



Single Channel LoRa IoT Kit v2 User Manual

Document Version: 1.0

Version	Description	Date
1.0	Release	2019-Jan-28

Index:

1	Overview	4
1.1	What is Dragino Single Channel LoRa IoT Kit v2?	4
1.2	What can you learn from the kit?.....	4
1.3	What parts Dragino LoRa IoT v2 includes?.....	5
2	Preparing.....	6
2.1	Software for End Node	6
2.1.1	Install Arduino IDE and CH340 driver.....	6
2.1.2	Install LoRa Library for Arduino.....	7
2.2	Prepare for LG01-N Gateway.....	8
2.2.1	Configure LG01-N for internet connection	8
2.2.2	Download putty tool to access LG01-N via SSH	11
3	Example 1: Test a LoRaWAN network	12
3.1	Typology and Data Flow	13
3.2	Create a gateway in TTN Server.....	14
3.3	Configure LG01-N Gateway	16
3.3.1	Configure to connect to LoRaWAN server	16
3.3.2	Configure LG01-N's LoRa Radio frequency	17
3.4	Create LoRa Shield End Node	18
3.4.1	Hardware Connection	18
3.4.2	Set up OTAA device in TTN and upload sketch to UNO	18
3.4.3	Configure to connect to Cayenne Application Server	22
3.4.4	Use downlink message to control relay	25
3.4.5	Test with Interrupt	27
3.5	Create LoRa/GPS Shield End Node	28
3.5.1	Hardware connection.....	28
3.5.2	Set up ABP device in TTN and upload software to UNO.....	28
3.6	Conclusion and limitation.....	32
3.6.1	Overview for the example.....	32
3.6.2	Limitations.....	34
4	Example 2: Test with a MQTT IoT Server	35
4.1	Typology and Data Flow	35
4.2	Set up sensor channels in ThingSpeak	36
4.3	Simulate MQTT uplink via PC's MQTT tool.....	38
4.4	Try MQTT Publish with LG01-N Linux command	39
4.5	Configure LG01-N Gateway	40
4.5.1	Publish Logic.....	40
4.5.2	Configure LG01-N's Radio frequency	41
4.6	Create LoRa Shield End Node	42
4.6.1	Hardware Connection	42
4.6.2	Test with uplink	43
4.6.3	Test with interrupt by flame detect	44

4.7	Conclusion and limitation.....	45
4.7.1	Overview for the example.....	45
5	Order Info.....	46
6	FAQ & Trouble Shooting.....	47
6.1	I can't upload sketch to LoRa Shield in MAC OS, shows "dev/cu.usbmodem1421 is not available"	47
6.2	My IoT Kit has the model LG01-P instead of LG01-N, Can I still use this manual.	47
7	Technical Support.....	48
8	Reference	49

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.

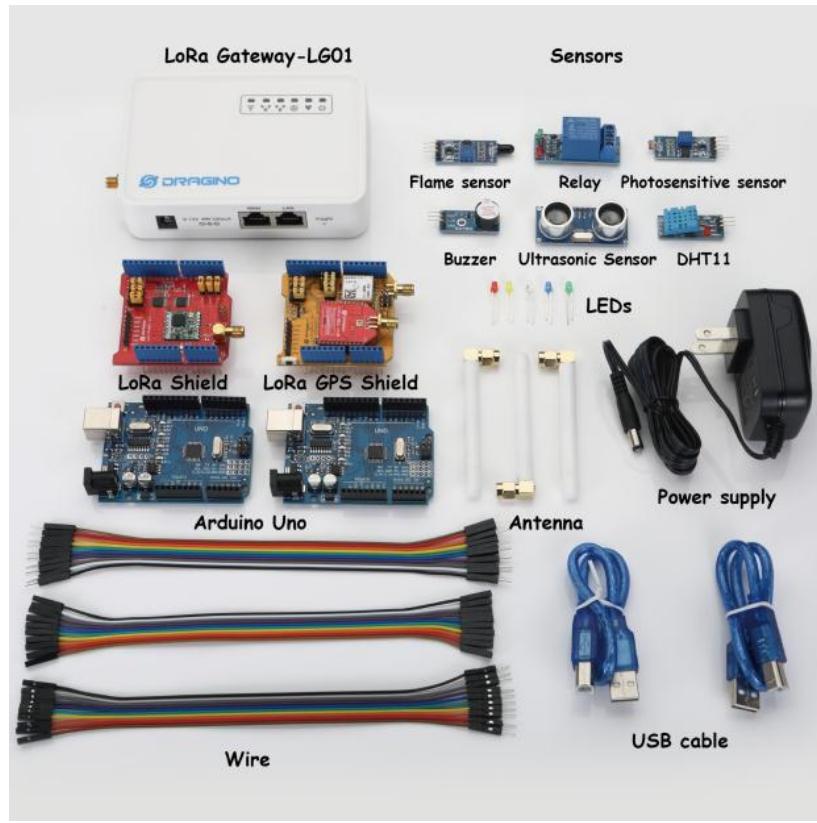
The LoRa IoT kit v2 shows how to build a LoRa network, and how to use the network to send data from a LoRa sensor node to the cloud server. Depends on the actually use environment, the LoRa gateway will connect your other LoRa nodes up to 500 ~ 5,000 meters.

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.

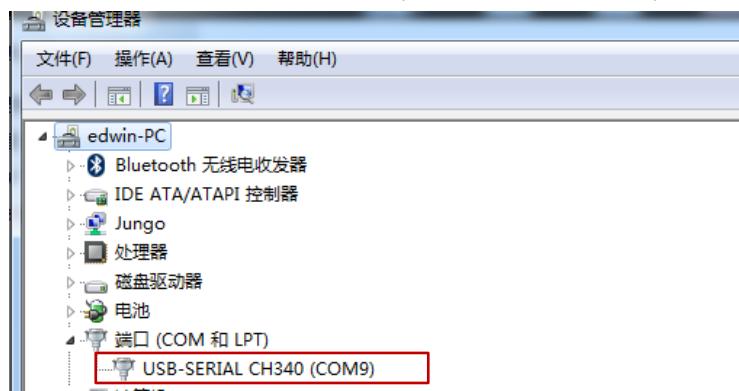
We need to program the Arduino UNO during our testing to support the required functions for end nodes. To finish this, we need to install some software and library first.

2.1 Software for End Node

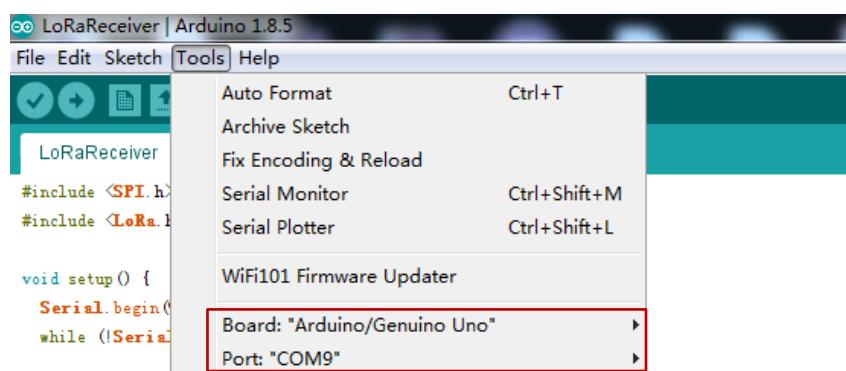
2.1.1 Install Arduino IDE and CH340 driver

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

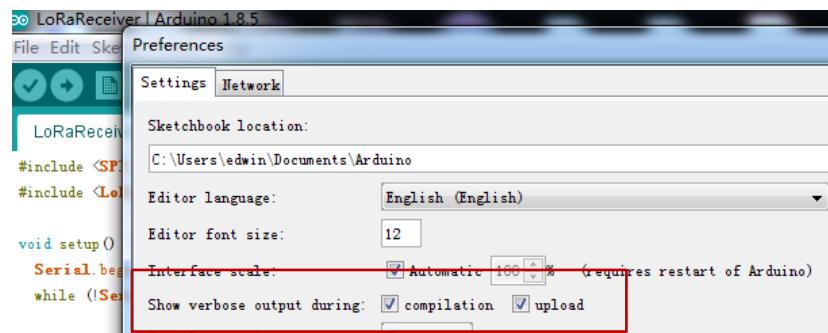
The Arduino UNO in the kit is clone version and is equipped with CH340 USB to UART chip. We need to install CH340 driver in the PC to let the Arduino IDE program it via USB. If we successful install the driver, a com port will show in the system device manager:



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

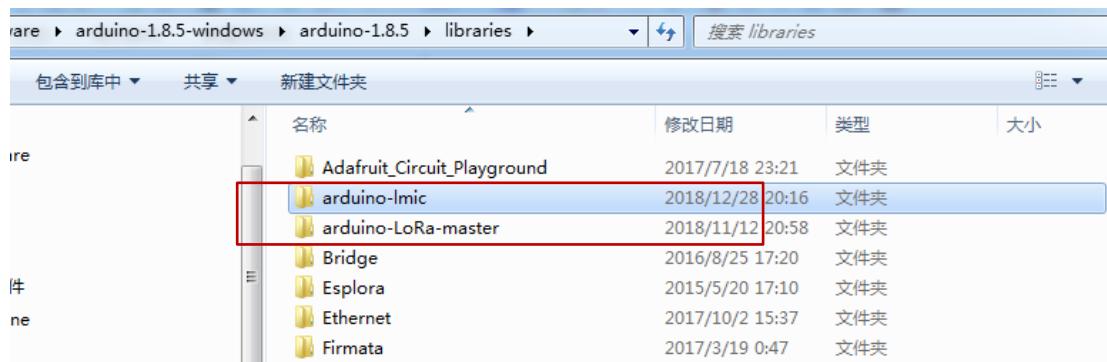
In our examples, we will use two different LoRa libraries for End Node to build different type of LoRa network. They are:

- [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

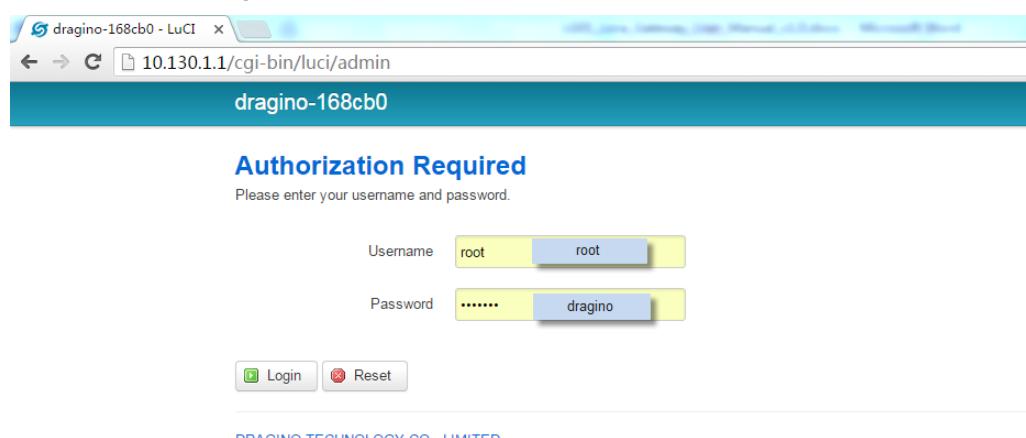
<http://10.130.1.1/cgi-bin/luci/admin>

User will see the login interface of LG01-N.

The account for Web Login is:

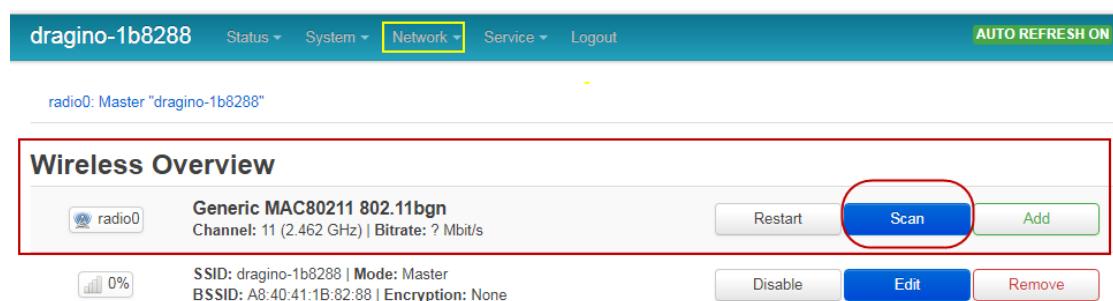
User Name: root

Password: dragino



Step3:

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



Step4:

Select the wireless AP and join the wifi network:

dragino-1b8288	Status	System	Network	Service	Logout	AUTO REFRESH ON
----------------	--------	--------	---------	---------	--------	-----------------

Join Network: Wireless Scan

Signal	SSID	Channel	Mode	BSSID	Encryption	
100%	dragino-office	8	Master	50:64:2B:1A:B8:4D	mixed WPA/WPA2 - PSK	Join Network
84%	ChinaNet-gLnB	2	Master	A4:29:40:66:F4:E7	mixed WPA/WPA2 - PSK	Join Network

dragino-1b8288	Status	System	Network	Service	Logout
----------------	--------	--------	---------	---------	--------

Joining Network: "dragino-office"

Replace wireless configuration

Check this option to delete the existing networks from this radio.

WPA passphrase * *

Specify the secret encryption key here.

Name of the new network

The allowed characters are: A-Z, a-z, 0-9 and _

Create / Assign firewall-zone ?"/>

Choose the firewall zone you want to assign to this interface. Select *unspecified* to remove the interface from the associated zone or fill out the *create* field to define a new zone and attach the interface to it.

[Back to scan results](#) Submit

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.

dragino-1b8288	Status	System	Network	Service	Logout	UNSAVED CHANGES: 13 AUTO REFRESH ON
----------------	--------	--------	---------	---------	--------	-------------------------------------

radio0: Master "dragino-1b8288"

Wireless Overview

 radio0	Generic MAC80211 802.11bgn Channel: 11 (2.462 GHz) Bitrate: ? Mbit/s	Restart Scan Add
 0%	SSID: dragino-1b8288 Mode: Master BSSID: A8:40:41:1B:82:88 Encryption: None	Disable Edit Remove
 0%	SSID: dragino-office Mode: Client BSSID: 50:64:2B:1A:B8:4D Encryption: -	Disable Edit Remove

Associated Stations

Network	MAC-Address	Host	Signal / Noise	RX Rate / TX Rate
No information available				

(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.

dragino-1b8288 Status ▾ System ▾ Network ▾ Service ▾ Logout **AUTO REFRESH ON**

WAN WWAN LAN

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	Restart Stop Edit Delete
WAN  eth1	Protocol: DHCP client MAC: A8:40:41:1B:82:8A RX: 4.30 MB (51840 Pkts.) TX: 55.77 KB (429 Pkts.)	Restart Stop Edit Delete
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	Restart Stop Edit Delete

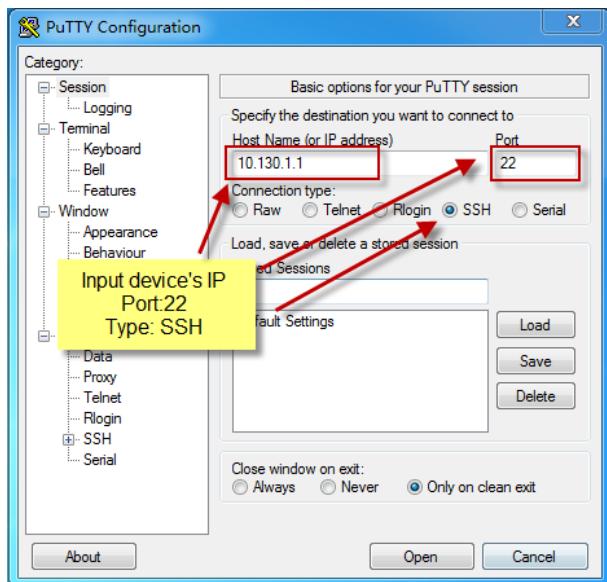
[Add new interface...](#)

Save & Apply [Save](#) [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:



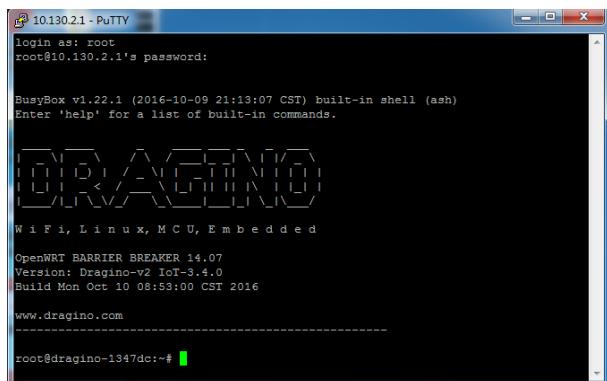
IP address: IP address of LG01-N

Port: 22

User Name: root

Password: dragino (default)

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



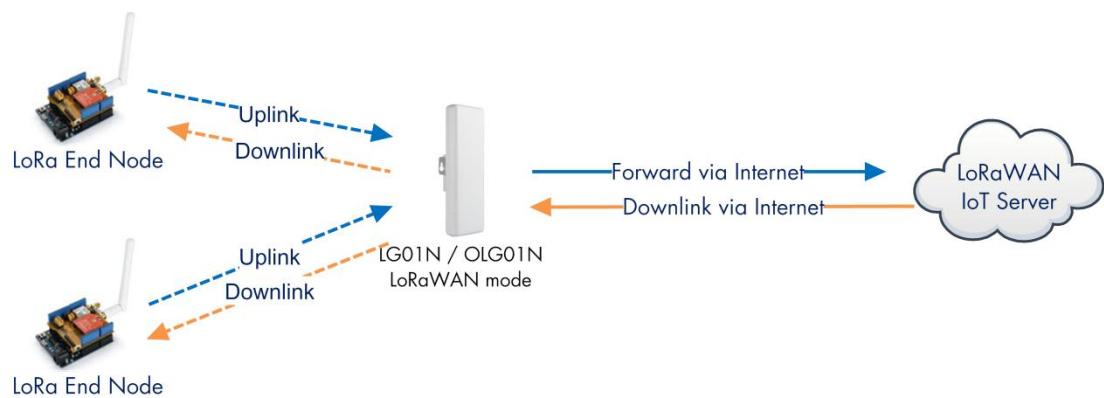
So we have prepare what we need and let's go for the examples!

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.

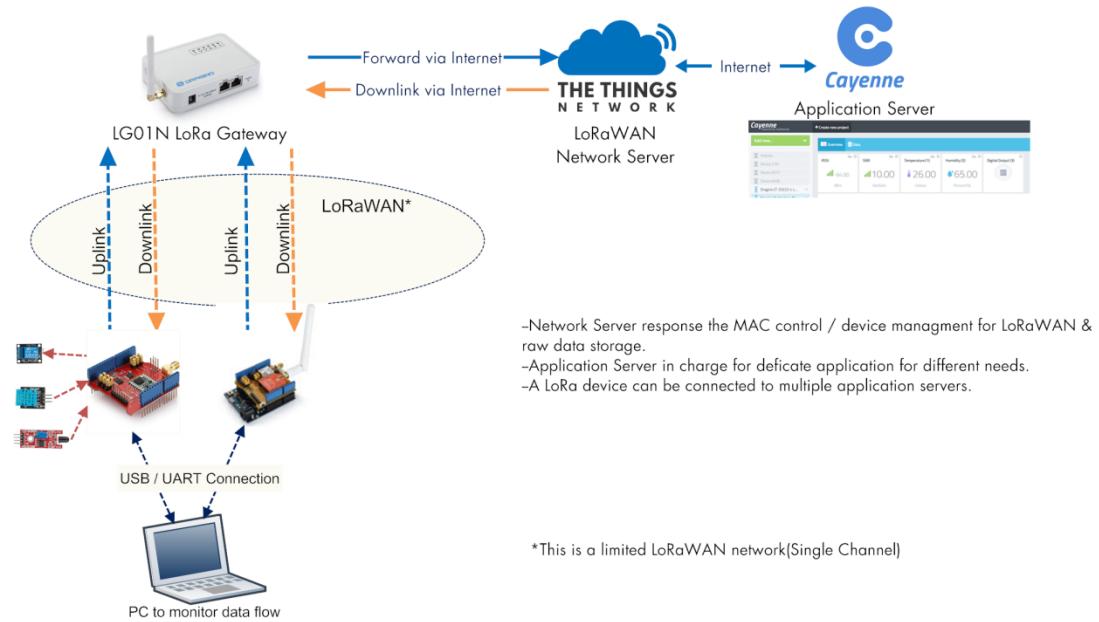
Limitation:

- 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:

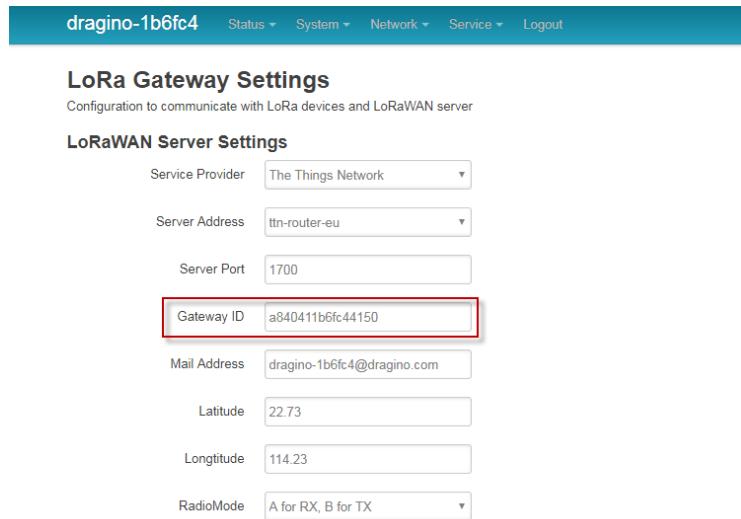


In next section we will start to configure for this example.

3.2 Create a gateway in TTN Server

Step 1: Get a Unique gateway ID.

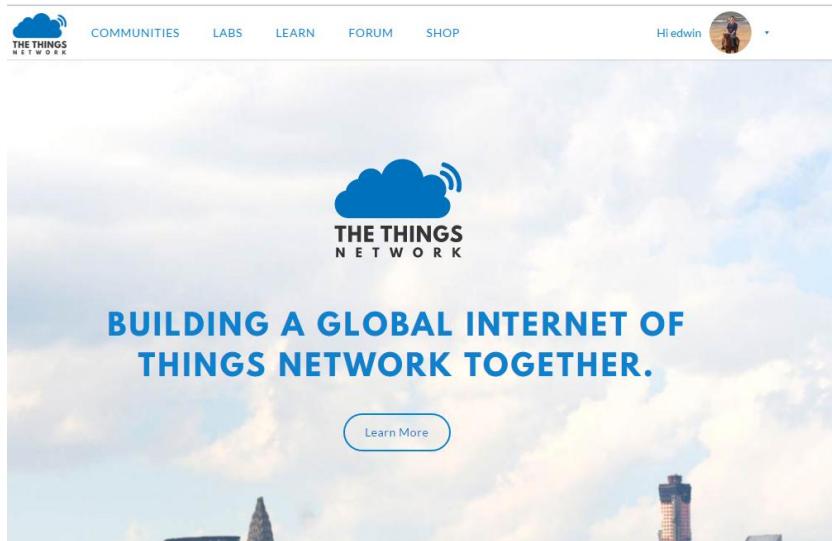
Every LG01-N has a unique gateway id. The id can be found at LoRaWAN page:



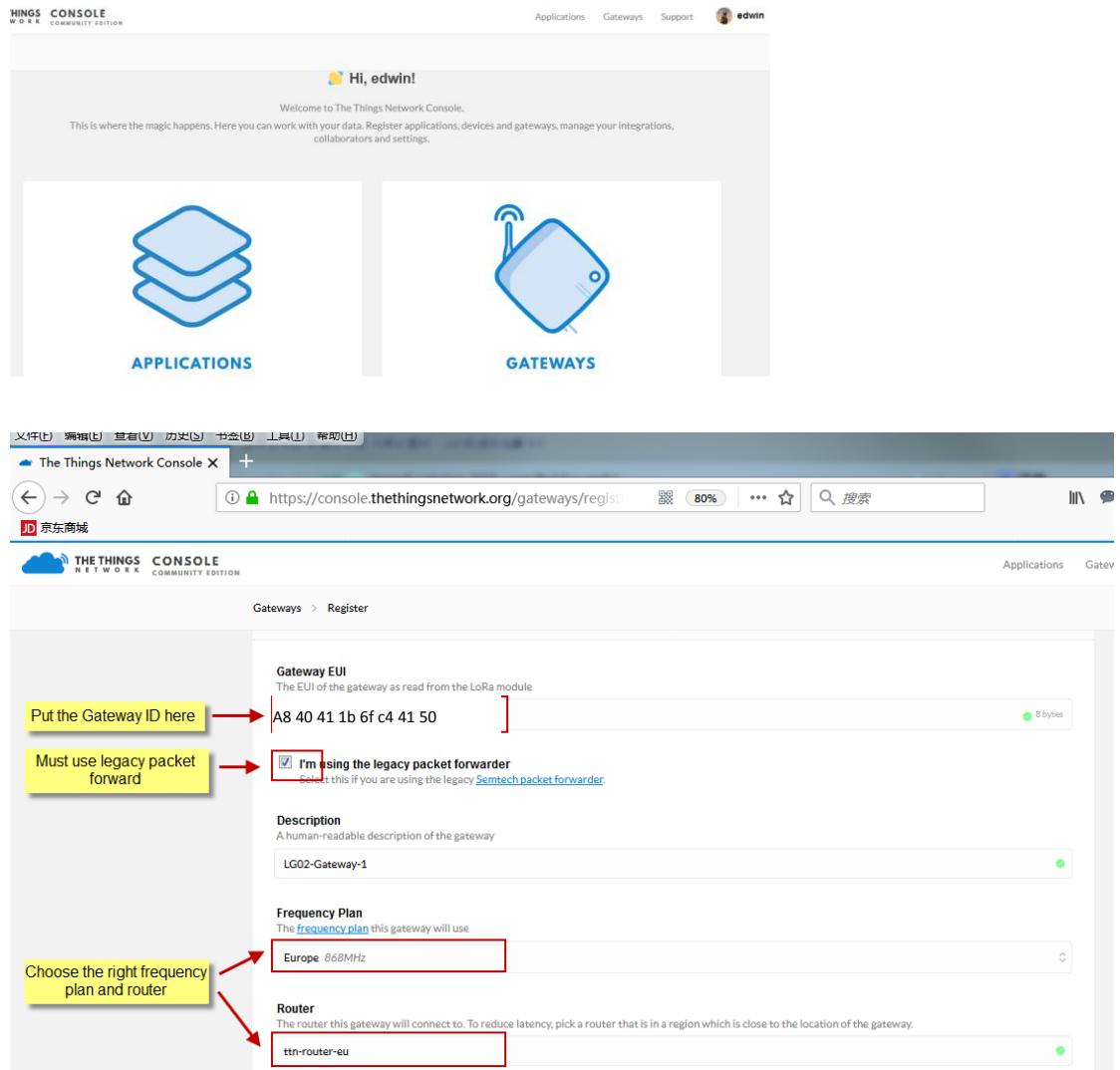
The screenshot shows the 'LoRa Gateway Settings' configuration page. It includes fields for Service Provider (The Things Network), Server Address (ttn-router-eu), Server Port (1700), Gateway ID (a840411b6fc44150), Mail Address (dragino-1b6fc4@dragino.com), Latitude (22.73), Longitude (114.23), and RadioMode (A for RX, B for TX). The 'Gateway ID' field is highlighted with a red border.

The gateway id is: **a840411b6fc44150**

Step 2: Sign up a user account in TTN server



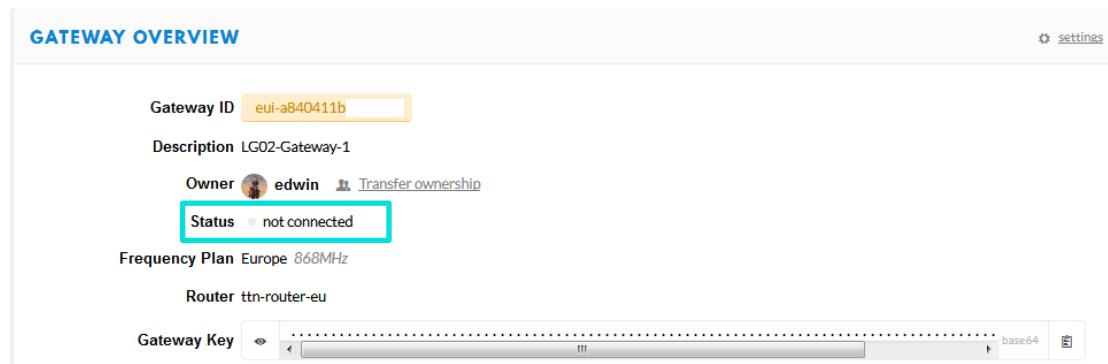
Step 3: Create a Gateway in TTN



The screenshot shows the 'Register' gateway creation page in The Things Network Console. Key fields highlighted include:

- Gateway ID:** A8 40 41 1b 6f c4 41 50
- Frequency Plan:** Europe 868MHz
- Router:** ttn-router-eu
- Legacy Forwarder:** I'm using the legacy packet forwarder (checkbox selected)

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.



GATEWAY OVERVIEW	
Gateway ID	eui-a840411b
Description	LG02-Gateway-1
Owner	edwin
Status	not connected
Frequency Plan	Europe 868MHz
Router	ttn-router-eu
Gateway Key

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



dragino-1893c4 Status System Network Service Logout

Single Channel LoRa Gateway
Configuration to communicate with LoRa devices and LoRaWAN server

LoRaWAN Server Settings

IoT Service	LoRaWan/Raw forwarder
Debug Level	Little message output

Step2: Input server info and gateway id

Choose the correct the server address and gateway ID.



dragino-1b8288 Status System Network Service Logout

LoRa Gateway Settings
Configuration to communicate with LoRa devices and LoRaWAN server

LoRaWAN Server Settings

Service Provider	The Things Network
Server Address	ttn-router-eu
Server Port	1700
Gateway ID	a840411b
Mail Address	edwin@dragino.com
Latitude	22.73
Longitude	114.23

Check Result

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



GATEWAY OVERVIEW

Gateway ID	eui-a840411b
Description	L Office-868
Owner	edwin
Status	connected
Frequency Plan	Europe 868MHz
Router	ttn-router-eu

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

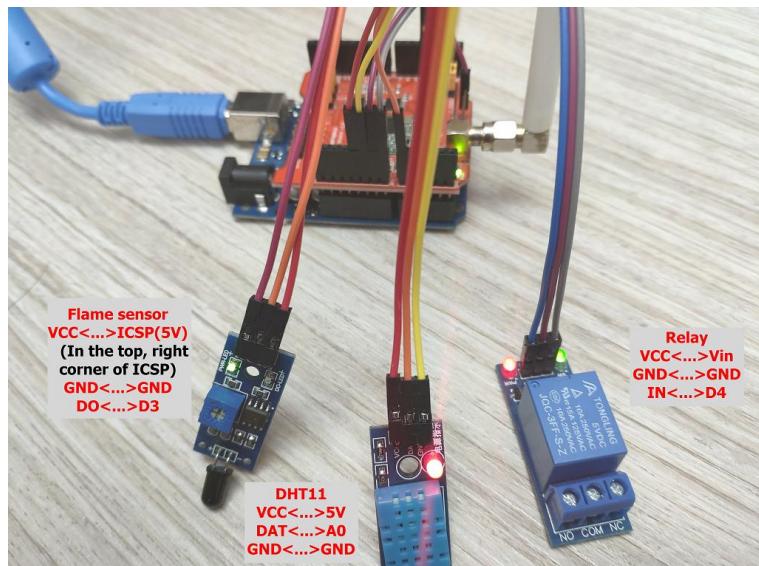
Radio settings for Channel

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



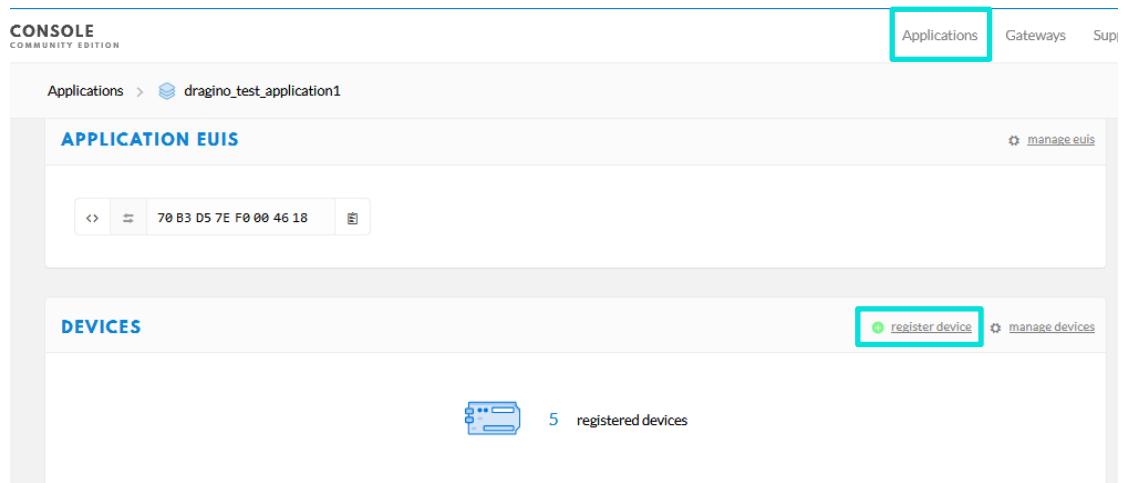
There are three sensors connect to the LoRa Shield + UNO. These sensors are flame sensors, DHT11 (Temperature & Humidity sensor) and Relay. Please use the connection as we show in the photo.

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.



The screenshot shows the TTN Application page with the following details:

- APPLICATION EUIS:** Shows the EUIS of the device: 70 B3 D5 7E F0 00 46 18.
- DEVICES:** Shows 5 registered devices. There is a button labeled "register device" highlighted with a red box.

dragino_test_application1 > Devices > otaa-device-1 > Settings

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

A8 40 41 12 34 56 78 90	8 bytes
-------------------------	---------

Application EUI

70 B3 D5 7E 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_SF8C=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
    LG02_DNWFreq: End Device Uplink (transmit) SF
    LG02_RXSF: End Device Uplink (transmit) SF
    LG02_TXSF: End Device Downlink (receive) SF
    The TXSF is now set to default value:
    US915/AS923 : 923300000 , SF12BW500
    EU868: 869525000, SF12BW125
#endif

//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.

Application ID: dragino_test_application1
 Device ID: otaa-device-1
 Activation Method: OTAA

Device EUI: { 0x90, 0x78, 0x56, 0x34, 0x12, 0x41, 0x40, 0xA8 }
Application EUI: { 0x18, 0x46, 0x00, 0xF0, 0x7E, 0xD5, 0xB3, 0x70 }
App Key: { 0xC3, 0x95, 0x15, 0x93, 0xAD, 0x55, 0x1A, 0x83, 0x2F, 0x31, 0x25, 0xB6, 0x7A, 0xF5, 0x74, 0x1D }

Device Address: 26 01 2D 5E
 Network Session Key:
 App Session Key:

ttn-otaa

```
#include "SPT.h"

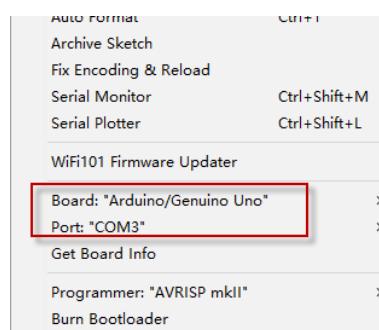
// This EUI must be in little-endian format, so least-significant-byte
// first. When copying an EUI from ttncctl output, this means to reverse
// the bytes. For TTN issued EUIs the last bytes should be 0xD5, 0xB3,
// 0x70.
static const ul_t PROGMEM APPEUI[8]={ 0x18, 0x46, 0x00, 0xF0, 0x7E, 0xD5, 0xB3, 0x70 };

void os_getArtEui (ul_t* buf) { memcpy_P(buf, APPEUI, 8); }

// This should also be in little endian format, see above.
static const ul_t PROGMEM DEVEUI[8]={ 0x90, 0x78, 0x56, 0x34, 0x12, 0x41, 0x40, 0xA8 };
void os_getDevEui (ul_t* buf) { memcpy_P(buf, DEVEUI, 8); }

// This key should be in big endian format (or, since it is not really a
// number but a block of memory, endianness does not really apply). In
// practice, a key taken from ttncctl can be copied as-is.
// The key shown here is the semtech default key.
static const ul_t PROGMEM APPKEY[16] = { 0xC3, 0x95, 0x15, 0x93, 0xAD, 0x55, 0x1A, 0x83, 0x2F, 0x31, 0x25, 0xB6, 0x7A, 0xF5, 0x74, 0x1D };
void os_getDevKey (ul_t* buf) { memcpy_P(buf, APPKEY, 16); }
```

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).

The screenshot shows the Arduino IDE interface with the TTN-ABP example sketch loaded. The serial monitor window displays the following output:

```
Connect to TTN and Send data to mydevice cayenne(Use DHT11 Sensor):  
RXMODE_RSSI  
##### COUNT=1 #####  
The temperature and humidity:  
[28.00°C, 63.00%]  
6024: engineUpdate, oemode=0x8  
Packet queued  
9013: EV_JOINING  
10182: engineUpdate, oemode=0xc  
196226: engineUpdate, oemode=0xc  
196532: TXMODE, freq=868100000, len=23, SF=7, BW=125, CR=4/5, IH=0  
503011: RXMODE_SINGLE, freq=868100000, SF=7, BW=125, CR=4/5, IH=0, rxsysms=255  
514692: JaccRX1, dataLen=33  
515265: EV_JOINED  
515290: engineUpdate, oemode=0x808  
516157: TXMODE, freq=868100000, len=20, SF=7, BW=125, CR=4/5, IH=0  
570725: RXMODE_SINGLE, freq=868100000, SF=7, BW=125, CR=4/5, IH=0, rxsysms=255
```

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

dragino-1b6fb0		Status	System	Network	Service	Logout
Logread						
FreqINFO	Report	RxTxJson	ErrorMSG	Gateway Log shows TX / RX LoRa Packet		
(TPXK): [down] "txpk": {"ime": "false", "tmst": "3667234979", "freq": "868.1", "rfch": "0", "powe": "14", "modu": "LORA", "datr": "SF7BW125", "codr": "4/5", "ipol": "true", "size": "33", "ncrc": "Receive(HEX):20f67562b6ba74b13d97b2d53841c4a2c3d2b3f5784edac0ee41c9b52aeed37"}, (RXPXK): [up] "txpk": {"ime": "2018-10-19T15:49:50.666162Z", "tmst": "3666685421", "chan": "0", "rfch": "1", "freq": "868.100000", "stat": "1", "modu": "LORA", "datr": "SF7BW125", "codr": "4/5", "ipol": "true", "size": "33", "ncrc": "Receive(HEX):20f67562b6ba74b13d97b2d53841c4a2c3d2b3f5784edac0ee41c9b52aeed37"}, (RXPXK): [up] "rxpk": {"ime": "2018-10-19T15:49:51.103837Z", "tmst": "3667330098", "chan": "0", "rfch": "1", "freq": "868.100000", "stat": "1", "modu": "LORA", "datr": "SF7BW125", "codr": "4/5", "ipol": "true", "size": "33", "ncrc": "Receive(HEX):001846000070ed5b3709078563412414083717b0ba365"}, (RXPXK): [up] "rxpk": {"ime": "2018-10-19T15:51:28.881342Z", "tmst": "3748307397", "chan": "0", "rfch": "1", "freq": "868.100000", "stat": "1", "modu": "LORA", "datr": "SF7BW125", "codr": "4/5", "ipol": "true", "size": "33", "ncrc": "Receive(HEX):20b2875f11263b8be0630173e1bb0303649d809aebef7d2b01acd12a8a1555b3f5"}, (RXPXK): [up] "rxpk": {"ime": "2018-10-19T15:51:16.768714Z", "tmst": "3752787797", "chan": "0", "rfch": "1", "freq": "868.100000", "stat": "1", "modu": "LORA", "datr": "SF7BW125", "codr": "4/5", "ipol": "true", "size": "33", "ncrc": "Receive(HEX):20b2875f11263b8be0630173e1bb0303649d809aebef7d2b01acd12a8a1555b3f5"}, (RXPXK): [up] "rxpk": {"ime": "2018-10-19T15:51:17.419193Z", "tmst": "3753438465", "chan": "0", "rfch": "1", "freq": "868.100000", "stat": "1", "modu": "LORA", "datr": "SF7BW125", "codr": "4/5", "ipol": "true", "size": "33", "ncrc": "Receive(HEX):40b5d2012680000169597679767d2a602097949a4d0a4"}, (RXPXK): [up] "rxpk": {"ime": "2018-10-19T15:51:17.529606Z", "tmst": "3753488666", "chan": "0", "rfch": "1", "freq": "868.100000", "stat": "1", "modu": "LORA", "datr": "SF7BW125", "codr": "4/5", "ipol": "true", "size": "33", "ncrc": "Receive(HEX):40b5d2012680000169597679767d2a602097949a4d0a4"}, (RXPXK): [up] "txpk": {"ime": "2018-10-19T15:52:20.726452Z", "tmst": "3816745715", "chan": "0", "rfch": "1", "freq": "868.100000", "stat": "1", "modu": "LORA", "datr": "SF7BW125", "codr": "4/5", "ipol": "true", "size": "33", "ncrc": "Receive(HEX):40b5d2012680000169597679767d2a602097949a4d0a4"}, (RXPXK): [up] "txpk": {"ime": "2018-10-19T15:53:24.029902Z", "tmst": "3880049163", "chan": "0", "rfch": "1", "freq": "868.100000", "stat": "1", "modu": "LORA", "datr": "SF7BW125", "codr": "4/5", "ipol": "true", "size": "33", "ncrc": "Receive(HEX):40b5d201268000300169597679767d2a602097949a4d0a4"}, (RXPXK): [up] "txpk": {"ime": "2018-10-19T15:54:00.878777Z", "tmst": "3914.41-19T15:54:27.346125Z", "chan": "0", "rfch": "1", "freq": "868.100000", "stat": "1", "modu": "LORA", "datr": "SF7RW1125", "codr": "4/5", "ipol": "true", "size": "33", "ncrc": "Receive(HEX):40b5d201268000300169597679767d2a602097949a4d0a4"}, (RXPXK): [up] "txpk": {"ime": "2018-10-19T15:54:00.878777Z", "tmst": "3914.41-19T15:54:27.346125Z", "chan": "0", "rfch": "1", "freq": "868.100000", "stat": "1", "modu": "LORA", "datr": "SF7RW1125", "codr": "4/5", "ipol": "true", "size": "33", "ncrc": "Receive(HEX):40b5d201268000300169597679767d2a602097949a4d0a4"}]						

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

ONSO IMMUNITY TTN Traffic Page shows the device status

Applications Gateways

Gateways > eui-a840411b6fb04150 > Traffic beta

	Date	Device	Protocol	Rate	SF	BW	DevAddr	payload size
▲	23:56:34	868.1	lora	4/5	SF 7	BW 125	61.7	26 bytes
▲	23:55:30	868.1	lora	4/5	Immediately send a Uplink message after join success			26 bytes
▲	23:54:27	868.1	lora	4/5	SF 7	BW 125	61.7	26 bytes
▲	23:53:24	868.1	lora	4/5	SF 7	BW 125	61.7	26 bytes
▲	23:52:20	868.1	lora	4/5	SF 7	BW 125	61.7	26 bytes
▲	23:51:17	868.1	lora	4/5	SF 7	BW 125	61.7	0 dev addr: 26 01 2F B3 payload size: 26 bytes
✓	23:51:16	868.1		4/5	SF 7	BW 125	71.9	
✗	23:51:12	868.1		4/5	SF 7	BW 125	61.7	app eui: 70 B3 D5 7E F0 00 46 18 dev eui: A8 40 41 12 34 56 78

TTN Send a Join reply. LoRa End node must get this packet to finish Join. The frequency shows use 868.1MHz frequency, must be the same as the "LG02_DNWREQ" in Lmic config.c file

TTN Get Join request

Note: The LG02_DNWREQ value in Arduino_LMIC/src/lmic/config.h should match downlink frequency from TTN.

TTN shows 868.1 here, So LG02_DNWREQ should be 868100000

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

APPLICATION DATA

Filters: uplink, downlink, activation, ack, error

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

91% 

3.4.3 Configure to connect to Cayenne Application Server

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

Step 1: Add Cayenne in Application page

ADD INTEGRATION

Integrations (red box)

Cayenne (highlighted with a red box)

COLLOS collaborative location service

Process ID: yourid

Status: Running

Platform: Cayenne (v2.6.0) [documentation](#)

Author: myDevices

Description: Quickly design, prototype and commercialize IoT solutions with myDevices Cayenne

SETTINGS

Access Key: The access key used for downlink

default key [devices](#) [messages](#)

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

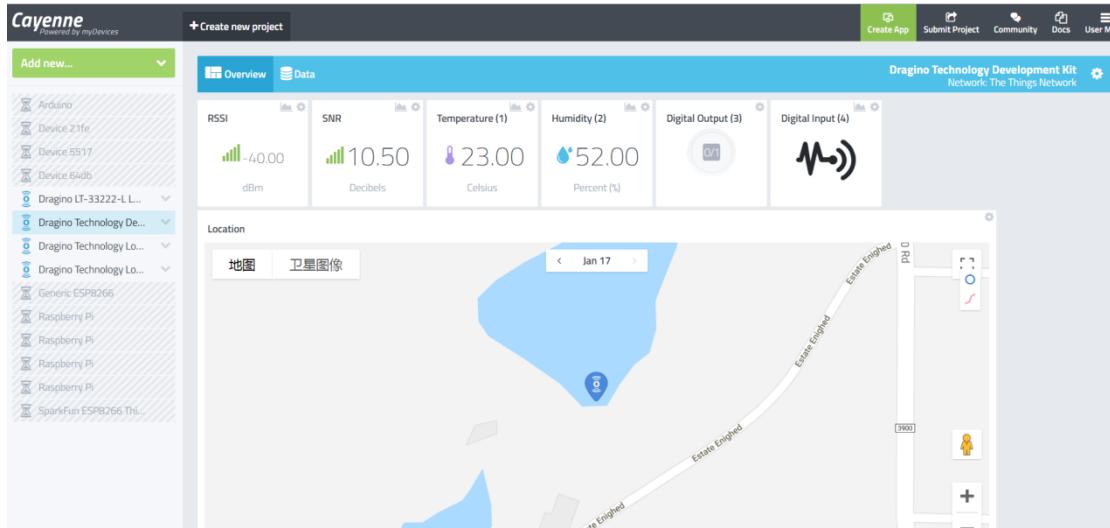
The screenshot shows the Cayenne Device Catalog. On the left, a sidebar menu has 'Device/Widget' selected, indicated by a red arrow labeled '1'. The main area displays a grid of LoRa modules from various manufacturers. A red arrow labeled '2' points to the 'Custom Widgets' section at the bottom right.

The screenshot shows the Cayenne Device Catalog with the search bar set to 'Devices & Widgets' (red arrow '1'). The search results list various device categories. A specific entry, 'Dragino Technology Development Kit', is highlighted with a red arrow labeled '2'.

Add DevEUI of the End node

The screenshot shows the 'Enter Settings' page for the Dragino Technology Development Kit. It includes a preview image, a name field ('Dragino Technology Development Kit'), a DevEUI field ('your DevEUI' with a red arrow '1'), an activation mode dropdown ('Already Registered'), and a tracking section with a location input ('Independence, KS 67301美国') and a green 'Add device' button. Red arrows labeled '2' and '3' point to the location input and the 'Add device' button respectively.

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



3.4.4 Use downlink message to control relay

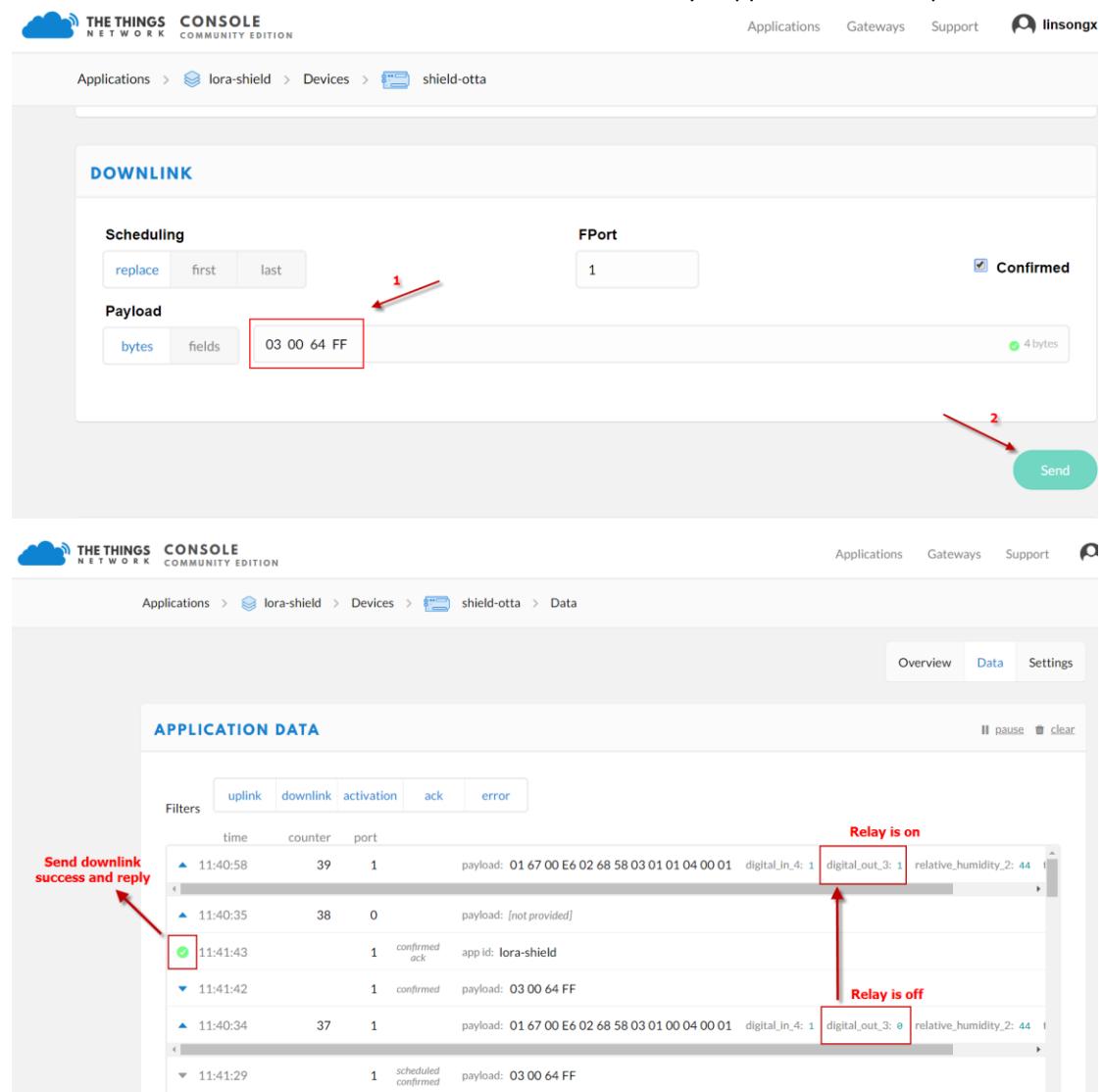
We can use either TTN or Cayenne to control the relay.

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.



The screenshot shows two parts of the The Things Network Console interface.

Top Part: DOWNLINK Configuration

- Scheduling:** Options: replace, first, last. The "replace" button is highlighted with a red box and arrow 1.
- FPort:** Value: 1. Checked: Confirmed.
- Payload:** Type: bytes. Value: 03 00 64 FF. This value is highlighted with a red box and arrow 2.
- Send:** A green rounded rectangle button.

Bottom Part: APPLICATION DATA

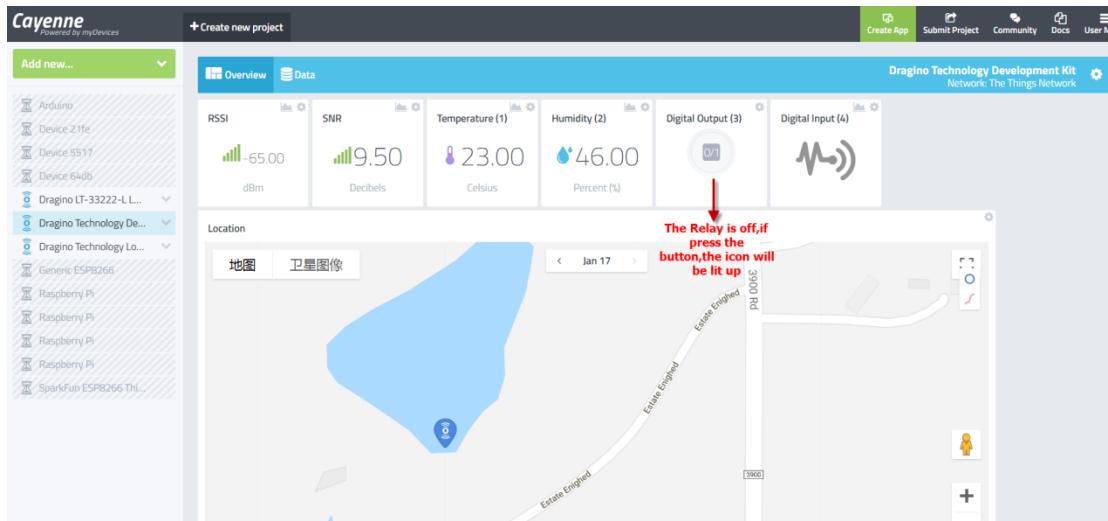
- Filters:** uplink, downlink, activation, ack, error. The "downlink" tab is selected.
- Data List:**
 - 11:40:58: 39, 1, payload: 01 67 00 E6 02 68 58 03 01 01 04 00 01, digital_in_4: 1, digital_out_3: 1, relative_humidity_2: 44. A red box highlights "digital_out_3: 1".
 - 11:40:35: 38, 0, payload: [not provided].
 - 11:41:43: 1, confirmed ack, app id: lora-shield. A red box highlights the green checkmark icon.
 - 11:41:42: 1, confirmed, payload: 03 00 64 FF.
 - 11:40:34: 37, 1, payload: 01 67 00 E6 02 68 58 03 01 00 04 00 01, digital_in_4: 1, digital_out_3: 0, relative_humidity_2: 44. A red box highlights "digital_out_3: 0".
 - 11:41:29: 1, scheduled confirmed, payload: 03 00 64 FF.
- Buttons:** pause, clear, Overview, Data, Settings.

Annotations in the screenshot:

- A red box highlights the "digital_out_3: 1" entry in the application data log.
- A red box highlights the "digital_out_3: 0" entry in the application data log.
- A red box highlights the "digital_out_3: 1" entry in the downlink payload field.
- A red box highlights the green checkmark icon in the application data log.
- A red arrow labeled "Send downlink success and reply" points to the green "Send" button in the downlink configuration.
- A red arrow labeled "Relay is on" points to the "digital_out_3: 1" entry in the application data log.
- A red arrow labeled "Relay is off" points to the "digital_out_3: 0" entry in the application data log.

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:

```

lora_shield_cayenne_and_ttn_abpClient | Arduino 1.8.5
文件 编辑 项目 工具 帮助
lora_shield_cayenne_and_ttn_abpClient
49 static const u1_t PROGMEM APPSKEY
50
51 // LoRaWAN end-device address (De
52 static const u4_t DEVADDR = 0x260
53
54 // These callbacks are only used
55 // left empty here (we cannot lea
56 // DISABLE_JOIN is set in config
57 void os_getArtEui (u1_t* buf) {
58 void os_getDevEui (u1_t* buf) {
59 void os_getDevKey (u1_t* buf) {
60

[25.00°C, 66.00%]
3955087: engineUpdate, opmode=0x908
3957856: TXMODE, freq=868100000, len=23, SF=7, BW=125, CR=4/5
Packet queued
4012411: RXMODE_SINGLE, freq=868100000, SF=7, BW=125, CR=4/5
4022496: Received downlink, window=RX1, port=1, ack=0
4022541: EV_TXCOMPLETE (includes waiting for RX windows)
Received :
3 0 64 FF
Set pin to HIGH.

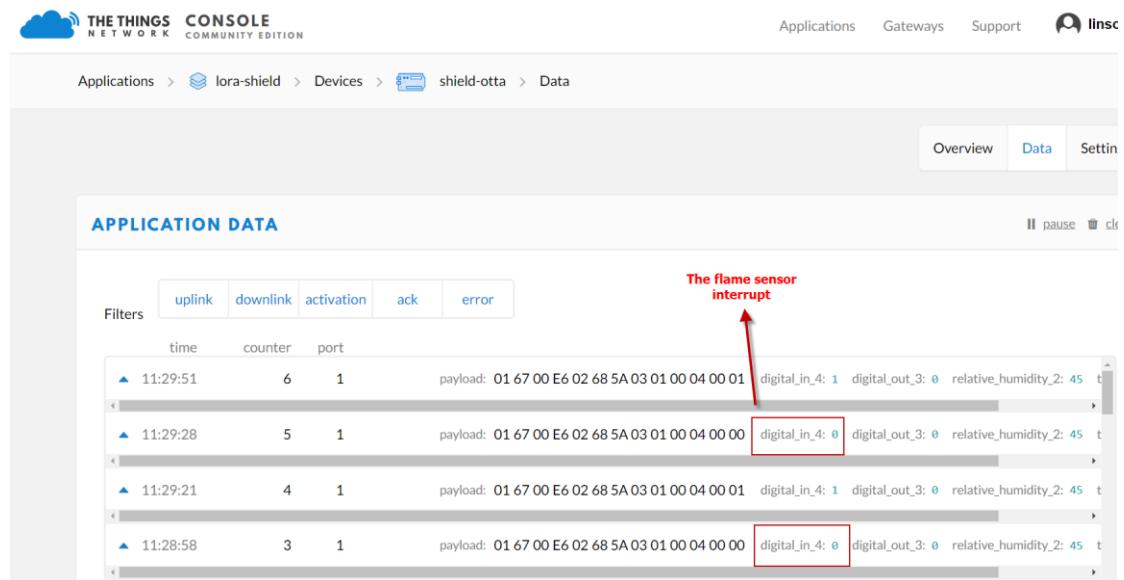
4028292: engineUpdate, opmode=0x810
4030903: TXMODE, freq=868100000, len=12, SF=7, BW=125, CR=4/5
4085487: RXMODE_SINGLE, freq=868100000, SF=7, BW=125, CR=4/5
4124275: RXMODE_SINGLE, freq=869525000, SF=9, BW=125, CR=4/5
4188702: EV_TXCOMPLETE (includes waiting for RX windows)
4188747: engineUpdate, opmode=0x900

```

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.

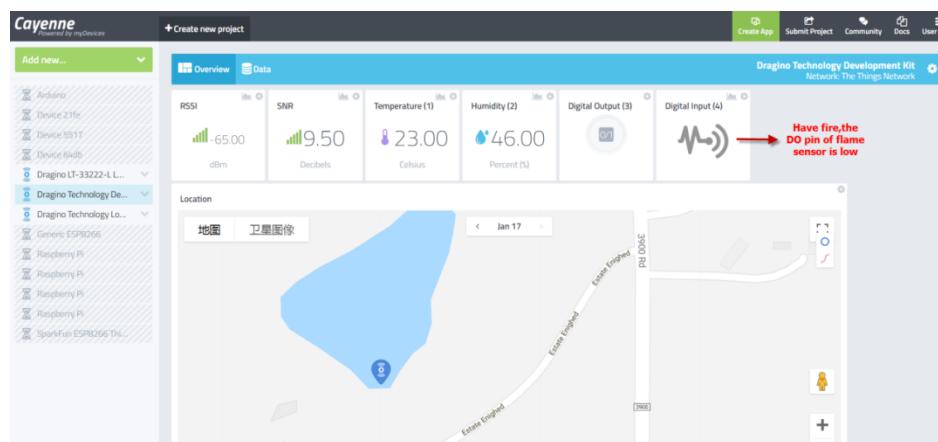
The DO pin of Flame sensor is high in normal case. While it detects a flame, this pin will become low and act as an external interrupt for Arduino. The Arduino UNO will then immediately upload the temperature and humidity to TTN



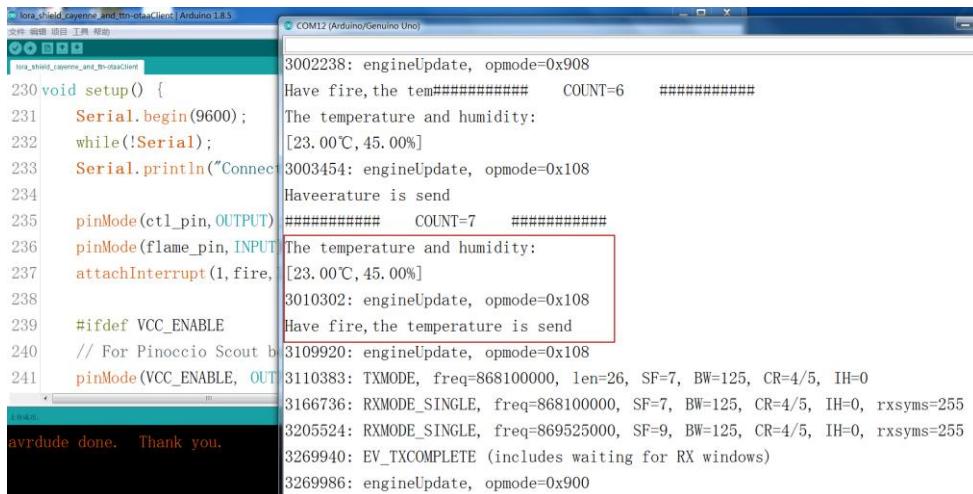
The screenshot shows the 'Data' tab in the The Things Network Console. It displays a list of uplink messages. The fourth message from the top is highlighted with a red box around the 'digital_in_4: 0' value. A red arrow points from the text 'The flame sensor interrupt' to this value. The message details are as follows:

time	counter	port	payload	digital_in_4	digital_out_3	relative_humidity_2
11:29:51	6	1	01 67 00 E6 02 68 5A 03 01 00 04 00 01	1	0	45
11:29:28	5	1	01 67 00 E6 02 68 5A 03 01 00 04 00 00	0	0	45
11:29:21	4	1	01 67 00 E6 02 68 5A 03 01 00 04 00 01	1	0	45
11:28:58	3	1	01 67 00 E6 02 68 5A 03 01 00 04 00 00	0	0	45

Then we can see on the cayenne:



The screenshot shows the Cayenne IoT platform interface. On the right, there's a summary card for 'Dragino Technology Development Kit'. The 'Digital Input (4)' section shows a status icon with a red arrow pointing to the text 'Have fire.the DO pin of flame sensor is low'. On the left, a sidebar lists various device types, and the main area shows a map with a blue location marker.



```

230 void setup() {
231     Serial.begin(9600);
232     while(!Serial);
233     Serial.println("Connect");
234
235     pinMode(ctl_pin, OUTPUT);
236     pinMode(flame_pin, INPUT);
237     attachInterrupt(1, fire, RISING);
238
239     #ifdef VCC_ENABLE
240     // For Pinoccio Scout board
241     pinMode(VCC_ENABLE, OUTPUT);
242     digitalWrite(VCC_ENABLE, HIGH);
243
244     avrdude done. Thank you.

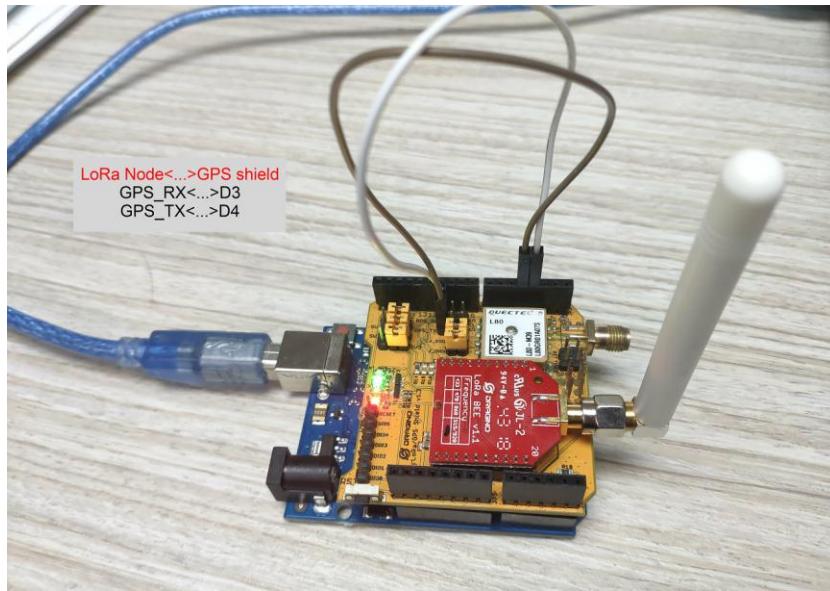
```

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, rxsysms=255
3205524: RXMODE_SINGLE, freq=869525000, SF=9, BW=125, CR=4/5, IH=0, rxsysms=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.

Applications > dragino_test_application1

APPLICATION 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 B3 D5 7E 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

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

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, !!! }

ttn-abp

Arduino Sketch ttn-abp

Input the keys from TTN

```

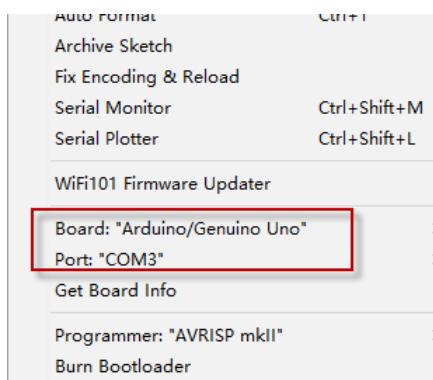
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>

// 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, 0x0D, 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, 0x85, 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!
    
```

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



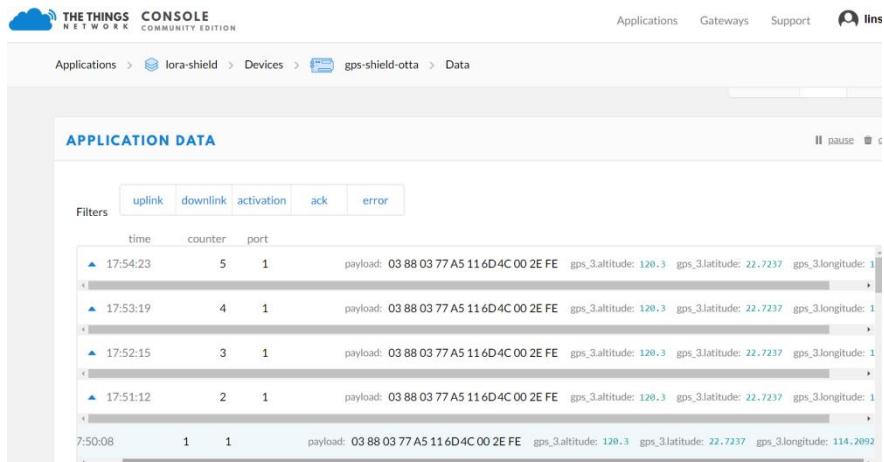
All other steps are similar with how we use with LoRa Shield.

Below are the outputs for reference:

Output from LoRa GPS Shield:

The screenshot shows the Arduino Serial Monitor window titled 'COM12 (Arduino/Genuino Uno)'. The window displays a series of log messages from the LoRa GPS Shield. A red box highlights the message 'The longitude and latitude and altitude are: [114.21, 22.72, 120.30]'. Other messages include engineUpdate, TXMODE, RXMODE_SINGLE, and EV_TXCOMPLETE events. The final message at the bottom is 'avrduke done. Thank you.'

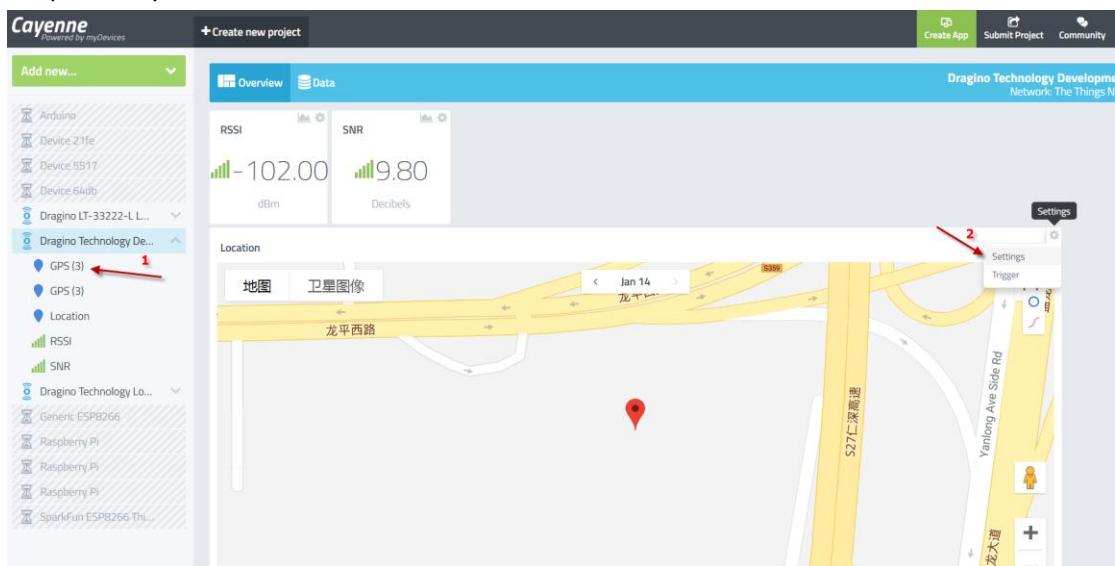
Upload GPS data to TTN:



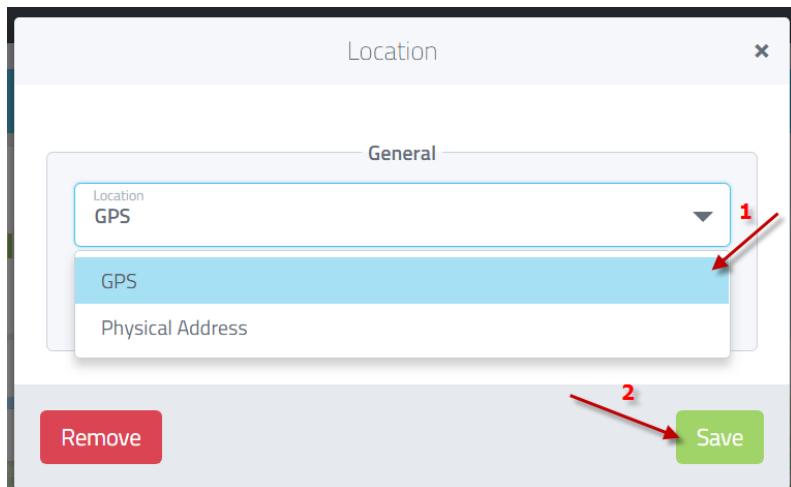
The screenshot shows the 'Data' section of the The Things Network Console for a device named 'gps-shield-otta'. The 'APPLICATION DATA' table lists several uplink messages. Each message includes a timestamp, counter, port, payload, and GPS coordinates (latitude and longitude). The last message listed is at 7:50:08.

time	counter	port	payload	gps_3.latitude	gps_3.longitude
17:54:23	5	1	03 88 03 77 A5 11 6D4C 00 2E FE	120.3	22.7237
17:53:19	4	1	03 88 03 77 A5 11 6D4C 00 2E FE	120.3	22.7237
17:52:15	3	1	03 88 03 77 A5 11 6D4C 00 2E FE	120.3	22.7237
17:51:12	2	1	03 88 03 77 A5 11 6D4C 00 2E FE	120.3	22.7237
7:50:08	1	1	03 88 03 77 A5 11 6D4C 00 2E FE	120.3	22.7237

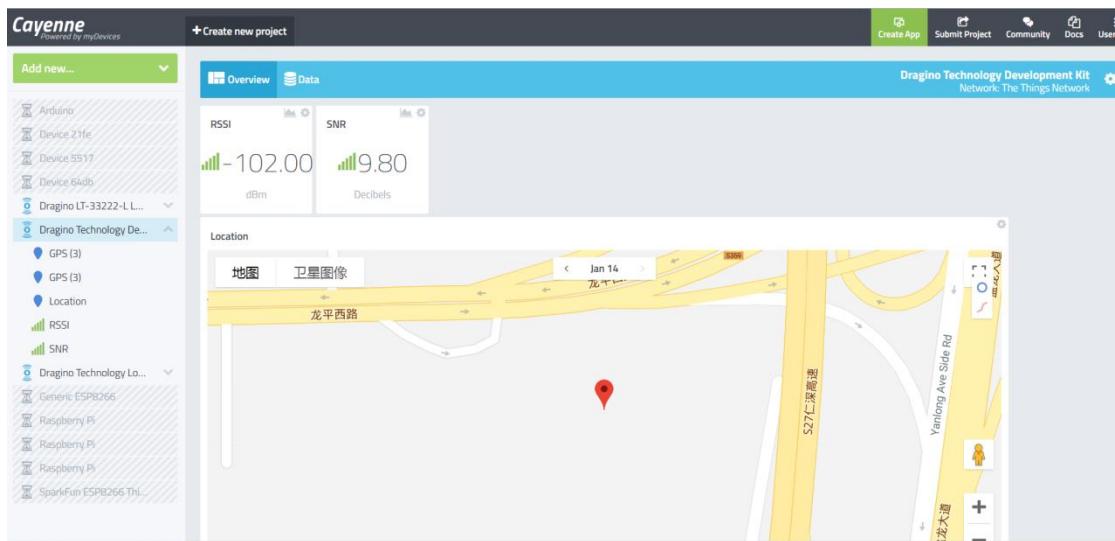
Output in Cayenne:



The Cayenne interface shows a map of a road with a red pin indicating the device's location. On the left sidebar, under 'Dragino Technology De...', there are two items labeled 'GPS (3)' with red arrow 1 pointing to them. On the right, a 'Settings' button is highlighted with red arrow 2.



A detailed view of the 'Location' configuration dialog in Cayenne. The 'General' tab is selected. Under 'Location', 'GPS' is chosen from a dropdown menu (red arrow 1). At the bottom right, a green 'Save' button is highlighted with red arrow 2.



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 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. And there are limitations:

1/ It works only on one channel 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.

2/ LoRaWAN compatible issue

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

3/ 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.

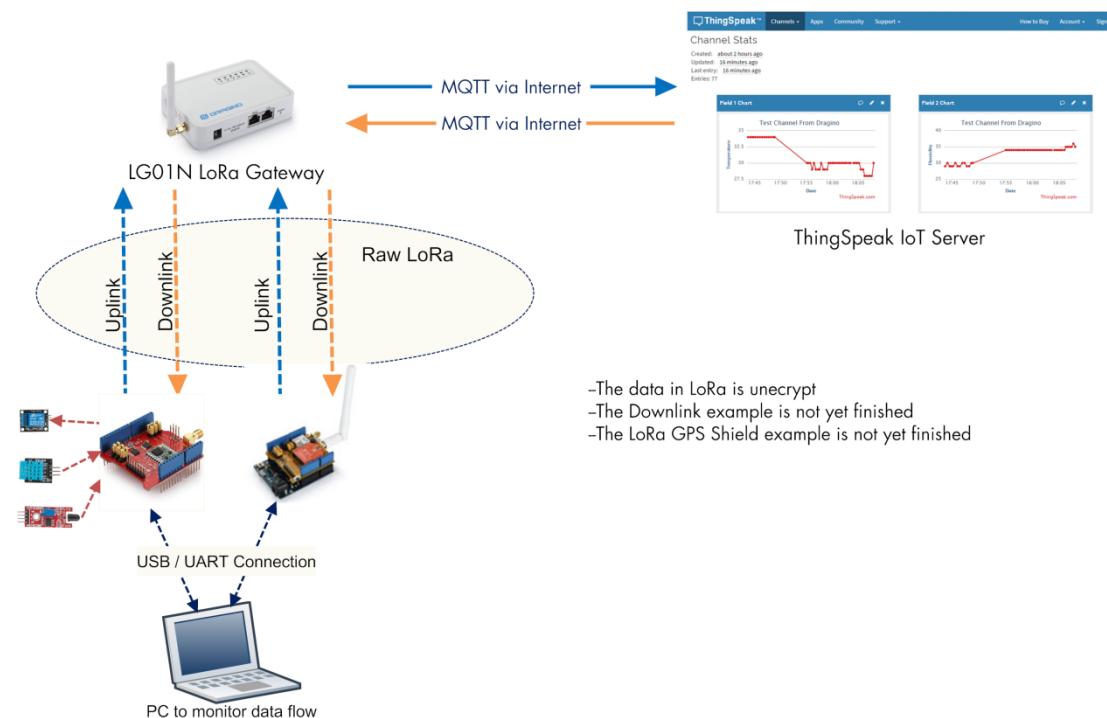
4 Example 2: Test with a MQTT IoT Server

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](#).

4.1 Typology and Data Flow

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

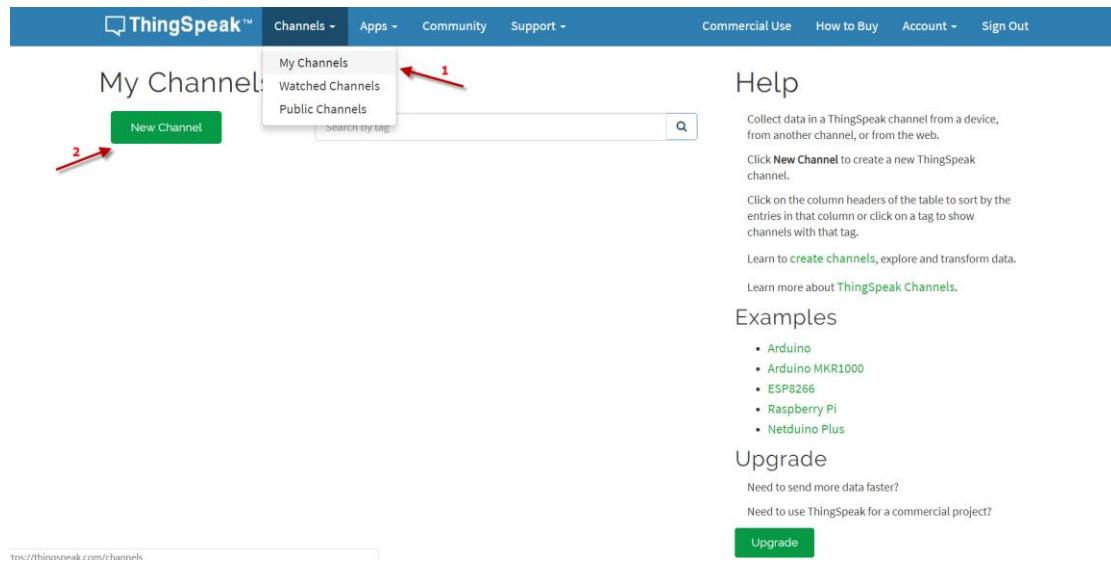
Topology for ThingSpeak Connection:



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



Help

Collect data in a ThingSpeak channel from a device, from another channel, or from the web.

Click **New Channel** to create a new ThingSpeak channel.

Click on the column headers of the table to sort by the entries in that column or click on a tag to show channels with that tag.

Learn to [create channels](#), explore and transform data.

Learn more about [ThingSpeak Channels](#).

Examples

- Arduino
- Arduino MKR1000
- ESP8266
- Raspberry Pi
- Netduino Plus

Upgrade

Need to send more data faster?
Need to use ThingSpeak for a commercial project?

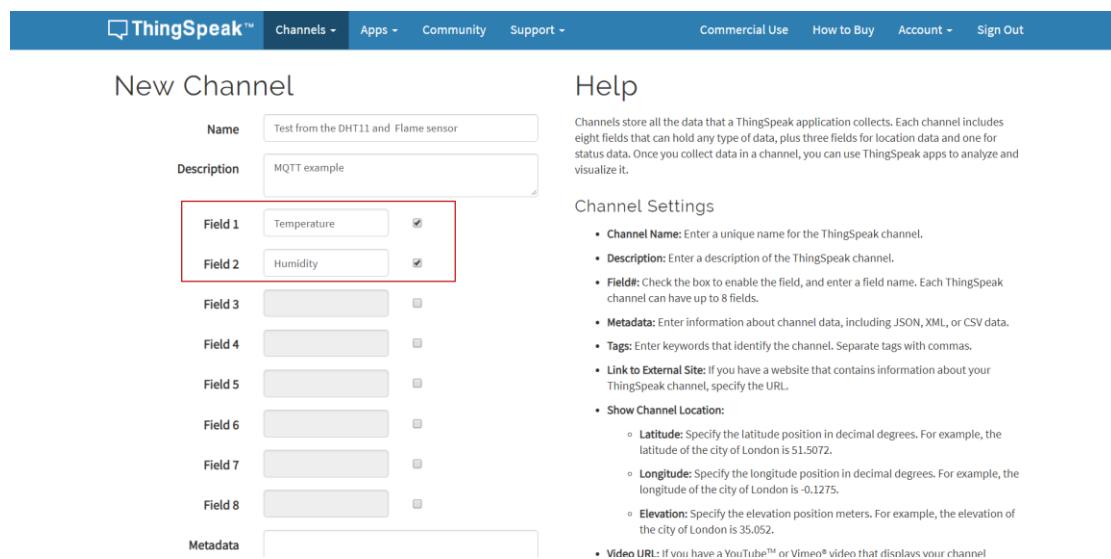
[Upgrade](#)

http://thingspeak.com/channels

Set up two channels:

Field 1: Temperature

Field 2: Humidity



New Channel

Name: Test from the DHT11 and Flame sensor

Description: MQTT example

Field 1	Temperature	<input checked="" type="checkbox"/>
Field 2	Humidity	<input checked="" type="checkbox"/>
Field 3		<input type="checkbox"/>
Field 4		<input type="checkbox"/>
Field 5		<input type="checkbox"/>
Field 6		<input type="checkbox"/>
Field 7		<input type="checkbox"/>
Field 8		<input type="checkbox"/>
Metadata		

Help

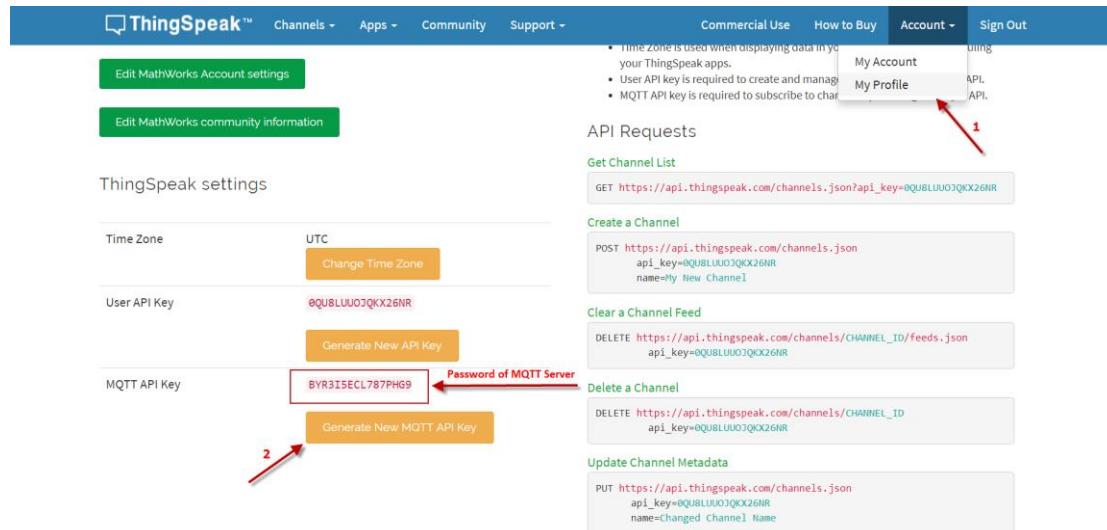
Channels store all the data that a ThingSpeak application collects. Each channel includes eight fields that can hold any type of data, plus three fields for location data and one for status data. Once you collect data in a channel, you can use ThingSpeak apps to analyze and visualize it.

Channel Settings

- **Channel Name:** Enter a unique name for the ThingSpeak channel.
- **Description:** Enter a description of the ThingSpeak channel.
- **Field:** Check the box to enable the field, and enter a field name. Each ThingSpeak channel can have up to 8 fields.
- **Metadata:** Enter information about channel data, including JSON, XML, or CSV data.
- **Tags:** Enter keywords that identify the channel. Separate tags with commas.
- **Link to External Site:** If you have a website that contains information about your ThingSpeak channel, specify the URL.
- **Show Channel Location:**
 - **Latitude:** Specify the latitude position in decimal degrees. For example, the latitude of the city of London is 51.5072.
 - **Longitude:** Specify the longitude position in decimal degrees. For example, the longitude of the city of London is -0.1275.
 - **Elevation:** Specify the elevation position meters. For example, the elevation of the city of London is 35.052.
- **Video URL:** If you have a YouTube™ or Vimeo® video that displays your channel

Step 2: Get MQTT keys for these channels.

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



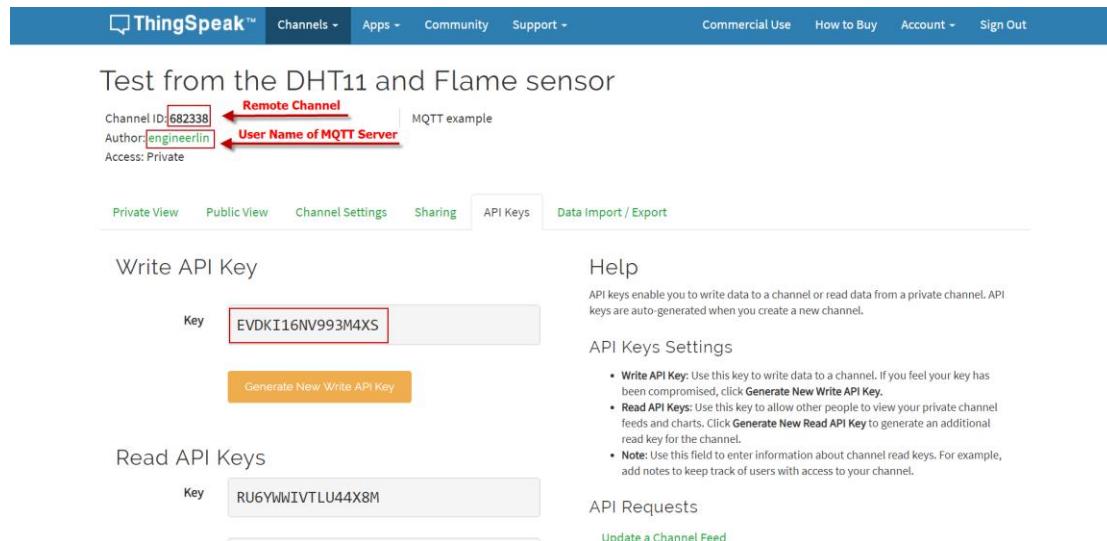
The screenshot shows the 'ThingSpeak™' account settings page. In the 'ThingSpeak settings' section, under 'MQTT API Key', the key value 'BYR3I5ECL787PHG9' is highlighted with a red box and labeled 'Password of MQTT Server'. A red arrow labeled '2' points to this key. Another red arrow labeled '1' points to the 'My Profile' link in the top right corner of the page.

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 'Test from the DHT11 and Flame sensor' channel page. In the 'API Keys' tab, the 'Key' field contains 'EVDKI16NV993M4XS' and is highlighted with a red box and labeled 'User Name of MQTT Server'. A red arrow labeled '1' points to this key. Below the key field is a 'Generate New Write API Key' button. To the right, there is a 'Help' section with information about API keys and a 'API Keys Settings' section with notes about write and read API keys.

4.3 Simulate MQTT uplink via PC's MQTT tool

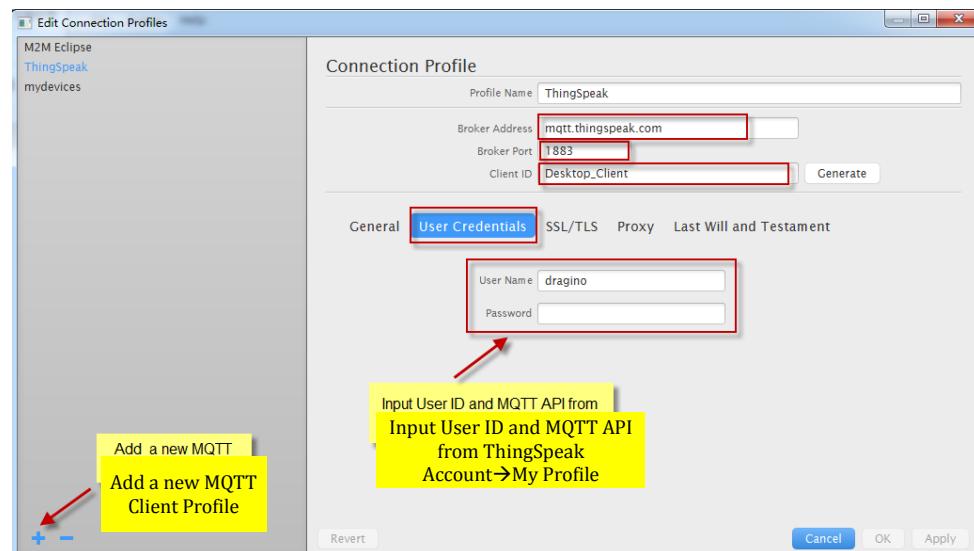
This step is not necessary, it just to help user to understand the MQTT protocol and simulate the MQTT connection to ThingSpeak. And check if the account info is valid and correct.

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

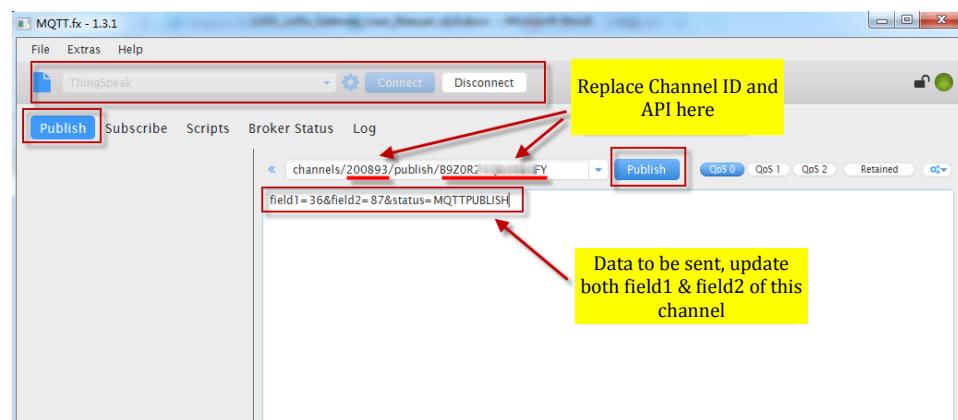
Broker Address: mqtt.thingspeak.com

Broker Port: 1883

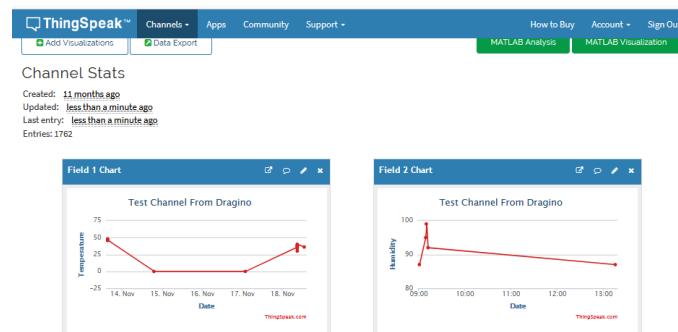
Client ID: User Defined



After add the profile, connect it and publish to the corresponding Channel with correct API key.



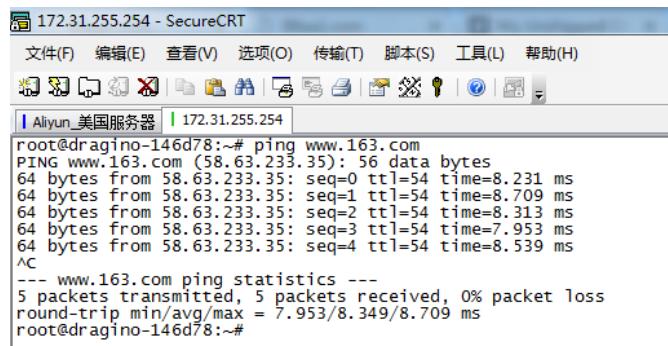
If update successful, we can see the update in the channel:



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)
阿里云_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 QZXTxxxxxO2J -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.

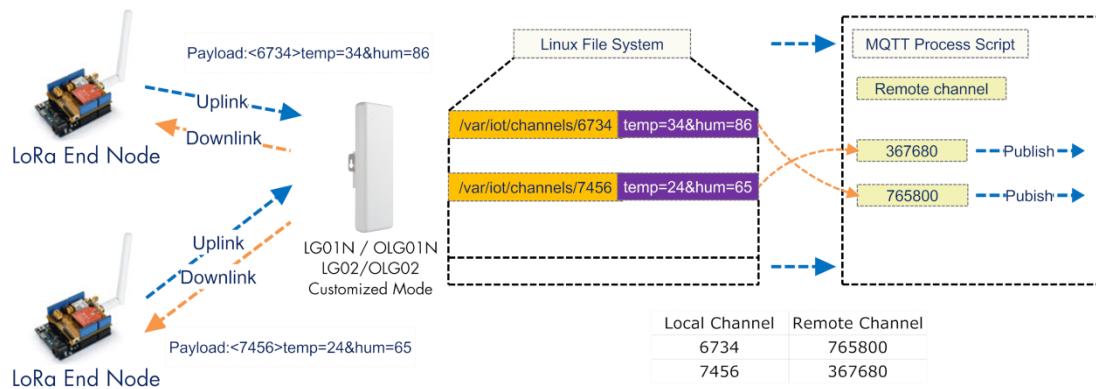
So we success to use LG01-N to uplink data to ThingSpeak, the [mosquitto_pub](#) command is executed in the Linux side, finally, we will have to call [mosquitto_pub](#) command while the LoRa sensor data arrive. We will explain how to do that in next step.

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:

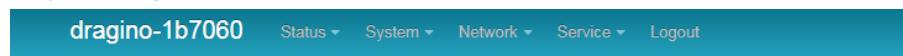
How MQTT script works:



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



LoRa Gateway Settings

Configuration to communicate with LoRa devices and LoRaWAN server

LoRaWAN Server Settings

IoT Service	LoRaRAW forward to MQTT server
Debug Level	Little message output
Service Provider	The Things Network

Step2: Configure MQTT server info

MQTT Server Settings

Configuration to communicate with MQTT server

Configure MQTT Server

Select Server	ThingSpeak MQTT
User Name [-u]	dragino1
Password [-P]	32W6GMEXYTEQ7049
Client ID [-i]	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

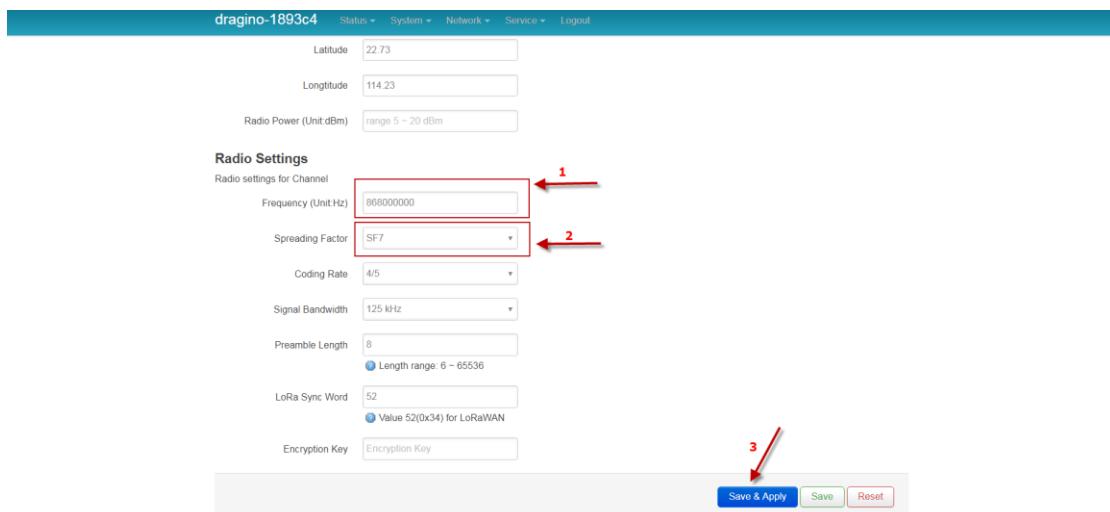
- ✓ Client_ID[-i]: dragino_Client (can put any string)
- ✓ Because we choose Thingspeak so we have below pre-set options but not show in web
 - Broker Address[-h]: mqtt.thingspeak.com
 - Broker Port[-p]: 1883
 - Topic Format[-t]: channels/CHANNEL/publish/WRITE_API
 - Data String Format[-m]: DATA&status=MQTTPUBLISH

And we configure this channel:

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

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:

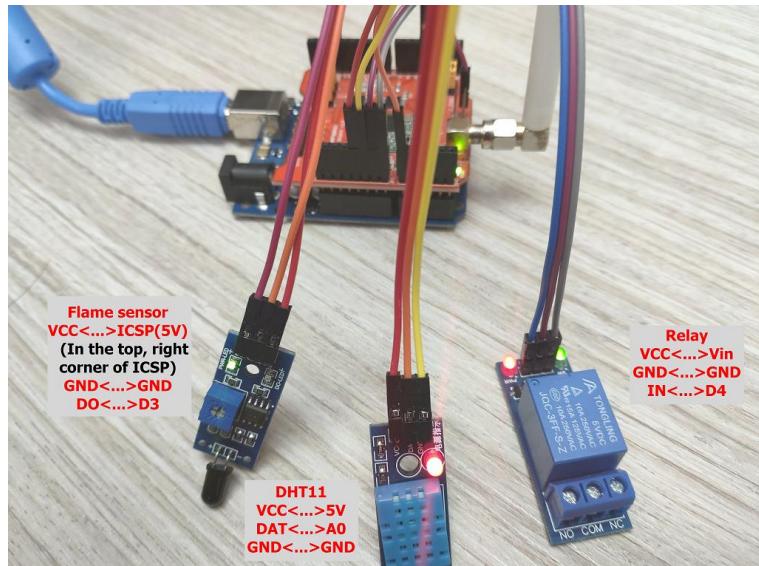


The screenshot shows the DRAGINO web interface with the following details:

- Latitude:** 22.73
- Longitude:** 114.23
- Radio Power (Unit dBm):** range 5 ~ 20 dBm
- Radio Settings:**
 - Frequency (Unit Hz):** 868000000 (highlighted by red box and arrow 1)
 - Spreading Factor:** SF7 (highlighted by red box and arrow 2)
 - Coding Rate:** 4/5
 - Signal Bandwidth:** 125 kHz
 - Preamble Length:** 8 (with note: Length range: 6 ~ 65536)
 - LoRa Sync Word:** 52 (with note: Value 52(0x34) for LoRaWAN)
 - Encryption Key:** [redacted]
- Buttons at the bottom:** Save & Apply (highlighted by red arrow 3), Save, Reset

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

```

MQTT_DHT11_and_Flame_sensor_Client_update_to_ThingSpeak
文件: 编辑(E) 工具(T) 帮助(H)
MQTT_DHT11_and_Flame_sensor_Client_update_to_ThingSpeak_
1 #include <dht.h>
2 #include <SPI.h>
3 #include <LoRa.h>
4
5 // Singleton instance of t
6
7 dht DHT;
8 #define DHT11_PIN A0
9 const int flame_pin=3; // 
10 float temperature, humidity
11 char tem[8]={"\0"}, hum[8]
avrduke done. Thank you.

```

The temperature and humidity:
[24.00°C, 65.00%]
The packet is send successful
COUNT=4 #####
The temperature and humidity:
[24.00°C, 65.00%]
The packet is send successful
COUNT=5 #####
The temperature and humidity:
[24.00°C, 65.00%]
The packet is send successful
COUNT=6 #####
The temperature and humidity:
[24.00°C, 65.00%]
The packet is send successful
COUNT=7

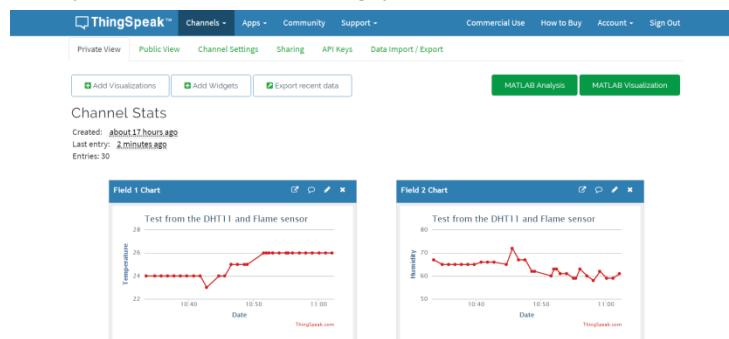
And we can see the logread of gateway as below, means the packet arrive gateway:

```

10.130.2.125
文件(F) 编辑(E) 查看(V) 选项(O) 传输(T) 脚本(S) 工具(L) 帮助(H)
10.130.2.125
Sun Jan 20 02:52:58 2019 user.notice root: [iot.mqtt]: Find Match entry for 3678
Sun Jan 20 02:52:58 2019 user.notice root: [iot.mqtt]: [-t] channel1=682338,publish/EVDK11AnV993M4X
Sun Jan 20 02:52:58 2019 user.notice root: [iot.mqtt]: [-t] Field1=26,0,0#Field2=01,0#0x00#status=MQTT_PUBLISH
Sun Jan 20 02:53:57 2019 daemon.info ig01_dkt_fwm[26279]: RXTC- receive(HEX):c3c3637383e6669656c6413d32362e30266669656c643
23d631a8#0
Sun Jan 20 02:53:58 2019 user.notice root: [iot.mqtt]: Find Match entry for 3678
Sun Jan 20 02:53:58 2019 user.notice root: [iot.mqtt]: [-t] channel1=682338,publish/EVDK11AnV993M4X
Sun Jan 20 02:53:58 2019 user.notice root: [iot.mqtt]: [-t] Field1=26,0,0#Field2=01,0#0x00#status=MQTT_PUBLISH
Sun Jan 20 02:54:57 2019 daemon.info ig01_dkt_fwm[26279]: RXTC- receive(HEX):c3c3637383e6669656c6413d32362e30266669656c643
23d631a8#0
Sun Jan 20 02:54:58 2019 user.notice root: [iot.mqtt]: Find Match entry for 3678
Sun Jan 20 02:54:58 2019 user.notice root: [iot.mqtt]: [-t] channel1=682338,publish/EVDK11AnV993M4X
Sun Jan 20 02:54:58 2019 user.notice root: [iot.mqtt]: [-t] Field1=26,0,0#Field2=01,0#0x00#status=MQTT_PUBLISH
Sun Jan 20 02:55:19 2019 daemon.info ig01_dkt_fwm[26279]: RXTC- receive(HEX):c3c3637383e6669656c6413d32362e30266669656c643
23d631a8#0
Sun Jan 20 02:55:22 2019 user.notice root: [iot.mqtt]: Find Match entry for 3678
Sun Jan 20 02:55:22 2019 user.notice root: [iot.mqtt]: [-t] channel1=682338,publish/EVDK11AnV993M4X
Sun Jan 20 02:55:22 2019 user.notice root: [iot.mqtt]: [-t] Field1=26,0,0#Field2=01,0#0x00#status=MQTT_PUBLISH
Sun Jan 20 02:55:22 2019 user.notice root: [iot.mqtt]: [-t] Field1=26,0,0#Field2=39,0#0x00#status=MQTT_PUBLISH
Sun Jan 20 02:55:58 2019 daemon.info ig01_dkt_fwm[26279]: RXTC- receive(HEX):c3c3637383e6669656c6413d32362e30266669656c643
23d631a8#0
Sun Jan 20 02:56:58 2019 user.notice root: [iot.mqtt]: Find Match entry for 3678
Sun Jan 20 02:56:58 2019 user.notice root: [iot.mqtt]: [-t] channel1=682338,publish/EVDK11AnV993M4X
Sun Jan 20 02:56:58 2019 user.notice root: [iot.mqtt]: [-t] Field1=26,0,0#Field2=01,0#0x00#status=MQTT_PUBLISH
Sun Jan 20 02:56:58 2019 user.notice root: [iot.mqtt]: [-t] Field1=26,0,0#Field2=39,0#0x00#status=MQTT_PUBLISH
Sun Jan 20 02:56:58 2019 user.notice root: [iot.mqtt]: [-t] Field1=26,0,0#Field2=40,0#0x00#status=MQTT_PUBLISH
Sun Jan 20 02:57:04 2019 user.notice root: [iot.mqtt]: Find Match entry for 3678
Sun Jan 20 02:57:04 2019 user.notice root: [iot.mqtt]: [-t] channel1=682338,publish/EVDK11AnV993M4X
Sun Jan 20 02:57:04 2019 user.notice root: [iot.mqtt]: [-t] Field1=26,0,0#Field2=02,0#0x00#status=MQTT_PUBLISH
Sun Jan 20 02:57:58 2019 user.notice root: [iot.mqtt]: RXTC- receive(HEX):c3c3637383e6669656c6413d32362e30266669656c643
23d6322e#0
Sun Jan 20 02:58:04 2019 user.notice root: [iot.mqtt]: Find Match entry for 3678
Sun Jan 20 02:58:04 2019 user.notice root: [iot.mqtt]: [-t] channel1=682338,publish/EVDK11AnV993M4X
Sun Jan 20 02:58:04 2019 user.notice root: [iot.mqtt]: [-t] Field1=26,0,0#Field2=01,0#0x00#status=MQTT_PUBLISH
Sun Jan 20 02:58:04 2019 user.notice root: [iot.mqtt]: [-t] Field1=26,0,0#Field2=39,0#0x00#status=MQTT_PUBLISH
Sun Jan 20 02:58:04 2019 user.notice root: [iot.mqtt]: [-t] Field1=26,0,0#Field2=40,0#0x00#status=MQTT_PUBLISH
Sun Jan 20 02:58:58 2019 user.notice root: [iot.mqtt]: RXTC- receive(HEX):c3c3637383e6669656c6413d32362e30266669656c643
23d6322e#0
Sun Jan 20 02:59:04 2019 user.notice root: [iot.mqtt]: Find Match entry for 3678
Sun Jan 20 02:59:04 2019 user.notice root: [iot.mqtt]: [-t] channel1=682338,publish/EVDK11AnV993M4X
Sun Jan 20 02:59:04 2019 user.notice root: [iot.mqtt]: [-t] Field1=26,0,0#Field2=01,0#0x00#status=MQTT_PUBLISH
Sun Jan 20 02:59:04 2019 user.notice root: [iot.mqtt]: [-t] Field1=26,0,0#Field2=39,0#0x00#status=MQTT_PUBLISH
Sun Jan 20 02:59:04 2019 user.notice root: [iot.mqtt]: [-t] Field1=26,0,0#Field2=40,0#0x00#status=MQTT_PUBLISH
Sun Jan 20 03:01:04 2019 user.notice root: [iot.mqtt]: Find Match entry for 3678
Sun Jan 20 03:01:04 2019 user.notice root: [iot.mqtt]: [-t] channel1=682338,publish/EVDK11AnV993M4X
Sun Jan 20 03:01:04 2019 user.notice root: [iot.mqtt]: [-t] Field1=26,0,0#Field2=01,0#0x00#status=MQTT_PUBLISH
Sun Jan 20 03:01:04 2019 user.notice root: [iot.mqtt]: [-t] Field1=26,0,0#Field2=39,0#0x00#status=MQTT_PUBLISH
Sun Jan 20 03:01:04 2019 user.notice root: [iot.mqtt]: [-t] Field1=26,0,0#Field2=40,0#0x00#status=MQTT_PUBLISH
Sun Jan 20 03:01:58 2019 daemon.info ig01_dkt_fwm[26279]: RXTC- receive(HEX):c3c3637383e6669656c6413d32362e30266669656c643
23d6322e#0
Sun Jan 20 03:02:04 2019 user.notice root: [iot.mqtt]: [-t] channel1=682338,publish/EVDK11AnV993M4X
Sun Jan 20 03:02:04 2019 user.notice root: [iot.mqtt]: [-t] Field1=26,0,0#Field2=01,0#0x00#status=MQTT_PUBLISH

```

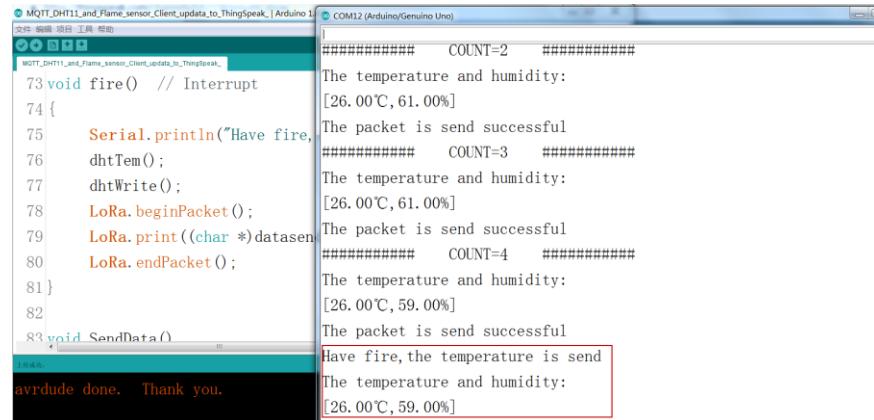
Finally, we can see on the ThingSpeak:



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:



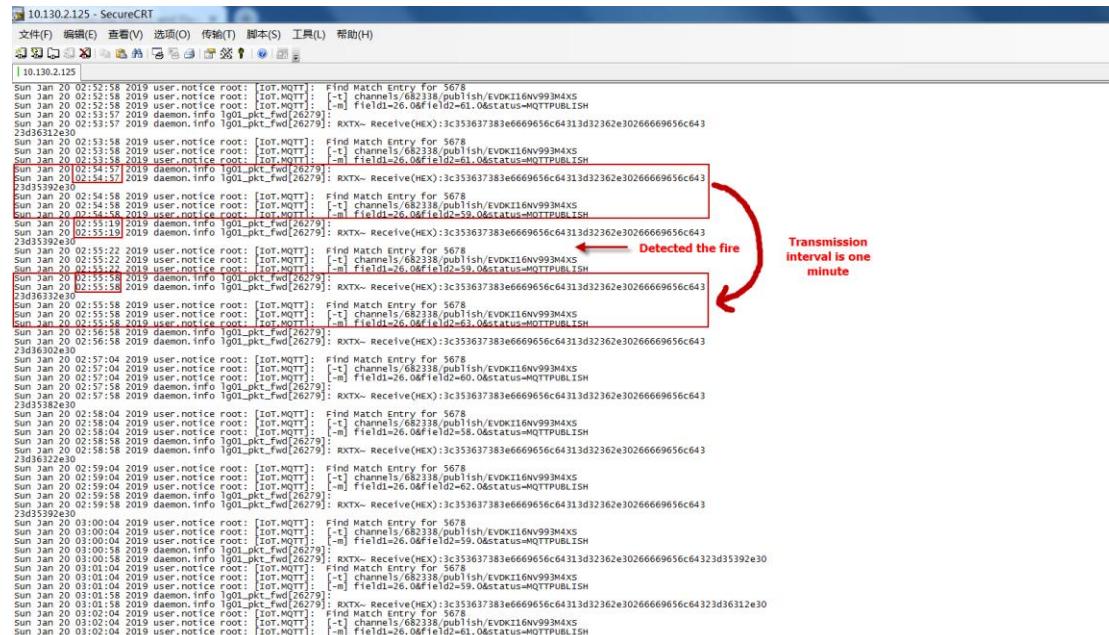
```

MQTT_DHT11_and_Flame_sensor_Client_update_to_ThingSpeak_.| Arduino 1
文件 帮助 项目 工具 附录
MQTT_DHT11_and_Flame_sensor_Client_update_to_ThingSpeak_
73 void fire() // Interrupt
74 {
75     Serial.println("Have fire,
76     dhtTem();
77     dhtWrite();
78     LoRa.beginPacket();
79     LoRa.print((char *)datasen
80     LoRa.endPacket();
81 }
82
83 void SendData()
84 {
85
86
87
88
89
avrduke done. Thank you.

```

The temperature and humidity:
[26.00°C, 61.00%]
The packet is send successful
COUNT=2 #####
The temperature and humidity:
[26.00°C, 61.00%]
The packet is send successful
COUNT=3 #####
The temperature and humidity:
[26.00°C, 61.00%]
The packet is send successful
COUNT=4 #####
The temperature and humidity:
[26.00°C, 59.00%]
The packet is send successful
Have fire, the temperature is send
The temperature and humidity:
[26.00°C, 59.00%]

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



```

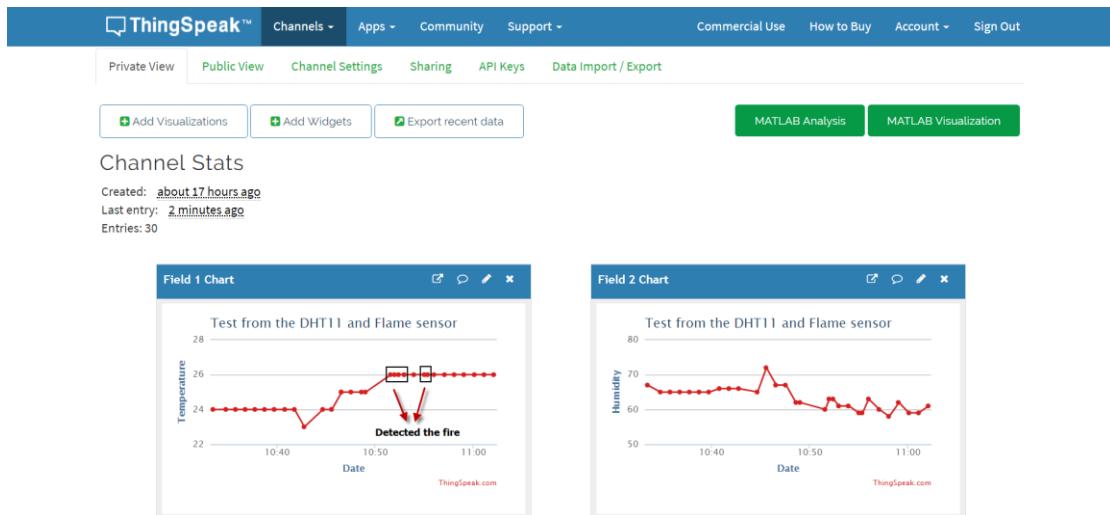
10.130.2.125 - SecureCRT
文件(F) 编辑(E) 查看(V) 选项(O) 传输(T) 脚本(S) 工具(L) 帮助(H)
10.130.2.125
Sun Jan 20 02:52:58 2019 user.notice root: [IOT-MQTT]: Find Match Entry for 5678
Sun Jan 20 02:52:58 2019 user.notice root: [IOT-MQTT]: [-] channels/682338/publish/EVDK11ENV993MAXS
Sun Jan 20 02:52:58 2019 user.notice root: [IOT-MQTT]: [-m] Field1=26,0&Field1=d2-61,0&status=MQTT_PUBLISH
Sun Jan 20 02:53:57 2019 daemon.info 1901-pkt-fwd[26279]: RXTx Receive(HEX):3c353637383e66696956c64313d32362e30266669656c643
23d3612e30
Sun Jan 20 02:53:58 2019 user.notice root: [IOT-MQTT]: Find Match Entry for 5678
Sun Jan 20 02:53:58 2019 user.notice root: [IOT-MQTT]: [-] channels/682338/publish/EVDK11ENV993MAXS
Sun Jan 20 02:53:58 2019 user.notice root: [IOT-MQTT]: [-m] Field1=26,0&Field1=d2-61,0&status=MQTT_PUBLISH
Sun Jan 20 02:53:58 2019 daemon.info 1901-pkt-fwd[26279]: RXTx Receive(HEX):3c353637383e66696956c64313d32362e30266669656c643
23d3592e30
Sun Jan 20 02:54:58 2019 user.notice root: [IOT-MQTT]: Find Match Entry for 5678
Sun Jan 20 02:54:58 2019 user.notice root: [IOT-MQTT]: [-] channels/682338/publish/EVDK11ENV993MAXS
Sun Jan 20 02:54:58 2019 user.notice root: [IOT-MQTT]: [-m] Field1=26,0&Field1=d2-59,0&status=MQTT_PUBLISH
Sun Jan 20 02:54:58 2019 daemon.info 1901-pkt-fwd[26279]: RXTx Receive(HEX):3c353637383e66696956c64313d32362e30266669656c643
23d3592e30
Sun Jan 20 02:55:19 2019 user.notice root: [IOT-MQTT]: Find Match Entry for 5678
Sun Jan 20 02:55:19 2019 user.notice root: [IOT-MQTT]: [-] channels/682338/publish/EVDK11ENV993MAXS
Sun Jan 20 02:55:19 2019 daemon.info 1901-pkt-fwd[26279]: RXTx Receive(HEX):3c353637383e66696956c64313d32362e30266669656c643
23d3592e30
Sun Jan 20 02:55:22 2019 user.notice root: [IOT-MQTT]: Find Match Entry for 5678
Sun Jan 20 02:55:22 2019 user.notice root: [IOT-MQTT]: [-] channels/682338/publish/EVDK11ENV993MAXS
Sun Jan 20 02:55:22 2019 user.notice root: [IOT-MQTT]: [-m] Field1=26,0&Field1=d2-59,0&status=MQTT_PUBLISH
Sun Jan 20 02:55:58 2019 daemon.info 1901-pkt-fwd[26279]: RXTx Receive(HEX):3c353637383e66696956c64313d32362e30266669656c643
23d3592e30
Sun Jan 20 02:55:58 2019 user.notice root: [IOT-MQTT]: Find Match Entry for 5678
Sun Jan 20 02:55:58 2019 user.notice root: [IOT-MQTT]: [-] channels/682338/publish/EVDK11ENV993MAXS
Sun Jan 20 02:55:58 2019 user.notice root: [IOT-MQTT]: [-m] Field1=26,0&Field1=d2-59,0&status=MQTT_PUBLISH
Sun Jan 20 02:55:58 2019 daemon.info 1901-pkt-fwd[26279]: RXTx Receive(HEX):3c353637383e66696956c64313d32362e30266669656c643
23d3592e30
Sun Jan 20 02:58:04 2019 user.notice root: [IOT-MQTT]: Find Match Entry for 5678
Sun Jan 20 02:58:04 2019 user.notice root: [IOT-MQTT]: [-] channels/682338/publish/EVDK11ENV993MAXS
Sun Jan 20 02:58:04 2019 user.notice root: [IOT-MQTT]: [-m] Field1=26,0&Field1=d2-60,0&status=MQTT_PUBLISH
Sun Jan 20 02:58:04 2019 daemon.info 1901-pkt-fwd[26279]: RXTx Receive(HEX):3c353637383e66696956c64313d32362e30266669656c643
23d3592e30
Sun Jan 20 02:59:04 2019 user.notice root: [IOT-MQTT]: Find Match Entry for 5678
Sun Jan 20 02:59:04 2019 user.notice root: [IOT-MQTT]: [-] channels/682338/publish/EVDK11ENV993MAXS
Sun Jan 20 02:59:04 2019 user.notice root: [IOT-MQTT]: [-m] Field1=26,0&Field1=d2-62,0&status=MQTT_PUBLISH
Sun Jan 20 02:59:04 2019 daemon.info 1901-pkt-fwd[26279]: RXTx Receive(HEX):3c353637383e66696956c64313d32362e30266669656c643
23d3592e30
Sun Jan 20 03:00:04 2019 user.notice root: [IOT-MQTT]: Find Match Entry for 5678
Sun Jan 20 03:00:04 2019 user.notice root: [IOT-MQTT]: [-] channels/682338/publish/EVDK11ENV993MAXS
Sun Jan 20 03:00:04 2019 user.notice root: [IOT-MQTT]: [-m] Field1=26,0&Field1=d2-59,0&status=MQTT_PUBLISH
Sun Jan 20 03:00:04 2019 daemon.info 1901-pkt-fwd[26279]: RXTx Receive(HEX):3c353637383e66696956c64313d32362e30266669656c643
23d3592e30
Sun Jan 20 03:01:04 2019 user.notice root: [IOT-MQTT]: Find Match Entry for 5678
Sun Jan 20 03:01:04 2019 user.notice root: [IOT-MQTT]: [-] channels/682338/publish/EVDK11ENV993MAXS
Sun Jan 20 03:01:04 2019 user.notice root: [IOT-MQTT]: [-m] Field1=26,0&Field1=d2-59,0&status=MQTT_PUBLISH
Sun Jan 20 03:01:04 2019 daemon.info 1901-pkt-fwd[26279]: RXTx Receive(HEX):3c353637383e66696956c64313d32362e30266669656c643
23d3592e30
Sun Jan 20 03:01:58 2019 user.notice root: [IOT-MQTT]: Find Match Entry for 5678
Sun Jan 20 03:01:58 2019 user.notice root: [IOT-MQTT]: [-] channels/682338/publish/EVDK11ENV993MAXS
Sun Jan 20 03:01:58 2019 user.notice root: [IOT-MQTT]: [-m] Field1=26,0&Field1=d2-59,0&status=MQTT_PUBLISH
Sun Jan 20 03:01:58 2019 daemon.info 1901-pkt-fwd[26279]: RXTx Receive(HEX):3c353637383e66696956c64313d32362e30266669656c643
23d3592e30
Sun Jan 20 03:02:04 2019 user.notice root: [IOT-MQTT]: Find Match Entry for 5678
Sun Jan 20 03:02:04 2019 user.notice root: [IOT-MQTT]: [-] channels/682338/publish/EVDK11ENV993MAXS
Sun Jan 20 03:02:04 2019 user.notice root: [IOT-MQTT]: [-m] Field1=26,0&Field1=d2-61,0&status=MQTT_PUBLISH
Sun Jan 20 03:02:04 2019 user.notice root: [IOT-MQTT]: [-m] Field1=26,0&Field1=d2-61,0&status=MQTT_PUBLISH

```

Detected the fire

Transmission interval is one minute

Finally, we can see on the ThingSpeak:



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/8fnvlfj90tgbnbcy whole0gn/T/arduino_cache_833512/core/core_arduino
_avr_arduino_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 have 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](#).

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

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](#)