In video #112 I introduced the LoRa technology and in episode #115 We built a gateway for the LoRaWAN network. Today, we will create the missing link: We will build a sensor node which can connect to the things network and will send some messages. In a future episode, of course, I will also check, if we get a few miles of range…

So, let’s start.

From the past videos, we know, that we have to find a module which is capable to communicate in the 868 MHz band (at least for Europe). If we search our usual source, we get a big choice of such modules:

SI4463

SI4432

CC1101

ASK/FSK Modules

RFM69HW

RFM95W

NRF905

SX1276

RFM12B

Each of these boards are based on another chip. Each of these chips needs another library. Fortunately, we get libraries for all of them.

Most of the modules can be purchased for the 315/433/868/915MHz bands. This does not mean, that one module can be used for all bands. The passive components on the boards like capacitors and inductors are optimized only for one band. This means, that you have to decide the band when you order the module.

The prices for the modules also vary. For example, you get a RFM69HW module for less than 3$. The RFM95W from the same supplier costs at least 7$. So, can we just use the cheapest module or is there another difference between these modules?

We already saw that the operating band and the needed libraries can be different. What else? If two humans want to talk to each other, we have to be close enough to hear each other. And because our ears are tuned to the same “frequency band” as the mouth of our counterpart, we hear each other. But this is not sufficient. We also have also to use the same language, or one of the two has to understand more than one language. Otherwise, we hear what the other person says, word by word, but we do not understand. A similar thing happens in radio systems. Even if the sender and the receiver are tuned to the same frequency, they also have to use the same modulation principle. Modulation is used to “load” your data to the channel. As an example I use FSK modulation, which means frequency shift keying. With this modulation principle, we chose two frequencies, let’s say 868 MHz minus 100 Hz and 868MHz plus 100 Hz. If we want now to transport a logical zero, the transmitter sends the 868 minus 100 frequency. If we want to transport a logical one, it uses the 868 plus 100 Hz frequency.

The receiver on the other hand, knows, if it “hears” a minus frequency that it is a logical zero and vice versa. So, it can rebuild the logical signal.

Why is this necessary? Without that, we just could mount an antenna to the sending pin and one on the receiving pin of our Arduinos. Just try it and you will discover, that this does not work… At least not over a long distance.

If we now check the data sheets of our modules, we see, that they speak and understand three completely different “languages”

1. FSK or GFSK or OOK
2. MSK or GMSK
3. LoRa

Each of these languages have many dialects, and if you want to have a successful communication, also the dialects have to be the same. This is, why often people use the same chips on both sides of their networks. Then, they are sure, that the sender and the receiver understand each-other.

So, for our purpose, we only can choose the RFM95W or the SX1276 modules, because only these understand, in addition to other modulation principles, also LoRa. So, they are quite talented modules… Unfortunately, they are also the most expensive ones. And because LoRa technology is patented, the RFM95W module behaves very similar to the SX1276. Hope got a license to build this chip from Semtech. As an advantage, we can use the same libraries for both modules.

So, the first two decisions are done. I use a RFM95W module on 868 MHz and the modulation will be LoRa, because the Gateway only accepts this “language”. Just to avoid some confusion: The RFM95W modules use RF96 chips. This got me confused at first.

This time we do not use an ESP8266 as a microcontroller because we will communicate through LoRa and do not need Wi-Fi. And because we want to use our nodes for measuring data, we profit from the many pins, especially also analog pins, and the vast number of compatible sensors of the Arduino.

And there is a nice Dragino shield for the Arduino available. I use it instead of soldering wires to the RFM95W. In addition, the module has the needed level shifters from 5 to 3.3 volt already built-in. So, the price is more than ok compared to the “barebone” module.

Now we have everything necessary for our LoRa node. I just plug the Dragino shield onto the Arduino Uno and make sure, that the jumpers are set like in this picture. Otherwise, it might not work.

Please do not forget to connect the antenna before you power the Arduino.

Next, we need a library for our hardware which is compatible with the LoRaWAN network. It is written by IBM in Switzerland. I do not exactly know, why they wrote such a library, it might have a relation with the fact, that IBM is also the inventor of the MQTT protocol.

This library is called “LMIC” and had to be ported to the Arduino IDE and therefore looks a little different than what we are used to.

You see this for example in the way the pins are defined. They are defined in a structure, not as we are used, as #defines. Nevertheless, these definitions are important. I had to find this out the hard way and it costed me quite a long time to find the right definition of these pins for the Dragino board. So, here they are:

If you want to wire a bare module to your Arduino, take care that your wiring is reflected in these definitions. Otherwise you just scratch your head, and nothing goes.

So, we use the “hello world” example for our first experiment. You find the link in the description. In addition to the pin definition, another area of the sketch is important: The different “keys” used by LoRaWAN.

You find them here and they are called: Network key, application key, and device address. But where do we find these numbers? We have to remember, that we want to connect our node via a gateway to the TTN network. So, we have to go to their homepage, and there, to the console. This is the same page we used to define our gateway. But now, we have to go to the applications part. Here, we have to create a new application and give it a name. I call it “YouTube Explanation”. Make sure, you comply with the naming conventions and enter a description. Then, you chose the handler.

Now, you see an Application EUIS and an access key. Both numbers are currently not important. We see also, that this application has no devices defined. So, that is the next step to do. We create such a device. We enter our name and a “fake” EUI. Then, we save the device. The activation method is OTA, but our device can only be activated with the ABP method. So, we go into the settings tab and change that. Now, we get the needed keys for our sketch. Fortunately, you can just click on this button, and you get the string exactly as needed in the sketch. Make sure, that “msb” is shown. Just cut- and paste the Network session key into the NWskey , the App session key into appskey, and the device address into devaddr. Make sure, you add a 0x in front of the devaddr.

Now we are ready to go and we can compile and upload the sketch. Make sure, your node can receive signals from the nearest gateway. In my case, this is not very hard, because it is only a few centimeters away…

Now, the magic should start and the node should log into the TTN network and start to send messages. During testing, we have to pay attention to the “fair access policy”. It limits the time we can use a gateway, even if it is our own one.

TTN’s policy is as follows:

* An average of 30 seconds’ uplink time on air, per day, per device.
* At most 10 downlink messages per day, including the ACKs for confirmed uplinks.

Which means:

For 10 bytes of payload, this translates in (approx.):

* 20 messages per day at SF12
* 500 messages per day at SF7

You find the link to this policy in the description.

But what are SF12 and SF7? SF means “Spreading factor” and explains, how data is transferred over the air. The lower the number, the faster the transmission. But of course, if your channel is somehow disturbed exactly during this short transmission, nothing arrives at the gateway. If you use higher SF numbers, the transmission takes longer and the chance for a successful arrival is therefore higher, but also the air time is longer. So, always use the smallest SF factor possible for your communication if you need many messages per day. And th highest, if you want to be more sure it arrives.

And no, I did not test what happens, if you exceed this limit.

Are there other SF numbers outside the range from 7 to 12 available? At least, the node software only accepts those SFs.

After this theoretical excursion, we return to the console. And really, messages arrived. And the message, again is two bytes long and the two asci letters are “hi”.

If we open the message, we see, which gateway was the receiver of the message. I assume, that it is also possible, that a message is received by two or more gateways. Then, we should see this here. I will try this later in the week from a hill near my home. There, I should be able to connect to more than one gateway.

Another fact is also interesting: The channel used for the communication changes with every message. At least, we see, that each message used a different channel, one after the other. This is particularly important, if one channel is blocked for a longer time by another signal. Then, we still have the chance, that at least all other messages can be transferred.

We see also the SF which was used by my node. And I can easily change it here in the sketch.

So, this is the end of the works in the lab. Now, I will place the gateway outside my house and try the node in the “wilderness”.

<https://arxiv.org/pdf/1607.08011.pdf>

https://www.thethingsnetwork.org/forum/t/limitations-data-rate-packet-size-30-seconds-uplink-and-10-messages-downlink-per-day-fair-access-policy/1300