

A Mobile Weather Station for Crowdsourc based on OceanConnect Platform

Yiwei Yang, Jiajun Li

May 5, 2018

In recent years, the concept of Internet of things(IOT) has become an extremely hot sector in the Mobile Internet. Particularly, a burgeoning standard of IOT, which named as NB-IOT(Narrow Band IOT), is developing rapidly around the world. In this article, by leveraging the low power consumption and cheap cost of NB-IoT, we build a mobile weather with own power supply and sensors attached to a crowdsourc platform to serve different business scenarios. We build our sever and write an Android application which is based on the Android SDK of Baidu Map open platform to visualize the AQI information to facilitate the life of the citizens.

Key Words: AQI, NB-IOT, OceanConnect, Mobile Internet, Sever, Android.

1 INTRODUCTION

As a burgeoning standard of IOT, most operators around the world chose NB-IOT as the first step in the evolution of the cellular Internet of things, since it has the advantages as low cost, low power consumption, wide coverage and multiple connections. In 2020, the NB-IOT network will be nationwide with more than 1.5 million base stations. Until today, several NB-IOT applications have already been officially launched such as Remote Meter Reading Application and Bike Sharing Application. So, besides just understanding the network architecture of NB-IOT, we also want to realize the integration of the NB-IOT application development.

In this article, we use a NB-IOT communication module to build a mobile weather sation for crowdsourc. It is a mini one which can be paste or play on any physical moving

platform such as shared bike or car to collect and send AQI information. Compared with the traditional large weather stations, it is designed for a crowdsourcing platform to serve different business scenarios. Furthermore, it is also possible to combine the data collected together with the data of the platform to bring much more values.

So, the main contributions of this work can be summarized as follows.

- Implement the hardware development, including PCB design and layout.
- Complete the profile and codec plug-in development of the OceanConnect platform[1].
- Realize the connection between terminal hardware and OceanConnect platform.
- Setup a server to subscribe messages from the platform.
- Make an Android app to receive messages and realize the data visualization.

2 HARDWARE SYSTEM

2.1 LOGICAL ARCHITECTURE

As a weather station, various sensors are necessary such as temperature sensor, humidity sensor, UV sensor and PM2.5 sensor. Besides, since we want it to be a mobile one, which means it can be easily moved and should have a steady supply of power, we need a positioning module and a power supply module as well.

Since our device is super energy-efficient, we only need a 700-mAh lithium-ion battery to provide energy. To further expand its life span, we add a solar panel to charge it, which can supply current up to 200mA. After calculation, our device can work for a month on cloudy days. Once there is the sun, the battery can be fully charged within four hours, which increases its competitiveness.

Taking the power consumption and business scenarios into account, we finally choose GPRS module to get an approximate location by its LBS(Location Bases Service) function and use both lithium-ion battery and solar panel to provide power, letting our mobile weather station can work such a long time without recharging. What needs to be explained is that, in the future, NB module will support OTDOA(Observed Time Difference of Arrival) station based LBS in 3GPP-R14, and the cost will be leveraged furthermore. To demonstrate our project within limited period, we use GPRS module just for LBS instead.

So, since now, we have just finished the design of our power supply module and data collecting module. Then, we need a MCU(Micro-controller Unit) to process the data and transmit them to our communication module, NB module, in order to send the data to the cloud platform, OceanConnect. So, the logical architecture of our hardware system is like Figure 2.1.

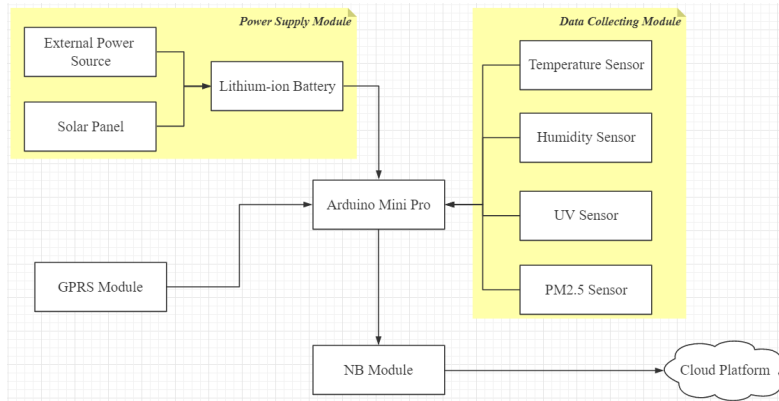


Figure 2.1: Logical architecture of the hardware system of mobile weather station

2.2 HARDWARE DEVICES

As a mobile weather station for crowdsourcing, its cost should be firstly considered. Since the idea of crowdsourcing, its future application scenarios are often widely and heavily deployed. So, the cost must be controlled within a reasonable range and its accurate position are not so important anymore in this scenario. And, that is also part of the reason why we choose GPRS module instead of GPS module and NB-IOT module instead of 4G/5G LTE module.

So, finally the hardware devices we chosen are shown in Table 2.1, as well as some of their critical parameters. The total cost is about 300 yuan, which is mainly because the PM2.5 sensor and NB-IOT module.

Table 2.1: The bill of our hardware devices and their critical parameters

No.	Module	Type	Measuring Range	Cost
1	PM2.5 Sensor	HLPM025K3	$0.1\mu\text{g}/\text{m}^3$ $1000\mu\text{g}/\text{m}^3$	79
2	UV Sensor	GUVA-S12SD	0.1V-3V	39
3	Temperature & Humidity Sensor	DHT11	T: 0°C - 50°C , H:20%-95%	5.14
4	GPRS Module	GA6 mini	/	21
5	Lithium-ion Battery	3.7V, 700mAh	/	13
6	Solar Panel	5V, 200mA	/	16
7	NB-IOT Module	NB101BC95	/	88
8	NB Sim Card	China Telecom	/	20
9	MCU	Arduino Mini Pro	/	9
10	Others	/	/	10
Total:				RMB 300.14

2.3 PCB DESIGN

We also need to design a PCB(Printed Circuit Board) to layout our hardware devices. Since we have already clarified the overall logical architecture , we can simply design the PCB like Figure 2.2 based on it.

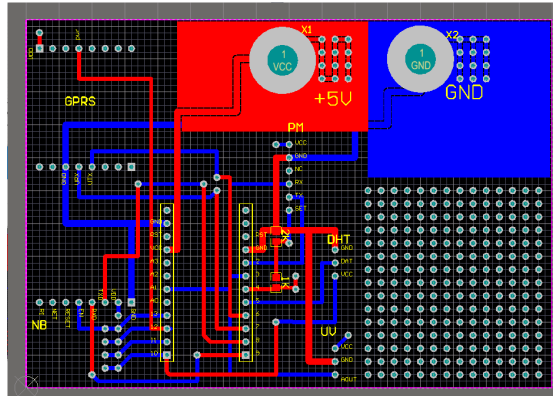


Figure 2.2: The PCB design

2.4 ASSEMBLY&PACKAGE

Finally, we can assemble all the hardware devices into our mobile weather station, and package it like Figure 2.3.

The solar panel and UV sensor are on the surface of the package so that it can work efficiently. We left the PM2.5 sensor an air vent for the paper box.

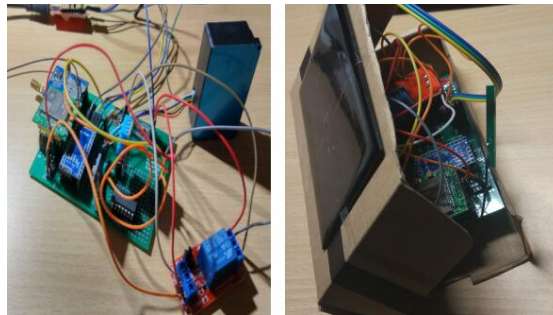


Figure 2.3: The demo of mobile weather station.

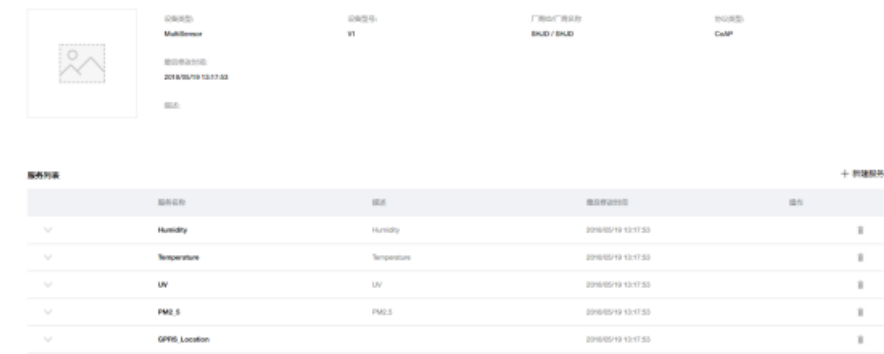
3 OCEANCONNECT PLATFORM

3.1 DEVICE PROFILE DEVELOPMENT

When we use the OceanConnect platform to integrate our own device, we need to prepare the capability description file for this device, which is the “Profile” of the device. The profile of the device is a document describing what the device is, what it can do, and how to control it. The profile will be uploaded to the OceanConnect platform, in order to standardize the management of the devices.

So, the profile should follow some certain design guidelines. According to the guidelines about the packaging structure of profile, which is in the material[2], and the guidelines about how to write the “devicetype-capability.json” and “servicetype-capability.json”, we put the profile in Appendix A.

So, since we have written the device profile and uploaded it to the OceanConnect, we can get such results on the platform as shown in Figure 3.1.



服务列表	服务名称	描述	服务创建时间	操作
▼	Humidity	Humidity	2019/05/19 13:17:53	⋮
▼	Temperature	Temperature	2019/05/19 13:17:53	⋮
▼	UV	UV	2019/05/19 13:17:53	⋮
▼	PM2.5	PM2.5	2019/05/19 13:17:53	⋮
▼	GPS Location		2019/05/19 13:17:53	⋮

Figure 3.1: The device profile shown on OceanConnect

3.2 CODEC PLUG-IN DEVELOPMENT

As shown in Figure 3.1, the communication protocol type between NB-IOT module and OceanConnect platform is CoAP. The payload of CoAP message is the data of the application layer and since the NB-IOT module has a high demand for power saving, the application layer data use hexadecimal format instead of json format. So, the platform invokes the codec plug-in to realize the function of converting hexadecimal message into json format in order to provides the message to the app server. The overall plan is like figure 3.2.

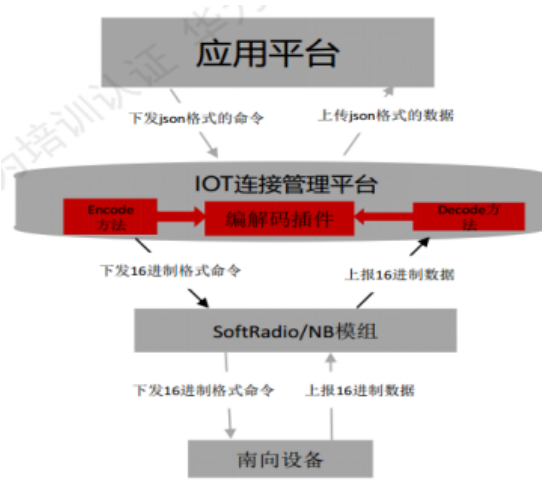


Figure 3.2: The overall plan of End-to-End development[2]

So, we need to develop the codec plug-in on the OceanConnect platform like Figure 3.3, with the help of the guidelines. Notice, the parameter names defined here should be as same as the one defined in the profile.

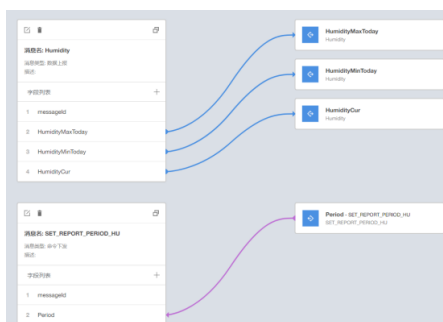


Figure 3.3: Part of the codec widget

3.3 CONNECTION

After we finish the development of the hardware side and the platform side respectively, the next step we should do is let the device online. According to the terminal docking process shown in material[2], we need to send a series of AT instructions by a serial port tool to control the NB-IOT module.

If the device has successfully been online, we now can manage the device by the Ocean-Connect platform. As shown in Figure 3.4 and Figure 3.5, we can use this platform to issue instructions and check the reported data.

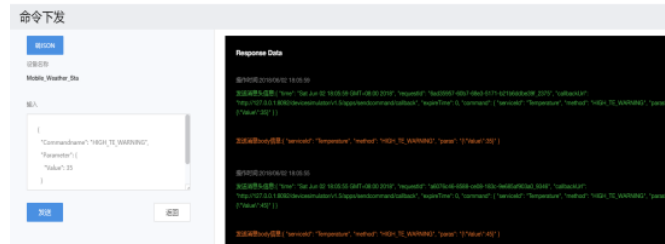


Figure 3.4: An example of issuing instructions



Figure 3.5: An example of the reported data

3.4 SERVER DEPLOYMENT

We first use `jdk`[3] to generate certificates, then use `openssl`[4] to generate pem files, which will be uploaded to OceanConnect to send legal messages, shown in Figure 3.6.



Figure 3.6: upload pem files to OceanConnect

Then we use `tomcat`[5] to setup and start the server, the main configurations is shown below.

```
<Connector port="443" protocol="HTTP/1.1"
            maxThreads="150" SSLEnabled="true"
            scheme="https" secure="true"
            clientAuth="true" sslProtocol="TLS"
            keystoreFile="C:/Program Files/tomcat/apache-tomcat-8.5.31/
            conf/tomcat.keystore" keystorePass="123456"
            truststoreFile="C:/Program Files/tomcat/
            apache-tomcat-8.5.31/conf/
            tomcat.keystore" truststorePass="123456">
</Connector>
```

After importing the configurations to the browser, we can access to the sever to check whether it works or not. The welcome window is shown in Figure 3.7

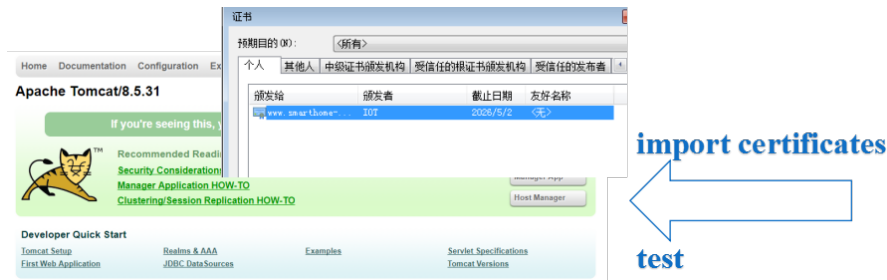


Figure 3.7: The welcome page of tomcat

To subscribe HTTPS messages from OceanConnect, we build a Restful interfaced Webapp to receive the informations from OceanConnect. We also put the sever on the internet. The main code is shown below.

```

<display-name>RESTfulWS</display-name>
<servlet>
  <servlet-name>REST Service</servlet-name>
  <servlet-class>com.sun.jersey.spi.container.servlet
    .ServletContainer</servlet-class>
  <init-param>
    <param-name>com.sun.jersey.config.property.packages
      </param-name>
    <param-value>com.huawei.com</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>REST Service</servlet-name>
  <url-pattern>/rest/*</url-pattern>
</servlet-mapping>

```

We first use postman[6] to sent test messages to the sever and it works well, so we can write a java application to subscribe notifications from OceanConnect. The core configura-tion code is shown in Appendix B.

After that, if OceanConnect receives messages from our IoT device, our server can receive the messages from OceanConnect, which is shown in Figure 3.8.


```
.6234, "Base_Code":58065}, "eventTime": "20180605T182719Z"}
<"notifyType": "deviceDataChanged", "requestId": null, "deviceId": "519ec47b-c108-4dca-86d8-428e08ab2c1d", "gatewayId": "519ec47b-c108-4dca-86d8-428e08ab2c1d", "services": [{"serviceId": "GPRS_Location", "serviceType": "GPRS_Location", "data": {"Cell_Code": 6234, "Base_Code": 58065}, "eventTime": "20180605T182719Z"}]
```

Figure 3.8: The messages received from OceanConnect

3.5 ANDROID APPLICATION

Since our server has received the messages from OceanConnect, we now can build up our android application to get the messages from the sever and realize the data visualization.

Thanks to the Android SDK of Baidu Map open platform, it seems not too difficult for us to write an Android application to realize the data visualization. We think the labs that we do on the Mobile Internet course helps a lot.

As shown in Figure 3.9, you can see the AQI information from several different places by touching the red points in this app, which means the positions the base station that our mobile weather station connected. Although the number of the stations is not too much in this demo, it still reflects the idea of crowdsourcing which we want to implement in this project. People can use their cellphones to watch the meteorological information across every conner of Shanghai, which is collected by the super lowcost mobile weather stations.

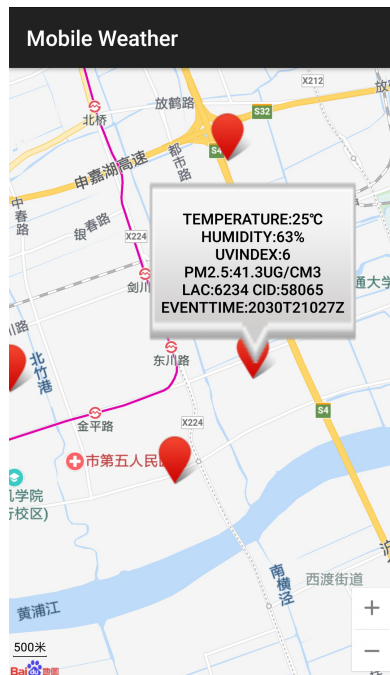


Figure 3.9: A demo of the Android app

4 CONCLUSION

NB-IOT focuses on the low power consumption and wide coverage part of the IOT, it has extensive application scenarios as well as a huge market scale. As the advantages of NB-IOT, it is quite suitable for applying in Meteorological measurement field. To strengthen our ability to practice the Internet of things, we build a mini mobile weather station which can be "pasteplay" on any physical moving platform (e.g. shared bike or car) to collect and send AQI information.

By leveraging the low power consumption and cheap cost of NB-IoT, we have an IoT device with own power supply and sensors attached to a crowdsource platform to serve different business scenarios.

Furthermore, it is also possible to combine the data collected together with the data of the platform to bring much more values.

Through this project, we have a full stack IoT experience, and strengthen our capability of mobile network programming. What's more, we have a deep understanding of the advanced communication technology, accumulate engineering experience and fostered the spirit of research. The future of IoT has open a door for us, and we are lucky to take our ideas into reality.

Finally we want to express our thanks to Professor Wang, Teacher Tian, Teacher Fu and dear teaching assistants for teaching us various knowledges of mobile internet.

Appendices

Appendix A

The json files on OceanConnect.

```
    "services": [
      {
        "serviceType": "Temperature",
        "description": "Temperature",
        "commands": [
          {
            "commandName": "SET_REPORT_PERIOD_TE",
            "params": [
              {
                "paraName": "Period",
                "dataType": "int",
                "required": true,
                "min": "0",
                "max": "300",
                "step": "1.0",
                "maxLength": "0",
                "unit": "min",
                "enumList": null
              }
            ]
          },
          {
            "commandName": "HIGH_TE_WARNING",
            "params": [
              {
                "paraName": "Value",
                "dataType": "int",
                "required": true,
                "min": "0",
                "max": "50",
                "step": "1.0",
                "maxLength": "0",
                "unit": "C",
                "enumList": null
              }
            ]
          }
        ]
      }
    ],
    "properties": [
      {
        "propertyName": "TemperatureCur",
        "dataType": "int",
        "required": true,
        "min": "0",
        "max": "50",
        "step": "1.0",
        "maxLength": "0",
        "method": "R",
        "unit": "C",
        "enumList": null
      },
      {
        "propertyName": "TemperatureMinToday",
        "dataType": "int",
        "required": true,
        "min": "0",
        "max": "50",
        "step": "1.0",
        "maxLength": "0",
        "method": "R",
        "unit": "C",
        "enumList": null
      },
      {
        "propertyName": "TemperatureMaxToday",
        "dataType": "int",
        "required": true,
        "min": "0",
        "max": "50",
        "step": "1.0",
        "maxLength": "0",
        "method": "R",
        "unit": "C",
        "enumList": null
      }
    ],
    "events": []
  }
}
```

Figure .1: The devicetype-capability.json

```
{
  "devices": [
    {
      "manufacturerName": "SHJD",
      "manufacturerId": "SHJD",
      "model": "V1",
      "protocolType": "CoAP",
      "deviceType": "MultiSensor",
      "serviceTypeCapabilities": [
        {
          "serviceId": "Humidity",
          "serviceType": "Humidity",
          "option": ""
        },
        {
          "serviceId": "Temperature",
          "serviceType": "Temperature",
          "option": ""
        },
        {
          "serviceId": "UV",
          "serviceType": "UV",
          "option": ""
        },
        {
          "serviceId": "PM2_5",
          "serviceType": "PM2_5",
          "option": ""
        },
        {
          "serviceId": "GPRS_Location",
          "serviceType": "GPRS_Location",
          "option": ""
        }
      ]
    }
  ]
}
```

Figure .2: The servicetype-capability.json of temperature

Appendix B

The core configuration code of the java application to subscribe notifications from Ocean-Connect.

```

public static final String CALLBACK_BASE_URL =
    "https://117.136.8.236:443";
public static final String DEVICE_DATA_CHANGED_CALLBACK_URL
= CALLBACK_BASE_URL + "/RESTfulWS/rest/UserInfoService/subscriber1";
/*
 * subscribe deviceDataChanged notification
 */
String callbackurl_deviceDataChanged =
Constant.DEVICE_DATA_CHANGED_CALLBACK_URL;
String notifyType_deviceDataChanged =
Constant.DEVICE_DATA_CHANGED;

Map<String, Object> paramSubscribe_deviceDataChanged =
new HashMap<>();
paramSubscribe_deviceDataChanged.put("notifyType",
notifyType_deviceDataChanged);
paramSubscribe_deviceDataChanged.put("callbackurl",
callbackurl_deviceDataChanged);

String jsonRequest_deviceDataChanged =
JsonUtil.jsonObj2Sting(paramSubscribe_deviceDataChanged);

Map<String, String> header_deviceDataChanged = new HashMap<>();
header_deviceDataChanged.put(Constant.HEADER_APP_KEY, appId);
header_deviceDataChanged.put(Constant.HEADER_APP_AUTH, "Bearer"
+ " " + accessToken);

HttpResponse httpResponse_deviceDataChanged =
httpsUtil.doPostJson(urlSubscribe,
header_deviceDataChanged,
jsonRequest_deviceDataChanged);

String bodySubscribe_deviceDataChanged =
httpsUtil.getHttpResponseBody(
httpResponse_deviceDataChanged);

System.out.println("SubscribeNotification: " +
notifyType_deviceDataChanged + ", response content:");
System.out.print(httpResponse_deviceDataChanged
.getStatusLine());

System.out.println(bodySubscribe_deviceDataChanged);
System.out.println();

```

REFERENCES

- [1] Huawei oceanconnect. <http://lbsyun.baidu.com/>.
- [2] Huawei hcda-iot training materials. <http://support.huawei.com/learning/trainFaceDetailAction?lang=zh&pbPath=term1000154640&courseId=Node1000012229>.
- [3] Java se. <http://support.huawei.com/learning/trainFaceDetailAction?lang=zh&pbPath=term1000154640&courseId=Node1000012229>.
- [4] Openssl - cryptography and ssl/tls toolkit. <https://www.openssl.org/>.
- [5] Apache tomcat. <https://tomcat.apache.org/>.
- [6] Postman api development. <https://www.getpostman.com/>.