



EUROPEAN UNION

**Interreg**   
Austria-Czech Republic  
European Regional Development Fund

# LoRa(WAN) Webinar

## ChirpStack

Author: Harald Eigner (TU Wien)

These slides show how to setup a LoRaWAN test environment. The presentation focuses on a setup based on the open source software ChirpStack, a Raspberry Pi, and an IMST iC880A concentrator board. It contains all steps starting from scratch until a successful OTAA join of a RN2483 based evaluation board, including the required commands for the RN2483.



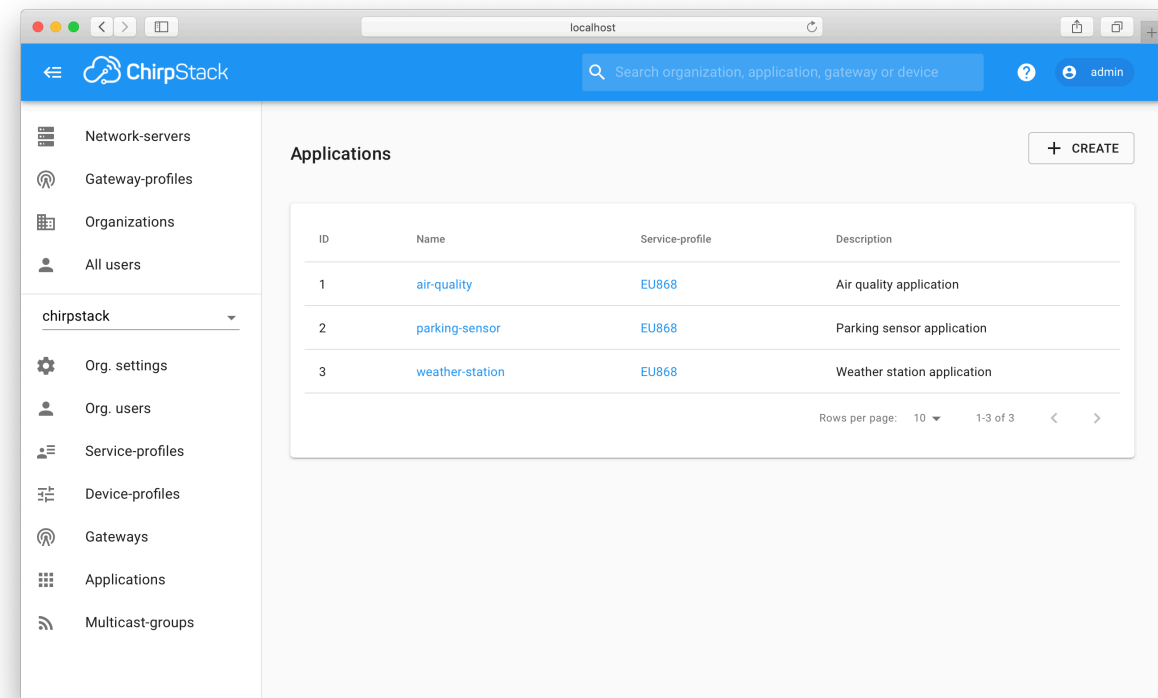
TECHNISCHE  
UNIVERSITÄT  
WIEN  
Vienna | Austria



- ChirpStack components
- Installation Guide of a LoRaWAN Communication System with ChirpStack
  - ChirpStack Installation Guide for Debian or Ubuntu
  - ChirpStack Gateway Installation Guide
  - Connecting a Device

- ChirpStack components
- Installation Guide of a LoRaWAN Communication System with ChirpStack
  - ChirpStack Installation Guide for Debian or Ubuntu
  - ChirpStack Gateway Installation Guide
  - Connecting a Device

- <https://www.chirpstack.io/>
- Open-source components for LoRaWAN networks
- Web interface for device management
- APIs for integration



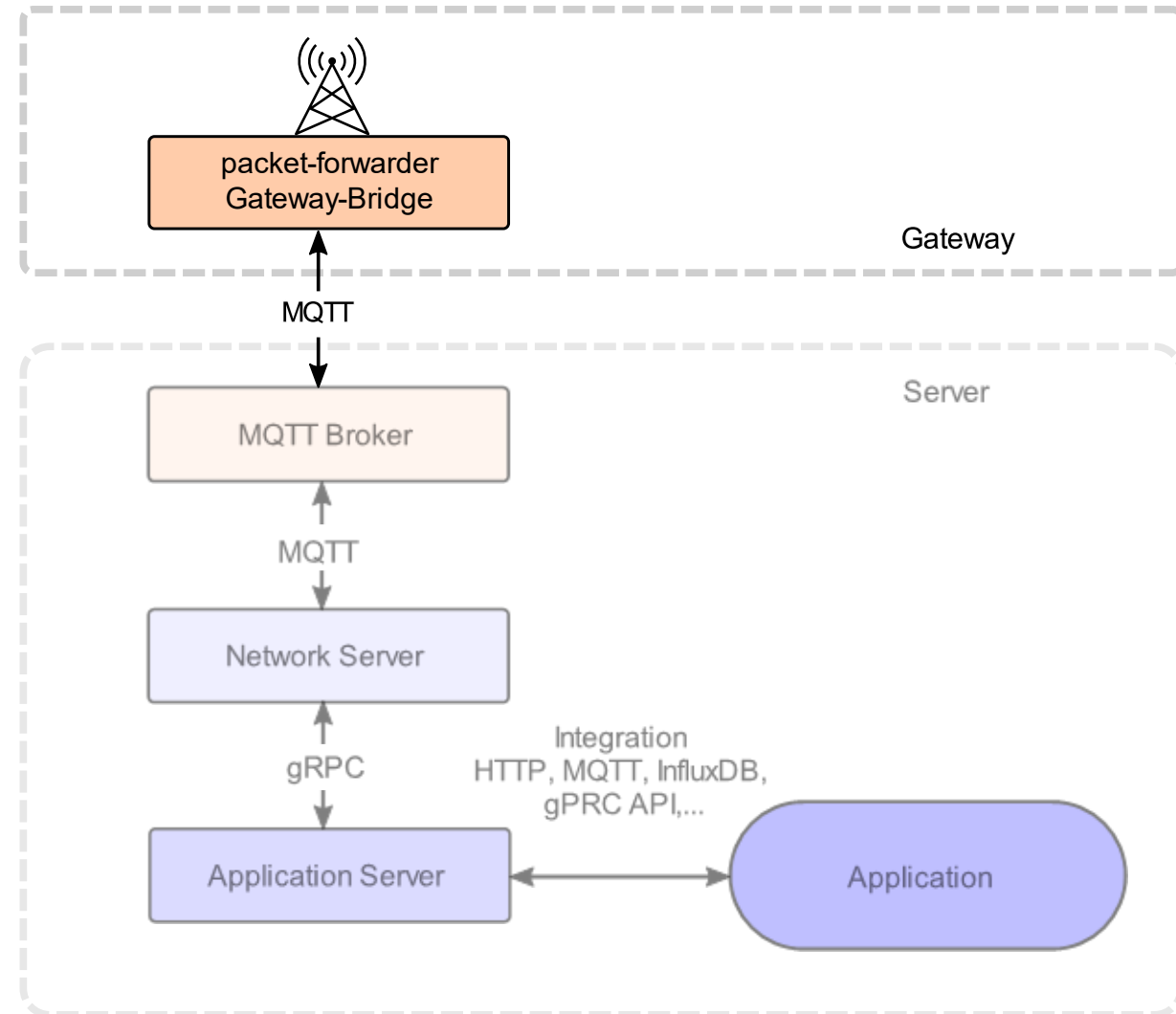
ChirpStack web interface

## Packet-forwarder:

- Forwards LoRa Packets to gateway-bridge

## Gateway Bridge:

- Converts Packet Forwarder protocol into JSON or Protobuf data format
- Can be integrated on gateway or server



## MQTT Broker:

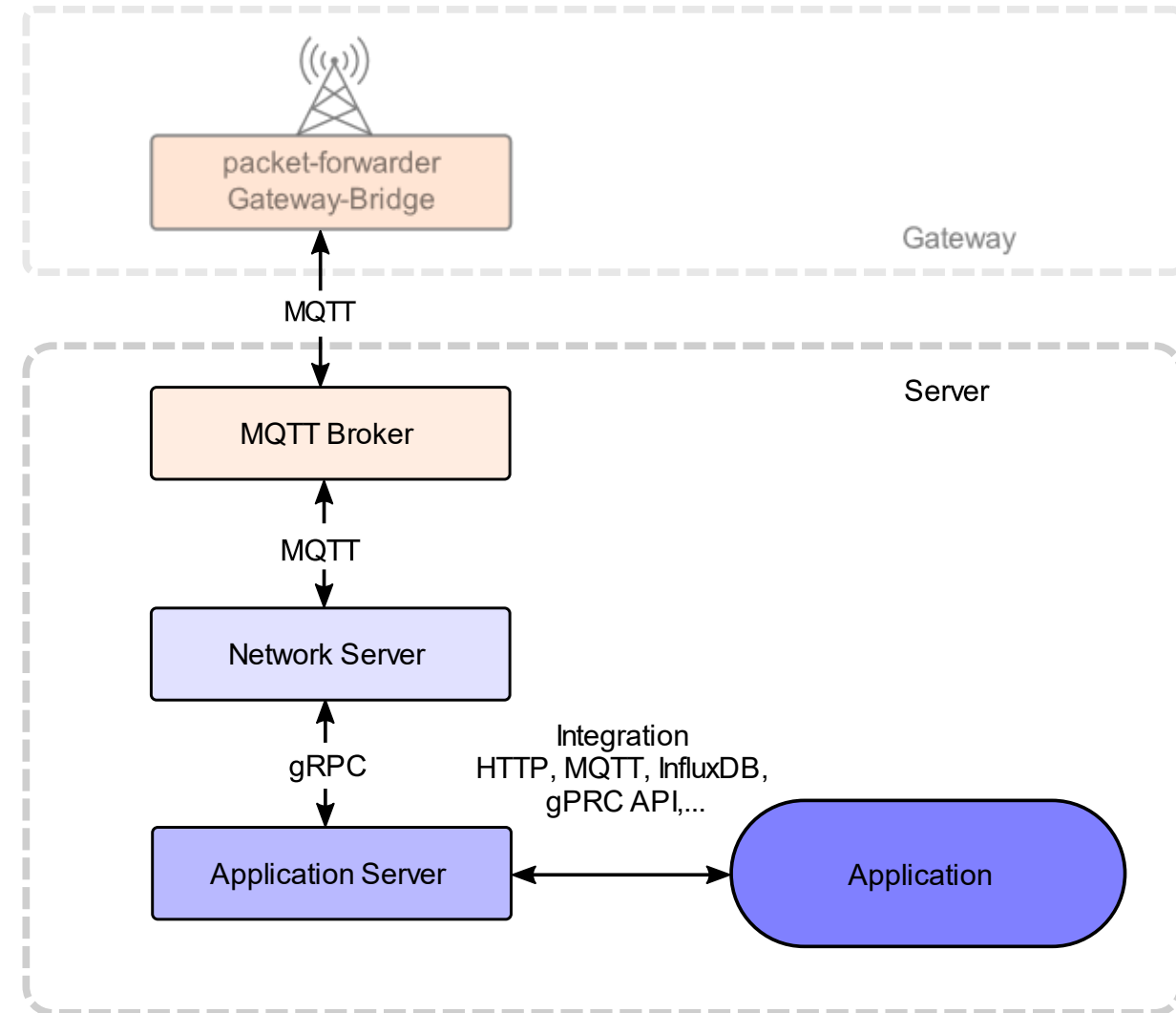
- Communication between gateway and network server via MQTT using MQTT broker like Eclipse Mosquitto

## Network Server:

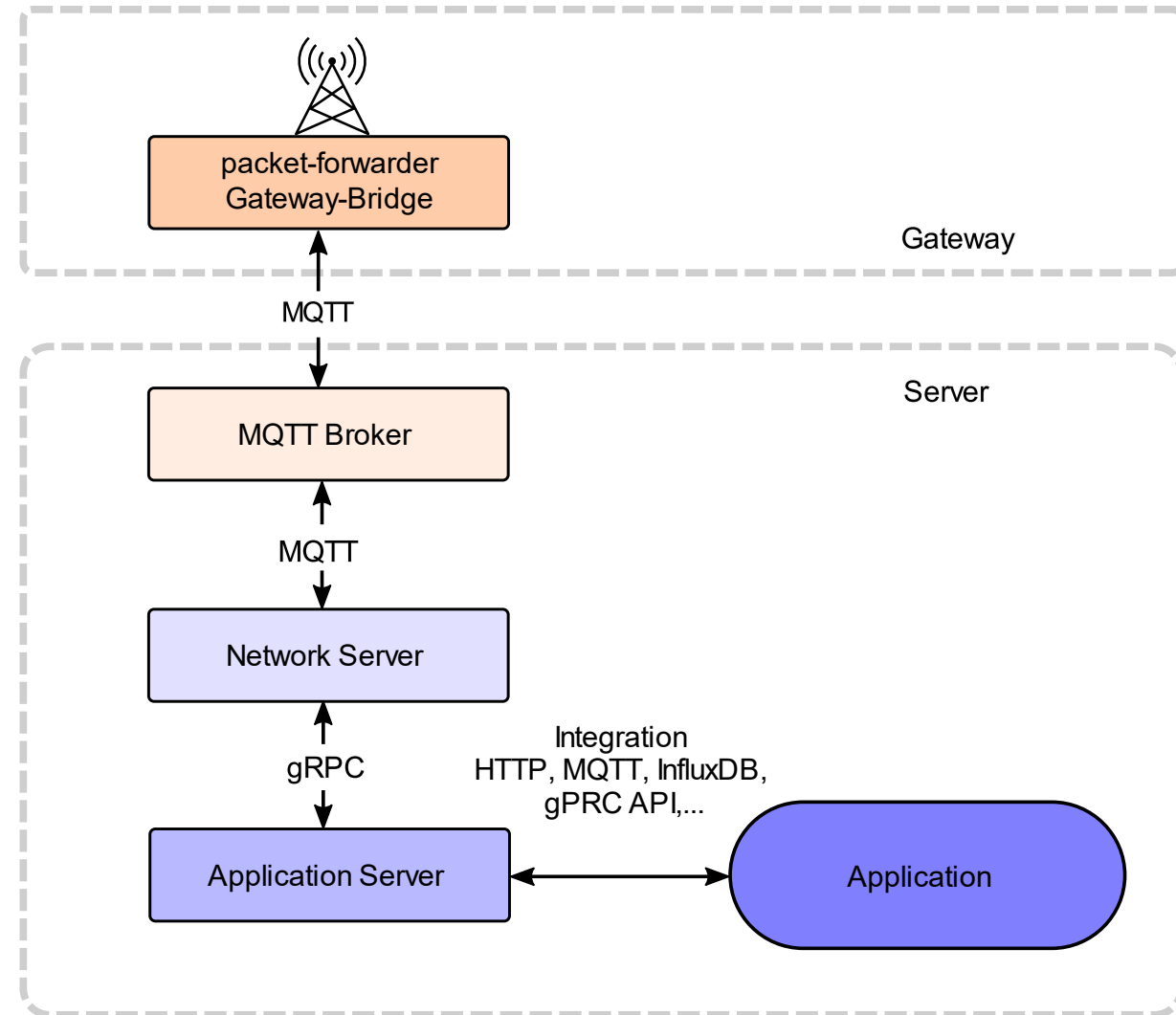
- Communication with application server via gRPC

## Application Server:

- Several implementations available to connect with application



- Gateway OS is an open source Linux based embedded operating system
- Two systems available:
  - Base OS system: provides Packet Forwarder and Gateway Bridge
  - Full OS system: provides full server environment



- Tutorial of steps needed to setup the ChirpStack server stack

Many configurations are possible. In this tutorial, the following assumptions of the deployment architecture are made:

- All ChirpStack components and their dependencies will be installed on a single server instance.
- The ChirpStack Gateway Bridge component will be installed on gateway but can also be installed on the server itself.
- No firewall rules are setup.



- ChirpStack components
- **Installation Guide of a LoRaWAN Communication System with ChirpStack**
  - ChirpStack Installation Guide for Debian or Ubuntu
  - ChirpStack Gateway Installation Guide
  - Connecting a Device

Full installation guide for:

- ChirpStack Server stack
  - Installation guide for Debian or Ubuntu
- Gateway
  - Installation guide for
    - Raspberry Pi 3
    - IMST – iC880A Concentrator shield
- Connecting a Device
  - Installation guide for including a Microchip RN2483 wireless module to the network

- ChirpStack components
- Installation Guide of a LoRaWAN Communication System with ChirpStack
  - **ChirpStack Installation Guide for Debian or Ubuntu**
  - ChirpStack Gateway Installation Guide
  - Connecting a Device

## Install dependencies:

- MQTT broker: A publish/subscribe protocol that allows users to publish information under topics that others can subscribe to.
- Redis: An in-memory database used to store relatively transient data
- PostgreSQL: The long-term storage database used by the open source packages

Command to install the dependencies:

```
sudo apt install mosquitto mosquitto-clients redis-server redis-tools postgresql
```

## Setup PostgreSQL databases and users:

- To enter the command line utility for PostgreSQL:
- Inside the prompt, the databases are set up used by the ChirpStack components.

```
sudo -u postgres psql
```

```
-- set up the users and the passwords
-- (note that it is important to use single quotes and a semicolon at the end!)
create role chirpstack_as with login password 'dbpassword';
create role chirpstack_ns with login password 'dbpassword';

-- create the database for the servers
create database chirpstack_as with owner chirpstack_as;
create database chirpstack_ns with owner chirpstack_ns;

-- change to the ChirpStack Application Server database
\c chirpstack_as

-- enable the pq_trgm and hstore extensions
-- (this is needed to facilitate the search feature)
create extension pq_trgm;
-- (this is needed to store additional k/v meta-data)
create extension hstore;
-- exit psql
\q
```

## Setup ChirpStack software repository:

ChirpStack provides a repository that is compatible with the Ubuntu apt package system.

- Installation of *dirmngr* and *apt-transport-https*:
- Set up the key for this new repository:
- Add the repository to the repository list by creating a new file:
- Update the apt package cache:

```
sudo apt install apt-transport-https dirmngr
```

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys  
1CE2AFD36DBCCA00
```

```
sudo echo "deb https://artifacts.chirpstack.io/packages/3.x/deb stable  
main" | sudo tee /etc/apt/sources.list.d/chirpstack.list
```

```
sudo apt update
```

## Installing the ChirpStack Network Server:

- Installing the package using apt:
- Next step is to update the configuration file to match the database and band configuration.
  - The configuration is located at:
- Example configuration files can be found in:
- After updating the configuration, the ChirpStack Network Server needs to be restarted and validated that there are no errors:
- Print the ChirpStack Network Server log-output:

```
sudo apt install chirpstack-network-server
```

```
/etc/chirpstack-network-server/chirpstack-network-server.toml
```

<https://www.chirpstack.io/network-server/install/config/>

```
# start chirpstack-network-server  
sudo systemctl start chirpstack-network-server  
  
# start chirpstack-network-server on boot  
sudo systemctl enable chirpstack-network-server
```

```
sudo journalctl -f -n 100 -u chirpstack-network-server
```

## Installing the ChirpStack Application Server:

- Installing the package using apt:
- Next step is to update the configuration file to match the database and band configuration.
  - The configuration is located at:
- Example configuration files can be found in:
- After updating the configuration, the ChirpStack Application Server needs to be restarted and validated that there are no errors:
- Print the ChirpStack Application Server log-output:

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

```
/etc/chirpstack-network-server/chirpstack-application-server.toml
```

<https://www.chirpstack.io/application-server/install/config/>

```
# start chirpstack-network-server
sudo systemctl start chirpstack-application-server

# start chirpstack-network-server on boot
sudo systemctl enable chirpstack-application-server
```

```
sudo journalctl -f -n 100 -u chirpstack-application-server
```



## Setting up Mosquitto MQTT Broker:

- First, the MQTT authentication and authorization must be set up. At the server, we change to the mosquito directory and create a new password file for authentication:
- Next, we add permissions for the Chirpstack components to the access control list file `/etc/mosquitto/acls`:
- The authentication for the subscription to the MQTT broker must be updated in the configuration files of the Network and Application Server

```
cd /etc/mosquito
```

```
# Create a password file, with users chirpstack_gw, chirpstack_ns,  
chirpstack_as # and bob.
```

```
sudo mosquitto_passwd -c /etc/mosquitto/passwd chirpstack_gw  
sudo mosquitto_passwd /etc/mosquitto/passwd chirpstack_ns  
sudo mosquitto_passwd /etc/mosquitto/passwd chirpstack_as  
sudo mosquitto_passwd /etc/mosquitto/passwd bob
```

```
# Secure the password file
```

```
sudo chmod 600 /etc/mosquitto/passwd
```

```
user chirpstack_gw  
topic write gateway/+event/+  
topic read gateway/+command/+
```

```
user chirpstack_ns  
topic read gateway/+event/+  
topic write gateway/+command/+
```

```
user chirpstack_as  
topic write application/+device/+event/+  
topic read application/+device/+command/+
```

```
user bob  
topic read application/123/device/+event/+  
topic write application/123/device/+command/+
```

- ChirpStack components
- Installation Guide of a LoRaWAN Communication System with ChirpStack
  - ChirpStack Installation Guide for Debian or Ubuntu
  - **ChirpStack Gateway Installation Guide**
  - Connecting a Device

## Gateway Operatins System (OS):

- ChirpStack Gateway OS is an open-source Linux based embedded OS which can run on various LoRa gateway models.
- Two different image types:
  - chirpstack-gateway-os-base: Provides the ChirpStack packet forwarder and ChirpStack Gateway Bridge pre-installed including a CLI utility for gateway configuration
  - chirpstack-gateway-os-full: Provides a full ChirpStack Network Server and ChirpStack Application Server environment running on the gateway, on top of all the features that are provided by the chirpstack-gateway-os-base image.

## Gateway components:

Several gateway models are supported by Chirpstack. Two components are needed to run a gateway:

- Embedded Linux board: Raspberry Pi
- Concentrator shield: IMST – iC880A, IMST – iC980A, RAK2245, ...

A list of supported hardware can be found at <https://www.chirpstack.io/gateway-os/>

The following installation guide is suited for:

- Raspberry Pi 3 +
- IMST – iC880A



## Connecting the gateway with ChirpStack:

- Next step is to connect the Gateway-Bridge with the MQTT broker.

By executing `sudo gateway-config` the following screen is prompted:

```
lqqqqqqqqqqqqqqqqqqqqChirpStack Gateway OSqqqqqqqqqqqqqqqqqqqqk
x Configuration options: x
x lqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqk x
x x 1 Set admin password x x
x x 2 Setup LoRa concentrator shield x x
x x 3 Edit packet-forwarder config x x
x x 4 Edit ChirpStack Gateway Bridge config x x
x x 5 Restart packet-forwarder x x
x x 6 Restart ChirpStack Gateway Bridge x x
x x 7 Configure WIFI x x
x mqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqj x
x
x
tqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqj
x < OK > < Quit > x
mqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqj
```

- In the packet-forwarder config, the IP address pointing to the MQTT broker must be inserted at the MQTT authentication section:
- Additionally, the authentication for the MQTT subscription must be updated

```
# Generic MQTT authentication.
[integration.mqtt.auth.generic]
# MQTT servers.
#
# Configure one or multiple MQTT server to connect to. Each item
must be in
# the following format: scheme://host:port where scheme is tcp, ssl or
ws.
servers=[
    "tcp://xxx.xxx.xxx.xxx:1883",
]
```

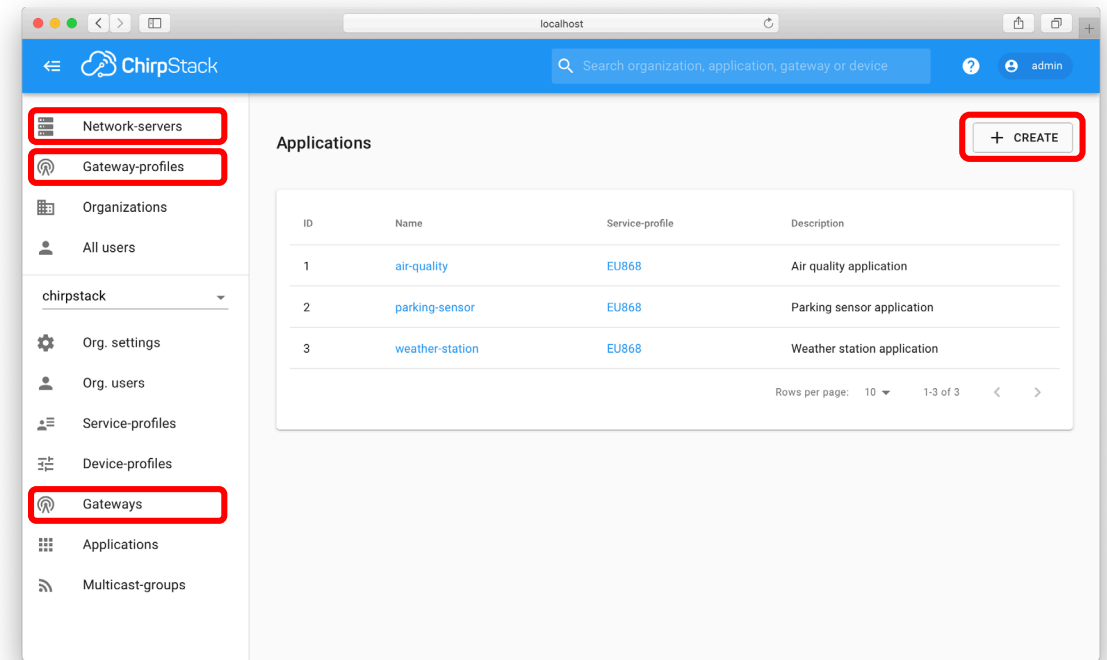
## Connecting the gateway with ChirpStack:

- Next step is to add the gateway to ChirpStack. To do this we must log into the web-interface of the Application Server:
- Three steps are needed to add a Gateway to the ChirpStack server:
  - Adding the Network server in tab “Network-servers”
  - Adding a Gateway-profile in tab “Gateway-profiles”
  - Adding a Gateway in tab “Gateways”

<http://localhost:8080/>

default username: *admin*

default password: *admin*



## Connecting the gateway with ChirpStack:

- Adding the Network Server:
  - Network Server name
  - Network Server server
  
- Adding a Gateway Profile
  - Gateway Profile name
  - Enabled channels according to LoRaWAN Regional Parameters
  - Network Server

Network-servers / Add

GENERAL GATEWAY DISCOVERY TLS CERTIFICATES

Network-server name \*

Network Server

A name to identify the network-server.

Network-server server \*

localhost:8000

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

ADD NETWORK-SERVER

Gateway-profiles / Create

Name \*

Gateway Profile

A short name identifying the gateway-profile.

Enabled channels \*

0,1,3

The channels active in this gateway-profile as specified in the LoRaWAN Regional Parameters specification. Separate channels by comma, e.g. 0, 1, 2. Extra channels must not be included in this list.

Network-server \*

Network Server

Network Server



## Connecting the gateway with ChirpStack:

- Adding the Gateway:
  - Gateway Name
  - Gateway Description
  - Gateway ID
  - Network Server
  - Gateway Profile

Gateways / Create

Gateway name \*  
Gateway1  
The name may only contain words, numbers and dashes.

Gateway description \*  
First Gateway

Gateway ID \*  
b8 27 eb ff fe 1f e4 38 MSB ↻

Network-server \*  
Network Server ▾  
Select the network-server to which the gateway will connect. When no network-servers are available in the dropdown, make sure a service-profile exists for this organization.

Gateway-profile  
Gateway Profile ✕ ▾  
An optional gateway-profile which can be assigned to a gateway. This configuration can be used to automatically re-configure the gateway when ChirpStack Gateway Bridge is configured so that it manages the packet-forwarder configuration.

## Requirements:

Before the device is connected to the system, the following information about the device needs to be known:

- DevEUI: Identifier assigned by the manufacturer
- LoRaWAN MAC version implemented by the device
- Regional Parameters revision implemented by the device

Additionally, the following information is needed specifically for each activation process:

### ABP:

- Device address
- Session Keys

### OTAA:

- Device root-keys

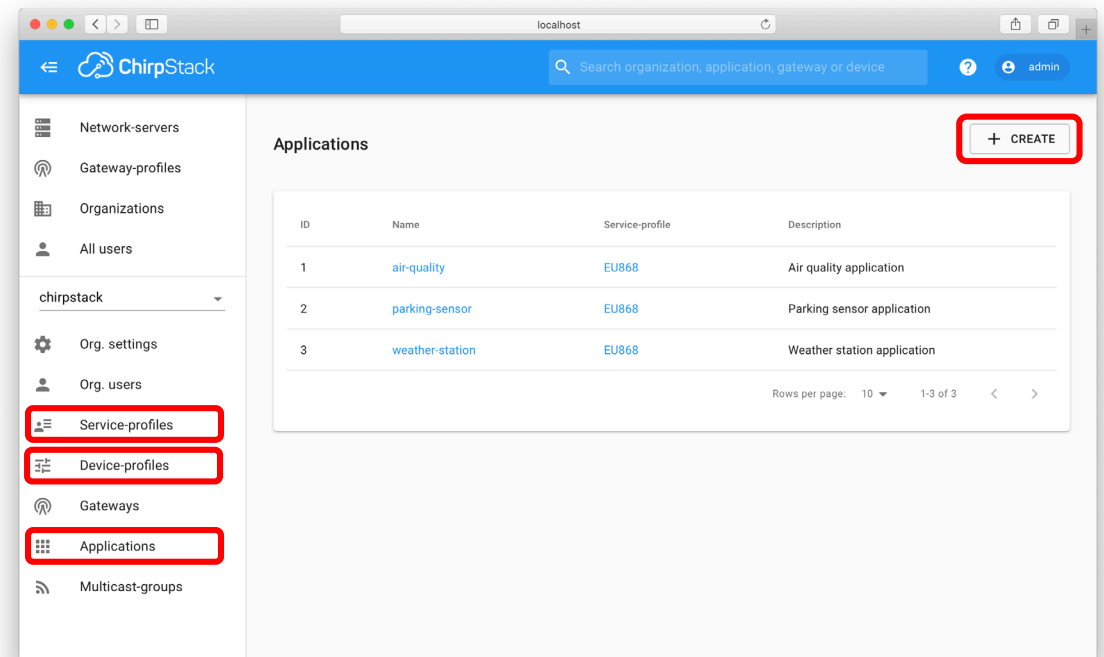
- ChirpStack components
- ChirpStack Installation Guide for Debian or Ubuntu
- ChirpStack Gateway Installation Guide
- **Connecting a Device**

- A new device can be added in the web-interface of the Application Server:
- Three steps are needed to add a Device to the ChirpStack server:
  - Adding a service profile in tab “Service-profiles”
  - Adding a device profile in tab “Device-profiles”
  - Adding an application in tab “Applications”
  - Adding a device inside the created application

<http://localhost:8080/>

default username: *admin*

default password: *admin*



## Adding a Service Profile:

The Service Profile is the "contract" between a user and the network. It describes the features that are enabled for the user(s) of the Service Profile and the rate of messages that can be sent over the network.

### Service-profiles / Create

Service-profile name \*  
Service Profile  
A name to identify the service-profile.

Network-server \*  
Network Server  
The network-server on which this service-profile will be provisioned. After creating the service-profile, this value can't be changed.

Add gateway meta-data  
GW metadata (RSSI, SNR, GW geoloc., etc.) are added to the packet sent to the application-server.

Enable network geolocation  
When enabled, the network-server will try to resolve the location of the devices under this service-profile. Please note that you need to have gateways supporting the fine-timestamp feature and that the network-server needs to be configured in order to provide geolocation support.

Device-status request frequency  
0  
Frequency to initiate an End-Device status request (request/day). Set to 0 to disable.

Minimum allowed data-rate \*  
0  
Minimum allowed data rate. Used for ADR.

Maximum allowed data-rate \*  
5  
Maximum allowed data rate. Used for ADR.

[CREATE SERVICE-PROFILE](#)

## Adding a Device Profile:

A Device Profile defines the device capabilities and boot parameters that are needed by the Network Server for setting the LoRaWAN radio access service. These information elements shall be provided by the end-device manufacturer.

Device-profiles / Create

GENERAL JOIN (OTAA / ABP) CLASS-B CLASS-C CODEC

Device-profile name \*

RN2483

A name to identify the device-profile.

Network-server \*

Network Server

The network-server on which this device-profile will be provisioned. After creating the device-profile, this value can't be changed.

LoRaWAN MAC version \*

1.1.0

The LoRaWAN MAC version supported by the device.

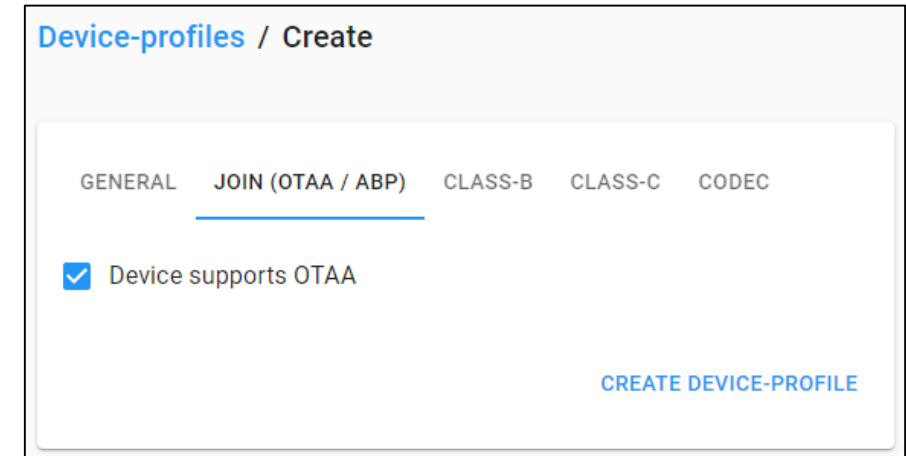
LoRaWAN Regional Parameters revision \*

A

Revision of the Regional Parameters specification supported by the device.

## Adding a Device Profile:

- Definition of the activation process in “JOIN”-tab
- Settings of the downlink windows RX1 and RX2
- List of factory preset frequency channels by the device
- Settings for Class-B if supported
- Settings for Class-C if supported



Device-profiles / Create

GENERAL JOIN (OTAA / ABP) CLASS-B CLASS-C CODEC

Device supports OTAA

CREATE DEVICE-PROFILE

## Adding an Application:

An application is a collection of devices with the same purpose / of the same type.

### Applications / Create

Application name \*

RN2483\_app

The name may only contain words, numbers and dashes.

Application description \*

Application to receive data from RN2483

Service-profile \*

Service Profile

The service-profile to which this application will be attached. Note that you can't change this value after the application has been created.

Payload codec

None

By defining a payload codec, ChirpStack Application Server can encode and decode the binary device payload for you. **Important note:** the payload fields have moved to the device-profile. For backward-compatibility and migration, existing codec settings are still visible. Codec settings on the device-profile have priority over the application codec settings.

[CREATE APPLICATION](#)



## Adding a Device:

Inside the application you can add devices

- Device EUI: Identifier of the device by the manufacturer.
- The device EUI of our RN2483 is *0004A30B001AEE05*

Applications / RN2483\_app / Devices / Create

GENERAL VARIABLES TAGS

Device name \*  
RN2483  
The name may only contain words, numbers and dashes.

Device description \*  
RN2483

Device EUI \*  
00 04 A3 0B 00 1A EE 05 MSB ↻

Device-profile \*  
RN2483 ▼

Disable frame-counter validation

Note that disabling the frame-counter validation will compromise security as it enables people to perform replay-attacks.

[CREATE DEVICE](#)

After these steps, it is now possible to join the network with our device:

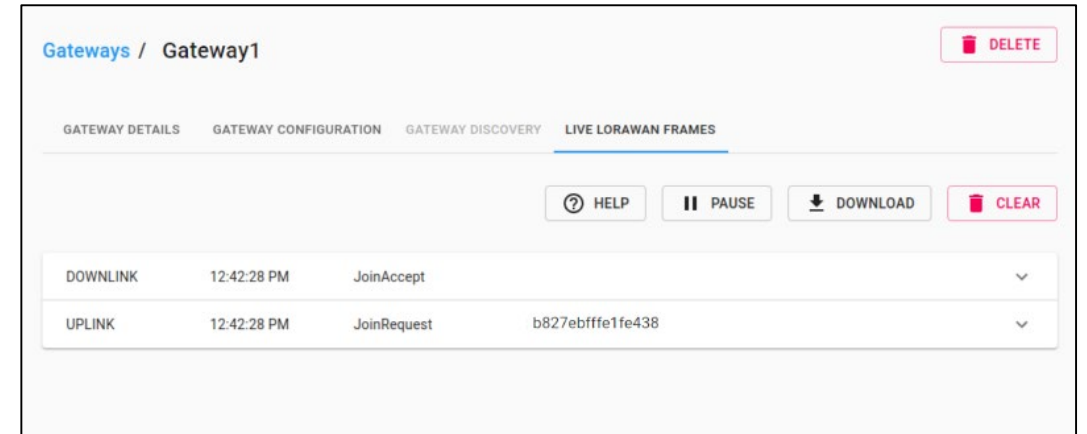
- Setting up the connection with the RN2483 via a RS232-terminal and the following settings:
- All commands to operate the device can be found in the RN2483 LoRa<sup>®</sup> Technology Module Command Reference User's Guide by Microchip. The commands for performing a OTAA join and an uplink message are:

```
bitrate: 57600
Data bits: 8
Stop bits: 1
Parity: none
Flow control: none
terminated with <CR><LF>
```

```
OTAA join: mac join otaa
TX message: mac tx <type> <portno> <data>
▪ <type>: cnf (confirmed) or uncnf (unconfirmed)
▪ <portno>: Port number (1 to 223)
▪ <data>: data message
```

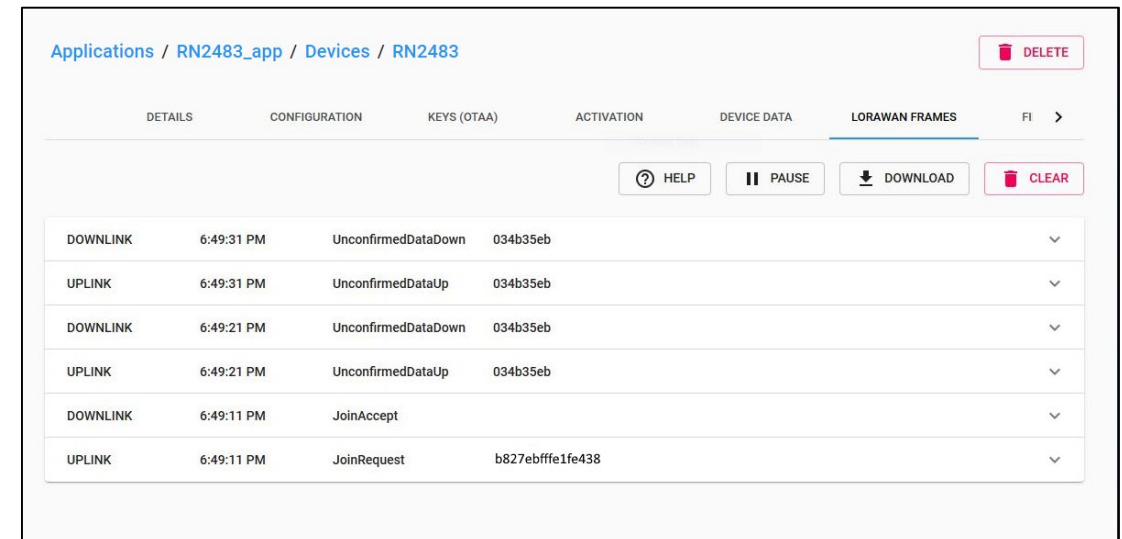
## Message Flow:

- The message flow of the LoRa messages received and transmitted by the gateway can be found in the gateway section in the web-interface.
  
- The LoRaWAN message flow can be found in the application section of the web-interface.



The screenshot shows the 'Gateway1' page in a web interface. The breadcrumb is 'Gateways / Gateway1'. There are tabs for 'GATEWAY DETAILS', 'GATEWAY CONFIGURATION', 'GATEWAY DISCOVERY', and 'LIVE LORAWAN FRAMES'. Below the tabs are buttons for 'HELP', 'PAUSE', 'DOWNLOAD', and 'CLEAR'. A table displays the message flow:

Direction	Time	Message Type	Message ID
DOWNLINK	12:42:28 PM	JoinAccept	
UPLINK	12:42:28 PM	JoinRequest	b827ebffe1fe438



The screenshot shows the 'RN2483' page in a web interface. The breadcrumb is 'Applications / RN2483\_app / Devices / RN2483'. There are tabs for 'DETAILS', 'CONFIGURATION', 'KEYS (OTAA)', 'ACTIVATION', 'DEVICE DATA', 'LORAWAN FRAMES', and 'FI >'. Below the tabs are buttons for 'HELP', 'PAUSE', 'DOWNLOAD', and 'CLEAR'. A table displays the message flow:

Direction	Time	Message Type	Message ID
DOWNLINK	6:49:31 PM	UnconfirmedDataDown	034b35eb
UPLINK	6:49:31 PM	UnconfirmedDataUp	034b35eb
DOWNLINK	6:49:21 PM	UnconfirmedDataDown	034b35eb
UPLINK	6:49:21 PM	UnconfirmedDataUp	034b35eb
DOWNLINK	6:49:11 PM	JoinAccept	
UPLINK	6:49:11 PM	JoinRequest	b827ebffe1fe438

## Message Flow:

An LoRaWAN message is divided in up to 3 parts

- RX info: contains information about the quality of the received signal like RSSI and SNR
- TX info: includes information of the LoRa parameters like spreading factor, code rate, bandwidth and frequency channel
- Physical payload

UPLINK 1:45:02 PM JoinRequest b827ebffe1fe438

```
rxinfo: [ ] 1 item
  0: [ ] 14 keys
    gatewayID: "647fdafffe008202"
    time: null
    timeSinceGPSEPOCH: null
    rssi: -9
    loRaSNR: 9.5
    channel: 1
    rfChain: 0
    board: 0
    antenna: 0
    location: [ ] 5 keys
      latitude: 0
      longitude: 0
      altitude: 0
      source: "UNKNOWN"
      accuracy: 0
      fineTimestampType: "NONE"
      context: "qJcrNA--"
      uplinkID: "a67b00f7-e0cc-4fe3-a221-7bc9625f72a7"
      crcStatus: "CRC_OK"
  txinfo: [ ] 3 keys
    frequency: 868100000
    modulation: "LORA"
    ioRaModulationInfo: [ ] 4 keys
      bandwidth: 125
      spreadingFactor: 12
      codeRate: "4/5"
      polarizationInversion: false
  phyPayload: [ ] 3 keys
    mhdr: [ ] 2 keys
      messageType: "JoinRequest"
      major: "LoRaWANR1"
    macPayload: [ ] 3 keys
      joinEUI: "58208a8800070000"
      devEUI: "58208a8800070000"
      devNonce: 48342
      mic: "ec529b78"
```

RX info

physical payload

TX info

UPLINK

DOWNLINK 1:45:02 PM JoinAccept

```
gatewayID: "647fdafffe008202"
txinfo: [ ] 9 keys
  frequency: 868100000
  power: 23
  modulation: "LORA"
  ioRaModulationInfo: [ ] 4 keys
    bandwidth: 125
    spreadingFactor: 12
    codeRate: "4/5"
    polarizationInversion: true
  board: 0
  antenna: 0
  timing: "DELAY"
  delayTimingInfo: [ ] 1 key
    delay: "5s"
    context: "qJcrNA=="
  phyPayload: [ ] 3 keys
    mhdr: [ ] 2 keys
      messageType: "JoinAccept"
      major: "LoRaWANR1"
    macPayload: [ ] 1 key
      bytes: "t/9feJ8AIBQJoiYF"
      mic: "224b5a45"
```

TX info

physical payload

DOWNLINK

## Message Flow OTAA join :

- Join Request
  - Contains joinEUI, devEUI, and devNonce for the calculation of the session keys
- Join Accept
  - Contains encrypted data like JoinNonce, NetID, and DevAddr

DOWNLINK	6:49:31 PM	UnconfirmedDataDown	26011a87
UPLINK	6:49:31 PM	UnconfirmedDataUp	26011a87
DOWNLINK	6:49:21 PM	UnconfirmedDataDown	26011a87
UPLINK	6:49:21 PM	UnconfirmedDataUp	26011a87
DOWNLINK	6:49:11 PM	JoinAccept	
UPLINK	6:49:11 PM	JoinRequest	b827ebfffe1fe438

JoinRequest:

```
▼ phyPayload: {} 3 keys
▼ mhdr: {} 2 keys
  mType: "JoinRequest"
  major: "LoRaWANR1"
▼ macPayload: {} 3 keys
  joinEUI: "58208a8800070000"
  devEUI: "58208a8800070000"
  devNonce: 48342
  mic: "ec529b78"
```

JoinAccept:

```
▼ phyPayload: {} 3 keys
▼ mhdr: {} 2 keys
  mType: "JoinAccept"
  major: "LoRaWANR1"
▼ macPayload: {} 1 key
  bytes: "t/9feJ8AIB0JoIYF"
  mic: "224b5a45"
```

## Message Flow after join procedure:

- In this example, several unconfirmed uplink messages were transmitted after the join procedure
- Every uplink is followed by a downlink with network parameters the device must adapt. In this case:
  - “RX ParamSetupReq”: sets the frequency and data rate of the second receive window.
  - “RX TimingSetupReq”: sets the delay between the end of the reception of an uplink and the opening of the first reception slot
- Further network parameters like maximum duty cycle or adding new channels
- A list of MAC Commands exchanged between receiver and transmitter are documented in the LoRaWAN specification by the LoRa Alliance

DOWNLINK	6:49:31 PM	UnconfirmedDataDown	26011a87
UPLINK	6:49:31 PM	UnconfirmedDataUp	26011a87
DOWNLINK	6:49:21 PM	UnconfirmedDataDown	26011a87
UPLINK	6:49:21 PM	UnconfirmedDataUp	26011a87
DOWNLINK	6:49:11 PM	JoinAccept	
UPLINK	6:49:11 PM	JoinRequest	b827ebfffe1fe438

### Uplink:

```

▼ phyPayload: {} 3 keys
▼ mhdr: {} 2 keys
  mType: "UnconfirmedDataUp"
  major: "LoRaWANR1"
▼ macPayload: {} 3 keys
▼ fhdr: {} 4 keys
  devAddr: "26011a87"
▼ fCtrl: {} 5 keys
  adr: true
  adrAckReq: false
  ack: false
  fPending: false
  classB: false
  fCnt: 1
  fOpts: null
  fPort: 1
▼ frmPayload: [] 1 item
  0: {} 1 key
    bytes: "b2Y3lX9O/1/mhRlabQ=="
  mic: "cee0d2e2"
    
```

### Downlink:

```

▼ phyPayload: {} 3 keys
▼ mhdr: {} 2 keys
  mType: "UnconfirmedDataDown"
  major: "LoRaWANR1"
▼ macPayload: {} 3 keys
▼ fhdr: {} 4 keys
  devAddr: "26011a87"
▼ fCtrl: {} 5 keys
  adr: true
  adrAckReq: false
  ack: false
  fPending: false
  classB: false
  fCnt: 1
▼ fOpts: [] 2 items
  0: {} 2 keys
    cid: "RXParamSetupReq"
    payload: {} 2 keys
      frequency: 868300000
      dlSettings: "08"
  1: {} 2 keys
    cid: "RXTimingSetupReq"
    payload: {} 1 key
      delay: 1
  fPort: null
  frmPayload: null
  mic: "579fcb05"
    
```

## Message Flow after join procedure:

- There are two different frame counters that keep track of uplink and downlink frames:
  - FcntUp: Number of uplink messages, transmitted and incremented by the device via “fCnt” field
  - FcntDown: Number of downlink messages, transmitted and incremented by the network server via “fCnt” field
- If either the device or the network receives a message with a frame counter that is lower than the last one, the message is ignored

DOWNLINK	6:49:31 PM	UnconfirmedDataDown	26011a87
UPLINK	6:49:31 PM	UnconfirmedDataUp	26011a87
DOWNLINK	6:49:21 PM	UnconfirmedDataDown	26011a87
UPLINK	6:49:21 PM	UnconfirmedDataUp	26011a87
DOWNLINK	6:49:11 PM	JoinAccept	
UPLINK	6:49:11 PM	JoinRequest	b827ebfffe1fe438

### Uplink:

```
▼ phyPayload: {} 3 keys
▼ mhdr: {} 2 keys
  mType: "UnconfirmedDataUp"
  major: "LoRaWANR1"
▼ macPayload: {} 3 keys
▼ fhdr: {} 4 keys
  devAddr: "26011a87"
▼ fCtrl: {} 5 keys
  adr: true
  adrAckReq: false
  ack: false
  fPending: false
  classB: false
  fCnt: 1
  fOpts: null
  fPort: 1
▼ frmPayload: [] 1 item
  0: {} 1 key
    bytes: "b2Y3lX9O/1/mhRlabQ=="
  mic: "cee0d2e2"
```

### Downlink:

```
▼ phyPayload: {} 3 keys
▼ mhdr: {} 2 keys
  mType: "UnconfirmedDataDown"
  major: "LoRaWANR1"
▼ macPayload: {} 3 keys
▼ fhdr: {} 4 keys
  devAddr: "26011a87"
▼ fCtrl: {} 5 keys
  adr: true
  adrAckReq: false
  ack: false
  fPending: false
  classB: false
  fCnt: 1
▼ fOpts: [] 2 items
  0: {} 2 keys
    cid: "RXParamSetupReq"
    payload: {} 2 keys
      frequency: 868300000
      dlSettings: "08"
  1: {} 2 keys
    cid: "RXTimingSetupReq"
    payload: {} 1 key
      delay: 1
  fPort: null
  frmPayload: null
  mic: "579fcb05"
```

## Message Flow ADR:

If the ADR flag is set to one, the network server can adapt parameters of spreading factor (data rate), power, and useable channels for an end device.

## Uplink message:

- ADR bit is set to one -> ADR is activated
- Spreading factor 12

UPLINK	7:01:25 PM	UnconfirmedDataUp	26011a87
DOWNLINK	7:01:13 PM	UnconfirmedDataDown	26011a87
UPLINK	7:01:13 PM	UnconfirmedDataUp	26011a87

```
▼ txInfo: {} 3 keys
  frequency: 868300000
  modulation: "LORA"
  ▼ loRaModulationInfo: {} 4 keys
    bandwidth: 125
    spreadingFactor: 12
    codeRate: "4/5"
    polarizationInversion: false
```

```
▼ phyPayload: {} 3 keys
  ▼ mhdr: {} 2 keys
    mType: "UnconfirmedDataUp"
    major: "LoRaWANR1"
  ▼ macPayload: {} 3 keys
  ▼ fhdr: {} 4 keys
    devAddr: "26011a87"
  ▼ fCtrl: {} 5 keys
    adr: true
    adrAckReq: false
    ack: false
    fPending: false
    classB: false
    fCnt: 3
    fOpts: null
    fPort: 6
  ▼ frmPayload: [] 1 item
    ▼ 0: {} 1 key
      bytes: "et4="
    mic: "b4198459"
```



## Message Flow ADR:

A downlink message follows with a “LinkADRReq” command to set new parameters for the device:

- dataRate: 5
- txPower: 5
- chMask: enabled channels; 0, 1, and 2 are the mandatory channels

UPLINK	7:01:25 PM	UnconfirmedDataUp	26011a87
DOWNLINK	7:01:13 PM	UnconfirmedDataDown	26011a87
UPLINK	7:01:13 PM	UnconfirmedDataUp	26011a87

```
▼ phyPayload: {} 3 keys
  ▼ mhdr: {} 2 keys
    mType: "UnconfirmedDataDown"
    major: "LoRaWANR1"
  ▼ macPayload: {} 3 keys
    ▼ fhdr: {} 4 keys
      devAddr: "26011a87"
    ▼ fCtrl: {} 5 keys
      adr: true
      adrAckReq: false
      ack: true
      fPending: false
      classB: false
      fCnt: 3
    ▼ fOpts: [] 1 item
      ▼ 0: {} 2 keys
        cid: "LinkADRReq"
        ▼ payload: {} 4 keys
          dataRate: 5
          txPower: 5
          ▼ chMask: [] 16 items
            0: true
            1: true
            2: true
            3: false
            4: false
            5: false
            6: false
            7: false
            8: false
            9: false
            10: false
            11: false
            12: false
            13: false
            14: false
            15: false
          ▼ redundancy: {} 2 keys
            chMaskCntl: 0
            nbRep: 1
```

## Message Flow ADR:

- Next uplink message adapted the parameters from the ADR request
- “LinkADRs” includes acknowledgments of the changed power, data rate, and channel mask.
- The network server changes transmission parameters until suitable parameters in terms of connectivity and battery efficiency is achieved

UPLINK	7:01:25 PM	UnconfirmedDataUp	26011a87
DOWNLINK	7:01:13 PM	UnconfirmedDataDown	26011a87
UPLINK	7:01:13 PM	UnconfirmedDataUp	26011a87

```
▼ txInfo: {} 3 keys
  frequency: 868300000
  modulation: "LORA"
  ▼ loRaModulationInfo: {} 4 keys
    bandwidth: 125
    spreadingFactor: 7
    codeRate: "4/5"
    polarizationInversion: false
```

```
▼ phyPayload: {} 3 keys
  ▼ mhdr: {} 2 keys
    mType: "UnconfirmedDataUp"
    major: "LoRaWANR1"
  ▼ macPayload: {} 3 keys
    ▼ fhdr: {} 4 keys
      devAddr: "26011a87"
    ▼ fCtrl: {} 5 keys
      adr: true
      adrAckReq: false
      ack: false
      fPending: false
      classB: false
      fCnt: 2
    ▼ fOpts: [] 1 item
      ▼ 0: {} 2 keys
        cid: "LinkADRs"
        ▼ payload: {} 3 keys
          channelMaskAck: true
          dataRateAck: true
          powerAck: true
      fPort: 4
  ▼ frmPayload: [] 1 item
    ▼ 0: {} 1 key
      bytes: "p3+VVj+u"
    mic: "c18eb467"
```

## Message Flow:

- After several responses to an uplink message with network and transmission parameter settings, there is no downlink response to an unconfirmed message.

UPLINK	6:15:16 PM
UPLINK	6:14:02 PM
UPLINK	6:12:48 PM
DOWNLINK	6:11:35 PM
UPLINK	6:11:35 PM
DOWNLINK	6:10:22 PM
UPLINK	6:10:22 PM
DOWNLINK	6:10:08 PM
UPLINK	6:10:08 PM
DOWNLINK	6:10:02 PM
UPLINK	6:10:02 PM