Demonstration of a library prototype to build LoRa mesh networks for the IoT

Joan Miquel Solé, Sergi Miralles Nogués, Roger Pueyo Centelles, Felix Freitag
Department of Computer Architecture. Technical University of Catalunya. Barcelona, Spain
{joan.miquel.sole, sergi.miralles}@estudiantat.upc.edu, {roger.pueyo, felix.freitag}@upc.edu

***Abstract*—LoRa has become popular in the Internet of Things (IoT) domain as a Low Power, Wide Area Network (LPWAN) radio technology providing low-power and long-range communication. In a typical IoT application, the LoRaWAN architecture is applied, where LoRa end nodes communicate their data to a gateway, which then over the Internet sends these data to a cloud-based service for further processing. However, LoRa can also be used standalone for the communication between LoRa nodes forming a mesh network. In this demo paper we present a library called LoRaMesher, which runs on LoRa nodes and forms a mesh network among these nodes. By implementing a distance vector routing protocol, LoRaMesher enables two nodes to communicate data packets with each other while the other nodes in the mesh network operate as routers. LoRaMesher can open the possibility for new distributed applications hosted only on such tiny IoT nodes.**

***Index Terms*—LoRa, Mesh networks.**

I. Introduction

LoRa is a wireless communication technology designed for the Internet of Things (IoT). It provides long range communication, with links of several km between devices, low power consumption and low data rate. Its Configuration parameters like the Spreading Factor (SF) can extend its range, although at the expense of lower data rates [1].

LoRaWAN is the dominant architecture for IoT deployments using LoRa [2]. In LoRaWAN LoRa-equipped sensor nodes (i.e., end devices) transmit their data to nearby gateways. The gateways, in turn, forward these messages to a network server over an Internet connection. This way, data from the nodes reach an application server, where the data can be further processed by higher level IoT services. An example of a large, public LoRaWAN deployment is The Things Network [1].

Researchers have made several proposals to extend the LoRaWAN architecture by means of multi-hop, mesh and routing protocols [3] and envisioned new application scenarios if the communication between end nodes was available [4]. A few works show deployments in real field environments, such as Ebi et al.'s [5], who proposed a synchronous LoRa mesh network, and Meshtastic [6], an open-source project to establish LoRa networks for short text messages. Neither of both, however, uses a routing protocol to forward messages.

We have implemented a library codenamed *LoRaMesher*. In this paper we demonstrate this library which implements a routing protocol for LoRa mesh networks. The routing protocol is based on a distance-vector designed in [7], where each LoRa node maintains a routing table about the network that is periodically updated taking into account the routing packets received from the directly connected neighbors.

The main contributions of this paper are:

- We highlight some of the implementation aspects of the LoRaMesher library.
- We demonstrate the library with real nodes. For this we run an experiment in which LoRa data and routing packets are sent between the nodes in a LoRa mesh network.

II. Library implementation

The LoRaMesher library [2] implements a distance-vector routing protocol for exchanging messages among LoRa nodes. For the interaction with the LoRa radio chip, we leverage RadioLib [3], a versatile communication library which supports the SX1276 LoRa series module available on the hardware we use.

We use FreeRTOS [4] to implement task handlers for the receiver, sender, packet processing, routing protocol and user processing. We have implemented a *packet queue* to share packets between tasks.

When a node in the LoRa mesh network receives a message, the *packet processing task* in the LoRaMesher library determines whether a message is a data or a routing packet. If the data packet's destination not the node itself and it has to be routed to another node, a path towards the destination is determined using the node's routing table.

It is important that the receiver task is occupied for the minimum time. Otherwise, if two packets arrive consecutively, one may get lost due to the lack of a time slot for listening to both LoRa signals. Therefore, in our implementation the receiver task encapsulates the packet and adds it to the *received packets queue*, to be processed when the microcontroller is available to do it.

The sender task checks if there are packets inside the *send packet queue*. If so, then the task sends them in a determined interval, by default every 10 seconds.

The packet process task identifies the two types of packet:

- Routing packet: includes a header of 6 bytes (*source*: 2 bytes, *destination (broadcast address)*: 2 bytes, *type*: 1

[1]https://www.thethingsnetwork.org/

[2]https://github.com/LoRaMesher/LoRaMesher
[3]https://github.com/jgromes/RadioLib
[4]https://www.freertos.org/

byte, *payload size*: 1 byte) and the payload containing the node's routing table.
- Data packet: includes a header of 8 bytes added by the LoRaMesher library to the packet (*source*: 2 bytes, *destination*: 2 bytes, *via*: 2 bytes, *type*: 1 byte, *payload size*: 1 byte) and the payload containing the application data.

Finally, for the packets to the application, there is the *user received packets queue*. When the process packet routine receives a data packet whose destination is the node's application, then the routine adds it to this priority queue and notifies the application task.

III. Experimentation with the library

For the experimentation we use ten TTGO T-Beam ESP32 boards as can be seen in Figure 1 and flash them with the LoRaMesher library. These boards use the ESP32 System on a Chip (SoC) and feature an SX1276 LoRa transceiver.

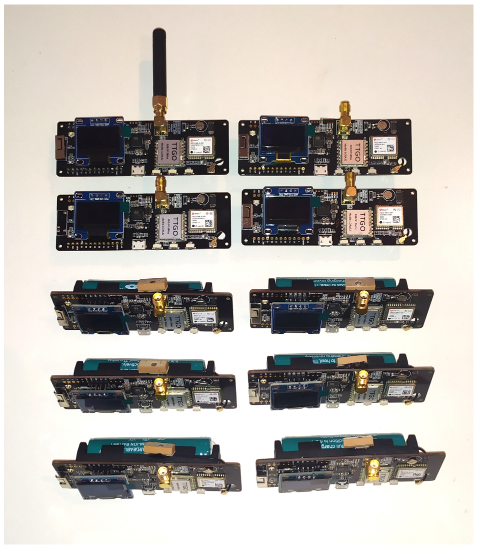


Figure 1. Ten T-Beam boards with LoRa radio used for the experimentation.

A. Experiment description

We design an experiment where the application on each of the T-Beam nodes sends LoRa *data packets* every 20 seconds. The data packets contain a numeric counter value, and they are sent to all reachable nodes following consecutively the routing table of the node.

In addition, *routing packets* are sent by the library between neighboring nodes for building and updating the nodes' routing tables. The routing packets are sent every 30 seconds. A routing packet contains a node's routing table, which is shared with the other nodes.

Figure 2 shows the OLED display of a T-Beam node that is part of the experiment. On each line of the display there is information about the protocol and the application. The first line is the device ID. The second line is the number of packets that have been sent. The third line is the last packet that has arrived, showing the device ID and the data of the packet. And lastly, the last two lines show in a rotating manner the routing table of the device in the format of "|Device ID (Hops) → Via ID|".

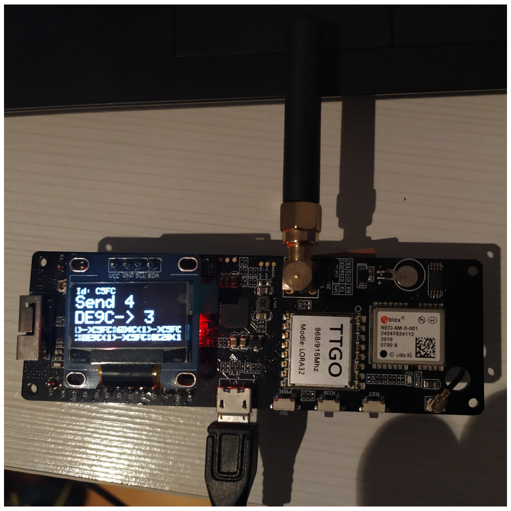


Figure 2. T-Beam with a functional OLED display.

In the experimentation we use ten devices and monitor the library interactions within a duration of 2 minutes. Given the above described configuration, within the experiment each node sends 4 routing table packets and 6 data packets. This results in a total of 100 packets sent among all the devices.

B. Experimental mesh network topology

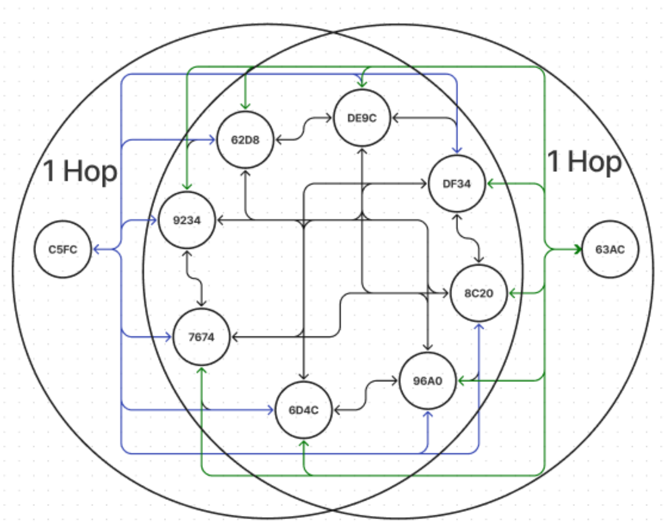
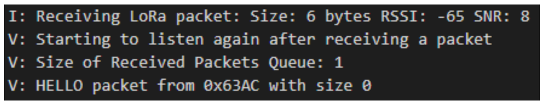


Figure 3. Topology of the experimental LoRa mesh network. There are two nodes, 0xC5FC and 0x63AC, that do not have a direct link and data packets need to be routed through one of the other 8 nodes.

In order to demonstrate the library, we have created a controlled environment, where the ten boards are placed close to each other. However, two of the network nodes, which have the identifiers 0xC5FC and 0x63AC, are modified so that they cannot communicate with each other directly. This means that if these two nodes want to transmit data packets to each other, the packets must be routed through one of the other eight nodes. The corresponding network topology, which is formed by the routing tables created by the library at each node, is shown in Figure 3.

C. Routing table packets



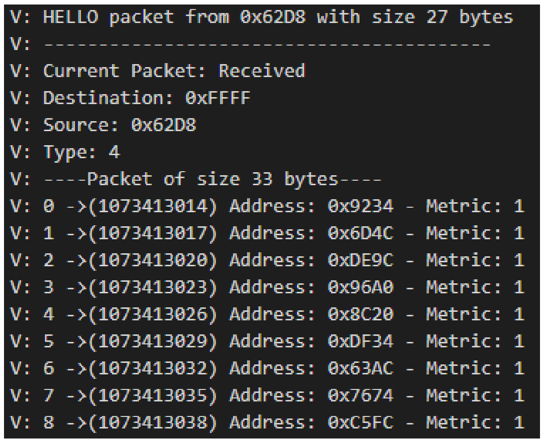
```
I: Receiving LoRa packet: Size: 6 bytes RSSI: -65 SNR: 8
V: Starting to listen again after receiving a packet
V: Size of Received Packets Queue: 1
V: HELLO packet from 0x63AC with size 0
```

Figure 4. Trace of the library obtained from the development environment. Shows the logs inside the Received and Process Task. Routing packet from node 0x63AC.

In this section we analyze the routing packets of the library. In order to observe the behavior, the T-Beam boards are connected over the serial port to a PC. Through the serial port monitor of the development environment we can see the logs of the library. Every time a packet is detected by the board the library write a log message containing relevant information.

Figure 4 shows the log messages of a routing packet received from node 0x63AC. The reception task receives the packet (line 1), encapsulates the packet and stores it into the received packets queue (line 3). In the last line, the process packets task decapsulates the packet and process it, identifying it as a routing packet.

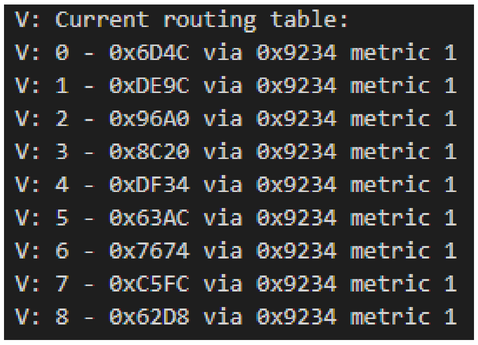
As it has been mentioned before, each node broadcast its routing table through a routing packet which is received by the neighboring nodes. In Figure 5 the information inside a routing packet can be seen, containing the source, the type of packet and the routing table, including the metrics. Furthermore, this packet is 33 bytes long with 27 bytes corresponding to the routing table (9 routes, 3 bytes each one). The number inside the brackets indicates the position in memory of the routing table.



```
V: HELLO packet from 0x62D8 with size 27 bytes
V: ----------------------------------------
V: Current Packet: Received
V: Destination: 0xFFFF
V: Source: 0x62D8
V: Type: 4
V: ----Packet of size 33 bytes----
V: 0 ->(1073413014) Address: 0x9234 - Metric: 1
V: 1 ->(1073413017) Address: 0x6D4C - Metric: 1
V: 2 ->(1073413020) Address: 0xDE9C - Metric: 1
V: 3 ->(1073413023) Address: 0x96A0 - Metric: 1
V: 4 ->(1073413026) Address: 0x8C20 - Metric: 1
V: 5 ->(1073413029) Address: 0xDF34 - Metric: 1
V: 6 ->(1073413032) Address: 0x63AC - Metric: 1
V: 7 ->(1073413035) Address: 0x7674 - Metric: 1
V: 8 ->(1073413038) Address: 0xC5FC - Metric: 1
```

Figure 5. Routing packet sent by the node 0x62D8 after 37 seconds, received in the node 0xC5FC.

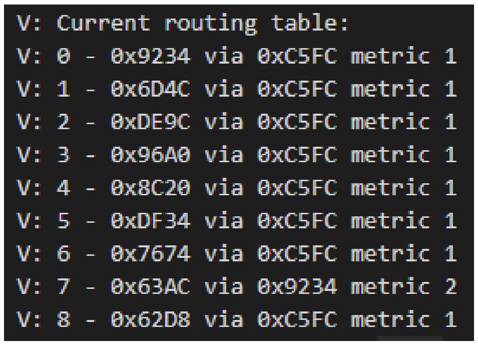
The topology shown in Figure 3 is fully built by the nodes' routing tables when all the nodes have sent between 1 and 2 routing packets. With an exchange of routing packets every 30 seconds it takes about one minute till the library achieves that the routing tables of all the nodes have been built to represent this topology.



```
V: Current routing table:
V: 0 - 0x6D4C via 0x9234 metric 1
V: 1 - 0xDE9C via 0x9234 metric 1
V: 2 - 0x96A0 via 0x9234 metric 1
V: 3 - 0x8C20 via 0x9234 metric 1
V: 4 - 0xDF34 via 0x9234 metric 1
V: 5 - 0x63AC via 0x9234 metric 1
V: 6 - 0x7674 via 0x9234 metric 1
V: 7 - 0xC5FC via 0x9234 metric 1
V: 8 - 0x62D8 via 0x9234 metric 1
```

Figure 6. Final routing table after 42 seconds of execution in the node 0x9234.

Figure 6 shows the routing table of the node with Id 0x9234, which is one of the middle nodes of the topology. Corresponding to the given network topology, all the entries are within one hop.



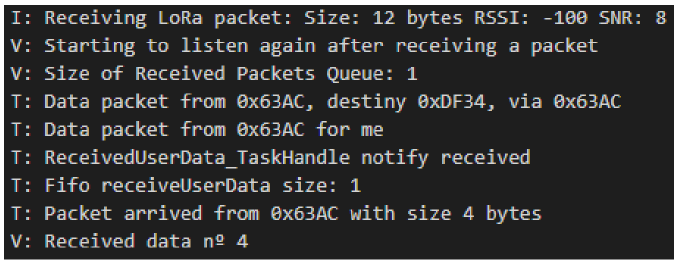
```
V: Current routing table:
V: 0 - 0x9234 via 0xC5FC metric 1
V: 1 - 0x6D4C via 0xC5FC metric 1
V: 2 - 0xDE9C via 0xC5FC metric 1
V: 3 - 0x96A0 via 0xC5FC metric 1
V: 4 - 0x8C20 via 0xC5FC metric 1
V: 5 - 0xDF34 via 0xC5FC metric 1
V: 6 - 0x7674 via 0xC5FC metric 1
V: 7 - 0x63AC via 0x9234 metric 2
V: 8 - 0x62D8 via 0xC5FC metric 1
```

Figure 7. Final routing table after 37 seconds of execution in the node 0xC5FC.

In Figure 7 the routing table of the node 0xC5FC can be seen. It can be observed that all the nodes are at one hop but one with two. This last entry is the other node with Id 0x63AC.

D. User data packets

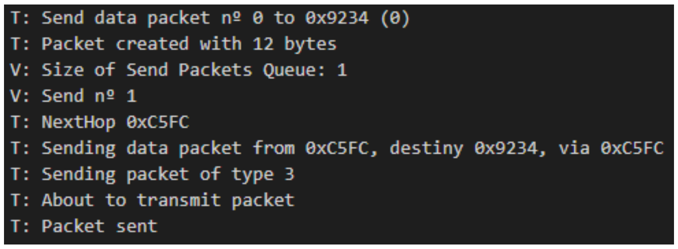
In this section we analyze the application's data packets that are routed by the library. In Figure 8 we see how a 12



```
I: Receiving LoRa packet: Size: 12 bytes RSSI: -100 SNR: 8
V: Starting to listen again after receiving a packet
V: Size of Received Packets Queue: 1
T: Data packet from 0x63AC, destiny 0xDF34, via 0x63AC
T: Data packet from 0x63AC for me
T: ReceivedUserData_TaskHandle notify received
T: Fifo receiveUserData size: 1
T: Packet arrived from 0x63AC with size 4 bytes
V: Received data nº 4
```

Figure 8. Trace of the application and library obtained from the development environment. Shows the logs inside the Received, Process Task and User Task. Data packet from node 0xDF34 to node 0x63AC.

bytes packet, 8 bytes of header + 4 bytes of payload, has been received (line 1), stored into the received packets queue (line 3), identified as a data packet and for this node (line 4 and 5) and delivered to the user task application (line 6 and 7), where it displays the numerical value (in this case 4) inside the packet.

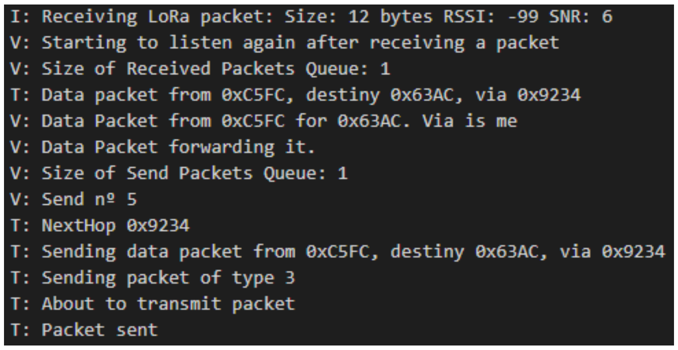


```
T: Send data packet nº 0 to 0x9234 (0)
T: Packet created with 12 bytes
V: Size of Send Packets Queue: 1
V: Send nº 1
T: NextHop 0xC5FC
T: Sending data packet from 0xC5FC, destiny 0x9234, via 0xC5FC
T: Sending packet of type 3
T: About to transmit packet
T: Packet sent
```

Figure 9. In the node 0xC5FC, sending a data packet to the node 0x9234.

There is an specified task to send the packets, which blocks the antenna and disabled the reception of packets. Once the user data task or the routing protocol task wants to send a packet, it will add it into the send packets queue and then, the send task will get this packets and, if needed, will add the next hop, and finally send it.

In Figure 9 the first two lines show the creation of the data packet, the third one adds the packet inside the send packets queue, the fifth line changes the next hop, and the last lines shows the task sending the packet.



```
I: Receiving LoRa packet: Size: 12 bytes RSSI: -99 SNR: 6
V: Starting to listen again after receiving a packet
V: Size of Received Packets Queue: 1
T: Data packet from 0xC5FC, destiny 0x63AC, via 0x9234
V: Data Packet from 0xC5FC for 0x63AC. Via is me
V: Data Packet forwarding it.
V: Size of Send Packets Queue: 1
V: Send nº 5
T: NextHop 0x9234
T: Sending data packet from 0xC5FC, destiny 0x63AC, via 0x9234
T: Sending packet of type 3
T: About to transmit packet
T: Packet sent
```

Figure 10. The node 0x9234 forwards a message from node 0xC5FC to node 0x63AC.

In Figure 10 it can be observed how node 0x9234 receives a data packet from node 0xC5FC, which has its Id in the packet's *via* field. When this happens the node forwards the message to the next hop, in this case it is the destination 0x63AC.

E. Aditional analysis on operational performance

With regards to the short term operation, we analyze the experiment described in section III-A. Specifically, we measure the packet delivery ratio. We power on the ten devices flashed with the library and application code in a consecutive way with a delay of approximately 1 second. Each device starts configuring the node and then running the application during 2 minutes. By analyzing the logs of nodes we observe an average packet delivery ratio of around 95%. When the devices are powered on simultaneously, we observe that the packet delivery ratio decreases, which can be attributed to a higher number of collisions of LoRa packets from nodes sending at the same time.

For a long term operation we run a 12 hour experiment. This experiment aims to confirm that the library implementation does not produce any overflows or memory exhaustion at the devices. Observing the logs of the nodes and the counter value of the data packets at the end of the experiment we found that the behavior of the library was correct.

IV. Demonstration

The demonstration of the experimentation with the LoRa-Mesher library highlights the following aspects:

1) Application level: It is demonstrated that the LoRa-Mesher library can be used by an application running at the nodes. For this, the sending of data packets and reception of these packets at the destination nodes, which are part of the LoRa mesh network, is shown.
2) LoRaMesher library operation: It is demonstrated that the implementation of the library sends routing packets and that the nodes build a routing table that corresponds to the network topology.

Future work includes conducting a detailed performance evaluation of the LoRa mesh network enabled by the library. Furthermore, we aim to explore the usage of the library by real IoT applications.

Acknowledgment

This work was partially supported by the Spanish Government under contracts PID2019-106774RB-C21, PCI2019-111851-2 (LeadingEdge CHIST-ERA), PCI2019-111850-2 (DiPET CHIST-ERA).

References

[1] A. Augustin, J. Yi, T. Clausen, and W. M. Townsley, "A study of LoRa: Long range & low power networks for the internet of things," *Sensors*, vol. 16, no. 9, 2016. [Online]. Available: https://www.mdpi.com/1424-8220/16/9/1466
[2] "LoRa alliance," https://lora-alliance.org/, accessed: 2022-03-03.
[3] J. R. Cotrim and J. H. Kleinschmidt, "LoRaWAN mesh networks: A review and classification of multihop communication," *Sensors*, vol. 20, no. 15, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/15/4273
[4] R. Pueyo Centelles, F. Freitag, R. Meseguer, and L. Navarro, "Beyond the star of stars: An introduction to multihop and mesh for LoRa and LoRaWAN," *IEEE Pervasive Computing*, vol. 20, no. 2, pp. 63–72, 2021.
[5] C. Ebi, F. Schaltegger, A. Rüst, and F. Blumensaat, "Synchronous LoRa mesh network to monitor processes in underground infrastructure," *IEEE Access*, vol. 7, pp. 57 663–57 677, 2019.
[6] "Meshtastic: Open source hiking, pilot, skiing and secure GPS mesh communicator," https://meshtastic.org/, accessed: 2022-03-03.
[7] R. Pueyo Centelles, "Towards LoRa mesh networks for the IoT," Ph.D. dissertation, Universitat Politècnica de Catalunya, Nov 2021.