

# IoT Engineering

## 8: Long Range Connectivity with LoRaWAN

CC BY-SA, Thomas Amberg, FHNW  
(unless noted otherwise)

Slides: [tmb.gr/iot-8](https://tmb.gr/iot-8)

# Overview

These slides introduce *LoRaWAN connectivity*.

How the network is built from nodes and gateways.

Sending small amounts of data over a long distance.

Getting the data to an application from the backend.

# Prerequisites

Set up the [Feather nRF52840 Express](#) for Arduino.

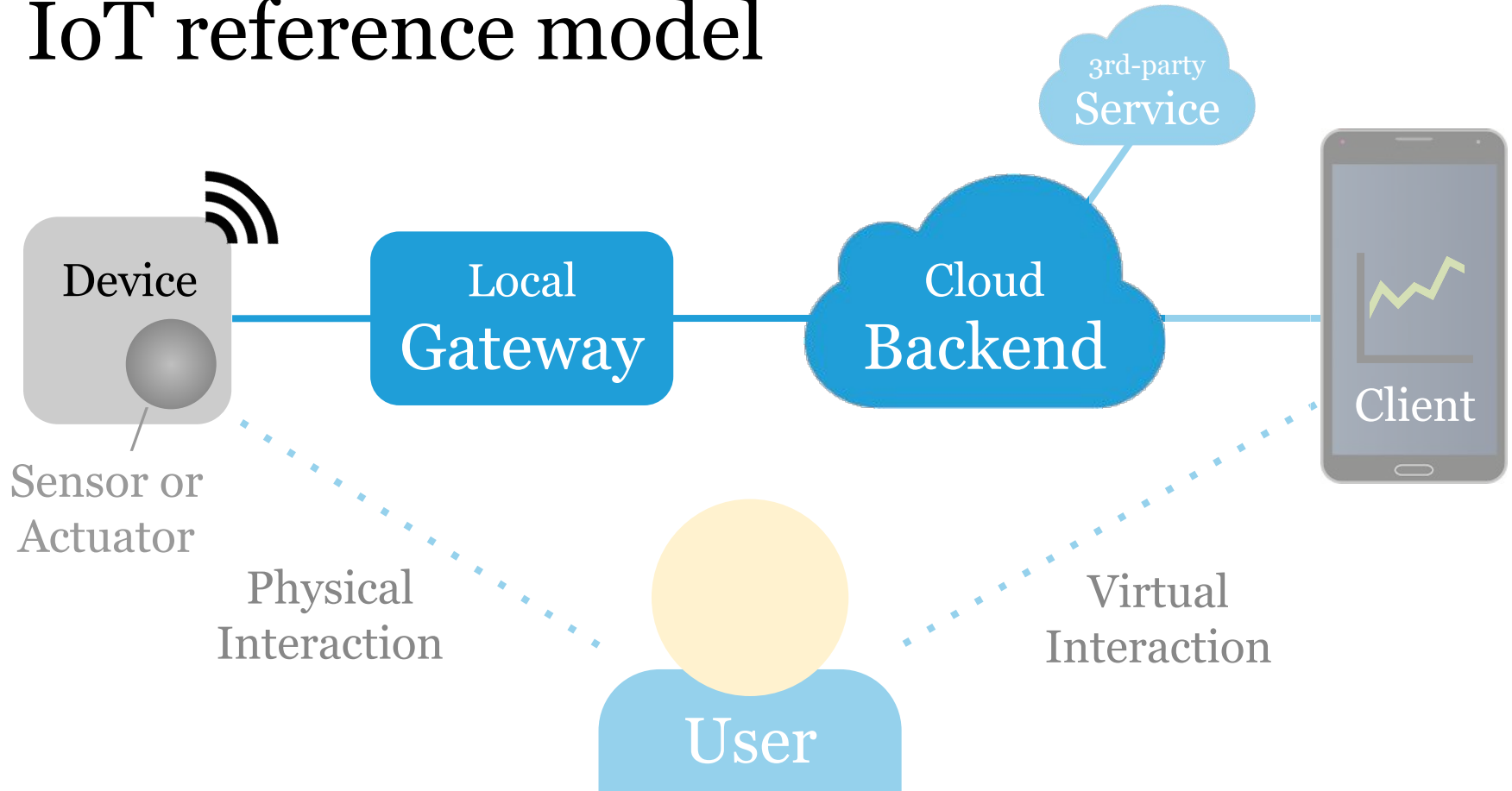
Or [set up the Feather Huzzah ESP8266](#), both work.

A LoRaWAN gateway has to be in range for testing\*.

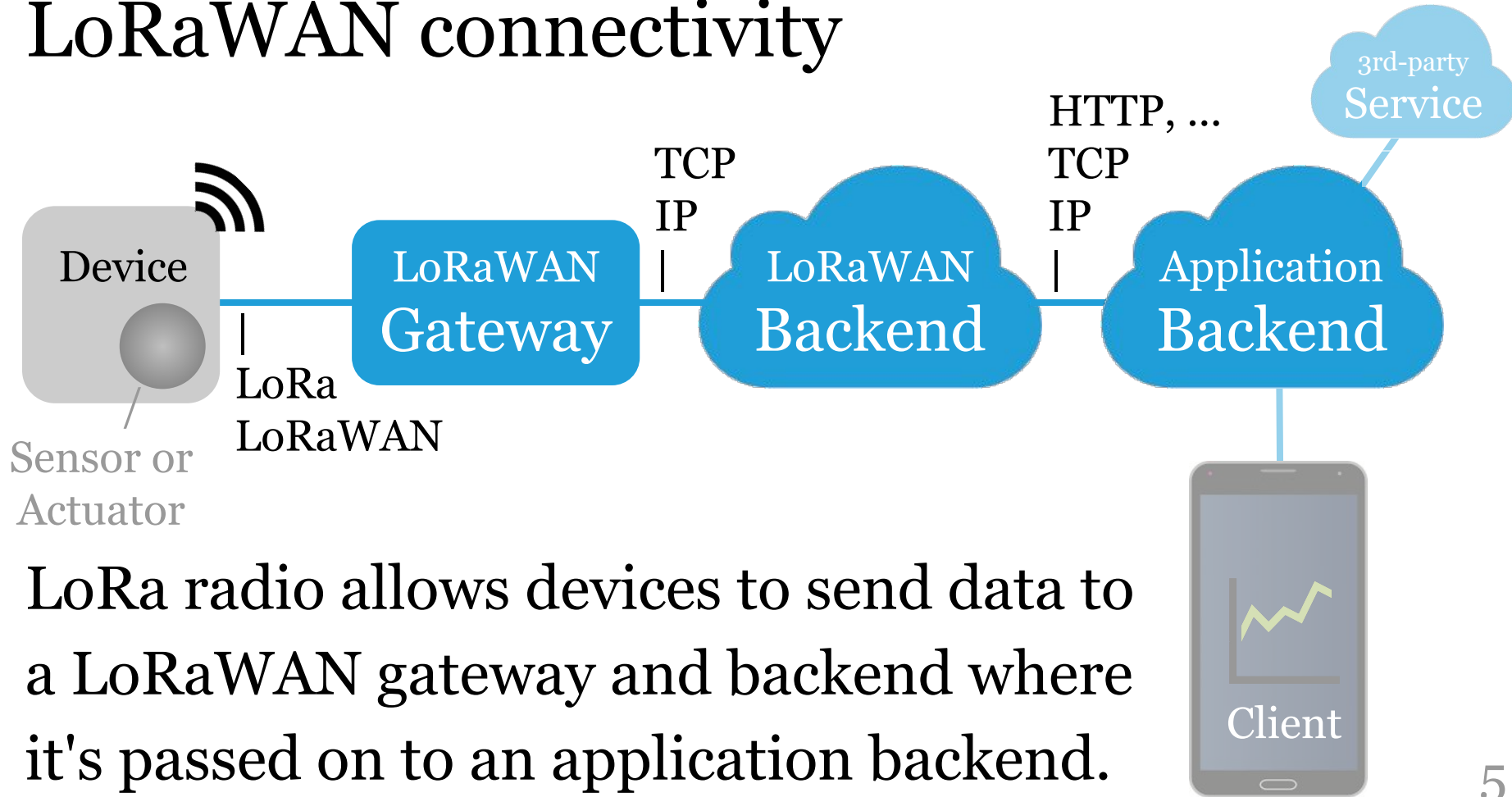
The [Raspberry Pi](#) with [Node.js](#) is our app backend.

\*Here's a [gateway map](#) for [thethingsnetwork.org](#).

# IoT reference model



# LoRaWAN connectivity



LoRa radio allows devices to send data to a LoRaWAN gateway and backend where it's passed on to an application backend.

# LoRa

LoRa is a digital wireless communication technology.

The LoRa physical layer protocol is proprietary.

Semtech, the owner, sells LoRa transceivers.

LoRa radio is long range\* and low power.

\*Around 1 km in cities, 10+ km in open terrain.

# LoRaWAN

LoRaWAN is a low power\*, wide area networking protocol (LPWAN) based on the LoRa physical layer.

The LoRaWAN specification (v1.0.3) is developed by the LoRa Alliance, a non-profit industry consortium.

LoRaWAN defines link layer parameters, addressing, a transport protocol and the network architecture.

\*E.g. LoRaWAN 10/30 mA vs. Wi-Fi 50/150 mA.

# LoRaWAN terminology

The LoRaWAN community uses the following terms:

Node — device with sensors, LoRaWAN connectivity.

Gateway — LoRa (to Internet, i.e. TCP/IP) gateway.

Network server — LoRaWAN specific backend.

Application server — application backend.



# LoRaWAN frequencies

LoRaWAN uses frequencies in license-free bands.

Frequencies depend on the geographic region.

EU 868 MHz, US 915 MHz, Asia 433 MHz, ...

There are **frequency plans**, **per country**\*.

\*Based on the **regional parameters** specification.

# LoRaWAN network providers

There are various ways to get LoRaWAN coverage, e.g.

LoRaWAN network providers like [Swisscom](#) ([Actility](#)).

LoRaWAN backend/solution providers like [Loriot](#).

Open infrastructure like [The Things Network](#).

This course uses The Things Network.

# The Things Network (TTN)

TTN is an **open source** project started in Amsterdam.

Everybody can put up a gateway to extend coverage.

Everybody can get an account and register devices.

The network is open, but your data stays private.

TTN has regional communities, e.g. **TTN Zürich**.

# Mapping network coverage

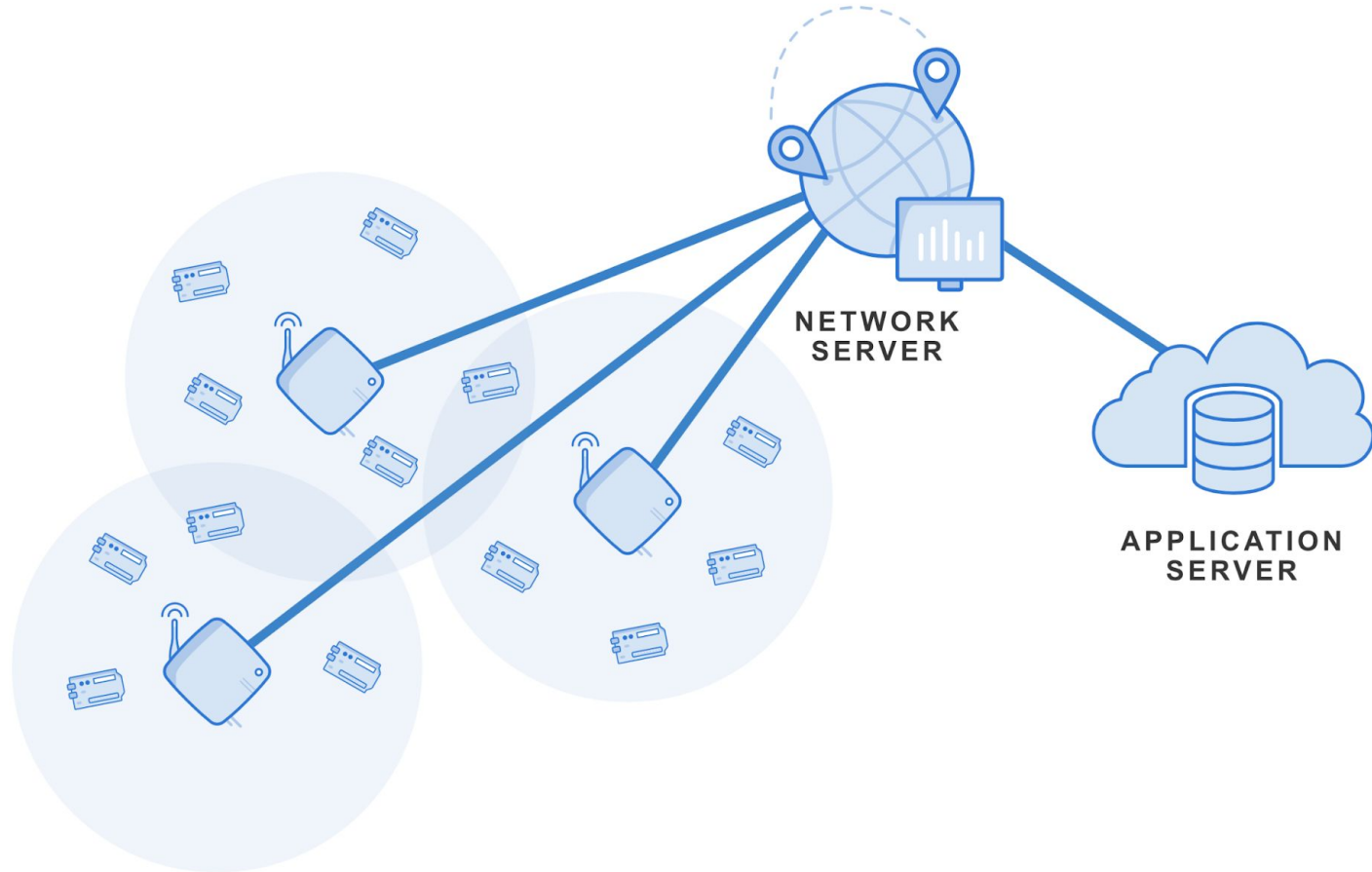
TTNMapper is a community effort to map coverage.

The iOS or Android app provides GPS location data.

The data is correlated with metadata from gateways.

Any LoRaWAN device sending\* packets works fine.

\*One packet can be received by multiple gateways.



# LoRaWAN gateways

Gateways forward *uplink* data packets to the backend.

There, they are deduplicated & routed to applications.

*Downlink* packets are "broadcast" to nearby devices.

No license is required\* to run a LoRaWAN gateway.

\*If operating in a unlicensed radio frequency band. 14

# LoRaWAN security

LoRaWAN transport **security** is based on 128 bit keys:

*Network Session Key* — admits a device to a network.

*Application Session Key* — encrypts/decrypts payload.

These keys can be static or generated for each session.

# Over The Air Activation (OTAA)

OTAA uses an app key to generate keys per session.

- Device has a *DevEUI*, *AppEUI* and *AppKey*
- Device sends a *Join Request*, uses *Join Response* and *AppKey* to derive an *AppSKey* and *NwkSKey*
- Device must be able to store the generated keys
- Join decision can be delegated to a *Join Server*

See [spec](#) or [TTN docs](#) on end-device activation.



# Activation by Personalization (ABP)

ABP uses static app session and network session keys.

- Device has a *Device Address*, *AppSKey* & *NwkSKey*
- No *DevEUI*, *AppEUI* or *AppKey* is needed here
- There is no *Join*, the device just sends data
- Overall ABP is simpler, but less flexible
- Changing the provider is not possible

See also [TTN docs on ABP vs. OTAA](#).

# Registering an application on TTN

An *application* is required to add *devices* later on, grouping devices with the same payload format.

- Open the [TTN console for Europe \(eu1\)](#)
- Go to *Applications > Add application*
- Enter an *Application ID*, e.g. `fhnw-iot`
- Click *Create application*

Similar steps work for other LoRaWAN providers.

# Registering a device on TTN

- Open the [TTN console for Europe \(eu1\)](#)
- Go to *Applications* > click, e.g. fhnw-iot
- Click *Add end device* > *Manually*
- Click *Show advanced activation*
- Choose either *OTAA* or *ABP*\*
- Do *not yet* click register

\*See next two pages to continue with OTAA or ABP. 19

# Getting OTAA keys on TTN

If you're in the process of registering a OTAA device:

- Click *Generate* on *DevEUI*, *AppEUI* and *AppKey*\*
- Set an *End device ID*, e.g. `fhnw-iot-device-0`
- Click *Register end device*

\*You'll add these keys to the device code later on.

# Getting ABP keys on TTN

If you're in the process of registering an ABP device:

- Click *Generate* on *DevEUI*, *AppSKey* and *NwkSKey*
- Click the refresh icon to generate a *Device address*
- Set an *End device ID*, e.g. fhnw-iot-device-0
- Click *Register end device*, then for the same device
- Go to *General settings* > *Network layer* > *Advanced MAC settings* > *Reset frame counters [x]*

# LoRaWAN hardware modules

Some LoRaWAN modules, based on Semtech [SX127x](#):

[RN2483](#) — via UART/AT commands (or stand-alone).

[RFM95W](#) — via SPI, stack runs on separate controller.

[Murata](#) — SoC including an ARM STM32 Cortex Mo.

Make sure the frequency is allowed in your [region](#). 22

# FeatherWing RFM95W

RFM95W is a popular 868 MHz LoRa radio module.

The RFM95W FeatherWing needs a microcontroller.

Both MCU boards work, nRF52840 and ESP8266.

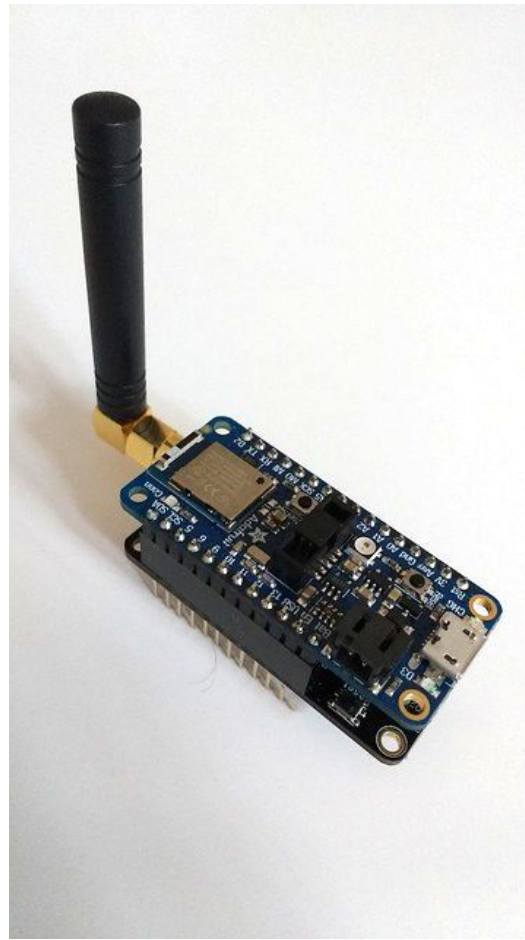
The pin mapping has to be adapted in the code.

# Setup RFM95W

First, make sure an antenna is present.

Then, nRF52840 goes on top of the LoRaWAN module.

Or, ESP8266 goes below the module.





# Arduino LoRaWAN with LMIC library

These examples work on nRF52840 and ESP8266.

They run LMIC to use the RFM95W LoRa module.

Install the [MCCI LMIC library](#) in the Arduino IDE.

Edit *Arduino/libraries/MCCI\_LoRaWAN\_LMIC\_library/project\_config/lmic\_project\_config.h*

```
#define CFG_eu868 1 // #define CFG_us915 1
```

# Arduino LoRaWAN ABP [.ino](#)<sup>ESP</sup>, [.ino](#)<sup>nRF</sup>

Set the *NwkSKey*, *AppSKey* and *Device Address*:

```
static const PROGMEM u1_t NWKSKEY[16] = { ... }  
static const u1_t PROGMEM APPSKEY[16] = { ... }  
static const u4_t DEVADDR = 0x01234567;
```

Double check to use the pin mapping for your board:

```
const lmic_pinmap lmic_pins = {...} // nRF52...
```

Set a custom message:

```
static uint8_t mydata[] = "Hello, world!";
```

# Arduino LoRaWAN OTAA [.ino](#)<sup>ESP</sup>, [.ino](#)<sup>nRF</sup>

Set the *AppEUI*, *DevEUI* and *AppKey*:

```
static const u1_t PROGMEM APPEUI[8]= { ... }  
static const u1_t PROGMEM DEVEUI[8]= { ... }  
static const u1_t PROGMEM APPKEY[16] = { ... }
```

Double check to use the pin mapping for your board:

```
const lmic_pinmap lmic_pins = {...} // nRF52...
```

Set a custom message:

```
static uint8_t mydata[] = "Hello, world!";
```

# Hands-on, 15': Arduino LoRaWAN

Get an account at <https://thethingsnetwork.org/>

Register a TTN application, and register two devices.

Get ABP keys for one device, OTAA keys for another.

Set these keys in the LoRaWAN *.ino* code examples.

Make sure to use the right code for your hardware\*.

\*The pin mapping varies for ESP8266, nRF52840. 28

THE THINGS NETWORK

THE THINGS STACK  
Community Edition

OverviewApplicationsGatewaysOrganizations

EU1 Community  
Fair use policy applies

tamberg

fhnw-iot

Overview

End devices

Live data

Payload formatters

Uplink

Downlink

Integrations

Collaborators

API keys

General settings

Applications > fhnw-iot > End devices > fhnw-iot-device-1 > Live data

fhnw-iot-device-1

ID: fhnw-iot-device-1

↑ 1 ↓ n/a Last activity 54 minutes ago

OverviewLive dataMessagingLocationPayload formattersGeneral settings

Time	Type	Data preview
11:22:10	Update end device	[ "activated_at" ]
11:22:10	Forward uplink data message	
11:22:10	Forward uplink data message	MAC payload: C2 51 03 D0 07 AC 5A D3 ... FPort: 1 Data rate: SF7BW125 SNR: -7 RSSI: -115

Verbose stream Pause Clear

# Uplink and downlink

*Uplink* — sending data from a device to the backend.

*Downlink* — sending from the backend to a device.

There's an *asymmetry* due to duty cycle limitations.

Gateways are *half-duplex*, they either send or listen.

LoRaWAN is better suited to send data *to* the cloud.

# MQTT integration

The TTN backend offers an [MQTT broker](#) API.

To get uplink packets from a device:

```
$ mqtt sub -t "v3/<AppID>@ttn/devices/<DevID>/up" \  
-h "eu.thethings.network" -u "<AppID>@ttn" \  
-P "<ApiKey>" # see TTN console > Integrations > MQTT
```

To send a packet downlink, to a device:

```
$ mqtt pub -t "v3/<AppID>@ttn/devices/<DevID>/down/push" \  
-m '{"downlinks":[{"f_port":1,"frm_payload":"<Base64>","priority": "NORMAL"}]}' -h ... -u ... -P ... # as above
```

# HTTP Webhook integration

The TTN backend provides a RESTful **HTTP API**.

A POST request allows to **send packets downlink**.

A *Webhook* URL can be set to **receive uplink data**.

The TTN backend calls this URL for each packet.

The backend also defines the **JSON data format**.



# How to debug Webhook calls

To debug Webhook calls, set up a simple Web service:

```
$ wget https://raw.githubusercontent.com/\  
tamberg/fhnw-iot/master/08/Nodejs/HttpLogger.js  
$ node HttpLogger.js # runs on 127.0.0.1:8080
```

Make it accessible via [Ngrok](#), [PageKite](#) or [Yaler](#) relay.

=> URL, e.g. [https://RELAY\\_DOMAIN.try.yaler.io/](https://RELAY_DOMAIN.try.yaler.io/)

Set this URL as Webhook URL, watch the shell.

# Product-specific integrations

LoRaWAN backends (here TTN) provide product specific **integrations** with 3rd-party services.

On The Things Network, to create a new integration:

- Open the **TTN console for Europe (eu1)**
- Go to *Applications* > click, e.g. fhnw-iot
- Go to *Integrations* > choose *AWS, Azure, ...*

# Hands-on, 15': TTN integrations

Read the TTN [MQTT](#) and [Webhooks](#) API docs.

Use the Raspberry Pi as your application backend.

Use a [MQTT SUB client](#) to log incoming messages.

Set up an [HTTP Logger](#) to see TTN Webhook calls.

# Data formats

Bandwidth is very limited, payload is  $\leq 51$  Byte.

JSON or plain ASCII formats use too much space:

`{"temperature":20.63}` vs. `20.63` vs. `0x080F`

TTN allows to use JS [payload decoders](#) & encoders.

A binary format suitable for LoRa is [CayenneLPP](#).

Less Bytes per message => send more often.

# Limitations

LoRaWAN has physical, legal & operator limitations:

**Duty cycle** limitations allow only 1% air time in EU\*, apply to nodes *and* gateways, creating asymmetry.

The **TTN Fair Use Policy** limits uplink **air time** to 30s and downlink to 10 messages per 24h per node.

\*See ETSI **EN300.220** standard, 7.2.3.

# Hands-on, 15': LoRaWAN use cases

Which applications become possible with LoRaWAN?

Does free wide area connectivity change anything?

Sketch the reference model for an application.

Find a case that clearly beats Wi-Fi, 3/4G.

# Summary

LoRaWAN brings long range, low power connectivity.

We learned about gateways and network architecture.

We sent packets uplink, from a device, and downlink.

We understand how data arrives at the app backend.

Next: Dashboards and Apps for Sensor Data.

# Feedback or questions?

Write me on <https://fhnw-iot.slack.com/>

Or email [thomas.amberg@fhnw.ch](mailto:thomas.amberg@fhnw.ch)

Thanks for your time.