lora-mote-emulator

houlu

Feb 22, 2022

CONTENTS

CHAPTER

ONE

TUTORIALS

1.1 Introduction

`lora-mote-emulator` is developed by Python. It can emulate LoRa motes (a.k.a end-devices) and LoRa Gateways to send and receive LoRaWAN packages. It can be mainly utilized to test LoRaWAN servers.

1.2 Installation

`lora-mote-emulator` supports Python 3.6+, and is distributed via Pypi. Therefore, we can directly use `pip` to install this program. Meanwhile, `pipenv` is recommended to manage virtual environments.

1. Make sure Python > 3.6 is installed on the machine, and `pip` is upgraded to the latest version. To upgrade `pip`, use `pip install --upgrade pip` .

2. Install `pipenv`:

```
pip install pipenv
```

3. Create a new virtual environment in an empty directory:

```
pipenv --python 3
```

4. Install `lora-mote-emulator`:

```
pipenv install lora-mote-emulator
```

Now, the emulator is installed successfully.

1.3 Usage

The emulator includes an executable `mote`, thus, `mote -h` can show the help message

```
usage: mote [-h] [-v version] [-c CONFIG] [--model MODEL]
    {join,app,pull,mac,rejoin,info,abp,create} ...

Tool to emulate LoRa mote (a.k.a end-device) and Gateway, supported command
list: ['join', 'app', 'pull', 'mac', 'rejoin', 'info', 'abp', 'create']

optional arguments:
```

(continues on next page)

(continued from previous page)

```
    -h, --help              show this help message and exit
    -v version, --version version
                            Choose LoRaWAN version, 1.0.2 or 1.1(default)
    -c CONFIG, --config CONFIG
                            Specify the directory of config files, default './config'
    --model MODEL         Specify the directory to save the model file, default './models
↪'

Supported commands:
    {join,app,pull,mac,rejoin,info,abp,create}
    join                Send join request.
    app                 Send application data.
    pull                Send PULL_DATA.
    mac                 Send MACCommand.
    rejoin              Send rejoin request.
    info                Show information of current mote.
    abp                 Initialize mote in ABP mode.
    create              Handle configurations.
```

1.3.1 Prepare gateway

1. At the very beginning, we need to create configuration files. Run `mote create [-c config_dir]` to generate configuration files at directory `config_dir` (`./config` in default). The configuration files include `device.json`, `gateway.json`, `config.json`, and `abp.json`.

2. Modify the content of each configuration file:

- `config.json` includes the server's IP address and port number, log level and timeout, etc.
- `gateway.json` includes the gateway's extended unique identifier (EUI),
- `device.json` includes the detailed parameters of mote, i.e., `DevEUI`, `JoinEUI` and two root keys, i.e., `AppKey` and `NwkKey`,
- `abp.json` includes required fields if mote is activated via Activation by Personalization (ABP) mode.

3. Before mote sending message, the gateway needs to register the IP address to server via `PULL_DATA` message. (**Note**: Some LoRaWAN servers may require gateways to send `PULL_DATA` periodically to keep-alive. Therefore, if we get nothing from servers, we can send a `PULL_DATA` first, then re-try the message. To send a `PULL_DATA` message, use following command:

```
mote pull
```

4. If `INFO - PULL ACK -` logs, it means a `PULL_ACK` message is received from server. Since then, mote can send real LoRaWAN messages.

1.3.2 Mote commands

All the supported LoRaWAN uplink message types are as follows:

- *Join Request* ,
- *Confirmed Uplink* ,
- *Unconfirmed Uplink* ,
- *Rejoin Request* ,
- *MACCommand* ,

Downlink message types include

- *Join Accept* ,
- *Confirmed Downlink* ,
- *Unconfirmed Downlink*

Activation

First of all, the mote needs to be activated before using. There are two modes of activations, i.e., Over-the-air Activation (OTAA), and ABP.

OTAA

OTAA stands for the activation by which mote need to negotiate with server to establish session and generate various session keys. To activate mote in OTAA mode, do the following:

1. Register application and mote EUI (JoinEUI and DevEUI), and the two root keys,
2. Modify `device.json` config file,
3. Run `mote pull` ,
4. Run `mote join -n` , where `-n` option means to establish brand new session which may override the old one.
5. When the log shows `INFO - Join Accept (MIC verified)` , it means that a *Join Accept* message is received and the message integrity code (MIC) is verified. Now the mote has been activated, and can be used to send application messages.

ABP

ABP mode means that the session parameters are preset in both server and mote sides, making them able to communicate directly. The process is shown as follows:

1. Set the activation mode to be ABP on LoRaWAN server, and set all the session parameters,
2. Modify `abp.json` file on all fields,
3. Run `mote abp`. If the mote information is printed, the ABP activation succeeds.

Uplink & Downlink Application message

After activation, the mode can send and receive application data. Downlink message can **only** be received after sending an uplink message successfully. To send uplink message, use:

```
mote app [-au] [-f fopts] [msg]
```

where `-a` option means to set the **ACK** flag in uplink package, `-u` option stands for *Unconfirmed Uplink* message type, and `-f fopts` means the package include *FOpts* (MACCommands). `msg` is the actual string that needs to be sent, and it will be encoded by `UTF-8`. For example:

```
mote app -au -f 01 hello_world
```

means send an *Unconfirmed Uplink* message, set the **ACK** flag, and include *FOpts* `0x01` . The application message is the string `hello_world` .

According to different implementation of LoRaWAN server, the downlink messages could be quite different. There may also be no reply (*Unconfirmed Uplink*). `lora-mote-emulator` will wait until timeout.

When a downlink message is received, the command line will display message `INFO - Downlink MACPayload (MIC verified)`, and show some important fields.

Rejoin request

Our program supports to send all three types of *Rejoin Request* (New in LoRaWAN Version 1.1.0), e.g.:

```
mote rejoin {0,1,2}
```

If server agrees the request, it will reply with *Join Accept* message.

MACCommand

`mote` can send *MACCommand* via *FRMPayload* field, i.e. *FPorts = 0*. The command is as follow:

```
mote mac [-au] [cmd]
```

where `-au` act the same as in `app` , and `[cmd]` stands for the actual commands (in hex string form) that needs to be sent. For example, to send `0x01` , use:

```
mote mac 01
```

Check mote information

Use `mote info` to display the information of current mote.

CHAPTER

TWO

2.1

PythonLoRaLoRaWANLoRa

2.2

Python 3.6+ pipenv

1. Python > 3.6pippippip install --upgrade pip
2. pipenv

```
pip install pipenv
```

3.

```
pipenv --python 3
```

4. Pipfile pypi https://pypi.tuna.tsinghua.edu.cn/simple
5.

```
pipenv install lora-mote-emulator
```

2.3

mote mote -h

```
usage: mote [-h] [-v version] [-c CONFIG] [--model MODEL]
    {join,app,pull,mac,rejoin,info,abp,create} ...

Tool to emulate LoRa mote (a.k.a end-device) and Gateway, supported command
list: ['join', 'app', 'pull', 'mac', 'rejoin', 'info', 'abp', 'create']

optional arguments:
    -h, --help              show this help message and exit
    -v version, --version version
                            Choose LoRaWAN version, 1.0.2 or 1.1(default)
```

(continues on next page)

(continued from previous page)

```
    -c CONFIG, --config CONFIG
                          Specify the directory of config files, default './config'
    --model MODEL         Specify the directory to save the model file, default './models
↪'

Supported commands:
    {join,app,pull,mac,rejoin,info,abp,create}
    join                  Send join request.
    app                   Send application data.
    pull                  Send PULL_DATA.
    mac                   Send MACCommand.
    rejoin                Send rejoin request.
    info                  Show information of current mote.
    abp                   Initialize mote in ABP mode.
    create                Handle configurations.
```

2.3.1

1. `mote create [-c config_dir] device.json` , `gateway.json`, `config.json`, `abp.json` `-c ./config`

   2.

      - `config.json` IP
      - `gateway.json` Extended Unique Identifier, EUI
      - `device.json` `DevEUI` , `JoinEUI` `AppKey` , `NwkKey` ;
      - `abp.json` Activation by Personalization, ABP

   3. `PULL_DATA` IP LoRaWAN `PULL_DATA` `PULL_DATA` `PULL_DATA`

```
mote pull
```

   4. `PULL_ACK`

```
INFO - PULL ACK -
```

2.3.2

LoRaWAN

- *Join Request* ,
- *Confirmed Uplink* ,
- *Unconfirmed Uplink* ,
- *Rejoin Request* ,
- *MACCommand* ,

- *Join Accept* ,
- *Confirmed Downlink* ,

- *Unconfirmed Downlink*

Over-the-air Activation, OTAA **ABP**

OTAA

OTAA

1. LoRaWANEUI
2. `device.json`
3. `mote pull`
4. `mote join -n -n`
5. `INFO - Join Accept (MIC verified)` *Join Accept* Message Integrity Code, MIC

ABP

ABP

1. LoRaWANABP
2. `abp.json`
3. `mote abp`

```
mote app [-au] [-f fopts] [msg]
```

`-a` **ACK** `-u` *Unconfirmed Uplink* `-f fopts` *FOpts* MACCommands `msg` `UTF-8`

```
mote app -au -f 01 hello_world
```

Unconfirmed Uplink **ACK** *FOpts* `0x01` `hello_world`

LoRaWAN *Unconfirmed Uplink* `config.json timeout`

`INFO - Downlink MACPayload (MIC verified)`

Rejoin Request LoRaWAN Version 1.1.0

```
mote rejoin {0,1,2}
```

Join Accept

MACCommand

FRMPayload MACCommand FPorts = 0

```
mote mac [-au] [cmd]
```

`-au [cmd]` MACCommand `0x01`

```
mote mac 01
```

```
mote info
```

CHAPTER

THREE

USE CHIRPSTACK AS LORA SERVER

Please refer to the official website for more information. https://www.chirpstack.io/guides/docker-compose/

1. Install `docker`.
2. Clone the repo `https://github.com/brocaar/chirpstack-docker.git`.
3. `docker-compose up`.
4. Open a browser, visit the default application server `https://localhost:8080`.
5. Login with default username and password, both are `admin`.
6. Setup a network-server. The default is `chirpstack-network-server:8000`.
7. Create `Service-profiles`.
8. Create `Device-profiles`.
9. Register a gateway, and fill in a Gateway EUI.
10. Create an application, select a service profile.
11. Click the application name, and create a device belongs to the application.
    - Fill in a Device EUI,
    - Choose a device profile,
    - Optional: uncheck the frame-counter validation for convenient test.
12. After click the CREATE DEVICE button, `NwkKey` and `AppKey` need to be filled (For **LoRaWAN 1.0.2**, only `AppKey` is needed), then the configuration of ChirpStack server is completed.
13. Now, we can use lora-motes-emulator to issue join request in OTAA mode. (This part is also shown in README.rst)
    - Prepare the config files.
      - For **LoRaWAN 1.0.2**, copy the template file `config/device102.yml.tpl` as `config/device.yml`, for **LoRaWAN 1.1**, copy the template file `config/device.yml.tpl` as `config/device.yml`.
      - Modify the `device.yml` file and fill in the information according to the register information at step 8.
      - Copy the `config/config.yml.tpl` as `config/config.yml`, fill in the IP and port information of ChirpStack server (Default port number is 1700).
    - Start the `pipenv` environment by `pipenv shell`.
    - Send a **PULL_DATA** to ChirpStack server by `python main.py pull`.
    - Send a **join request message** to ChirpStack server by `python main.py join`.

- If the **join accept message** is decoded successfully, we can check the device information by `python main.py info`.
- An **Uplink message** can be sent by `python main.py app -m YOUR_MESSAGE`, which can also combine with MAC command by option `-f MAC_COMMAND_ID`.

14. Key Points:

    - The **Uplink data rate index** and the **Channel index** is required to calculate the MIC field (B1 message) in version 1.1.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search