



## Single Channel LoRa IoT Kit v2 User Manual

---

Document Version: 1.0.1

Version	Description	Date
1.0.0	Release	2019-Jan-28
1.0.1	Modify limitation words	2019-May-10
1.0.2	Add description of MQTT publish format	2019-Jun-15
1.0.3	Add ThingSpeak downlink example, and Video Link	2019-Jun-19
1.0.4	Correct broken links.	2019-Jun-25
1.0.5	Add trouble shooting for Arduino 1.8.10, add CN470 support, Change Cayenne to Mydevices	2019-Oct-25



## Index:

1	Overview .....	5
1.1	What is Dragino Single Channel LoRa IoT Kit v2?.....	5
1.2	What can you learn from the kit?.....	5
1.3	What parts Dragino LoRa IoT v2 includes?.....	6
2	Preparing.....	7
2.1	Software for End Node .....	7
2.1.1	Install Arduino IDE and CH340 driver.....	7
2.1.2	Install LoRa Library for Arduino.....	8
2.2	Prepare for LG01-N Gateway.....	9
2.2.1	Configure LG01-N for internet connection. ....	9
2.2.2	Download putty tool to access LG01-N via SSH.....	12
3	Example 1: Test a LoRaWAN network .....	13
3.1	Typology and Data Flow .....	14
3.2	Create a gateway in TTN Server.....	15
3.3	Configure LG01-N Gateway .....	17
3.3.1	Configure to connect to LoRaWAN server .....	17
3.3.2	Configure LG01-N's LoRa Radio frequency .....	18
3.4	Create LoRa Shield End Node .....	19
3.4.1	Hardware Connection .....	19
3.4.2	Set up OTAA device in TTN and upload sketch to UNO .....	19
3.4.3	Configure to connect to Mydevices Application Server.....	23
3.4.4	Use downlink message to control relay .....	27
3.4.5	Test with Interrupt .....	29
3.5	Create LoRa/GPS Shield End Node .....	30
3.5.1	Hardware connection.....	30
3.5.2	Set up ABP device in TTN and upload software to UNO.....	30
3.6	Conclusion and limitation.....	34
3.6.1	Overview for the example.....	34
3.6.2	Limitations.....	36
4	Example 2: Test with a MQTT IoT Server .....	38
4.1	Typology and Data Flow .....	38
4.2	Set up sensor channels in ThingSpeak .....	39
4.3	Simulate MQTT uplink via PC's MQTT tool.....	41
4.4	Try MQTT Publish with LG01-N Linux command .....	42
4.5	Configure LG01-N Gateway .....	44
4.5.1	Publish Logic.....	44
4.5.2	Configure LG01-N's Radio frequency .....	45
4.6	Create LoRa Shield End Node .....	47
4.6.1	Hardware Connection .....	47
4.6.2	Test with uplink .....	48
4.6.3	Test with interrupt by flame detect .....	49

---

4.6.4	Test with downlink .....	51
4.7	Conclusion and limitation.....	53
4.7.1	Overview for the example.....	53
5	Order Info.....	54
6	FAQ & Trouble Shooting.....	55
6.1	I can't upload sketch to LoRa Shield in MAC OS, shows " dev/cu.usbmodem1421 is not available " .....	55
6.2	My IoT Kit has the model LG01-P instead of LG01-N, Can I still use this manual. ....	55
6.3	Duplicate library issue while upload in Arduino IDE 1.8.10. ....	56
6.4	How can I set to use CN470 band?.....	57
7	Technical Support.....	58
8	Reference .....	59

## 1 Overview

### 1.1 What is Dragino Single Channel LoRa IoT Kit v2?

**Dragino Single Channel LoRa IoT Kit v2** is designed to facilitate beginners and developers to quickly learn LoRa and IoT technology. It helps users to turn the idea into a practical application and make the Internet of Things a reality. It is easy to program, create and connect your things everywhere. A number of telecom operators are currently rolling out networks, but because LoRa operates in the open spectrum you can also set up your own network.

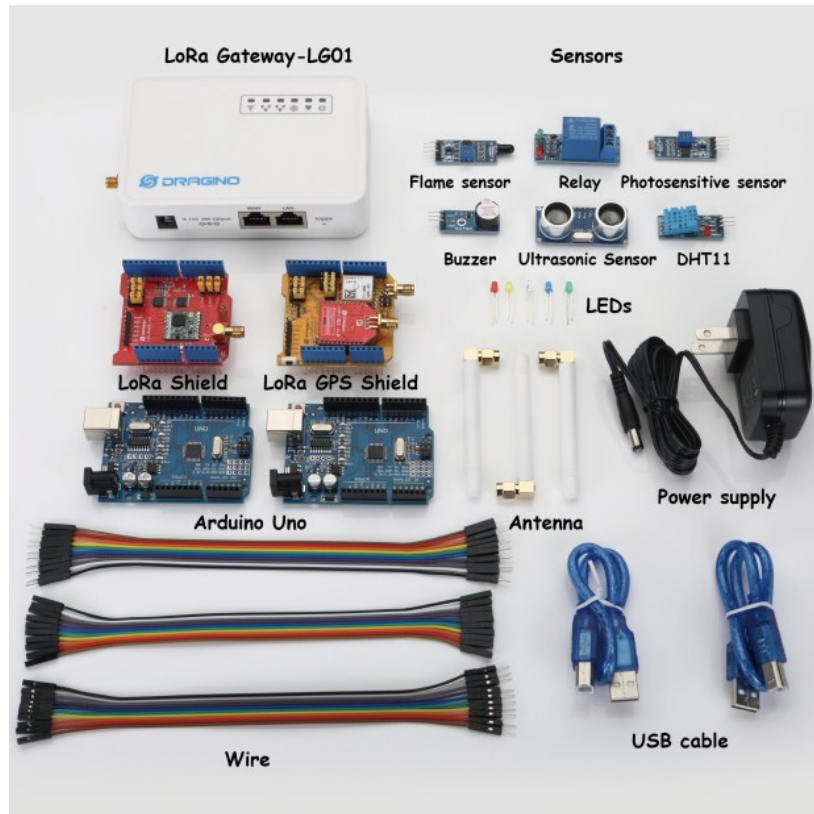
The LoRa IoT kit v2 [shows how to build a LoRa network](#), and [how to use the network to send data from a LoRa sensor node to the cloud server](#). Depends on the actually use environment, the LoRa gateway will connect your other LoRa nodes up to 500 ~ 5,000 meters.

### 1.2 What can you learn from the kit?

#### The goals through this LoRa IoT kit v2:

- ✓ Understand the structure of an Internet of Things network, and how does an IoT network works
- ✓ Learn coding method for Arduino micro controller
- ✓ Learn some common sensors.
- ✓ Learn some basic commands for Linux and
- ✓ Learn about LoRa and how to set up a LoRa network.
- ✓ Learn different way to connect LoRa network to IoT Server and compare their advantages / disadvantages.

### 1.3 What parts Dragino LoRa IoT v2 includes?



#### Single Channel LoRa IoT Kit Packing List.

- ✓ 1 x [LG01-N](#) single channel LoRa Gateway
- ✓ 1 x LoRa end node ([LoRa Shield](#) + Arduino UNO)
- ✓ 1 x LoRa end node ([LoRa/GPS Shield](#) + Arduino UNO)
- ✓ 1 x flame Sensor
- ✓ 1 x relay
- ✓ 1 x photosensitive sensor
- ✓ 1 x buzzer
- ✓ 1 x ultrasonic sensor
- ✓ 1 x DHT11 temperature and humidity sensor
- ✓ 20 x dupont cable (male to male)
- ✓ 20 x dupont cable (female to female)
- ✓ 20 x dupont cable (male to female)

## 2 Preparing

In the kit, there are two LoRa End Node, they are LoRa Shield + UNO and LoRa/GPS Shield + UNO. Both of them use Arduino UNO as MCU to control the LoRa transceiver.

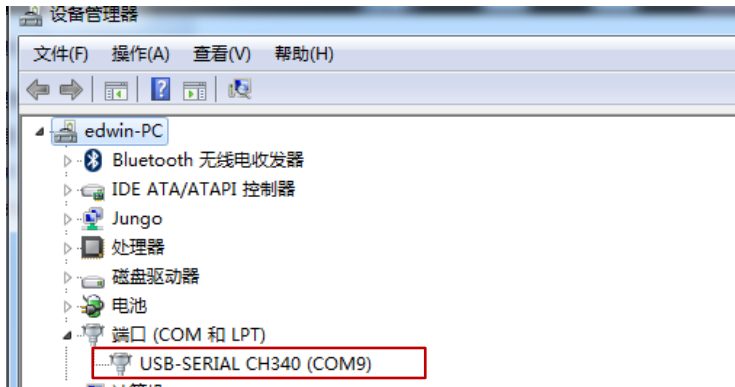
We need to program the Arduino UNO during our testing to support the required functions for end nodes. To finish this, we need to install some software and library first.

### 2.1 Software for End Node

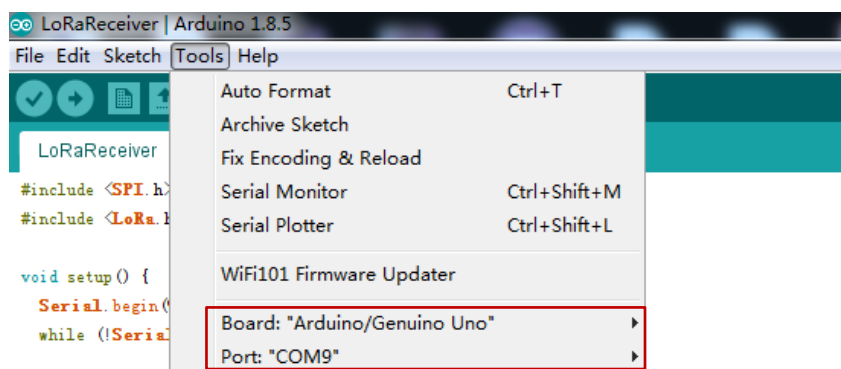
#### 2.1.1 Install Arduino IDE and CH340 driver

First download and install [Arduino IDE](#). This is the tool to program the Arduino UNO.

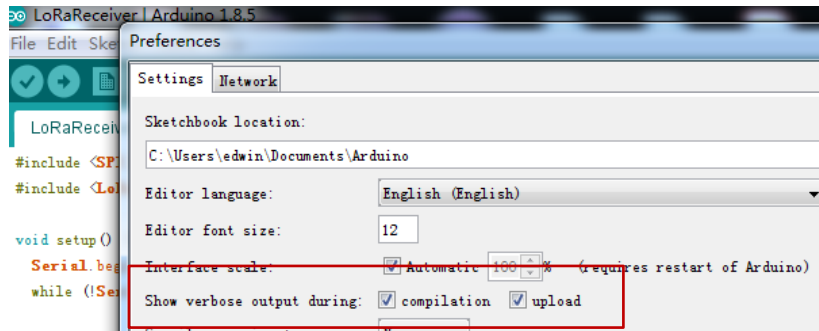
The Arduino UNO in the kit is clone version and is equipped with CH340 USB to UART chip. We need to install CH340 driver in the PC to let the Arduino IDE program it via USB. If we successful install the driver, a com port will show in the system device manager:



After install the driver, start Arduino and we will be able to use the board Arduino UNO and corresponding COM port to program UNO now.



We can enable compilation and upload in Arduino → File → Preference. This will help us in debug.



### 2.1.2 Install LoRa Library for Arduino

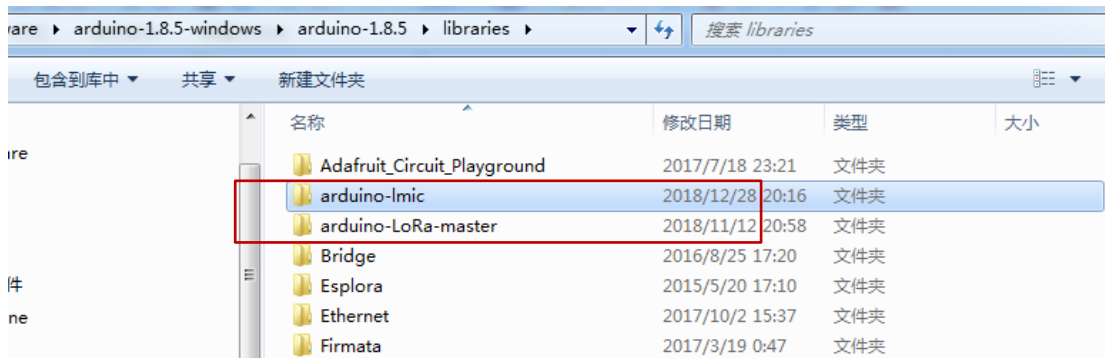
In our examples, we will use two different LoRa libraries for End Node to build different type of LoRa network. They are:

- [Arduino-LMIC](#) : LoRaWAN library to configure the End node as a standard LoRaWAN end node.
- [LoRa-raw](#): This is a simple library for LoRa transmit & receive, all data transfer without ID control, encryption. If user wants to develop a LoRa network with private LoRa protocol, he can modify base on this Library.

We also need to install some libraries to connect to different sensors:

- [DHTlib](#): This is the library to use DHT11 temperature & humidity sensor.
- [TinyGPS](#): Library for LoRa GPS Shield to get the GPS data.

Download all above libraries and put them in the [Arduino → Libraries](#) directory





## 2.2 Prepare for LG01-N Gateway

In LoRa IoT Kit v2, we use LG01-N as LoRa Gateway. Unlike LG01-P in v1 kit, the LG01-N has its own LoRa utility and not need to program it via Arduino. Since we need to connect to Internet IoT Server, we need to configure the LG01-N to have internet access.

### 2.2.1 Configure LG01-N for internet connection.

Below steps show how to set up LG01-N to use WiFi for internet access.

#### Step1:

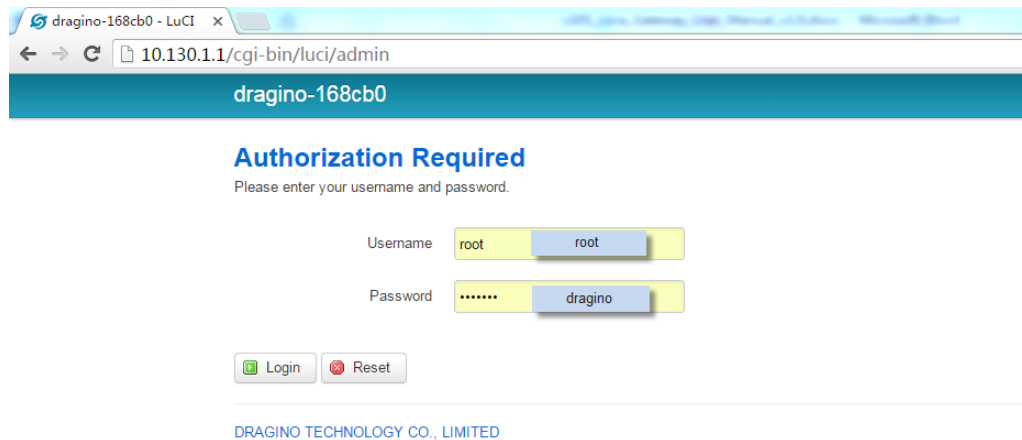
Connect PC to LG01-N's LAN port via RJ45 cable and set up PC Ethernet port to DHCP. PC will then get IP from LG01-N. The ip range is 10.130.1.xx  
Use browser to access the LG01-N via IP 10.130.1.1. (Recommend use Chrome here)

#### Step2:

Open a browser in the laptop and type <http://10.130.1.1/cgi-bin/luci/admin>  
User will see the login interface of LG01-N.  
The account for Web Login is:

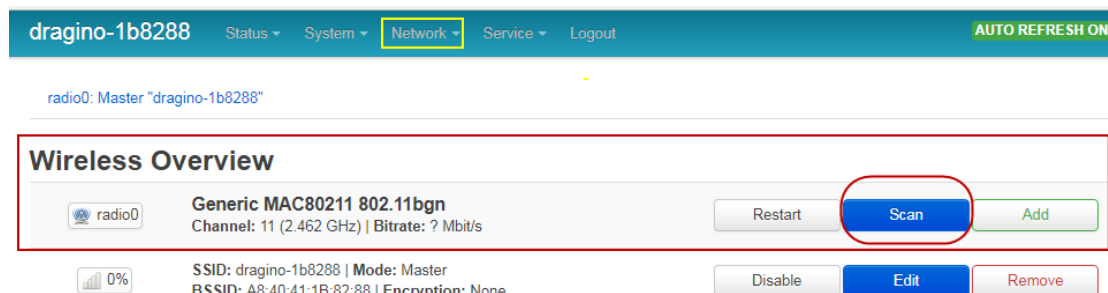
**User Name:** root

**Password:** dragino



#### Step3:

In network -> Wireless, select radio0 interface and scan.



**Step4:**

Select the wireless AP and join the wifi network:

**Step5:**

In network->>wireless page, disable WiFi AP network. Notice: After doing that, you will lose connection if your computer connects to the LG01-N via its WiFi network.

(Note: make sure click the Save & Apply after configure)

After successful associate, the WiFi network interface can be seen in the same page and see LG01-N get the ip from the uplink router.

The screenshot shows the web interface for a Dragino device (dragino-1b8288). At the top, there are navigation tabs for WAN, WWAN, and LAN. Below this is the 'Interfaces' section, which lists three network interfaces:

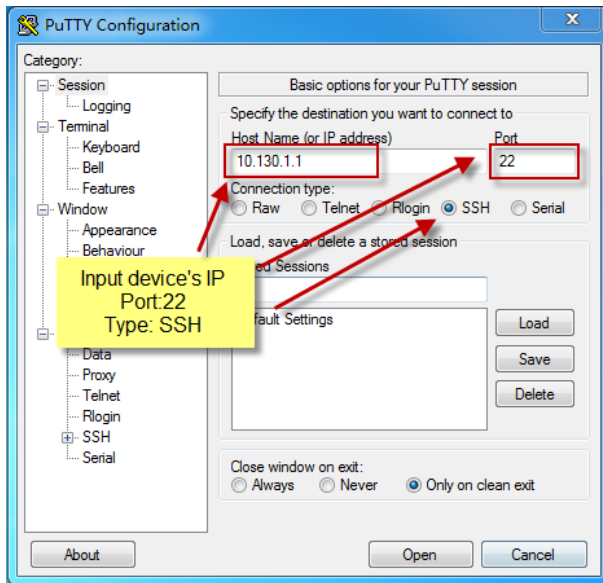
- LAN** (br-lan): Protocol: Static address, Uptime: 2h 0m 4s, MAC: A8:40:41:1B:82:8B, RX: 1.40 MB (13346 Pkts.), TX: 2.79 MB (10321 Pkts.), IPv4: 10.130.1.1/24.
- WAN** (eth1): Protocol: DHCP client, MAC: A8:40:41:1B:82:8A, RX: 4.30 MB (51840 Pkts.), TX: 55.77 KB (429 Pkts.).
- WWAN** (Client "dragino-office"): Protocol: DHCP client, Uptime: 0h 6m 6s, MAC: A8:40:41:1B:82:88, RX: 549.38 KB (5659 Pkts.), TX: 14.90 KB (94 Pkts.), IPv4: 10.130.2.169/24.

The WWAN interface is highlighted with a red border. At the bottom of the interface list, there is a button labeled 'Add new interface...'. At the very bottom of the page, there are three buttons: 'Save & Apply', 'Save', and 'Reset'.

### 2.2.2 Download putty tool to access LG01-N via SSH

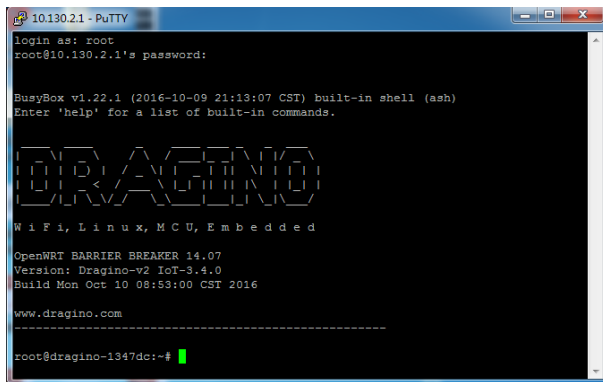
It will be helpful to see the LG01-N inside Linux system to understand the data flow and debug.

User can access to the Linux console via SSH protocol. Make sure your PC and the LG01-N is in the same network, then use a SSH tool (such as [putty](#)) to access it. Below are screenshots:



IP address: IP address of LG01-N  
Port: 22  
User Name: root  
Password: dragino (default)

After log in, you will be in the Linux console and can input commands here.



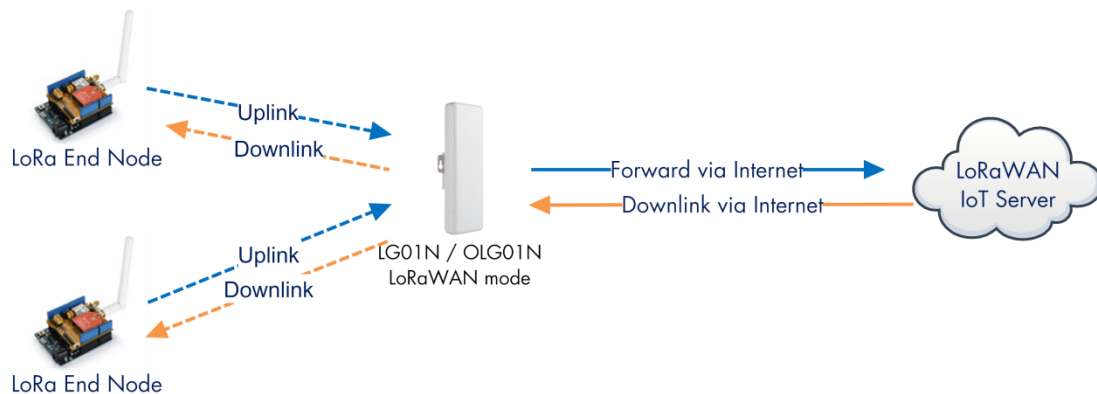
So we have prepare what we need and let's go for the examples!

### 3 Example 1: Test a LoRaWAN network

This example describes how to use LG01-N, LoRa Shield & LoRa GPS Shield to set up a LoRaWAN network and connect it to [TTN LoRaWAN Server](#). It also shows how to use external application server to monitor / manage the LoRa Nodes.

#### LoRaWAN mode:

Use LG01N / OLG01N as a LoRaWAN gateway\* to forward packet to LoRaWAN IoT Server



#### Operate Principle:

- > LG01N/OLG01N running packet forward and will forward the uplink LoRa packet from end node to LoRaWAN server.
- > It will also forward downlink LoRa packet from LoRaWAN server to end node.
- > The end node can use OTAA or ABP mode in the LoRaWAN protocol.

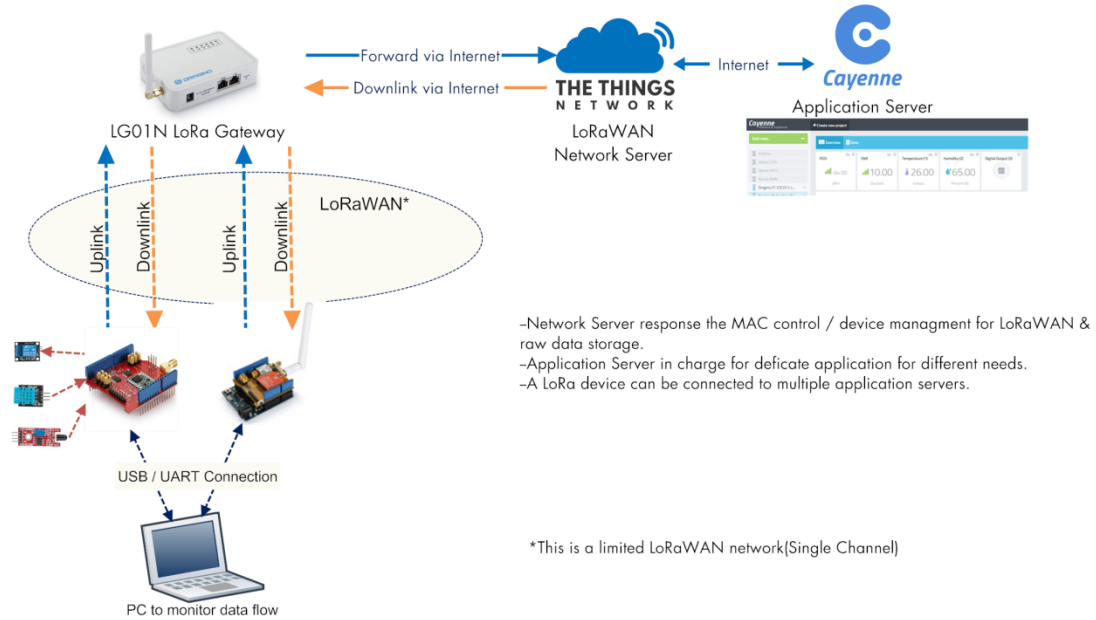
#### Limitation:

- > The LG01 only support one LoRaWAN frequency for uplink. So the end node should be set to fix frequency.
- > If end node use multiply frequencies to transfer, The LG01 will only be able to receive the same frequency set in LG01N.

### 3.1 Typology and Data Flow

The network topology and dataflow for the example is as below:

#### Topology for Thethingsnetwork Connection:

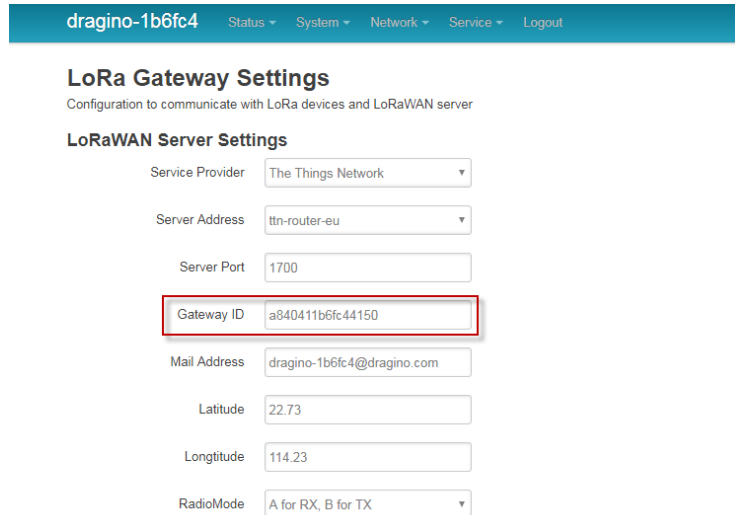


In next section we will start to configure for this example.

### 3.2 Create a gateway in TTN Server

#### Step 1: Get a Unique gateway ID.

Every LG01-N has a unique gateway id. The id can be found at LoRaWAN page:



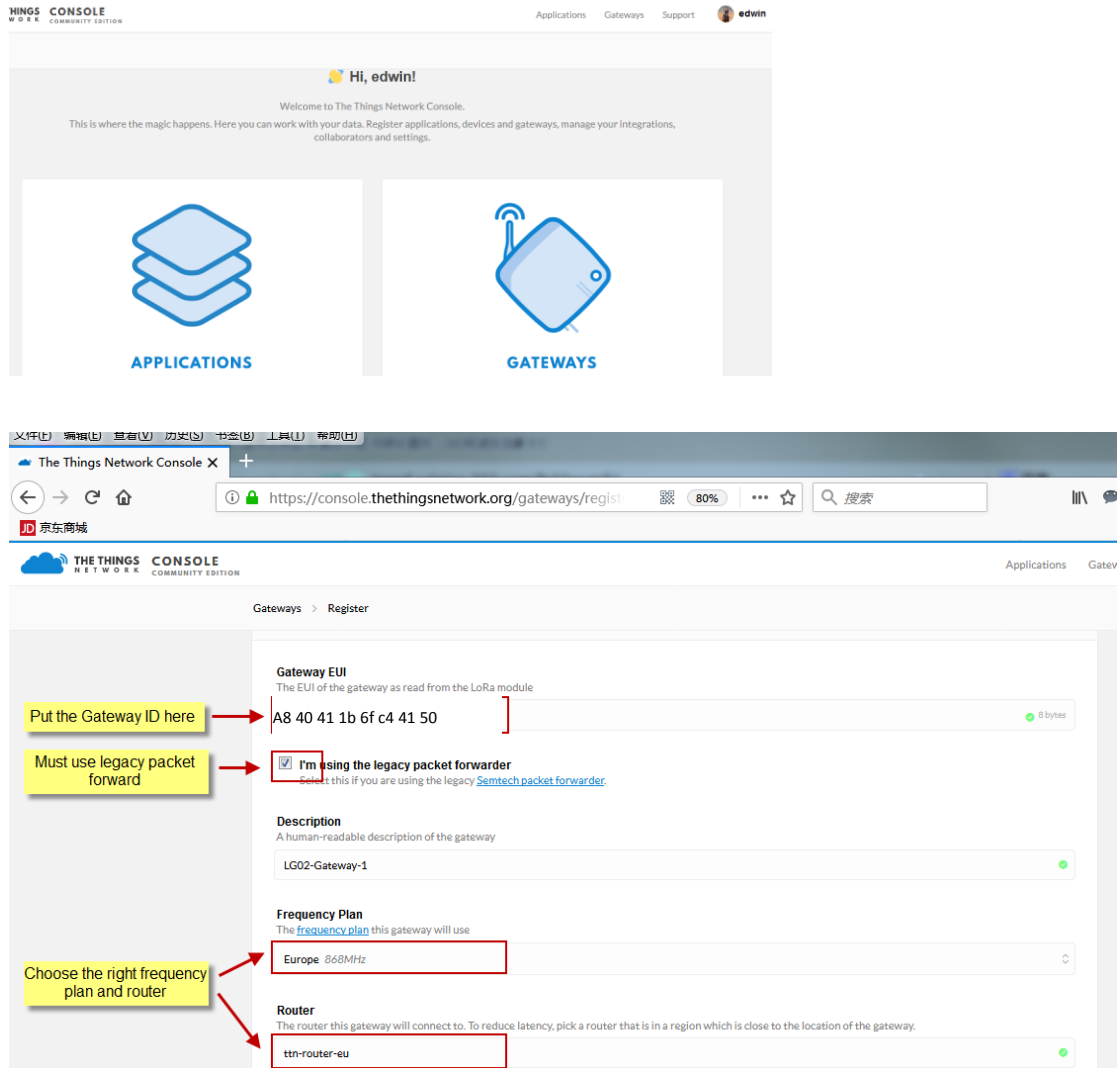
The screenshot shows the 'LoRa Gateway Settings' page in the TTN server. The page title is 'dragino-1b6fc4' and it includes navigation links for Status, System, Network, Service, and Logout. The main heading is 'LoRa Gateway Settings' with a subtitle 'Configuration to communicate with LoRa devices and LoRaWAN server'. Under 'LoRaWAN Server Settings', there are several input fields: Service Provider (The Things Network), Server Address (ttn-router-eu), Server Port (1700), Gateway ID (a840411b6fc44150, highlighted with a red box), Mail Address (dragino-1b6fc4@dragino.com), Latitude (22.73), Longitude (114.23), and RadioMode (A for RX, B for TX).

The gateway id is: **a840411b6fc44150**

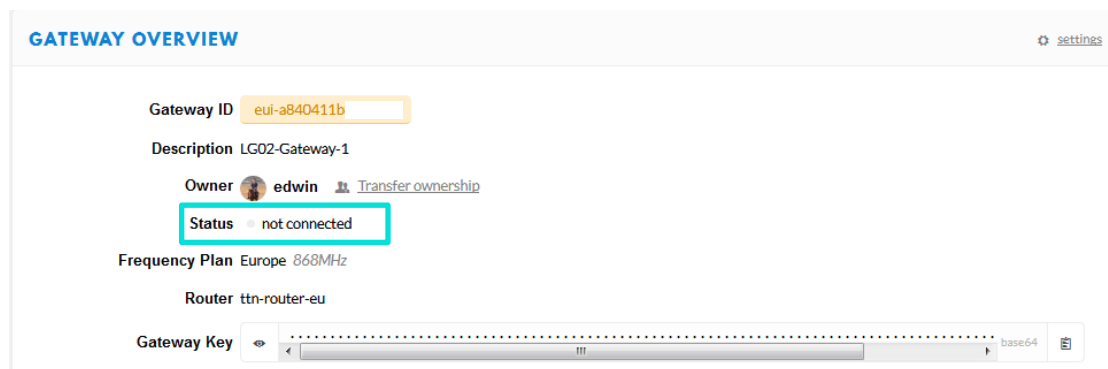
#### Step 2: Sign up a user account in TTN server



### Step 3: Create a Gateway in TTN



After create the gateway, we can see the gateway info, as below, the **Status** shows “not connected” because the LG01-N doesn’t configure to send update status yet.



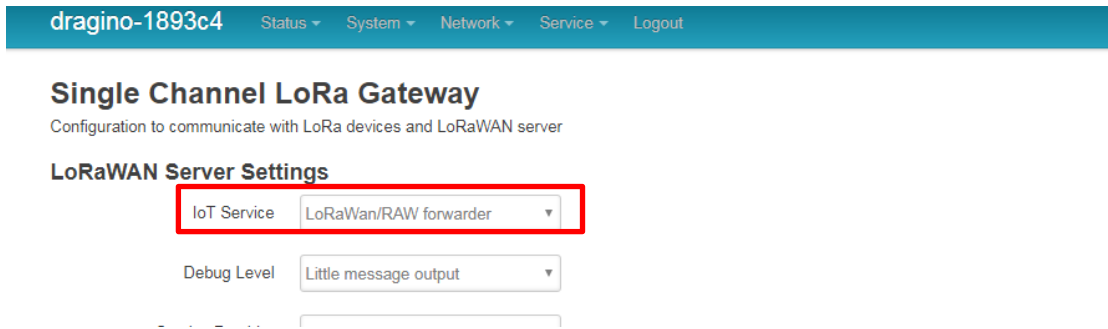


### 3.3 Configure LG01-N Gateway

#### 3.3.1 Configure to connect to LoRaWAN server

We should configure the LG01-N now to let it connect to TTN network. Make sure your LG01-N has Internet Connection first.

#### Step1: Configure LG01-N to act as LoRaWAN forwarder mode



#### Step2: Input server info and gateway id

Choose the correct the server address and gateway ID.



#### Check Result

After above settings, the LG01-N will be able to connect to TTN, as shown in below:



### 3.3.2 Configure LG01-N's LoRa Radio frequency

Now we should configure LG01-N's radio parameter to receive the LoRaWAN packets. We are using 868.1Mhz and other parameters as below:

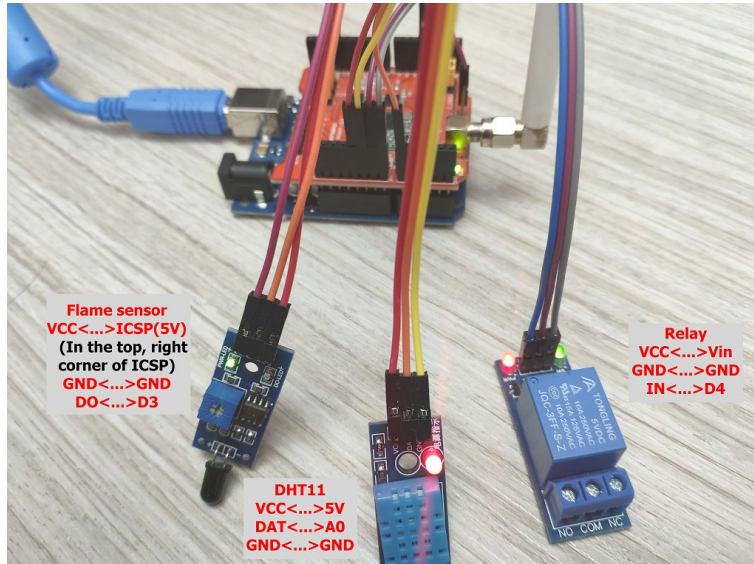
**Radio Settings**  
Radio settings for Channel

Frequency (Unit:Hz)	<input type="text" value="868100000"/>
Spreading Factor	<input type="text" value="SF7"/>
Coding Rate	<input type="text" value="4/5"/>
Signal Bandwidth	<input type="text" value="125 kHz"/>
Preamble Length	<input type="text" value="8"/> <small>Length range: 6 ~ 65536</small>
LoRa Sync Word	<input type="text" value="52"/> <small>Value 52(0x34) for LoRaWAN</small>
Encryption Key	<input type="text" value="Encryption Key"/>

This parameters set is for uplink (receive data for LoRa End Node).According to LoRaWAN spec, the downlink radio parameters frequency is defined by network server (TTN). LG01-N will adjust downlink parameters according to info from TTN.

### 3.4 Create LoRa Shield End Node

#### 3.4.1 Hardware Connection



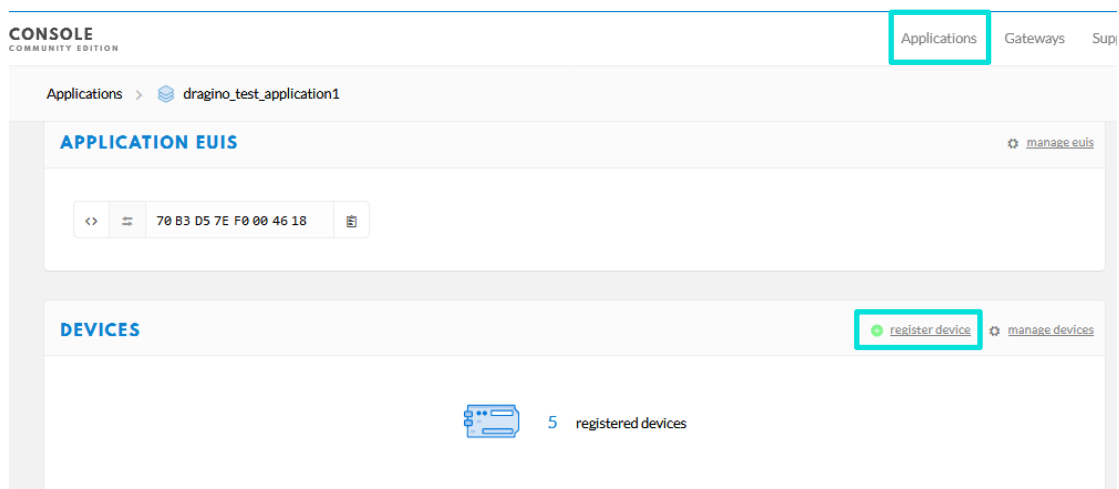
There are three sensors connect to the LoRa Shield + UNO. These sensors are flame sensors, DHT11 (Temperature & Humidity sensor) and Relay. Please use the connection as we show in the photo.

**Note:** There is a trick in above connection, the relay connects to VIN. In this case, The UNO can only be power via USB port. If user need to power via DC power adapter, please use another 5v pin to power the relay.

#### 3.4.2 Set up OTAA device in TTN and upload sketch to UNO

Here we set up the LoRa Shield + UNO as an OTAA device in TTN. We will tell the difference of OTAA and ABP mode later.

**Step 1:** Create an OTAA device in TTN server --> Application page.



dragino\_test\_application1 > Devices > otaa-device-1 > Settings

**Device EUI**  
The serial number of your radio module, similar to a MAC address

AB 40 41 12 34 56 78 90 8 bytes

**Application EUI**

70 B3D57E F0 00 46 18

**Activation Method**

OTAA  ABP

**App Key**  
The key your device will use to set up sessions with the network

C3 95 15 93 AD 55 1A 83 2F 31 25 B6 7A F5 74 1D 16 bytes

For this device, set up to use Cayenne payload, so TTN can parse the sensor data properly.

Overview Devices **Payload Formats** Integrations Data Settings

**PAYLOAD FORMATS**

**Payload Format**  
The payload format sent by your devices

Cayenne LPP

### Step 2: Modify the LMIC library

To use LoRaWAN with LG01-N, we need to modify the LMIC library to support single channel mode.

Find the [Arduino LMIC](#) install path in Arduino library. Before compiling the code, user needs to change the Frequency Band to use with LG01-N. The change is in the file `arduino\libraries\arduino-lmic\src\lmic\config.h`. Changes are as below:

```

#define CFG_eu868 1
// #define CFG_us915 1
// #define CFG_au921 1
// #define CFG_as923 1
// #define CFG_in866 1

#define LG02_LG01 1

//US915: DR_SF10=0, DR_SF9=1, DR_SF8=2, DR_SF7=3, DR_SF6C=4
// DR_SF12CR=8, DR_SF11CR=9, DR_SF10CR=10, DR_SF9CR=11, DR_SF8CR=12, DR_SF7CR
//if defined(CFG_us915) && defined(LG02_LG01) LG02_UPFREQ:End Device Uplink Frequency
// CFG_us915 || CFG_as923 LG02_DNWFREQ:End Device Uplink Frequency
#define LG02_UPFREQ 902320000 LG02_RXSF: End Device Uplink (transmit) SF
#define LG02_DNWFREQ 923300000 LG02_TXSF: End Device Downlink (receive) SF
#define LG02_RXSF 3 // DR_SF7
#define LG02_TXSF 8 // DR_SF12CR
// The TXSF is now set to default value:
// #elif defined(CFG_eu868) && defined(LG02_LG01) US915/AS923 : 923300000 , SF12BW500
// CFG_eu868 EU868: 869525000, SF12BW125
//EU868: DR_SF12=0, DR_SF11=1, DR_SF10=2, DR_SF9=3, DR_SF8=4, DR_SF7=5, DR_SF7B=1, DR_FSK, DR_NONE
#define LG02_UPFREQ 868100000
#define LG02_DNWFREQ 869525000
#define LG02_RXSF 5 // DR_SF7
#define LG02_TXSF 0 // DR_SF12
#endif

```

Choose the Frequency Band, same as in LoRaWAN server

uncomment this for LG01 / LG02

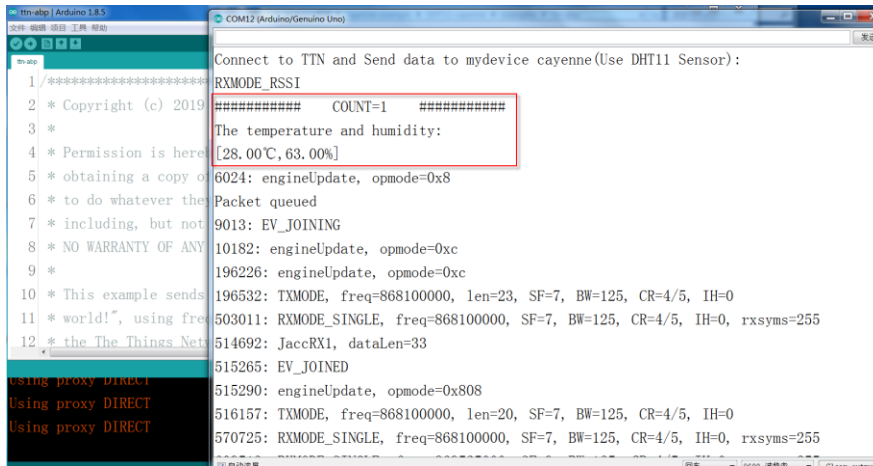
**Step 3:** Input keys in Arduino Sketch and upload to device.

The sketch for this example is [lora\\_shield\\_cayenne\\_and\\_ttn-otaaClient.ino](#). Download and open it, we need to modify the keys to match the keys in TTN. Get Device EUI/Application EUI & APP Key from TTN and put them in the sketch, make sure the Device EUI and Application Key are lsb and the APP key is msb.

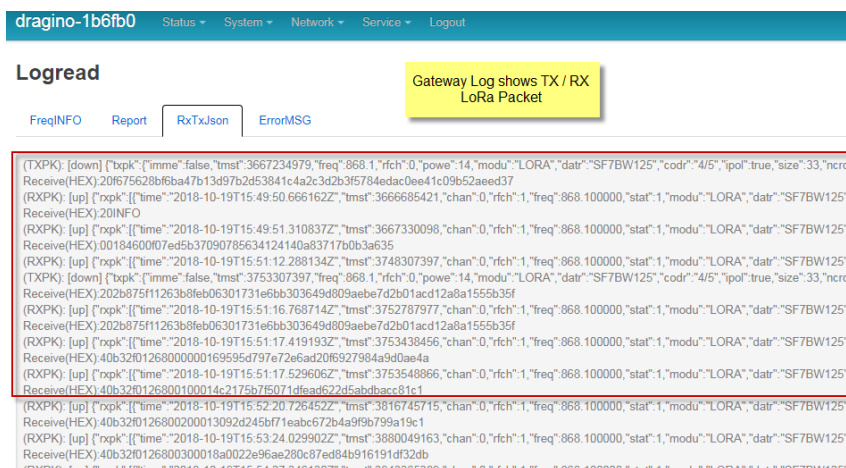
**Upload the code to UNO:**

#### Step 4: Analyze output result

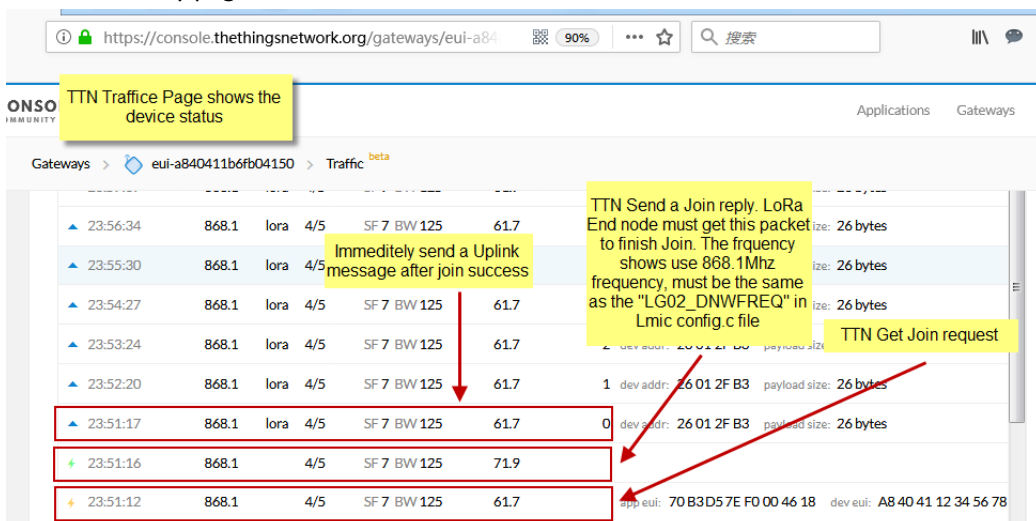
From output of LoRa Node Serial Monitor, we can see it send Joining after start(TX), then get join ACK (RX), then upload the data to TTN (TX).



From gateway logread, we can see the data send from end node (txpk), data get from server(rxpk).



In TTN-Gateway page, we can also see the traffic.



Note: The LG02\_DNWFREQ value in Arduino\_LMIC/src/lmic/config.h should match downlink frequency from TTN.

TTN shows 868.1 here, So LG02\_DNWFREQ should be 868100000

After success Joined, we can see the data in the device page:

**APPLICATION DATA** || pause

Filters: uplink downlink activation ack error

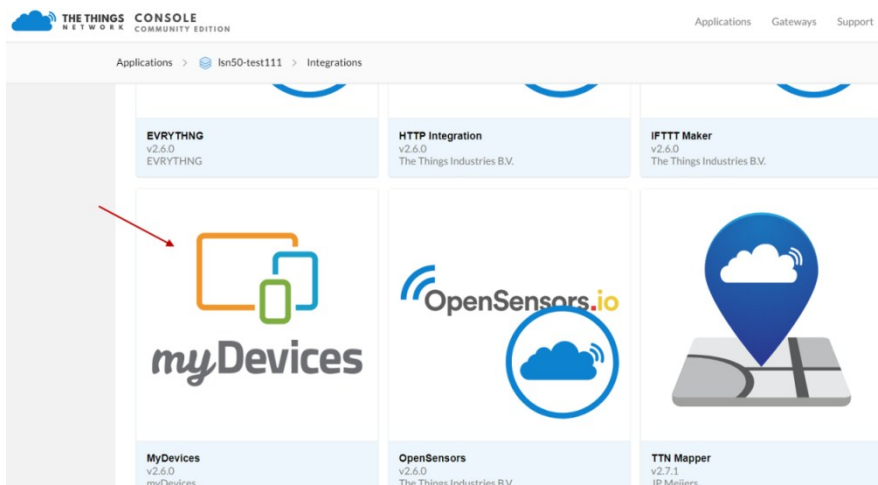
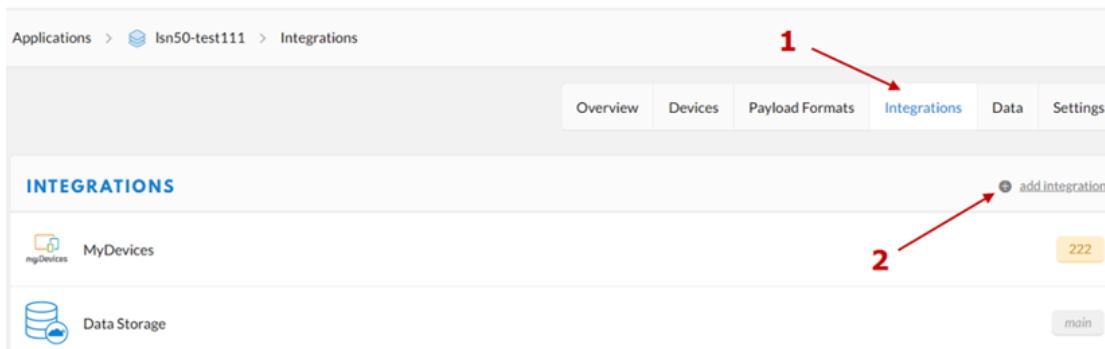
time	counter	port	payload	digital_out_3	relative_humidity_2	temperature_1
▲ 11:57:36	8	1	payload: 01 67 01 0E 02 68 7C 03 01 01	1	62	27
▲ 11:56:33	7	1	payload: 01 67 01 0E 02 68 7A 03 01 01	1	61	27
▲ 11:55:30	6	1	payload: 01 67 01 18 02 68 78 03 01 01	1	60	28
▲ 11:54:28	5	1	payload: 01 67 01 18 02 68 76 03 01 01	1	59	28

91%

### 3.4.3 Configure to connect to Mydevices Application Server


In TTN, we can see the raw data, now we try to connect it to the application server.

Step 1: Add Mydevice in Application page



Applications > Isn50-test111 > Integrations

### ADD INTEGRATION

**MyDevices** (v2.6.0)  
myDevices  
Quickly design, prototype and commercialize IoT solutions with myDevices Cayenne  
[documentation](#)

**Process ID**  
The unique identifier of the new integration process

**Access Key**  
The access key used for downlink

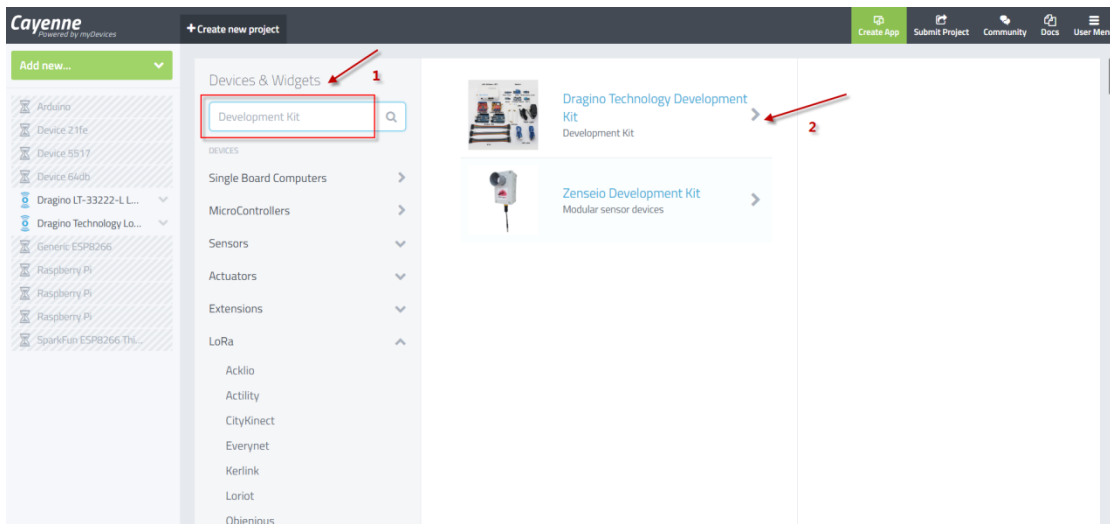
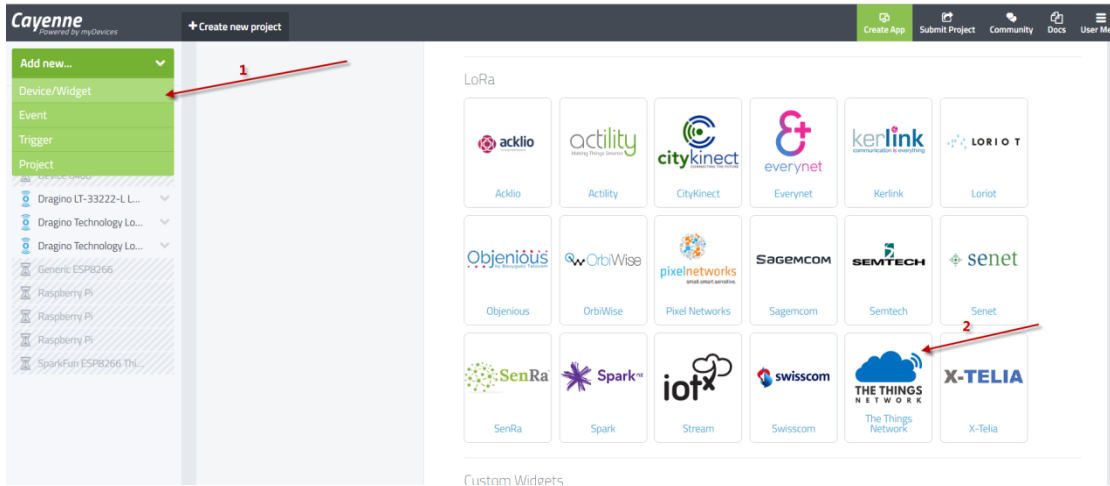
 devices messages

Cancel Add Integration

**3**




Step2: Log in [Mydevices account](#) and add devices.



Add DevEUI of the End node

Enter Settings



**Dragino Technology Development Kit**  
Development Kit

This device uses **Cayenne LPP**

Name  
Dragino Technology Development Kit

DevEUI **your DevEUI**

Activation Mode  
Already Registered

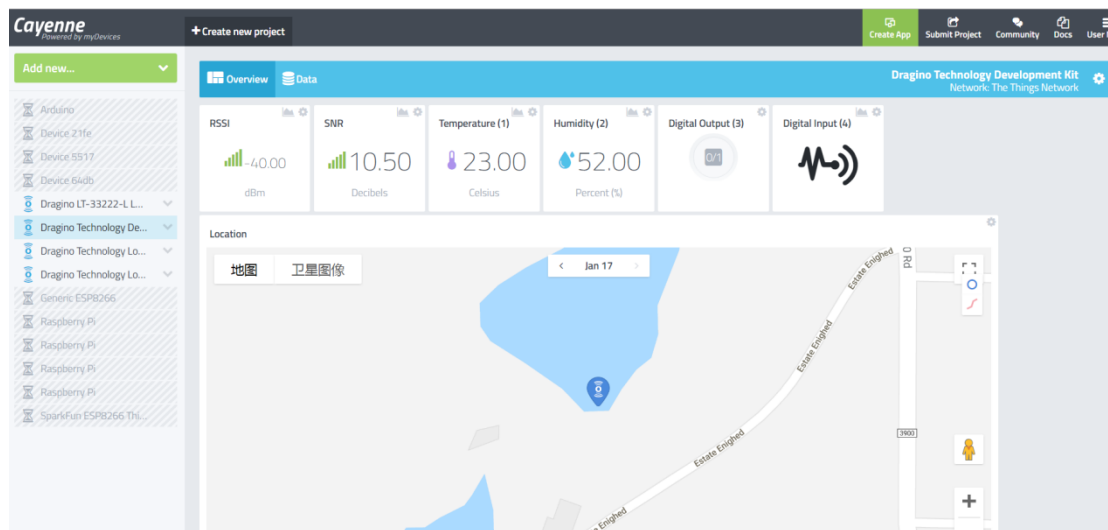
Tracking

Location  
This device doesn't move

Independence, KS 67301美国

**Add device**

After above steps, we can now the sensor data in Mydevices:



### 3.4.4 Use downlink message to control relay

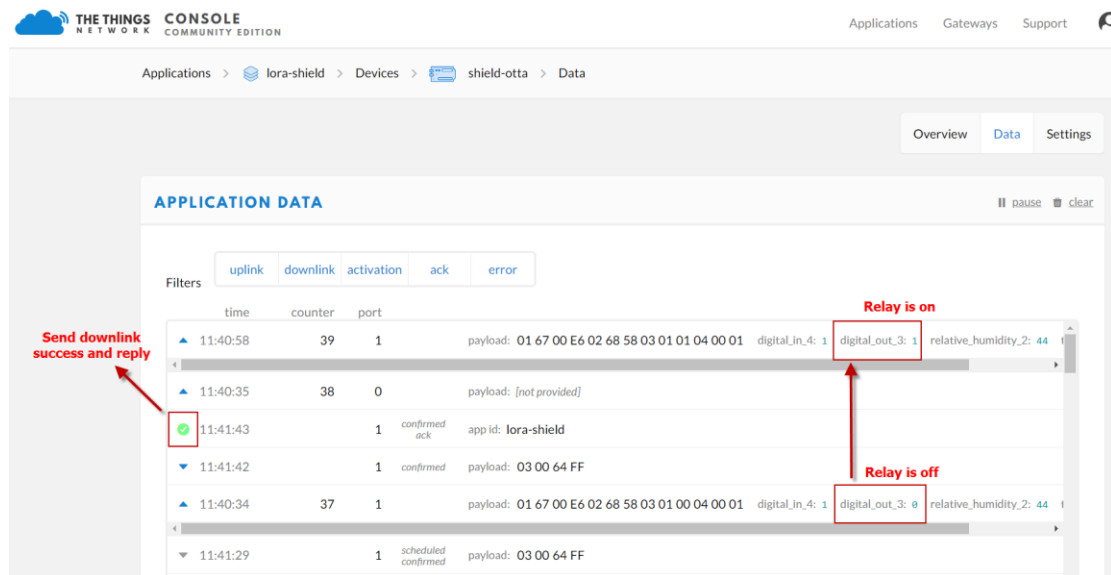
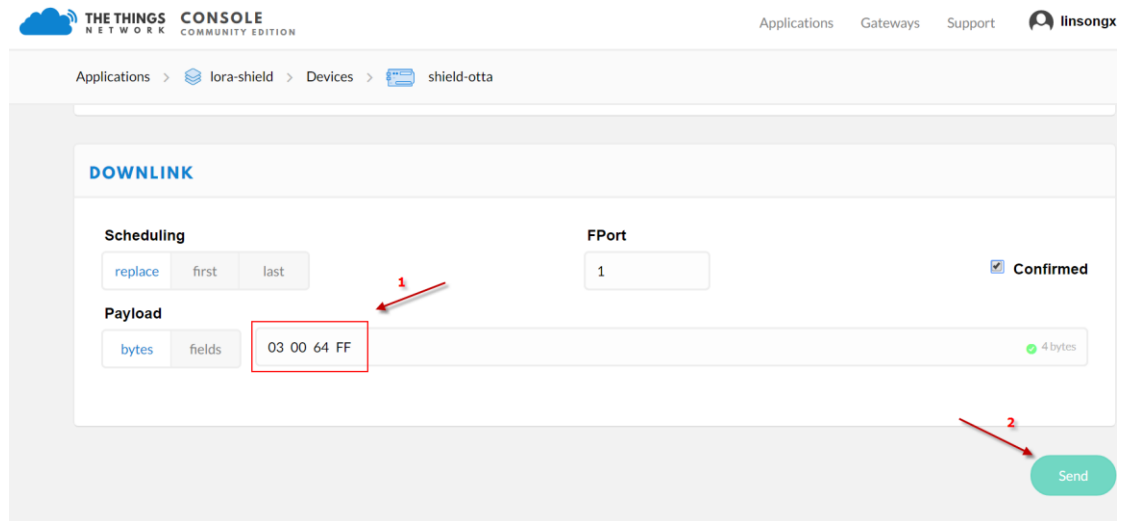
We can use either TTN or Cayenne to control the relay.

#### Control relay via TTN:

The string for ON is: 03 00 64 FF

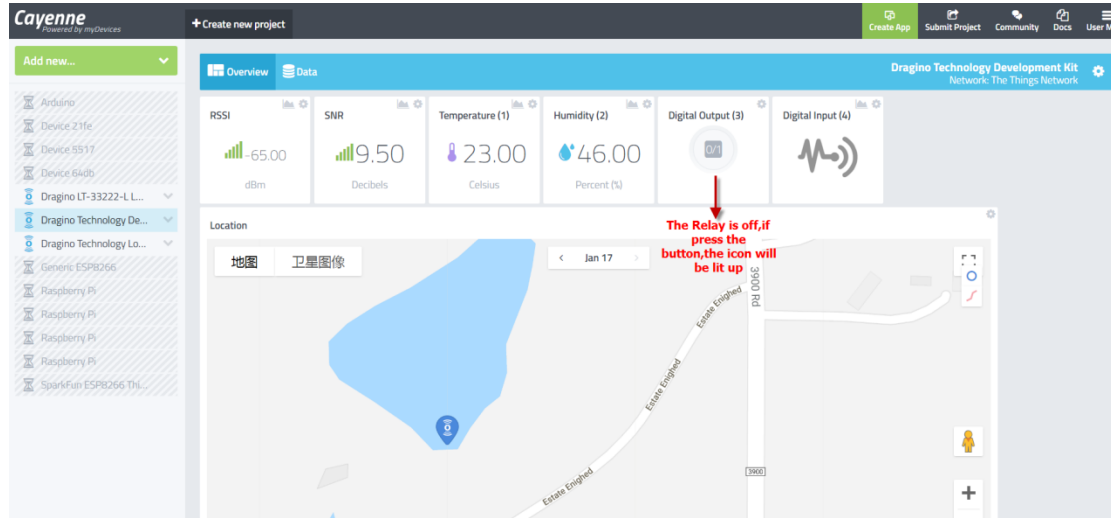
The string for OFF is: 03 00 00 FF

In put above value in the TTN Downlink payload, we can see the relay can switch between different states, since we are in Class A, the downlink will only happen after each uplink.

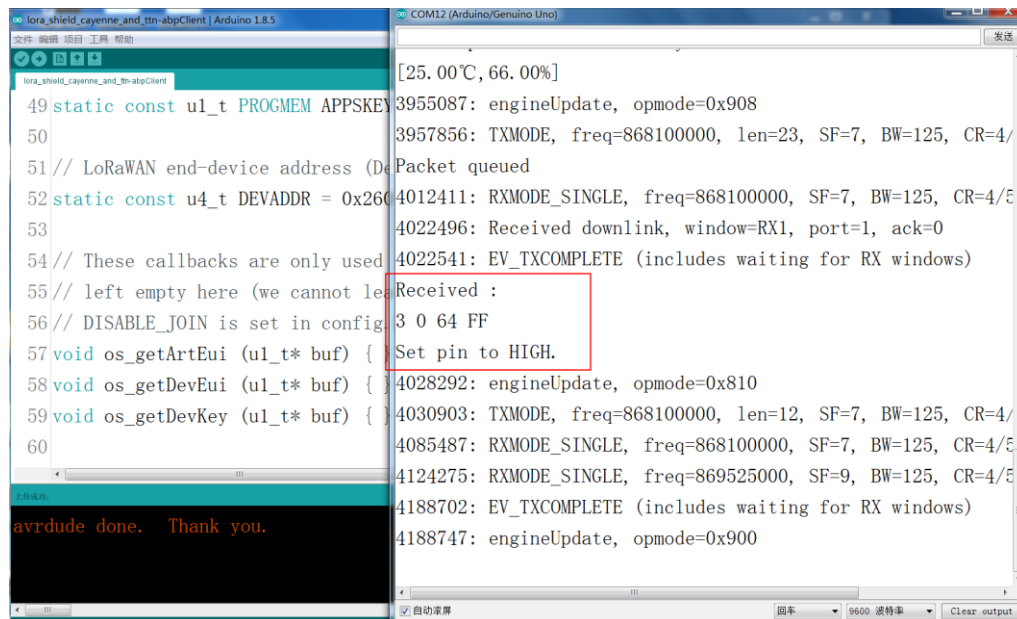


### Control relay via Cayenne

In Cayenne, just click the digital output button, it will auto send out the command strings: ON: 03 00 64 FF , OFF is: 03 00 00 FF



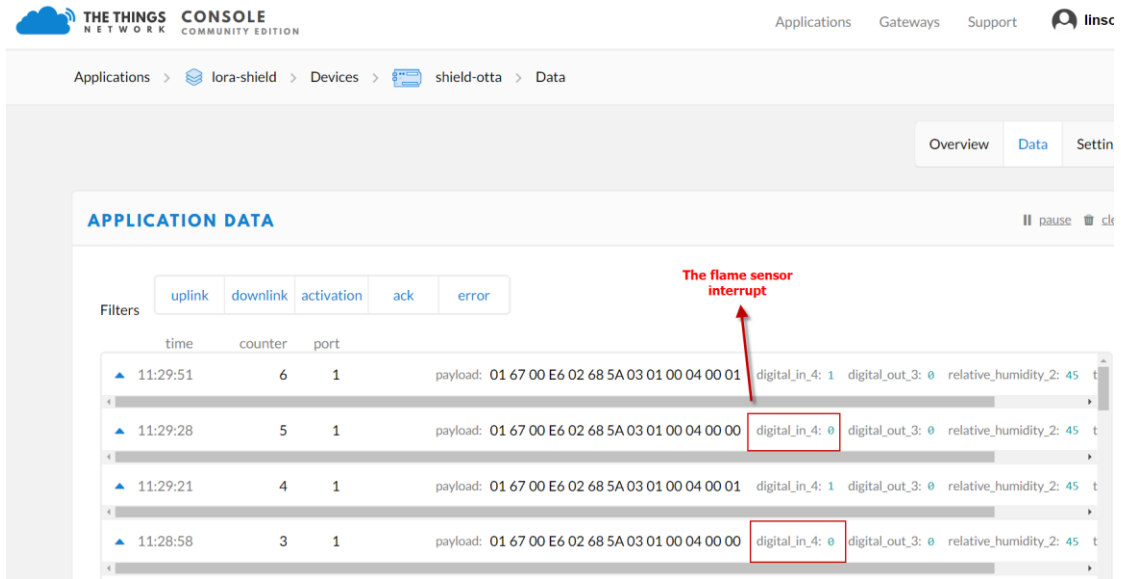
Cayenne will pass the string to TTN and TTN will show as above. In the serial monitor of End Node, we can see below output if downlink string arrives:



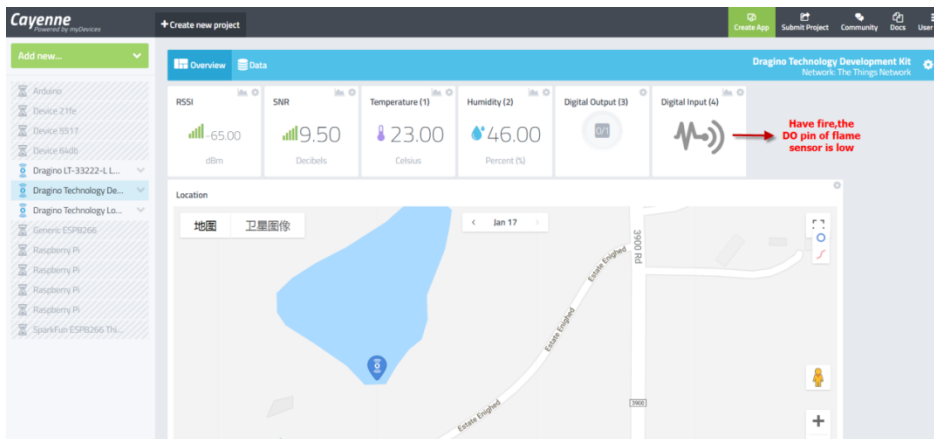
### 3.4.5 Test with Interrupt

The temperature & humidity in this example are updated periodically (once several minutes/hours), in some case, we need to update the data once an action is happen. So we need to use interrupt.

The DO pin of Flame sensor is high in normal case. While it detects a flame, this pin will become low and act as an external interrupt for Arduino. The Arduino UNO will then immediately upload the temperature and humidity to TTN



Then we can see on the cayenne:



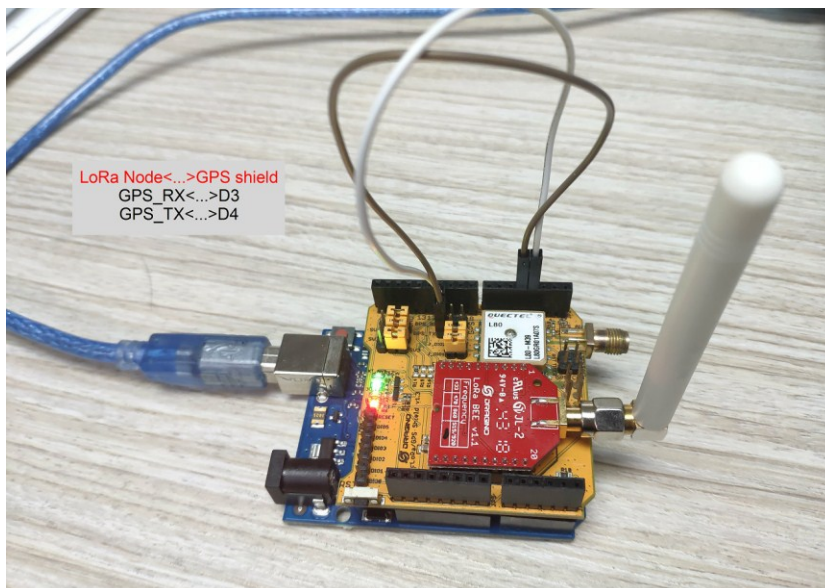
```

3002238: engineUpdate, opmode=0x908
Have fire,the tem##### COUNT=6 #####
The temperature and humidity:
[23.00°C, 45.00%]
3003454: engineUpdate, opmode=0x108
Haveerature is send
##### COUNT=7 #####
The temperature and humidity:
[23.00°C, 45.00%]
3010302: engineUpdate, opmode=0x108
Have fire,the temperature is send
3109920: engineUpdate, opmode=0x108
3110383: TXMODE, freq=868100000, len=26, SF=7, BW=125, CR=4/5, IH=0
3166736: RXMODE_SINGLE, freq=868100000, SF=7, BW=125, CR=4/5, IH=0, rxsyms=255
3205524: RXMODE_SINGLE, freq=869525000, SF=9, BW=125, CR=4/5, IH=0, rxsyms=255
3269940: EV_TXCOMPLETE (includes waiting for RX windows)
3269986: engineUpdate, opmode=0x900
  
```

### 3.5 Create LoRa/GPS Shield End Node

#### 3.5.1 Hardware connection

The method to use LoRa/GPS Shield is similar with LoRa Shield. Below is the hardware connection of LoRa GPS Shield.



#### 3.5.2 Set up ABP device in TTN and upload software to UNO

In LoRa Shield, we set up OTAA for connection. In this example, we will try ABP mode.

**Step 1:** Create an ABP device in TTN server --> Application page. And change it to ABP mode.

CONSOLE COMMUNITY EDITION

Applications Gateways Supp

Applications > dragino\_test\_application1

**APPLICATION EUIS** manage euis

70 B3 D5 7E F0 00 46 18

**DEVICES** register device manage devices

5 registered devices

Applications > dragino\_test\_application1 > Devices > edwintest1 > Settings

Overview Data Settings

**DEVICE SETTINGS**

General Location

**SETTINGS**

**Description**  
A human-readable description of the device

**Device EUI**  
The serial number of your radio module, similar to a MAC address  
00 BA DE A0 36 70 68 72 8 bytes

**Application EUI**  
70 B3D57E F0 00 46 18

**Activation Method**  
OTAA **ABP**

Step 2: Input keys into Arduino Sketch.

The sketch for the LoRa /GPS Shield is [LoRa GPS Sketch code](#)

Applications > dragino\_test\_application1 > Devices > edwintest1

TTN LoRaWAN End Device page

Application ID dragino\_test\_application1

Device ID edwintest1

Activation Method ABP

Device EUI 00 BA DE A0 36 70 68 72

Application EUI 70 B3 D5 7E F0 00 46 18

Device Address 26 01 1C 22

Network Session Key msb 0x9A, 0xEA, 0xD0, 0x93, 0x06, 0xE3, 0x2B, 0x73, 0xDD, 0x54, 0x7B, 0x8B, 0xFF

App Session Key msb 0xB6, 0x07, 0x5B, 0xB5, 0xE4, 0xCE, 0x40, 0xA2, 0xA3, 0xEE, 0x7B, 0xDF, 0xDC

Make sure the Network Session Key and App Session Key are in MSB order

```

ttn-abp

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

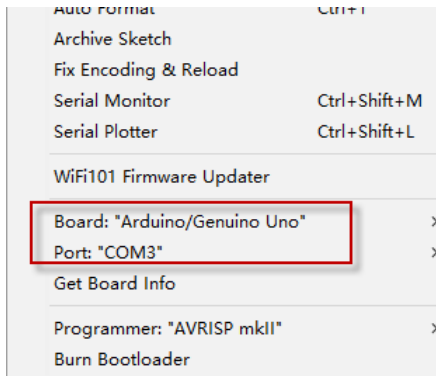
// LoRaWAN NwkSKey, network session key
// This is the default Semtech key, which is used by the early prototype TTN
// network
static const PROGMEM u1_t NwksKey[16] = { 0x9A, 0xEA, 0xD0, 0x93, 0x06, 0xE3, 0x2B, 0x73, 0xDD, 0x54, 0x7B, 0x8B, 0xFF, 0xDC, 0x20, 0xF9 };

// LoRaWAN AppSKey, application session key
// This is the default Semtech key, which is used by the early prototype TTN
// network
static const u1_t PROGMEM AppSKey[16] = { 0xB6, 0x07, 0x5B, 0xB5, 0xE4, 0xCE, 0x40, 0xA2, 0xA3, 0xEE, 0x7B, 0xDF, 0xDC, 0x23, 0x0E, 0x2B };

// LoRaWAN end-device address (DevAddr)
static const u4_t DEVADDR = 0x26011C22 ; // <- Change this address for every node!
    
```

Input the keys from TTN

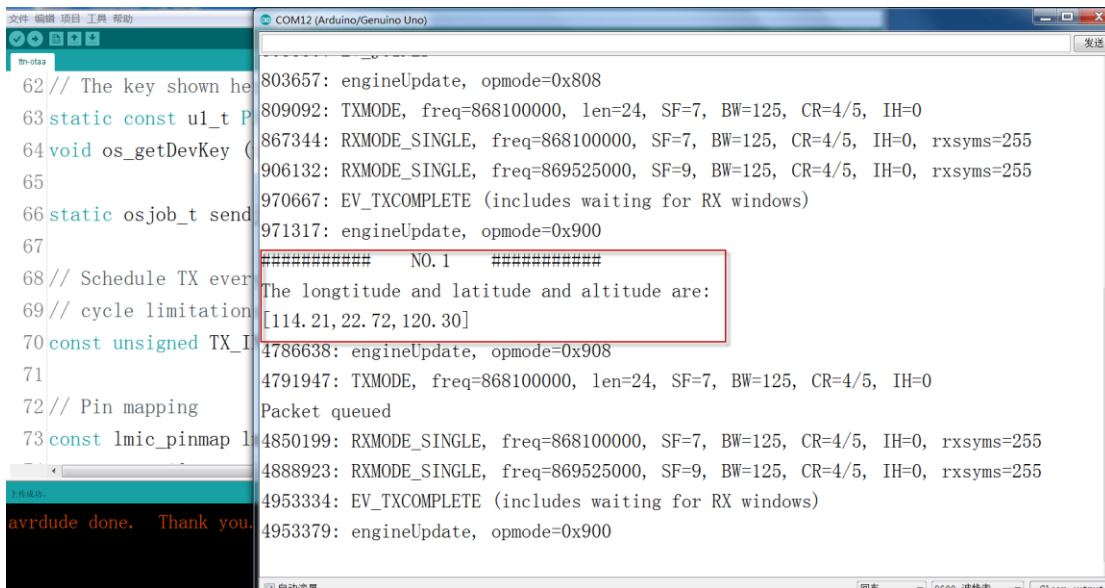
Choose Arduino UNO to upload the sketch to LoRa GPS Shield and UNO



All other steps are similar with how we use with LoRa Shield.

Below are the outputs for reference:

Output from LoRa GPS Shield:





### Upload GPS data to TTN:

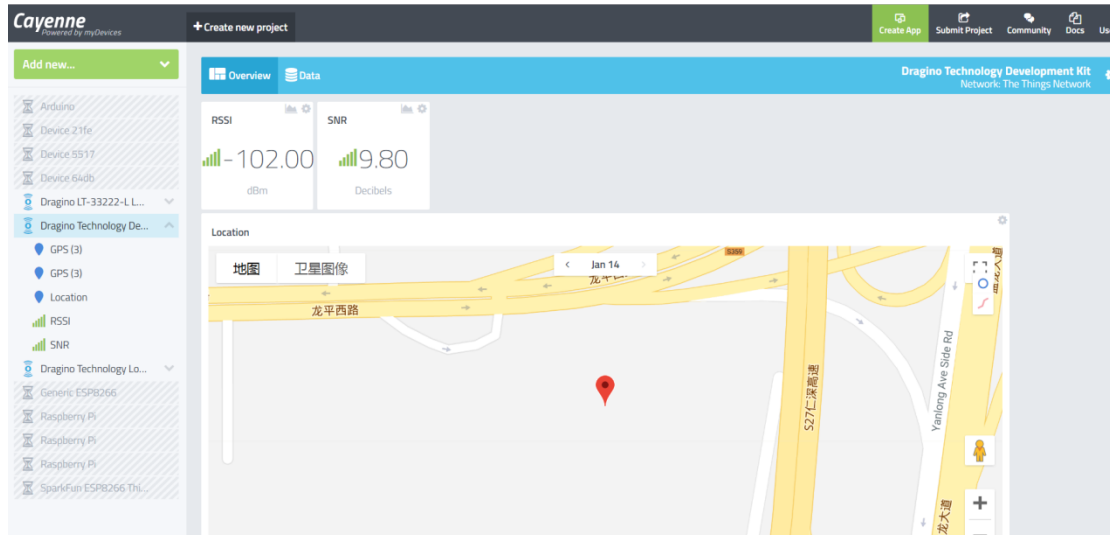
The screenshot shows the 'APPLICATION DATA' section in the TTN console. It displays a table of data points with filters for uplink, downlink, activation, ack, and error. The data points include time, counter, port, payload, and GPS coordinates (altitude, latitude, longitude).

time	counter	port	payload	gps_3.altitude	gps_3.latitude	gps_3.longitude
17:54:23	5	1	payload: 03 88 03 77 A5 11 6D 4C 00 2E FE	120.3	22.7237	114.2092
17:53:19	4	1	payload: 03 88 03 77 A5 11 6D 4C 00 2E FE	120.3	22.7237	114.2092
17:52:15	3	1	payload: 03 88 03 77 A5 11 6D 4C 00 2E FE	120.3	22.7237	114.2092
17:51:12	2	1	payload: 03 88 03 77 A5 11 6D 4C 00 2E FE	120.3	22.7237	114.2092
7:50:08	1	1	payload: 03 88 03 77 A5 11 6D 4C 00 2E FE	120.3	22.7237	114.2092

### Output in Cayenne:

The screenshot shows the Cayenne web interface. On the left, there is a sidebar with a list of devices and sensors. A red arrow labeled '1' points to the 'GPS (3)' sensor. The main area displays a map with a location pin. A red arrow labeled '2' points to the 'Settings' button for the location pin. The map shows a street view with labels like '龙平西路' and 'Yanlong Ave Side Rd'.

The screenshot shows the 'Location' configuration dialog box. It has a 'General' section with a dropdown menu for 'Location' set to 'GPS'. A red arrow labeled '1' points to the dropdown menu. Below the dropdown, there are two options: 'GPS' and 'Physical Address'. A red arrow labeled '2' points to the 'Save' button at the bottom right of the dialog.



### 3.6 Conclusion and limitation

#### 3.6.1 Overview for the example

This example shows how to set up a simple LoRaWAN network with public server. The LoRaWAN specification is for easy deploy the IoT network base on LoRa wireless. It contains the encryption, MAC control, device management etc. More info about LoRaWAN, please see [this link](#).

There are some frequently ask points for the example:

#### 1/ Difference between OTAA & ABP mode:

We have tested OTAA and ABP mode for LoRaWAN. They are two different modes. In OTAA mode, we can see the device will send a join request, the IoT server will send back a Join confirm with dynamic device address, network session key and app session key. Then the device will use these key to communicate with the LoRaWAN server. This make sure the device will only communicate with one server.

In ABP mode, it will use the FIX device address, network session key and app session key. It doesn't have join process. So in theory, any server with match keys is possible to decrypt the data from this end device.

We can see OTAA has better Data security than ABP mode.

#### 2/ AES 128 encryption:

The data between end device and server are AES128 encryption. So the gateway can't parse the packets, it just forward them.

#### 3/ LoRaWAN Network Server:

A LoRaWAN network server is necessary in a LoRaWAN network for device control/management/data management. If user wants to build the NS , there are some open sources LoRaWAN NS such as [LoRaServer](#) can be used. And some gateways already include LoRaWAN NS (this is also a plan for LG01-N).

#### **4/ Downlink message**

In this example, we use LoRaWAN Class A. The end node will open two short downlink windows after each uplink. More info about LoRaWAN class A, please refer [LoRaWAN specification](#).

### 3.6.2 Limitations

The LG01-N is a single channel gateway (Same for LG02). And there are limitations:

**1/ It works only on one frequency at a time.** It can support multiply end nodes, but all end nodes must transmit data at the same frequency so the LG01-N can receive it. For example: if the End node transmits at 868.1Mhz, The LG01-N's RX setting must be 868.1Mhz so to receive this packet.

**2/ It works only for one DR at a time.** DR specifies the Spreading Factor and Bandwidth. In LG01-N, even the rx frequency match , if DR doesn't match, it still can't get the sensor data.

#### **3/ LoRaWAN compatible issue**

In LoRaWAN protocol, the LoRaWAN end nodes send data in a hopping frequency. Since LG01-N only supports one single frequency, it will only be able to receive the packets sent from the same radio parameters (frequency & DR) in LG01-N.

For example, in EU868, a standard LoRaWAN device may send the data in eight frequencies with different Frequency & SF, such as:

```
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
```

So the LG01-N will only able to receive the 868100000, SF7 packet and will not receive others. Means only one packet will arrive the TTN server in every 8 packet sent from the LoRaWAN end node.

If user wants to receive all packets from LoRaWAN end node, user needs to set up the LoRaWAN node to send packets in a single frequency.

#### **4/ Downlink & OTAA issue**

According to the LoRaWAN class A spec, the end node will open two receive windows to get the message from LoRaWAN server for OTAA or downlink function. These two receive windows are quite short (milliseconds), if LoRa packet from the gateway can't reach End Node in the receive window time, the end node won't get the rx message and Downlink / OTAA won't work.

---

In our example, the Arduino LMIC library is modified to enlarge the RX window to let OTAA & downlink works.

## 4 Example 2: Test with a MQTT IoT Server

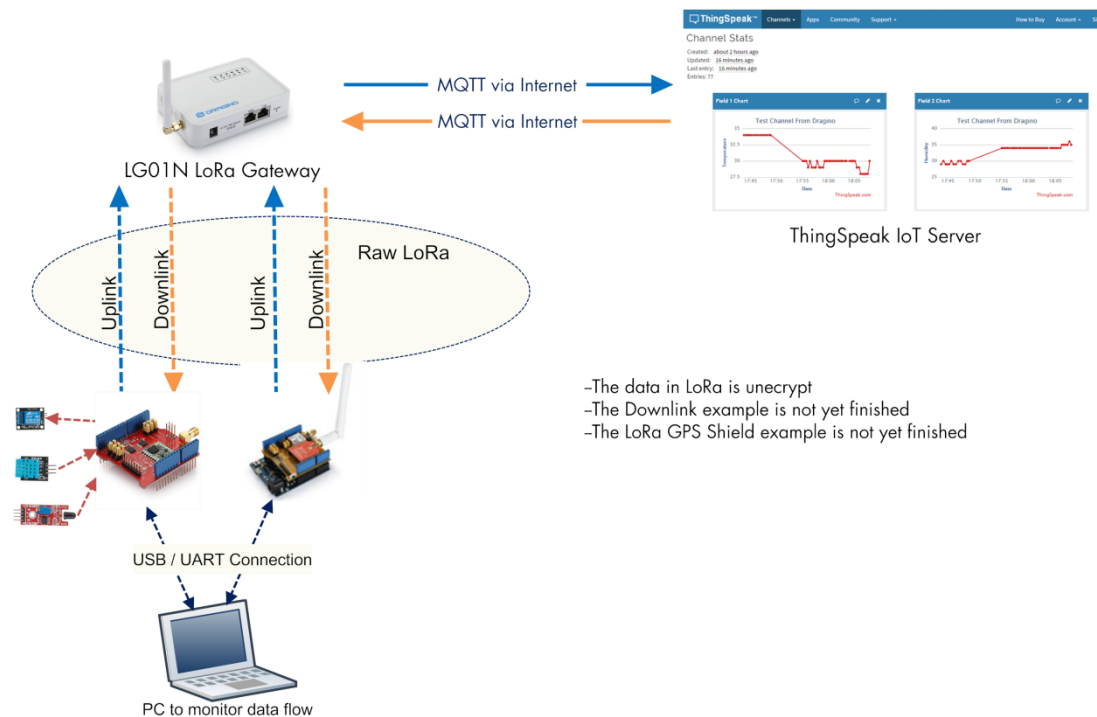
This example describes how to use LG01-N, LoRa Shield & LoRa GPS Shield to set up a LoRa network and connect it to [ThingSpeak IoT Server](https://thingspeak.com).

A Video Instruction of this example can be found at this url: <https://youtu.be/asoNyFYZam0>

### 4.1 Typology and Data Flow

The network topology and dataflow for the example is as below:

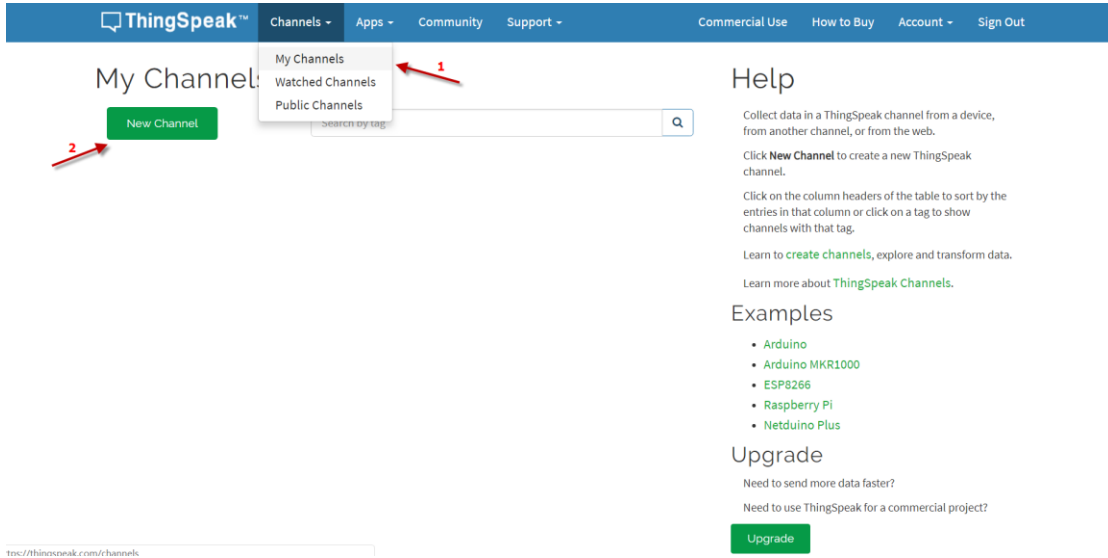
#### Topology for ThingSpeak Connection:



In next section we will start to configure for this example.

## 4.2 Set up sensor channels in ThingSpeak

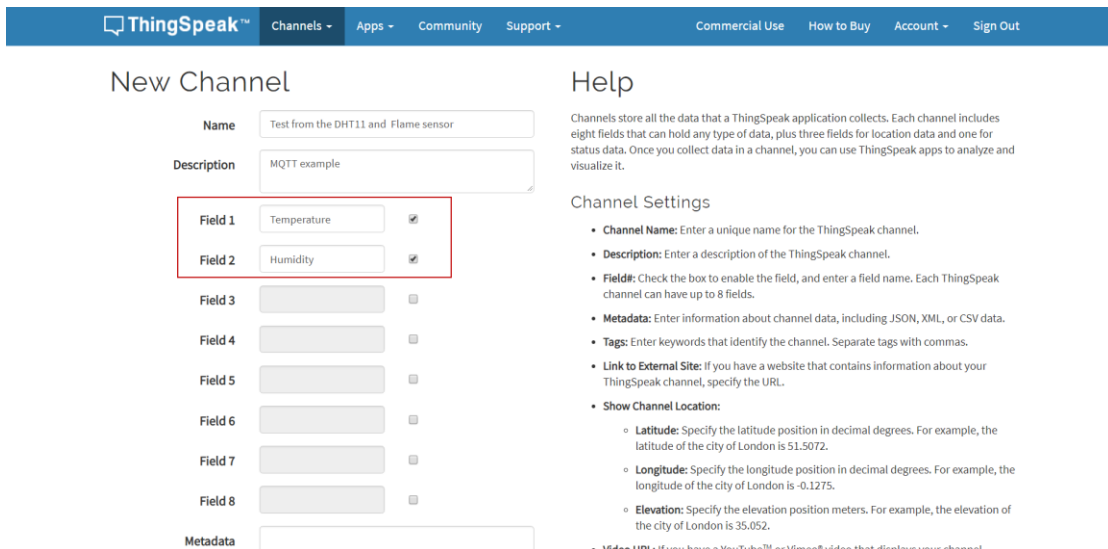
### Step 1: Log in ThingSpeak and set up channels



Set up two channels:

**Field 1: Temperature**

**Field 2: Humidity**



## Step 2: Get MQTT keys for these channels.

Go to Account → My profile and get the [MQTT API Key](#)

The screenshot shows the ThingSpeak Account page. In the 'Account' dropdown menu, 'My Profile' is highlighted with a red arrow and the number '1'. In the 'ThingSpeak settings' section, the 'MQTT API Key' field contains the value 'BYR3I5ECL787PHG9', which is highlighted with a red box and labeled 'Password of MQTT Server'. Below this field is a 'Generate New MQTT API Key' button, which is highlighted with a red box and labeled '2'.

Go to channel page: get the sensor channel:

**Channel ID:** This is the remote Channel ID in ThingSpeak

**Author:** User Name for MQTT connection

**Write API Key:** API key for each channel

The screenshot shows the ThingSpeak Channel page for Channel ID 682338. The 'Channel ID' field is highlighted with a red box and labeled 'Remote Channel'. The 'Author' field is highlighted with a red box and labeled 'User Name of MQTT Server'. The 'Write API Key' section shows a key 'EVDKI16NV993M4XS' in a text input field. The 'Read API Keys' section shows a key 'RU6YwIvTLU44X8M' in a text input field.



### 4.3 Simulate MQTT uplink via PC's MQTT tool

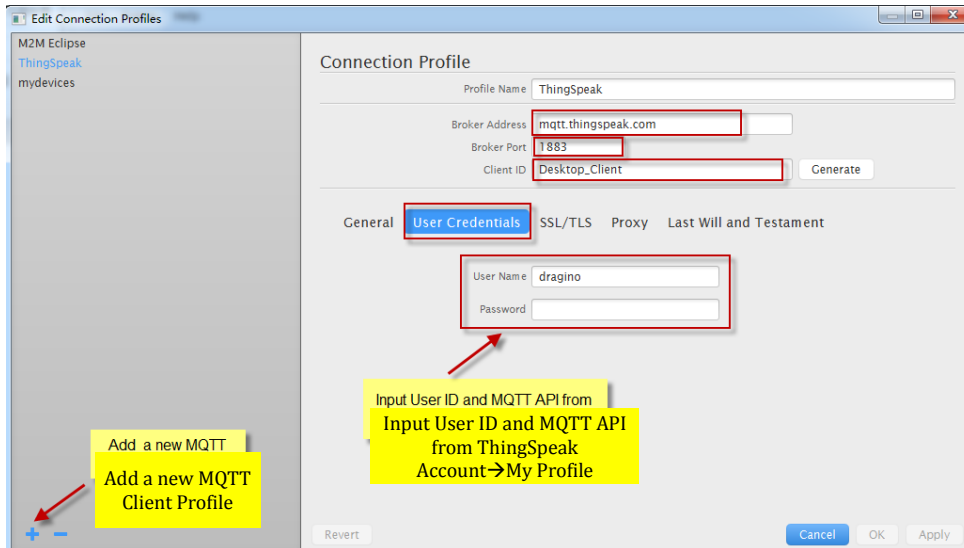
This step is not necessary, it just to help user to understand the MQTT protocol and simulate the MQTT connection to ThingSpeak. And check if the account info is valid and correct.

In the PC, download and install [MQTT.fx](http://MQTT.fx). Open MQTT.fx and configure add a new MQTT client, as below:

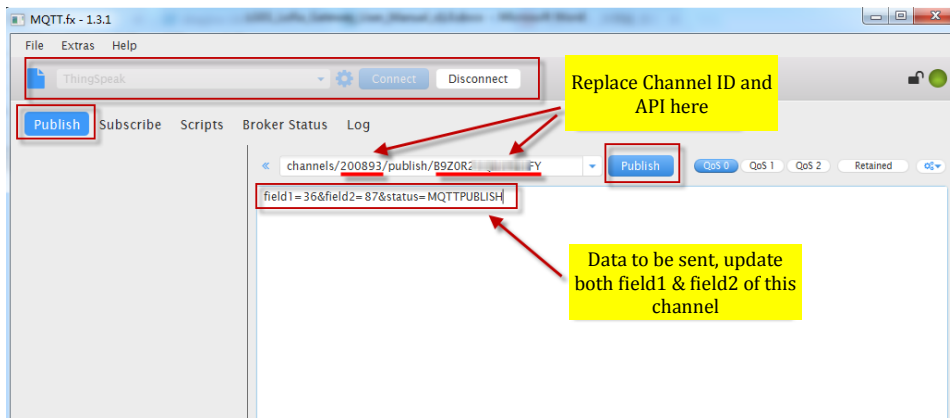
Broker Address: [mqtt.thingspeak.com](http://mqtt.thingspeak.com)

Broker Port: 1883

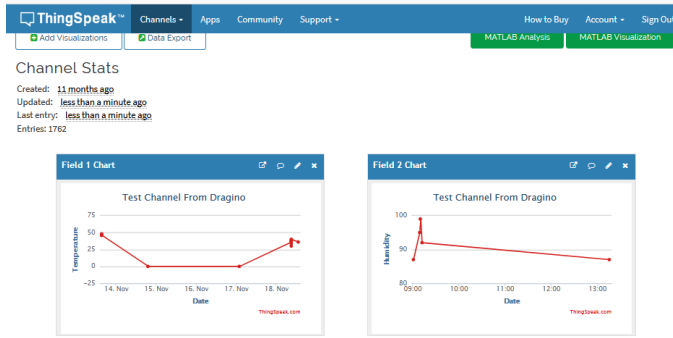
Client ID: User Defined



After add the profile, connect it and publish to the corresponding Channel with correct API key. MQTT API see [this document](#):



If update successful, we can see the update in the channel:



#### 4.4 Try MQTT Publish with LG01-N Linux command

This step is also not necessary; it is to show the basic command used for MQTT connection and will help for further debug when connection fails.

First, we need to make sure the LG01-N has internet access. We can log in the SSH and ping an Internet address and see if there is reply. As below:

```
172.31.255.254 - SecureCRT
文件(F) 编辑(E) 查看(V) 选项(O) 传输(T) 脚本(S) 工具(L) 帮助(H)
| Aliyun_美国服务器 | 172.31.255.254
root@dragino-146d78:~# ping www.163.com
PING www.163.com (58.63.233.35): 56 data bytes
64 bytes from 58.63.233.35: seq=0 ttl=54 time=8.231 ms
64 bytes from 58.63.233.35: seq=1 ttl=54 time=8.709 ms
64 bytes from 58.63.233.35: seq=2 ttl=54 time=8.313 ms
64 bytes from 58.63.233.35: seq=3 ttl=54 time=7.953 ms
64 bytes from 58.63.233.35: seq=4 ttl=54 time=8.539 ms
^C
--- www.163.com ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 7.953/8.349/8.709 ms
root@dragino-146d78:~#
```

LG01-N has built-in Linux utility **mosquitto\_pub**. We can use this command to publish the data to ThingSpeak.

The command to update a feed is as below:

```
mosquitto_pub -h mqtt.thingspeak.com -p 1883 -u dragino -P QZXTxxxxxO2J -i
dragino_Client -t channels/200893/publish/B9Z0R25QNVEBKIFY -m
"field1=34&field2=89&status=MQTTPUBLISH"
```

(Make sure the "" is included, otherwise only one data will be uploaded)

Below is the output window:

```
172.31.255.254 - SecureCRT
文件(F) 编辑(E) 查看(V) 选项(O) 传输(T) 脚本(S) 工具(L) 帮助(H)
| 172.31.255.254
root@dragino-146d78:~# mosquitto_pub -h mqtt.thingspeak.com -p 1883 -u dragino -P Q
ZXTxxxxxO2J -i dragino_Client -t channels/200893/publish/B9Z0R25QNVEBKIFY -m "
field1=34&field2=89&status=MQTTPUBLISH"
root@dragino-146d78:~#
```

After running this command, we can see the data are updated to ThingSpeak, which has same result as what we did at mqtt.fx.

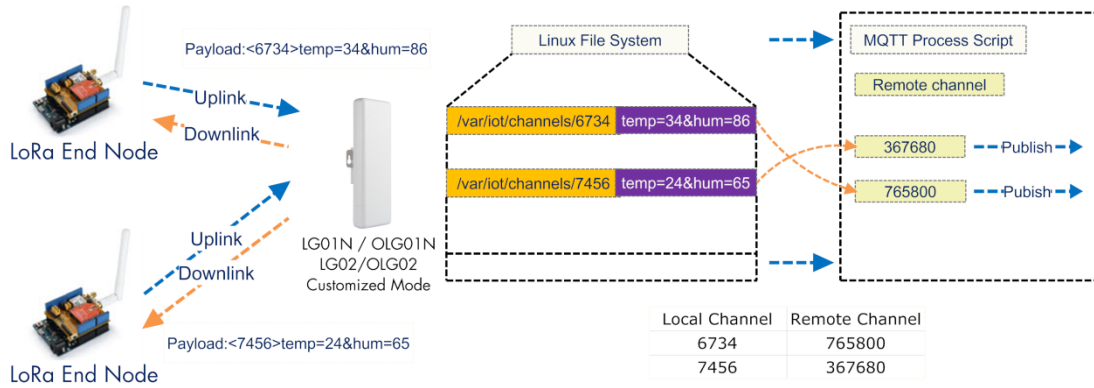
So we success to use LG01-N to uplink data to ThingSpeak, the **mosquitto\_pub** command is executed in the Linux side, finally, we will have to call **mosquitto\_pub** command while the LoRa sensor data arrive. We will explain how to do that in next step.

## 4.5 Configure LG01-N Gateway

### 4.5.1 Publish Logic

In LG01-N (firmware version > LG02\_LG08--build-v5.1.1545908833-20181227-1908), there is a built-in script to process the MQTT data. The logic of this flow is as below:

#### How MQTT script works:



#### Operate Principle:

- > LoRa End Node sends the data to gateway in specify format: <node\_ID>value
- > Gateway get the data and will put the data in corresponding files under /var/iot/channels.
- > MQTT Process Script will publish data to remote channel according to the pre-configure mapping

### Step1: Configure LG01-N to act as MQTT mode

The screenshot shows the 'dragino-1b7060' web interface. Under 'LoRa Gateway Settings', the 'LoRaWAN Server Settings' section is highlighted. The 'IoT Service' dropdown is set to 'LoRaRAW forward to MQTT ser'. Other settings include 'Debug Level' set to 'Little message output' and 'Service Provider' set to 'The Things Network'.

### Step2: Configure MQTT server info

The screenshot shows the 'MQTT Server Settings' configuration page. Under 'Configure MQTT Server', the following settings are visible: 'Select Server' is 'ThingSpeak MQTT', 'User Name [-u]' is 'dragino1', 'Password [-P]' is '32W6GMEXYTEQ7049', and 'Client ID [-i]' is 'dragino\_Client'.

In step 2, we have below settings:

- ✓ UserName[-u option]: Input Author (user name for MQTT Connection)
- ✓ Password[-P option]: Input MQTT API key

- ✓ Client\_ID[-i]: dragino\_Client (can put any string)
- ✓ Because we choose Thingspeak so we have below pre-set options but not show in web
  - Broker Address[-h]: mqtt.thingspeak.com
  - Broker Port[-p]: 1883
  - Topic Format[-t]: channels/CHANNEL/publish/WRITE\_API.
  - Data String Format[-m]: DATA&status=MQTTPUBLISH

And we configure this channel:

- ✓ Local Channel ID: 10009
- ✓ Remote Channel ID: 396640
- ✓ Write\_api\_key: Write API key for this channel.

In the mqtt script, the upper **CHANNEL** will be replaced by the parameter (remote channel in IoT server). and the **WRITE\_API** will be replaced by the settings in write api key. The **DATA** will be replaced by the value stored in the /var/iot/channels/LOCAL\_CHANNEL file.

MQTT script will keep checking the files in /var/iot/channels/. If it finds a match Local channel, then the MQTT script will send out the data of this local channel to a remote channel according to the setting above.

User can also enable MQTT debug level and run logread in Linux console to see how the mqtt command is compose. Below is an example:

```
Tue Nov 27 15:07:43 2018 kern.notice syslog: [IoT.MQTT]: Found Local Channels:
Tue Nov 27 15:07:49 2018 kern.notice syslog: [IoT.MQTT]: Check for sensor update
Tue Nov 27 15:07:49 2018 kern.notice syslog: [IoT.MQTT]: Found Local Channels:
Tue Nov 27 15:07:55 2018 kern.notice syslog: [IoT.MQTT]: Check for sensor update
Tue Nov 27 15:07:55 2018 kern.notice syslog: [IoT.MQTT]: Found Local Channels:
Tue Nov 27 15:07:59 2018 kern.notice syslog: [IoT]: Internet Connection Check: FAIL
Tue Nov 27 15:08:01 2018 kern.notice syslog: [IoT.MQTT]: Check for sensor update
Tue Nov 27 15:08:01 2018 kern.notice syslog: [IoT.MQTT]: Found Local Channels:
Tue Nov 27 15:08:02 2018 kern.notice syslog: [IoT]: DNS Resolve Check: FAIL
Tue Nov 27 15:08:03 2018 kern.notice syslog: [IoT.MQTT]: Broker-Port:mqtt.mydevices.com:1883
Tue Nov 27 15:08:03 2018 kern.notice syslog: [IoT.MQTT]: Topic Format: v1/USERNAME/things/CLIENTID/data/CHANNEL
Tue Nov 27 15:08:03 2018 kern.notice syslog: [IoT.MQTT]: Data Format: DATA
Tue Nov 27 15:08:09 2018 kern.notice syslog: [IoT.MQTT]: Check for sensor update
Tue Nov 27 15:08:09 2018 kern.notice syslog: [IoT.MQTT]: Found Local Channels: 100
Tue Nov 27 15:08:09 2018 kern.notice syslog: [IoT.MQTT]: Find Match Entry For 100
Tue Nov 27 15:08:09 2018 kern.notice syslog: [IoT.MQTT]: [-t] v1/e74b78d0-3858-11e7-afce-8d5fd2a310a7/things/2b1fab30-3859-11e7-afce-8d5fd2a310a7/data/0
Tue Nov 27 15:08:09 2018 kern.notice syslog: [IoT.MQTT]: [-m] temp,c=30.2
root@dragino-193a18:~#
```

### 4.5.2 Configure LG01-N's Radio frequency

Now we should configure LG01-N's radio parameter to receive the LoRaWAN packets. We are using 868.0Mhz (868000000 Hz) as below:

dragino-1893c4 Status System Network Service Logout

Latitude

Longitude

Radio Power (Unit:dBm)

### Radio Settings

Radio settings for Channel

Frequency (Unit:Hz)  **1**

Spreading Factor  **2**

Coding Rate

Signal Bandwidth

Preamble Length

Length range: 6 ~ 65536

LoRa Sync Word

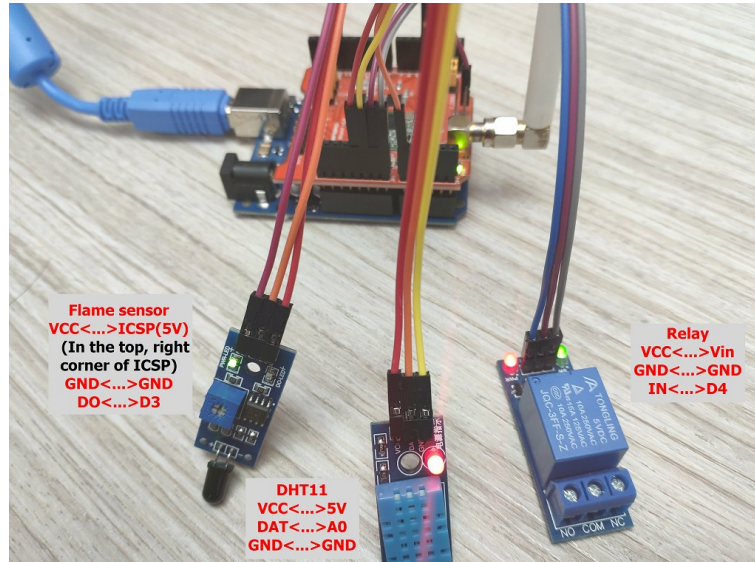
Value 52(0x34) for LoRaWAN

Encryption Key

**3**

## 4.6 Create LoRa Shield End Node

### 4.6.1 Hardware Connection



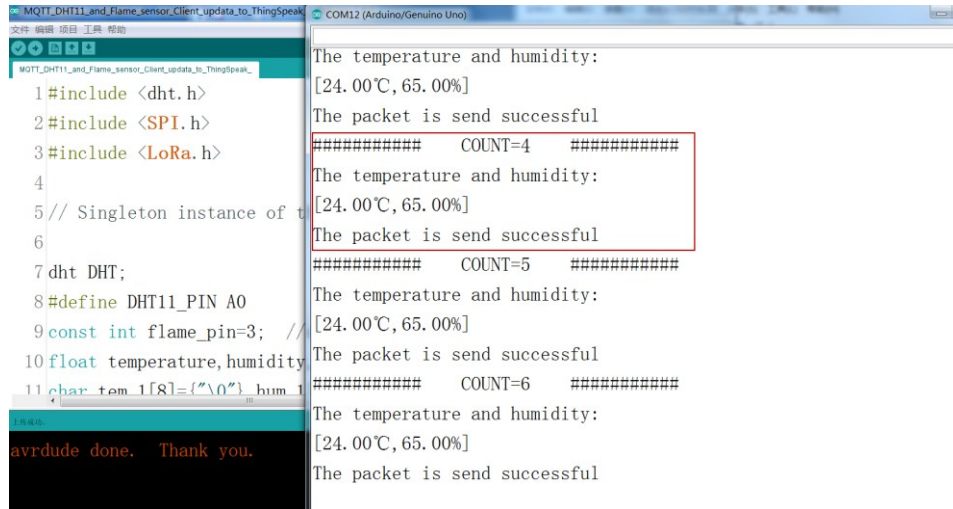
There are three sensors connect to the LoRa Shield + UNO. These sensors are flame sensors, DHT11 (Temperature & Humidity sensor) and Relay. Please use the connection as we show in the photo.

**Note:** There is a trick above, the relay is connected to VIN. In this case, The UNO can only be power via USB port. If need to power via DC power adapter, please use another 5v pin to power relay.

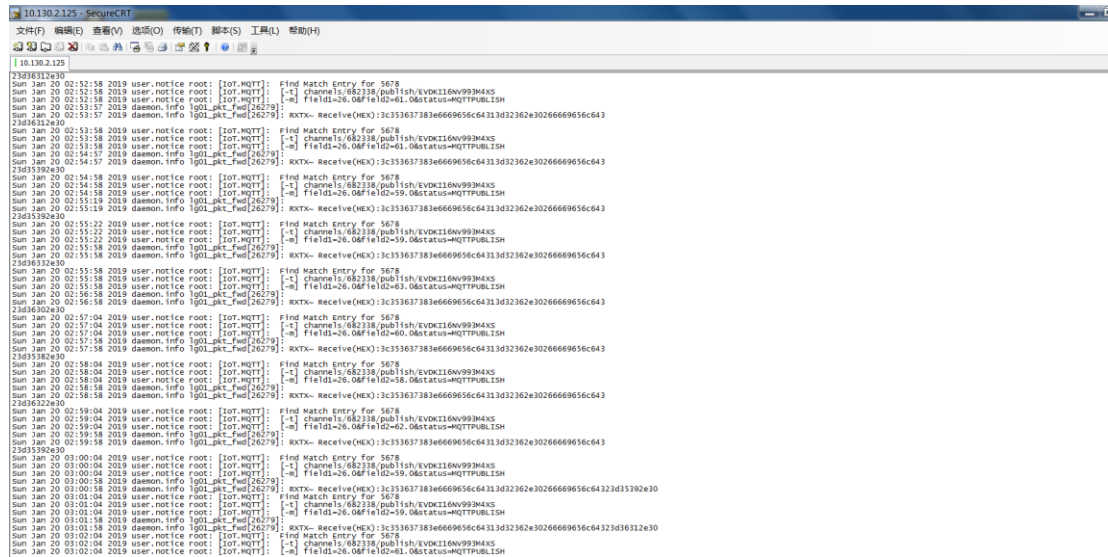
Upload [this sketch](#) to the UNO, this sketch will send temperature and humidity data to gateway at every 60 seconds. If there is a flame detect, it will then immediately send the value to gateway and then upload to the IoT Server.

## 4.6.2 Test with uplink

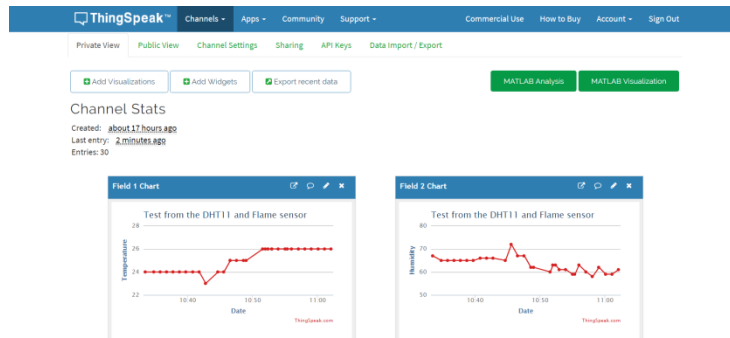
After we upload the sketch to UNO, we can see below output from Arduino



And we can see the logread of gateway as below, means the packet arrive gateway:



Finally, we can see on the ThingSpeak:

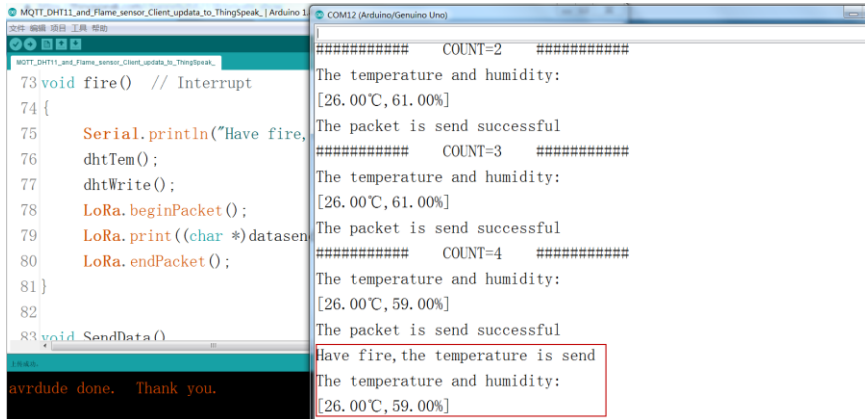




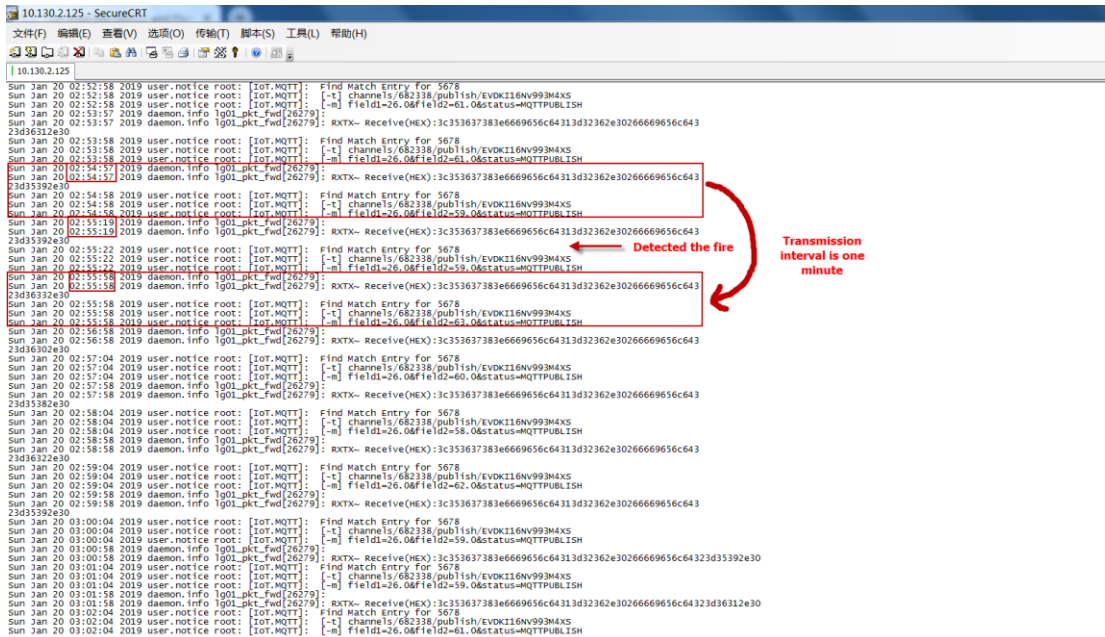
### 4.6.3 Test with interrupt by flame detect

The DO pin of Flame sensor is high in normal state. When a flame is detected, the DO pin of Flame sensor will become low, then, the UNO generates an external interrupt, and immediately uploads the temperature and humidity to the server.

The DO pin of Flame sensor is low when a flame is detected, and we can see on the Serial Monitor:



Similarly, we can see the logread of gateway via SSH access:



Finally, we can see on the ThingSpeak:

The screenshot displays the ThingSpeak interface for a channel. At the top, there is a navigation bar with 'ThingSpeak™' and various menu items like 'Channels', 'Apps', 'Community', 'Support', 'Commercial Use', 'How to Buy', 'Account', and 'Sign Out'. Below this, there are tabs for 'Private View', 'Public View', 'Channel Settings', 'Sharing', 'API Keys', and 'Data Import / Export'. There are also buttons for 'Add Visualizations', 'Add Widgets', 'Export recent data', 'MATLAB Analysis', and 'MATLAB Visualization'.

**Channel Stats**  
Created: [about 17 hours ago](#)  
Last entry: [2 minutes ago](#)  
Entries: 30

**Field 1 Chart**  
Title: Test from the DHT11 and Flame sensor  
Y-axis: Temperature (22 to 28)  
X-axis: Date (10:40 to 11:00)  
A red line graph shows temperature fluctuating around 24-25. At approximately 10:55, there is a sharp spike to about 26.5, which is highlighted with a red box and labeled 'Detected the fire' with two red arrows pointing to the peak.

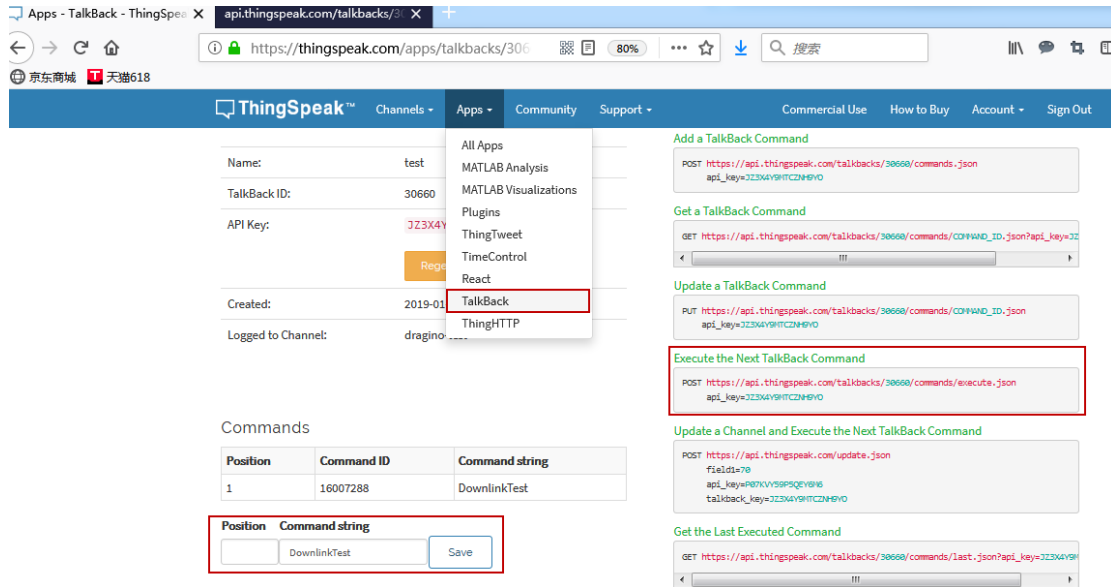
**Field 2 Chart**  
Title: Test from the DHT11 and Flame sensor  
Y-axis: Humidity (50 to 80)  
X-axis: Date (10:40 to 11:00)  
A red line graph shows humidity fluctuating between 60 and 70.

#### 4.6.4 Test with downlink

The http downlink feature is now support since firmware LG02\_LG08--build--v5.2.1560931576--20190619-1607.

ThingSpeak downlink command can be found in TalkBack App.

The **Command String input box** is the command you want to send to LoRa device.

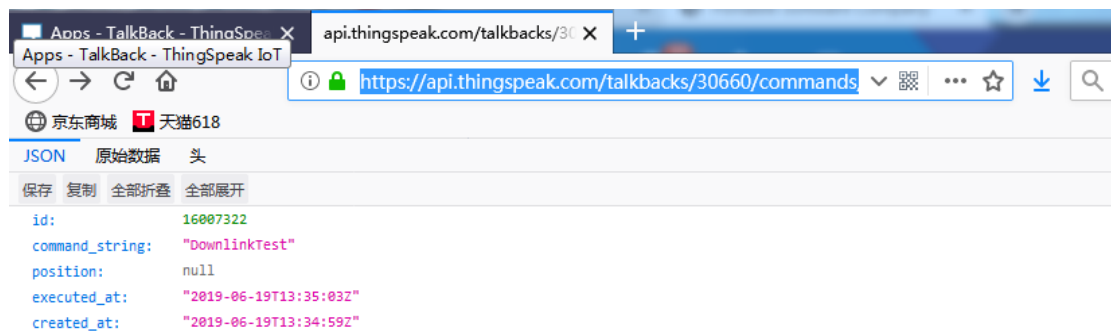


**Execute The next Talkback Command** is the API to get one command from the commands queue.

We can test in the web with this API. Format is:

[https://api.thingspeak.com/talkbacks/XXXXXX/commands/execute.json?api\\_key=XXXXXXXXXX](https://api.thingspeak.com/talkbacks/XXXXXX/commands/execute.json?api_key=XXXXXXXXXX)

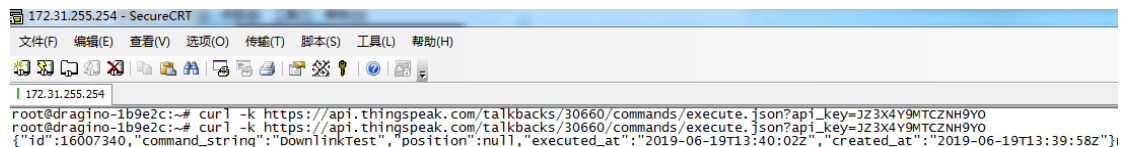
Result as below:



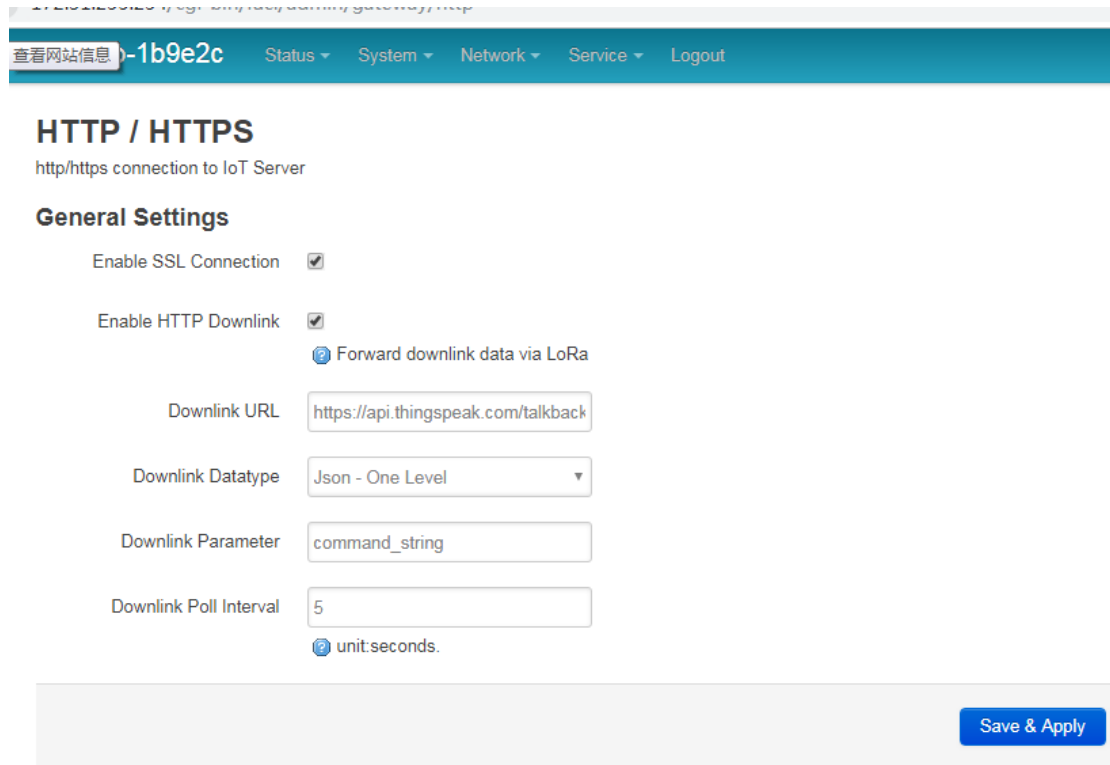
We can also test this API in LG01-N Linux console:

By using:

`curl -k https://api.thingspeak.com/talkbacks/XXXXXX/commands/execute.json?api_key=XXXXXXXXXX`

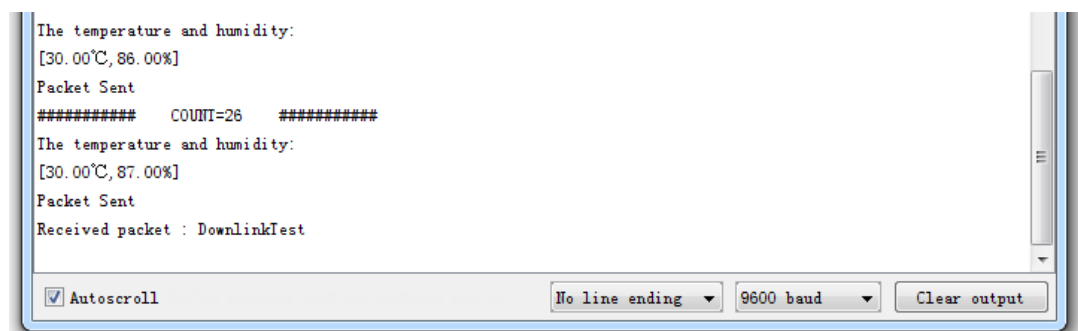


To get this result automatically in LG01-P and send out via LoRa, we can configure as below:



- Because URL is https, So need to Enable SSL Connection
- Downlink URL use the URL we use in Web and Curl
- Downlink datatype for ThingSpeak is Json.
- Downlink Parameter is command\_string. We will fetch the value of `command_string` from the downlink data string.
- LG01-N will poll the URL every 5 seconds. When there is valid `command_string` found, it will send out via LoRa (Radio parameter is defined in LoRaWAN gateway Radio settings or Radio2 settings for LG02)

Result in the LoRa Shield:



## 4.7 Conclusion and limitation

### 4.7.1 Overview for the example

This example shows how to set up a simple LoRa network with ThingSpeak IoT server. In this example, we use the raw LoRa protocol (private protocol) for transmission. It is simpler compare via LoRaWAN protocol

There are some frequently ask points for the example:

#### 1/ Difference between LoRaWAN & Private LoRa protocol:

- The private LoRa protocol here doesn't have MAC control/management, (of course developer can develop this). In LoRaWAN protocol, this feature is supported already.
- The transmission is unencrypted in this example, user can see the data in gateway. In LoRaWAN, the transmission is designed in AES encryption.
- Private LoRa protocol means the gateway only works with specify LoRa End node which runs the same protocol, the gateway can't work with a standard LoRaWAN devices.
- Private LoRa protocol doesn't need the LoRaWAN IoT Server. Gateway can send data to user defined IoT server, in terms the gateway and the server can communicate with each other.
- User can more features in the private protocol such as MAC control, encryption, that is how LoRaWAN protocol comes, the advantage of LoRaWAN protocol is that it is designed for carrier level use, and developer can use it directly with many features and compatible with the LoRaWAN node from different manufacturers.

## 5 Order Info

### LoRa\_IoT\_Kit-v2-XXX-YYY

#### XXX: Frequency Band

433: For Bands: EU433, CN470

868: For Bands: EU868, IN865

915: For Bands: US915, AU915, AS923, KR920

#### YYY: 4G Cellular Option

EC25-E: EMEA, Korea, Thailand, India.

EC25-A: North America/ Rogers/AT&T/T-Mobile.

EC25-AU: Latin America, New Zeland, Taiwan

EC25-J: Japan, DOCOMO/SoftBank/ KDDI

More info about valid bands, please see EC25-E product page

(<https://www.quectel.com/product/ec25.htm>)

## 6 FAQ & Trouble Shooting

### 6.1 I can't upload sketch to LoRa Shield in MAC OS, shows " dev/cu.usbmodem1421 is not available "

Error Info as below:

```
Arduino: 1.8.3 (Mac OS X), Board: "Arduino/Genuino Uno"  
Archiving built core (caching) in:  
/var/folders/jq/8fnvlfj90tgbnbcyd16_bbw0000gn/T/arduino_cache_833512/core/core_arduino  
_avr_uno_fc9a32205aafa27e4eda988d5ed9b7ac.a  
Sketch uses 20142 bytes (62%) of program storage space. Maximum is 32256 bytes.  
Global variables use 1189 bytes (58%) of dynamic memory, leaving 859 bytes for local variables.  
Maximum is 2048 bytes.  
Board at /dev/cu.usbmodem1421 is not available
```

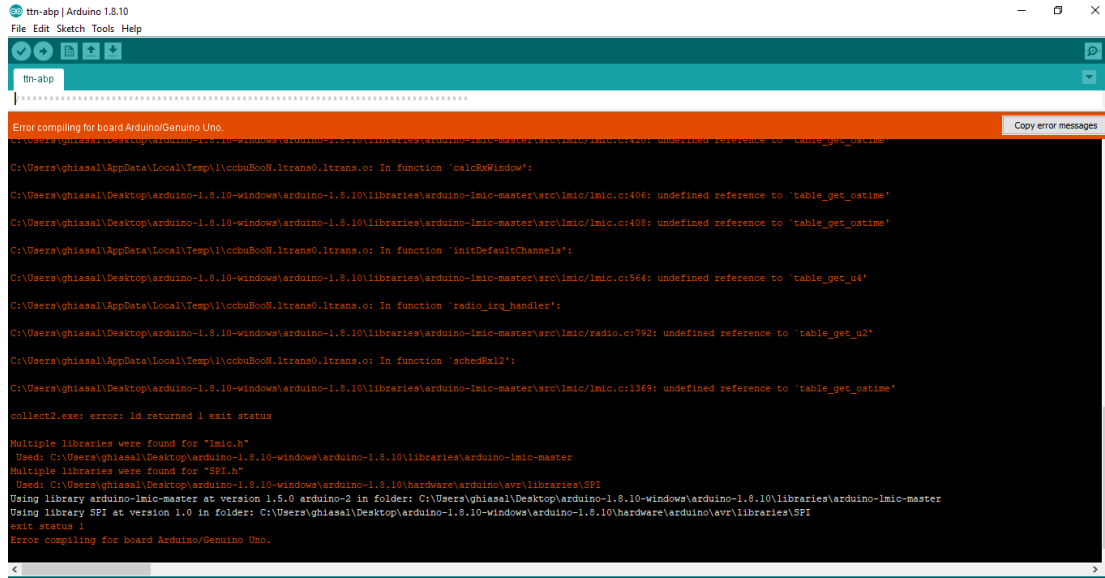
The Arduino UNOs in the Kit are clone version and use CH340 USB to serial chip. User has to install the CH340 driver in PC to make it work. Above issue means the MAC OS doesn't has CH340 driver.

### 6.2 My IoT Kit has the model LG01-P instead of LG01-N, Can I still use this manual.

The gateway part of this manual is for LG01-N, if user has the LG01-P version, please check the [LG01-P gateway manual](#).

### 6.3 Duplicate library issue while upload in Arduino IDE 1.8.10.

While compile the LMIC library in Arduino IDE 1.8.10. This error will happen:



```

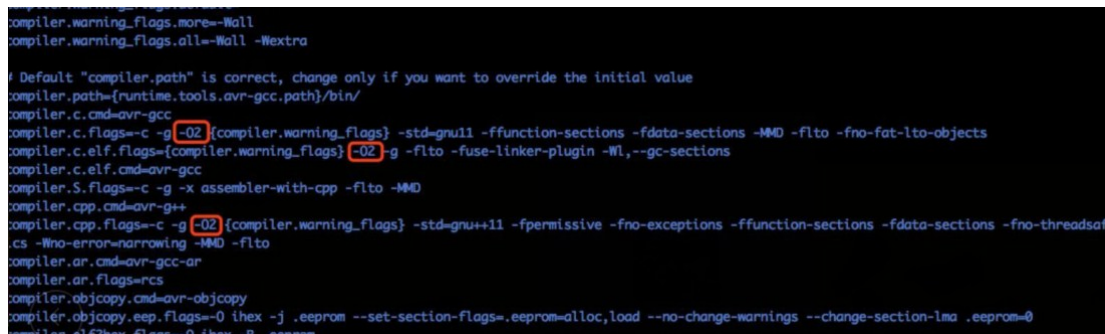
ttn-abp | Arduino 1.8.10
File Edit Sketch Tools Help
ttn-abp
Error compiling for board Arduino/Genuino Uno.
C:\Users\ghiasal\AppData\Local\Temp\1\cchuBooN.ltrans0.ltrans.o: In function `calcRxWindow':
C:\Users\ghiasal\Desktop\arduino-1.8.10-windows\arduino-1.8.10\libraries\arduino-lmic-master\src\lmic\lmic.c:406: undefined reference to `table_get_ostime'
C:\Users\ghiasal\Desktop\arduino-1.8.10-windows\arduino-1.8.10\libraries\arduino-lmic-master\src\lmic\lmic.c:408: undefined reference to `table_get_ostime'
C:\Users\ghiasal\AppData\Local\Temp\1\cchuBooN.ltrans0.ltrans.o: In function `initDefaultChannels':
C:\Users\ghiasal\Desktop\arduino-1.8.10-windows\arduino-1.8.10\libraries\arduino-lmic-master\src\lmic\lmic.c:564: undefined reference to `table_get_u4'
C:\Users\ghiasal\AppData\Local\Temp\1\cchuBooN.ltrans0.ltrans.o: In function `radio_irq_handler':
C:\Users\ghiasal\Desktop\arduino-1.8.10-windows\arduino-1.8.10\libraries\arduino-lmic-master\src\lmic\radio.c:792: undefined reference to `table_get_u2'
C:\Users\ghiasal\AppData\Local\Temp\1\cchuBooN.ltrans0.ltrans.o: In function `schedRx12':
C:\Users\ghiasal\Desktop\arduino-1.8.10-windows\arduino-1.8.10\libraries\arduino-lmic-master\src\lmic\lmic.c:1369: undefined reference to `table_get_ostime'
collect.exe: error: ld returned 1 exit status
Multiple libraries were found for "lmic.h"
Used: C:\Users\ghiasal\Desktop\arduino-1.8.10-windows\arduino-1.8.10\libraries\arduino-lmic-master
Multiple libraries were found for "SPI.h"
Used: C:\Users\ghiasal\Desktop\arduino-1.8.10-windows\arduino-1.8.10\hardware\arduino\avr\libraries\SPI
Using library arduino-lmic-master at version 1.5.0 in folder: C:\Users\ghiasal\Desktop\arduino-1.8.10-windows\arduino-1.8.10\libraries\arduino-lmic-master
Using library SPI at version 1.0 in folder: C:\Users\ghiasal\Desktop\arduino-1.8.10-windows\arduino-1.8.10\hardware\arduino\avr\libraries\SPI
exit status 1
Error compiling for board Arduino/Genuino Uno.

```

To solve this, user can modify the file:

~\Dev\Arduino.app\Contents\Java\hardware\arduino\avr\platform.txt

, Change the compile flag from -Os to -O2. Like below:



```

compiler.warning_flags.more=-Wall
compiler.warning_flags.all=-Wall -Wextra

# Default "compiler.path" is correct, change only if you want to override the initial value
compiler.path={runtime.tools.avr-gcc.path}/bin/
compiler.cmd=avr-gcc
compiler.c.flags=-c -g -O2 {compiler.warning_flags} -std-gnu11 -ffunction-sections -fdata-sections -MMD -flto -fno-fat-lto-objects
compiler.c.elf.flags={compiler.warning_flags} -O2 -g -flto -fuse-linker-plugin -Wl,--gc-sections
compiler.c.elf.cmd=avr-gcc
compiler.S.flags=-c -g -x assembler-with-cpp -flto -MMD
compiler.cpp.flags=-c -g -O2 {compiler.warning_flags} -std-gnu++11 -fpermissive -fno-exceptions -ffunction-sections -fdata-sections -fno-threadsafe
cs -Wno-error=narrowing -MMD -flto
compiler.ar.cmd=avr-gcc-ar
compiler.ar.flags=rscs
compiler.objcopy.cmd=avr-objcopy
compiler.objcopy.eep.flags=-O ihex -j .eeprom --set-section-flags=.eeprom=alloc,load --no-change-warnings --change-section-lma .eeprom=0
compiler.elf2hex.flags=-O ihex -R .eeprom

```



## 6.4 How can I set to use CN470 band?

```
11 // #define CFG_as921 1
12 // #define CFG_as923 1
13 // #define CFG_in866 1
14
15 #define LG02_LG01 1
16
17 // US915: DR_SF10=0, DR_SF9=1, DR_SF8=2, DR_SF7=3, DR_SF8C=4
18 //      DR_SF12CR=8, DR_SF11CR=9, DR_SF10CR=10, DR_SF9CR=11, DR_SF8CR=12, DR_SF7CR
19 #if defined(CFG_us915) && defined(LG02_LG01)
20 // CFG_us915 || CFG_as923
21 #define LG02_UPFREQ 902320000
22 #define LG02_DNWFREQ 923300000
23 #define LG02_RXSF 3 // DR_SF7 For LG01/LG02 Tx
24 #define LG02_TXSF 8 // DR_SF12CR For LG02/LG02 Rx
25 #elif defined(CFG_eu868) && defined(LG02_LG01)
26 // CFG_eu868
27 // EU868: DR_SF12=0, DR_SF11=1, DR_SF10=2, DR_SF9=3, DR_SF8=4, DR_SF7=5, DR_SF7B=1, DR_FSK, DR_
28 #define LG02_UPFREQ 505300000
29 #define LG02_DNWFREQ 505300000
30 #define LG02_RXSF 0 // DR_SF7 For LG01/LG02 Tx
31 #define LG02_TXSF 0 // DR_SF12 For LG02/LG02 Rx
32 #endif
33
```

## 7 Technical Support

- Support is provided Monday to Friday, from 09:00 to 18:00 GMT+8. Due to different timezones we cannot offer live support. However, your questions will be answered as soon as possible in the before-mentioned schedule.
- Provide as much information as possible regarding your enquiry (product models, accurately describe your problem and steps to replicate it etc) and send a mail to

[support@dragino.com](mailto:support@dragino.com)

## 8 Reference

- 1) [LoRaWAN official website. And Technical document for LoRaWAN.](#)
- 2) [LG01-N LoRa Gateway User Manual](#)
- 3) [LoRa Low Energy design guide](#) and [Calculator Tool](#).
- 4) About Distance: [LoRa Modem Design Guide](#)
- 5) [SX1276 download resource](#).
- 6) User Manual: [LG01-N](#), [LoRa Shield](#), [LoRa/GPS Shield](#)