What are the advantages of the LoRa modulation?

* Better battery life: LoRa devices consume very little power making it ideal for battery-powered devices
* Long range: It can transmit and receive data for up to 15 km in suburban areas and 5 km in urban areas.
* Cost effective for large deployment
* Both bandwidth and frequency are scalable
* It is perfect for single-building applications.
* You can set up and manage your own network.

How LoRa is compatible with LPWAN requirements and constraints?

|  |  |
| --- | --- |
| LPWAN Requirements | Lora specifications |
| Low device complexity and cost | ✔ |
| Reliability under extreme coverage conditions | ✔ |
| Low power consumption: long battery lifetime | ✔ |
| High capacity: support for massive number of low-rate devices | ✔ |
| Simplified network topology and deployment | ✔ |

As previously mentioned in the LoRa advantages, this modulation has all the LPWAN requirements.

* 1. **Hardware Platform**
* We are using 2 Arduinos both of them are model **UNO.**
* Number of pins: 32?
* Digital I/O Pins: 14 (of which 6 provide PWM output)

PWM Digital I/O Pins: 6

Analog Input Pins: 6

DC Current per I/O Pin: 20 mA

DC Current for 3.3V Pin: 50 mA

* Memory size:

EEPROM: 1KB

Flash memory: for program storage: 32 KB

SRAM: used for local variables: 2KB

* Main characteristics of LoRa shields:
* Compatible with 3.3v or 5v I/O Arduino Board.
* Frequency Band: 915MHz/868 MHZ/433 MHZ (Pre-configure in factory)
* Low power consumption
* Compatible with **Arduino Leonardo, Uno, Mega, DUE**
* External Antenna via I-Pex connector
* 168 dB maximum link budget.
* +20 dBm - 100 mW constant RF output vs.
* +14 dBm high efficiency PA.
* Programmable bit rate up to 300 kbps.
* High sensitivity: down to -148 dBm.
* Bullet-proof front end: IIP3 = -12.5 dBm.
* Excellent blocking immunity.
* Low RX current of 10.3 mA, 200 nA register retention.
* Fully integrated synthesizer with a resolution of 61 Hz.
* FSK, GFSK, MSK, GMSK, LoRaTM and OOK modulation.
* Built-in bit synchronizer for clock recovery.
* Preamble detection.
* 127 dB Dynamic Range RSSI.
* Automatic RF Sense and CAD with ultra-fast AFC.
* Packet engine up to 256 bytes with CRC.

**4.1. Time on Air**

* We have changed the following parameters:
  + SF (7,9, 10)
  + BW (20 800, 31 250, 62 500, 5 000 000, 1 250 000)
  + CR (5, 6, 7, 8)
  + Packet size (10, 22 443)

Then using Python, we have changed the Serial Monitor in order to measure the amount of time that Arduino took to send a message. We measured 100 samples of TOA for each of the parameters, then drew a boxplot showing the 100 values and the median values in order to visualize the distribution of TOA for each parameter.

Code I : (listener qui va écouter Arduino en utilisant le serial correspondant)

import serial #Serial imported for Serial communication

import time #Required to use delay functions

*def* initialize() :

    ArduinoSerial = serial.Serial('com3',115200)

    time.sleep(2)

    return ArduinoSerial

*def* read(*ArduinoSerial*, *liste1*):

    a = ArduinoSerial.readline()

    liste1.append(int(a.decode().replace("\r\n", "")))

    return liste1

*def* main():

    AS = initialize()

    print("initialized")

    liste1 = []

    i = 0

    print("listening")

    while (i< 100):

        print(".\n")

        liste1 = read(AS, liste1)

        i += 1

    f = open("DATA10.txt", "w")

    f.write(str(liste1))

    f.close()

    print(liste1)

main()

Code II : (code qui dessine SF en fonction du temps en utilisant matplotlib)

import matplotlib.pyplot as plt

import ast

*def* getData(*file*):

    SF = open(file, "r")

    values = SF.read()

    SF.close()

    return values

*def* drawFig(*Data*):

    fig1, ax1 = plt.subplots()

    ax1.set\_title('Basic Plot')

    ax1.boxplot(Data)

    plt.show()

*def* main():

    SF7values = ast.literal\_eval(getData("SF7.txt"))

    SF9values = ast.literal\_eval(getData("SF9.txt"))

    SF10values = ast.literal\_eval(getData("SF10.txt"))

    drawFig([SF7values, SF9values, SF10values])

main()