LoRaWAN

Tutorial
Raspberry Pi Single Gateway and Arduino Node

Oene Bakker © 2018
1 Introduction

1.1 Preface

LoRa (Long Range) and LoRaWAN (LoRa Wide Area Network) are treated in this book.

In the current world, more and more devices are linked together. Often via WiFi or a mobile connection (3G / 4G). Mobile connections generally cost money and Wi-Fi is not available everywhere. There are also situations where devices are completely out of the reach of any network.

In addition, it is possible that there is no electricity available in the places where these devices are placed. If this is the case, the equipment is dependent on batteries.

LoRa communication was introduced for these situations. Features are low energy consumption and a wide long range.

A fully-fledged LoRaWAN compatible network costs hundreds to thousands of euros. Usually too expensive for an "normal" hobbyist.

In this book we will make a simple, relatively inexpensive LoRa network with a Raspberry Pi and Arduino Mega. This solution has the necessary limitations but offers a good introduction to LoRa and LoRaWAN.

In this solution we will also use The Things Network (TTN).

1.2 Accountability

The texts and images included in this book come as far as I as a writer could verify from the so-called free domain (free pics, public text). In the unlikely event that this is not the case, I hereby offer my sincere apologies.

This book has been compiled with the utmost care and the solutions shown have been extensively tested. Should errors nevertheless have crept into the text or sources, this has been done without any intention. Despite all the care devoted to the composition of this book, the author can not be held liable for any damage resulting from any error in this publication.

The starting point is that the reader is in possession of a PC, laptop or tablet with preferably Windows 10.

Working with the necessary hardware and voltage adapters can be a risk if careless handling is involved. The solution shown works with low voltages (max 5 volts), which of course limits the risks. Wrong connection can of course damage certain hardware irreparably! The author can not be held liable for any damage resulting from this. It is all entirely at your own risk.
The contents of this book may not be used commercially. The reader is free to use the contents of this book for private and hobby purposes. This also applies to use in education. The sources belonging to this book may be copied, used and / or changed without restriction.

The content is translated from Dutch to English so there may be some flaws in the translation.

Also some software may use Dutch menus, buttons, messages and texts.

### 1.3 Abbreviations

<table>
<thead>
<tr>
<th>Afkorting</th>
<th>Betekenis</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABP</td>
<td>Activation By Personalisation</td>
</tr>
<tr>
<td>CN</td>
<td>China</td>
</tr>
<tr>
<td>DR</td>
<td>Data Rate</td>
</tr>
<tr>
<td>EU</td>
<td>Europe</td>
</tr>
<tr>
<td>EUI</td>
<td>Extended Unique Identifier</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>JDK</td>
<td>Java Development Kit</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>LMIC</td>
<td>LoraMAC-In-C libary</td>
</tr>
<tr>
<td>LoRa</td>
<td>Long Range</td>
</tr>
<tr>
<td>LoRaWAN</td>
<td>Long Range Wide Area Network</td>
</tr>
<tr>
<td>M2M</td>
<td>Machine to Machine</td>
</tr>
<tr>
<td>MQTT</td>
<td>Message Queuing Telemetry Transport</td>
</tr>
<tr>
<td>OTAA</td>
<td>Over The Air Activation</td>
</tr>
<tr>
<td>RF</td>
<td>Radio Frequency</td>
</tr>
<tr>
<td>RPI</td>
<td>Raspberry Pi</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SF</td>
<td>Spreading Factor</td>
</tr>
<tr>
<td>SSH</td>
<td>Secure Shell</td>
</tr>
<tr>
<td>TTN</td>
<td>The Things Network</td>
</tr>
<tr>
<td>US</td>
<td>United Stated</td>
</tr>
<tr>
<td>WAN</td>
<td>Wide Area Network</td>
</tr>
<tr>
<td>Wi-Fi</td>
<td>Wireless Fidelity</td>
</tr>
</tbody>
</table>
2 Content

Content

1 Introduction.................................................................................................................. 2
  1.1 Preface.................................................................................................................... 2
  1.2 Accountability......................................................................................................... 2
  1.3 Abbreviations ......................................................................................................... 3

2 Content ......................................................................................................................... 4

3 What is a LoRaWAN? ................................................................................................. 7
  3.1 LoRa........................................................................................................................ 7
    3.1.1 Things (sensors and actors) .............................................................................. 7
  3.2 LoRaWAN .............................................................................................................. 8
  3.3 The Things Network (TTN) .................................................................................. 9

3.4 Architecture ............................................................................................................ 9
  3.4.1 Nodes ............................................................................................................... 9
  3.4.2 Gateways ......................................................................................................... 9
  3.4.3 Application ....................................................................................................... 9
  3.4.4 Uplink and downlink messages ...................................................................... 10
  3.4.5 Architecture map ............................................................................................. 10
  3.4.6 Frequencies ..................................................................................................... 11
  3.4.7 Data frequency ................................................................................................. 11

4 Raspberry Pi Single Channel Gateway .................................................................. 13
  4.1 Introduction .......................................................................................................... 13
  4.2 Hardware ............................................................................................................. 13
  4.3 Headless installation ............................................................................................ 15
    4.3.1 Download Raspbian Stretch ........................................................................... 15
    4.3.2 Prepare SD card ............................................................................................... 16
    4.3.3 Transfer operation system .............................................................................. 17
    4.3.4 Headless installation ...................................................................................... 19
    4.3.5 Start Raspberry Pi and determine IP-address ................................................. 20
    4.3.6 SSH via PuTTY ................................................................................................. 21
    4.3.7 Configure Raspbian Stretch .......................................................................... 24
    4.3.8 Install Gateway software ................................................................................. 31

Versie 1.0 4 Date: 18-07-2018
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.6.2</td>
<td>Directory structure</td>
<td>80</td>
</tr>
<tr>
<td>8.6.4</td>
<td>Dependencies: pom.xml</td>
<td>81</td>
</tr>
<tr>
<td>8.6.5</td>
<td>PushBullet.java</td>
<td>82</td>
</tr>
<tr>
<td>8.6.6</td>
<td>TTNClient.java</td>
<td>83</td>
</tr>
<tr>
<td>8.6.7</td>
<td>Compile and test</td>
<td>85</td>
</tr>
</tbody>
</table>
What is a LoRaWan?

To get a good understanding of what this book is about, first a piece of theory. A number of terms that are discussed are LoRa, LoRaWAN, nodes and frequencies.

3.1 LoRa

LoRa stands for Long Range. It is a patented technology from the company SemTech (see: https://www.semtech.com/technology/lora).

Features are:
- low energy use
- a wide range (up to 15 kilometers in rural areas and up to 2 kilometers in urban areas)
- secure communication in both directions
- a limited data rate (at least 300, to a maximum of 50,000 bytes per second)

LoRa uses a technology called spread spectrum. Spread spectrum is a form of wireless communication in which the energy of the transmitted signal is deliberately distributed over a certain frequency domain. Such scattered signals have a much larger bandwidth than the information they contain, creating a noisy signal that is difficult to detect or to intercept. Furthermore, it is also very difficult to disturb a spread spectrum signal with a different signal. Because of these properties, spread spectrum is ideal for military applications and other environments where a high reliability of the signal is desired.

For broadcasting a spread spectrum we use so-called RF (Radio Frequency) chips. This tutorial uses a the RFM95W HopeRF transceiver.

3.1.1 Things (sensors and actors)

LoRa is therefore also ideal for Internet-of-Things (IoT) applications. With LoRa all sorts of things can talk to the internet without having to use mobile communication (3G / 4G) or Wi-Fi. Ideal for places where no WiFi is available or where no mobile communication is possible. This requires a so-called gateway to the internet. In other words thing talks to a gateway. Gateways then communicate with the internet. Things can be sensors or actors.

A sensor reports information about for example as the temperature and humidity. An actor can provide control from a given assignment such as turning a light on or off.

Sensors and actors are relatively inexpensive and they can be used for all kinds of applications. A few examples:
- Remote opening of a garage door.
- Locating a pet.
- Measuring the air quality.
- Measuring the water level in a ditch.
- Registering whether or not a parking space is occupied.
In principle things don’t constantly seek connection with the gateway, but only when needed.

Things uses the so-called Wide Area Network (WAN) technology and therefore are ideal for all kinds of applications that are beyond the reach of WiFi network or Local Area Network (LAN).

For example, LoRa could be used for smart cities. In such cities all kinds of devices are connected to each other that are normally not connect. Examples are lampposts, parking meters, traffic control installations, location monitors, air quality monitors and so on.

With LoRa, communities have a good opportunity to bring their countless IoT applications together at an affordable rate.

Another characteristic of LoRa is that there is seamless interoperability between smart things without the need for locally complex installations. Since the costs are relatively low (certainly compared to other technologies) it is possible to install secure networks. Not only for the professionals but also for the hobbyist.

3.2 LoRaWAN

LoRaWAN (Long Range Wide Area Network) is a specification for a telecommunication network suitable for long-distance communication with little power. The technology is used for machine-to-machine communication (M2M) and is therefore ideal for the internet of things (IoT).

LoRaWAN is designed to allow devices with low power to communicate with applications connected to the Internet via long-distance wireless connections. The LoRaWAN protocols are defined by the LoRa Alliance and formalized in the LoRaWAN specification that can be requested on the LoRa Alliance website (https://lora-alliance.org/lorawan-for-developers).
3.3 The Things Network (TTN)

The Things Network (TTN) started in 2015 in Amsterdam. The city of Amsterdam has been covered since July 2015 with the open LoRa (Long Range) network that is open and free to use. The Things Network is a community to create an internet-of-things data network with the LoRaWAN technology. With a simple 1000-euro LoRa router you can connect 10,000 devices, explains founder Wienke Giezeman. That was the start of the IoT data network in Amsterdam that The Things Network set up together with ten parties in the capital. All these parties purchased a lora gateway and on 21 August the city coverage network was live.

The roll-out of the TTN network is therefore financed by users that install gateways (antennas). At the moment, people from more than 600 cities from more than 90 countries worldwide are working on realizing the project.

The technique is specifically designed for low-cost, energy-efficient sensors that can communicate over a large distance. The technology is widely used for applications such as smart parking systems, automation in agriculture and smart water management. Bicycles, boats and pets can be also connected to the internet.

3.4 Architecture

3.4.1 Nodes

A Node or end device is a small device or microcontroller that is connected to LoRaWAN, for example an Arduino or Raspberry Pi. Nodes are used to collect data (i.e. temperature) or to receive commands (i.e. turn on the light). The devices use LoRa modules to communicate with the LoRaWAN. The nodes send encrypted messages.

3.4.2 Gateways

Gateways receive messages from nodes. A message sent by a node is received by all gateways within the range of that node. A gateway is connected to the internet and forwards the messages to network servers. In this book the messages are sent to the TTN.

3.4.3 Application

An application uses the messages that are available on TTN. The application can use these messages to execute certain actions, for example sending an SMS to warn the owner of a certain event. This book uses a TTN client application that sends alerts to the PushBullet mobile app. But you also could send an email or sms.
3.4.4 Uplink and downlink messages

An uplink message is a message that is sent via the node to the application that then can process the message.

A downlink message is a message that is sent via the application to the node that then can process the message.

This book only uses uplink messages. Downlink messages are not (yet) supported by the software that is used.

The solution therefore is LoRaWAN compatible, but the solution does NOT fully comply with the LoRaWAN specifications. This does not have to be an objection in gaining experience with LoRaWAN.

3.4.5 Architecture map

N = Node
G = Gateway

3.4.6 Frequencies

LoRaWAN uses so-called free radio frequencies that are currently specified as follows:

<table>
<thead>
<tr>
<th>Region</th>
<th>Frequency Ranges</th>
<th>Region Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Europa</td>
<td>863-870 MHz and 433-434 MHz</td>
<td>EU868 en EU433</td>
</tr>
<tr>
<td>United States</td>
<td>902-928 MHz</td>
<td>US915</td>
</tr>
<tr>
<td>China</td>
<td>470-510 MHz and 779-787 MHz</td>
<td>CN780</td>
</tr>
</tbody>
</table>

All frequencies are listed in the PDF document:

Gateways and nodes in Europe operate at 868 MHz and in the United States at 915 MHz. It is a good thing to pay attention to this when purchasing hardware.

TTN uses three channels with the following frequencies:
- Channel 0: 868.1 MHz
- Channel 1: 868.3 MHz
- Channel 2: 868.5 MHz

3.4.7 Data frequency

The data frequency relies on:
- Transmission power (tx value),
- Bandwidth and
- spreading factor (SF).

If you lower the tx value, you save on the battery, but the range of the signal is shorter.

Bandwidth and spreading factor together form the data rate. This determines how quickly the bytes are sent.

With LoRa, the signal (chirp) is transmitted over the entire bandwidth. How long this signal is transmitted is called a spreading factor (SF). This is the shortest with SF7 and with SF12 this is the longest.

If you increase the data rate (making the bandwidth wider or the spreading factor lower), the bytes are sent in a shorter time.

By making the bandwidth 2x wider (for example from 125 KHz to 250 KHz), you can send two extra bytes in one go.

By making the spreading factor 1 step lower (for example from SF10 to SF9), you can send two extra bytes in one go. By reducing the spreading factor, however, it becomes more difficult for the gateway to receive a transmissions because it is more sensitive to noise. You could compare this with two people who occupy a noisy place (for example in a bar). If you are far apart, you have to talk slowly (SF12), but if you are close, you can talk faster (SF7).
TTN has three bandwidths: 125, 250 and 500 KHz. This depends on the frequency plan. In Europe the bandwidth 500 KHz is not used. In addition, TTN has six spreading factors: SF7 to SF12.

Overview for EU868 and EU433:

<table>
<thead>
<tr>
<th>Data Rate</th>
<th>Configuratie</th>
<th>bits/seconde</th>
<th>Max payload (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DR0</td>
<td>SF12/125kHz</td>
<td>250</td>
<td>59</td>
</tr>
<tr>
<td>DR1</td>
<td>SF11/125kHz</td>
<td>440</td>
<td>59</td>
</tr>
<tr>
<td>DR2</td>
<td>SF10/125kHz</td>
<td>980</td>
<td>59</td>
</tr>
<tr>
<td>DR3</td>
<td>SF9/125kHz</td>
<td>1760</td>
<td>123</td>
</tr>
<tr>
<td>DR4</td>
<td>SF8/125kHz</td>
<td>3125</td>
<td>230</td>
</tr>
<tr>
<td>DR5</td>
<td>SF7/125kHz</td>
<td>5470</td>
<td>230</td>
</tr>
<tr>
<td>DR6</td>
<td>SF7/250kHz</td>
<td>11000</td>
<td>230</td>
</tr>
</tbody>
</table>
4 Raspberry Pi Single Channel Gateway

4.1 Introduction

To set up a single channel gateway we use a Raspberry Pi Model 3B (RPI) and a Dragino Lora / GPS HAT v1.4.

4.2 Hardware

Hardware used:
- Raspberry Pi 3(b) (RPI)
- Dragino v1.4 hat

The Dragino can be placed on the RPI without any problems. No further installation is required.

Note: This Dragino also has a GPS, but this is not used in this book.
Our Dragino has a Hope RF96 chip which uses SX1276 chipset.

It works on 868MHz:
4.3 Headless installation

4.3.1 Download Raspbian Stretch

Download the latest version of Raspbian Stretch Lite.


Unzip the ZIP file:
4.3.2 Prepare SD card

Use a Micro SD card of at least 8 GB. Use a Micro SD adapter if necessary.

Download and install the SD Card formatter.

Download link: https://www.sdcard.org/downloads/index.html

Start SD Card formatter, select the SD card → Format

→ Yes (Ja)
4.3.3 Transfer operation system

Download and install Etcher.

Download link: https://etcher.io/

Start Etcher.
Select the Raspbian Stretch Lite image.

→ Flash!
This can take some time!

4.3.4 Headless installation

To set up SSH and WiFi, two files must be created in the boot part of the SD card.

- Gandalf (C:)
- Recover (D:)
- boot (E:)
- USB-station (G)

Create two files:
- ssh
- wpa_supplicant.conf
The ssh files should be an empty file.

Content of the wpa_supplicant.conf file:

```plaintext
country=NL
ctrl_interface=DIR=/var/run/wpa_supplicant
update_config=1

network={
  ssid="MySSID"
  psk="MySecretPassword"
  scan_ssid=1
}
```

Change country, ssid and psk to fit your settings.

### 4.3.5 Start Raspberry Pi and determine IP-address

Start the Raspberry Pi with just prepared SD card. The start-up can take a few minutes.

Download IP Scan.

Download link: [https://www.advanced-ip-scanner.com/nl/](https://www.advanced-ip-scanner.com/nl/)

Start (or install and start) IP Scan. Click Scan and wait for the scan to finish. It may take some time before all IP addresses become visible. If necessary, start a new scan if the IP address of the Raspberry Pi is not yet visible.
4.3.6 SSH via PuTTY

Download and install PuTTY.

Download link: [https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html](https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html)

Start PuTTY
Host Name: IP-adres

PuTTY Configuration

PuTTY Security Alert

WARNING - POTENTIAL SECURITY BREACH!

The server's host key does not match the one PuTTY has cached in the registry. This means that either the server administrator has changed the host key, or you have actually connected to another computer pretending to be the server.

The new ssh-ed25519 key fingerprint is:
ssh-ed25519 2048

If you were expecting this change and trust the new key, hit Yes to update PuTTY's cache and continue connecting.

If you want to carry on connecting but without updating the cache, hit No.

If you want to abandon the connection completely, hit Cancel. Hitting Cancel is the ONLY guaranteed safe choice.

Yes (Ja)
Log in:
- Login as: pi
- Password: raspberry
4.3.7 Configure Raspbian Stretch

Command: sudo raspi-config

Result:
4.3.7.1 Change password

To prevent unauthorized login, the password must be changed with option 1 Change User Password.

```
pi@raspberrypi ~
pi@raspberrypi:~ $ sudo raspi-config
Enter new UNIX password:
Retype new UNIX password:
```

```
pi@raspberrypi ~
Password changed successfully
```
4.3.7.2 Change localisation settings

With option 4 Localization Options the localization options can be set.

Option I1 Change Locale and <Select> with the tab key:

Scroll down with the cursor key and delete selection at en_GB.UTF-8 UTF-8 with the space key:

Scroll down and select nl_NL.UTF-8 UTF-8 (or your own country) with the space key and <Ok> (tab key).

Select nl_NL.UTF-8 (or your selection) and <Ok>
4.3.7.3 Change timezone

Select option 4 and I2 Change Timezone and <Select>

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>Change Locale</td>
</tr>
<tr>
<td>I2</td>
<td>Change Timezone</td>
</tr>
<tr>
<td>I3</td>
<td>Change Keyboard Layout</td>
</tr>
<tr>
<td>I4</td>
<td>Change Wi-fi Country</td>
</tr>
</tbody>
</table>

Set up language and regional settings
Set up timezone to match your location
Set the keyboard layout to match
Set the legal channels used in your location

Select Europe (or your own geographic area) and <Ok>

Geographic area:

- Asia
- Atlantic Ocean
- Europe
- Indian Ocean
- Pacific Ocean
- System V timezones
- US
- None of the above

Select Amsterdam (or your own time zone) and <Ok>

Time zone:

- Amsterdam
- Andorra
4.3.7.4 Change interfacing options

Select option 5 Interfacing Options, P4 SPI and <Select>

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1 Camera</td>
<td>Enable/Disable connection to the</td>
</tr>
<tr>
<td>P2 SSH</td>
<td>Enable/Disable remote command line</td>
</tr>
<tr>
<td>P3 VNC</td>
<td>Enable/Disable graphical remote access</td>
</tr>
<tr>
<td>P4 SPI</td>
<td>Enable/Disable automatic loading</td>
</tr>
<tr>
<td>P5 I2C</td>
<td>Enable/Disable automatic loading</td>
</tr>
<tr>
<td>P6 Serial</td>
<td>Enable/Disable shell and kernel modules</td>
</tr>
<tr>
<td>P7 1-Wire</td>
<td>Enable/Disable one-wire interface</td>
</tr>
<tr>
<td>P8 Remote GPIO</td>
<td>Enable/Disable remote access to GPIO</td>
</tr>
</tbody>
</table>

Select <Yes>

Select OK
Reboot the Raspberry Pi with the command: sudo reboot
4.3.7.5 Update Raspbian Stretch

Restart PuTTY and login with the new password.

Update the operation system with the commands:
- `sudo apt-get update`
- `sudo apt-get upgrade`

This will take some time. Answer Y (J) on the displayed question:
4.3.8 Install Gateway software

You need the following installations on your RPI:
• WiringPi
• Git
• Single Channel Packet Forwarder software

WiringPi is a so-called GPIO interface library which is used for interfacing with the Dragino LoRa shield.

Git is required for retrieving the single channel packet forwarder software.

The single channel packet software is required for setting up the gateway. At the end of 2017 it seemed that TTN would no longer allow single channel packet software.

In a TTN statement:
Single-Channel gateways are not LoRaWAN compliant and are currently not officially supported by The Things Network.

Fortunately, an alternative has come (for the time being?). Which is discussed in this book.

4.3.8.1 WiringPi

Command: sudo apt-get install wiringpi

```
pi@raspberrypi:~
pi@raspberrypi:~ $ sudo apt-get install wiringpi
```

4.3.8.2 Git

Command: sudo apt-get install git

```
pi@raspberrypi:~
pi@raspberrypi:~ $ sudo apt-get install git
```

Answer Y (J) on the asked question.
4.3.8.3 Single channel packet software

Commands:
- `cd ~`
- `mkdir scpf`
- `cd scpf`
- `git clone https://github.com/hallard/single_chan_pkt_fwd`

Change `global.conf`

Command: `nano global_conf.json`

For the Dragino, the following changes must be made to the pin configuration:
- "pin_nss": 6,
- "pin_dio0": 7,
- "pin_rst": 0

It is also useful to adjust the gateway configuration. Here we are a bit ahead on the TTN configuration (chapter 5).

If you know the GPS position of the gateway, it can be entered here (optional).
Save the file with Ctrl + X, J and Enter.

The software now has to be compiled and linked.

Commands:
- cd single_chan_pkt_fwd
- make

Note:
For TTN you can use the following three channels (channels):
- Channel 0 = “freq” : 868100000
- Channel 1 = “freq” : 868300000
- Channel 2 = “freq” : 868500000
4.3.9 Limitations

This gateway solution has the following limitations:
1. The gateway has only one channel. A LoRaWAN compatible gateway has 8.
2. The gateway works with only one spreading factor (SF).
3. The gateway does not (yet) have a downlink function *)

*) There is a downlink solution but for that you need two Raspberry Pi's and two Draginos.

Normally nodes (end devices) will use all three channels. This means that in principle 1 in 3 messages are picked up and processed by the Gateway. Our gateway will only receive messages on channel 0. Changing channels is called frequency hopping. In this book this is solved by having the node only send messages channel 0 (so channels 1 and 2 are skipped).

In addition, the gateway and node must use the same spreading factor. In this book we use SF7.
4.4 Start gateway

The gateway is started with the command: `./single_chan_pkt_fwd`

```
pi@raspberrypi:~$./scpif/single_chan_pkt_fwd

pi@raspberrypi:~$./scpif/single_chan_pkt_fwd
```

Result:

```
pi@raspberrypi:~$./scpif/single_chan_pkt_fwd
pi@raspberrypi:~/scpif/single_chan_pkt_fwd $ ./single_chan_pkt_fwd

Gateway ID: b8:27:eb:ff:ff:20:ca:5b
Listening at SF7 on 868.100000 MHz.

stat update: 2018-05-18 17:59:21 GMT no packet received yet
```

Please make a note of the Gateway ID (we need it later on):

```
SX1276 detected, starting.
Gateway ID: b8:27:eb:ff:ff:20:ca:5b
Listening at SF7 on 868.100000 MHz.
```

Remove the “:” by a space and convert the lowercase letters to uppercase letters. In this example, the gateway is declared as: `B8 27 EB FF FF 20 CA 5B`

The gateway is now ready for use. In the next step, the gateway will be announced in TTN.
5 Add the gateway to TTN

5.1 Create a TTN account

Open a browser and go to: https://www.thethingsnetwork.org/

Click SIGN UP and create an account.

5.2 TTN add a gateway

Open a browser and go to: https://www.thethingsnetwork.org/

Click LOGIN.

Select CONSOLE from the dropdown (right top of the page):
Click GATEWAYS to create a new gateway:

Click register gateway:

You do not have any gateways
Get started by registering one!
Input fields:

- Gateway ID: B8 27 EB FF FF 20 CA 5B
- Check I'm using the legacy packet forwarder
- Description: choose your own description
- Frequency Plan: Europe 868MHz
- Router: ttn-router-eu (wordt automatisch gevuld)

Note: Some of the values have already been discussed in section 4.3.8.3 (global_conf.json).
Location
The exact location of your gateway. This will be used if your gateway cannot determine its location by itself. Set a location by clicking on the map.

Antenna Placement
The placement of the gateway antenna

indoor  outdoor

Click Register Gateway at the bottom of the page:
If the Raspberry Pi Gateway is still active, the following Overview will be shown in TTN:

![Gateway Overview Image]

You also can check the status in the browser with the link:

http://noc.thethingsnetwork.org:8085/api/v2/gateways/eui-xxxxxxxxxxxxxxxx

In this example:
http://noc.thethingsnetwork.org:8085/api/v2/gateways/eui-b827ebffff20ca5b

5.3 Add application

Click Applications:
Click add application:

![Image of The Things Network console](image)

Input fields:
- Application ID: choose your own, but only use alphanumerics, no uppercase letters and use only – and _.
- Description: choose your own.

ADD APPLICATION

**Application ID**
The unique identifier of your application on the network
appid001001

**Description**
A human readable description of your new app
Waldpyk Sensor Network

Scroll down and click Add application.
The application is added:

![Application Details](image-url)
5.4 Add device

Click register device:

Input field:
- Device ID: Choose your own.

Klik op om het Device EUI automatisch te genereren.

**REGISTER DEVICE**

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

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

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

this field will be generated
Click Register:

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

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

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

App EUI
78 B3 D5 7E D0 00 F8 1A

Click Settings:

Applications > appid001001 > Devices > appid001002

DEVICE OVERVIEW

Application ID appid001001
Device ID appid001002

Activation Method OTAA

Device EUI 00 C2 C6 87 02 71 8E 84
Application EUI 78 B3 D5 7E D0 00 F8 1A
App Key ✋
Select ABP (Activation by Personalization)

In ABP mode it is advisable to switch off the Frame Counter Checks:

- **Frame Counter Checks**
  - Disabling frame counter checks drastically reduces security and should only be used for development purposes

If the Frame Counter Checks is not turned off, data may not be shown on the console.

Scroll down and click Save
Scroll down where the various keys are shown:

Make a copy of the keys with:

- Device EUI: 00C2C6B70271BEB4
- Application EUI: 70B3D57ED000F01A
- Device Address: 26011235
- Network Session Key: FD 19 C6 3D 0D A9 78 E3 8D C7 A7 5D 9C 3A D7 2D
- App Session Key: 93 4C 4E 59 1C A9 6C 20 EB C8 F4 C8 55 99 5A 40
6 Node

6.1 Arduino hardware setup

Required:
• Arduino Mega
• Dragino LoRa hat

The installation is simple, the Dragino can be placed on the Arduino Mega without any further installation.
This Dragino uses a SX1276 chipset.

And works on 868 MHz.
6.2 Arduino IDE

Download and install the Arduino IDE if necessary.

Download link: https://www.arduino.cc/en/Main/Software
6.3 TTN library

Start the Arduino IDE.

Select menu, Sketch, Include Library and Manage Libraries
Look for lmic

Select IBM LMIC framework and Install

Result:
6.4 Hello World sketch

With the Arduino sketch in this example a LoRaWAN message is sent with the text: Hello, world!

Connect the Arduino Mega to the PC or laptop.

Select Tools in the menu, Board, Arduino Mega ADK

Select menu, Tools, Port and the active com port, in this example COM4.
6.4.1 Arduino sketch

Copy the following source code from Thomas Telkamp and Matthijs Kooijman and change the following variables:

- **NWKSKEY**
- **APPSKEY** and
- **DEVADDR**

/* *******************************************************************************
* Copyright (c) 2015 Thomas Telkamp and Matthijs Kooijman
* Permission is hereby granted, free of charge, to anyone
* obtaining a copy of this document and accompanying files,
* to do whatever they want with them without any restriction,
* including, but not limited to, copying, modification and redistribution.
* NO WARRANTY OF ANY KIND IS PROVIDED.
* This example sends a valid LoRaWAN packet with payload "Hello, world!", using frequency and encryption settings matching those of
* the (early prototype version of) The Things Network.
* Note: LoRaWAN per sub-band duty-cycle limitation is enforced (1% in g1,
* 0.1% in g2).
* Change DEVADDR to a unique address!
* See http://thethingsnetwork.org/wiki/AddressSpace
* Do not forget to define the radio type correctly in config.h.
* *************************************************/

#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>

// LoRaWAN NwkSKey, network session key
// This is the default Semtech key, which is used by the prototype TTN
// network initially.
static const PROGMEM u1_t NWKSKEY[16] = { 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX,
0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX};

// LoRaWAN AppSKey, application session key
// This is the default Semtech key, which is used by the prototype TTN
// network initially.
static const u1_t PROGMEM APPSKY[16] = { 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX,
0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX};

// LoRaWAN end-device address (DevAddr)
// See http://thethingsnetwork.org/wiki/AddressSpace
static const u4_t DEVADDR = 0xFFFFFF00;

// These callbacks are only used in over-the-air activation, so they are
// left empty here (we cannot leave them out completely unless
// DISABLE_JOIN is set in config.h, otherwise the linker will complain).
void os_getArtEui (u1_t* buf) { }
void os_getDevEui (u1_t* buf) { }
void os_getDevKey (u1_t* buf) { }

static uint8_t mydata[] = "Hello, world!";
static osjob_t sendjob;

// Schedule TX every this many seconds (might become longer due to duty
// cycle limitations).
const unsigned TX_INTERVAL = 60;
```c
// Pin mapping
const lmic_pinmap lmic_pins = {
   .nss = 10,
   .rxtx = LMIC_UNUSED_PIN,
   .rst = 9,
   .dio = {2, 6, 7},
};

void onEvent (ev_t ev) {
   Serial.print(os_getTime());
   Serial.print(":");
   switch(ev) {
      case EV_SCAN_TIMEOUT:
         Serial.println(F("EV_SCAN_TIMEOUT"));
         break;
      case EV_BEACON_FOUND:
         Serial.println(F("EV_BEACON_FOUND"));
         break;
      case EV_BEACON_MISSED:
         Serial.println(F("EV_BEACON_MISSED"));
         break;
      case EV_BEACON_TRACKED:
         Serial.println(F("EV_BEACON_TRACKED"));
         break;
      case EV_JOINING:
         Serial.println(F("EV_JOINING"));
         break;
      case EV_JOINED:
         Serial.println(F("EV_JOINED"));
         break;
      case EV_RFU1:
         Serial.println(F("EV_RFU1"));
         break;
      case EV_JOIN_FAILED:
         Serial.println(F("EV_JOIN_FAILED"));
         break;
      case EV_REJOIN_FAILED:
         Serial.println(F("EV_REJOIN_FAILED"));
         break;
      case EV_TXCOMPLETE:
         Serial.println(F("EV_TXCOMPLETE (includes waiting for RX windows)"));
         if(LMIC.dataLen) {
            // data received in rx slot after tx
            Serial.print(F("Data Received: ");
            Serial.write(LMIC.frame+LMIC.dataBeg, LMIC.dataLen);
            Serial.println();
         }
         // Schedule next transmission
         os_setTimedCallback(&sendjob, os_getTime()+sec2osticks(TX_INTERVAL), do_send);
         break;
      case EV_LOST_TSYNC:
         Serial.println(F("EV_LOST_TSYNC"));
         break;
      case EV_RESET:
         Serial.println(F("EV_RESET"));
         break;
      case EV_RXCOMPLETE:
         // data received in ping slot
         Serial.println(F("EV_RXCOMPLETE"));
         break;
      case EV_LINK_DEAD:
         Serial.println(F("EV_LINK_DEAD"));
         break;
      case EV_LINK_ALIVE:
         Serial.println(F("EV_LINK_ALIVE"));
         break;
      default:
         Serial.println(F("Unknown event"));
         break;
   }
}
```
void do_send(osjob_t* j) {
    // Check if there is not a current TX/RX job running
    if (LMIC.opmode & OP_TXRXPEND) {
        Serial.println(F("OP_TXRXPEND, not sending");
    } else {
        // Prepare upstream data transmission at the next possible time.
        LMIC_setTxData2(1, mydata, sizeof(mydata)-1, 0);
        Serial.println(F("Packet queued");
    }
    // Next TX is scheduled after TX_COMPLETE event.
}

void setup() {
    Serial.begin(115200);
    Serial.println(F("Starting");
    #ifdef VCC_ENABLE
    // For Pinoccio Scout boards
    pinMode(VCC_ENABLE, OUTPUT);
    digitalWrite(VCC_ENABLE, HIGH);
    delay(1000);
    #endif
    // LMIC init
    os_init();
    // Reset the MAC state. Session and pending data transfers will be discarded.
    LMIC_reset();
    // Set static session parameters. Instead of dynamically establishing a session
    // by joining the network, precomputed session parameters are be provided.
    #ifdef PROGMEM
    // On AVR, these values are stored in flash and only copied to RAM
    // once. Copy them to a temporary buffer here, LMIC_setSession will
    // copy them into a buffer of its own again.
    uint8_t appskey[sizeof(APPSKEY)];
    uint8_t nwkskey[sizeof(NWKSKEY)];
    memcpy_P(appskey, APPSKEY, sizeof(APPSKEY));
    memcpy_P(nwkskey, NWKSKEY, sizeof(NWKSKEY));
    LMIC_setSession (0x1, DEVADDR, nwkskey, appskey);
    #else
    // If not running an AVR with PROGMEM, just use the arrays directly
    LMIC_setSession (0x1, DEVADDR, NWKSKEY, APPSKEY);
    #endif
    // Set up the channels used by the Things Network, which corresponds
    // to the defaults of most gateways. Without this, only three base
    // channels from the LoRaWAN specification are used, which certainly
    // works, so it is good for debugging, but can overload those
    // frequencies, so be sure to configure the full frequency range of
    // your network here (unless your network autoconfigures them).
    // Setting up channels should happen after LMIC_setSession, as that
    // configures the minimal channel set.
    LMIC_setupChannel(0, 868100000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);  // g-band
    LMIC_setupChannel(1, 868300000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);  // g-band
    LMIC_setupChannel(2, 868500000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);  // g-band
    LMIC_setupChannel(3, 867100000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);  // g-band
    LMIC_setupChannel(4, 867300000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);  // g-band
    LMIC_setupChannel(5, 867500000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);  // g-band
    LMIC_setupChannel(6, 867700000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);  // g-band
    LMIC_setupChannel(7, 867900000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);  // g-band
    LMIC_setupChannel(8, 868800000, DR_RANGE_MAP(DR_FSK, DR_FSK), BAND_MILLI);  // g2-band
    // TTN defines an additional channel at 869.525MHz using SF9 for class B
    // devices' ping slots. LMIC does not have an easy way to define set this
    // frequency and support for class B is spotty and untested, so this
    // frequency is not configured here.
    // Disable link check validation
    LMIC_setLinkCheckMode(0);
    // Set data rate and transmit power (note: txpow seems to be ignored by the library)
    LMIC_setDrTxpow(DR_SF7, 14);
    // Start job
    do_send(&sendjob);
void loop() {
    os_runloop_once();
}

The values of the variables are determined in section 5.4:
Device Address: 26011235
Network Session Key: FD 19 C6 3D 0D A9 78 E3 8D C7 A7 5D 9C 3A D7 2D
App Session Key: 93 4C 4E 59 1C A9 6C 20 EB C8 F4 C8 55 99 5A 40

6.5 Testen Hello World sketch

Be sure the gateway is active:

Compile and upload the sketch:

Open an terminal after the upload:
Result on the console (115.200 baud):

![Console Output]

Result on the gateway console:

![Gateway Configuration]

---

Versie 1.0  
Date: 18-07-2018
Go to the TTN, click on Console, Applications and then click on the application.

Click on Data and open the message:

Result:
In the Payload the transmitted data is in hexadecimal format. Translated with the use of an ASCII table:

```
48 65 6C 6C 6F 2C 20 77 6F 72 6C 64 21
Hello, world!
```

Note:
A message is sent every minute. However, since we are dealing with a single channel gateway and there are 8 channels, there is one message every 8 minutes!!

The translation of the payload can be done directly via a so called decoder. Select Payload Formats and create a decoder.

Various examples can be found using Google. For decoding we take the following decoder:

```javascript
function Decoder(bytes, port) {
  return {
    payload: String.fromCharCode.apply(null, bytes)
  };
}
```

You can test the decoder directly:
The next time data is coming in the result will be:

Message:

```
payload: 48 65 6C 6F 2C 20 77 6F 72 6C 64 21 payload: "Hello, world!"
```
7 Python MQTT Client

7.1 Introduction

In this chapter, a Python MQTT client is used to read the node message sent to the gateway by the node.

7.2 Python3 en pip3

In this chapter we use is python3 and pip3. Start a new PuTTY session and log on to the Raspberry Pi. Make sure that the gateway is not closed (otherwise start a new session).

Python3 is already installed on the Raspberry Pi 3.

Open a new session with PuTTY.

The Python3 version can be checked with the command: python3 -V

```
pi@raspberrypi:~ $ python3 -V
Python 3.5.3
```

And pip3 can be checked with the command: pip3 -V

```
pi@raspberrypi:~ $ pip3 -V
pip 9.0.1 from /usr/lib/python3/dist-packages (python 3.5)
```

If Python3 and pip3 are not installed, you can use the following commands:

- `sudo apt-get install python3`
- `sudo apt-get install python3-pip`

7.3 Install TTN package

Use the following command to install the TTN package:

- `sudo pip3 install ttn`


7.4 Python TTN MQTT Client

The client consists of:
- A number of necessary imports
- Variables for the application id and the access key
- A callback function for handling an incoming uplink message
- Defining and initiating the TTN and MQTT handlers
- An endless loop (the client waits for incoming uplink messages)
- And finally handling the closing of the client (by pressing Ctrl + C)

Create the source with the commands:
- cd ~
- mkdir ttn
- cd ttn
- nano ttnclient.py

7.4.1 Source: ttnclient.py

```python
# Imports
import sys
import time
import ttn
import base64

# Globals
app_id = "xxxxxxxxx"
access_key = ":ttn-account-v2.xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"

# Uplink callback function
def uplink_callback(data, client):
    print("*** Received uplink ***")
    # Show uplink message
    print(str(data))
    print()
    print("App_id    : ", str(data.app_id))
    print("Dev_id    : ", str(data.dev_id))
    print("HW serial : ", str(data.hardware_serial))
    print("Frequency : ", str(data.metadata.frequency))
    print("Gateway ID: ", str(data.metadata.gateways[0].gtw_id))
    print("Data      : ", str(data.payload_fields.payload))
    print()
    print("Raw data  : ", str(data.payload_raw))
    # Decode base64 message
    message = base64.b64decode(data.payload_raw)
    print("Decode B64: ", message.decode(\"utf-8\"))
    print()

# Define TTN handler
print("TTN MQTT client started")
handler = ttn.HandlerClient(app_id, access_key)

# MQTT client
mqtt_client = handler.data()
mqtt_client.set_uplink_callback(uplink_callback)
mqtt_client.connect()
print("Client connected")
```
```python
# Endless loop
try:
    # Wait for uplink message
    while True:
        time.sleep(1)
except KeyboardInterrupt:
    # Ctrl+C pressed
    print()
    print('Interrupted...')

# Graceful exit
mqtt_client.close()
print("Client closed")
sys.exit(0)
```

Sluit af met Ctrt+X en J.

## 7.5 Starten TTN MQTT Client

Start the client with the command:
- `python3 ttnclient.py`

Wait for incoming messages.

Result on the gateway:
Result on the TTN MQTT Client:

```
pi@raspberrypi:~/tn

TTN MQTT client started
Client connected
*** Received uplink ***
MSG(dev_id='appid001002', app_id='appid001001', hardware_serial='00C2C6B70271BEB4', counter=40, metadata=MSG(frequency=868.1, modulation='LORA', airtime=61696000, data_rate='SF7BW125', coding_rate='4/5', time='2018-05-30T16:53:09.017698444Z', gateways=[MSG(altitude=1, gtw_id='eui-b827ebfffe20ca5b', channel=0, snr=9, altitude=53.26072, rf_chain=0, time='', rssi=-46, timestamp=796646898, longitude=6.0324183)], payload_fields=MSG(payload='Hello, world!'), port=1, payload_raw='SGVsbG8sIHNlY3vuZw==')

App_id    :  appid001001
Dev_id    :  appid001002
HW serial :  00C2C6B70271BEB4
Frequency :  868.1
Gateway ID:  eui-b827ebfffe20ca5b
Data      :  Hello, world!
Raw data  :  SGVsbG8sIHNlY3vuZw==
Decode B64:  Hello, world!
```

The output is:

```
*** Received uplink ***
MSG(dev_id='appid001002', app_id='appid001001', hardware_serial='00C2C6B70271BEB4', counter=40, metadata=MSG(frequency=868.1, modulation='LORA', airtime=61696000, data_rate='SF7BW125', coding_rate='4/5', time='2018-05-30T16:53:09.017698444Z', gateways=[MSG(altitude=1, gtw_id='eui-b827ebfffe20ca5b', channel=0, snr=9, latitude=53.26072, rf_chain=0, time='', rssi=-46, timestamp=796646898, longitude=6.0324183)], payload_fields=MSG(payload='Hello, world!'), port=1, payload_raw='SGVsbG8sIHNlY3vuZw==')

App_id    :  appid001001
Dev_id    :  appid001002
HW serial :  00C2C6B70271BEB4
Frequency :  868.1
Gateway ID:  eui-b827ebfffe20ca5b
Data      :  Hello, world!
Raw data  :  SGVsbG8sIHNlY3vuZw==
Decode B64:  Hello, world!
```
8 A practical example

8.1 Introduction

In this example, we will use the node to measure the soil moisture. Via a so-called Pushbullet alert on the mobile phone, the user is warned if the ground is too wet or too dry.

8.2 Hardware

A ground humidity sensor is used to measure the humidity. This is a combination of a sensor (FC-28) and a so-called comparator chip module LM393. This can be used to set a limit value if the digital pin (D0) is used. The module contains a potentiometer with which the limit value can be set. Since we will use the analog output we will not use this module.

There are 4 connections that are connected to the Dragino as follows:

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Dragino</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCC</td>
<td>5V</td>
</tr>
<tr>
<td>GND</td>
<td>GND</td>
</tr>
<tr>
<td>A0</td>
<td>A0</td>
</tr>
<tr>
<td>D0</td>
<td>Not connected</td>
</tr>
</tbody>
</table>
Connected to the Dragino:

FC-28 sensor connection:

LM393 connection:
The sensor generates values between 0 (wet) and 1023 (dry). For the determination of the limit values to be used, you have to do a few experiments with dry and wet soil yourself. In this example I use the limit values:

- 250 (and lower) is too wet and
- 750 (and higher) is too dry.

### 8.3 PushBullet

To use PushBullet you need a Google or Facebook account. Go to: [https://www.pushbullet.com/](https://www.pushbullet.com/)

Log in with a Google or Facebook account.
Select Settings:

Select Account:

Go to the right and scroll down a little and select Create Access Token:

Access Tokens

Using an access token grants full access to your account. Don’t share this lightly. You need the access token in order to use the API.

Create Access Token

Make a copy of the access token.
8.3.1 PushBullet App

To receive PushBullet notifications, we need to install the PushBullet App. There are versions for Windows, Android and IOS.

8.4 Arduino Node

On the Arduino node the soil moisture is measured with the connected sensor. The TTN sensor part is suppressed. After all, we only use one channel. In default mode, 3 channels are used. This means that now a valid message is sent every 3 minutes (instead of the previous 9 minutes).

8.4.1 Arduino sketch

```c
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>
#include "moisturestate.h"

// LoRaWAN NwkSKey, network session key
// This is the default Semtech key, which is used by the prototype TTN
// network initially.
static const PROGMEM u1_t NWKSKEY[16] = { 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX };

// LoRaWAN AppSKey, application session key
// This is the default Semtech key, which is used by the prototype TTN
// network initially.
static const PROGMEM u1_t APPSKKEY[16] = { 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX };

// LoRaWAN end-device address (DevAddr)
// See http://thethingsnetwork.org/wiki/AddressSpace
static const u4_t DEVADDR = 0xXXXXXXXX;
```
// These callbacks are only used in over-the-air activation, so they are
// left empty here (we cannot leave them out completely unless
// DISABLE_JOIN is set in config.h, otherwise the linker will complain).
void os_getArtEui (u1_t* buf) { }
void os_getDevEui (u1_t* buf) { }
void os_getDevKey (u1_t* buf) { }

static uint8_t data_DRY[] = "DRY";
static uint8_t data_ACC[] = "ACC";
static uint8_t data_WET[] = "WET";

static osjob_t sendjob;

// Schedule TX every this many seconds (might become longer due to duty
// cycle limitations).
const unsigned TX_INTERVAL = 60;

// Soil moisture thresolds
const unsigned TRESSHOLD_DRY = 750;
const unsigned TRESSHOLD_WET = 250;

// Soil moisture state
MOISTURESTATE moistureState;

// Pin mapping Dragine hat
const lmic_pinmap lmic_pins = {
    .nss = 10,
    .rxtx = LMIC_UNUSED_PIN,
    .rst = 9,
    .dio = {2, 6, 7},
};

// Pin mapping soil moisture sensor
const int FC28_SENSOR_PIN = A0;

// Event handler
void onEvent (ev_t ev) {
    Serial.print(os_getTime());
    Serial.print("|");
    switch (ev) {
        case EV_SCAN_TIMEOUT:
            Serial.println(F("EV_SCAN_TIMEOUT"));
            break;
        case EV_BEACON_FOUND:
            Serial.println(F("EV_BEACON_FOUND"));
            break;
        case EV_BEACON_MISSED:
            Serial.println(F("EV_BEACON_MISSED"));
            break;
        case EV_BEACON_TRAKED:
            Serial.println(F("EV_BEACON_TRAKED"));
            break;
        case EV_JOINING:
            Serial.println(F("EV_JOINING"));
            break;
        case EV_JOINED:
            Serial.println(F("EV_JOINED"));
            break;
        case EV_RFU1:
            Serial.println(F("EV_RFU1"));
            break;
        case EV_JOIN_FAILED:
            Serial.println(F("EV_JOIN_FAILED"));
            break;
        case EV_REJOIN_FAILED:
            Serial.println(F("EV_REJOIN_FAILED"));
            break;
    }
case EV_TXCOMPLETE:
    Serial.println(F("EV_TXCOMPLETE (includes waiting for RX windows)");
    if (LMIC.dataLen) {
        // data received in rx slot after tx
        Serial.print(F("Data Received: ");
        Serial.write(LMIC.frame + LMIC.dataBeg, LMIC.dataLen);
        Serial.println();
    }
    // Schedule next transmission
    os_setTimedCallback(&sendjob, os_getTime() + sec2osticks(TX_INTERVAL),
                       do_send);
    break;

    case EV_LOST_TSYNC:
        Serial.println(F("EV_LOST_TSYNC");
        break;

    case EV_RESET:
        Serial.println(F("EV_RESET");
        break;

    case EV_RXCOMPLETE:
        // data received in ping slot
        Serial.println(F("EV_RXCOMPLETE");
        break;

    case EV_LINK_DEAD:
        Serial.println(F("EV_LINK_DEAD");
        break;

    case EV_LINK_ALIVE:
        Serial.println(F("EV_LINK_ALIVE");
        break;

    default:
        Serial.println(F("Unknown event");
        break;
}
}

void do_send(osjob_t* j) {
    // Check if there is not a current TX/RX job running
    if (LMIC.opmode & OP_TXRXPEND) {
        Serial.println(F("OP_TXRXPEND, not sending");
    } else {
        readFC28Sensor();
        Serial.print(F("Read sensor: ");
        Serial.println(moistureState);
        // Prepare upstream data transmission at the next possible time.
        if (moistureState == TOO_DRY) {
            LMIC_setTxData2(1, data_DRY, sizeof(data_DRY) - 1, 0);
        } else if (moistureState == TOO_WET) {
            LMIC_setTxData2(1, data_WET, sizeof(data_WET) - 1, 0);
        } else {
            LMIC_setTxData2(1, data_ACC, sizeof(data_ACC) - 1, 0);
        }
        Serial.println(F("Packet queued");
    }
    // Next TX is scheduled after TX_COMPLETE event.
}

// Read FC28 sensor value
void readFC28Sensor() {
    int value = analogRead(FC28_SENSOR_PIN);
    if (value >= TRESHOLD_DRY) {
        moistureState = TOO_DRY;
    } else if (value <= TRESHOLD_WET) {
        moistureState = TOO_WET;
    } else {
        moistureState = ACCEPTABLE;
    }
}
void setup() {
    Serial.begin(115200);
    Serial.println(F("Starting"));
    #ifdef VCC_ENABLE
        // For Pinoccio Scout boards
        pinMode(VCC_ENABLE, OUTPUT);
        digitalWrite(VCC_ENABLE, HIGH);
        delay(1000);
    #endif
    // LMIC init
    os_init();
    // Reset the MAC state. Session and pending data transfers will be discarded.
    LMIC_reset();
    // Set static session parameters. Instead of dynamically establishing a session
    // by joining the network, precomputed session parameters are be provided.
    #ifdef PROGMEM
        // On AVR, these values are stored in flash and only copied to RAM
        // once. Copy them to a temporary buffer here, LMIC_setSession will
        // copy them into a buffer of its own again.
        uint8_t appskey[sizeof(APPSKEY)];
        uint8_t nwkskey[sizeof(NWKSKEY)];
        memcpy_P(appskey, APPSKEY, sizeof(APPSKEY));
        memcpy_P(nwkskey, NWKSKEY, sizeof(NWKSKEY));
        LMIC_setSession(0x1, DEVADDR, nwkskey, appskey);
    #else
        // If not running an AVR with FPROGMEM, just use the arrays directly
        LMIC_setSession(0x1, DEVADDR, NWKSKEY, APPSKEY);
    #endif
    // Set up the channels used by the Things Network, which corresponds
    // to the defaults of most gateways. Without this, only three base
    // channels from the LoRaWAN specification are used, which certainly
    // works, so it is good for debugging, but can overload those
    // frequencies, so be sure to configure the full frequency range of
    // your network here (unless your network autoconfigures them).
    // Setting up channels should happen after LMIC_setSession, as that
    // configures the minimal channel set.
    >/*
        LMIC_setupChannel(0, 868100000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
        LMIC_setupChannel(1, 868300000, DR_RANGE_MAP(DR_SF12, DR_SF7B), BAND_CENTI); // g-band
        LMIC_setupChannel(2, 868500000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
        LMIC_setupChannel(3, 867100000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
        LMIC_setupChannel(4, 867300000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
        LMIC_setupChannel(5, 867500000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
        LMIC_setupChannel(6, 867700000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
        LMIC_setupChannel(7, 867900000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
        LMIC_setupChannel(8, 868800000, DR_RANGE_MAP(DR_FSK, DR_FSK), BAND_MILLI); // g2-band
    */
    // TTN defines an additional channel at 869.525 MHz using SF9 for class B
    // devices' ping slots. LMIC does not have an easy way to define set this
    // frequency and support for class B is spotty and untested, so this
    // frequency is not configured here.
    // Disable link check validation
    LMIC_setLinkCheckMode(0);
    // Set data rate and transmit power (note: txpow seems to be ignored by the library)
    LMIC_setDrTxpow(DR_SF7, 14);
    // Start job
    do_send(&sendjob);
}

void loop() { 
    os_runloop_once();
}
8.4.2 Source: moisturestate.h

```c
enum MOISTURESTATE {
    TOO_DRY,
    ACCEPTABLE,
    TOO_WET
};
```

8.5 RPI TTN MQTT PushBullet Client

The Python3 TTN Client handles node messages by sending a PushBullet message to the mobile app in case the soil is too dry or too wet. The client uses the previously generated PushBullet access token.

The content of the message is also dumped on the screen.

Commands:
- Open a new PuTTy session
- Log in
- cd ~
- mkdir ttn
- cd ttn
- nano ttnclient.py

8.5.1 Source: TTNClient.py

```python
#!/usr/bin/python3
# Imports
import sys
import time
import ttn
import base64
import requests
import json
# Globals
ttn_app_id = "xxxxxxxxxxx"
ttn_access_key = "ttn-account-v2.xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
pushbullet_url = "https://api.pushbullet.com:443/v2/pushes"
pushbullet_token = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
# Send telegram notification
def send_notification(status):
    postData = {
        "type": "note",
        "title": "RPI Python Soil Alert!",
        "body": "Alert: Soil is too " + status + "!"
    }
    resp = requests.post(pushbullet_url, data=json.dumps(postData),
                         headers={'Authorization': 'Bearer ' + pushbullet_token,
                                  'Content-Type': 'application/json'})
    if resp.status_code != 200:
        raise Exception('Error sending telegram notification')
    else:
        print('Send telegram notification')
```
# Uplink callback function

def uplink_callback(data, client):
    print("*** Received uplink ***")

    # Show uplink message
    print(str(data))
    print()
    print("Data : ")
    print("App_id : ", str(data.app_id))
    print("Counter : ", str(data.counter))
    print("Dev_id : ", str(data.dev_id))
    print("Port : ", str(data.port))
    print("HW serial : ", str(data.hardware_serial))
    print("MetaData : ")
    print("Airtime : ", str(data.metadata.airtime))
    print("CodingRate : ", str(data.metadata.coding_rate))
    print("DataRate : ", str(data.metadata.data_rate))
    print("Frequency : ", str(data.metadata.frequency))
    print("Modulation: ", str(data.metadata.modulation))
    print("Time : ", str(data.metadata.time))
    print("Gateways : ")
    print("Altitude : ", str(data.metadata.gateways[0].altitude))
    print("Channel : ", str(data.metadata.gateways[0].channel))
    print("Gateway ID: ", str(data.metadata.gateways[0].gtw_id))
    print("Latitude : ", str(data.metadata.gateways[0].latitude))
    print("Longitude : ", str(data.metadata.gateways[0].longitude))
    print("RF_chain : ", str(data.metadata.gateways[0].rf_chain))
    print("RSSI : ", str(data.metadata.gateways[0].rssi))
    print("Time : ", str(data.metadata.gateways[0].time))
    print("Timestamp : ", str(data.metadata.gateways[0].timestamp))
    print("Payload :")

    soilStatus = str(data.payload_fields.payload)
    print("Data : ", soilStatus)
    print()
    print("Raw data : ", str(data.payload_raw))

    # Send notification if soil is too dry or too wet
    if soilStatus == "DRY":
        send_notification("dry");
    if soilStatus == "WET":
        send_notification("wet");

    # Decode base64 message
    message = base64.b64decode(data.payload_raw)
    print("Decode B64: ", message.decode("utf-8"))
    print()

    # Define TTN handler
    print("TTN MQTT client started")
    handler = ttn.HandlerClient(ttn_app_id, ttn_access_key)

    # MQTT client
    mqtt_client = handler.data()
    mqtt_client.set_uplink_callback(uplink_callback)
    mqtt_client.connect()
    print("Client connected")

    # Endless loop
    try:
        # Wait for uplink message
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        # Ctrl+C pressed
        print("Interrupted...")

    # Graceful exit
    mqtt_client.close()
    print("Client closed")
    sys.exit(0)
8.5.2 Testen

Start the client with the commands:

- cd ~
- cd ttn
- python3 ttnclient.py

Result:

```bash
pi@raspberrypi:~/ttn $ python3 ttnclient.py
TTN MQTT client started
Client connected
```
As soon as a node sends a message this will be shown. A PushBullet message is also sent if the soil is too dry or too wet.
A PushBullet message is shown on the mobile:

8.6 Windows 10 Java client

A client can run on any platform. In this example, we are going to create a Java client that runs under Windows 10.

8.6.1 JDK8 en Maven

8.6.1.1 Java SDK (JDK8)

If necessary, download the 32 or 64 bit version of the Java Software Development Kit. Go to: http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
Install the JDK.

Create/change the following environment variables:

- JAVA_HOME: refer to the JDK installation directory
- PATH: add% JAVA_HOME% \ bin directory

Check the Java Runtime version with the command: java -version

```
C:\>java -version
java version "1.8.0_152"
Java(TM) SE Runtime Environment (build 1.8.0_152-b16)
Java HotSpot(TM) 64-Bit Server VM (build 25.152-b16, mixed mode)
C:\>
```

Check the Java compiler version with the command: javac -version

```
C:\>javac -version
javac 1.8.0_152
C:\>
```
8.6.1.2 Maven

If necessary, download the latest version of Maven. Go to: https://maven.apache.org/download.cgi

Files

Maven is distributed in several formats for your convenience. Simply pick a rear yourself.

In order to guard against corrupted downloads/installations, it is highly recommen

<table>
<thead>
<tr>
<th>Link</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary tar.gz archive</td>
<td>apache-maven-3.5.3-bin.tar.gz</td>
</tr>
<tr>
<td>Binary zip archive</td>
<td>apache-maven-3.5.3-bin.zip</td>
</tr>
<tr>
<td>Source tar.gz archive</td>
<td>apache-maven-3.5.3-src.tar.gz</td>
</tr>
<tr>
<td>Source zip archive</td>
<td>apache-maven-3.5.3-src.zip</td>
</tr>
</tbody>
</table>

Unzip to a location of your choice, for example:
C:\Applicaties

Create/change the following environment variables:
- MAVEN_HOME: refer to the installation directory of Maven
- PATH: add %MAVEN_HOME%\bin
Check the installed Maven version: `mvn -version`

```
C:\>mvn -version
C:\
Apache Maven 3.5.3 (3383c37e1f9e9b3bc3df050c29c8aff9f295297; 2018-02-24T20:49:05+01:00)
Maven home: C:\Applicaties\apache-maven-3.5.3\bin\..
Java version: 1.8.0_152, vendor: Oracle Corporation
Java home: C:\Program Files\Java\jdk1.8.0_152\jre
Default locale: nl NL, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

8.6.2 Directory structure

Create the following structure:

```
- TTNClient
  - src
    - nl
      - ttn
```

Commands example:
- C:
- `mkdir projecten`
- `cd projecten`
- `mkdir TTNClient`
- `cd TTNClient`
- `md src`
- `cd src`
- `md nl`
- `cd nl`
- `md ttn`
- `cd ttn`

Create a `pom.xml` file in the map TTNClient.

Create the files:
- `TTNClient.java` and `PushBullet.java`

in the directory `TTNClient\src\nl\ttn`

Use a tekst editor like Notepad or Notepad++. 
8.6.4 Dependencies: pom.xml

Content of the pom.xml file:

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>
  <groupId>TTNClient</groupId>
  <artifactId>TTNClient</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <build>
    <sourceDirectory>src</sourceDirectory>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.7.0</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
    </plugins>
  </build>

  <dependencies>
    <dependency>
      <groupId>org.json</groupId>
      <artifactId>json</artifactId>
      <version>20180130</version>
    </dependency>
    <dependency>
      <groupId>org.thethingsnetwork</groupId>
      <artifactId>data-mqtt</artifactId>
      <version>2.1.2</version>
    </dependency>
  </dependencies>

</project>
```
Class PushBullet source:

```java
package nl.ttn;

import java.io.IOException;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.ProtocolException;
import java.net.URL;

public class PushBullet {
    // Constants
    final static String accessToken = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
    final static String host = "https://api.pushbullet.com:443/v2/pushes";

    public static void send(String status) {
        // Locals
        URL url = null;
        HttpURLConnection con = null;
        int responseCode;
        String postData = "{"\"body\":\"Soil is too " + status + ",\n                           \"title\":\"Java Soil Alert!\",\n                           \"type\":\"note\"}";

        // Send message
        try {
            url = new URL(host);
            con = (HttpURLConnection) url.openConnection(
                true); // Establish connection
            con.setDoOutput(true);
            con.setRequestProperty("Authorization", "Bearer " + accessToken);
            con.setRequestProperty("Content-Type", "application/json");
            con.setRequestProperty("Content-Length", 
                Integer.toString(postData.length()));
            con.getOutputStream().write(postData.getBytes("UTF8"));

            // Check response
            responseCode = con.getResponseCode();
            System.out.println("nSending 'POST' request to URL : " + url);
            System.out.println("Response Code : " + responseCode);
        } catch (ProtocolException e) {
            e.printStackTrace();
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

8.6.5 PushBullet.java
8.6.6 TTNClient.java

Class TTNClient source:

```java
package nl.ttn;

import java.net.URISyntaxException;
import java.util.List;
import org.thethingsnetwork.data.common.Connection;
import org.thethingsnetwork.data.common.Metadata;
import org.thethingsnetwork.data.common.Metadata.Gateway;
import org.thethingsnetwork.data.common.messages.DataMessage;
import org.thethingsnetwork.data.common.messages.UplinkMessage;
import org.thethingsnetwork.data.mqtt.Client;

public class TTNClient {
    private static void showData(DataMessage data) {
        // Data
        System.out.println();
        System.out.println("Data ");
        System.out.println("App_id : ");
        System.out.println("Counter : ");
        System.out.println("Dev_id : ");
        System.out.println("HW serial : ");
        System.out.println("Port : ");

        // Metadata
        Metadata md = ((UplinkMessage) data).getMetadata();
        System.out.println("Metadata ");
        System.out.println("BitRate : ");
        System.out.println("CodingRate : ");
        System.out.println("DataRate : ");
        System.out.println("Frequecy : ");
        System.out.println("Modulation : ");
        System.out.println("Time : ");

        // Gateway data
        List<Gateway> gw = ((UplinkMessage) data).getMetadata().getGateways();
        System.out.println("Gateways ");
        System.out.println("Altitude : ");
        System.out.println("Channel : ");
        System.out.println("GatewayId : ");
        System.out.println("Latitude : ");
        System.out.println("Longitude : ");
        System.out.println("RSSI : ");
        System.out.println("SNR : ");
        System.out.println("Time : ");
        System.out.println("Timestamp : ");

        // Payload
        System.out.println("Payload ");
        byte[] payload = ((UplinkMessage) data).getPayloadFields().get("payload");
        for(int i=0; i<payload.length; i++) {
            char ch = (char) payload[i];
            System.out.print(ch);
        }
        System.out.println();
    }
}
```

public static void alertTest(String status) {
    if(status.equals("DRY")) {
        System.out.println("+----------------------+");
        System.out.println("| Status: Soil to dry! | ");
        PushBullet.send("dry");
    } else if(status.equals("WET")) {
        System.out.println("+----------------------+");
        System.out.println("| Status: Soil to wet! | ");
        System.out.println("+----------------------+");
        PushBullet.send("wet");
    } else {
        System.out.println("+----------------------+");
        System.out.println("| Status: OK           | ");
        System.out.println("+----------------------+");
    }
}

public static void main(String[] args) { 
    String region = "eu";
    String appId = "appid001001";
    String accessKey = "ttnc-account-v2.kYhYtD8v-1D13MEvSB-o4rNTgUceQprmyTK9Fnbpz9g";
    try {
        Client client = new Client(region, appId, accessKey);
        client.onError((Throwable _error) -> System.err.println("Error: " + _error.getMessage()));
        client.onConnected((Connection _client) -> System.out.println("Connected"));
        client.onMessage((String devId, DataMessage data) -> {
            System.out.println("-----------------------------------------------------------------");
            showData(data);
        });
        client.start();
    } catch (URISyntaxException e1) {
        System.err.println(e1.getMessage());
        e1.printStackTrace();
    } catch (Exception e2) {
        System.err.println(e2.getMessage());
        e2.printStackTrace();
    }
}
8.6.7 Compile and test

Open a command prompt and compile and run with the command:

```
mvn clean compile exec:java -Dexec.mainClass="nl.ttn.TTNClient"
```

As soon as a node message is sent, this will become visible on the console:
A PushBullet message will appear on the mobile if the soil is too dry or too wet.