LoRaWAN: a preliminary implementation and evaluation

Alessandro Zini, Filippo Morselli, Carlo Stomeo
Dipartimento di Informatica - Scienza e Ingegneria, University of Bologna, Italy
Emails: alessandro.zini3@studio.unibo.it, filippo.morselli@studio.unibo.it, carlo.stomeo@studio.unibo.it

***Abstract*—This report describes the final project for the Internet of Things course. The aim of the project is to implement a single-channel packet-forwarder LoRaWAN gateway.**

In order to build the gateway, we equipped a Raspberry Pi 3 B+ with a LoRa module; for testing and performance comparison purposes we used two different modules, one from Futura Elettronica and another one from Dragino.

The network and application server used by our gateway are both provided by The Things Network, to which we also connected an MQTT client application which receives and stores the data in a time-series database and plots them for further analysis.

The results obtained with the two LoRa modules are equivalent, and they more or less matched the performance capabilities claimed by the LoRa technology.

The main purpose of this work is to put in practice the cutting-edge technologies seen during the IoT course, and also to contribute to the LoRaWAN network diffusion in the Bologna Area.

I. Introduction

LoRa is a digital wireless communication technology for Internet of Things applications. It enables long range, low power and inexpensive transmissions, which are all essential requirements in an IoT scenario.

The technology is divided in two parts: **LoRa**, which defines the physical layer, and **LoRaWAN**. LoRaWAN is a standard defining the communication protocol and system architecture for the network; it operates at the data link (MAC) and network layer of the ISO/OSI stack.

The purpose of this project is to setup a working LoRaWAN network using low cost hardware components, as well as testing the performances that can be achieved by this kind of infrastructure in a urban scenario.

During the development of the project we went through the following steps:

- enable a point-to-point LoRa link communication between two Raspberry Pis equipped with LoRa modules;
- create a working single-channel LoRaWAN gateway (achieved by using a Raspberry Pi running a packet-forwarder script);
- setup The Things Network as both network and application server and interface the gateway to its services;
- test the behavior of the gateway using two different LoRaWAN end nodes (an Arduino and a Raspberry Pi);
- create a client application to process the data sent by the end nodes;

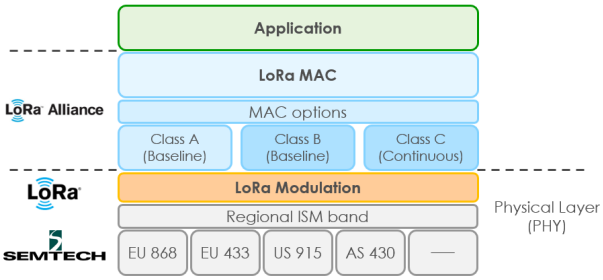

Fig. 1: LoRa stack.

- evaluate the performance of the constructed LoRa/LoRaWAN infrastructure in terms of distance and packet delivery ratio.

The remainder of this document is organized as follows. In Section 2 LoRa, LoRaWAN and the components of the network infrastructure are described in details. In Section 3 we provide an analysis of our network implementation and a guide to replicate the setup. Section 4 describes the results of the performance evaluation. Last, Section 5 enlightens the conclusions and future works.

II. LoRa / LoRaWAN Architecture

Long Range (LoRa) is a proprietary technology belonging to the LPWAN (Low Power Wide Area Networks) family. These technologies have the purpose of providing wide area connectivity to low power and low data rate devices. In particular, LoRa defines the Physical layer of the technology.

On the other hand, LoRaWAN (Long Range Wide Area Network) is an open standard defining the communication protocol and system architecture for the network in which LoRa operates. LoRaWAN defines both the Media Access Control layer protocol and the Network layer protocol, enabling the communication between LPWAN gateways and end node devices. This second one is defined as a routing protocol, maintained by the LoRa Alliance. An overview of the LoRa stack is shown in Fig.1.

In terms of propagation medium, LoRa uses license-free sub-gigahertz radio frequencies like 169 MHz, 433 MHz, 868 MHz (Europe) and 915 MHz (North America).

The LoRaWAN architecture is composed by four components (fig. 2):

- End node(s)
- Gateway(s)

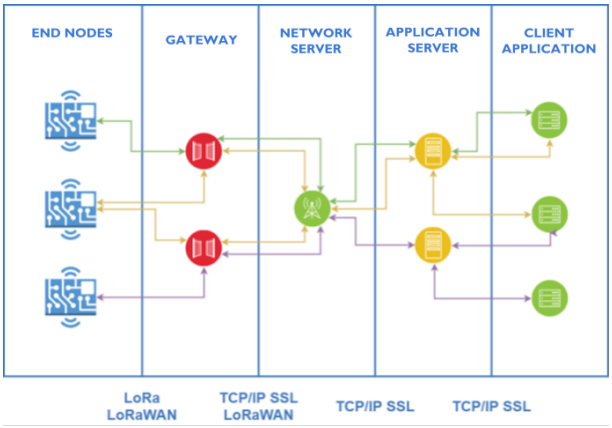

Fig. 2: LoRaWAN Architecture.

- Network server
- Application server

LoRa end nodes (clients) are not associated to any specific gateway, meaning that the data transmitted can be received by multiple gateways at the same time. At the MAC layer the communication uses the ALOHA protocol, therefore there is no listen-before-transmit mechanism, but instead a packet, as soon as it is ready, is transmitted after waiting a random number of time slots. As defined at the LoRa physical layer, multiple devices can communicate at the same time using different frequencies or spreading factors; moreover, the messages can be both confirmable or not confirmable, and a packet re-transmission can occur only at end node discretion.

LoRa gateways receive data from one or more end nodes and re-transmit the messages to a Network Server. The gateway should have the capability to receive and manage messages from a huge quantity of nodes, in order to create a long range star-topology network. By definition, a gateway can be single- or multi-channel; for simplicity, our project covers the implementation of a single-channel gateway.

The **Network Server** is the back-end component of the architecture and serves the purpose of completing complex tasks such as duplicate packets detection, Adaptive Data Rate control, and both the decryption and routing of packets payload to the Application Server.

The **Application Server** consists in an IoT-oriented application and it usually performs domain-specific data (pre)processing, therefore its interfacing with the Network Server is not standard.

III. Implementation

In this section we present our implementation of the LoRaWAN architecture described in the previous section.

A. End nodes

For our project we built two different types of end nodes, one using a Raspberry Pi and another using an Arduino. As described in the performance evaluation section, the choice for the underlying controller board had a remarkable impact on the performance of the entire system.

For both configurations we tried two different LoRa modules as well:

- a Futura Elettronica module [1], built over a HopeRF RFM98W transceiver (Semtech SX1278 chip) operating at 433 Mhz;
- a Dragino module [2], built over a HopeRF RFM96W transceiver (Semtech SX1276 chip) operating at 868 Mhz.

In addition to the frequency, the main difference between the two modules is the pinout, which is slightly different; regarding the integrated LoRa transceivers, they expose similar SPI interfaces, and we proved them to be inter-operable if set on the same frequency.

In order to emulate a real-world scenario we also equipped our end node with a DHT11 temperature/humidity sensor.

For each of our physical end nodes we developed the software to enable the up-link phase of the LoRaWAN protocol. The behaviour is straightforward: the value of a sensor (in our case the DHT11) is read, and a LoRaWAN packet is created; then, the payload of the packet is encrypted using two keys, a *network session key* and an *application session key* (their purpose is described later). Last, the message is sent in broadcast using LoRa in order to reach every gateway in the devices range.

The code used for the above communication was not written from scratch, but instead we used publicly available libraries to create the end nodes software:

- for the Arduino client, we used the LMIC (LoRaMAC in C) [3] library for both the creation of the packets and the as an interface to the actual LoRa chip;
- for the Raspberry Pi, we use a LoRaWAN Python library ([4]) for the creation of the packet, and a custom version of the popular RadioHead library [5] for the transmission of the message.

Our implementation for the end node has some limitations and it is not fully compliant to the LoRaWAN specifications. First of all, it only allows up-link transmissions, since it does not have any time windows for receiving down-link communications; second, it only works only via *Activation By Personalization* (ABP), and does not support the more recent *Over The Air Activation* (OTAA) mechanism for joining the network.

B. Gateway

The gateway is composed of a Raspberry Pi and a LoRa chip; as for the chip, we tested both the the Futura Elettronica and the Dragino modules. The Raspberry also needs to be connected to the Internet.

Our gateway performs the role of a LoRa packet forwarder, which is always listening for LoRa packets and forwards them to The Things Network (TTN) [6] server (via UDP) as soon as they are received.

The software running on the Raspberry Pi is a fork of a public single-channel packet forwarder [7] which resulted to

be the most used one. It natively supports both RFM9x and SX127x chips and requires the configuration of only a few parameters, *i.e.* the pins used to interface the RPi with the module, the operating frequency and the TTN server hostname.

The following is a sum up of the behaviour of the library:

- every time a LoRa packet is received it logs the payload and some relevant parameters, *i.e.* sender address, RSSI, working frequency, coding rate and spreading factor;
- a JSON object describing the received packet is built[8]
- such object is sent via UDP to TTN server;
- in addition, every 30 seconds the gateway also sends to TTN a status update, which consists in a separate JSON object that specifies the gateway status and the number of received packets.

Similarly to the end node, the gateway that we built has limited functionalities since it is not able to send messages to the devices, even though TTN would support this functionality; also, it is restricted to the use of a single channel at a time, but that is an unavoidable restriction since both the LoRa modules used are single channel.

C. Network and Application Server

Instead of developing a custom version of the LoRaWAN Network and Application Server we chose to rely on the services offered by The Things Network: it provides an open source implementation of both servers with the aim of building a global and interoperable network based on LoRaWAN for IoT applications; therefore, our decision to use TTN is also motivated by the fact that we wanted to use world-wide spread and already working solution supported by a vast and growing community. Moreover, it has the advantage of making the application data always available and easy to retrieve, also providing libraries to access its services implemented in several programming languages.

The configuration for our gateway on TTN required us to specify its EUI, a colon-separated identifier derived by the MAC address of the Raspberry Pi running the packet forwarder software.

The TTN network server receives the messages forwarded by the Raspberry Pi, discarding every packet whose content is not in the right format according to the LoRaWAN protocol, and discarding also every duplicate packet in case multiple gateways received the same transmission.

The correct messages are then forwarded to the application server. TTN is able to autonomously detect the correct destination application using the (unique) application session key specified by the end node. The application server itself performs a payload processing, interpreting the content (figure 3).

D. Client Application

The last component of our implementation is the Client Application. Its purpose is to demonstrate how data gathered from a remote sensor can be retrieved from the application server and used according to some logic. The application is composed of three separate sub-modules, which in our case are

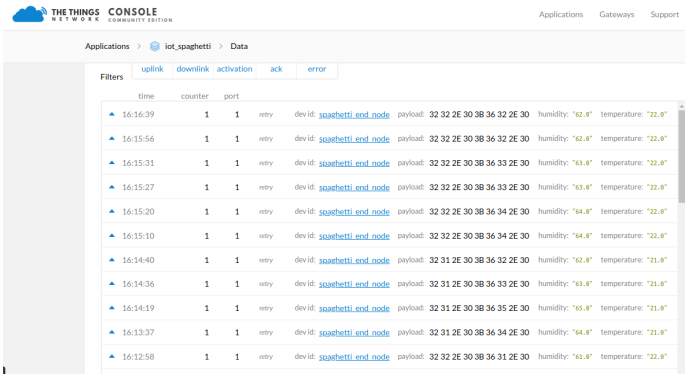

Fig. 3: TTN traffic.

.

hosted on the same physical machine, but that can be deployed on different machines if needed: an MQTT client, a time-series database (InfluxDB) and a web application (Grafana).

MQTT Client. A Python script that implements an MQTT client. It subscribes to the topic related to a single application server on the TTN MQTT broker. In order to successfully subscribe the script need to specify the application ID and its secret access key.

The client defines an handler for the ”Up-link Message Received” event, which retrieves the content and the source of the message and insert them into an InfluxDB instance.

The script is written using the TTN Python SDK [9] and the InfluxDB Python library.

InfluxDB. Time-series are an effective way to represent time-related data, such as sensor data. They are highly optimized for dealing with large quantity of data, which leads them to be effective in IoT scenarios.

Data from different LoRaWAN end nodes are stored in different database measurements. Every data point contains one field for every significant value (in our case temperature and humidity), plus an additional one for the RSSI of the message, useful for the performance evaluation phase.

Grafana. In order to process and visualize our data we used Grafana (Fig.4), a data visualization and monitoring tool which also integrate support for time-series databases, including InfluxDB.

In our implementation, Grafana periodically performs a query on the database to simply obtain new data points to plot, but the tool itself supports more complex tasks, like data analysis and forecasting.

IV. Setup Guide

While working on our project we were unable to find a complete and updated guide or tutorial covering all the necessary actions to perform in order to setup a LoRaWAN network, from the end nodes to the custom application logic. For this reason, and also to let other researchers easily re-create our testing setup for further performance evaluations, we decided to add to this report a brief summary on how to

Fig. 4: Grafana Interface with temperature and humidity plots.

setup a working prototype of a LoRaWAN network and how to build an application on top of it.

The configuration steps are the following:

1) on the Raspberry Pi, clone the content of our project repository [1]
2) connect the LoRa module to the Raspberry Pi following the wirings as in the scheme, which can be found in the repository linked above. Since we used both Futura Elettronica and Dragino boards, we provided the wirings for both of them
3) configure the *global_conf.json* file of the single-channel packet-forwarder software. We already provided a configured version which matches the wirings indicated at the previous step and which uses default values for spreading factor (7) and TTN server url
4) compile and run the packet forwarder software. During the start-up phase the EUI of the gateway is printed on the console: this should correspond to the MAC address of the Raspberry Pi, plus an "ff:ff" in the middle. Write it down, since it is required in the next step
5) use The Things Network web console to register a new gateway, paying attention to:
   a) check the "Legacy Packet Forwarder" option
   b) insert the EUI obtained at the previous step when required
6) using The Things Network web console register a new application
7) write down the default access key of the application, since it will be necessary for the MQTT client in order to retrieve the application data
8) from the application web page configure a new device; in its settings, select ABP as activation method (default is OTAA) and uncheck the "Frame Counter Check" option
9) from the TTN page of the device write down the following parameters, which are necessary for the end node software configuration:
   a) Device Address
   b) Network Session Key
   c) Application Session Key
10) setup the end node (the mentioned scripts can be found in our repository):
   a) *Arduino*: simply plug the LoRa shield on the Arduino and use the *arduino_end_node.ino* script, where the default values are configured with the ones obtained in the previous step;
   b) *Raspberry*: connect the LoRa module using the same wirings from step 2; the end node script is *raspberry_end_node.py*, where the default values are configured with the ones obtained at the previous step.
11) from the TTN application page define a decoder function, whose role is to transform the message payload from an array of bytes into a JavaScript object
12) insert the application ID and default access key in the script *retrieve_message.py*: the script will also log every message received by the gateway

You now have a working single-channel packet-forwarder, an end node, and an MQTT client, all of them interfaced with TTN.

[1] https://github.com/aleeraser/TTN-LoRaWAN-Python-Interface

V. Performance evaluation

The performance evaluation has been done using Dragino LoRa shields, with an arduino as end node and Futura Elettronica LoRa shields with a raspberry pi 3b+ as end node.

We evaluated the following aspects of the wireless transmission:

- Maximum Distance
- Package Delivery Ratio (PDR)
- (Average) Received Signal Strength Indication (RSSI)

We gathered the PDR and the average RSSI in different scenarios, also changing the transmission frequency (number of packet transmitted per time unit) and the distance between the end node and the gateway. In any of these cases, the gateway has been placed at the 6th floor of a residential building in the periphery of Bologna; on the other hand, the end node has been connected to a mobile power source and moved around.

The first experiment simply consisted in a **maximum covered distance** test. The path shown in Fig 5 delineates the path followed by the end node.

We placed the end nodes as a maximum distance of 1.9 km. In such scenario, we reached a maximum transmission distance of 1 km for the end node equipped with the Raspberry Pi, while the one equipped with an Arduino reached a transmission distance of 1.6 kilometers.

The last experiment evaluates the PDR and the average RSSI while varying the distance between the end node and the gateway. The results are reported in Table I and II for the Raspberry Pi and Arduino nodes respectively.

We found the performances of the Raspberry Pi to be pretty bad, considering the fact that this technology has been designed for high range communications. We suspect that, being the LoRa shields conceptually made for an Arduino, there could be some electronical incompatibilities (*e.g.* different pin

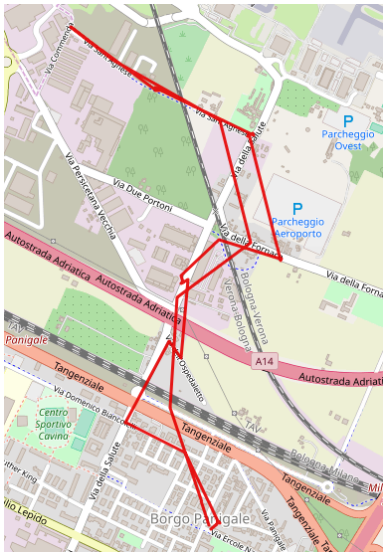

Fig. 5: Path covered by the end node

Distance (m)	PDR (%)	avg RSSI
20	17.23	-50.34
35	5.0	-56.6
220	15.0	-57.09
500	0	N/A
650	2.0	-92
850	1.0	-92
1000	2.0	-93

Indoor

TABLE I: Values relative to a Rapberry Pi end node equipped with a Futura Elettronica LoRa shield.

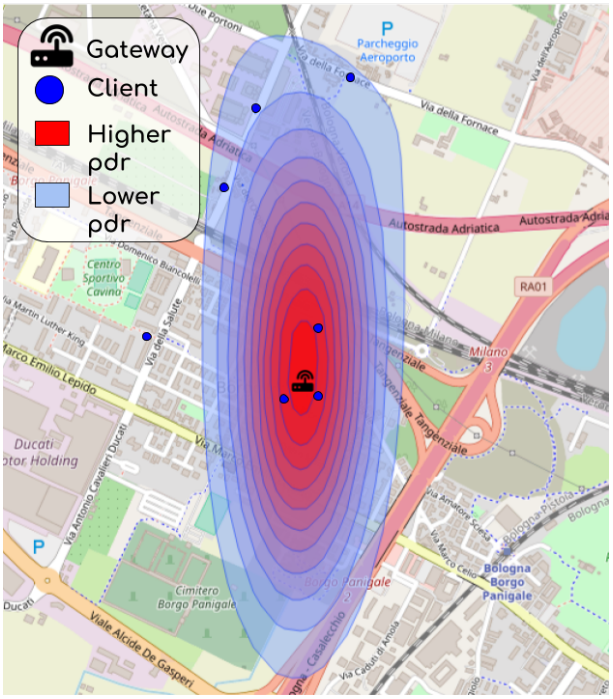

Fig. 6: PDR heatmap for a Raspberry Pi and Futura Elettronica LoRa shield.

voltage, low maximum pin current output, ecc.), which are however outside of the scope of this project.

In both cases the PDR started to decrease consistently starting from 500 meters, probably due to increasing interference.

Fig. 6 and 7 represent two heatmap relative to the experiment: the red zones are the one with highest PDR (∼20% for the Raspberry Pi and 100% for the Arduino). Since the heatmaps use the same scale, they show clearly how the transmission range achieved with an Ardunio end node is way larger than the one obtained with a Raspberry Pi.

VI. Conclusions and Future Works

In this report we described briefly the LoRa technology and the LoRaWAN network. We also showed how using relatively

Distance (m)	PDR (%)	avg RSSI
20	100	-80.23
35	100	-67
220	92	-103.48
500	18.0	-105.42
650	100	-104.44
850	79.0	-105.08
1000	70.0	-105.62
1400	70.0	-106.02
1600	50.0	-108.46
1900	0.0	

Indoor

TABLE II: Values relative to an Arduino end node equipped with a Futura Elettronica LoRa shield.

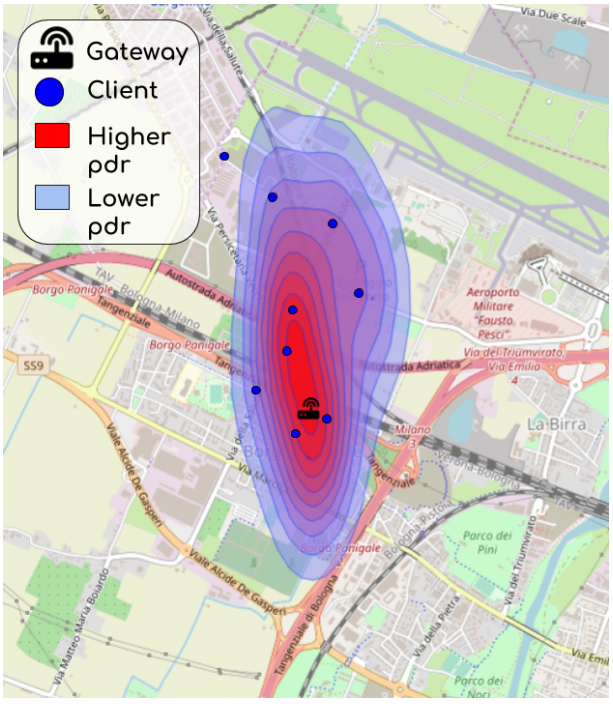

Fig. 7: PDR heatmap for an Arduino and Dragino LoRa shield.

cheap components, such as a Raspberry Pi and a LoRa shield, it is possible to build a simple single-channel packet-forwarder LoRaWAN gateway. The gateway is also fully inter-operable with The Things Network, which offers the advantage of being able to take part in one of the most spread and fast-growing LoRaWAN network.

As described in the report, our work currently has some limitations, which could be explored in future works. The most relevant limitation is the lack of down-link communication support, on both end nodes and gateway. A second possible improvement is related to the mechanism used by the gateway to join the TTN network, which currently happens via *Activation By Personalization*: it would be advisable to use the more recent *Over The Air Activation* (OTAA) which, among the other advantages, does not require the network and application secret keys to be hard-coded into the gateway software but are instead negotiated with TTN. Last, a great improvement would be the implementation of a multi-channel gateway, but this would require a totally different approach and hardware support.

In order to ease future improvements for other researchers, and to save them some headaches caused by the lack of documentation, we included in the report a complete guide on how to reproduce our case of study.

Overall, this project allowed us to get in touch for the first time with the LoRa/LoRaWAN technology. Being able to receive messages from distances in the order of kilometers with an hardware cost of less than 50 euros is amazing, and it is not difficult to reckon how promising this technology is in the IoT world.

REFERENCES

[1] Futura elettronica lora shield. https://www.futurashop.it/lora-shield-arduino-fishino-montato-3085-LORASHIELD433. accessed : 2018-11-23.

[2] Dragino lora shield. http://www.dragino.com/products/module/item/102-lora-shield.html. accessed : 2018-11-23.

[3] Arduino lmic. https://github.com/matthijskooijman/arduino-lmic. accessed : 2018-11-23.

[4] Jeroen Nijhof. Lorawan. https://github.com/jeroennijhof/LoRaWAN. accessed : 2018-10-24.

[5] Radiohead python wrapper. https://github.com/exmorse/pyRadioHeadRF95. accessed : 2018-11-23.

[6] The things network. https://www.thethingsnetwork.org/. accessed : 2018-11-23.

[7] Hallard single-channel-packet-forwarder. https://github.com/hallard/single_chan_pkt_fwd. accessed : 2018-11-23.

[8] The things network protocol. https://github.com/Lora-net/packet_forwarder/blob/master/PROTOCOL.TXT. accessed : 2018-11-23.

[9] Ttn python sdk. https://github.com/TheThingsNetwork/python-app-sdk. accessed : 2018-11-23.