Demonstration of concurrent application provision with LoRaMesher

Joan Miquel Solé
Computer Architecture Department
Universitat Politècnica de Catalunya
Barcelona, Spain
joan.miquel.sole@estudiantat.upc.edu

Felix Freitag
Computer Architecture Department
Universitat Politècnica de Catalunya
Barcelona, Spain
felix.freitag@upc.edu

Leandro Navarro
Computer Architecture Department
Universitat Politècnica de Catalunya
Barcelona, Spain
leandro.navarro@upc.edu

Mennan Selimi
Max van der Stoel Institute
South East European University
Tetovo, North Macedonia
m.selimi@seeu.edu.mk

***Abstract*—There is an increasing interest in extending traditional LoRaWAN-based IoT applications by LoRa mesh networks. The mesh technology introduces the capacity for multi-hop and node-to-node communication and new opportunities for IoT applications. This demonstration showcases the LoRaMesher library, which enables the concurrently provisioning of multiple applications within a LoRa mesh network. The paper demonstrates the library operating with real IoT boards. This includes showing the self-configuring of the nodes to build the LoRa mesh networks. Then, we will run concurrent applications to demonstrate to the audience the different communication patterns of distributed applications within a LoRa mesh network. Finally, we will review how the LoRaMesher library can be used. The implementation of LoRaMesher is open source.**

***Index Terms*—LoRa, embedded systems, mesh networks**

I. Introduction

Applications in Low Power Wide Area Networks (LPWANs) can use LoRa for the communication of data. Often, these applications implement the LoRaWAN architecture [1]. In LoRaWAN, LoRa end nodes send sensor values to a nearby gateway connected to the Internet. While LoRaWAN provides a downlink, it is not possible to have a symmetric communication between uplink and downlink. Thus, data transfer in LoRaWAN is typically mostly unidirectional from an end node to a service in the Internet, passing though a gateway.

LoRa mesh networks have been proposed recently [2], including routing protocols for LoRa mesh networks such as in [3]. Nodes can connect with each other and data can be sent over multiple hops to its destination [4]. Also, nodes can have the role of routers or focusing on running applications [5]. As shown in section IV, the IoT applications running in the demonstration can operation inside of the mesh and reach hosts in the Internet.

In this demo paper, we present the scenario where multiple distributed applications can run concurrently on Internet of Things (IoT) nodes within a LoRa mesh network and present the operation of these on real IoT boards.

II. Implementation

For this demo, we use the LoRaMesher implementation [6][1] that is publicly available. LoRaMesher leverages FreeRTOS. The implementation runs on boards with ESP32 microcontroller. An example of a board that can be used for LoRaMesher is the T-Beam. It uses the ESP32 microcontroller and an SX1276 LoRa radio (Figure 5).

The routing of packets in LoRaMesher uses a proactive distance-vector protocol. A node builds a routing table with the nodes in the network as illustrated in the OLED display shown in Figure 5.

The basic network service of LoRaMesher is the sending of LoRa packets to a destination over a routed network. LoRaMesher is extended by LoRaChat to facilitate running multiple applications in the network. LoRaChat is available in a public git repository[2].

III. LoRaMesher basics

For starting with LoRaMesher, the repository provides a set of examples[3] to show LoRaMesher operating on nodes. Each example can be installed following the steps of the corresponding `Readme.md`. In these examples, LoRaMesher is included with the default settings as a library in `main.cpp` that represents the application on the node.

A LoRaMesher node sends data and routing packets. They can be distinguished at a node by the packet header. Every node periodically broadcasts routing packets where the periodicity is given by the `HELLO_PACKETS_DELAY` value. Routing packets are not forwarded by the receiving nodes. A receiving node uses them to update its routing table. Data packets can be routed over multiple hops between any source and destination within the LoRa mesh network.

[1]https://github.com/LoRaMesher/LoRaMesher
[2]https://github.com/Jaimi5/LoRaChat
[3]https://github.com/LoRaMesher/LoRaMesher/tree/main/examples

LoRaMesher uses RadioLib[4], which allows managing the LoRa radio. The settings are configured in the `buildOptions.h` file (Figure 1). Besides the LoRa parameter settings, such as the Spreading Factor (SF), also LoRaMesher-specific parameters, e.g., `MAXPACKETSIZE`, can be configured.

```
#define LM_BAND 869.900F
#define LM_BANDWIDTH 125.0
#define LM_LORASF 9U
...
//Comment this line if crc for each packet should
    not be applied
#define LM_ADDCRC_PAYLOAD

//Maximum size of the routing table
#define RTMAXSIZE 256

//Maximum size of a packet
#define MAXPACKETSIZE 100
...
//Periodicity of routing packet sending in seconds
#define HELLO_PACKETS_DELAY 120
```

Figure 1. Code fragment of the `buildOptions.h` file of LoRaMesher that allows configuring LoRa- and LoRaMesher-specific parameters.

An Interrupt Service Routine (ISR) is defined that runs each time when a LoRa packet is received. If the packet that arrives at a node is a data packet, then that node checks its destination. If the node is the destination of the packet, a *ReceiveLoRaMessage* task for receiving LoRa messages processes the packet at the application level, i.e. in the `main.cpp` file. Figure 2 illustrates the implementation. The *ReceiveLoRaMessage* task handler is provided to LoRaMesher. The Receive Task is notified each time a data packet is received with that node being its destination.

```
void processReceivedPackets(void*) {
    for (;;) {
        /* When processReceivedPackets is notificed
            then enter blocking */
        ulTaskNotifyTake(pdPASS, portMAX_DELAY);

        //Loop to process all packets of the
            Received User Packets Queue
        while (radio.getReceivedQueueSize() > 0) {
            //Get an element of the Received User
                Packets Queue
            AppPacket<dataPacket>* packet = radio.
                getNextAppPacket<dataPacket>();

            printDataPacket(packet);

            //After processing the packet, it must
                be deleted to release the used
                memory
            radio.deletePacket(packet);
        }
    }
}
```

Figure 2. Example of receive function implementation for Receive task in `main.cpp` file.

[4]https://github.com/jgromes/RadioLib

Packet sending can be implemented as a periodic task in the `main.cpp` file. In the example shown in Figure 3, every time the task is executed, it will get the routing table and if there is one or more elements, it will send to the first one a dataCounter value.

```
void sendLoRaMessage(void*) {
    int dataTablePosition = 0;

    for (;;) {
        if (radio.routingTableSize() == 0) {
            vTaskDelay(120000 / portTICK_PERIOD_MS);
            continue;
        }
        if (radio.routingTableSize() <=
            dataTablePosition)
            dataTablePosition = 0;

        LM_LinkedList<RouteNode>* routingTableList =
             radio.routingTableList();

        uint16_t addr = (*routingTableList)[
            dataTablePosition]->networkNode.address;

        dataTablePosition++;

        //Packet is created and sent
        radio.createPacketAndSend(addr, helloPacket,
             1);

        //Data counter is incremented
        helloPacket->counter[0] = dataCounter++;

        //120 seconds wait before the next packet is
             sent
        vTaskDelay(120000 / portTICK_PERIOD_MS);
    }
}
```

Figure 3. Example of send function implementation for periodic Send task in `main.cpp` file.

IV. Demonstration

The demonstration is performed with three TTGO T-Beam boards (Figure 5). For this, the boards have been flashed with the LoRaMesher library integrated into the LoRaChat code. Hence, we apply the basic network service of LoRaMesher described in the previous section III in combination with the application management provided by LoRaChat.

The operational applications are shown with a deployment of three boards as illustrated in Figure 6. The three boards interconnect over LoRa with each other. The nodes DD88h and DF44h are connected only to the LoRa mesh network. The node DD40h is also connected to a Wifi access point. Therefore, it also acts as a gateway. In addition, a laptop is connected to the same Wifi network and runs a MQTT broker and MQTT clients.

Once the interconnection of the nodes is shown, we experiment with two application types in a concurrent operation. The nodes will run a monitoring application that periodically sends data to an MQTT broker, showing the reception of the data from each node by the MQTT subscriber client on the laptop. An application is run at each node within the LoRa

Figure 4. Demonstration of reception of data at the MQTT subscriber on the Internet from two applications on a LoRa node.

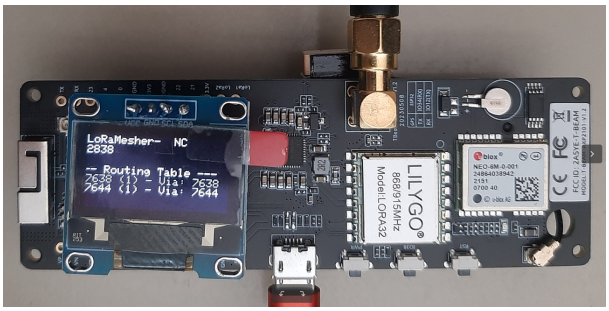

Figure 5. T-Beam boards with running LoRaMesher library.

mesh network that can be queried at from the laptop. Therefore, this second case demonstrates the possibility of making active queries to specific nodes. Both applications run concurrently at each node.

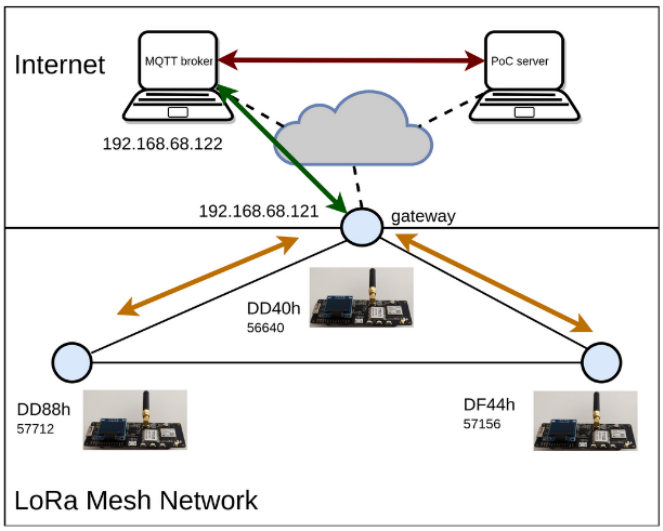

Figure 6. Demonstration setup with LoRa mesh network nodes and bidirectional communication from applications on LoRa nodes to Internet services over MQTT.

Figure 4 illustrates with a terminal screenshot how the MQTT subscriber on the laptop for a specific node received both the monitoring data and data from the query application. This experiment demonstrates LoRaChat's capability to run multiple applications over a LoRaMesher networking substrate.

Besides showing the experimentation, the demonstration will also provide the opportunity to review with practitioners the configuration details we used from LoRaMesher, our best practice experiences, and how we integrated the LoRaChat code into the monitoring and query applications.

Both LoRaMesher and LoRaChat are available in open git repositories. The implementation is operational for prototyping LoRa mesh network applications and be used with real IoT boards. The repositories include examples that enable the community to start using the implementation.

Acknowledgment

This work received support from the Spanish AEI (State Research Agency) by grant PDC2023-145809-I00/AEI/10.13039/501100011033, and by grant PID2023-146066OB-I00 (AEI 2023 Knowledge Generation Projects), being supported by the European Union NextGenerationEU, EU Recovery and Resilience Mechanism.

References

[1] C. Li and Z. Cao, "Lora networking techniques for large-scale and long-term iot: A down-to-top survey," *ACM Comput. Surv.*, vol. 55, no. 3, feb 2022. [Online]. Available: https://doi.org/10.1145/3494673

[2] A. W.-L. Wong, S. L. Goh, M. K. Hasan, and S. Fattah, "Multi-hop and mesh for lora networks: Recent advancements, issues, and recommended applications," *ACM Comput. Surv.*, vol. 56, no. 6, jan 2024. [Online]. Available: https://doi.org/10.1145/3638241

[3] R. P. Centelles, R. Meseguer, F. Freitag, R. B. Viñas, and L. Navarro, "A minimalistic distance-vector routing protocol for lora mesh networks," *IEEE Access*, pp. 1–1, 2024.

[4] "Meshtastic: Open source hiking, pilot, skiing and secure GPS mesh communicator," https://meshtastic.org/, accessed: 2024-08-06.

[5] A. Ghosh, S. Misra, V. Udutalapally, and D. Das, "Loraute: Routing messages in backhaul lora networks for underserved regions," *IEEE Internet of Things Journal*, vol. 10, no. 22, pp. 19 964–19 971, 2023.

[6] J. M. Solé, R. P. Centelles, F. Freitag, and R. Meseguer, "Implementation of a lora mesh library," *IEEE Access*, vol. 10, pp. 113 158–113 171, 2022.