



Building a TTN node

Identifying the components

Our bill of materials (set available at [TinyTronics](https://www.tinytronics.net/))

- RFM Module RFM95W (868 Mhz)
- Pro Mini 3.3V 8Mhz (5V will NOT work!!)
- FT232RL-3.3v-5v-TTL-USB-Serial-Port-Adapter (has to have a 3.3V option!)
- USB cable for FTDI
- 1x Led 3mm red
- 1x resistor 1k (brown-black-red)
- 2x resistor 4k7 (yellow-violet-red)
- 1x resistor 10k (brown-black-orange)
- 1x resistor 100k (brown-black-yellow)
- Header 6p female (Arduino programming)
- Header 4p female (sensor)
- Header 4p male 90 degrees (gps) (cut 2p from the header included with the Pro Mini)
- 2x header 8p 2mm (RFM95W)
- 2x header 12p 2.54mm male (Arduino)
- 1x header 2p 2.54mm male (Arduino)
- PCB 'Loratracker RFM98
- Sensor TH06 (si7102) (Temperature and Humidity)
- Sensor BH1750 (light sensor)
- Ball sensor

This work is licensed under the Creative Commons Attribution-NonCommercial 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/4.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

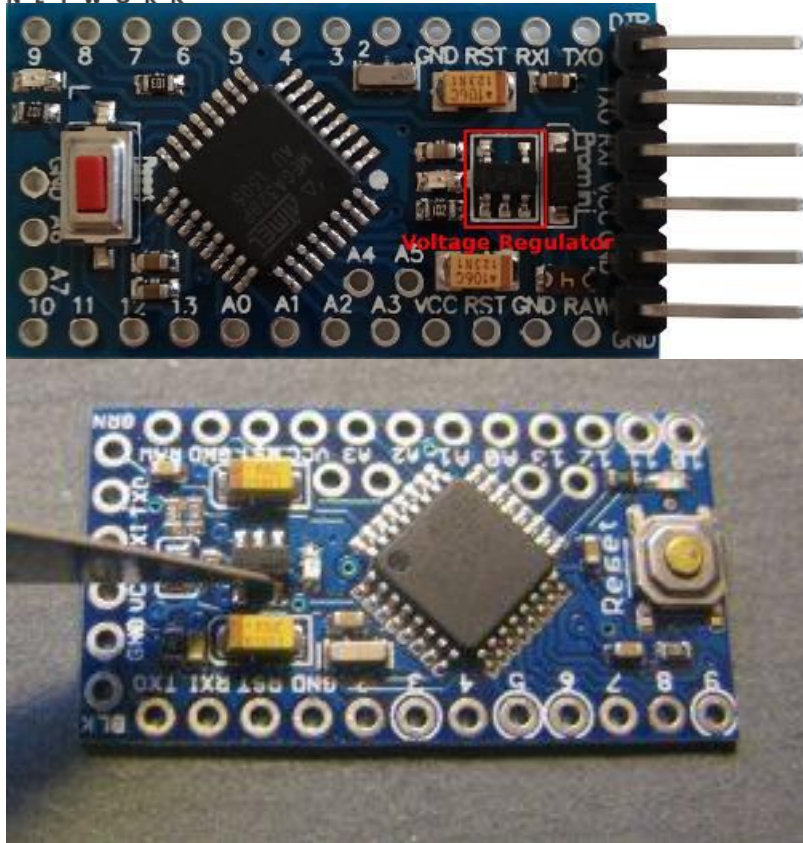
We use a very compact 'sandwich' construction. So you have to follow the steps in the manual in the right order! You cannot reach some soldering points when you mess up the order!

Low Power the Arduino?

There is an option to 'low power' the Arduino. BUT this requires some precise soldering!! The setup will work without removing the components. So if you are a novice solderer, skip the first part! It requires a sharp knife to remove the power converter, and to cut the connections for the two on board leds. If you want to battery power your node for a longer time (weeks or more) this is the option for you. Current in sleep mode will be reduced to <100uA.

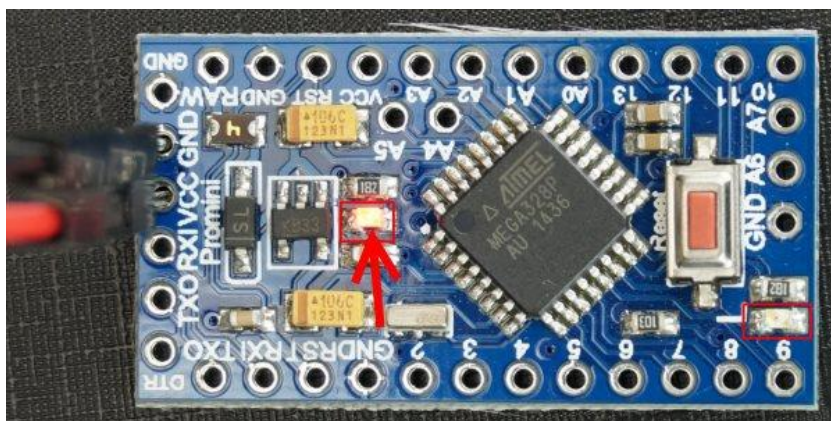
(OPTIONAL) Remove the Regulator

The easiest way to remove the regulator and not damage the board is to use a very sharp scalpel to cut through the regulator leads at the point they join the regulator body.



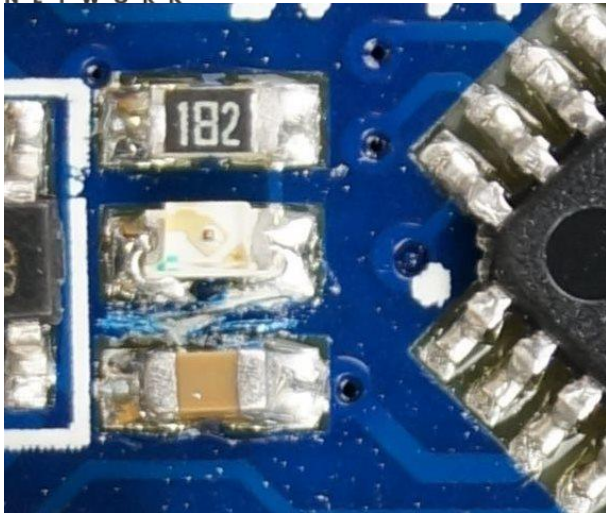
(Optional) Remove the LEDs

There are two LEDs on the Pro Mini board. The two LEDs are marked with a red square in the following picture. The power LED is marked with an arrow. If you are not sure where the power LED is on your board, then you can just power it and you will see the LED.



When you found the power LED, then try to locate at least one trace that leads to the LED. In the second picture below, I marked the traces on my board. A high-resolution picture with a lot of light helps to find the traces.

When you found a trace to the power LED, then you take a knife and break the trace, so that it will not conduct any more. You can see my result in the third picture below.



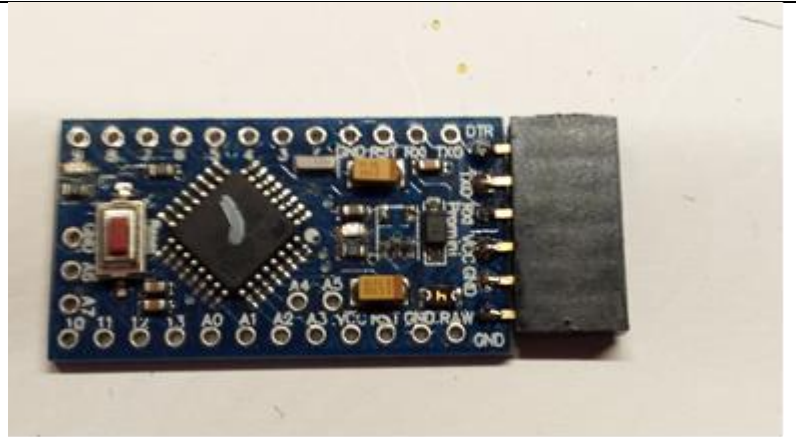
Removing the LEDs can be tricky, so it's easier to remove the series resistors for the LEDs instead. This version of Pro Mini also has a resistor feedback network for the regulator across VCC, these consume power so should be removed. Just push the resistors aside with a soldering iron. The picture shows the Pro Mini with unwanted parts removed, locations of the removed components are circled in red.

More on this subject you may find at: <https://andreasrohner.at/posts/Electronics/How-to-modify-an-Arduino-Pro-Mini-clone-for-low-power-consumption/>

Building the Node

Follow the steps in the right order, or you cannot reach some soldering point afterwards!

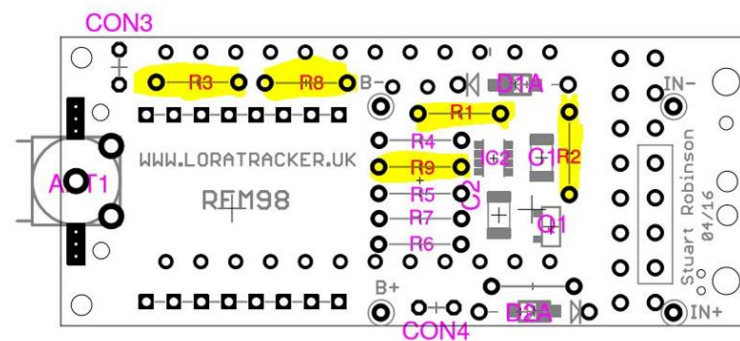
Connect the 6p female header to the Arduino. To get it low profile bow it so it is straight with the PCB.



Mount the resistors on the PCB.

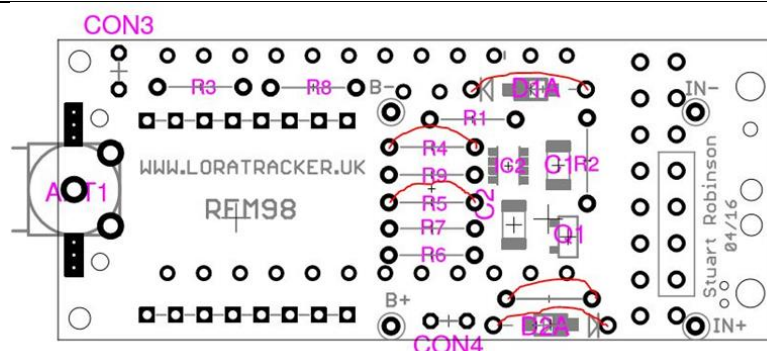
There is no identification in the Silk screen, but the resistors have to be mounted on the silk screen side (with the RFM98 text on top).

- R3: 1k resistor (brown-black-red)
- R8: 4k7 (violet, yellow, red).
- R1: 100k (brown, black, yellow).
- R2: 10k (brown, black, orange).
- R9: 4k7 (violet, yellow, red).

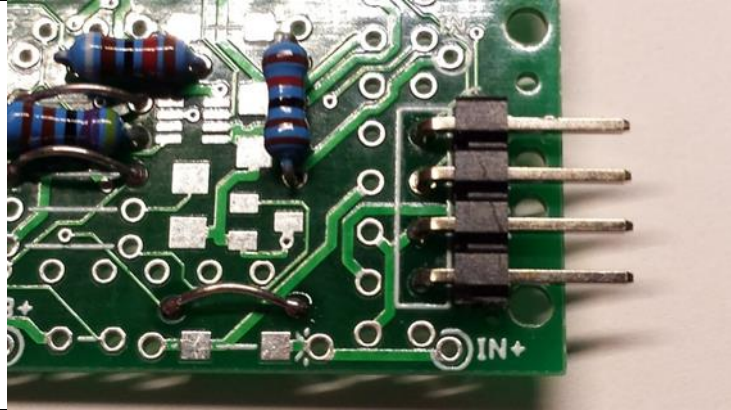


Cut the wires on the bottom, and create some bridges. These Bridges have to be true bridges, so not too close to the PCB.

- D1A will be a bridge
- R4 will be a bridge
- R5 will be a bridge
- C3 will be a bridge
- D2A will be a bridge

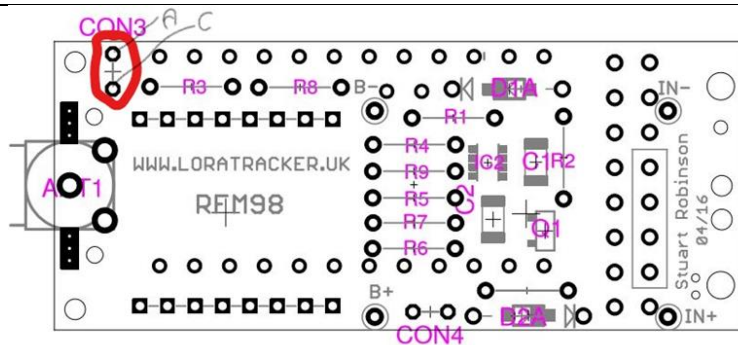


Mount the 4p male header 90 degrees at the marked four pins (one above the side). The pins should point to the side of the board. You can use the left over program connector from your Pro Mini board (cut off 2 pins to get 4). On the picture bridge D2A is not yet mounted!



The battery wires connect to IN+ (RED) and IN- (BLACK)

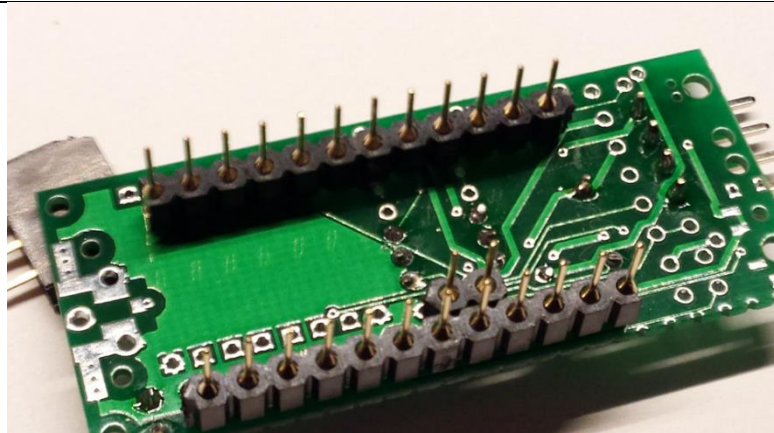
Solder the LED on the CON3 position. The longest wire (Anode) should be on the outside of the board.



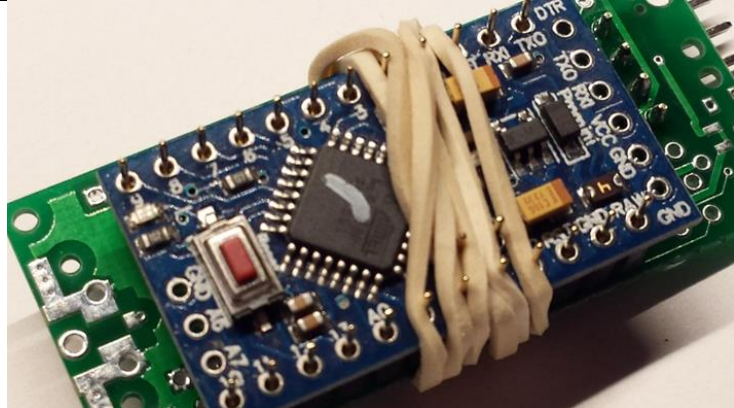
Mount the two headers of the Arduino Board, they are mounted on the other side of the board.

Put the small 2 pin header as shown on the picture.

Take note of the right place of the A4 and A5 pins! These pins are referenced on the silk screen of the Arduino board.

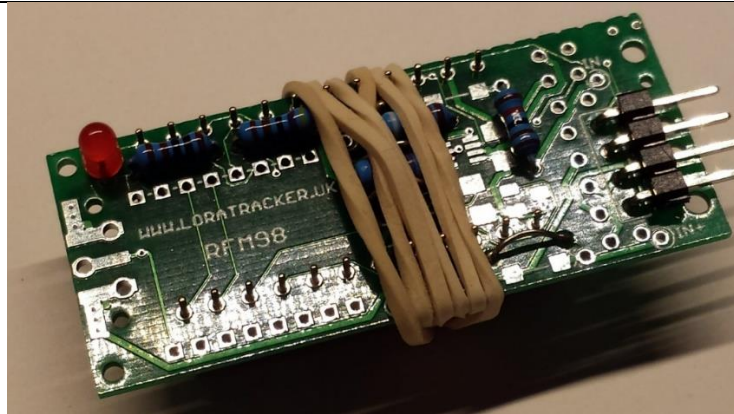


You can use the Arduino board and an elastic band to hold the pins while soldering the pins to the board.



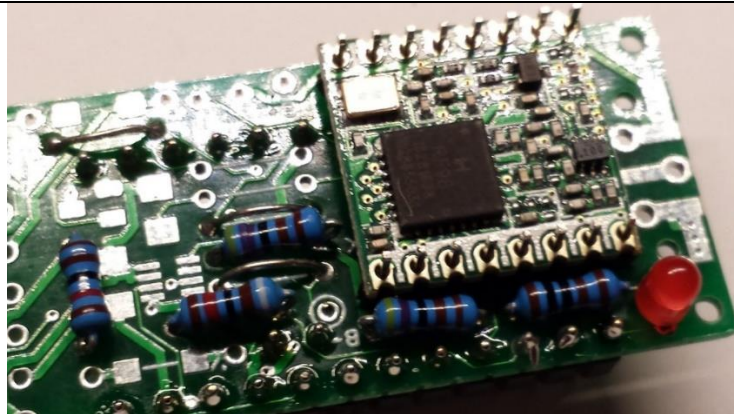
DO NOT SOLDER THE ARDUINO AT THIS MOMENT!!!

So only solder the pins on the LED side!

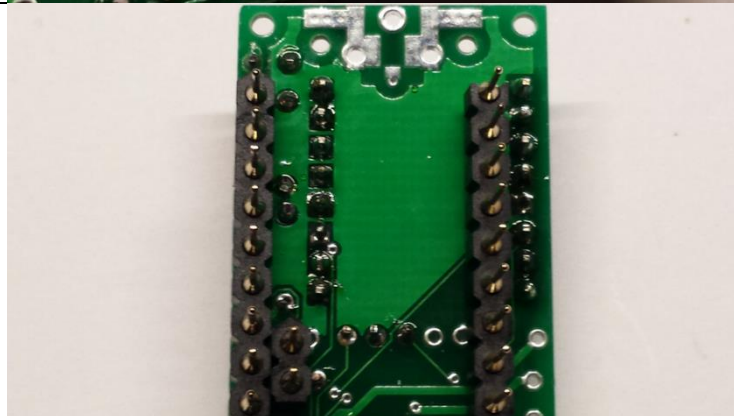


After soldering the 12+12+2 pins remove the elastic band and the Arduino.

Now follow the same procedure for the RFM95 board. Put the 8pin 2mm headers on the 'resistor' side of the board with the RFM95 on top.



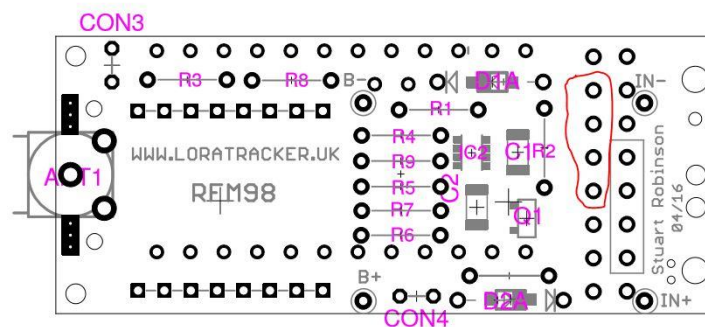
Now solder the other side, besides the two Arduino connectors.



Cut a wire for the antenna. It should be 82mm long, measured from the PCB. So add some 5 mm to solder it. We use the mid hole of the antenna connector. You can as well use a SMA or uC connector on these pads for later experiments.



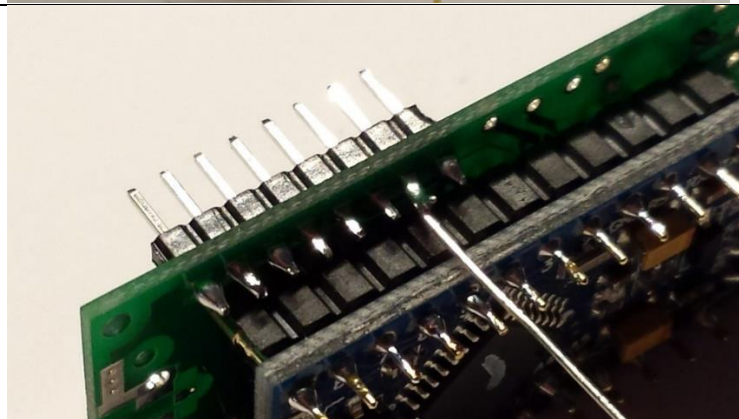
Put the 4 pin female header on position 2, 3, 4 and 5 of the left row as shown in the red circle.



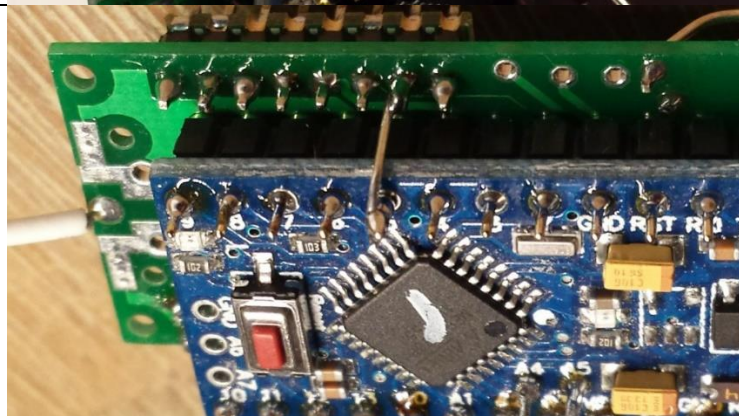
Put the Arduino in place. The reset button goes on the Antenna side. Solder the Arduino, do not forget A4 and A5.



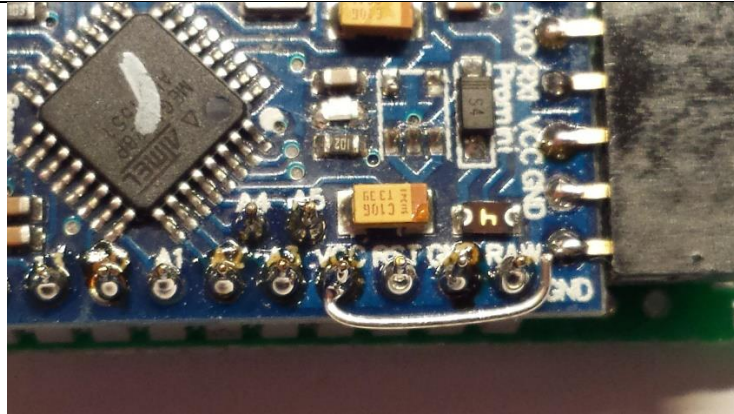
We have to modify two points here. The RFM95 is used in 'LoRa' mode and therefore pin Dio1 should be connected. The easiest way is to connect the unused D5 pin from the Arduino to Dio1.



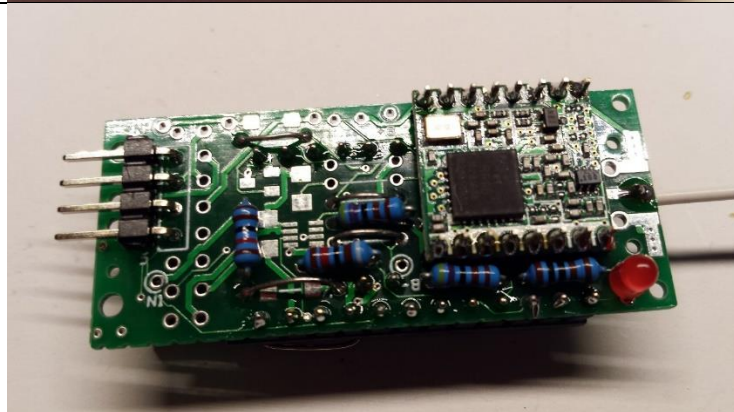
DIO1 is the second pin in line of pin 5 of the Arduino. Solder a short piece of wire on it. Cut the wire in the right length, bend it to pin 5 and connect it (without connecting the other pins).



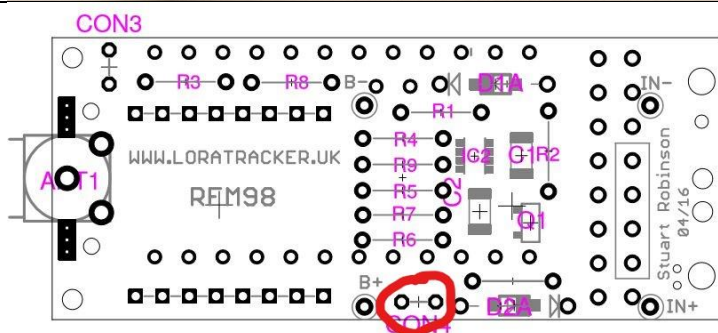
Now connect with a bridge pen 'RAW' and 'VCC' on the Arduino.



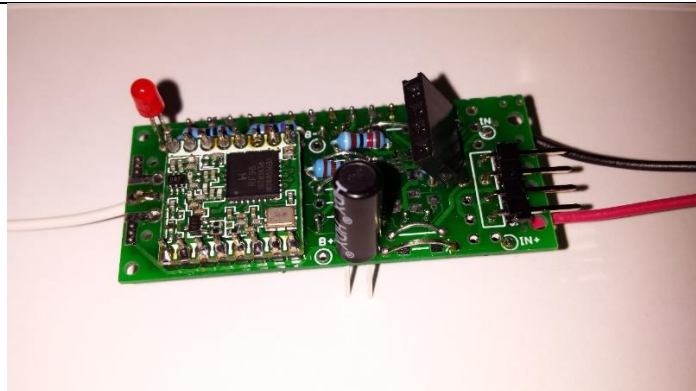
Solder the RFM95 Board. Look at the picture for the orientation. The big black chip should be on the inner side of the PCB. Remark: Bridge D2A and header NOT mounted on this picture!



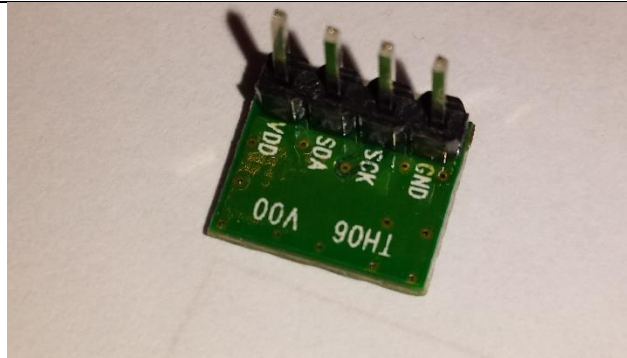
Mount the ball switch 'looks like a Condensator' into the CON4 position. The ball switch has no polarity, so you can place it in any direction.



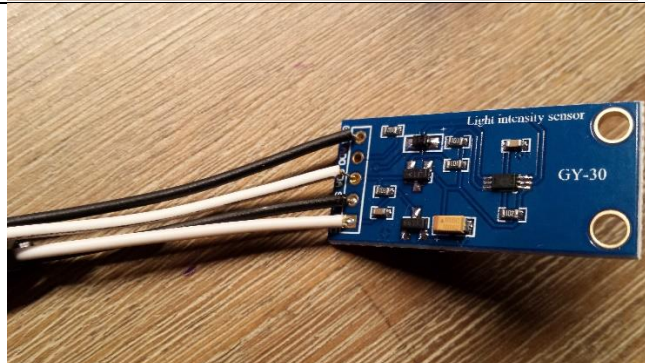
REMARK: if you do not want to use this you can connect any digital signal or switch on CON4!



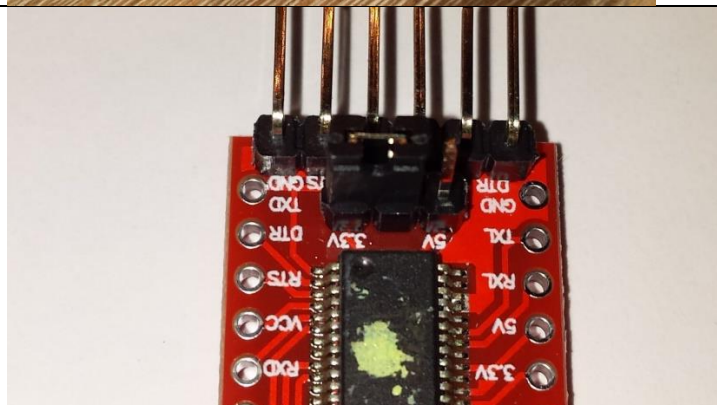
Solder a header on the si7021 sensor (marked as TH06). You can connect the TH06 sensor direct on the board. First solder a 4p header on the sensor.



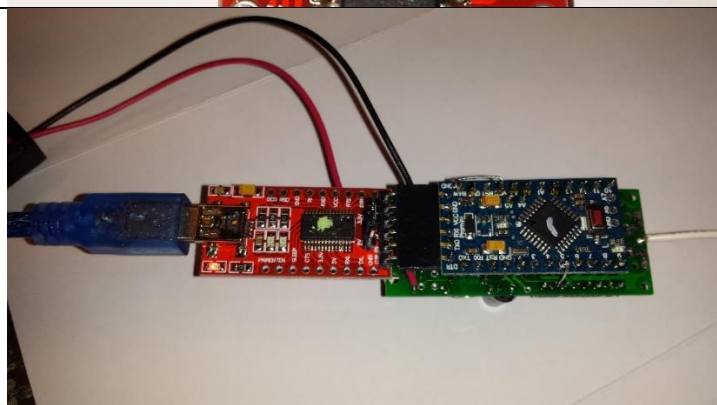
Cut two prototype wires so we have four. Solder these wires to the GY-30 sensor (we will not use the included header). Connect them to GND, SDA, SCL and VCC (leaving ADO unconnected)



Put the Jumper on the FTDI board on 3.3V position!



Connect the USB cable to the FTDI board, the FTDI board to the Arduino as shown.



Starting up the Arduino Environment

<https://www.arduino.cc/en/Guide/HomePage>

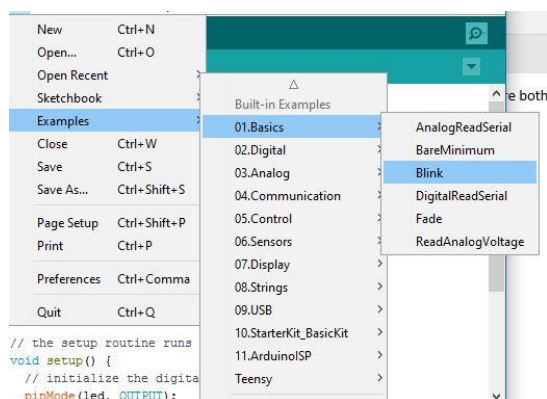
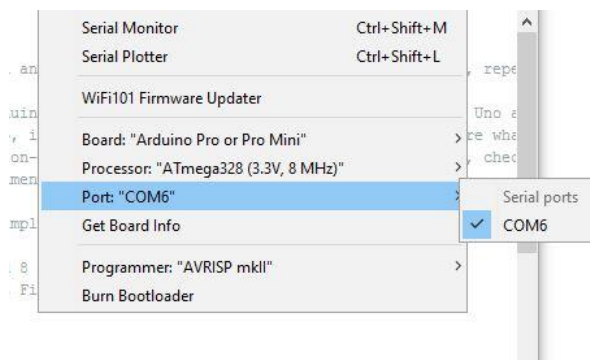
Install the Arduino IDE on your favourite platform (Mac, Linux or Windows). These examples work only with the latest Arduino IDE (tested on 1.6.9)!

<https://www.arduino.cc/en/Guide/ArduinoProMini>

Use the FTDI, ensure that the jumper is on the '3V' side. Battery power turned off (check the switch)

Connect the FTDI and the Pro Mini so that the components are both facing up.

- Start the Arduino IDE
- Select the board: 'Arduino Pro or Pro mini'
- Select the Processor 'ATmega328 (3.3V, 8Mhz)
- Select the COM port given



Select the default 'BLINK' example, change the LED pin from 13 to 10.



THE THINGS NETWORK

```

// Pin 13 has an LED connected on most
// Pin 11 has the LED on Teensy 2.0
// Pin 6 has the LED on Teensy++ 2.0
// Pin 13 has the LED on Teensy 3.0
// give it a name:
int led = 10;

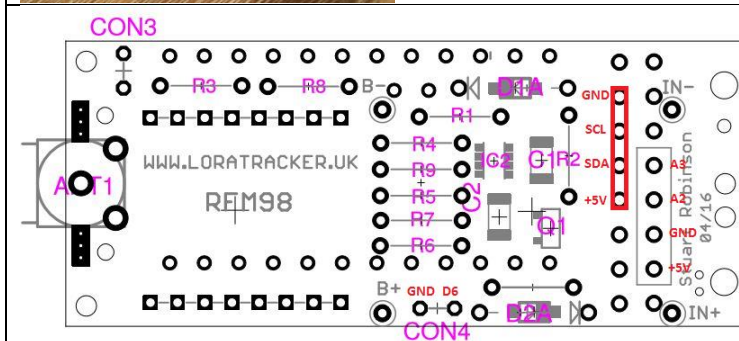
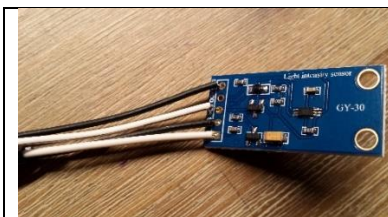
// the setup routine runs once when yo
void setup() {

```

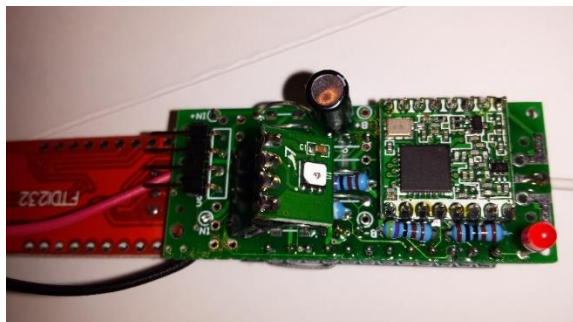
Load the code with CTRL-U to the Pro Mini. After compilation and uploading the led on the board should blink at 1 Hz. Great: we have a working programming environment!

Testing the sensors

1. Download the sensor libraries: <https://github.com/claws/BH1750>
2. Import the library in the Arduino environment: include library -> Add .ZIP library and select the downloaded ZIP file.
3. Select 'manage libraries' and search for 'si7021'. You will find the universal i2c sensor library. Install this one.
4. Connect the GY-30 / BH1750 light sensor: GND-> GND, SDA->SDA, SCL->SCL, VCC->+5V.



5. Run the BH1750test example. By opening the Serial Monitor (9600 bps) you will see the LUX value of the sensor. If you cover the sensor, the value will change.
6. Connect the TH06 (temperature) sensor. It will fit in the header, pointed to the RFM95 radio.



7. Run the i2c_si7021 test example. You will see the temperature and humidity in the monitor. This example uses 115200 as baudrate, select this in the monitor interface!



Getting things started on 'The Things Network'

Download the LMIC library from Github:

1. Goto <https://github.com/matthijskooijman/arduino-lmic>
2. Choose 'Clone or download'
3. Download ZIP
4. Mark the location
5. Go to Arduino IDE
6. Select 'Sketch->Include Library->Add .ZIP Library'
7. Select the downloaded Arduino-lmic-master.zip file from your download location
8. Goto <https://github.com/rockscream/Low-Power>
9. Choose 'Clone or Download'
10. Download ZIP
11. Mark the location
12. Go to Arduino IDE
13. Select 'Sketch->Include Library->Add .ZIP Library'
14. Select the downloaded low-power-master.zip file from your download location

To help joining OTAA faster (seconds instead of 7 minutes) you may alter the LMIC library. You can find LMIC.C file in <My Documents>\Arduino\Libraries\Arduino-lmic-master\src\lmic

1. Open the file in your favorite source code browser (or use WordPad)
2. Go to line 685 (or find 'setDrJoin' to find the line below)
3. Change `setDrJoin(DRCHG_SET, DR_SF7);` to: `setDrJoin(DRCHG_SET, DR_SF9);`
4. Save LMIC.C

The Things Network Dashboard

Your applications and devices can be managed by The Things Network Dashboard.

Production vs staging environment

The production environment is the next step in making TTN a professional supported network. After Q1 2017, the staging environment will no longer be available. If you have a lot of nodes running in staging, it is good practice to migrate them. This is described in <https://www.thethingsnetwork.org/docs/network/migrate.html>, describing to migrate applications, devices and functions. If you already put your node in the staging environment it will be easy to follow the next steps to put your node into the production environment.

Create an Account

To use the dashboard, you need a The Things Network account. You can create an account here:

<https://account.thethingsnetwork.org/users/login>. You can use your 'staging' account also in the production environment.

After registering and validating your e-mail address, you will be able to log in to The Things Network Dashboard.

Create an Application

<https://console.thethingsnetwork.org/applications>

Choose 'add application'



Give your Application a unique ID. You can use ONLY lowercase! You can add an unique number to get uniqueness over the ttn network (this is a global ID)

Your description can be any description you like.

ADD APPLICATION

Application ID
The unique identifier of your application on the network

fb_nodes

Description
A human readable description of your new app

My experimental nodes

☒ **Register application on the handler ttn-handler-eu**
If selected, registers your app to the handler ttn-handler-eu, to make it usable right away

Cancel Add application

First download the code for ttn_temphumi.ino from https://github.com/galagaking/ttn_nodeworkshop

You can download the ZIP and unzip the file to get the examples:

1. Goto https://github.com/galagaking/ttn_nodeworkshop
2. Choose 'Clone or download'
3. Download ZIP
4. Open the ZIP in explorer
5. Open the example ttn_temphumi.ino
6. Create the sketch folder for the application

Register the device

Now register your device. We will use OTAA in this example. This is 'Over the Air Authentication'. You have to define an address for yourself. This is a kind of MAC address, but because there is no registration yet of these, define some RANDOM 8 byte value, using your postcode and some values in between.



Device ID

This is the unique identifier for the device in this app. The device ID will be immutable.

sensor01

Device EUI

The device EUI is the unique identifier for this device on the network. You can change the EUI later.

56 29 AA BB 56 29 CC DD

8 bytes

App Key

The App Key will be used to secure the communication between you device and the network.



App Key will be generated

App EUI

70 B3 D5 7E F0 00 1B 79

DEVICE OVERVIEW

Application ID fb_nodes

Device ID sensor_01

Activation Method OTAA

Device EUI <> 56 29 AA BB 56 29 CC DE hex

Application EUI <> 70 B3 D5 7E D0 00 17 BE hex

App Key <>

Device Address <> 26 01 21 78 hex

There are three values you should copy into your Arduino Code. Dev EUI (the device identifier), App EUI (The application identifier) and the App key.

Dev EUI


With the <> sign you can select different 'views'. We need the 'LSB' view, also called little Endian or Least Significant Byte First.


DEVICE OVERVIEW



Application ID **fb_nodes**

Device ID **sensor_01**

Activation Method **OTAA**

Device EUI **<> { 0xDE, 0xCC, 0x29, 0x56, 0xBB, 0xAA, 0x29, 0x56 } lsb** 

Application EUI **<> { 0xBE, 0x17, 0x00, 0xD0, 0x7E, 0xD5, 0xB3, 0x70 } lsb** 

App Key **<>  hex** 

Device Address **<> 26 01 21 78 hex** 

Both Dev EUI and App EUI will be used in LSB or little Endian format.

First copy the Dev EUI with the clip board sign on the right. Paste the string into your code at the static const u1_t DEVEUI. Remember to respect the semicolon at the end of the line.

```
// This EUI must be in little-endian format, so least-significant-byte
// first. When copying an EUI from ttnctl output, this means to reverse
// the bytes. For TTN issued EUIs the last bytes should be 0xD5, 0xB3,
// 0x70.
```

```
static const u1_t DEVEUI[8] = { 0xDD, 0xCC, 0x29, 0x56, 0xBB, 0xAA, 0x29, 0x56 };
static const u1_t APPEUI[8] = { 0xBE, 0x17, 0x00, 0xD0, 0x7E, 0xD5, 0xB3, 0x70 };
```

```
// This key should be in big endian format (or, since it is not really a
// number but a block of memory, endianness does not really apply). In
// practice, a key taken from ttnctl can be copied as-is.
// The key shown here is the semtech default key.
```

APP EUI

Next copy APP EUI, select LSB for this one as well. Copy to APPEUI in your code.

APP KEY

The last key is a secret hash, so this one just shows when you click the 'EYE' icon. After that you can copy this to the clipboard. This one can be copied in MSB format.

Copy the 16 bytes APPKEY after static const u1_t APPKEY[16] = .

The Temperature sensor should be connected!

Upload your code to the Arduino.

The LED will blink during the JOIN process. The Arduino gets his keys from TTN, thereby all information sent will be encrypted with these keys.

You can also follow this process by activating the 'data' view:

APPLICATION DATA

uplink downlink activation

	cntr	port	dev id	payload	fields
▲ 22:56:24	2	1	sensor_01	D8 12	
▲ 22:55:08	1	1	sensor_01	D8 12	
▲ 22:54:03	0	1	sensor_01	D8 12	
⚡ 22:53:57			sensor_01	Activation	

After a few seconds, the LED will stop blinking. You will see the information coming in into the dashboard. This is coded information, because some algorithm is used to put temperature and humidity in two bytes.

To get the right display format, we can create a decoder in our application. Select your application and open the 'Payload Functions':

PAYLOAD FUNCTIONS

decoder converter validator encoder

```

1 function Decoder(bytes, port) {
2   // Decode an uplink message from a buffer
3   // (array) of bytes to an object of fields.
4   var decoded = {};
5
6   // if (port === 1) decoded.led = bytes[0];
7
8   return decoded;
9 }

```

Enter the function below, overwriting the standard function

```

function Decoder(bytes, port) {

var humi = 20+5*((bytes[1] >> 2) & 0x0F);
var temperature = -2400+6.25*(((bytes[1] & 0x03) << 8) | bytes[0]);
return {
humidity: humi,
celcius: temperature / 100.0
};
}

```

PAYLOAD FUNCTIONS

decoder

converter

validator

encoder

```
1 function Decoder(bytes, port) {
2
3   var humi = 20+5*((bytes[1] >> 2) & 0x0F);
4   var temperature = -2400+6.25*(((bytes[1] & 0x03) << 8) | bytes[0]);
5   return {
6     humidity: humi,
7     celcius: temperature / 100.0
8   };
9 }
10
```

Payload

First test the function with two dummy bytes 01 01, And then Save the function.

Return to your sensor and look what happens:

APPLICATION DATA [clear log](#)

uplink downlink activation Pause

	cntr	port	dev id	payload	fields	
▲	23:06:32	10	1 sensor_01	D7 12	{"celcius":21.4375,"humidity":40}	>
▲	23:05:16	9	1 sensor_01	D7 12	{"celcius":21.4375,"humidity":40}	>
▲	23:04:00	8	1 sensor_01	D7 12		>
▲	23:02:44	7	1 sensor_01	D6 12		>

This way you can put several values into a byte string. Sending in clear ASCII is possible but due to bandwidth limitations not preferable in production environments. To make this even more scalable take a look at <https://github.com/thesolarnomad/lora-serialization> .

To send bytes to our node we can use the 'Download' section. Enter one byte (two hex digits, 05 for example) and click Send.

DOWNLINK

bytes fields FPort 1

05 1 byte

Send

The information will be send to the node after the next time the node contacts the gateway. With a sample rate of approx. 60 seconds, that is the maximum time this data will be received. The Led will blink this amount of times, with a maximum of 10 (0x0A).



The light sensor acts like a temperature sensor. It is also under i2C control. Connect it like described in the sensor test section and load the `ttn_lux` example. You can create another sensor to follow the OTAA procedure once again.

To get the right values we have to change the decoder function:

```
function (bytes,port)
{
  var lux = (((bytes[1] ) << 8) | bytes[0]);
  return { lux: lux
};
}
```

Battery measurement

You want to check your batteries when your node is somewhere out in the field. Pin A0 is connected to a voltage divider connected to the battery. With the `ttn_AnalogReadSerial` example we can check the circuit. The voltage dividers are explained in the code. From the example you can figure out a threshold to get the battery indicator working.

```
318 - Battery OK
318 - Battery OK
318 - Battery OK
318 - Battery OK
317 - Battery OK
318 - Battery OK
318 - Battery OK
318 - Battery OK
318 - Battery OK
```

To integrate this example in the ttn network we have to do two things:

1. Put the battery status into the last free bit of our temperature / humidity example
2. Extract the bit with a function in the console.

First we download our example `ttn_temphumibattery.ino` , find the 'BatteryThreshold' define and fill in a value according to your previous measurements. In the code after measuring the battery voltage, the code will set bit 7 of `byte[1]` to send if the battery voltage is lower than the threshold value.

We add two lines to our Payload Function:

[decoder](#)[converter](#)[validator](#)[encoder](#)

```
1 function Decoder(bytes, port) {  
2   var batlow = ((bytes[1]&0x80) > 0)  
3   var humi = 20+5*((bytes[1] >> 2) & 0x0F);  
4   var temperature = -2400+6.25*((bytes[1] & 0x03) << 8) | bytes[0];  
5   return {  
6     humidity: humi,  
7     celcius: temperature / 100.0,  
8     batterylow: batlow  
9   };  
10 }  
11
```

Payload

By checking bit 7 and put the result in batlow, we will see a battery indicator in our code.

	cntr	port	dev id	payload	fields
▲	00:45:50	11	1 sensor_01	AD 1E	{"batterylow":false,"celcius":18.8125,"humidity":55} >
▲	00:44:41	10	1 sensor_01	AD 9E	{"batterylow":true,"celcius":18.8125,"humidity":55} >

ABP

Activation by Personalization (ABP) is a method where the security keys are stored in the device. Not as safe as the OTAA method, but for experiments it works OK. There is no join procedure, nodes will work right away.

Device ID
This is the unique identifier for the device in this app. The device ID will be immutable.

abp_node

Device EUI
The device EUI is the unique identifier for this device on the network. You can change the EUI later.

56 29 AA BB 56 29 CC DF

App Key
The App Key will be used to secure the communication between you device and the network.

App Key will be generated

App EUI

70 B3 D5 7E D0 00 17 BE

Register the device.

Now edit the settings of the device and choose ABP (OTAA will be selected by default)



Device EUI

56 29 AA BB 56 29 CC DF

App EUI

70 B3 D5 7E D0 00 17 BE

Activation method

☐ OTAA ☐ ABP

Device Address

The device address will be assigned by the network server and is not customizable

Select 'save'. Now some values are system generated and we have to copy them to our code. We use the `ttn_abp.ino` example here.

Device Address <> 26 01 1A 32 hex

Network Session Key <> { 0x21, 0xEE, 0x1E, 0x77, 0x58, 0xBD, 0xFE, 0x47, 0x45, 0x34, 0x... msb

App Session Key <> { 0xFE, 0xC6, 0xE8, 0x6E, 0x4D, 0x31, 0x35, 0x4D, 0xA5, 0x65, ... msb

- Copy the Device Address as a HEX value to DEVADDR in the example, so 26 01 1A 32 will be 0x26011A32.
- Copy the Network Session Key as MSB to NWSKEY.
- Copy the App Session Key as MSB to APPSKEY.

Compile and upload the code. Check in the dashboard the working.

You might want to uncheck the frame counter check

Frame counter width

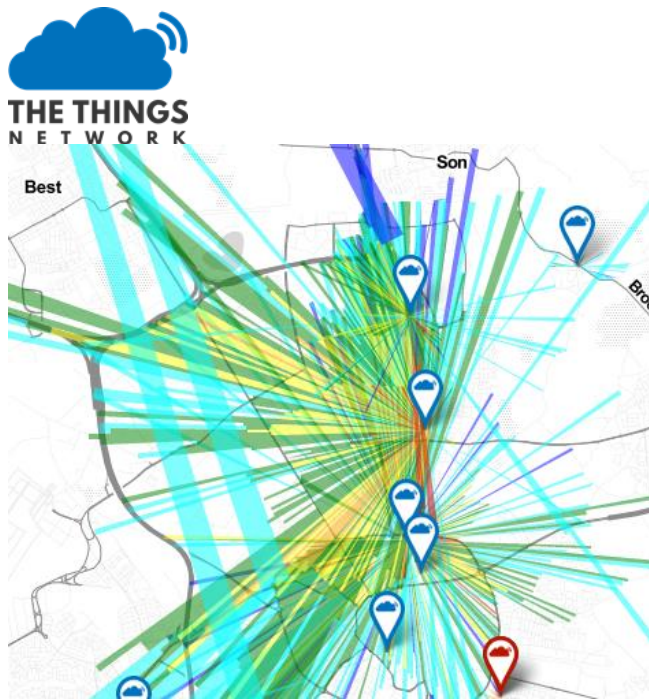
☐ 16 bit ☐ 32 bit

☒ Frame counter checks

If the frame counter is checked, you must respect the sequence number, and probably copied packages are refused on the network. Though restarting you node, and thereby resetting the frame counter, will disable your node. So to get this working, you have to disable this check by unchecking this box. You can integrate the code of `ttn_temphumibattery` in the `abp` example.

TTN Mapper

On www.ttnmapper.org you can look at the coverage of The Things Network in your neighbourhood.



You can even contribute to this map by using the ttnmapper app on your Android Smartphone. At this moment only the staging environment is supported.

LMIC Pin Mapping

If you are using other 'LMIC' or TTN examples, there is ONE part to take care of, the pin setting of the RFM95 module. Most times you have to adjust this to the pinout of the board:

```
// Pin mapping is hardware specific
const lmic_pinmap lmic_pins = {
    .nss = 8,
    .rxtx = LMIC_UNUSED_PIN,
    .rst = 9,
    .dio = {2,5, LMIC_UNUSED_PIN}, //DIO0 and DIO1 connected
};
```

And even more...

- You can add professional Antenna's on the PCB by using the appropriate connector.
- On the 4 pin connector you can add a GPS or other general purpose I/O (Using pins A2 and A3 on the Arduino).
- Support: <https://www.thethingsnetwork.org/forum/>

Thanks to Stuart Robinson for his PCB design and sharing his manuals: www.loratracker.uk and <http://www.loratracker.uk/?p=30> (we use the RFM98 version of the board, replacing the RFM98 for RFM95).

Frank Beks

December 2016