

# POLITECNICO DI TORINO

Department of Electronics and Telecommunications

## Communications and Computer Networks Engineering



**Degree of Master of Science in**

## **Design of Low Power Gateway for LoRaWAN Applications in Remote Areas**

**supervisors:**

Dr. Daniele Trinchero

**Co-supervisor:**

Dr. Giovanni Colucci

**Author:**

Behnaz Asadollahseraj

December 2018

## Abstract

Nowadays, the demand for connecting to the internet for exchanging information everywhere, every time and anyhow is noticeable. Thanks to Information communication technologies (ICT) services, through The Internet of Things commonly abbreviated “IoT” that focuses on inert-connecting of devices or things to each other and to the users. the main descriptive term of vision in IoT is shift the connections from “users to things” to “things to things”. Today the size of IoT has been increasing swiftly and more and more devices are being introduced into the IoT networks every day; also is expected to grow appreciably in the immediate future.

The main aim of this thesis is to design a Low-Power Gateway and implement IoT system based on Low-Power Wide-Area Networks (LPWANs) and in this project focuses on specific LPWAN, namely, “LoRaWAN” protocol. LoRaWAN is able to communicate and transmit messages between devices and gateways. The LoRaWAN specification provides seamless interoperability among smart things without the need of complex local installations, enabling the rollout of IoT applications, according to the association.

In this thesis, to further familiarize the reader with the foundations of LPWAN technology, a completely open-source LoRaWAN server is set up on a Raspberry Pi Linux-operated device and communications between the Network server and Gateways are discussed.

The project had two phases. The first used The Things Network and Lora server for designing and implementing of network server for LoRaWAN Application. In these methods the connection between network-server and gateway must be continual and essential Which is the main cause of energy consumption. The second phase investigated about eliminate the power-line for connection between network-server and gateway by building a Low-Power Gateway on Raspberry Pi.

Keywords: IoT, LoraWAN, LoRa, Lora Server, TTN

## Acknowledgments

First of all, I would like to thank Prof. Dr. Daniele Trincherò for giving me the opportunity to write my Master thesis on the very current and fascinating topic of Low Power Wide Area Network at his laboratory 'IXEM' and Group at the Department of Electronics and Telecommunications at the Politecnico di Torino University.

I would also like to thank Dr. Giovanni Colucci, who fulfilled his role as advisor for this thesis with great enthusiasm. also for his guidance and advise on many issues including the technical correctness of this thesis. his valuable feedback helped to improve both the solution as well as this written document. Also, his positive attitude encouraged me throughout the process to do my best for this thesis. Furthermore, I would like to thank IXem team Dr.Mattia Poletti and Mr.Paolo Cielo, for their valuable insights into LoRaWAN and helped me to further improve the object tracking device and which I am very thankful for.

Finally,I acknowledge the people who mean a lot to me, my parents, for showing faith in me and giving me liberty to choose what I desired. I salute you all for the selfless love, care, pain and sacrifice you did to shape my life. Although you hardly understood what I researched on, you were willing to support any decision I made. I would never be able to pay back the love and affection showered upon by my parents.

## Table of Contents

Acknowledgments.....	ii
List of Figures .....	vi
Chapter 1. Internet Of Things .....	8
1-1    Internet of Things Overview .....	8
1-2    IoT Building Block .....	9
1-3    Conclusion: Why IoT? .....	11
Chapter 2. Solution For IoT Connectivity .....	13
2-1    LPWANs Low-Power Wide-Area Networks.....	13
2-2    LoRa™ PHY layer.....	13
2-2-1    LoRa Characteristics .....	14
2-3    LoRaWAN™ .....	15
2-3-1    LoRaWAN™ MAC Layer .....	15
2-3-2    LoRaWAN Frequency Bands.....	16
2-3-3    LoRaWAN™ Network Architecture .....	17
2-3-4    LoRaWAN Endpoint Classes.....	19
2-3-5    MAC Message Formats.....	24
2-3-6    Encryption and Device Activation .....	29
Chapter 3. The Things Network.....	32
3-1    Network Architecture.....	32
3-2    TTN Packet Forwarder .....	36

3-3	Setting up a Private TTN .....	37
Chapter 4. LoRaServer.....		39
4-1	LoRaServer Architecture .....	39
4-1-1	Gateway.....	41
4-1-2	End-Node .....	42
4-2	LoRaServer Installation and setup .....	42
4-2-1	Requirements.....	43
4-3	setting up the gateway:.....	48
4-4	Installing the LoRa Server project .....	48
4-4-1	installing LoRa-Gateway-Bridge .....	49
4-4-2	installing LoRa Server.....	50
4-4-3	installing LoRa App Server.....	51
Chapter 4. The Architecture of Gateway .....		64
4-1	LoRa Gateway Architecture .....	65
4-1-1	Hardware .....	65
4-2-2	Software .....	65
Chapter 5. Experiments on the Network and Programming .....		67
5-1	A Python script to automatically send data to the end- nodes .....	68
Chapter 6. Conclusion.....		70
5-1	Technical Conclusion.....	70
Appendix 1. Setting Up LoRaServer Using the Docker .....		72
Appendix 2. A Code to subscribe to MQTT and send data to the end-nodes.....		73
Bibliography .....		75



## List of Figures

Figure 1. IoT Building Block.....	9
Figure 2. Internet of Things. ....	12
Figure 3. LoRaWAN Protocol Stack. ....	16
Figure 4. LoRaWAN Architecture.....	19
Figure 5. Class A Receive Windows. ....	20
Figure 6. Class B Receive Windows.....	21
Figure 7. Class C Receive Windows.....	22
Figure 8. LoRaWAN <sup>TM</sup> classes comparative.....	23
Figure 9. LoRa Message Format.....	23
Figure 10. PHY payload. ....	25
Figure 11. LoRa Message Type. ....	25
Figure 12. Encryption Format.....	31
Figure 13. TTN Architecture. ....	33
Figure 14. Uplink and Downlink Message Process. ....	35
Figure 15. Packet Forwarder.....	36
Figure 16. LoRa Server Architecture.....	40
Figure 17. End-node. ....	42
Figure 18. MQTT Sub/Pub. ....	43
Figure 19. MQTT Sub/Pub Example.....	44
Figure 20. Broker Example.....	45

Figure 21. LoRa-App-Server Web Interface .....	53
Figure 22. Low-Power Gateway Architecture.....	65
Figure 23. Mosquitto Subscribe topic.....	67
Figure 24. Python script at work.....	68



## Chapter 1.

### Internet Of Things

The internet of things “IoT” is the new concept in Information and communications technology or (ICT) though we can say it is an old term; since it was presented by Kevin Ashton in 1999 at Proctor & Gamble. he Linked the new idea of RFID (radio frequency identification) hot topic of the Internet was a good way to get executive attention [1].

#### **1-1 Internet of Things Overview**

Nowadays, IoT is growing at an exponential rate. Vehicles, wearable gadgets, RFID sensors and software, are advancing past basic function and the network is growing to include even more. In 2008, it was noted that the number of objects connected to the internet was more than the number of people connected.

The IoT can be viewed from different prospects, we can consider communication and connection prospects. from these perspective anything from everywhere in anytime can connect to network of interconnected things. Hence IoT refers to a recent paradigm supports transfer and analytics of data generated by smart devices (e.g. sensors).

The IoT provides objects to connect, communicate together and be controlled remotely through applications by existing internet infrastructures. With IoT, users can share both information provided by humans that contained in databases and also information provided by things in physical world therefore it enables a direct integration and communication between the physical and the digital worlds.

By connecting the devices to each other and the internet, are some of the top uses of IoT such as smart homes, smart city, IoT in agriculture, Smart Retail, Healthcare, Farming and etc.

## 1-2 IoT Building Block

clearly this is an interesting point that IoT is not a single technology; this is an aggregation of technology that work together in an organization. Internet of Things evolves the network of physical objects with sensors and actuators, software and network connectivity that enable these objects to gather and transmit data and user's tasks.

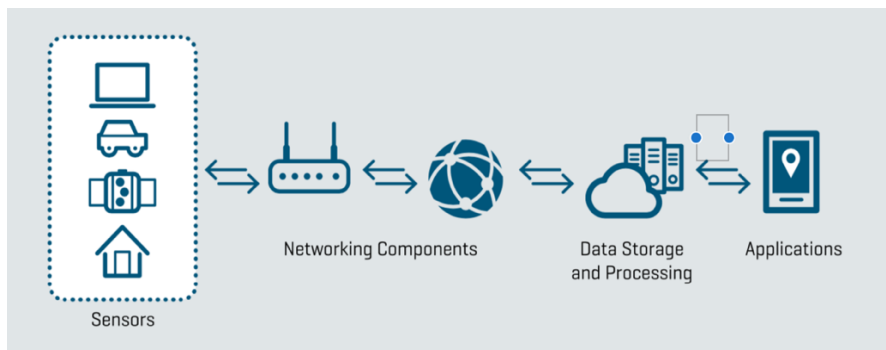


Figure 1. IoT Building Block.

- **Sensors and Actuator:** Sensors are the key components in IoT. this technology is impossible without sensors because iot is context awareness. one or more sensors collect real data from environment and convert that data into the machine understandable codes. IoT sensors are mostly small in size, have low cost, and consume less power. There are available specific purposes of sensors, such as measuring temperature, humidity, motion, light, pressure, altitude and color. These sensors are used in almost all the IoT applications.

in simple word an actuator is a device that acts inverse of sensors. It takes an electrical input and turns it into physical action. As an example, we can consider a smart home system, which consists of many sensors and actuators. The actuators are used to lock/unlock the doors, switch on/off the lights or other electrical appliances, alert users of any threats through alarms or notifications, and control the temperature of a home.

- **Networking:** This is the main aim of my project. physical objects called things, sensors, actuators, human are connected together by networks, by using various wireless technologies, standards and protocols. In fact network is responsible for connecting to other smart things, network devices, and servers. Its features are also used for transmitting and processing sensor data. IoT network technologies to be aware of toward the bottom of the protocol stack include cellular, wifi, and Ethernet, as well as more specialized solutions such as LPWAN, Bluetooth Low Energy (BLE), ZigBee, NFC, and RFID. We explain about LPWAN solution complete in next chapters in detailed.
- **Data storage and processing:** The storage and processing of data can be done on the edge of the network itself or in a remote server. If any preprocessing of data is possible, then it is typically done at either the sensor or some other proximate device. The processed data is then typically sent to a remote server. The storage and processing capabilities of an IoT object are also restricted by the resources available, which are often very constrained due to limitations of size, energy, power, and computational capability. As a result, the main research challenge is to ensure that we get the right kind of data at the desired level of accuracy. Along with

the challenges of data collection, and handling, there are challenges in communication as well. The communication between IoT devices is mainly wireless because they are generally installed at geographically dispersed locations.

- **Application:** The IoT has huge potential for developing new intelligent applications in nearly every domain, such as personal, social, societal, medical, environmental and logistics aspects. The number of application domains has been also increasing due to its ability to perform contextual sensing. It allows, for instance, to collect information of environment, natural phenomena, medical parameters and user habits and then can offer tailored services based on information received. Such phenomenon should enhance the quality of everyday life, and should have a reflective impact on the society and economy irrespective of the application domain. Globally, various applications domains can be categorized in three major areas: smart city domain, industrial domain, and health and well-being domain. In fact, each domain is partially or completely overlapped but is not isolated from the others since most of the applications are common and share the same resources.

### **1-3 Conclusion: Why IoT?**

Nowadays, information and communication technologies are become one of the essential elements of today's life and society. Both public and private parts across the world are transforming their countries and businesses with ICT programs ranging from research and innovation, infrastructure building, and skills development. internet of Things (IoT) as one of the ICT innovations has attracted attention of companies across the world in the past decade. It describes a world where anything can be connected and can interact in an

intelligent fashion. Therefore, it is popular to realize where internet connectivity and computing capability extends to a variety of connecting things.

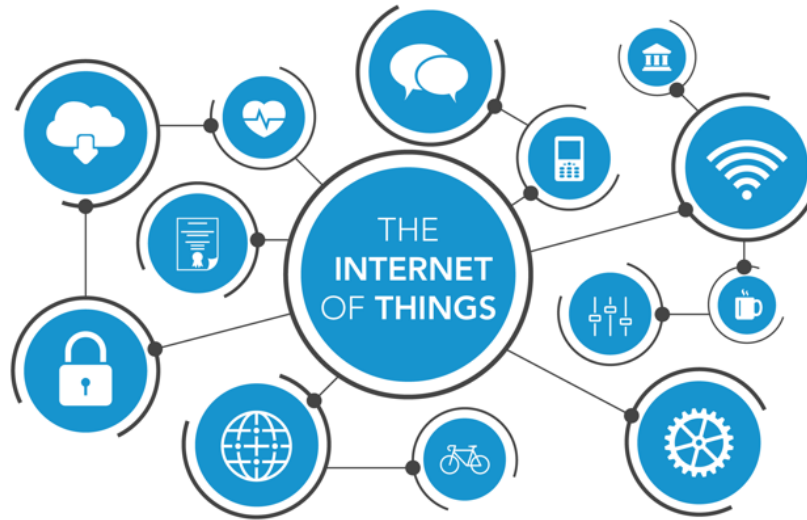


Figure 2. Internet of Things.

## Chapter 2.

### Solution For IoT Connectivity

In this project we implement an IoT system using a LoRaWAN private network. so we provide a preliminary technical overview to LPWAN, LoRa and LoRaWAN. Low Power, Wide-Area Networks (LPWAN) are projected to support a major portion of the billions of devices forecasted for the IoT.

#### **2-1 LPWANs Low-Power Wide-Area Networks**

The IoT movement creates the need for new wireless technologies, capable of supporting the large numbers of devices found in the IoT space. A low-power wide-area networks are wireless wide area networks designed to allow long range communications with a low bit rates. The great benefit brought by LPWANs is an increased power efficiency. Particularly, they proved effective in typical IoT applications such as environmental monitoring and smart metering.

#### **2-2 LoRa<sup>TM</sup> PHY layer**

LoRa<sup>TM</sup> is the physical layer or the wireless modulation utilized to create the long range communication link. LoRa<sup>TM</sup> is based on Chirp Spread Spectrum (CSS) modulation, which maintains the same low power characteristics as FSK modulation but significantly increases the communication range [2]. Chirp Spread Spectrum has been used in military and space communication for decades due to the long communication distances that can be achieved and due to its robustness to interferences. Nevertheless, LoRa<sup>TM</sup> arrives as the

first low cost implementation for commercial usage [3]. With LoRa<sup>TM</sup>, the communication between end-devices and gateways is spread out on different frequency channels and data rates.

The advantage of LoRa<sup>TM</sup> is in the technology's long range capability. A single gateway can cover entire cities or hundreds of square kilometers. Range highly depends on the environment or obstructions in a given location, but LoRa<sup>TM</sup> has a link budget greater than any other standardized communication technology.

With LoRa<sup>TM</sup>, the communication between end-devices and gateways is spread out on different frequency channels and data rates. The selection of the data rate is a trade-off between communication range and message duration, communications with different data rates do not interfere with each other. LoRa data rates range from 0.3 kbps to 20 kbps [4].

Spreading Factor	Bandwidth [kHz]	Spreading Factor [chips/symbol]	Bit rate of the signal [bits/sec]	Chip rate [chips/sec]	Time per symbol [sec/symbol]
SF7	125	128	5469	125000	0,001024
SF8	125	256	3125	125000	0,002048
SF9	125	512	1758	125000	0,004096
SF10	125	1024	977	125000	0,008192
SF11	125	2048	537	125000	0,016384
SF12	125	4096	293	125000	0,032768

Table 1. Spreading Factors Attributes.

### 2-2-1 LoRa Characteristics

aspects of using a LoRa radio in in a sensor networks are:

first aspect is range, since range is large about hundreds meters' network can span large areas without routing over many hops. Second aspect, if transmission is on the same carrier frequency but spreading factor are different are orthogonal.

## 2-3 LoRaWAN™

LoRaWAN is a low power wide area network (LPWAN) specification intended for wireless battery operated things in a regional, national or global network. The LoRaWAN specification provides seamless interoperability among smart things without the need of complex local installations, enabling the rollout of IoT applications, according to the association.

### 2-3-1 LoRaWAN™ MAC Layer

LoRaWAN™ defines Media Access Control (MAC) layer protocol and system architecture for the network while the LoRa™ physical layer (PHY) enables the long-range communication link. The protocol and network architecture have the most influence in determining the battery lifetime of a node, the network capacity, the quality of service, the security, and the variety of applications served by the network. LoRaWAN™ is open-source and it is assembled by the LoRa™ Alliance.

The LoRaWAN protocol stack is presented in Figure 2. The stack consists of an application layer, a MAC layer and a PHY layer. Data from the application is mapped into the MAC Payload, then the MAC layer constructs the MAC frame using MAC payload. Finally, the PHY layer uses MAC frame as PHY payload and constructs the PHY frame after inserting preamble, PHY header, CRC and entire frame CRC. This final frame is transmitted into the air on the required RF carrier. The RF parameters including frequencies, bands, power levels, modulation and the basic RF protocols are all encapsulated in the LoRa™ RF or physical layer attributes.



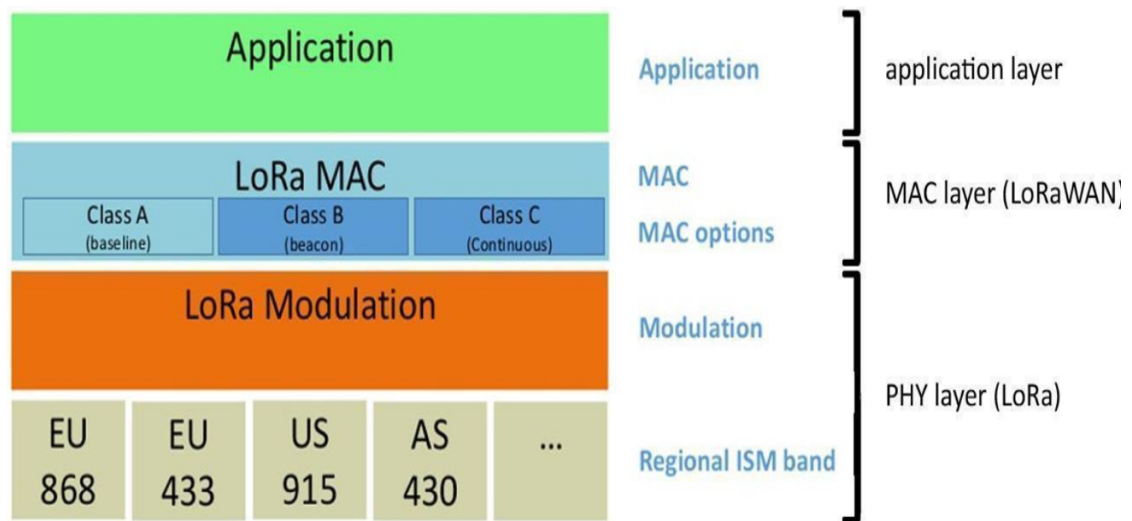


Figure 3. LoRaWAN Protocol Stack.

### 2-3-2 LoRaWAN Frequency Bands

LoRaWAN operates in unlicensed radio spectrum. The fact that frequencies have a longer range also comes with more restrictions that are often country-specific. This poses a challenge for LoRaWAN, that tries to be as uniform as possible in all different regions of the world. As a result, LoRaWAN is specified for a number of bands for these regions. These bands are similar enough to support a region-agnostic protocol, but have a number of consequences for the implementation of the backend systems.

Using lower frequencies than those of the 2.4 or 5.8 GHz ISM bands enables much better coverage to be achieved by the LoRa wireless modules and devices, especially when the nodes are within buildings.

Although the sub-1GHz ISM bands are normally used, the technology is essentially frequency agnostic and can be used on most frequencies without fundamental adjustment.

	Europe	North America	China	Korea	Japan	India
Frequency band	867-869MHz	902-928MHz	470-510MHz	920-925MHz	920-925MHz	865-867MHz
Channels	10	64 + 8 + 8	In definition by Technical Committee	In definition by Technical Committee	In definition by Technical Committee	In definition by Technical Committee
Channel BW Up	125/250kHz	125/500kHz				
Channel BW Dn	125kHz	500kHz				
TX Power Up	+14dBm	+20dBm typ (+30dBm allowed)				
TX Power Dn	+14dBm	+27dBm				
SF Up	7-12	7-10				
Data rate	250bps- 50kbps	980bps-21.9kbps				
Link Budget Up	155dB	154dB				
Link Budget Dn	155dB	157dB				

Table 2: LoRaWAN Protocol Stack

These LoRaWAN regional specifications do not specify everything either. They only cover a region by specifying the common denominator. For example, the LoRaWAN regional parameters for Asia only specify a common subset of channels - but there are variations between regulations in Asian countries. Furthermore, each network server operator is free to select additional parameters, such as additional emission channels.

### 2-3-3 LoRaWAN<sup>TM</sup> Network Architecture

Most of the modern IoT LAN technologies use mesh network architecture. By using mesh network, the system can increase the communication range and cell size of the network. But, nodes in a mesh network has additional responsibility of forwarding messages to other nodes, typically irrelevant to them. This affect the device battery life significantly.

LoRaWAN uses star topology as it increases battery lifetime when long-range connectivity is used. A LoRa network consists of several elements:

**End points:** The endpoints are the elements of the LoRa network where the sensing or control is undertaken. They are normally remotely located.

**LoRa gateway:** The gateway receives the communications from the LoRa endpoints and then transfers them onto the backhaul system. This part of the LoRa network can be Ethernet, cellular or any other telecommunications link wired or wireless. The gateways are connected to the network server using standard IP connections. On this way the data uses a standard protocol, but can be connected to any telecommunications network, whether public or private. In view of the similarity of a LoRa network to that of a cellular one, LoRaWAN gateways may often be co-located with a cellular base station. In this way they are able to use spare capacity on the backhaul network.

**LoRa Network Server:** The LoRa network server manages the network and as part of its function it acts to eliminate duplicate packets, schedules acknowledgement, and adapts data rates. In view of the way in which it can be deployed and connected, makes it very easy to deploy a LoRa network.

**Remote computer:** a remote computer can then control the actions of the endpoints or collect data from them - the LoRa network being almost transparent.

In terms of the actual architecture for the LoRa network, the nodes are typically in a star-of-stars topology with gateways forming a transparent bridge. These relay messages between end-devices and a central network server in the backend.

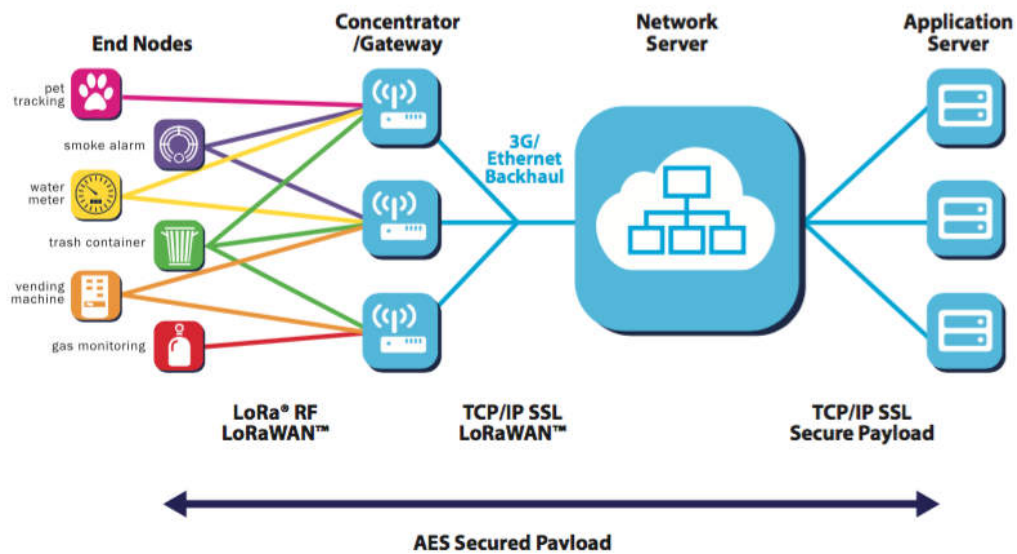


Figure 4. LoRaWAN Architecture.

### 2-3-4 LoRaWAN Endpoint Classes

Like other networks, where end devices can have different capabilities depending on devices classes, end nodes in LoRaWAN network can have different device classes. Each device class is a trade-off between network downlink communication latency verses battery-life.

This three classes serve different applications and have different requirements in order to optimize a variety of end applications. The device classes trade off network downlink communication latency versus battery lifetime. In a control or actuator-type application, the downlink communication latency is an important factor.

- **Class A:** LoRaWAN class A endpoint devices provide bidirectional communications. To achieve this, each endpoint transmission is followed by two short downlink receive windows. The transmission slot scheduled by the particular

endpoint is based upon the needs of the end point and also there is a small variation determined using a random time basis.

LoRa Class A operation provides the lowest power option for end points that only require downlink communication from the server shortly after the end-device has sent an uplink transmission. Downlink communications from the server at any other time wait until the next scheduled uplink time.

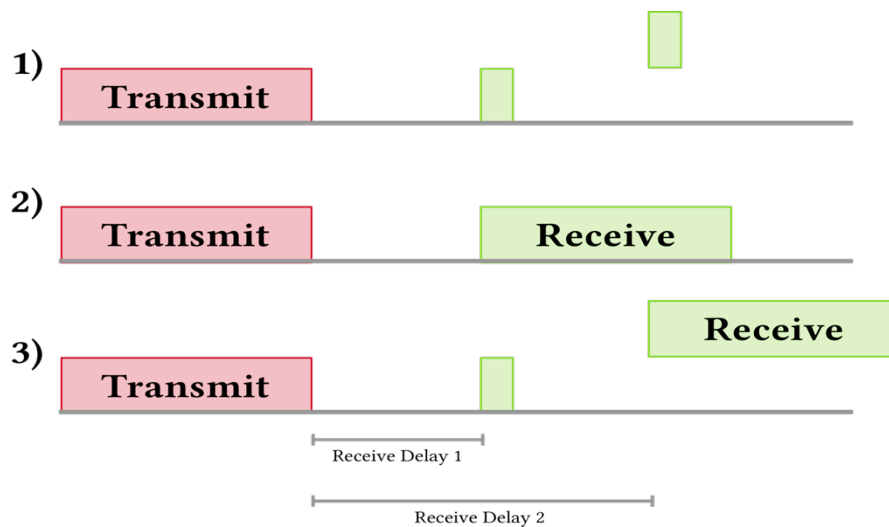


Figure 5. Class A Receive Windows.

- **Class B:** Class B: End-devices of class B allow for more receive slots. In addition to the class A random receive windows, class B devices open extra receive windows at scheduled times. In order for the end-device to open its receive window at the scheduled time it receives a time synchronized beacon from the gateway. This allows the server to know when the end-device is listening.

This class can be useful for battery powered devices where bidirectional sensors links are applicable, such as, reading a sensor with occasional control/configuration, alarm sensors with guaranteed alarm delivery.

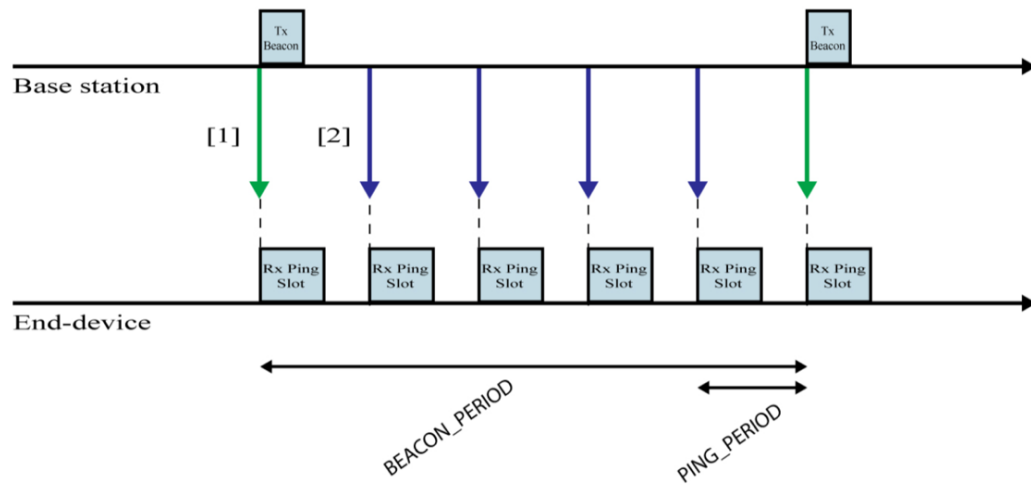


Figure 6. Class B Receive Windows.

1- All gateways synchronously by the network server must broadcast a beacon providing a timing reference to the end-devices.

2- End-devices open periodically receive windows called "ping slots", which can be used by the network infrastructure to initiate a downlink communication.

- **Class c:** devices extend Class A by keeping the receive windows open unless they are transmitting, as shown in the figure below. This allows for low-latency communication but is many times more energy consuming than Class A devices. This class can be useful for powered devices where a downlink communication is required at any moment: industrial control, real time control of pumps/valves, residential gateways, lighting control, car engine status, car tracking....

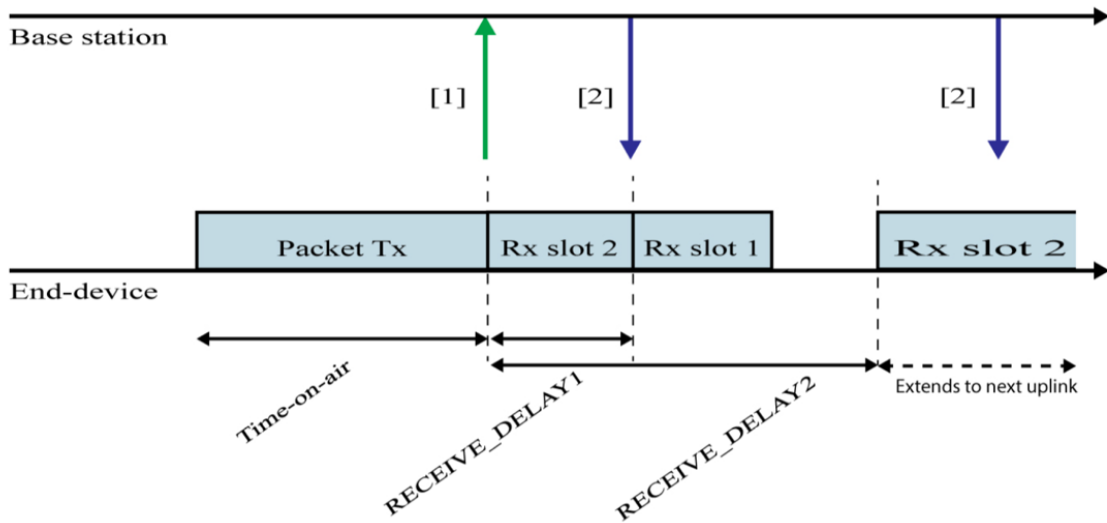


Figure 7. Class C Receive Windows.

1- Class C implements the same two receive windows as class A, with the difference that they do not close the second reception window until the next packet needs to be send. A short listening window is open between the end of the transmission and the beginning of the reception slot 1.

2- Gateway can communicate with the end-devices at any time.

This three classes serve different applications and have different requirements in order to optimize a variety of end applications. The device classes trade off network downlink communication latency versus battery lifetime. In a control or actuator-type application, the downlink communication latency is an important factor. Figure 7 compares latency of these classes with battery life.

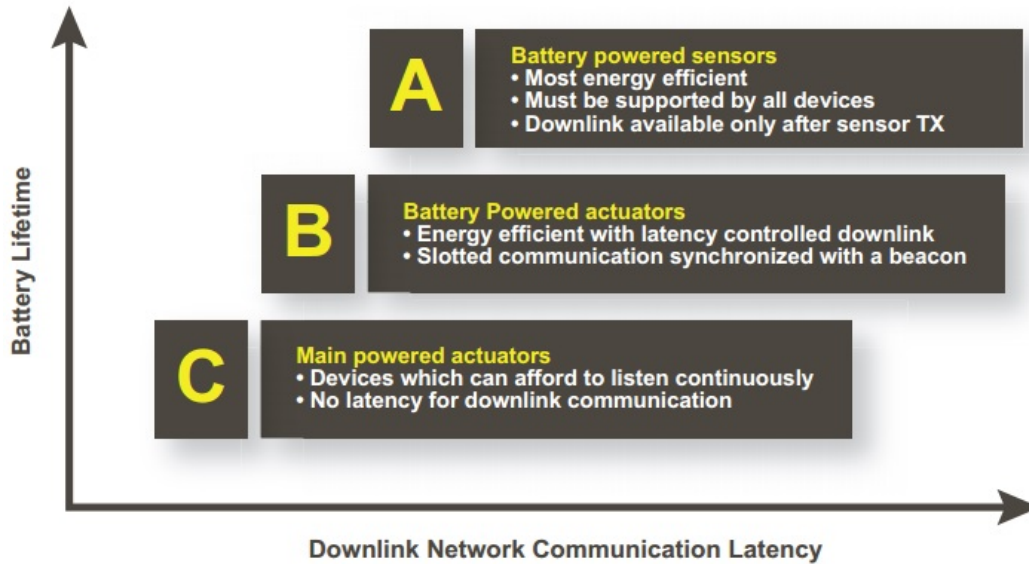


Figure 8. LoRaWAN™ classes comparative

LoRa™ messages are divided to uplink and downlink messages. Uplink messages are sent by end-devices to the network server relayed by one or many gateways, a downlink message is sent by the network server to only one end-device and is relayed by a single gateway.

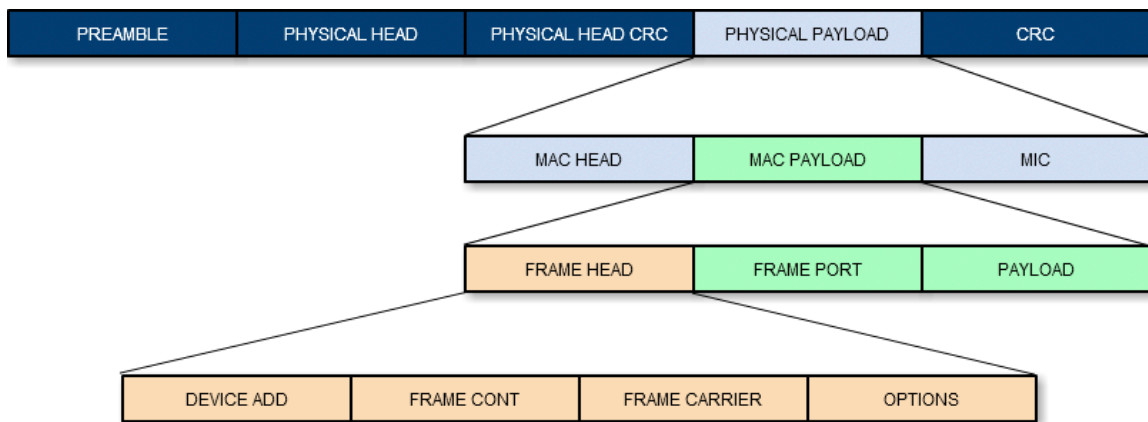


Figure 9. LoRa Message Format



Depending upon the chosen mode of operation two types of header are available.

In Explicit mode, header is the default mode of operation and provides information on the payload, namely: the payload length in bytes, the forward error correction code rate and the presence of an optional 2 Bytes CRC of the payload. In this case the header is transmitted with maximum error correction code. It also has its own CRC to allow the receiver to discard invalid headers.

In Implicit mode, where the payload, coding rate and CRC presence are fixed or known in advance, it may be advantageous to reduce transmission time by invoking implicit header mode. In this mode the header is removed from the packet. In this case the payload length, error coding rate and presence of the payload CRC must be manually configured on both sides of the radio link.

Uplink and downlink messages use the LoRa radio packet explicit mode in which the LoRa physical header (PHDR) plus a header CRC (PHDR\_CRC) are included.

Radio PHY layer				
Preamble	PHDR (only in explicit mode)	PHDR_CRC (only in explicit mode)	PHYPayload	CRC (optional)

Table 3. LoRa Physical Layer Format.

### 2-3-5 MAC Message Formats

All LoRa uplink and downlink messages carry a PHY payload (Payload) starting with a single- octet MAC header (MHDR), followed by a MAC payload (MAC Payload), and ending with a 4-octet message integrity code (MIC).

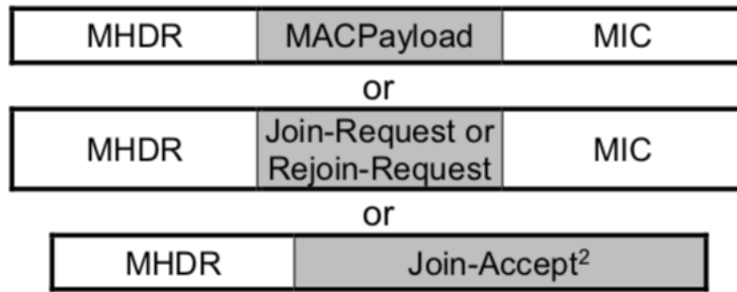


Figure 10. PHY payload.

- Join packets are the first packet that is sent by a device attempting to enter a network.
- Data messages can be either uplink or downlink. Additionally, it is stated whether the message requires an acknowledgement or not. A **confirmed-data message** must be acknowledged by the receiver, whereas an **unconfirmed-data message** does not require an acknowledgment.
- Proprietary messages can be used to implement non-standard message formats that can be used between devices having a common understanding of some proprietary extensions.

Message Type	Description
<b>000</b>	Join Request
<b>001</b>	Join Accept
<b>010</b>	Unconfirmed Data Up
<b>011</b>	Unconfirmed Data Down
<b>100</b>	Confirmed Data Up
<b>101</b>	Confirmed Data Down
<b>110</b>	Reserved for Future Use
<b>111</b>	Proprietary

Figure 11. LoRa Message Type.

## MAC Payload

The MAC payload of the data messages, also-called” data frame”, contains a frame header (FHDR) followed by an optional port field (FPort) and an optional frame payload field (FRMPayload).

**FHDR (Frame header)** The FHDR contains the short device address of the end-device (DevAddr), a frame control octet (FCtrl), a 2 Bytes frame counter (FCnt), and up to 15 Bytes of frame options (FOpts) used to transport MAC commands.

<b>FHDR</b>			
4 Bytes	1 Byte	2 Bytes	0..15 Bytes
DevAddr	FCntrl	FCnt	FOpts

Table 4. Frame Header.

- **FCtrl:** The Frame Control byte. The value of FCtrl changes when issuing an uplink or a downlink message.

<b>FCtrl</b>				
Bit# 7	Bit# 6	Bit# 5	Bit# 4	Bit# 3 to 0
ADR	ADRACKReq	ACK	RFU	FOptLen

Table 5. Control Field Structure for Uplink frames.

<b>FCtrl</b>				
Bit# 7	Bit# 6	Bit# 5	Bit# 4	Bit# 3 to 0
ADR	ADRACKReq	ACK	FPending	FOptLen

Table 6. Control Field Structure for Downlink frames.

- **DevAddr:** consists of 32 bits and identifies the end-device within the current network. DevAddr is allocated by the Network Server of the end-device.
- **ADR and ADDRACKReq:** Two bits that control the Adaptive Data Rate operation, enabling the server to control the flow of data of an end-node by issuing MAC commands. ADR bit being set allows the server to control the data rate. The Acknowledge request bit is set when the new data rate, set by the server, is not functional and no message is received within a specific amount of time. The end-node will keep on dropping the data rate until it could transmit data to the server [5].
- **ACK:** When receiving a confirmed data message, the receiver shall respond with a data frame that has the acknowledgment bit (ACK) set. If the sender is an end-device, the network will send the acknowledgement using one of the receive windows opened by the end-device after the send operation. If the sender is a gateway, the end-device transmits an ACK at its own discretion.
- **FPending:** The frame pending bit is only used in downlink communication, indicating that the gateway has more data pending to be sent and therefore asking the end-device to open another receive window as soon as possible by sending another uplink message.
- **FOptsLen:** The frame-options length field denotes the actual length of the frame options field (FOpts) included in the frame.
- **FCnt:** Each end-device has two “Frame Counters” to keep track of the number of data frames sent uplink to the network server (FCntUp), incremented by the end-

device and received by the end-device downlink from the network server (FCntDown), which is incremented by the network server.

The network server tracks the uplink frame counter and generates the downlink counter for each end-device. After a join accept, the frame counters on the end-device and the frame counters on the network server for that end-device are reset to 0. Subsequently FCntUp and FCntDown are incremented at the sender side by 1 for each data frame sent in the respective direction.

- **FOpts:** The Frame Options part of FHDR transports MAC commands of a maximum length of 15 bytes that are piggybacked onto data frames. If FOptLen is 0, the FOpt field is absent. If FOptLen is different from 0, i.e. if MAC commands are present in the FOpt field, the port 0 cannot be used (FPort must be either not present or different from 0). MAC commands cannot be simultaneously present in the payload field and the frame options field.
- **FPort (Port Field):** If the frame payload field is not empty, the port field must be present. If present, an FPort value of 0 indicates that the FRMPayload contains MAC commands only. FPort values 1 to 223 (0x01..0xDF) are application-specific. FPort values 224 to 255 (0xE0..0xFF) are reserved for future standardized application extensions.
- **FRMPayload (MAC Frame Payload Encryption):** If a data frame carries a payload, FRMPayload must be encrypted before the message integrity code (MIC) is calculated. The encryption scheme used is based on the generic algorithm described in IEEE 802.15.4/2006 Annex B [IEEE802154] using AES with a key length of 128 bits.

### 2-3-6 Encryption and Device Activation

**Security:** LoRaWAN security is designed satisfying the general LoRaWAN design criteria: low power consumption, low cost and high scalability. This technology uses AES (Advanced Encryption Standard) algorithm. to share secret keys and allow the fundamental properties of mutual authentication, integrity protection and confidentiality.

LoRaWAN's security protocol is based on the IEEE 802.15.4 standard. the end-device and the backend receivers share symmetric keys. LoRaWAN specifies a number of security keys, unique per each end-device: NwkSKey, AppSKey and AppKey, all of them with a length of 128 bit.

The **NwkSKey** is a 128-bit AES encryption key, which is unique per device. It provides communication security as it is used to validate the integrity of each message using the message's Message Integrity Code (MIC). The MIC prevents attackers from intentionally tampering with a message.

The **AppSKey** is a 128-bit AES encryption key, which is unique per device. The payload of each message is end-to-end encrypted between the application and the end-device (and vice-versa) using the AppSKey.

**Device Activation:** This mechanism is to control the access from unrecognized end-devices to a LoRaWAN network server and prevent these devices from participating in communications. From the LoRaWAN specification, there are two activation methods for end devices: Activation by Personalization (ABP) and Over-the- Air Activation (OTAA).

- **Over-The-Air Activation (OTAA):** allows a device to be activated by exchanging a series of messages with the network server in order to obtain a Network key over

the air in a secure way. globally unique identifiers to derive the three keys for activation: Device Extended Unique Identifier (DevEUI), Application Extended Unique Identifier (AppEUI), and Application Key (AppKey). In an over-the-air message handshaking process consisting of six steps, the end device is able to obtain DevAddr, NwkSKey and AppSKey. First, the end-device transmits a “Join Request” containing DevEUI, AppEUI, and AppKey. The end-device then receives a “Join Accept” from the network if the credentials are correct. It then authenticates this Join Accept and decrypts it. From the decrypted Join Accept, the DevAddr is extracted and stored. The two security keys – NwkSKey and AppSKey – can then be derived using the AppKey.

- **Activation by Personalization (ABP):** implies that the device comes preconfigured with an address and both the Network and the Application keys, thus being able to directly accessing a predefined network without exchanging data with the network server in join procedures. offers the three required keys directly, by hard coding DevAddr, NwkSKey and AppSKey into the end device’s software. This can be useful for developing as no downlink messages are needed to start sending uplink messages to the network. For production, the use of ABP is discouraged because the device’s activation remains unconfirmed. Also, both session keys stay the same over time, which limits the security of the LoRaWAN transmission.





## Chapter 3.

### The Things Network

The Things Network is about to enable low-power devices to use long-distance Gateways to connect to an open-source, decentralized Network to exchange data with Applications [6]. The Things Network is the network that is responsible for routing the broadcast of LoRaWAN messages by the node over the LoRa radio protocol. Moreover, it works as application server simultaneous.

The things network is very flexible in deployment options. connecting to the public community that hosted by foundation of TTN is one of these options. Furthermore, it is able to deploy private network in a private environment. Hybrid deployments and exchanging date between private and public (vice versa) TTN network are other options of TT deployment.

#### **3-1 Network Architecture**

The Things Network consists of components which are making TTN platform. Each component has a number of duties for processing downlink and uplink messages which sent from end-nodes to applications and vice versa. These components are shown in figure 13.

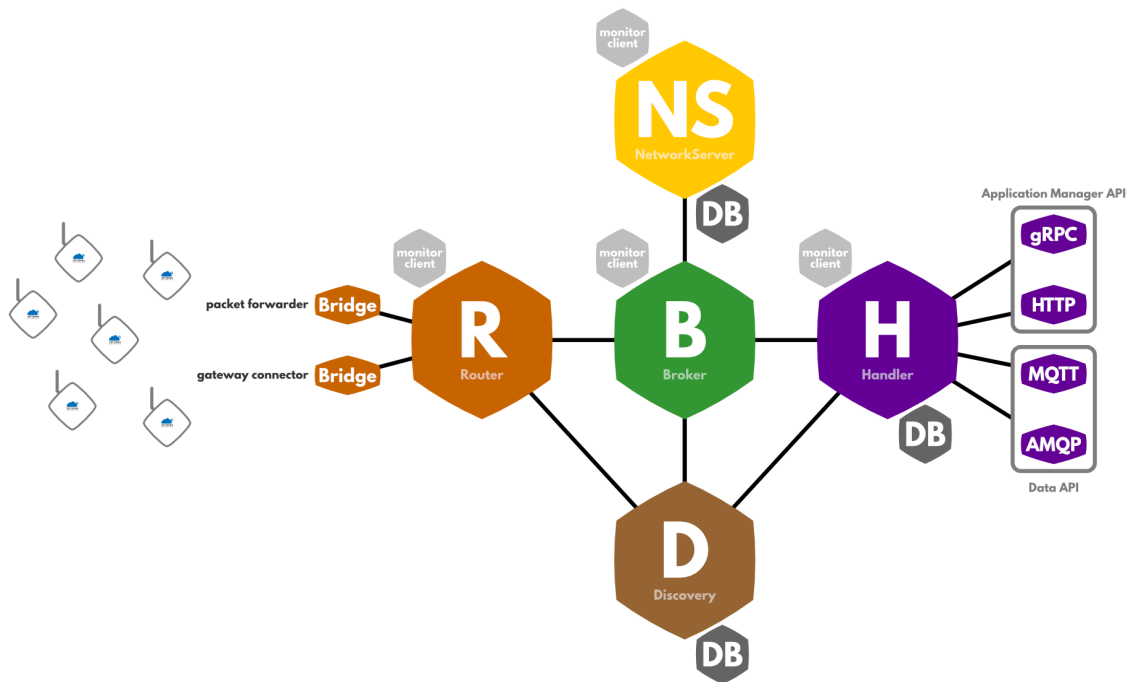


Figure 13. TTN Architecture.

Considering the end-nodes broadcast the messages, the first hardware that received these messages are gateways. Each gateway must connect to one router. The router of TTN architecture is connected to one or more than one broker. the broker is the central part of the TTN network. In addition to the router, the Broker connects to another three components: Network server, Discovery, and Handler. The handling of application data is done by handler and this is the last part of this community.

- **Router/Bridge:** the main responsibility of this is translating the protocol of gateways. When a gateway received the data, it uses internet to transmit messages. The reference and structure of gateway protocol are same but TTN has its own protocol that is more appropriate than other reference. TTN is developed it in order

to security and control access. Since the gateways are from different vendors and protocol so translate protocols to internal protocol used in the backend of TTN.

- **Router:** status of the gateway, uplink message metadata, configuration of downlink and also device address extraction are router duties.
- **Broker:** the broker plays the main role in the TTN. It has a number of responsibilities in the network. De-duplication, looking up the device and application, checking the frame counter, collecting the metadata and selecting the best downlink option are the broker tasks.
- **Network Server:** first of all, uplink messages are sent to the network server and after that forward to handler. The downlink template is added to a message by the network server, handler uses this template to send a downlink message to device.
- **Handler:** handler decodes and converts the message payload and pass it up to application.
- **Discovery Server:** The Discovery Server is the key to the decentralized architecture of The Things Network. This is where routers, brokers and handlers announce themselves and where you can look them up.

### **Processing Flow of Uplink and Downlink Messages:**

First of all, gateway received messages from nodes and transfer it to the router, router translate gateway protocol and after preparing the downlink configuration, it extracts the device address and sends uplink to the broker. Broker guarantees if there will be de-duplicated messages, they will be sent once to the application hence the payload of these duplicated messages are same. since the device address, it is non-unique, the broker

checks the MIC of each device that has the same device address in the network session key case and if not matched, it will drop the message. After validating the MIC, broker Checks the frame counter for avoiding the reply attacks. After these checking steps, broker continues transmitting process. Broker sorts the options of downlink response and choose the best one. Then, the message sends to the network server and it adds Mac commands and downlink template to message then send to handler. Handler decrypts the message then decode the payload by JavaScript function and then convert in format that accessible for applications easily. By sending the message to the application, the uplink transmitting process is finishing. Now application must reply to device with downlink message when it has an available payload, confirmation uplink message or when network server needs to send MAC command to device. The handler takes a message from application and after encryption and conversion of downlink message send to a network server and then broker. broker forwards this message to the router. Router put the schedule message in the buffer then send the message just in time to the gateway and finally, end-nodes will receive downlink message.

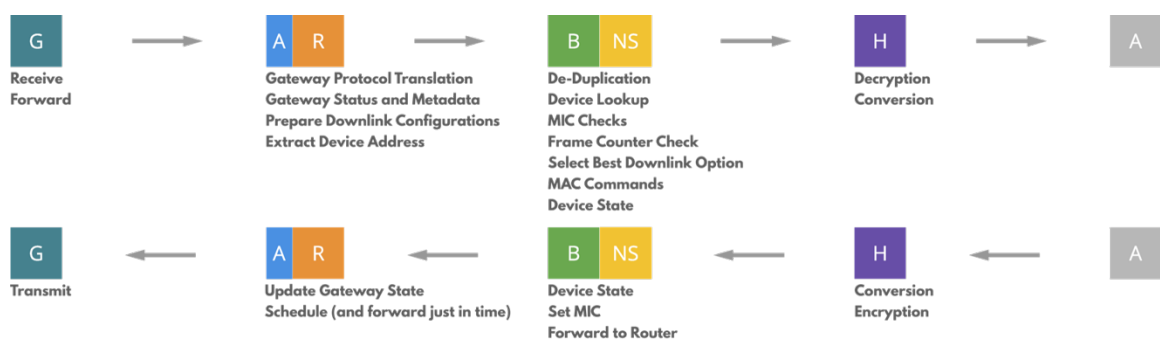


Figure 14. Uplink and Downlink Message Process.

### 3-2 TTN Packet Forwarder

Packet forwarder is a program on LoRa gateway and forwards packets to the server. It communicates with LoRa Chip and network. after receiving packets from Lora chip it transmits them to applications. TTN works with two types of packet forwarders: Semtech UDP Packet Forwarder and TTN Packet Forwarder.

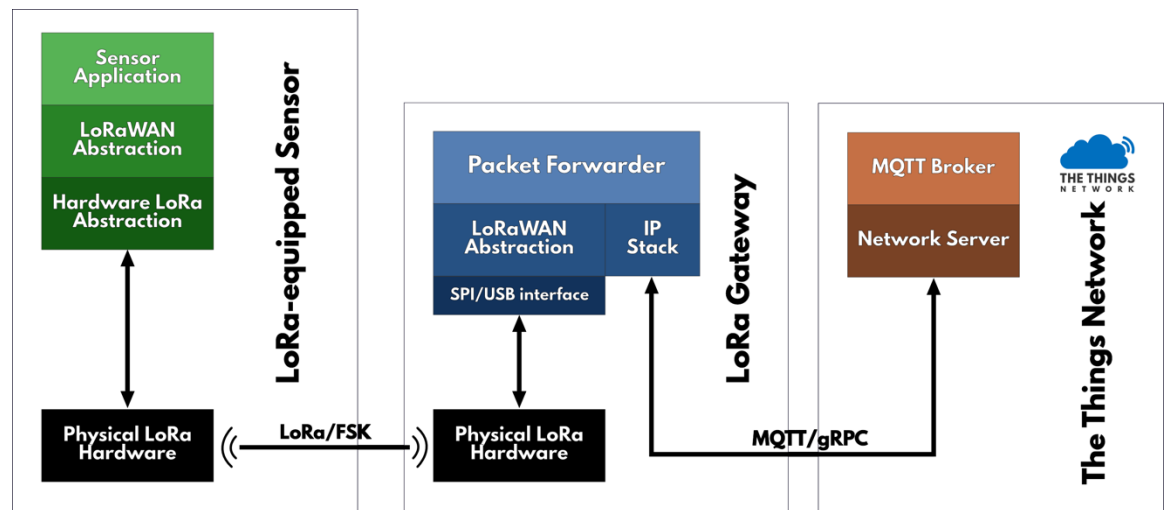


Figure 15. Packet Forwarder.

Semtech UDP Packet Forwarder will be explained in detail in next chapter. but about the TTN packet forwarder, this a new packet forwarder that developed by TTN team. The programming language of this packet forwarder is GO. Its features are as below:

- Built in Golang and open-source
- Connects with the Gateway Connector protocol: authenticfied, reliable and encrypted
- Pre-built for multiple gateways, and build instructions available for all SPI LoRa gateways.

TTN team prepared installation documents for Multitech Conduit, Kerlink IoT Station and Raspberry Pi + iC880a. but if you have different gateways from these, it put installation document to build packet forwarder for your own gateway.

TTN Packet forwarder is configured by a configuration file that is located in `$HOME/.pktfwd.yml`. you should edit this file with your gateway values: gateway ID, gateway KEY and.... If you have a private network, you must set hostname of your private network, discovery server port and also URL of your account server. If you have a specific router, you must put this router ID in the configuration file.

### **3-3 Setting up a Private TTN**

Setting up the components for running the TTN is complex. Because as mentioned before TTN has much more components. For running TTN network server in your local machine, you need the account server [account.thethingsnetwork.org](https://account.thethingsnetwork.org). if you want to use individual account server you can implement your account server with the Account Server API Specification.

There are five steps for installing the TTN network server. It requires Redis database and MQTT broker client server as Mosquitto which will be explained in next chapter with details. After installing both of them, for each component it needs to make a directory and put configuration and authentication information. One directory for Routers, Brokers, Handlers, Network Server, Bridge and Discovery server.

It requires to download two executing program *ttn* and *gateway-connector-bridge and* and by run the exec file of ttn, the network server run if everything will work well.

Routers, Brokers and Handlers authenticate with the discovery server using "access tokens".

The configuration for each component will be stored in `~/ttn/component name/ttn.yml`. note that the discovery-address for Broker, Router and Handler is local address with port 1900. When you configure all components configuration files and do authenticate step, you can start everything together as shown below and Network Server is being run.

```
ttn discovery --config discovery/ttn.yml
ttn router --config router/ttn.yml
ttn networkserver --config networkserver/ttn.yml
ttn broker --config broker/ttn.yml
ttn handler --config handler/ttn.yml
gateway-connector-bridge --root-ca-file "bridge/ca.cert" --ttn-router
"localhost:1900/mynetwork-router"
```

for managing and administering of your network server and application server instead of TTN console, you must configure `ttntctl` and it needs to have a discovery server certificate. Now you can register application and join your device to application with `ttntctl` commands.

## Chapter 4.

### LoRaServer

in terms of actual architecture of LoRaWan networks, gateways work as a bridge between network server and end nodes and network server. This network server has main duty of managing the LoRa networks. it has intelligence information. One of the its tasks is the duplicated packets from gateways are eliminated by network server. it does security checking, also sending acknowledge to the gateways and adjusts data rates. In the end, the network server sends packet to the specific application server if it predestinates for an application server. Hence network server makes to deploy a LoRaWan network very easy. The server that I chose for this thesis is “LoraServer”. It is qualified for doing all of the responsibilities said above.

“LoraServer” has open source components for building LoRaWAN networks. LoRaServer supports LoRaWAN 1.0 and LoRaWAN 1.1 devices, including all regional parameter and bands. All A, B and C End-nodes classes are supported by LoraServer.

#### **4-1 LoRaServer Architecture**

LoRaServer contains multiple components. In addition to LoRaWan devices, gateways and application which are common in other architectures, there are 3 main components and one optional component. Lora Server, LoRa Gateway Bridge, LoRa App Server and LoRa Geo Server (optional).



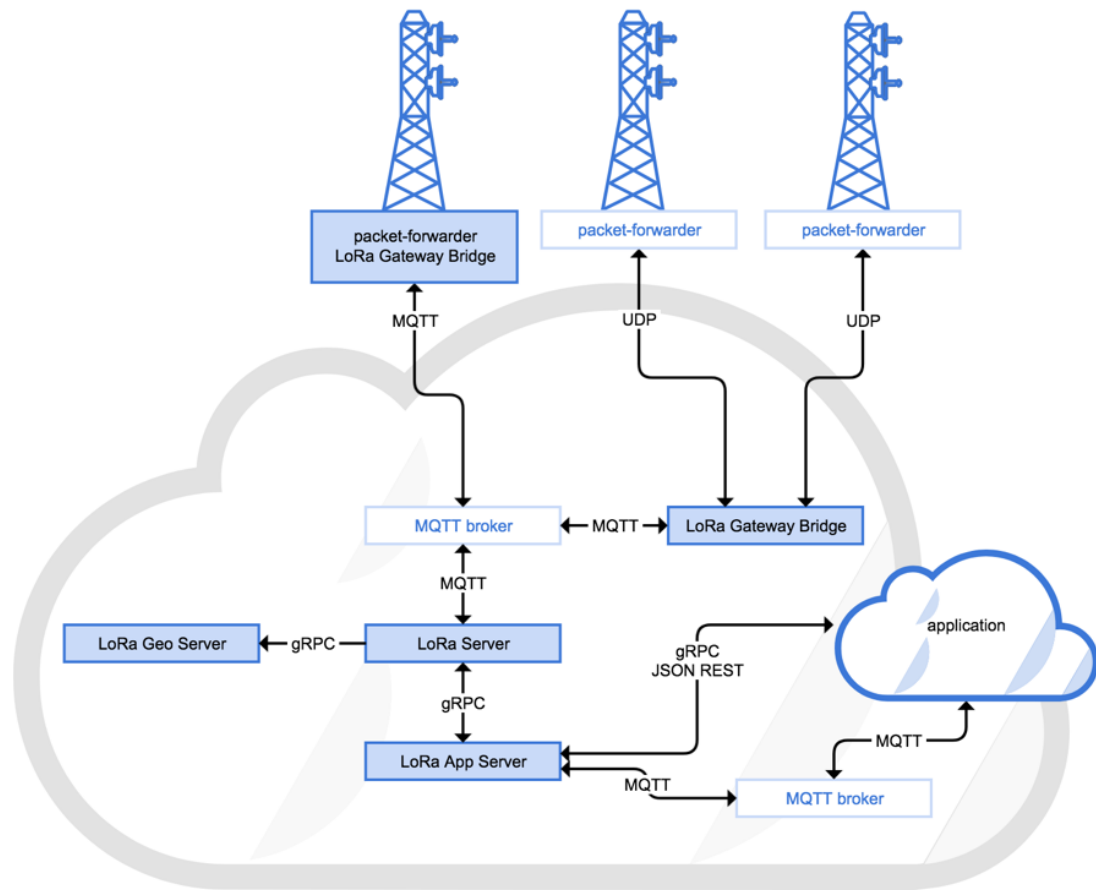


Figure 16. LoRa Server Architecture.

The server is written in Go or golang language that created at Google. its compilation time is very fast and it is very concise, simple and safe.

As can be seen in figure 12, the first component of LoraServer which connects with gateways is the LoRa Gateway Bridge so this component is responsible for the communication with gateways. When gateways send their packets thought UDP protocol, the bridge gateway converts these packet to mqtt, so these data could be understood and manipulated by LoRa Server. The LoRa Server is responsible for managing the state of network. LoRa Server provides an API which can be used when implementing your own

application-server. If we want to provide geolocations services for our devices, we can use LoRa Geo Server.

Lora App Server is the core of LoRaServer architecture. it handles data and stores them in database and communicates with other applications. Two different databases are utilized for storing data which are Redis and PostgreSQL. Redis uses for non-persistence

And session-related data such as errors, logs, cryptography, times of sent and arrival data while persistence data like application key, device addresses and generally end-nodes data are stored in PostgreSQL.

#### 4-1-1 Gateway

One of the hardware capable of acting as a LoRa gateway is SX1301 Semtech that company holds the license to the LoRa technology.

The SX1301 digital baseband chip is a massive digital signal processing engine specifically designed to offer breakthrough gateway capabilities in the ISM bands worldwide. It integrates the LoRa concentrator IP [7]. By using this module, manufacturers are provided to build their customized hardware. The SX1301 is targeted at smart metering fixed networks and Internet of Things applications. The control of the SX1301 by the host system is made using a Hardware Abstraction Layer (HAL). The Hardware Abstraction Layer source code is provided by Semtech and can be adapted by the host system developers [8]. To use this hardware, plugging the chip and combining with other module as a gateway. In this project, the chip is main concentrator that assembled on other module and directly connected to Raspberry pi.

#### 4-1-2 End-Node

In LoRaWAN system the end devices are sensors that are located remotely. the device is shown in figure 17 is used in Lora server project in this thesis and serves as the end-device in the LoRaWAN network infrastructure.

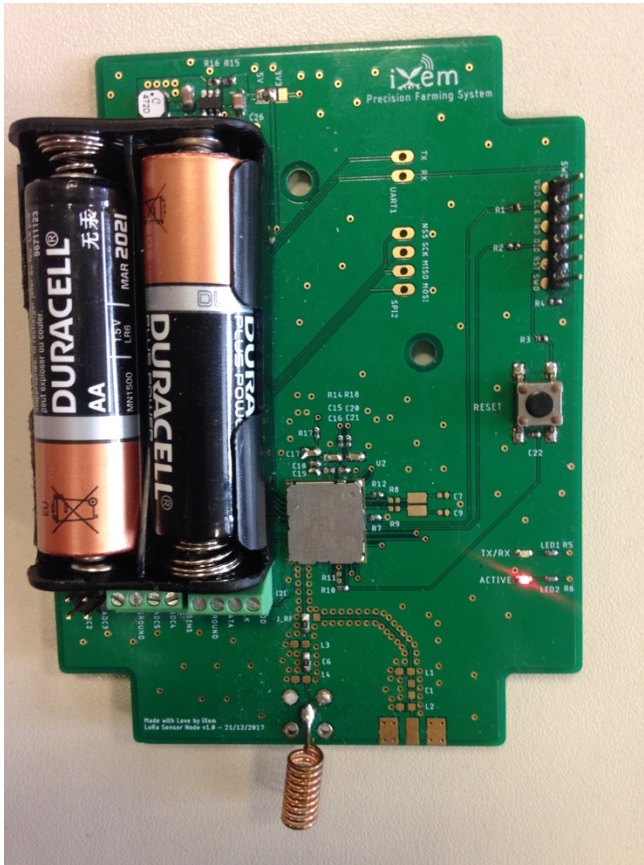


Figure 17. End-node.

#### **4-2 LoRaServer Installation and setup**

The LoRa Server components can be setup in multiple ways which are manual or setup by docker. In this project both of them were used. First of all, manual setup will be explained. Before getting started, there are a couple of requirements such as MQTT broker, Redis and PostgreSQL databases which will be explained.

#### 4-2-1 Requirements

### **MQTT**

MQTT “Message Queuing Telemetry Transport” is a Client Server MQTT publish/subscribe messaging transport protocol.

MQTT is a machine to machine (M2M) internet of things connectivity protocol. MQTT handle messaging between network server and applications. It was designed as lightweight publish/subscribe messaging transport. MQTT is based on TCP/IP. It allows you to send commands to control outputs, read and publish data from sensor nodes and much more.



Figure 18. MQTT Sub/Pub.

### **MQTT Concepts**

there are few basics concepts in MQTT:

- **Publish/subscribe:** publish and subscribe provide an alternative to traditional client-server architecture. a device can publish a message through a topic and also it can subscribe to particular topic to receive messages. For example device1 publish on a topic and device 2 i s subscribed to the same topic as device 1 is publishing in. hence device 2 receive the message.



Figure 19. MQTT Sub/Pub Example.

- **Topics:** MQTT topics are UTF\_8 string consisting of one or more topics level that are separated by / character. Each slash indicates a topic level and creates hierarchy of information for organizing topics. They are the way for registering interesting incoming messages.
- **Messages:** Messages are the information that you want to exchange between your devices. each message has a payload which contains the data to transmit in byte format.
- **Broker:** The publishers and subscribers never contact together directly. Actually, they are not even aware that the other exists. The connection between them is handled by a broker. The broker is responsible for receiving all messages, filtering the messages, decide who is interested in them and then publishing the message to all subscribed clients.

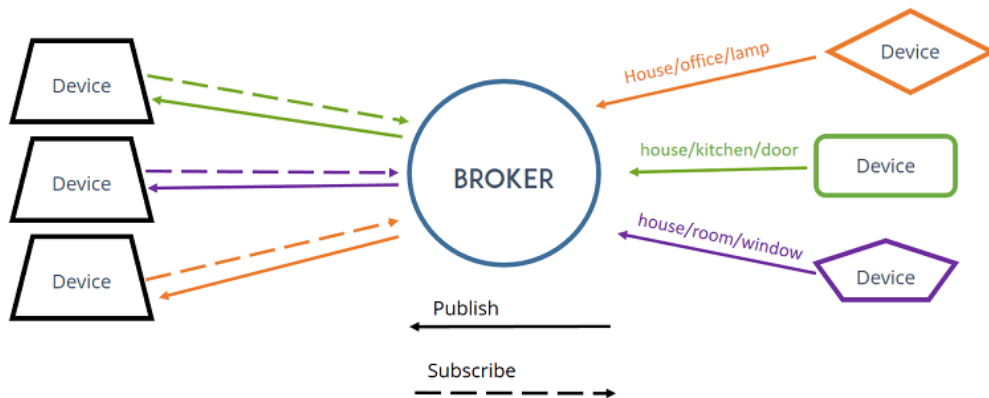


Figure 20. Broker Example

MQTT is used by LoRa Gateway Bridge, LoRa Server, and Lora App Server.

### Mosquitto

Eclipse Mosquitto is a popular open source message broker that implements the MQTT protocol. In this project Mosquitto is used as a MQTT protocol. it is suitable for LoRaWAN messaging such as sensors or mobile devices.

As can be seen in Figure 12, lora server establish a contact between itself and different client by MQTT protocol. MQTT is also used for the connection between LoRa server and LoRa Gateway bridge packet forwarder. Every connection makes lora app server to subscribe to end-nodes channels.

To install Mosquitto on Ubuntu use the command as a below:

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

To test the Mosquitto, execute commands as below. first of all open two terminals, on the first one run this command:

```
mosquitto -v -t "test"
```

And open another terminals and run this command:

```
mosquitto_pub -t "test" -m "This is Test"
```

this sends a simple “this is test” message to the “test” MQTT channels. More complicated commands will be given in the next chapters.

### **PostgreSQL Database**

The LoRa Server components are using PostgreSQL for persistent data-storage. PostgreSQL is a powerful open-source object-relational database management system. the server uses databases to store critical. PostgreSQL is used by LoRa Server and LoRa App Server. PostgreSQL to save all the persistent data and Redis to save all the session-related and non-persistent data. To install PostgreSQL on Ubuntu use:

```
sudo apt-get install postgresql
```

To create a new database, start the PostgreSQL prompt as the `Postgres` user:

```
sudo -u postgres psql
```

## **Redis**

Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker [9]F. As mentioned above, it is need as a database for storing of non-persistence data. This Redis database is used by LoRa Server. Redis stores not only the keys of a node, it stores also the state of the nodes (activated channels, ADR parameters, pending mac-commands, mac-command queues).

To install Redis on Ubuntu use this command:

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

## **Python and Paho-MQTT**

Python is a powerful and object orientated programming language. It is ideal for developing application and scripting on most platforms. The MQTT protocol has been written by python called Paho-MQTT. The source code of this library provides a client to connect to MQTT broker for publishing a message and subscribing to a topic. By Paho-MQTT can write any related MQTT script easily. In this project, the python language was used for writhing script for server to connect end-nodes through MQTT commands.



#### **4-3     setting up the gateway:**

as mentioned earlier, the Raspberry pi was used in this project as a platform for gateway hardware. After attaching the device and setting up this, the gateway software must be installed.

While the “Lora Gateway” is the HAL library for building the gateway based on Semtech Lora SX1301, the “packet forwarder” is a program running on the host of a Lora gateway that forwards RF packets received by the concentrator to a server through a IP/UDP link, and emits RF packets that are sent by the server. It can also emit a network wide GPS synchronous beacon signal used for coordinating all nodes of the network. [10]

“Packet Forwarder” translates data from LoRa frequency and changing format of them to TCP/IP, finally they can be sent via Internet to the server.

Before packet forwarder will be used, we must change two configuration files: The `global_conf.json` and `local_conf.json`. In the global configuration file, scroll down to end of page after antenna characteristics and in gateway configuration part change amount of the "serv\_port\_up" and "serv\_port\_down" from 1680 to 1700. This is the default value that designed for gateway UDP configuration. After that you must change gateway ID of your concentrator, this ID should be 16-digit Hexadecimal. After global configuration changes, we must change local parameters in local configuration file such as gateway id. Now to run the gateway, we can execute packet forwarder easily.

#### **4-4     Installing the LoRa Server project**

this project needs three main configuration files, each of them are responsible for main components of Loraserver. Lora user needs to download and compile these files and configure them for using effectively. So these components will be started as services on

boot on system and can be stopped and started when necessary. Notation that Lora server is contagious and ongoing project which is being updated and changed. It is better to check repository of documentation of lora server page for new and updated information. The first component is lora-gateway-bridge.

#### 4-4-1 installing LoRa-Gateway-Bridge

LoRa Gateway Bridge is a service which abstracts the packet-forwarder UDP protocol into JSON over MQTT. LoRa Gateway Bridge publishes the content of the UDP packets as JSON over MQTT, it becomes trivial to monitor the data that is sent and received by each gateway [11]. by using “apt” command:

```
sudo apt install lora-gateway-bridge
```

the configuration file of lora-gateway-bridge is located at **/etc/lora-gateway-bridge/lora-gateway-bridge.toml**. the default configuration file is sufficient for this service but if you have username and password for MQTT, you must add username and password in configuration file.

Now you can start the Lora-gateway-bridge with this command:

```
sudo systemctl start lora-gateway-bridge
```

if you need start service on boot you can run command as below

```
sudo systemctl start lora-gateway-bridge
```

restart service

```
sudo systemctl restart lora-gateway-bridge
```

stop service

```
sudo systemctl stop lora-gateway-bridge
```

#### 4-4-2 installing LoRa Server

the main components of lora server project is loraserver manage all connections. it is set up after lora-gateway-bridge. Because it needs to connect to physical gateway for working properly. to start installation by using apt install the package:

```
sudo apt install loraserver
```

The configuration file is located at /etc/loraserver/loraserver.toml and must be updated to match the database and band configuration. You must check and update the frequency region, redis and postgres url and authentication of MQTT. After updating the configuration, you need to start loraserver service:

Start LoRaServer service:

```
sudo systemctl start loraserver
```

start on boot:

```
sudo systemctl enable loraserver
```

Restart LoRaServer service:

```
sudo systemctl restart loraserver
```

Stop LoRaServer service:

```
sudo systemctl stop loraserver
```

for printing the lorasever log output use the commands below and when satisfied use Ctrl+C command.

```
sudo journalctl -f -n 100 -u lorasever
```

#### 4-4-3 installing LoRa App Server

LoRa App Server is an open-source LoRaWAN application-server, compatible with LoRa Server. It is responsible for the end-node "inventory" part of a LoRaWAN infrastructure, handling of received application payloads and the downlink application payload queue. It comes with a web-interface and API (RESTful JSON and gRPC) and supports authorization by using JWT tokens. Received payloads are published over MQTT and payloads can be enqueued by using MQTT or the API. [12]

To install this service first run command as below:

```
sudo apt install lora-app-server
```

lora app server configuration file is located at `/etc/lora-app-server/lora-app-server.toml` and it must be updated to database configuration and also postgres authentication.

Start the LoRa App Server service:

```
sudo systemctl start lora-app-server
```

start LoRa App Server on boot:

```
sudo systemctl enable lora-app-server
```

Restart LoRa App Server service:

```
sudo systemctl restart lorasever
```

Stop LoRa App Server service:

```
sudo systemctl stop lorasever
```

for printing the LoRa App Server service log output use the commands below and when satisfied use Ctrl+C command.

```
sudo journalctl -f -n 100 -u lora-app-server
```

after all components are installed, you can access to LoRa App Server web-interface.

To access LoRa App Server web-interface, open the browser and enter the ip address of server by port 8080. For example:

`https://localhost:8080/`

After setting up components and starting services, the gateway must configure so it sends data to Lora-bridge-gateway component and as before mentioned packet forwarder was modified then restarted and lines of logs appearing in Lora-bridge-gateway component logs.

After these steps, gateway must be added to organization of Lora-App-server web interface. In next chapter LoRa-App-Server will be explained completely.

### **App-Server Configurations:**

Open the web browser and enter the server IP address

`https://localhost:8080`

Note that the server is accessible only with https protocol, it may give you a warning about certificate in your browser. This certificate is self-signed and allow your browser to open this web page. Next step, you can login with default username and password which are admin in both case. So it is better to change this password for security reasons.

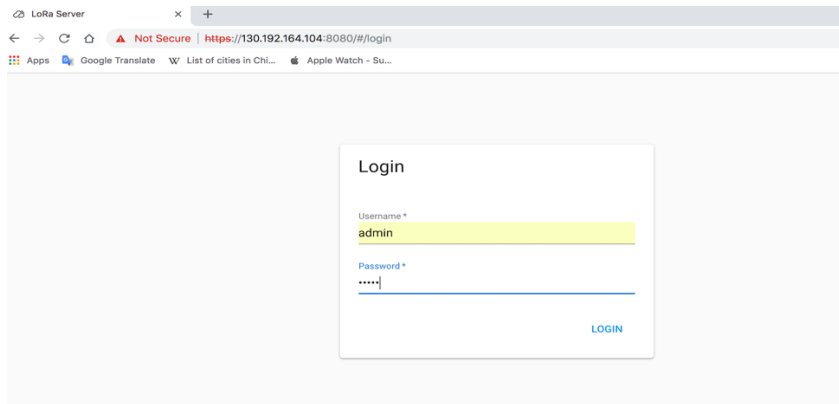
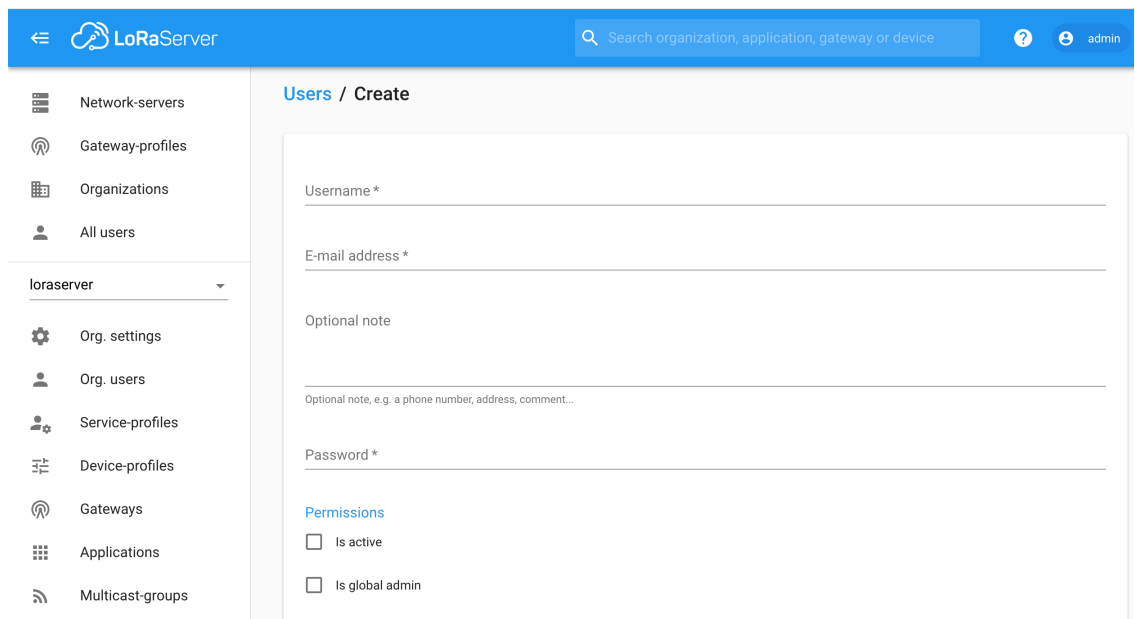


Figure 21. LoRa-App-Server Web Interface

after login in lora-app-server web-interface, you can create and manage users for authentication and authorization reasons. There are two different kind of users, regular users or global admin. Regular user doesn't have any permission but this type of user can be assigned to one or more than one organization.

Global admin user has permission for any action. it can create and organize applications, users, gateways and nodes.



LoRaServer

Search organization, application, gateway or device

?

admin

Network-servers

Gateway-profiles

Organizations

All users

loraserver

Org. settings

Org. users

Service-profiles

Device-profiles

Gateways

Applications

Multicast-groups

Organization users / Create

ASSIGN EXISTING USER

CREATE AND ASSIGN USER

Username \*

Select username

☐ Is organization admin

ADD USER

LoRaServer

Search organization, application, gateway or device

?

admin

Network-servers

Gateway-profiles

Organizations

All users

loraserver

Org. settings

Org. users

Service-profiles

Device-profiles

Gateways

Applications

Multicast-groups

Organization users / Create

ASSIGN EXISTING USER

CREATE AND ASSIGN USER

Username \*

E-mail address \*

Optional note

Optional note, e.g. a phone number, address, comment...

Password \*

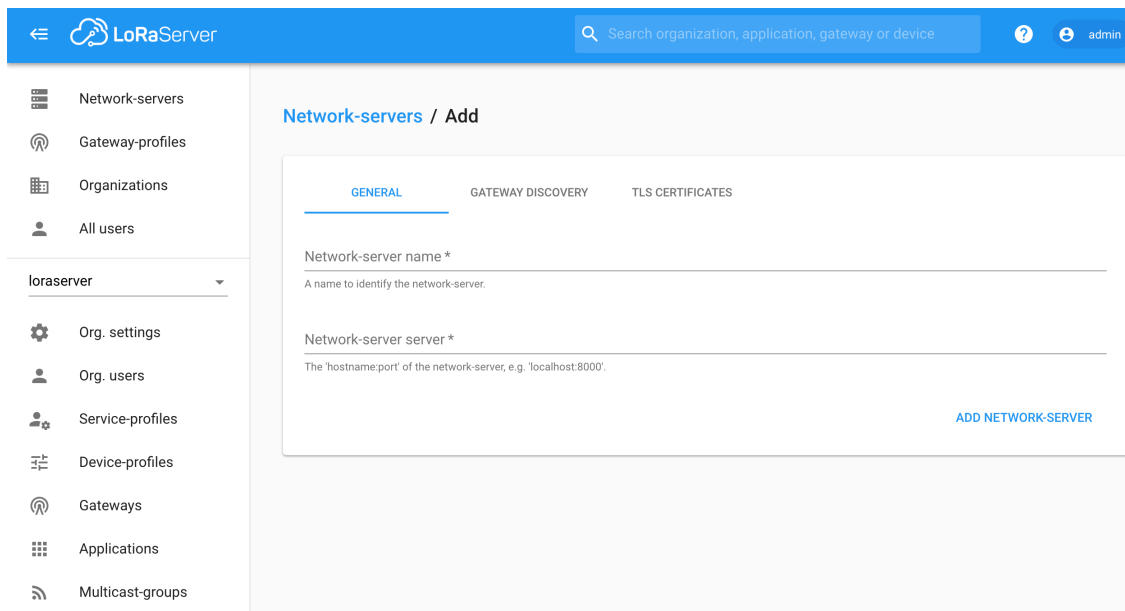
☐ Is organization admin

CREATE USER

## Network-server management:

In lora-app-server, global user can create one or multiple network servers and manage them. Service and device profile are assigned to network server and you can not remove network server before that eliminates those profiles.

You can add network server with three features: General, Gateway Discovery and TLS certificates. In General part, you must put a name for network server and type localhost with default port 8000 in the network server field.



The screenshot displays the LoRaServer web application interface. The top navigation bar is blue with the LoRaServer logo, a search bar, and a user profile icon labeled 'admin'. The left sidebar contains a list of menu items: Network-servers, Gateway-profiles, Organizations, All users, loraserver (selected), Org. settings, Org. users, Service-profiles, Device-profiles, Gateways, Applications, and Multicast-groups. The main content area is titled 'Network-servers / Add' and features three tabs: GENERAL (active), GATEWAY DISCOVERY, and TLS CERTIFICATES. The GENERAL tab contains two text input fields: 'Network-server name \*' with a hint 'A name to identify the network-server.' and 'Network-server server \*' with a hint 'The 'hostname:port' of the network-server, e.g. 'localhost:8000''. An 'ADD NETWORK-SERVER' button is located at the bottom right of the form.

the next option is Gateway discovery. By enabling this feature, each gateway can broadcast a periodical ping that received by gateways its neighborhood.



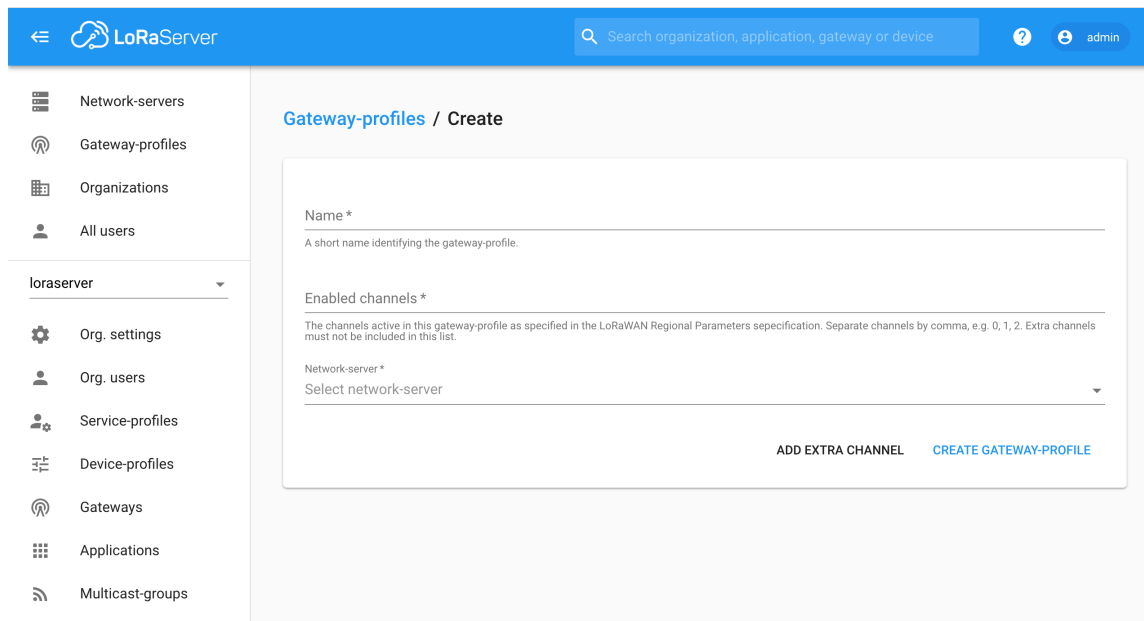


## Gateway-Profiles:

while you are creating a network server, you can make one or more profiles for the gateway.

When you add a gateway to the server, you can select one of these profiles and be sure the gateway configuration is matched with the channel which used by the network-server.

Enabled and Extra channels are two features for gateway profile. You can enter list of channels that you want user for gateway-profiles under Enable channels. In the extra channels you can add extra channels that are not defined in LoRa regional parameter.



The screenshot shows the LoRaServer web interface. The top navigation bar is blue with the LoRaServer logo, a search bar, and a user profile icon labeled 'admin'. The left sidebar contains a list of menu items: Network-servers, Gateway-profiles, Organizations, All users, loraserver (selected), Org. settings, Org. users, Service-profiles, Device-profiles, Gateways, Applications, and Multicast-groups. The main content area is titled 'Gateway-profiles / Create' and contains a form with the following fields:

- Name \***: A text input field with a placeholder 'A short name identifying the gateway-profile.'
- Enabled channels \***: A text input field with a placeholder 'The channels active in this gateway-profile as specified in the LoRaWAN Regional Parameters sepecification. Separate channels by comma, e.g. 0, 1, 2. Extra channels must not be included in this list.'
- Network-server \***: A dropdown menu with the placeholder 'Select network-server'.

At the bottom right of the form, there are two buttons: 'ADD EXTRA CHANNEL' and 'CREATE GATEWAY-PROFILE'.

## Organization management:

Now we can organize our Lora-App-Server under organization, manage menu. This involves service and device profiles, gateway and applications.

Organization managed part give you an ability to create a device profile which will be chosen when you need to add a new device. It defines manufacture information of end-device and also version of LoRaWAN used by end-device. In addition to service profile,

you can create a service-profile that is like a contract between network and users. This contract indicates the features that are enabled for users of service-profile.

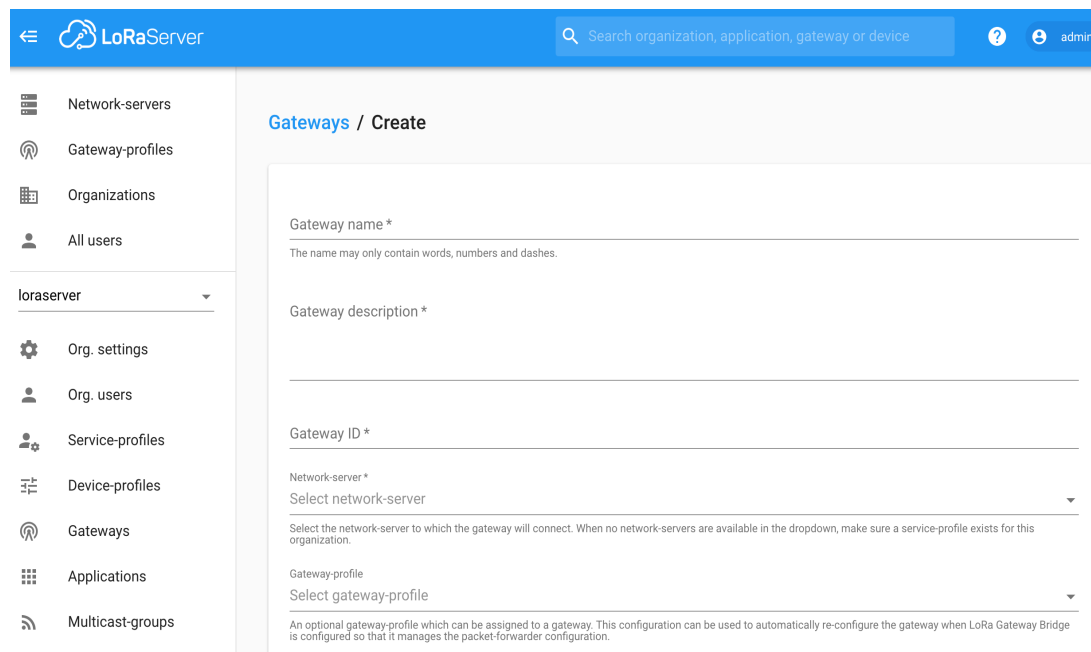
Note that, by activate of add meta data option under service profile, gateway meta data will be added to packet which sent to application server.

The screenshot shows the 'Device-profiles / Create' page in the LoRaServer interface. The left sidebar contains a menu with items: Network-servers, Gateway-profiles, Organizations, All users, loraserver (selected), Org. settings, Org. users, Service-profiles, Device-profiles, Gateways, Applications, and Multicast-groups. The main content area has a blue header with the LoRaServer logo and a search bar. Below the header, there are four tabs: GENERAL (selected), JOIN (OTAA / ABP), CLASS-B, and CLASS-C. The form fields under the GENERAL tab are: Device-profile name \* (with a description: A name to identify the device-profile.), Network-server \* (with a description: Select network-server and a note: 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 \* (with a description: Select LoRaWAN MAC version and a note: The LoRaWAN MAC version supported by the device.), LoRaWAN Regional Parameters revision \* (with a description: Select LoRaWAN Regional Parameters revision and a note: Revision of the Regional Parameters specification supported by the device.), and Max EIRP \* (with a value of 0 and a note: Maximum EIRP supported by the device.).

The screenshot shows the 'Service-profiles / Create' page in the LoRaServer interface. The left sidebar is identical to the previous screenshot. The main content area has a blue header with the LoRaServer logo and a search bar. Below the header, there are four tabs: Service-profiles (selected), JOIN (OTAA / ABP), CLASS-B, and CLASS-C. The form fields under the Service-profiles tab are: Service-profile name \* (with a description: A name to identify the service-profile.), Network-server \* (with a description: Network-server and a note: 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 (checkbox, with a note: GW metadata (RSSI, SNR, GW geoloc., etc.) are added to the packet sent to the application-server.), Enable network geolocation (checkbox, with a note: 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 (with a value of 0 and a note: Frequency to initiate an End-Device status request (request/day). Set to 0 to disable.), and Minimum allowed data-rate \* (with a value of 0).

## Gateways:

As seen before, in the gateway profile part we created a profile with the arbitrary configuration according to our needs. An organization is managing set of the gateways. By using gateway or gateways, each node in our network can communicate together. This menu is able to manage characteristics of gateway such as name, location, ID and.... moreover, live-logs and statistics can be seen under the gateways menu.



The screenshot displays the LoRaServer web application interface. The top navigation bar is blue with the LoRaServer logo, a search bar, and a user profile icon labeled 'admin'. The left sidebar contains a menu with icons and labels for various system components: Network-servers, Gateway-profiles, Organizations, All users, loraserver (selected), Org. settings, Org. users, Service-profiles, Device-profiles, Gateways, Applications, and Multicast-groups. The main content area is titled 'Gateways / Create' and contains a form with the following fields:

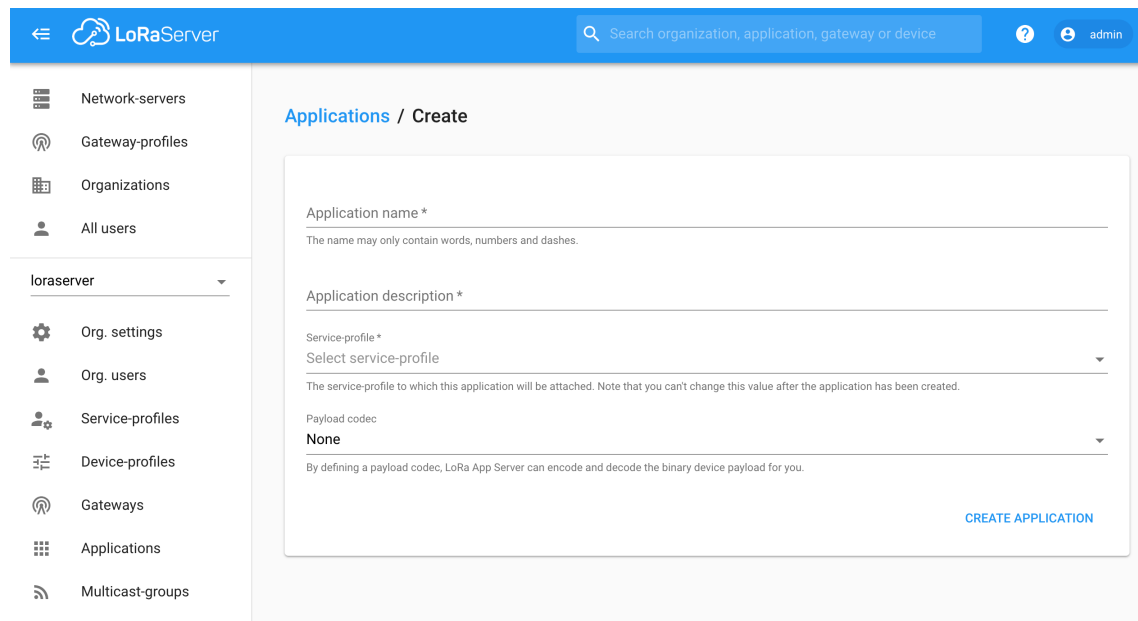
- Gateway name \***: A text input field with a note below it stating 'The name may only contain words, numbers and dashes.'
- Gateway description \***: A text input field.
- Gateway ID \***: A text input field.
- Network-server \***: A dropdown menu with the option 'Select network-server'.
- Gateway-profile**: A dropdown menu with the option 'Select gateway-profile'.

Below the 'Gateway-profile' dropdown, there is a small note: 'An optional gateway-profile which can be assigned to a gateway. This configuration can be used to automatically re-configure the gateway when LoRa Gateway Bridge is configured so that it manages the packet-forwarder configuration.'

here, you choose a name for gateway and Gateway ID is the assigned gateway that you set on the configuration file of your packet forwarder. By assigning Gateways-profile, an updated configuration is sent to the gateway by LoRa-Server. Hence configuration of packet-forwarder, lora-gateway-bridge, and gateways setting in lora-app-server must be harmony and updated.

## Application management:

The collection of devices with same aim is called application in lora-app-server. When an application is created, we can choose service-profile that was explained in previous part. This will be used for devices which will be created under applications menu.



The screenshot displays the LoRaServer web interface. The top navigation bar is blue with the LoRaServer logo, a search bar, and a user profile icon labeled 'admin'. The left sidebar contains a menu with icons and labels: Network-servers, Gateway-profiles, Organizations, All users, loraserver (selected), Org. settings, Org. users, Service-profiles, Device-profiles, Gateways, Applications, and Multicast-groups. The main content area is titled 'Applications / Create' and contains a form with the following fields: 'Application name \*' with a note 'The name may only contain words, numbers and dashes.', 'Application description \*', 'Service-profile \*' with a dropdown menu showing 'Select service-profile', and 'Payload codec' with a dropdown menu showing 'None'. A note below the payload codec field states: 'By defining a payload codec, LoRa App Server can encode and decode the binary device payload for you.' A blue 'CREATE APPLICATION' button is located at the bottom right of the form.

if you specify a payload codec for your application, binary device payload can be encoded and decoded by Lora-APP-Server. Here you can select two type of codec: cayenne LPP or custom java function [13]. the raw base64 encoded payload will always be available, even when a codec has been configured. In this project cayenne LLP was used for device payload so Lora-App-Server decoded and encoded base on cayenne low power payload.

The Cayenne Low Power Payload (LPP) provides a convenient and easy way to send data over LPWAN networks such as LoRaWAN. The Cayenne LPP is compliant with the payload size restriction, which can be lowered down to 11 bytes, and allows the device to send multiple sensor data at one time [14].

Additionally, the Cayenne LPP allows the device to send different sensor data in different frames. In order to do that, each sensor data must be prefixed with two bytes:

**Data Channel:** Uniquely identifies each sensor in the device across frames.

**Data Type:** Identifies the data type in the frame.

If you would like to write your own javascript for decoding and encoding of array of bytes to a java subject, you can use custom Custom JavaScript codec functions.

## Device management:

The server supports both type of over-the-air activation (OTAA) and activation by personalization (ABP) devices. By selecting the device-profile we can configure the new device. Also, you can add one of three classes of device: A, B, or C. In this thesis, our end-nodes are from class A that was described in **Chapter 2**.

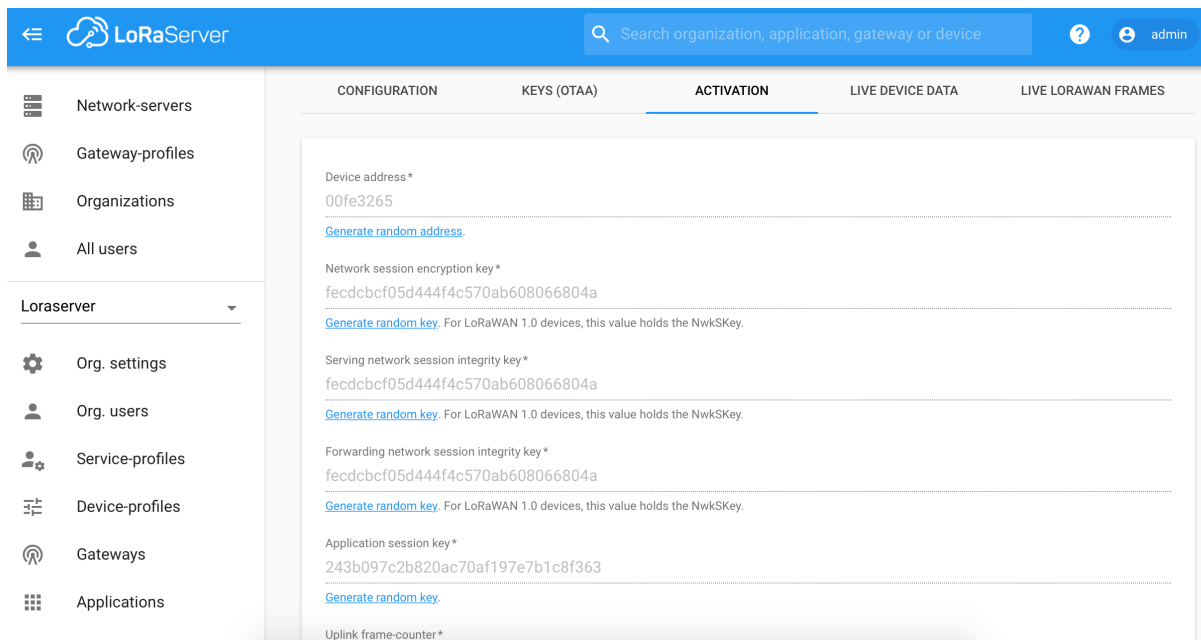
The screenshot displays the LoRaServer web interface. The top navigation bar is blue with the LoRaServer logo, a search bar, and user information. A left sidebar lists various management options. The main content area shows the 'Device-profiles / Create' form with four tabs: GENERAL, JOIN (OTAA / ABP), CLASS-B, and CLASS-C. The GENERAL tab is active, showing fields for Device-profile name, Network-server, LoRaWAN MAC version, LoRaWAN Regional Parameters revision, and Max EIRP.

GENERAL	JOIN (OTAA / ABP)	CLASS-B	CLASS-C
<b>Device-profile name *</b> A name to identify the device-profile.			
<b>Network-server *</b> Select network-server The network-server on which this device-profile will be provisioned. After creating the device-profile, this value can't be changed.			
<b>LoRaWAN MAC version *</b> Select LoRaWAN MAC version The LoRaWAN MAC version supported by the device.			
<b>LoRaWAN Regional Parameters revision *</b> Select LoRaWAN Regional Parameters revision Revision of the Regional Parameters specification supported by the device.			
<b>Max EIRP *</b> 0 Maximum EIRP supported by the device.			

[CREATE DEVICE PROFILE](#)

We configure the LoRaWAN Network by Lora server method. We have the gateway, end-node and network server for managing everything. Now we can view the information and only frame logs of device that is related to added end-node in device-frame logs.

In the join Tab there is an option to use the OTAA type of device, when this option is used, device name and device EUI are need to fil. The value of device EUI must be 16 hexadecimal characters, and device name is the standard name includes the characters. After create the device in next tab you must enter App EUI is 16 hexadecimal characters. After generating these value, you must register your device with these values by mac command. After join the device with this values, it is activated in Lora server. Network Session key and Application Session key are generated automatically.



The screenshot displays the LoRaServer web interface. The top navigation bar is blue with the LoRaServer logo, a search bar, and a user profile icon labeled 'admin'. The left sidebar contains a menu with icons and labels for 'Network-servers', 'Gateway-profiles', 'Organizations', 'All users', and a dropdown for 'Loraserver'. The main content area has tabs for 'CONFIGURATION', 'KEYS (OTAA)', 'ACTIVATION' (which is selected), 'LIVE DEVICE DATA', and 'LIVE LORAWAN FRAMES'. Under the 'ACTIVATION' tab, there are several input fields with generated values and 'Generate random' links: 'Device address \*' (00fe3265), 'Network session encryption key \*' (fecdcbcf05d444f4c570ab608066804a), 'Serving network session integrity key \*' (fecdcbcf05d444f4c570ab608066804a), 'Forwarding network session integrity key \*' (fecdcbcf05d444f4c570ab608066804a), 'Application session key \*' (243b097c2b820ac70af197e7b1c8f363), and 'Uplink frame-counter \*'.

Otherwise you can choose the Class B or C.

You observe the device status and live frame logs after sending the uplink.

The screenshot shows the LoRaServer web interface. The left sidebar contains navigation links: Network-servers, Gateway-profiles, Organizations, All users, Loraserver (selected), Org. settings, Org. users, Service-profiles, Device-profiles, Gateways, and Applications. The main content area is titled 'Applications / Lora-app / Devices / dev2' and includes a 'DELETE' button. Below the title are tabs for CONFIGURATION, KEYS (OTAA), ACTIVATION, LIVE DEVICE DATA (selected), and LIVE LORAWAN FRAMES. Under the 'LIVE DEVICE DATA' tab, there are buttons for HELP, PAUSE, DOWNLOAD, and CLEAR. A table displays two log entries:

Time	Event	Action
5:27:38 PM	uplink	▼
5:27:38 PM	join	▼

The screenshot shows the LoRaServer web interface with the 'LIVE LORAWAN FRAMES' tab selected. It displays a detailed log entry for a downlink frame. The log entry is titled 'DOWNLINK 5:36:42 PM UnconfirmedDataDown 005f88b4'. The log content is as follows:

```
▼ txInfo: {} 10 keys
  gatewayId: "b827ebfffebecf3"
  immediately: false
  timeSinceGpsEpoch: null
  timestamp: 2946006188
  frequency: 868100000
  power: 14
  modulation: "LORA"
  ▼ loraModulationInfo: {} 4 keys
    bandwidth: 125
    spreadingFactor: 12
    codeRate: "4/5"
    polarizationInversion: true
  board: 0
  antenna: 0
  ▼ phyPayload: {} 3 keys
    mhdr: {} 2 keys
      mType: "UnconfirmedDataDown"
      major: "LoRaWANR1"
    macPayload: {} 3 keys
      fhdr: {} 4 keys
        devAddr: "005f88b4"
        ▼ fCtrl: {} 5 keys
          adr: true
          adrAckReq: false
          ack: false
          fPending: false
          classB: false
          fCnt: 0
        ▼ fOpts: [] 1 item
          0: {} 1 key
            bytes: "A1EHAAE="
        fPort: null
        frmPayload: null
        mic: "72fa1bd9"
```



## Chapter 4.

### The Architecture of Gateway

By implementing the two methods above, LoRaServer and TTN as a network server for LoRaWAN system, we conclude that the connection between network server and gateway is a very critical issue. Because if it will be lost, the uplink and downlink messages will be missed. This connection causes a lot of power consumption, since in many scenarios we are forced to use cellular connection. Also it should be always active and not turn-off even for few minutes. So power consuming is a big problem when network server and gateway must connect together by power line.

Since in this thesis, the main goal is to build a low-power gateway in order to receive data from sensors and send them to Application anytime and anywhere, through LoRaWAN network, it is needed to change the architecture of the system, using an offline gateway, that activate the cellular connection only to push data on a remote server periodically.

Since the gateway will not do process a lot, the alternative architecture for the gateway is shown in Figure 4. By integrating network server and gateway together we can solve the power consumption problem of two previous methods.

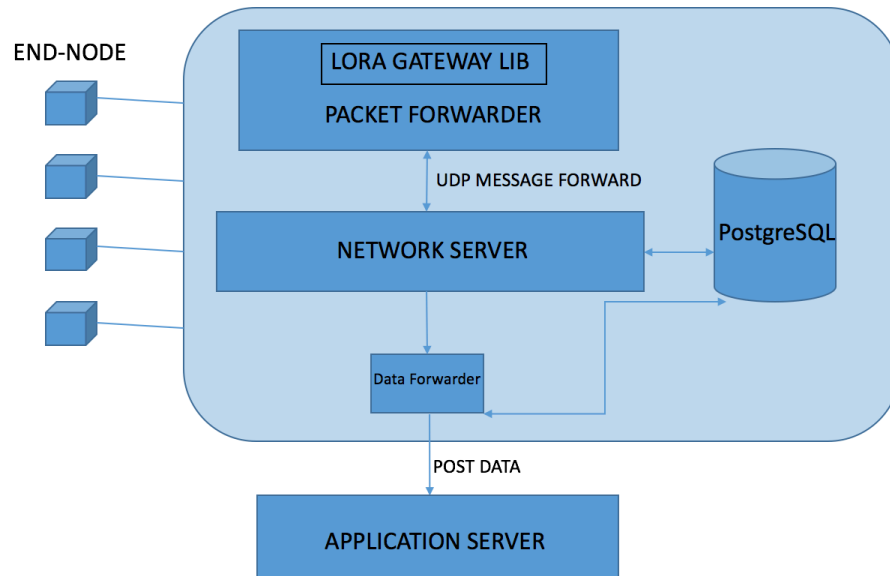


Figure 22. Low-Power Gateway Architecture.

## 4-1 LoRa Gateway Architecture

### 4-1-1 Hardware

The main hardware of this system includes Raspberry pi that is a single-board computer (SBC) used to do small computing and networking operations, it has a GSM module on its shield for Internet connectivity. The radio front-end is provided by concentrator board is iC880A.

But powering of Raspberry is one of the main challenges, adding a solar panel as energy harvester is mandatory to charge the battery and allow the system to work without a power line.

### 4-2-2 Software

The structure of the network server is same to TTN and LoRaServer but the location of packet forwarder had been only changed. Packet forwarder is a software running on the

gateway to encapsulate data and to send it via internet. Fortunately, it didn't need to write a software gateway from the scratch Since in this project Semtech packet forwarder was used. Two utilities provided on the lora\_gateway repository:

- Lib\_lora\_gateway: driver library source code which has been adapted to use the Raspberry Pi hardware.
- packet\_forwarder: Gateway application source code

There are many ways for forwarding data to the IoT platform and vice versa. in this case, MQTT is a machine-to-machine (M2M)/"Internet of Things" connectivity protocol. that make use of the publish and subscribe pattern. PostgreSQL was used in this system for the long-term storage database.

Data Forwarder is a python application that receive data from the MQTT broker and store them into the PostgreSQL database, this application will periodically power on the cellular connection to post the data to an application server.

Now LoRaWAN gateway is implemented by placing the network server and packet forwarder on Raspberry directly, we designed low-power gateway that not need to power line for creating the connection between networks server and gateway as two previous methods.

## Chapter 5.

### Experiments on the Network and Programming

Consider LoRa Server Network Server and run it on the Docker-Compose when logs of components were seen successfully, open web-page interface and login to the LoRaServer console. Concurrently run packet forwarder on the gateway. Now gateway status was shown in the console. Join the device to the application and turn on the sensor, sensor status as seen option was shown in console. When the connection between gateway, network server and end-node was connected, open the terminal and use mosquito command to topic and print the message that is receive. See the Figure 23.

```
behnaz@ubuntu:~$ mosquitto_sub -t "application/3/device/5b87d6a68aee6114/rx"
{"applicationID":3,"applicationName":"Lora-app","deviceName":"dev2","devEUI":"5b87d6a68aee6114","rxInfo":{"gatewayID":"b827ebfffebecef3","name":"my-gateway","rssi":-34,"loRaSNR":9.5,"location":{"latitude":45.0672066,"longitude":7.678099999999999,"altitude":0}},{"txInfo":{"frequency":868500000,"dr":0,"adr":true,"fCnt":0,"fPort":99,"data":"AAAnAQIAAA==","object":{"digitalInput":{"0":39},"analogInput":{"1":0}}}}
{"applicationID":3,"applicationName":"Lora-app","deviceName":"dev2","devEUI":"5b87d6a68aee6114","rxInfo":{"gatewayID":"b827ebfffebecef3","name":"my-gateway","rssi":-37,"loRaSNR":10.5,"location":{"latitude":45.0672066,"longitude":7.678099999999999,"altitude":0}},{"txInfo":{"frequency":868100000,"dr":0,"adr":true,"fCnt":0,"fPort":99,"data":"AAAnAQIAAA==","object":{"digitalInput":{"0":39},"analogInput":{"1":0}}}}
```

Figure 23. Mosquitto Subscribe topic.

The received data is in Base64 format and needs to be decoded (data is already decrypted by the server itself) first to be understood.

## 5-1 A Python script to automatically send data to the end-nodes

```
Connecting to the server...
database connected
Connected with result code 0
Now the payload is printed
{'applicationName': u'Lora-app', u'deviceName': u'dev2', u'adr': True, u'txInfo': {u'frequency': 868500000, u'dr': 0}, u'fCnt': 0, u'data': u'AAAKAQIAAA==', u'fPort': 99, u'object': {u'digitalInput': {u'0': 36}, u'analogInput': {u'1': 0}}, u'rxInfo': [{u'gatewayID': u'b827ebffffe3', u'rssi': -45, u'name': u'my-gateway', u'loRaSNR': 6, u'location': {u'latitude': 45.0672066, u'altitude': 0, u'longitude': 7.678099999999999}], u'devEUI': u'5b87d6a68aee6114', u'applicationID': u'3'}
```

Figure 24. Python script at work.

The code script facilitates the subscription to different end-nodes and saves application id to the configuration “.ini” file. This way, the problem of constant subscription and feeding the same data every time the program is launched, is resolved. The script uses MQTT broker package as paho-mqtt to program easy and client can import MQTT commands inside the script directly. First purpose of this code is to send data to each node any time that a message from these devices. It understood the MQTT message and responded. So when the uplink message was sent, it responded when device has two open receive windows. When the connection was made successfully, script works as a MQTT subscriber and show the information which usually is the Base64-encoded data of the server and store this data to the PostgreSQL database that uses from other useful python package.

After importing required library, first step is to check existence of configuration file, with command `os.path.isfile` of the OS library check a file name. if it exists, it uses the data on the configuration file and if it doesn't exist, it creates the configuration file with a command like `cfgfile = open()` and adding the “w” letter for write-enabled.

After this step, the script starts using the paho.mqtt package and subscribes to each end-node's MQTT channel respectively. `on_connect` function is responsible for subscribing, so this way if the connection to the server is interrupted, when they reconnect

the subscriptions occur again, not allowing the interruption to fail the whole script. Next function is `on_message` is activated when we have an uplink. If needed the information of message function like port number, date/time, SNR, RSSI and frame counter, it uses the `print()` command by `json.loads`. the result of print store in database. using command `psycopg2.connect` with server ip address and port connect to database.

`Client.connect()` is the most important command in MQTT library. using the server's address and port number. The third number is a "keep alive" which by default is 60 seconds and overviews the program-server connection and sends pings every 60 seconds of inactivity to keep the connection alive. The last part of code is **`loop_forever()`** which puts the whole connection inside a loop so that it will never disconnect. To exit the script, the **Ctrl+C** is used.

The code for the script is available in Appendix 2.

## Chapter 6.

### Conclusion

Presently LoRaWAN is not known very much if compared to IoT standards like Bluetooth or Zigbee, all around the world but the free projects such as TTN and LoRaServer are existed for involving to show there are no bound and limitation for LoRaWAN path.

In conclusion, this project shows how anyone can build their own gateway and start their own project without the need for additional authority. Thanks to packet forwarder gateway is able to connect to network server and doing transmitting process.

As seen in this thesis, the network server is needed to be on whole time. In conclusion, the amount of energy consumed for keeping the alive connection between the gateway and network server is much more.

#### **5-1 Technical Conclusion**

If consuming power has not been considered and the first method for connection between network server and gateway will be chosen, in the comparison TTN and LoRaServer, the author's opinion is the LoRaServer is a best solution. Since the LoraServer has a practical console and it is more visibility of what is going over the network, you can have your own gateway. This would guarantee that you have a connection and it would allow you to see what is actually transmitted through the gateway. This would speed up testing and developing a lot.

If it is preferred to keep the network server outside of the gateway, an alternative path would be to modify the packet forwarder in order to locally store the data inside it, and asynchronously forward the UDP datagram. But this would end up breaking the direct connection between end nodes and application server, which is fundamental for many protocol features like the adaptive data rate.



## Appendix 1.

### Setting Up LoRaServer Using the Docker

First of all, you need to install Docker. Docker is a implement designed to make it easier to create, deploy, and run applications by using containers. Containers let a developer to pack an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. in this project each requirements and components for setting up the LoRServer is a container. Then you must install Docker Compose. Docker Compose makes possible to organize the configuration file of multiple Docker conditioner; this configuration file is a docker-compose.yml file.

#### Installing Docker and Docker-compose

The LoRa Server project provides an example docker-compose.yml file that you can use as a starting-point.

Clone this repository with

```
git clone https://github.com/brocaar/loraserver-docker.git
```

```
cd loraserver-docker
```

after updating the configuration file, you can start LoRaServer with command:

```
docker-compose up
```

## Appendix 2.

### A Code to subscribe to MQTT and send data to the end-nodes

```
1. import paho.mqtt.client as mqtt #used for connecting to mqtt broker
2. import json
3. import psycopg2 #used for connecting to postgresql
4. import configparser #for creating config file
5. import requests #used for post url
6. import os #used for check path file
7.
8.
9. def on_connect(client, userdata, flags, rc): #connect to mqtt broker
10.
11.     print("Connected with result code "+str(rc))
12.     client.subscribe("application/"+appID+"/device+/rx") #subscribe the topic b
y using applicationID
13.     print("Now the payload is printed")
14.
15.
16. def on_message(client, userdate, msg): #function for show the payloads as resul
t of subscribe
17.
18.     data = json.loads(msg.payload) #convert string to json
19.     print(data)
20.     #for entry in data['object']:
21.     cur = conn.cursor()
22.     for entry in data['rxInfo']:
23.         cur.execute("INSERT INTO rxinfo (\"devicename\", \"gatewayid\", \"rssi\
") VALUES (%s, %s, %s)", (data['deviceName'], entry['gatewayID'], entry['rssi'])
)
24.     #add specific data in the database
25.     conn.commit()
26.     for obj in data['object'].iteritems():
27.         for channel, value in obj[1].iteritems():
28.             cur.execute("INSERT INTO data (\"devicename\", \"channel\", \"data
\") VALUES (%s, %s, %s)", (data['deviceName'], channel, value))
29.     conn.commit()
30.
31.     #url = "https://api.ixem.wine/post"
32.     #r = requests.post(url, data)
33.     #print (r.text)
34.
35. config = configparser.ConfigParser(allow_no_value=True)
36. if not os.path.isfile("config.ini"):
37.     appID = raw_input("Please enter the applicaionID of your node: ")
38.     url = raw_input("Please enter your url: ")
39.     cfgfile = open("config.ini", "w")
40.     config.add_section("Defaults")
41.     config.set("Defaults","appid", appID)
42.     config.set("Defaults","url", url)
43.     config.write(cfgfile)
44.     cfgfile.close()
45.     print("configuration file created")
46. #else
47. #read the cofiguration file to put the value inside appid url...
48. else:
```

```

49.     config.read("./config.ini")
50.     appID = config["Defaults"]["appid"]
51.     url = config["Defaults"]["url"]
52. try:
53.     conn = psycopg2.connect(host="localhost", database="mqtt", user="postgres", pas
        sword="postgres", port="5432") #connect to database
54. except:
55.     print("unable to connect to the database.")
56. else:
57.     print("database connected")
58.     client = mqtt.Client()
59.     client.on_connect = on_connect
60.
61. client.on_message = on_message
62. broker_address="localhost"
63. client.connect(broker_address, 1883, 60)
64. client.loop_forever()

```

## Bibliography

- [1] K. Ashton, "That 'Internet of Things'," 22 June 2009. [Online]. Available: <http://www.rfidjournal.com/articles/view?4986>.
- [2] LoRa® Alliance Technical Marketing Workgroup, "LoRaWAN What is it?" November 2015.
- [3] E. Ruano, "LoRaTM protocol Evaluations, limitations and practical test" 11 May 2016.
- [4] M. Centenaro, L. Vangelista, A. Zanella, and M. Zorzi, "Long-Range Communications in Unlicensed Bands: the Rising Stars in the IoT and Smart City Scenarios, IEEE Wireless Communications, Vol. 23, Oct. 2016. rXiv:1510.00620v2
- [5] N. Sornin, M. Luis, T. Eirich, T. Kramp, and O. Hersent. Lorawan specification. Technical report, LoRa Alliance, 2015.
- [6] "The Things Network" , [Online]. Available: <https://www.thethingsnetwork.org/>. Accessed: 2018-11-30.
- [7] "SX1301 Datasheet" , Semtech Corporation, May 2017.
- [8] "WiMOD iC880A Datasheet," 4100/40140/0074, IMST GmbH, Mar. 2015.
- [9] "Redis", [Online]. Available: <https://redis.io/>. Accessed: 2018-11-30.
- [10] Lora-net "packet\_forwarder, " [Online]. Available: [https://github.com/Lorant/packet\\_forwarder](https://github.com/Lorant/packet_forwarder)
- [11] O. Brocaar, " LoRa Gateway Bridge", [Online]. Available: <https://www.loraserver.io/lora-gateway-bridge/overview/>
- [12] O. Brocaar, "LoRa Server Documentation," [Online]. Available: <https://docs.loraserver.io/loraserver/>.
- [13] O. Brocaar, " Application management," [Online]. Available: <https://www.loraserver.io/lora-app-server/use/applications/> Accessed: 2018-11-30.
- [14] M. Diez, "Secure Position Data Transmission for Object Tracking using LoRaWAN" , 24 August, 2017