PROJECT

Topic: LoRa

Description:

In the project, students will instantiate a LoRa System, composed of several LoRa nodes. This will allow the students to get familiar with the LoRa-2-LoRa communications, its main functional blocks and processes. The project will be implemented using opensource software and programmable boards (LoPy).

General objectives:

1. get an hands-on understanding on the internals of a LoRa-to-LoRa link
2. be able to identify and configure the main parameters of a LoRa-to-LoRa link
3. Analyze the performance of the implemented system in different conditions

Components to be used:

1. PyCom Expansionboard v2.1A
2. PyCom Lopy 1.0r

Project proposed initial steps:

1. Analyse the provided links, realizing the tutorials (you can try to find more on your own)
2. Prepare the LoPy, making sure that it is connected into the ExpansionBoard with the RGB led and the reset button oriented to the USB connector side. Connect the USB cable to the computer and to the device. Check if the USB led of the ExpansionBoard turns on. It is preferable to use a IDE, such as Atom (recommended) or VSCode, with the "pymakr" extension installed. This allows you to program the LopY by either flashing the desired program to its memory or program it directly in a type of terminal (REPL) inside the IDE. You need to install node.js for this to work.
3. Establish a LoPy to LoPy link, between two nodes (check Annex I)

Suggested additional steps:

4. Change the information being sent between the nodes from 'ping' to something else (i.e., changing the content, or using the received ping to 'act' on something on the receiving node, such as activating/deactivating a LED)
5. Try to analyze the impact to communications when increasing the distance between nodes (i.e., collect the SNR and RSSI values using the "lora.stats()" function)

For all the experimentations above, annotate and explain the messages exchanges between the involved components.

URLs:

1: https://docs.pycom.io/gettingstarted/

2: https://ttm4175.iik.ntnu.no/iot-lopy.html

3: In case you are considering using IDE's based on Visual Studio (i.e., VSCode, Atom), you have a dedicated extension to interact with the device (note that it requires node.js) : https://marketplace.visualstudio.com/items?itemName=pycom.Pymakr https://atom.io/packages/pymakr

4: https://docs.pycom.io/tutorials/networks/lora/module-module/

Note: The objective of this work is not to teach you Python. The code here presented has not been maintained throughout new versions of the IDE's, Pymkr, versions of the Python language itself, so it is not guaranteed to work or not have syntax errors. The code itself is suggestive in nature, you can or might need to adopt changes or variations in order to achieve the result. It is expected that the students are autonomous in programming and operating the devices.

Annex I – Initialization

On a location of your choosing, create a directory and open it in the chosen IDE. Inside it, create 2 folder for the code of both devices. Name them "NodeA" and "NodeB" and create a file **main.py** in both. Start with writing the following code in the files:

```
from network import LoRa
import socket
import time

# Initialize LoRa in raw LORA mode
lora = LoRa ( mode = LoRa . LORA )
# Create a raw LoRa socket
s = socket . socket ( socket . AF_LORA , socket . SOCK_RAW )
s. setblocking ( False )
```

Initializing the variable lora in the mode **LoRa.LORA** bypasses the LoRaWAN layer, allowing its usage using only the LoRa radio capabilities needed for this part. The variable s is used to create a raw LoRa socket that will allow sending and receiving messages between the devices. Also, it is set as non-blocking to prevent being blocked if there's no data received. Keep on to Annex II.

Annex II – Communication between devices

Now it is necessary to "configure" the devices to communicate between them. For this task, one of the devices will be sending information while the other will be only listening.

Node A

This device will be the one sending information to the other node. For that,use the following code:

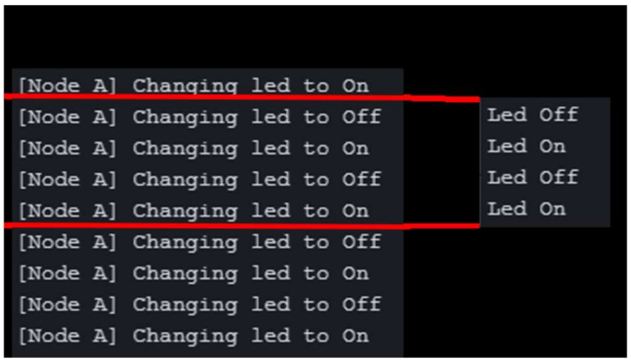
```
enable_led = True
while True :
      s. send (str ( enable_led ))
      print ('[ Node A] Changing led to {} '. format ('On ' if
enable_led else 'Off '))
      enable_led = not enable_led
      time . sleep (5)
```

Node B

While Node A sends information, this device needs to listen and, for that, it will do the following:

```
while True :
      message_received = s. recv (64)
      if message_received == b'True ':
            print ('Led On ')
      elif message_received == b'False ':
            print ('Led Off ')
      time . sleep (5)
```

Run both codes, in each device, and check if the output seems like below:



```
[Node A] Changing led to On
[Node A] Changing led to Off        Led Off
[Node A] Changing led to On         Led On
[Node A] Changing led to Off        Led Off
[Node A] Changing led to On         Led On
[Node A] Changing led to Off
[Node A] Changing led to On
[Node A] Changing led to Off
[Node A] Changing led to On
```

In order to better showing that the devices are really communicating, Node B will now react to the information sent by Node A using one of its LEDs. Add this code to Node B’s previous code:

```
# Add these lines
from machine import Pin
led = Pin ('P9 ', mode =Pin .OUT )
...
if message_received == b'True ':
      # Add this line
      led . value (0)
      print ('Led On ')
elif message_received == b'False ':
      # Add this line
      led . value (1)
      print ('Led Off ')
```

Run the code again and check if it is working properly. Move on to the next annex.

Annex III – Bilateral connectivity

Most of the cases it is necessary that the communication between devices is done bilaterally, i.e., both the devices are able to send and receive data. Thus, it is required to add some code to both devices.

Node A

Between the lines "`enable_led = not enable_led`" and "`time.sleep(5)`" add:

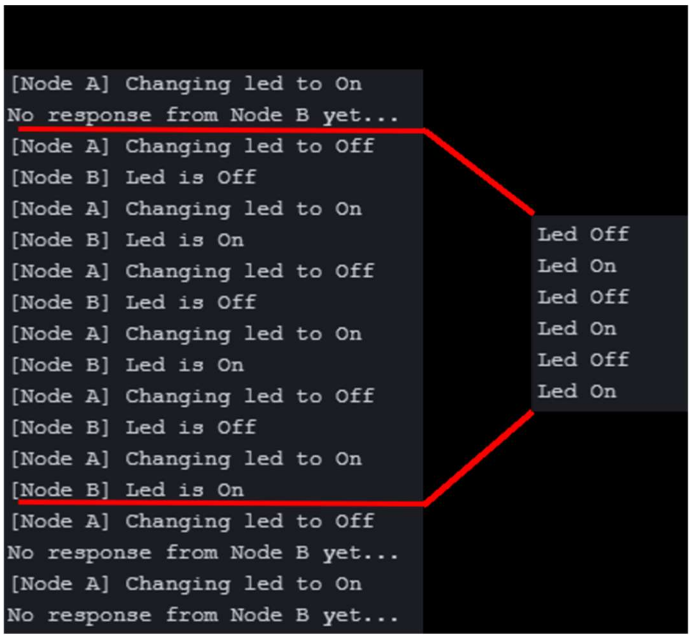
```
...
message_received = s. recv (64)
if message_received == b '':
      print ('No response from Node B yet ... ')
else :
print ( message_received . decode ())
```

Node B:

```
if message_received == b'True ':
      led . value (0)
      # Add this line
      s. send ('[ Node B] Led is On ')
      print ('Led On ')
elif message_received == b'False ':
      led . value (1)
      # Add this line
      s. send ('[ Node B] Led is Off ')
      print ('Led Off ')
```

Finally, run both codes and see if they are communicating as expected.

Compare the results with the following figure:



```
[Node A] Changing led to On
No response from Node B yet...
[Node A] Changing led to Off
[Node B] Led is Off
[Node A] Changing led to On
[Node B] Led is On
[Node A] Changing led to Off
[Node B] Led is Off
[Node A] Changing led to On
[Node B] Led is On
[Node A] Changing led to Off
[Node B] Led is Off
[Node A] Changing led to On
[Node B] Led is On
[Node A] Changing led to Off
No response from Node B yet...
[Node A] Changing led to On
No response from Node B yet...
```

```
Led Off
Led On
Led Off
Led On
Led Off
Led On
```