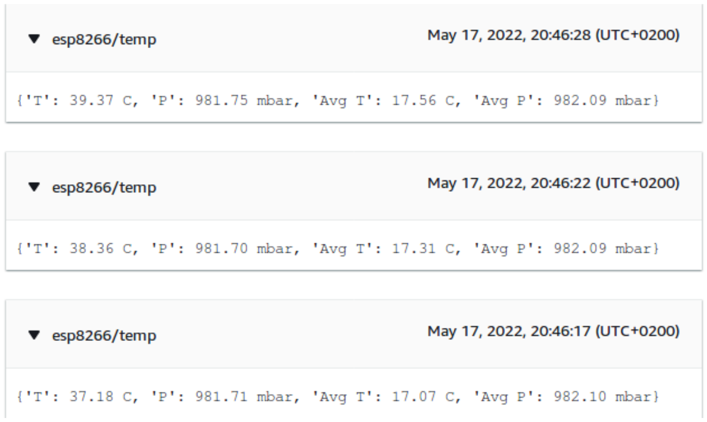
```
▼ esp8266/temp                                May 17, 2022, 20:45:46 (UTC+0200)
{'T': 29.19 C, 'P': 981.88 mbar, 'Avg T': 15.80 C, 'Avg P': 982.12 mbar}

▼ esp8266/temp                                May 17, 2022, 20:45:42 (UTC+0200)
{'T': 27.68 C, 'P': 981.95 mbar, 'Avg T': 15.63 C, 'Avg P': 982.12 mbar}

▼ esp8266/temp                                May 17, 2022, 20:45:38 (UTC+0200)
{'T': 26.13 C, 'P': 981.94 mbar, 'Avg T': 15.47 C, 'Avg P': 982.13 mbar}
```

Fig. 1.7: Rising temperatures

Then, the board was taken out for a few minutes. What followed is the short pressure reading test. The idea was to take the board from the first floor to the ground



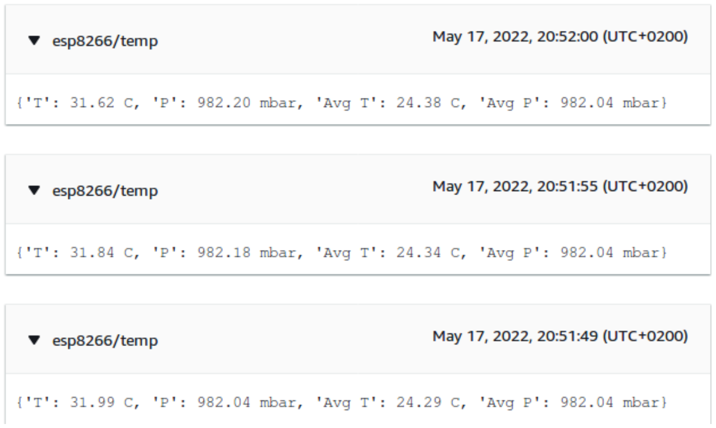
```
▼ esp8266/temp                                  May 17, 2022, 20:46:28 (UTC+0200)
{'T': 39.37 C, 'P': 981.75 mbar, 'Avg T': 17.56 C, 'Avg P': 982.09 mbar}

▼ esp8266/temp                                  May 17, 2022, 20:46:22 (UTC+0200)
{'T': 38.36 C, 'P': 981.70 mbar, 'Avg T': 17.31 C, 'Avg P': 982.09 mbar}

▼ esp8266/temp                                  May 17, 2022, 20:46:17 (UTC+0200)
{'T': 37.18 C, 'P': 981.71 mbar, 'Avg T': 17.07 C, 'Avg P': 982.10 mbar}
```

Fig. 1.8: ”Highest” temperature in the oven

floor, and check if it was able to detect the small difference in altitude. And it sure did - notice that pressure is around 981.7 mbar in the kitchen (figure 1.8). As soon as we started taking it down the stairs, the pressure was starting to slowly increase, as seen in figure 1.9. It reached its maximum as seen in figure 1.10.



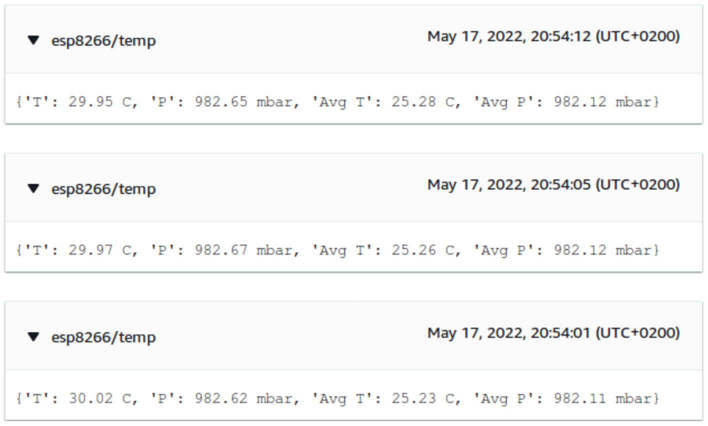
```
▼ esp8266/temp                                  May 17, 2022, 20:52:00 (UTC+0200)
{'T': 31.62 C, 'P': 982.20 mbar, 'Avg T': 24.38 C, 'Avg P': 982.04 mbar}

▼ esp8266/temp                                  May 17, 2022, 20:51:55 (UTC+0200)
{'T': 31.84 C, 'P': 982.18 mbar, 'Avg T': 24.34 C, 'Avg P': 982.04 mbar}

▼ esp8266/temp                                  May 17, 2022, 20:51:49 (UTC+0200)
{'T': 31.99 C, 'P': 982.04 mbar, 'Avg T': 24.29 C, 'Avg P': 982.04 mbar}
```

Fig. 1.9: Rising pressure going down the stairs

Taking it back up to the first floor, in the same room we started in, the readings stabilized around the baseline readings (figures 1.4, 1.11). Besides this, what is also quite interesting is that the board, if plugged out and then back in the same computer after some time, will just carry on sending readings to AWS - notice the time difference between the last and second to last reading in figure 1.11. The only thing



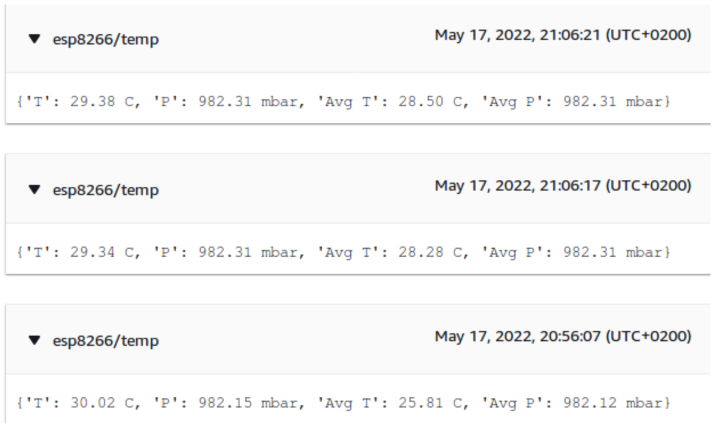
```
▼ esp8266/temp                                   May 17, 2022, 20:54:12 (UTC+0200)
{'T': 29.95 C, 'P': 982.65 mbar, 'Avg T': 25.28 C, 'Avg P': 982.12 mbar}

▼ esp8266/temp                                   May 17, 2022, 20:54:05 (UTC+0200)
{'T': 29.97 C, 'P': 982.67 mbar, 'Avg T': 25.26 C, 'Avg P': 982.12 mbar}

▼ esp8266/temp                                   May 17, 2022, 20:54:01 (UTC+0200)
{'T': 30.02 C, 'P': 982.62 mbar, 'Avg T': 25.23 C, 'Avg P': 982.11 mbar}
```

Fig. 1.10: Highest pressure readings on the ground floor

that was done in this time was that the USB cable was taken out and then put back in.



```
▼ esp8266/temp                                   May 17, 2022, 21:06:21 (UTC+0200)
{'T': 29.38 C, 'P': 982.31 mbar, 'Avg T': 28.50 C, 'Avg P': 982.31 mbar}

▼ esp8266/temp                                   May 17, 2022, 21:06:17 (UTC+0200)
{'T': 29.34 C, 'P': 982.31 mbar, 'Avg T': 28.28 C, 'Avg P': 982.31 mbar}

▼ esp8266/temp                                   May 17, 2022, 20:56:07 (UTC+0200)
{'T': 30.02 C, 'P': 982.15 mbar, 'Avg T': 25.81 C, 'Avg P': 982.12 mbar}
```

Fig. 1.11: The board still sends data to AWS

1.4.2 Test conclusions

We can see that the sensor performs the readings quite well, especially for pressure. It almost instantly detects the change, even though it is less than 1 mbar. While it obviously cannot send instantaneous differences in temperature, it manages to keep up quite well, and it stabilizes well after not too much time. In the real world, it is highly unlikely that one will go from a -10 to a +40 degree environment in a matter of seconds, so these results are quite satisfactory. We would not vouch that the sensor is the *most* accurate, as, even though it is noticeably warmer in the room where the baseline readings were performed in, a difference of almost 9 or 10 degrees outside and inside the room is possible, but we would say highly unlikely, especially around 9PM.

In any case, we can deem this test rather successful for the BMP280 sensor, as it is definitely able to detect even the slightest changes in temperature or pressure.

1.4.3 Difficulties

Unfortunately, we observed two downsides. The first one is that after many different attempts, we did not manage to keep the average temperature and pressure information stored inside a file (local or otherwise), so we could not keep track of that ”over time”, but only during one ”session”.

The other difficulty we came across is, even though ”under the hood”, the function for providing humidity measurements is the same as the one that gives temperature and pressure measurements, but unfortunately humidity was unable to be measured, even locally. We found the answer in the comments of the source code - humidity is only able to be read by the BME280 sensor, and not the BMP280 one that we have.

1.5 Conclusion

We have been able to successfully register our board as an MQTT client and establish a connection to the AWS IoT service. We managed to send temperature and pressure readings to AWS, and read it through the console. Potential upgrades include two things mentioned in part 1.4.3 - storing the data in a file, and providing humidity readings. We would also like to be able to send the data from AWS to a let’s say Python script so that data can be further processed, and perhaps visualized.

References

1. W. source, “FreeRTOS operating system.” Link to FreeRTOS.
2. W. source, “AWS IoT Core.” Link to AWS.
3. W. source, “MQTT - Message Queuing Telemetry Transport.” Link to MQTT.
4. N. Zimic, “Brezžična senzorska omrežja,” 2022.
5. W. source, “Reference GitHub repository.” Link to repo.