ceotdocs

Bruno

Nov 11, 2022

CONTENTS

CHAPTER

ONE

CHIRPSTACK

Chirpstack is an open-source LoRaWAN Network Server stack that provides the essential components to build user-friendly web interfaces for device management. Chirpstack APIs are provided for integration purposes, allowing LoRaWAN stack to be easily integrated with applications.

Chirpstack is composed of three modules that work together for a client to be able to communicate with end-devices. These operate as follows.

Chirpstack Arquitecture - Image taken from Chirpstack Website

1.1 Gateway Bridge

The Chirpstack Gateway Bridge (CGB) is the module of Chirpstack LoRaWAN stack that converts the Packet Forwarder message payloads into a serialized data format (JSON or Protobuf) and publishes it in a Message Queuing Telemetry Transport (MQTT) server, to which the Network Server subscribes. The message will be published with a topic that indicates the source (by default the gateway MAC number and event type: up, down, status). This module can also be configured to support different types of packet forwarders.

1.1.1 Requirements

- Any MQTT broker for instance Mosquitto

```
sudo apt-get install mosquitto
```

If you wanna pub or sub to mqtt you will need to install a pub/sub package (optional: used for debugging).

```
sudo apt-get install mosquitto-clients
```

1.1.2 Instalation

```
sudo apt install chirpstack-gateway-bridge
```

1.1.3 Configurations

The configurations for the gateway bridge will be located at `/etc/chirpstack-gateway-bridge/chirpstack-gateway-bridge.toml`

By default the gateway bridge will listen to port 1700 and will encode it with json and sent it to port localhost:1883 (MQTT default).

If you want to do add new message formats that arent `stats|uplinks|downlinks|execs|acks` you can add them in the configuration file. The topic format for up/downlinks is usually `gateway/[gateway_id]/event/[event_type]`. You can check more about the formats and events in here.

You can also change the detail of the logs, it goes from 0 to 5, where 5 is the one with the most details and mostly used for debugging. This is achieved by after `[general]` write `[log_level=number]`.

1.1.4 Logs and Debugging

```
journalctl -u chirpstack-gateway-bridge -f -n 50
```

To track if the gateway bridge is sending packets at mqtt you can use `$mosquitto_sub -h <ipaddress> -t <topic>`. By default the topic that you wanna use is `gateway/#/event/up` which will track all the uplinks sent from the gateway bridge to the network server. To check any kid of package you can just use `gateway/#`. For more information on debugging you can check here.

If you wanna track if the gateway is sending packets towards your server, you can use `tcpdump`.

If the ChirpStack Gateway Bridge is installed on the gateway itself, then the following command must be executed on the gateway itself:

```
sudo tcpdump -AUq -i lo port 1700
```

If the ChirpStack Gateway Bridge is installed on a server, then you must execute the following command (either on the gateway or on the receiving server):

```
sudo tcpdump -AUq port 1700
```

more details can be foun in here.

1.2 Network Server

Once the message has been published in the MQTT server, the Chirpstack Network Server (subscribing the topic) will read the message and confirm if it is a duplicate message. The duplication of LoRa frames can happen because a LoRa frame is broadcasted, allowing multiple gateways to read the frame for delivery to the CNS. In this case the CNS will keep the frame from the gateway that has the best connection with the device. The CNS also remembers gateway-device associations for downlink purposes. After dealing with frame duplication it will communicate with the Application Server through google Remote Procedure Call (gRPC).

1.2.1 Requirements

- Any MQTT broker for instance Mosquitto

```
sudo apt-get install mosquitto
```

If you wanna pub or sub to mqtt you will need to install a pub/sub package (optional: used for debugging).

```
sudo apt-get install mosquitto-clients
```

- Database: by default uses PSQL.

```
sudo apt install postgresql
```

In the SQL console:

```
-- create role for authentication
create role chirpstack with login password 'chirpstack';

-- create database
create database chirpstack with owner chirpstack;

-- change to chirpstack database
\c chirpstack

-- create pg_trgm extension
create extension pg_trgm;

-- exit psql
\q
```

- Redis

```
sudo apt install redis-server
```

1.2.2 Instalation

```
sudo apt install chirpstack-gateway-bridge
```

1.2.3 Configurations

The configurations for the gateway bridge will be located at `/etc/chirpstacknetworkserver/chirpstacknetworkserver.toml`

First thing it needs to be here is to set up your database in the line `dns=postgres://user:password@hostname/database?sslmode=disable`.

In here you can configure the **delays** related information such as **deduplication_delay** which is the interval for how long the network server waits for duplication packets and **get_downlink_data_delay** which is how long the network server waits to send the downlink after receiving the uplink.

Can also easily add **ADR plugins** which are alternate versions to the default ADR. To do that just add the location of your built GO plugin example.

```
// Handle handles the ADR and returns the (new) parameters.
func (h *Handler) Handle(req adr.HandleRequest) (adr.HandleResponse, error) {
        resp := adr.HandleResponse{
                DR:           req.DR,
                TxPowerIndex: req.TxPowerIndex,
                NbTrans:      req.NbTrans,
        }

// if ADR is disabled, return current values
        if !req.ADR {
            return resp, nil
        }

// gets info related to the last package received
        up := req.UplinkHistory[len(req.UplinkHistory)-1]

// calculates the nstep value using SNR
        snrM := up.MaxSNR
        snrMargin := snrM - req.RequiredSNRForDR - req.InstallationMargin
        nStep := int(snrMargin / 3)


// gets the SF and BW value corresponding to the datarate received
        sf, bw := get_sf_bw_val(req.DR)
        tx := up.TXPowerIndex

// dummy data example
// if TXPOWER index is 0 then increase to 2 just for demonstration
        if up.TXPowerIndex > 1 {
// get the values from the solution table generated with RL Agent
                newsf, newtx := RLAGENT(sf, tx, nStep)

                log.Info("New SF: ", newsf, " NewTX: ", newtx)

// calculates new datarate based on the new datarate and new bw
                newdr := get_dr_val(newsf, bw)

// if new datarate is higher than the max dr of the device,
// it sets the value of the datarate to max datarate
                if newdr > req.MaxDR {
                        resp.DR = req.MaxDR
                } else {
                        resp.DR = newdr
                }
                resp.TxPowerIndex = newtx
        } else {
                resp.TxPowerIndex = 2
        }

        // resp.DR = 3
        return resp, nil
}
```

```
go build projectdir
```

You can also change the detail of the logs, it goes from 0 to 5, where 5 is the one with the most details and mostly used for debugging. This is achieved by after `[general]` write `[log_level=number]`.

More details about configuration of the network server can be found in here.

1.2.4 Logs and Debugging

```
journalctl -u chirpstack-network-server -fn 50
```

This allows to check if the Network Server is communicating with the Application Server.

1.3 Application Server

This is the last module of Chirpstack that is responsible for the join-requests and encrypts the payloads of the application. It also provides a web interface for users where they can control users where they can control the features mentioned in the Network Server and a few more. It also offers a restful API and allows sending post whenever it receives information from a device to a user application.

1.3.1 Requirements

- Any MQTT broker for instance Mosquitto

```
sudo apt-get install mosquitto
```

If you wanna pub or sub to mqtt you will need to install a pub/sub package (optional: used for debugging).

```
sudo apt-get install mosquitto-clients
```

- Database: by default uses PSQL.

```
sudo apt install postgresql
```

In the SQL console:

```
sudo -u postgres psql

-- create the chirpstack_ns user with password 'dbpassword'
create role chirpstack_ns with login password 'dbpassword';

-- create the chirpstack_ns database
create database chirpstack_ns with owner chirpstack_ns;

-- exit the prompt
\q
```

- Redis

```
sudo apt install redis-server
```

1.3.2 Instalation

```
sudo apt install chirpstack-application-server
```

1.3.3 Configurations

The settings in this section are more related to association with other applications like Influxdb, AWS, etc. The App Server provides an API port, which is set by default to port 8003, and the API provides information available on the Network Server, which was mentioned earlier, for example subscribing to the frames received by a certain device as long as it has permission to do so.

There are not many settings in this section. This is due to the fact that the Application Serv\er mostly works as a server with a web interface that allows you to easily manage the Network Server, allowing you to create gateways, devices, profiles and all other content mentioned in the Network Server.

It also offers the possibility to integrate the Chirpstack with other applications, depending on your needs. To do so, different steps have to be followed depending on the application. It is in this process that you can integrate an application created by the user, and for that, you just have to select the ip:port of the app's server.

1.3.4 Debugging and Logging

```
journalctl -u chirpstack-application-server -fn 50
```

This allows to check if the Application Server is successfully sending and receiving packets.

1.4 API

The API allows to easily control chirpstack Network Server using your own Application and it is quite simple to use. Only requires the user to create a key in the application server. The key is a JWT Token that is originated by the key in the Application Server config file in conjuction with the user account information.

More information in here.

JWT Token: https://jwt.io/

Http Examples: https://www.chirpstack.io/application-server/api/go-examples/

Enqueue Downlink Example with curl:

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json
↪' --header 'Grpc-Metadata-Authorization: Bearer <API TOKEN>' -d '{ \
  "deviceQueueItem": { \
    "confirmed": false, \
    "data": "AQID", \
    "fPort": 10 \
  } \
}' 'http://localhost:8080/api/devices/0101010101010101/queue'
```

CHAPTER

TWO

DEVICES

2.1 Gateway RAK 7244

The internet connection setup is easy to make, it can be easily done by following the setup on rakwireless website.

After the setup the only required setting that needs to be done is to reroute your packets towards your server. For that you simply need to:

1) connect to the gateway using ssh or any other mechanism
2) use the command `$sudo gateway-config`
3) select 'Setup RAK Gateway Channel Plan' and pick which service you are using (Chirpstack/loraserver or TTN/TTI).

Do not forget to turn off the Chirpstack services in the gateway to ease the processor work, they come installed and active by default.

2.2 DRAGINO LHT65

The Dragino Temperature and & Humidity (LHT65) module is an environment module that comes with a built-in SHT20 Temperature & Humidity sensor and has a external sensor that allows external sensors to be connected, such as an illumination sensor. This device works both as a data logger, recording up to 3200 environment measurements and also as a wireless sensor for network applications. The battery of this device is a 2400mAh non chargeable battery that is ratted for up to 10 years of utilization under optimal conditions (LoS communications and sparse uplink rates).

Built-in temperature sensor information:

- Accuracy tolerance : Typ $\pm$ 0.3 ∘C
- Long term drift: < 0.02 ∘ C/yr
- Operating range: -40 125 ∘C

Built-in humidity sensor information:

- Resolution: 0.04 % RH
- Accuracy tolerance: Typ $\pm$ 3 % PH
- Long term drift: < 0.02 ∘C/yr
- Operating range: 0 96 % PH

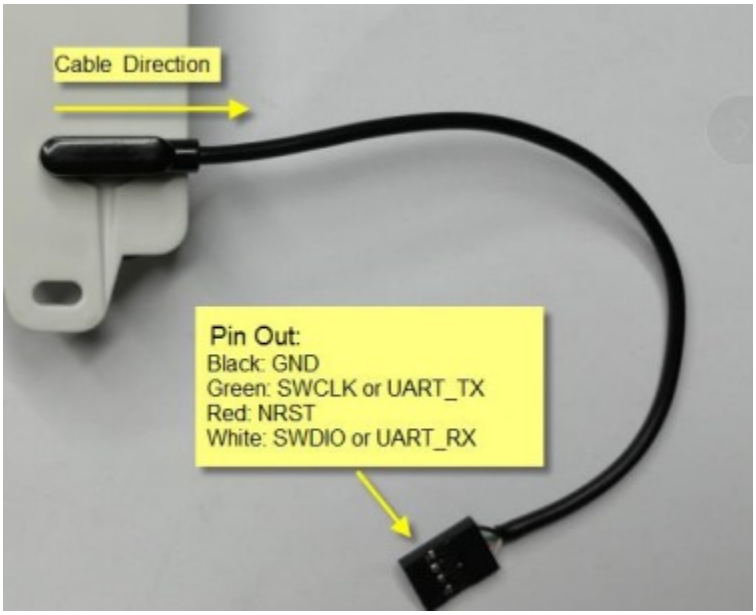
External illumination sensor information:

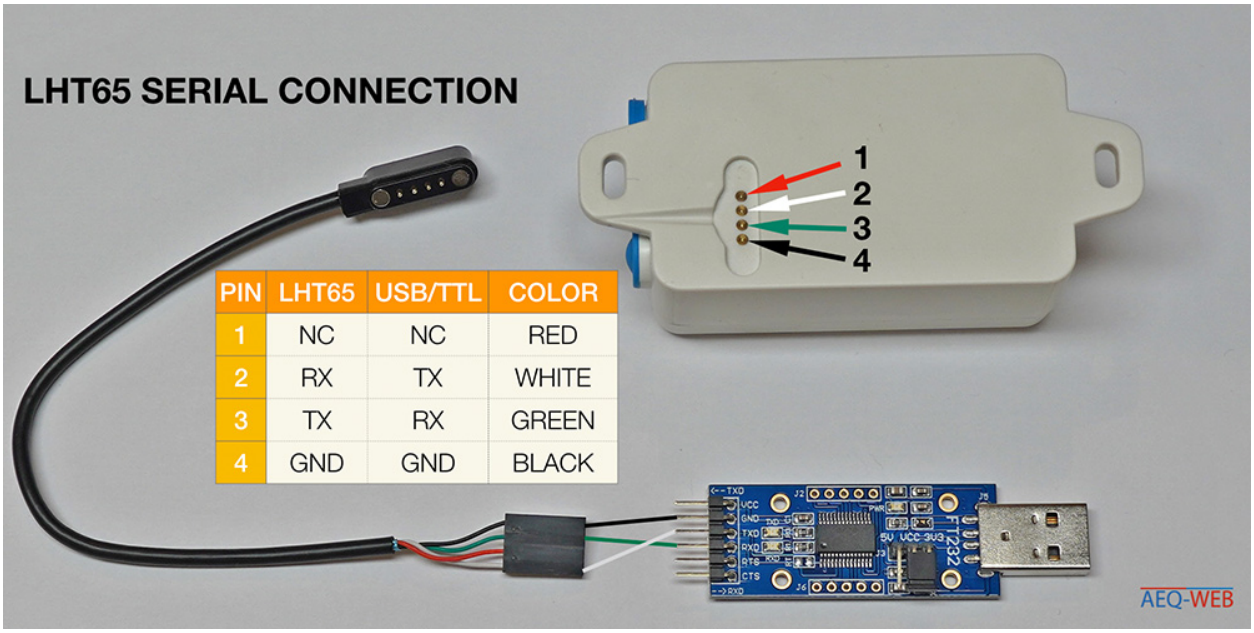
- Resolution: 1 lx

- Range: 0-65535 lx
- Operating range: -40 ∘C 85 ∘C With this device setup, the power consumption of the device is 4uA in idle mode and 130mA when transmitting at maximum power

2.2.1 Set Up

To config the lht65 it is necessary a TTL to USB converter which much be set up according to images following:





After this its necessary to find to which USB port the device is connected which can be done using the following commnad:

```
$ dmesg | grep tty
```

```
bruno@bruno-ThinkPad-X1-Carbon-2nd:~/Documents/CEOT$ dmesg | grep tty
[    0.073165] printk: console [tty0] enabled
[    0.412175] 0000:00:16.3: ttyS4 at I/O 0x30b0 (irq = 17, base_baud = 115200) is a 16550A
[    2.922739] usb 2-2: pl2303 converter now attached to ttyUSB0
[ 4619.687996] pl2303 ttyUSB0: pl2303_set_control_lines - failed: -19
[ 4619.688005] pl2303 ttyUSB0: error sending break = -19
[ 4619.688149] pl2303 ttyUSB0: pl2303 converter now disconnected from ttyUSB0
[ 5671.093106] usb 2-1: pl2303 converter now attached to ttyUSB0
```

TTL/USB converter name is pl2303, therefore the device is connected to COM ttyUSB0 (last line).

To be sure, before connecting to the device, you must have permission to access the port.

```
# sudo chmod 666 /dev/<COMNAME>
$ sudo chmod 666 /dev/ttyUSB0
```

Connecting to the device can be done using putty, cu(https://www.cyberciti.biz/faq/find-out-linux-serial-ports-with-setserial/) or comtools (https://github.com/Neutree/COMTool).

After that, the LHT65 can be accessed by any of the previous serial tools. The example bellows uses ComTool

To change the device values, first insert the password and a **paragraph** (mandatory).

Examples of commands:

Function	Command
Deactivate ADR	AT+ADR=0
Change DR	AT+DR=”number between 0 and 7”
Change Transmissing time	AT+TDC=”number in miliseconds”
Get TXP from device	AT+TXP=?
Change the EXT device	AT+EXT=5

2.2.2 Decoder Code

The decoder code for Chirpstack.

```
function Decode(fPort, bytes){
var data = {
      //External sensor
      Ext_sensor:
      {
```

(continues on next page)

(continued from previous page)

```
    "0":"No external sensor",
    "1":"Temperature Sensor",
    "4":"Interrupt Sensor send",
    "5":"Illumination Sensor",
    "6":"ADC Sensor",
    "7":"Interrupt Sensor count",
  }[bytes[6]&0x7F],

  //Battery,units:V
  BatV:((bytes[0]<<8 | bytes[1]) & 0x3FFF)/1000,

  //SHT20,temperature,units:
  // TempC_SHT:((bytes[2]<<24>>16 | bytes[3])/100).toFixed(2),
  // DARIO - Original (above) changed to return number instead of str
  TempC_SHT:((bytes[2]<<24>>16 | bytes[3])/100),

  //SHT20,Humidity,units:%
  //Hum_SHT:((bytes[4]<<8 | bytes[5])/10).toFixed(1),
  // DARIO - original (above) changed to return number instead of str
  Hum_SHT:((bytes[4]<<8 | bytes[5])/10),

  //DS18B20,temperature,units:
  TempC_DS:
  {
    "1":((bytes[7]<<24>>16 | bytes[8])/100).toFixed(2),
  }[bytes[6]&0xFF],

  //Exti pin level,PA4
  Exti_pin_level:
  {
    "4":bytes[7] ? "High":"Low",
  }[bytes[6]&0x7F],

  //Exit pin status,PA4
  Exti_status:
  {
    "4":bytes[8] ? "True":"False",
  }[bytes[6]&0x7F],

  //BH1750,illumination,units:lux
  ILL_lux:
  {
    "5":bytes[7]<<8 | bytes[8],
  }[bytes[6]&0x7F],

  //ADC,PA4,units:V
  ADC_V:
  {
    "6":(bytes[7]<<8 | bytes[8])/1000,
  }[bytes[6]&0x7F],

  //Exti count,PA4,units:times
```

(continues on next page)

(continued from previous page)

```
      Exit_count:
      {
        "7":bytes[7]<<8 | bytes[8],
      }[bytes[6]&0x7F],

      //Applicable to working mode 4567,and working mode 467 requires short circuit PA9␣
↪and PA10
      No_connect:
      {
        "1":"Sensor no connection",
      }[(bytes[6]&0x80)>>7],
  };
  return data;
}
```

You can calculate the estimated batery time using this spreadsheet

More information can be found here.