



oneM2M-LoRa Interworking

Yungi Park (yun0429@keti.re.kr)

Korea Electronics Technology Institute

2020.11.13

The project "International Digital Cooperation - ICT Standardisation" is funded by the European Union



Motivation



- LoRa is being widely adopted for IoT and smart city services
- Opensource LoRaWAN S/W implementation is leveraged to have LoRa interworking
- This tutorial evaluates how to connect oneM2M with LoRaWAN network configuration and installation

- Prerequisites
- LoRa Communication and ChirpStack Network Overview
 - LoRa Communication
 - LoRaWAN
 - ChirpStack Network
- ChirpStack-based Device Interworking practice
 - LoRa Gateway Configuration
 - LoRa Server Configuration using ChirpStack
- LoRaWAN– oneM2M IPE Interworking practice
 - LoRaWAN – oneM2M IPE Architecture
 - OneM2M resource design of LoRaWAN
 - IPE working scenario
 - oneM2M-LoRa IPE Overview
 - oneM2M-LoRa IPE Interworking

Prerequisites

- H/W
 - Raspberry PI 3 b+ or 4
 - LoRa Gateway RF Module(RHF0M0301-915MHZ)
- S/W
 - Node.js
 - Visual Studio Code
 - <https://code.visualstudio.com/download>
 - Rspberry PI OS (HypriotOS)
 - <https://github.com/DieterReuter/image-builder-rpi64/releases>
 - LoRa_IPE Source
 - https://github.com/loTKETI/Lora_IPE.git

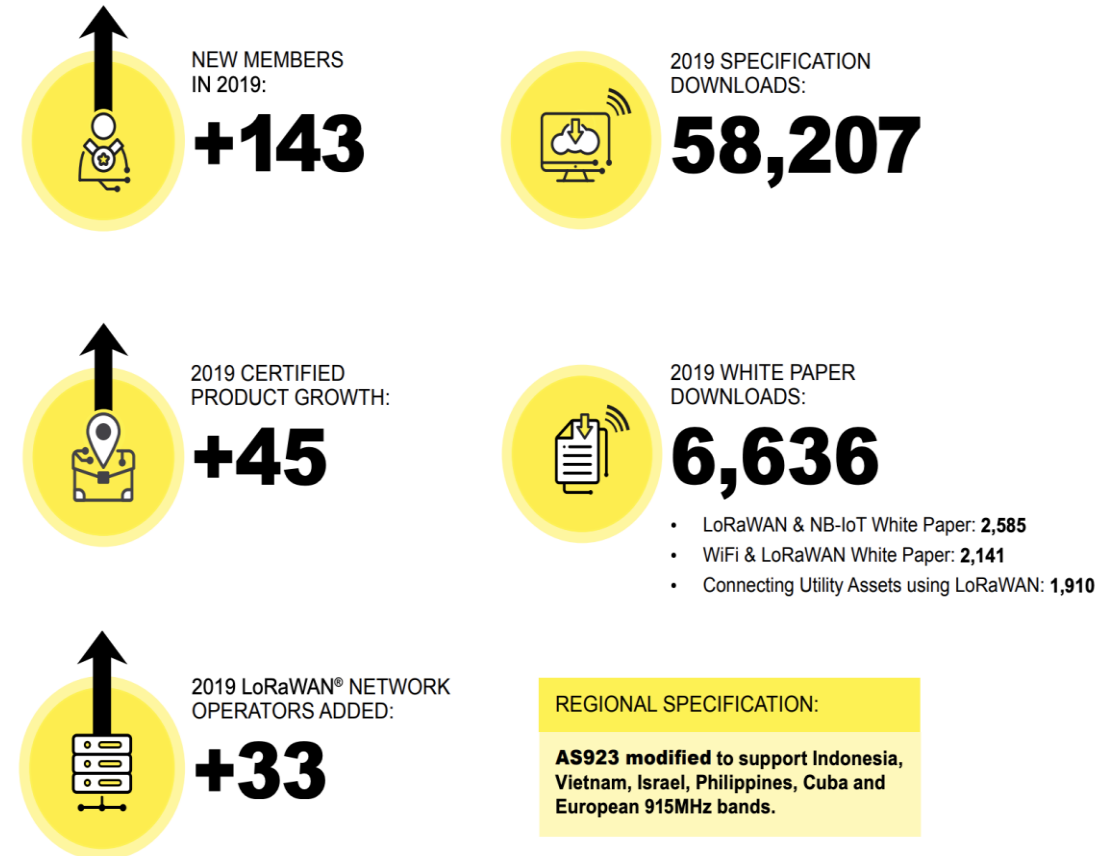


LoRa Communication and ChirpStack Network Overview

LoRa Communication

- LoRa Alliance Overview
 - LoRa Alliance sets LoRaWAN standard
 - Defines physical communication technology and software protocol/architecture for long distance and low power communication
 - More than 400 companies are involved in the alliance
 - Large companies such as Amazon, Alibaba, Cisco, and SemTech participate as sponsors

KEY FACTS & FIGURES



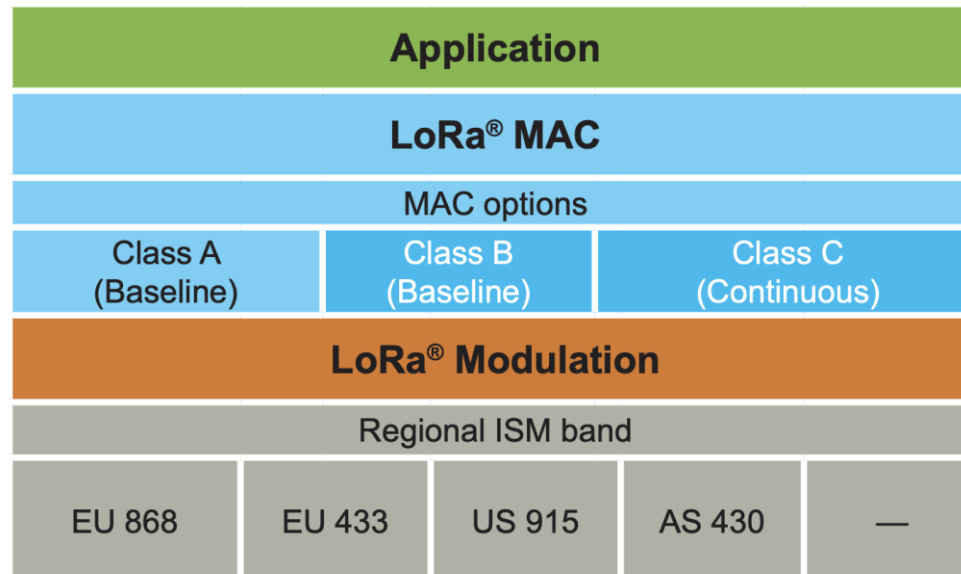
- LoRa technology overview
 - Chirp Spread Spectrum (CSS)-based Spread Spectrum Modulation Technique for Low Power Long Distance Communication
 - Modulation is a technology about how to send and receive digital signals by converting them into analog signals
 - Uses CSS method that supports long distance communication while having efficiency similar to the existing Frequency Shift Keying (FSK) method for low power communication
 - The CSS method has already been used for a long time in the military or space industry, but LoRa is a technology that commercialized it

LoRa Communication

- Communication standards by country

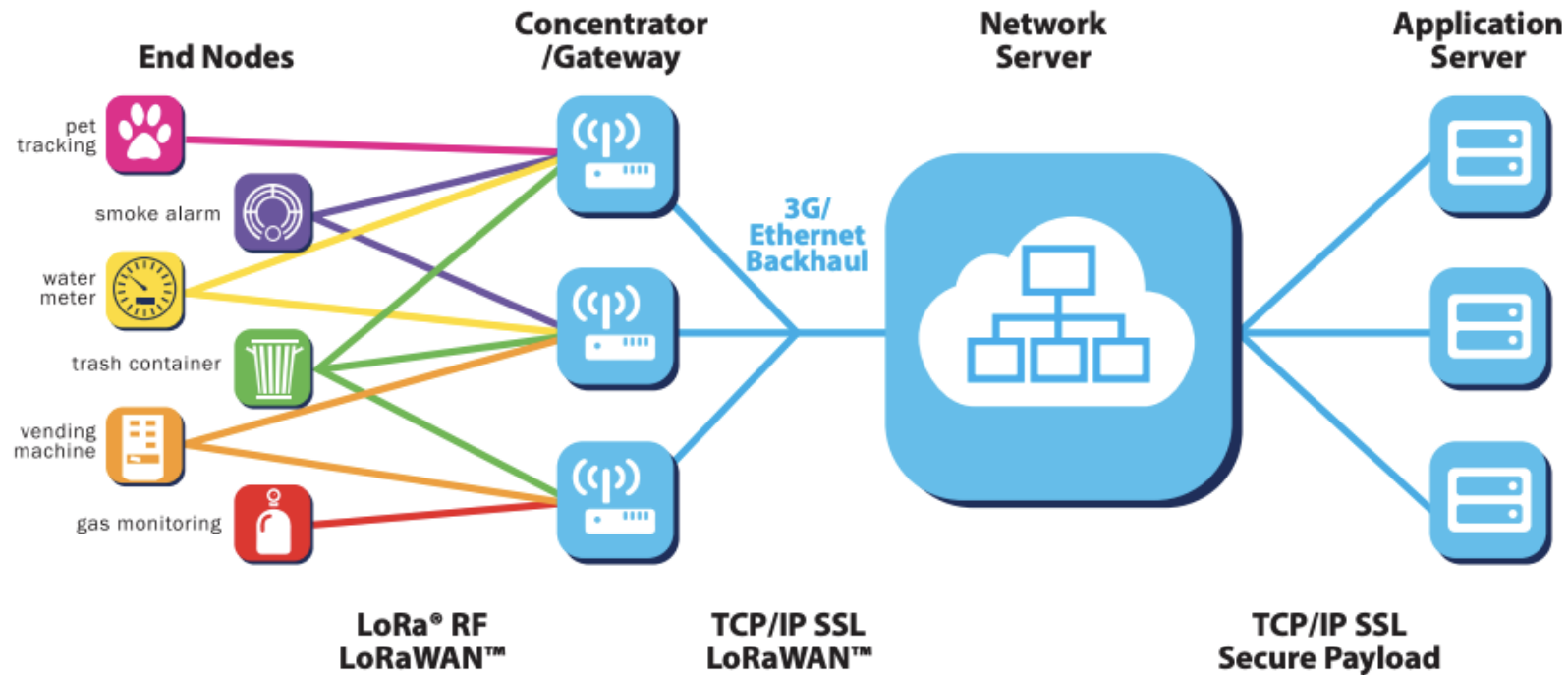
	Modulation	Bandwidth [kHz]	Channel Frequency [MHz]	FSK Bitrate or LoRa DR / Bitrate	NB Channel	Duty Cycle
EU868	LoRa	125	868.1/868.3/868.5	DR0 to DR5 / 0.3-5 Kbps	3	< 1%
US915						
CN779	LoRa	125	779.5/779.7/779.9	DR0 to DR5 / 0.3-5 Kbps	3	< 1%
EU433	LoRa	125	433.175/433.375/433.575	DR0 to DR5 / 0.3-5 Kbps	3	< 1%
AU915						
CN470						
AS923	LoRa	125	923.2~923.4	DR0 to DR5 / 0.3-5 Kbps	2	< 1%
KR920						
IN865	LoRa	125	865.0625/865.4025/865.985	DR0 to DR5 / 0.3-5 Kbps	3	< 1%
RU864	LoRa	125	868.9/869.1	DR0 to DR5 / 0.3-5 Kbps	2	< 1%

- LoRaWAN
 - Defines the protocol and system architecture required for actual communication based on LoRa technology
 - Also includes LoRa communication technology



- LoRaWAN Class
 - Three class definitions in which the data receiving section is set differently according to the requirements of the terminal
 - Class A (basic class, all LoRa terminals must support class A)
 - Data can be received only when the terminal uploads data to the gateway
 - It operates at the lowest power, and it is impossible to receive only without uploading
 - Class B
 - Data can be received at predetermined times other than the data receiving section of Class A
 - Receive beacon messages for time synchronization with the gateway
 - Class C
 - Data can always be received except when uploading data
 - Communication latency is the fastest, but power consumption is high

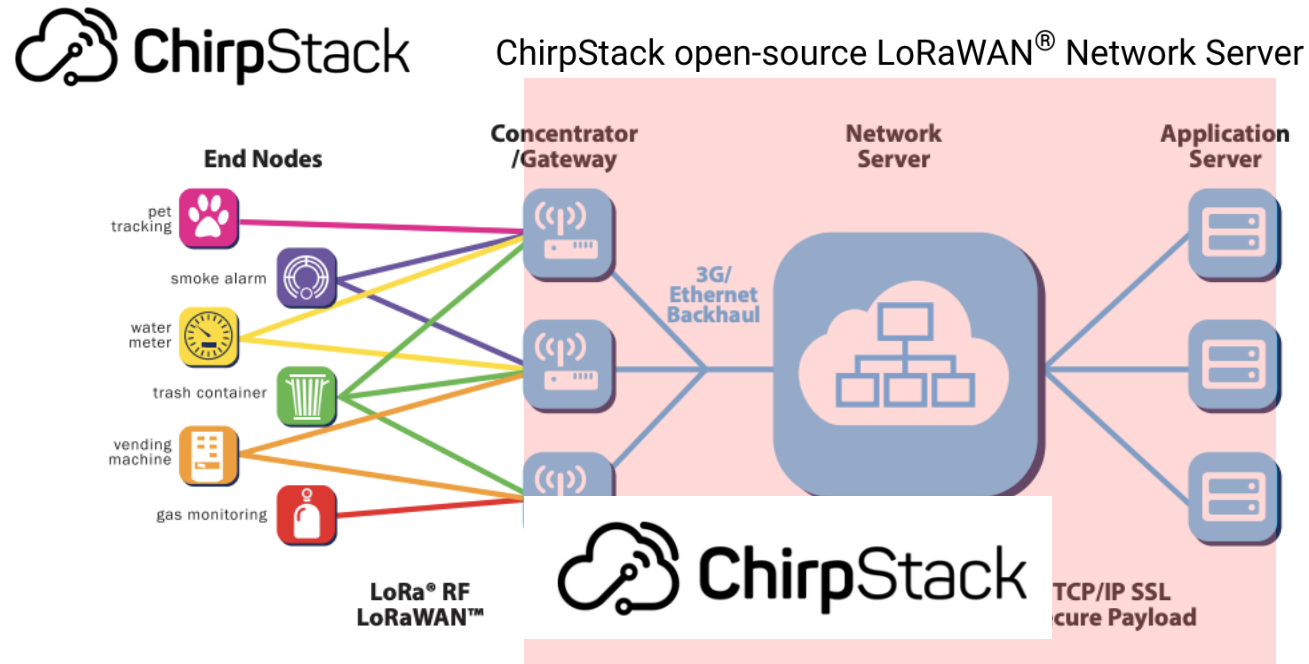
- LoRaWAN Architecture



source: <https://tech-journal.semtech.com/understanding-the-lorawan-architecture>

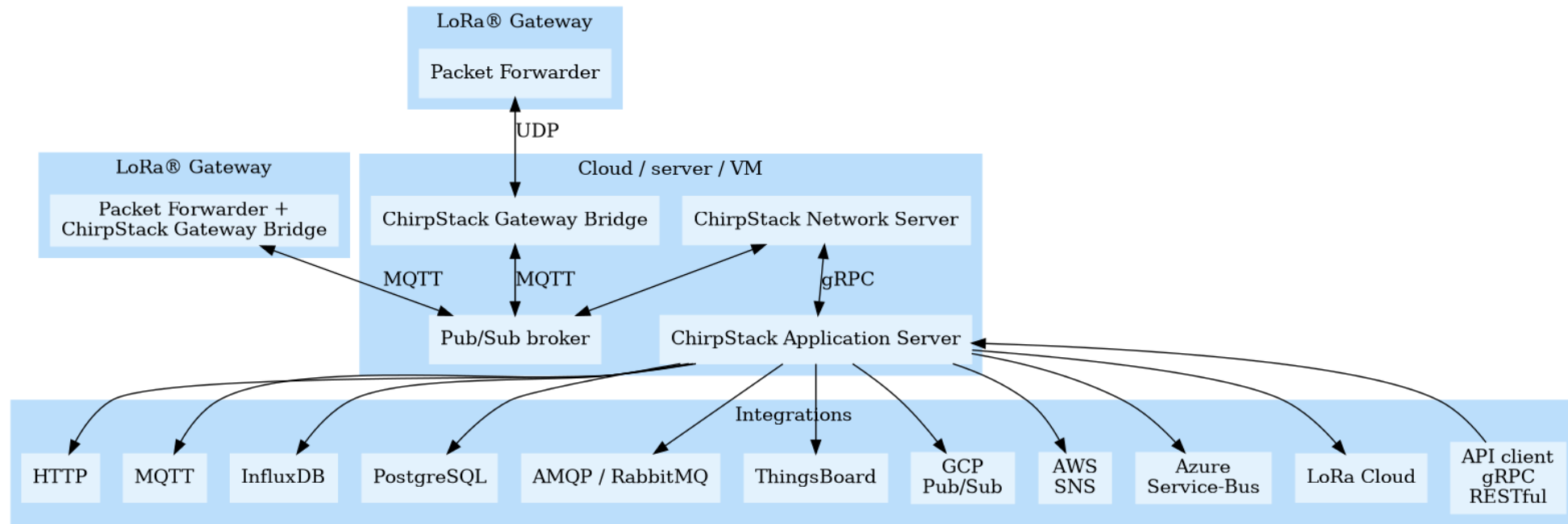
ChirpStack Network

- What is the ChirpStack network stack?
 - Open source platform that supports LoRaWAN network configuration to provide LoRa-based services



ChirpStack Network

- ChirpStack Network Architecture
 - LoRa gateway bridge, LoRa network server, and application server support network configuration required for LoRa-based services



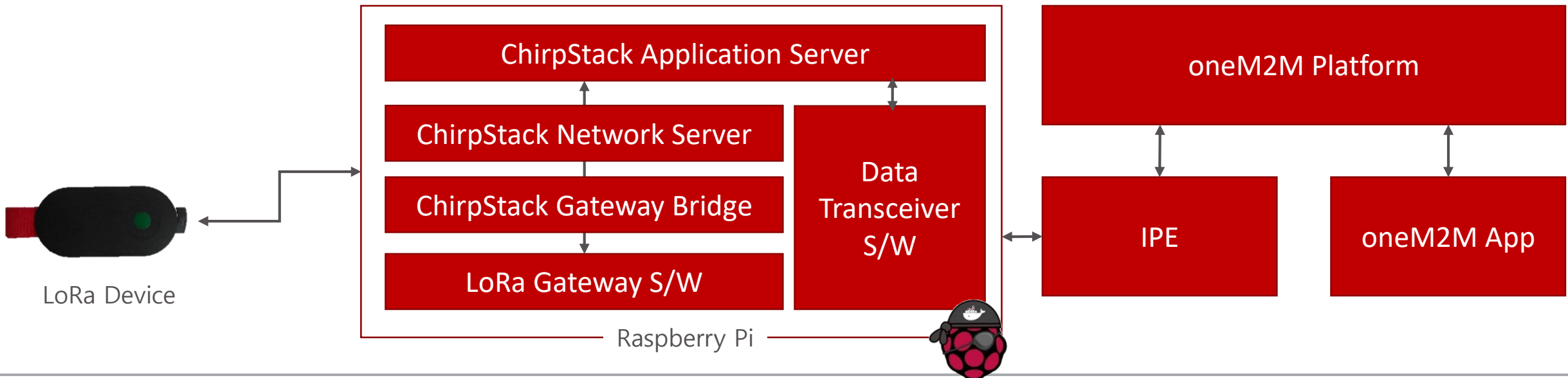
source: <https://www.chirpstack.io/project/architecture/>



ChirpStack-based Device Interworking practice

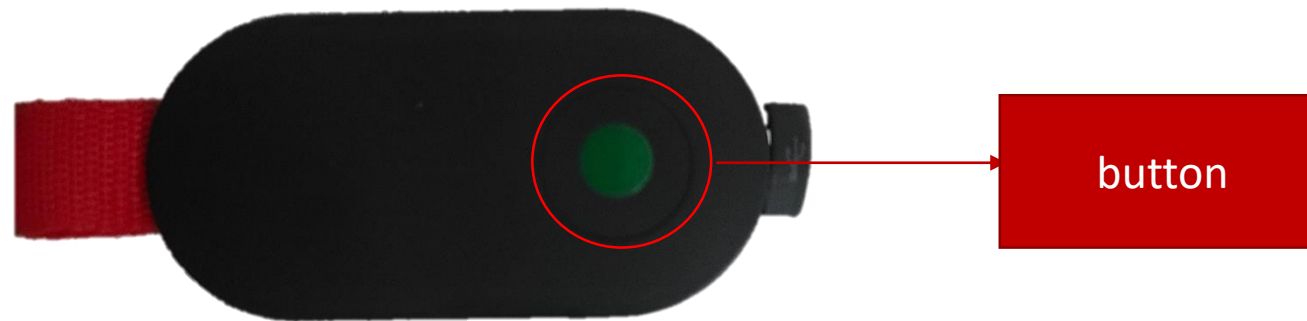
LoRa Gateway Configuration

- ChirpStack-based device integration scenario
 - Configure gateway/server based on ChirpStack on Raspberry Pi
 - Writing LoRa data transmission/reception program based on ChirpStack API on Raspberry Pi
 - Register LoRa terminal to perform data transmission/reception test



LoRa Gateway Configuration

- LoRa Device(GPS Tracker)
 - When the button is pressed, GPS information is transmitted
 - Support LoRaWAN v1.0.1
 - Join Request transmission possible with reset button
 - Supports communication up to several KM
 - GPS Tracker receives the downlink but has no any data to process



LoRa Gateway Configuration

- LoRa Gateway hardware environment
 - IMST IC880a LoRa Gateway board supports 915MHz and operates with the following devices
 - Raspberry Pi 3B+
 - LoRa Gateway RF Module(RHF0M0301-915MHZ)
 - 5V Pin(2ea)
 - 915MHz Antena



LoRa Gateway Configuration

- LoRa Gateway software environment
 - Basic environment configuration of HypriotOS
 - > sudo apt-get update
 - > sudo apt-get install gcc
 - > sudo apt-get install make
 - > sudo apt-get install build-essential
 - > sudo apt-get install git

```
HyprIoT05/arm64: pirate@black-pearl in ~  
$ sudo apt-get update  
Hit:1 http://deb.debian.org/debian stretch InRelease  
Hit:2 http://deb.debian.org/debian stretch-updates InRelease  
Get:3 http://deb.debian.org/debian stretch InRelease [144 kB]  
Fetched 144 kB in 0s (1.1 MB/s)  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
Suggested packages:  
  gettext-base git-daemon-run | git-daemon-sysvinit git-doc git-el git-email git-gui gitk  
  rsync  
Recommended packages:  
  git  
The following packages will be upgraded:  
  git  
1 upgraded, 0 newly installed, 0 to remove and 66 not upgraded.  
Need to get 3,772 kB of archives.  
After this operation, 43.0 kB of additional disk space will be used.
```

LoRa Gateway Configuration

- SPI¹⁾ Interface Activation (1)
 - In order to use the Lora Gateway board, SPI must be activated
 - Install the tool for configuration hypriotOS
 - `cd /tmp`
 - `sudo apt-get install -y alsa-utils`
 - `wget http://archive.raspberrypi.org/debian/pool/main/r/raspi-config/raspi-config_20200226_all.deb`
 - `dpkg -i raspi-config_20200226_all.deb`(If an error occurs, input the following command)
 - `apt --fix-broken install`
 - `sudo apt-get install raspi-config`
 - `Sudo apt-get install raspi-config`

1) Serial Peripheral Interface

LoRa Gateway Configuration

- SPI Interface Activation (2)
 - How to SPI activation
 - > sudo raspi-config
 - [5] Interfacing options -> P4 SPI -> yes

```
Raspberry Pi Software Configuration Tool (raspi-config)

1 Change User Password Change password for the 'pirate' user
2 Network Options       Configure network settings
3 Boot Options          Configure options for start-up
4 Localisation Options  Set up language and regional settings to match your location
5 Interfacing Options   Configure connections to peripherals
6 Overclock             Configure overclocking for your Pi
7 Advanced Options      Configure advanced settings
8 Update               Update this tool to the latest version
9 About raspi-config    Information about this configuration tool

<Select> <Finish>
```

```
Raspberry Pi Software Configuration Tool (raspi-config)

P1 Camera Enable/Disable connection to the Raspberry Pi Camera
P2 SSH     Enable/Disable remote command line access to your Pi using SSH
P3 VNC     Enable/Disable graphical remote access to your Pi using RealVNC
P4 SPI     Enable/Disable automatic loading of SPI kernel module
P5 I2C     Enable/Disable automatic loading of I2C kernel module
P6 Serial  Enable/Disable shell and kernel messages on the serial connection
P7 1-Wire  Enable/Disable one-wire interface
P8 Remote GPIO Enable/Disable remote access to GPIO pins

<Select> <Back>
```

```
Would you like the SPI interface to be enabled?

<Yes> <No>
```

LoRa Gateway Configuration

- LoRa Gateway software install (Native)
 - This LoRa Gateway software is open source and supports ic880a board
 - Install process
 - `git clone -b spi https://github.com/ttn-zh/ic880a-gateway.git ~/ic880a-gateway`
 - `> cd ~/ic880a-gateway`
 - `> sudo ./install.sh`
 - `spi -> N` (user don't need any other setting)
 - After Install gateway software, It will reboot automatically
 - `> sudo ./install.sh spi`
 - Execute the command, Gateway ID is displayed such as B827EBFFFECEB4C9
 - This Gateway ID will be used when registering with the Lora server

```
HyprIoT05/arm64: pirate@black-pearl in ~/ic880a-gateway
$ sudo ./install.sh spi
The Things Network Gateway installer
Version spi
Updating installer files...
Already up-to-date.
Gateway configuration:
Detected EUI B827EBFFFECEB4C9 from eth0
Do you want to use remote settings file? [y/N]
```

LoRa Gateway Configuration

- LoRa Gateway software configuration (1)
 - configuration/rpi_ttn_gateway/local_conf.json
 - Input the gateway_ID

```
{  
    "gateway_conf": {  
        "gateway_ID": "b827ebfffeceb4c9",  
        "keepalive interval": 10,  
    },  
}
```

- This command can change the gateway ID
 - > sudo ./change_gwid.sh start local_conf.json

```
HypriotOS/armv7: pirate@black-pearl in ~/lora_server_docker/configuration/rpi_ttn_gateway  
$ ls  
change_gwid.sh  global_conf.json  local_conf.json
```

```
HypriotOS/armv7: pirate@black-pearl in ~/lora_server_docker/configuration/rpi_ttn_gateway  
$ sudo ./change_gwid.sh  
Usage: ./change_gwid.sh {start|stop} [filename]  
filename: Path to JSON file containing Gateway_ID for packet forwarder
```

LoRa Gateway Configuration

- LoRa Gateway software configuration (2)
 - configuration/rpi_ttn_gateway/global_conf.json
 - Change the 'server_address' to 127.0.0.1
 - Lora device data transmitted to that server
 - serv_port_up is uplink message port and serv_port_down is downlink message port
 - ChirpStack gateway bridge listen to 1700 port of localhost

```
    "gateway_conf": {  
        "server_address": "127.0.0.1",  
        "serv_port_up": 1700,  
        "serv_port_down": 1700  
    }
```

LoRa Server Configuration using ChirpStack

- ChirpStack Install (1)
 - ChirpStack supports Docker image and docker-compose file

```
services:
  chirpstack-network-server:
    image: chirpstack/chirpstack-network-server:3
    volumes:
      - ./configuration/chirpstack-network-server:/etc/chirpstack-network-server

  chirpstack-application-server:
    image: chirpstack/chirpstack-application-server:3
    ports:
      - 8080:8080
    volumes:
      - ./configuration/chirpstack-application-server:/etc/chirpstack-application-server

  chirpstack-gateway-bridge:
    image: chirpstack/chirpstack-gateway-bridge:3
    ports:
      - 1700:1700/udp
    volumes:
      - ./configuration/chirpstack-gateway-bridge:/etc/chirpstack-gateway-bridge

  chirpstack-geolocation-server:
    image: chirpstack/chirpstack-geolocation-server:3
    volumes:
      - ./configuration/chirpstack-geolocation-server:/etc/chirpstack-geolocation-server
```


LoRa Server Configuration using ChirpStack

- ChirpStack Install (2)
 - docker-compose.yml is not include Gateway S/W
 - Therefore, integrate the gateway S/W into the docker-compose.yml file
 - Note the location of the configurations (JSON files below)

```
services:
  rpi_ttn_gateway:
    image: mont08790/rpi-ttn-gateway:latest
    network_mode: host
    privileged: true
    volumes:
      - ./configuration/rpi_ttn_gateway/local_conf.json:/opt/ttn-gateway/bin/local_conf.json
      - ./configuration/rpi_ttn_gateway/global_conf.json:/opt/ttn-gateway/bin/global_conf.json
    command: ./start.sh

  chirpstack-network-server:
    image: ceticasbl/chirpstack-network-server:v3.4.1
    network_mode: host
    depends_on:
      - mosquitto
      - postgresql
      - redis
      - rpi_ttn_gateway
    volumes:
      - ./configuration/chirpstack-network-server:/etc/chirpstack-network-server
```

LoRa Server Configuration using ChirpStack

- ChirpStack Configuration (1)
 - Gateway bridge configuration

```
[integration.mqtt.auth.generic]  
servers=[ "tcp://127.0.0.1:1883"  
username=""  
password=""
```

- Gateway bridge receive the message from 1700 port and transmit localhost using MQTT
- If you want detail, follow the URL(<https://www.chirpstack.io/gateway-bridge/install/config>)

LoRa Server Configuration using ChirpStack

- ChirpStack Configuration (2)
 - Network Server Configuration
 - Input postgres and redis address for data storage
 - Frequency band input
 - In Korea, use 'KR_920_923'
 - Input receive message server at gateway bridge
 - MQTT of localhost

```
[postgresql]
dsn="postgres://loraserver_ns:keti@localhost/loraserver_ns?sslmode=disable"

[redis]
url="redis://localhost:6379"

[network_server]
net_id="000000"

[network_server.band]
name="KR_920_923"

[network_server.network_settings]

[[network_server.network_settings.extra_channels]]
frequency=922100000
min_dr=0
max_dr=5

[[network_server.network_settings.extra_channels]]
frequency=922300000
min_dr=0
max_dr=5

[[network_server.network_settings.extra_channels]]
frequency=922500000
min_dr=0
max_dr=5

[[network_server.network_settings.extra_channels]]
frequency=922700000
min_dr=0
max_dr=5

[[network_server.network_settings.extra_channels]]
frequency=922900000
min_dr=0
max_dr=5

[[network_server.network_settings.extra_channels]]
frequency=923100000
min_dr=0
max_dr=5

[[network_server.network_settings.extra_channels]]
frequency=923300000
min_dr=0
max_dr=5

[network_server.gateway.backend.mqtt]
server="tcp://localhost:1883"

[join_server.default]
#server="http://chirpstack-application-server:8003"
server="http://localhost:8003"
```

LoRa Server Configuration using ChirpStack

- ChirpStack Configuration (3)
 - Application server configuration
 - JWT key (jwt key is using for application server and security)
 - Get new JWT key using the command

```
pirate@ttn-gateway:~/lora_server_docker$ openssl rand -base64 32  
7k09aebEm5i3HRChClJuXnPn9iKzjplf6HguYfn1NQw=
```

- New JWT key input application server config file

```
[application_server.external_api]  
bind="0.0.0.0:8080"  
jwt_secret="7k09aebEm5i3HRChClJuXnPn9iKzjplf6HguYfn1NQw="
```

LoRa Server Configuration using ChirpStack

- ChirpStack Application server configuration (1)
 - Run LoRa and access app server([http://\[ChirpStack IP\]:8080](http://[ChirpStack IP]:8080)) using command
 - > docker-compose up -d
 - Add Network-servers and input network server name
 - This practice don't use Gateway Discovery and TLS Certificates



Network-servers / Add

GENERAL GATEWAY DISCOVERY TLS CERTIFICATES

Network-server name *

KETINetworkServer

A name to identify the network-server.

Network-server server *

localhost:8000

The 'hostname:port' of the network-server, e.g. 'localhost:8000'.

Network-servers

Name	Server
KETINetworkServer	localhost:8000

LoRa Server Configuration using ChirpStack

- ChirpStack Application server configuration (2)
 - Add profile from Gateway-profiles
 - KR920-923 frequency band use 3 channel
 - Input 0, 1, 2 at Enabled channels
 - Input the network server name

Modulation	Bandwidth [kHz]	Channel Frequency [MHz]	FSK Bitrate or LoRa DR / Bitrate	Nb Channels
LoRa	125	922.10 922.30 922.50	DR0 to DR5 / 0.3-5 kbps	3

< Lora frequency band table >

Gateway-profiles / Create

Name *

KETIGatewayProfile

A short name identifying the gateway-profile.

Enabled channels *

0, 1, 2

The channels active in this gateway-profile as specified in

Network-server *

KETINetworkServer

LoRa Server Configuration using ChirpStack

- ChirpStack Application server configuration (3)

- Add profile from Service-profiles

- Minimum/Maximum allowed data-rate

- Define LoRaWAN Standard table

- Input 0, 5

DataRate	Configuration	Indicative physical bit rate [bit/s]
0	LoRa: SF12 / 125 kHz	250
1	LoRa: SF11 / 125 kHz	440
2	LoRa: SF10 / 125 kHz	980
3	LoRa: SF9 / 125 kHz	1760
4	LoRa: SF8 / 125 kHz	3125
5	LoRa: SF7 / 125 kHz	5470
6..14	RFU	
15	Defined in LoRAWAN ³⁹	

< Lora Standard table >

Service-profiles / Create

Service-profile name *

KETIServiceProfile

A name to identify the service-profile.

Network-server *

KETINetworkServer

The network-server on which this service-profile will be pr

☐ Add gateway meta-data

GW metadata (RSSI, SNR, GW geoloc., etc.) are added to tl

☐ Enable network geolocation

When enabled, the network-server will try to resolve the lo

Device-status request frequency

00

Frequency to initiate an End-Device status request (reques

Minimum allowed data-rate *

0

Minimum allowed data rate. Used for ADR.

Maximum allowed data-rate *

5

Maximum allowed data rate. Used for ADR.

LoRa Server Configuration using ChirpStack

- ChirpStack Application server configuration (4)
 - Add profile from Service-profiles
 - LoRa version is 1.0.0
 - Regional Parameter is B
 - Max EIRP follow the standard → 14
 - Check 'Device supports OTAA

TXPower	Configuration (EIRP)
0	Max EIRP
1	Max EIRP – 2dB
2	Max EIRP – 4dB
3	Max EIRP – 6dB
4	Max EIRP – 8dB
5	Max EIRP – 10dB
6	Max EIRP – 12dB
7	Max EIRP – 14dB
8..14	RFU
15	Defined in LoRAWAN ³⁹

< Lora Max EIRP table >

Device-profiles / Create

GENERAL JOIN (OTAA / ABP)

Device-profile name *

KETIDeviceProfile

A name to identify the device-profile.

Network-server *

KETINetworkServer

The network-server on which this device-profile will be

LoRaWAN MAC version *

1.0.0

The LoRaWAN MAC version supported by the device.

LoRaWAN Regional Parameters revision *

B

Revision of the Regional Parameters specification supp

Max EIRP *

14

Maximum EIRP supported by the device.

Geolocation buffer TTL (seconds)

0

The time in seconds that historical uplinks will be store

Geolocation minimum buffer size

0

The minimum buffer size required before using geoloci

Device-profiles / Create

GENERAL JOIN (OTAA / ABP)

☒ Device supports OTAA

LoRa Server Configuration using ChirpStack

- ChirpStack Application server configuration (5)
 - Add gateway from Gateways
 - Gateway_ID is the Gateway ID used when setting the gateway S/W

```
{  
  "gateway_conf": {  
    "gateway_ID": "b827ebfffeceb4c9",  
    "keepalive interval": 10,  
  }  
}
```

- Other parameters can find in the list

Gateways / Create

Gateway name *

KETIGateway

The name may only contain words, numbers and

Gateway description *

test gateway

Gateway ID *

B8 27 EB FF FE CE B4 C9

Network-server *

KETINetworkServer

Select the network-server to which the gateway

Gateway-profile

KETIGatewayProfile

An optional gateway-profile which can be assign

LoRa Server Configuration using ChirpStack

- ChirpStack Application server configuration (6)
 - Add application from Applications
 - Service-profile can find in the list

Applications / Create

Application name *

KETIApplication

The name may only contain words, numbers

Application description *

test

Service-profile *

KETIServiceProfile

The service-profile to which this application

Payload codec

None

By defining a payload codec, ChirpStack App. device-profile have priority over the applicati

LoRa Server Configuration using ChirpStack

- ChirpStack Application server configuration (7)
 - Choose the **KETIApplication** on the Applications list and add the device
 - Device EUI is 00 00 00 00 00 00 02 XX(GPS Tracker EUI)
 - XX can find from the back of device
 - Device-profile can find the list

Applications / KETIApplication / Devices / Create

GENERAL	VARIABLES	TAGS
<p>Device name *</p> <p>KETIDevice</p> <p>The name may only contain words, numbers and dashes.</p>		
<p>Device description *</p> <p>test</p>		
<p>Device EUI *</p> <p>00 00 00 00 00 00 02 13</p>		
<p>Device-profile *</p> <p>KETIDeviceProfile</p>		
<p><input type="checkbox"/> Disable frame-counter validation</p> <p>Note that disabling the frame-counter validation will compromise security as it enable</p>		

LoRa Server Configuration using ChirpStack

- ChirpStack Application server configuration (8)
 - KETIDevice configuration
 - Input Application Key for OTAA
 - That key value can find device data sheet
 - GPS Tracker application key(1234567890abcdef1234567890abcdef)
 - Ignore Gen Application Key

Applications / KETIApplication / Devices / KETIDevice

DETAILS CONFIGURATION KEYS (OTAA)

Application key *

12 34 56 78 90 ab cd ef 12 34 56 78 90 ab cd ef

For LoRaWAN 1.0 devices. In case your device supports LoRaWAN 1.1, update the device

Gen Application key

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

For LoRaWAN 1.0 devices. This key must only be set when the device implements the 1

LoRa Server Configuration using ChirpStack

- ChirpStack Application server configuration (9)
 - The Frequency of KR920-923 follow this table

Center frequency (MHz)	Bandwidth (kHz)	Maximum EIRP output power (dBm)	
		For end-device	For gateway
920.9	125	10	23
921.1	125	10	23
921.3	125	10	23
921.5	125	10	23
921.7	125	10	23
921.9	125	10	23
922.1	125	14	23
922.3	125	14	23
922.5	125	14	23
922.7	125	14	23
922.9	125	14	23
923.1	125	14	23
923.3	125	14	23

LoRa Server Configuration using ChirpStack

- How to use ChirpStack REST API (1)
 - `http://[ChirpStack IP]:8080/api`

ChirpStack Application Server REST API

ChirpStack Application Server REST API

For more information about the usage of the ChirpStack Application Server (REST) API, see <https://www.chirpstack.io/application-server/api/>.

ApplicationService

[Show/Hide](#) | [List Operations](#) | [Expand Operations](#)

GET	/api/applications	List lists the available applications.
POST	/api/applications	Create creates the given application.
PUT	/api/applications/{application.id}	Update updates the given application.
GET	/api/applications/{application_id}/integrations	ListIntegrations lists all configured integrations.
DELETE	/api/applications/{application_id}/integrations/http	DeleteIntegration deletes the HTTP application-integration.
GET	/api/applications/{application_id}/integrations/http	GetHTTPIntegration returns the HTTP application-integration.
DELETE	/api/applications/{application_id}/integrations/influxdb	DeleteInfluxDBIntegration deletes the InfluxDB application-integration.
GET	/api/applications/{application_id}/integrations/influxdb	GetInfluxDBIntegration returns the InfluxDB application-integration.
DELETE	/api/applications/{application_id}/integrations/thingsboard	DeleteThingsBoardIntegration deletes the ThingsBoard application-integration.
GET	/api/applications/{application_id}/integrations/thingsboard	GetThingsBoardIntegration returns the ThingsBoard application-integration.
DELETE	/api/applications/{id}	Delete deletes the given application.
GET	/api/applications/{id}	Get returns the requested application.
POST	/api/applications/{integration.application_id}/integrations/http	CreateHTTPIntegration creates a HTTP application-integration.
PUT	/api/applications/{integration.application_id}/integrations/http	UpdateHTTPIntegration updates the HTTP application-integration.
POST	/api/applications/{integration.application_id}/integrations/influxdb	CreateInfluxDBIntegration create an InfluxDB application-integration.

LoRa Server Configuration using ChirpStack

- How to use ChirpStack REST API (2)
 - For authorization, call the Login api and check the JWT Token

POST /api/internal/login Log in a user

Response Class (Status 200)
A successful response.

Model Example Value

```
{
  "jwt": "string"
}
```

Response Content Type

Parameters

Parameter	Value	Description	Parameter Type	Data Type
body	<pre>{ "password": "admin", "username": "admin" }</pre>		body	Model Example Value

Model Example Value

```
{
  "password": "string",
  "username": "string"
}
```

Response Body

```
{
  "jwt": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOiJjaGlycHN0YWNRWFWcGxpY2F0aW9uLXNlcnZlciIsImV4cCI6MTU5NzY3M"
}
```

LoRa Server Configuration using ChirpStack

- How to use ChirpStack REST API (3)
 - Input JWT TOKEN at upper right of API webpage

The image shows a comparison of the ChirpStack Application Server REST API interface before and after a JWT token is entered. A large red arrow points from the 'JWT TOKEN' input field in the top screenshot to the token value in the bottom screenshot.

Top Screenshot: The header bar is green with the text 'ChirpStack Application Server REST API' on the left and a white input field labeled 'JWT TOKEN' on the right. Below the header, the page title is 'ChirpStack Application Server REST API', followed by a link to the API documentation. The 'ApplicationService' section shows a 'GET /api/applications' endpoint with a description 'List the available applications.' and links for 'Show/Hide', 'List Operations', and 'Expand Operations'.

Bottom Screenshot: The header bar is green with the text 'ChirpStack Application Server REST API' on the left and a white input field containing the JWT token 'eyJhbGciOiJIUzI1NiIsInR5cCI6I...' on the right. The rest of the page content is identical to the top screenshot.

LoRa Server Configuration using ChirpStack

- How to use ChirpStack REST API (4)
 - ex: get device Information
 - GET /api/devices/
 - Example Value
 - Input Parameters
 - Click Try it out! button

GET /api/devices List returns the available devices.

Response Class (Status 200)
A successful response.

Model Example Value

```
{
  "result": [
    {
      "applicationID": "string",
      "description": "string",
      "devEUI": "string",
      "deviceProfileID": "string",
      "deviceProfileName": "string",
      "deviceStatusBattery": 0,
      "deviceStatusBatteryLevel": 0,
      "deviceStatusBatteryLevelAvailable": true
    }
  ]
}
```

Response Content Type

Parameters

Parameter	Value	Description	Parameter Type	Data Type
limit	<input type="text" value="1"/>	Max number of devices to return in the result-set.	query	string
offset	<input type="text"/>	Offset in the result-set (for pagination).	query	string
applicationID	<input type="text"/>	Application ID to filter on.	query	string
search	<input type="text"/>	Search on name or DevEUI.	query	string
multicastGroupID	<input type="text"/>	Multicast-group ID to filter on (string formatted UUID).	query	string
serviceProfileID	<input type="text"/>	Service-profile ID to filter on (string formatted UUID).	query	string

[Hide Response](#)

LoRa Server Configuration using ChirpStack

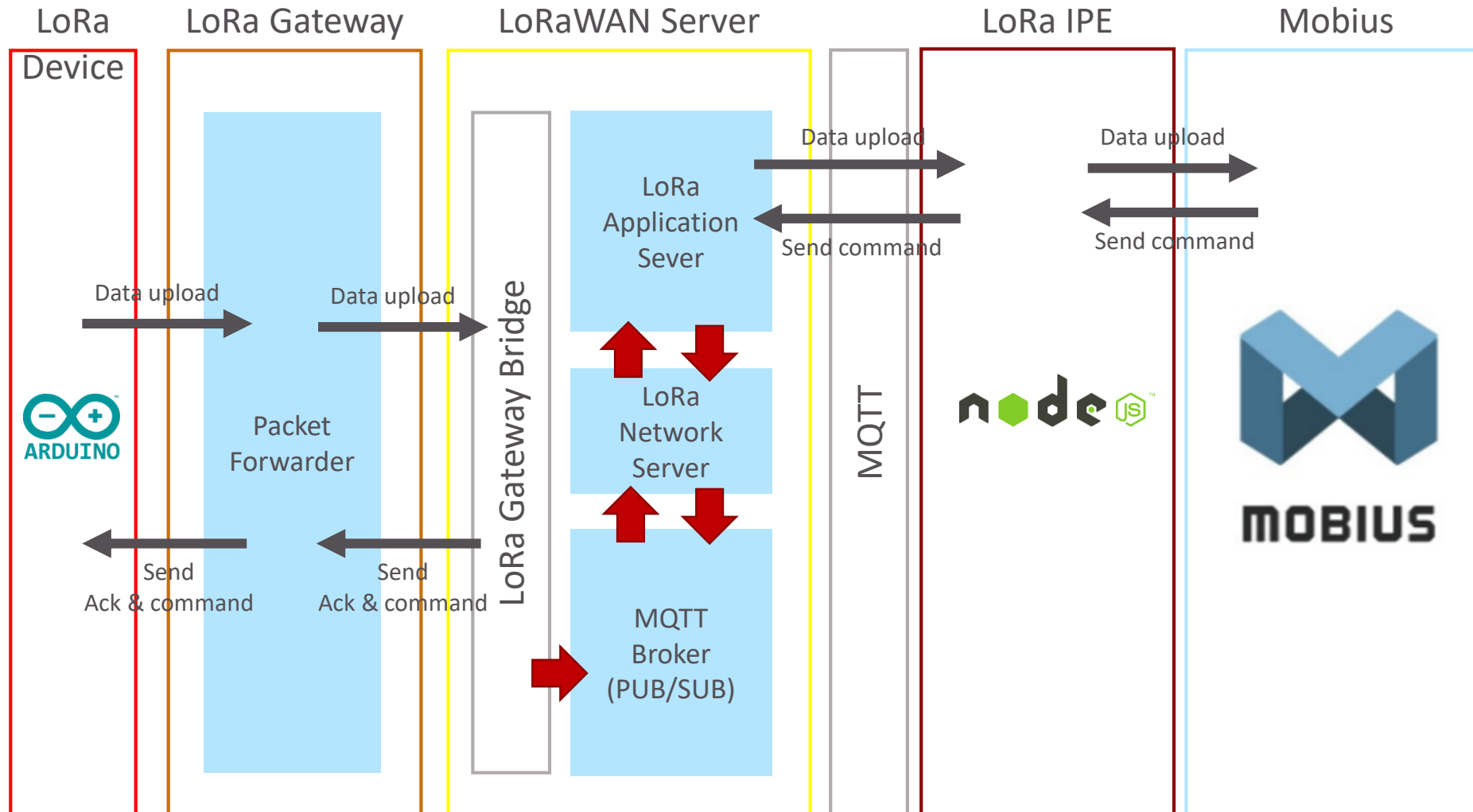
- How to use ChirpStack REST API (5)
 - ex: get device Information
 - Check Curl command
 - Request URL
 - Check Response Body
 - You can find Device information such as devEUI and name

```
Curl
curl -X GET --header 'Accept: application/json' --header 'Grpc-Metadata-Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5c
Request URL
http://172.30.1.6:8080/api/devices?limit=1
Response Body
{
  "totalCount": "1",
  "result": [
    {
      "devEUI": "0000000000000213",
      "name": "KETIDevice",
      "applicationID": "1",
      "description": "test",
      "deviceProfileID": "081f9c0a-38f1-432f-91be-f9a311b78878",
      "deviceProfileName": "KETIDeviceProfile",
      "deviceStatusBattery": 255,
      "deviceStatusMargin": 256,
      "deviceStatusExternalPowerSource": false,
      "deviceStatusBatteryLevelUnavailable": true,
      "deviceStatusBatteryLevel": 0,
      "lastSeenAt": null
    }
  ]
}
Response Code
200
Response Headers
{
  "content-length": "398",
  "content-type": "application/json",
  "date": "Sun, 16 Aug 2020 13:19:12 GMT",
  "grpc-metadata-content-type": "application/grpc",
  "grpc-metadata-ctx-id": "c3798c89-20c9-46a2-aea0-03d5abdf4459",
  "grpc-metadata-trailer": "Grpc-Status, Grpc-Message, Grpc-Status-Details-Bin"
}
```



LoRa – oneM2M IPE development practice

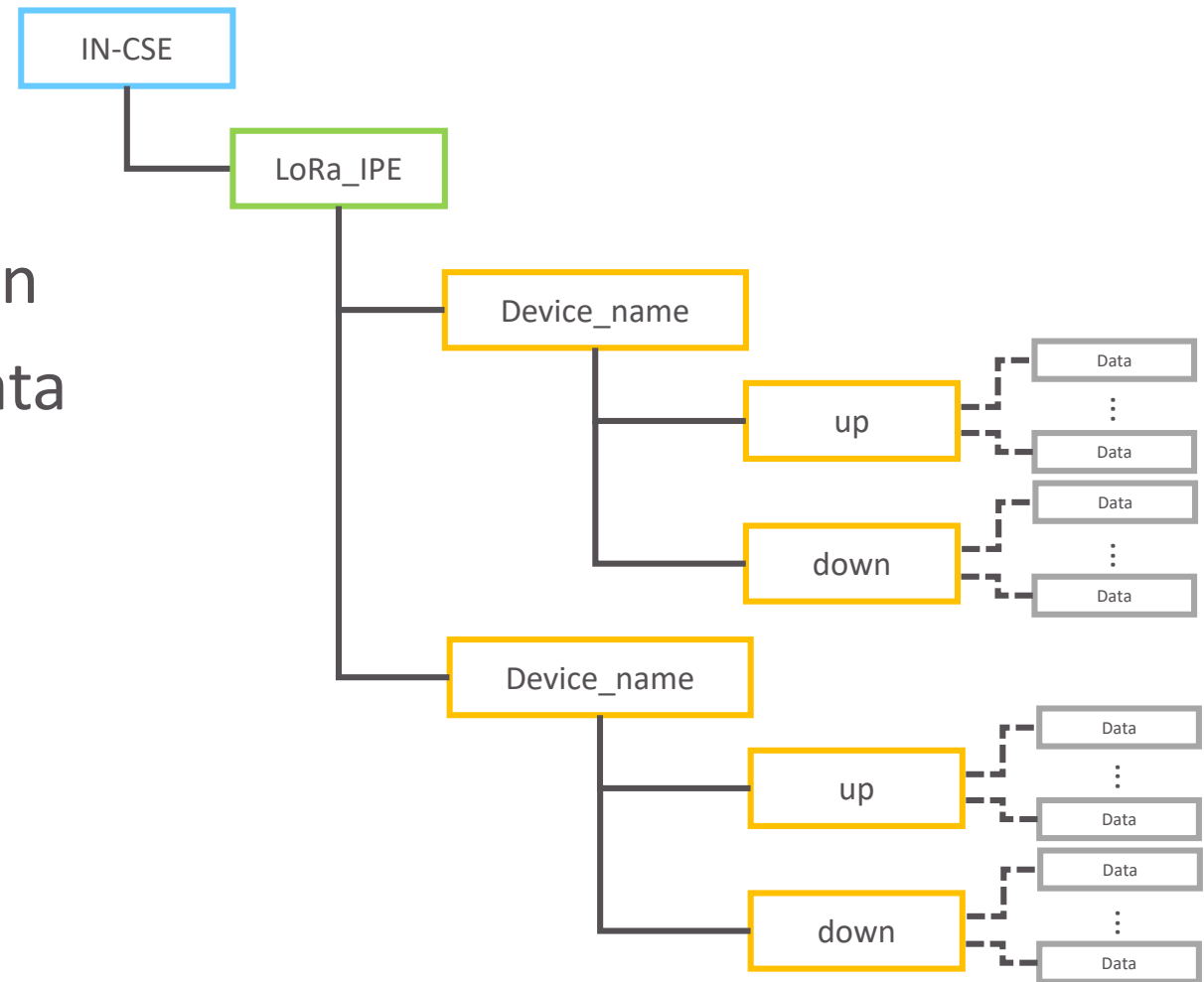
LoRa – oneM2M IPE Architecture



oneM2M resource design of LoRa network



- CSE => Mobius
- AE => LoRa_IPE(Name)
- Container => Device_name,up,down
- ContentInstance => LoRa Device Data



IPE working scenario

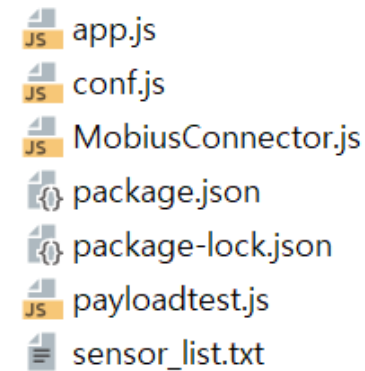


- Uplink
 - First, IPE call resource creation request in oneM2M platform using API
 - Next, if the resource has been deleted, create it again (e.g. subscription)
 - IPE access to MQTT server and register topic for receive any event
 - IPE maps the data received from LoRa server, convert to oneM2M API, and saves it in the platform
- Downlink
 - The timing at which Downlink messages are delivered is different for each LoRa communication Class
 - IPE Downlink supports two ways
 - The first method is to deliver a downlink message when an uplink occurs in the LoRa device
 - The second method is to immediately receive and process downlink from the device when a downlink occurs

oneM2M-LoRa IPE Overview



- oneM2M-LoRa IPE S/W setting
 - LoRa IPE working the role of delivering the data received from the lora server to the oneM2M-based platform
 - https://github.com/loTKETI/Lora_IPE.git
- oneM2M-LoRa IPE configuration
 - app.js : Main
 - conf.js : IPE host and ae name setting
 - Sensor list : Lora Device EUI list
 - MobiusConnector.js : Mobius API
 - Package.json : List of node module required when the IPE working



```
[ 702c1ffffe370001,
  702c1ffffe3cd70c,
  702c1ffffe3cd80b,
  702c1ffffe3cd815,
  702c1ffffe4a2c9e ]
create ae: POST -> http://203.253.128.161:7579/Mobius
create ae: 409 <- {"m2m:dbg":"resource is already exist"}
{ code: 409, body: {"m2m:dbg":"resource is already exist"} }
create cnt: POST -> http://203.253.128.161:7579/Mobius/lora_test
create cnt: 409 <- {"m2m:dbg":"resource is already exist"}
create cnt: POST -> http://203.253.128.161:7579/Mobius/lora_test/702c1ffffe370001
create cnt: 409 <- {"m2m:dbg":"resource is already exist"}
create cnt: POST -> http://203.253.128.161:7579/Mobius/lora_test/702c1ffffe370001
create cnt: 409 <- {"m2m:dbg":"resource is already exist"}
702c1ffffe370001 CNT_Complete!!
create cnt: POST -> http://203.253.128.161:7579/Mobius/lora_test
create cnt: 409 <- {"m2m:dbg":"resource is already exist"}
create cnt: POST -> http://203.253.128.161:7579/Mobius/lora_test/702c1ffffe3cd70c
create cnt: 409 <- {"m2m:dbg":"resource is already exist"}
create cnt: POST -> http://203.253.128.161:7579/Mobius/lora_test/702c1ffffe3cd70c
create cnt: 409 <- {"m2m:dbg":"resource is already exist"}
702c1ffffe3cd70c CNT_Complete!!
create cnt: POST -> http://203.253.128.161:7579/Mobius/lora_test
create cnt: 409 <- {"m2m:dbg":"resource is already exist"}
create cnt: POST -> http://203.253.128.161:7579/Mobius/lora_test/702c1ffffe3cd80b
create cnt: 409 <- {"m2m:dbg":"resource is already exist"}
create cnt: POST -> http://203.253.128.161:7579/Mobius/lora_test/702c1ffffe3cd80b
create cnt: 409 <- {"m2m:dbg":"resource is already exist"}
702c1ffffe3cd80b CNT_Complete!!
create cnt: POST -> http://203.253.128.161:7579/Mobius/lora_test
create cnt: 409 <- {"m2m:dbg":"resource is already exist"}
create cnt: POST -> http://203.253.128.161:7579/Mobius/lora_test/702c1ffffe3cd815
create cnt: 409 <- {"m2m:dbg":"resource is already exist"}
create cnt: POST -> http://203.253.128.161:7579/Mobius/lora_test/702c1ffffe3cd815
create cnt: 409 <- {"m2m:dbg":"resource is already exist"}
702c1ffffe3cd815 CNT_Complete!!
create cnt: POST -> http://203.253.128.161:7579/Mobius/lora_test
create cnt: 409 <- {"m2m:dbg":"resource is already exist"}
create cnt: POST -> http://203.253.128.161:7579/Mobius/lora_test/702c1ffffe4a2c9e
create cnt: 409 <- {"m2m:dbg":"resource is already exist"}
create cnt: POST -> http://203.253.128.161:7579/Mobius/lora_test/702c1ffffe4a2c9e
create cnt: 409 <- {"m2m:dbg":"resource is already exist"}
702c1ffffe4a2c9e CNT_Complete!!
```

oneM2M-LoRa IPE Overview



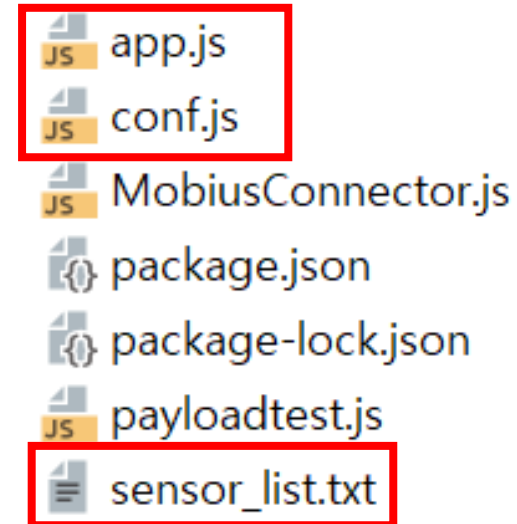
- oneM2M-LoRa IPE S/W setting
 - In conf.js file set CSE address, port, AE name, LoRa Server address and port base on oneM2M
 - CSE set onem2m platform information and AE set IPE information for connect with LoRa server

```
var conf = {};  
var cse = {};  
var ae = {};  
var lora = {};  
  
//cse config  
cse.host = "203.253.128.161";  
cse.port = "7579";  
cse.name = "Mobius";  
cse.id = "/Mobius2";  
cse.mqttport = "1883";  
  
//ae config  
ae.name = "keti_tracker";  
ae.id = "S" + ae.name;  
ae.parent = "/" + cse.name;  
ae.appid = "lora"  
  
//lora config  
lora.host = "203.253.128.164";  
lora.mqttport = "1883";  
  
conf.cse = cse;  
conf.ae = ae;  
conf.lora = lora;  
  
module.exports = conf;
```


oneM2M-LoRa IPE Overview



- oneM2M-LoRa IPE S/W modifications
 - S/W modifiable list
 - App.js : User can modify Interworking function with oneM2M platform and add other functions
 - Sensor list : Add LoRa Device



```
1 {"id" : "702c1ffffe370001"},  
2 {"id" : "702c1ffffe3cd70c"},  
3 {"id" : "702c1ffffe3cd80b"},  
4 {"id" : "702c1ffffe3cd815"},  
5 {"id" : "702c1ffffe4a2c9e"}]
```

Installation oneM2M-LoRa IPE



- Install required node modules
 - Install the required node modules using the following command:
 - > npm install

```
C:\Users\Parkyun\Desktop\Lora_IPE-master (lore_ipe@1.0.0)
λ npm install freeboard-master

> jsonpath@1.0.2 postinstall C:\Users\Parkyun\Desktop\Lora_IPE-master\node_modules\jsonpath
> node lib/aesprim.js > generated/aesprim-browser.js

npm WARN lore_ipe@1.0.0 No repository field.

added 194 packages from 201 contributors and audited 194 packages in 5.31s

16 packages are looking for funding
  run `npm fund` for details

found 1 high severity vulnerability
  run `npm audit fix` to fix them, or `npm audit` for details
```

oneM2M-LoRa IPE Interworking



- Start oneM2M-LoRa IPE
 - Start using the following command:
 - > node app.js

```
C:\Users\Parkyun\Desktop\Lora_IPE-master (lore_ipe@1.0.0)
λ node app.js
[ '000000000000213' ]
create ae: POST -> http://203.253.128.161:7579/Mobius
create ae: 409 <- {"m2m:dbg":"resource is already exist"}
{ code: 409, body: '{"m2m:dbg":"resource is already exist"}' }
create cnt: POST -> http://203.253.128.161:7579/Mobius/keti_tracker
create cnt: 409 <- {"m2m:dbg":"resource is already exist"}
create cnt: POST -> http://203.253.128.161:7579/Mobius/keti_tracker/000000000000213
create cnt: 409 <- {"m2m:dbg":"resource is already exist"}
create cnt: POST -> http://203.253.128.161:7579/Mobius/keti_tracker/000000000000213
create cnt: 409 <- {"m2m:dbg":"resource is already exist"}
000000000000213 CNT_Complete!!
retrieve sub: GET -> http://203.253.128.161:7579/Mobius/keti_tracker/sub_container_check
retrieve sub: 200 <- {"m2m:sub":{"pi":"SUxy2HYhSZja","ri":"23-20200924060756004","ty":23,"ct":"20200924T060756","rn":"sub_contai
56","et":"20220924T060756","enc":{"net":[4],"exc":100,"nu":["mqtt://203.253.128.161/Sketi_tracker?ct=json"],"nct":2,"cr":"S"}}}
delete resc: DELETE -> http://203.253.128.161:7579/Mobius/keti_tracker/sub_container_check
delete resc: 200 <- {"m2m:sub":{"pi":"SUxy2HYhSZja","ri":"23-20200924060756004","ty":23,"ct":"20200924T060756","rn":"sub_contain
6","et":"20220924T060756","enc":{"net":[4],"exc":100,"nu":["mqtt://203.253.128.161/Sketi_tracker?ct=json"],"nct":2,"cr":"S"}}}
create sub: POST -> http://203.253.128.161:7579/Mobius/keti_tracker
create sub: 201 <- {"m2m:sub":{"rn":"sub_container_check","ty":23,"pi":"SUxy2HYhSZja","ri":"23-20200924062530551","ct":"20200924
","et":"20220924T062530","nu":["mqtt://203.253.128.161/Sketi_tracker?ct=json"],"enc":{"net":[4],"exc":100,"nct":2,"cr":"S"}}}
retrieve sub: GET -> http://203.253.128.161:7579/Mobius/keti_tracker/000000000000213/down/sub_downlink
retrieve sub: 200 <- {"m2m:sub":{"pi":"3-20200924060123628","ri":"23-20200924060756385","ty":23,"ct":"20200924T060756","rn":"sub
56","et":"20220924T060756","enc":{"net":[3],"exc":100,"nu":["mqtt://203.253.128.161/Sketi_tracker?ct=json"],"nct":2,"cr":"S"}}}
delete resc: DELETE -> http://203.253.128.161:7579/Mobius/keti_tracker/000000000000213/down/sub_downlink
delete resc: 200 <- {"m2m:sub":{"pi":"3-20200924060123628","ri":"23-20200924060756385","ty":23,"ct":"20200924T060756","rn":"sub_
6","et":"20220924T060756","enc":{"net":[3],"exc":100,"nu":["mqtt://203.253.128.161/Sketi_tracker?ct=json"],"nct":2,"cr":"S"}}}
create sub: POST -> http://203.253.128.161:7579/Mobius/keti_tracker/000000000000213/down
create sub: 201 <- {"m2m:sub":{"rn":"sub_downlink","ty":23,"pi":"3-20200924060123628","ri":"23-20200924062530908","ct":"20200924
","et":"20220924T062530","nu":["mqtt://203.253.128.161/Sketi_tracker?ct=json"],"enc":{"net":[3],"exc":100,"nct":2,"cr":"S"}}}
SUB_Complete!!
init_mqtt_client!!!
[mqtt_connect] noti_topic : /oneM2M/req+/Sketi_tracker/#
[ls_mqtt_connect] ls_noti_topic : application/6/device/000000000000213/rx
```

oneM2M-LoRa IPE Interworking



- Check the data uploaded to the oneM2M platform through Postman tool

GET http://203.253.128.161:7579/Mobius/keti_tracker/0000000000000213/up/la

Params Authorization Headers (10) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE
Key	Value

Body Cookies Headers (13) Test Results Status: 200 OK

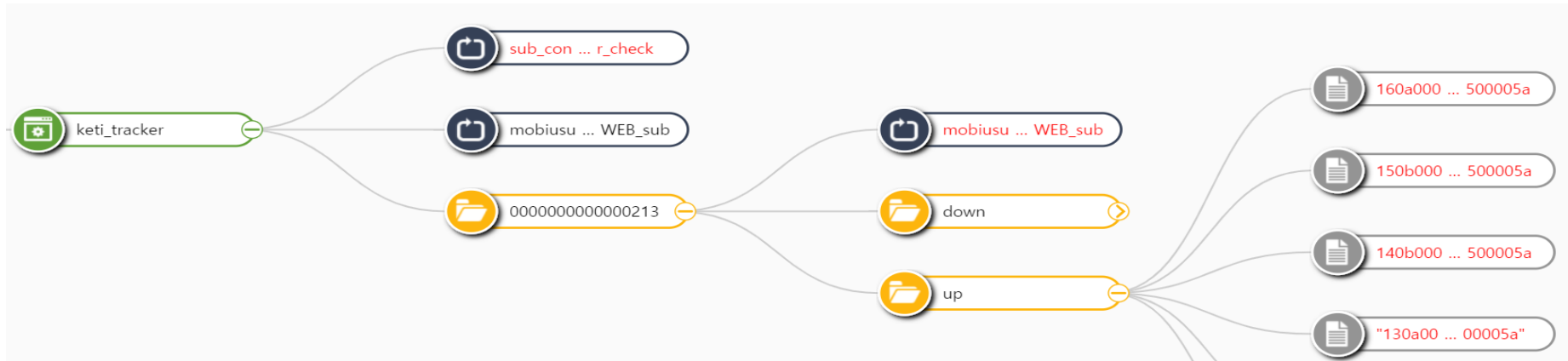
Pretty Raw Preview Visualize JSON

```
1 {
2   "m2m:cin": {
3     "pi": "3-20200924060123510",
4     "ri": "4-20200924062835088",
5     "ty": 4,
6     "ct": "20200924T062835",
7     "st": 6,
8     "rn": "4-20200924062835087",
9     "lt": "20200924T062835",
10    "et": "20220924T062835",
11    "cs": 32,
12    "cr": "S",
13    "con": "140b000213423faaac517015500005a"
14  }
15 }
```

```
[ls_mqtt_connect] ls_noti_topic : application/6/device/0000000000000213/rx
receive message from topic: <- application/6/device/0000000000000213/rx
receive message: {"applicationID":"6","applicationName":"tracker","deviceName":"000
0000000000213","devEUI":"0000000000000213","rxInfo":[{"gatewayID":"b827ebfffe851ed0
","name":"b827ebfffe851ed0","rssi":-117,"loRaSNR":-11,"location":{"latitude":0,"lon
gitude":0,"altitude":0}},{"gatewayID":"b827ebfffe714bf","name":"b827ebfffe714bf",
"rssi":-117,"loRaSNR":-13,"location":{"latitude":0,"longitude":0,"altitude":0}]},"t
xInfo":{"frequency":922100000,"dr":0},"adr":true,"fCnt":20,"fPort":2,"data":"FAsAAh
NCP6qqxRcBVQAAWg==" }
payload_message is FAsAAhNCP6qqxRcBVQAAWg==
create cin: POST -> http://203.253.128.161:7579/Mobius/keti_tracker/000000000000021
3/up
create cin: 201 <- {"m2m:cin":{"rn":"4-20200924062835087","ty":4,"pi":"3-2020092406
0123510","ri":"4-20200924062835088","ct":"20200924T062835","lt":"20200924T062835","
st":6,"et":"20220924T062835","cs":32,"con":"140b000213423faaac517015500005a","cr":
"S"}}
{ code: 201,
  body:
    '{"m2m:cin":{"rn":"4-20200924062835087","ty":4,"pi":"3-20200924060123510","ri":"
4-20200924062835088","ct":"20200924T062835","lt":"20200924T062835","st":6,"et":"202
20924T062835","cs":32,"con":"140b000213423faaac517015500005a","cr":"S"}}' }
```

oneM2M-LoRa IPE Interworking

- Uplink LoRa Device data uploaded to oneM2M platform



The diagram illustrates the data flow from the 'keti_tracker' application to various devices. The devices are connected to a central data stream, which is then processed into four output streams: '160a000 ... 500005a', '150b000 ... 500005a', '140b000 ... 500005a', and '130a00 ... 00005a'.

[Applications](#) / [tracker](#) / [Devices](#) / **0000000000000213** DELETE

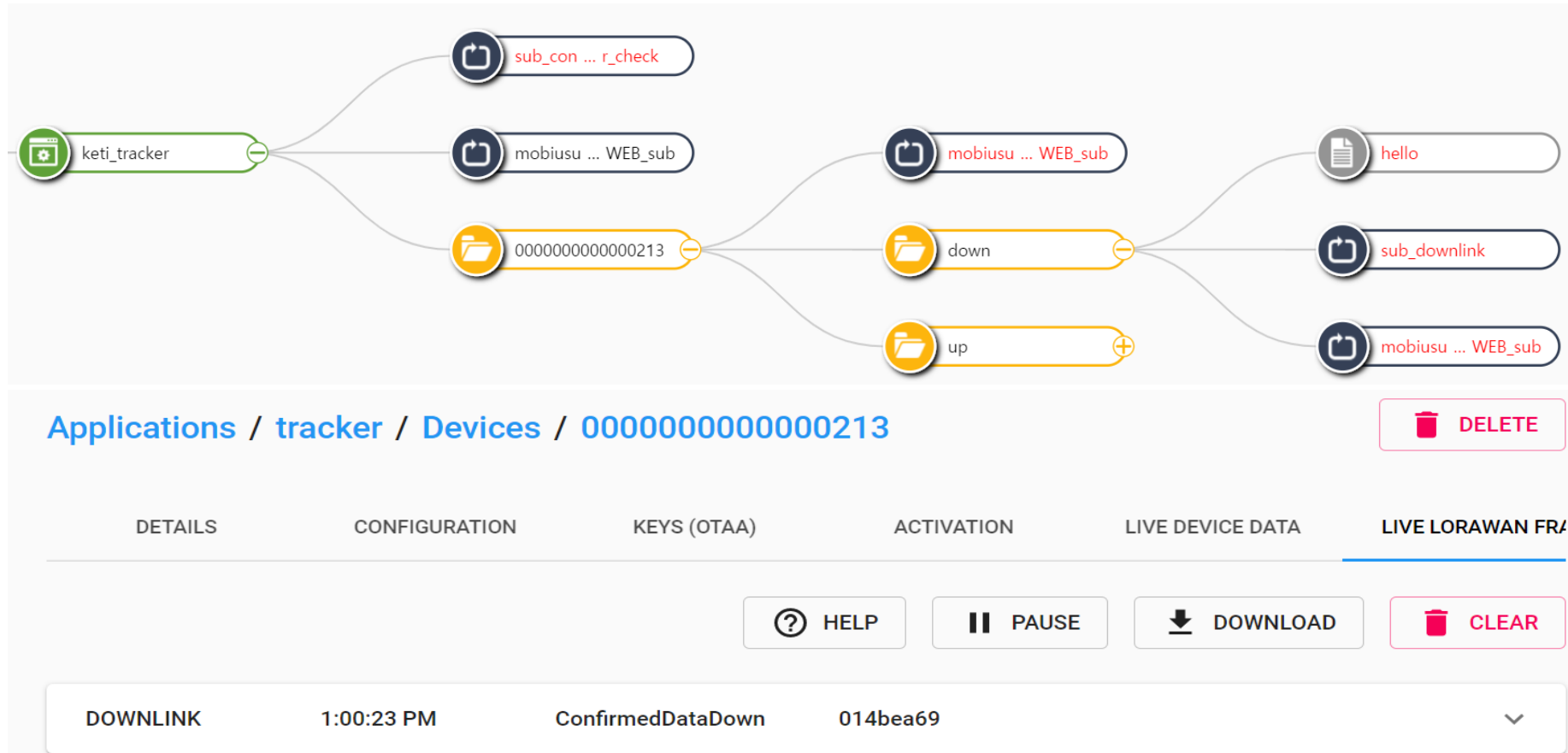
DETAILS CONFIGURATION KEYS (OTAA) ACTIVATION LIVE DEVICE DATA **LIVE LORAWAN I**

HELP PAUSE DOWNLOAD CLEAR

DOWNLINK	1:54:08 PM	UnconfirmedDataDown	014bea69	▼
UPLINK	1:54:08 PM	ConfirmedDataUp	014bea69	▼

oneM2M-LoRa IPE Interworking

- Downlink LoRa Device data through oneM2M platform



The screenshot displays the oneM2M platform interface for managing LoRa devices. At the top, a diagram illustrates the data flow from the 'keti_tracker' application to various devices, including 'sub_con ... r_check', 'mobiusu ... WEB_sub', and '0000000000000213'. The '0000000000000213' device is further connected to 'mobiusu ... WEB_sub', 'down', and 'up' nodes. These nodes are then connected to 'hello', 'sub_downlink', and 'mobiusu ... WEB_sub' nodes respectively.

Below the diagram, the breadcrumb navigation shows 'Applications / tracker / Devices / 0000000000000213'. A 'DELETE' button is visible next to the device ID.

The interface includes tabs for 'DETAILS', 'CONFIGURATION', 'KEYS (OTAA)', 'ACTIVATION', 'LIVE DEVICE DATA', and 'LIVE LORAWAN FR/'. The 'LIVE DEVICE DATA' tab is currently selected.

Below the tabs, there are buttons for 'HELP', 'PAUSE', 'DOWNLOAD', and 'CLEAR'. A table at the bottom shows the downlink data for the selected device.

Direction	Time	Message	Message ID
DOWNLINK	1:00:23 PM	ConfirmedDataDown	014bea69