Large and reliable data transfer service for LoRa mesh network applications

Joan Miquel Solé[a], Roger Pueyo Centelles[a], Felix Freitag[a], Roc Meseguer[a], Roger Baig[a]

[a] *Universitat Politècnica de Catalunya–BarcelonaTech (UPC), Barcelona, 08034, Spain*

Abstract

Recently, LoRa mesh networks have appeared as a communication technology for Internet of Things (IoT) devices. Through node-to-node communication, novel distributed IoT applications that extend beyond the capabilities of the LoRaWAN architecture can be enabled. However, current technologies for LoRa networks do not provide mechanisms for large and reliable data transfers between IoT nodes. This paper presents a service for such data transfers in LoRa mesh network applications, along with the protocol and formats used for inter-node communication. We explain the design choices and detail the implementation decisions to ensure that this service is practically usable. To this end, the service was integrated into the LoRaMesher library and is available as an open-source operational implementation. In experiments with ten real nodes and two network topologies, we observe that the service effectively achieves a large and reliable message delivery in an environment of concurrent transmissions and packet losses. In contrast, the cost of reliability for large data transfers is an increased number of messages and a higher delivery time. With the integration of the service into the LoRaMesher technology, developers now have a library that provides a reliable and large payload service for LoRa mesh network applications, eliminating the need to develop such capacity as a specific application-level solution.

Keywords: LoRa, mesh network, reliable communication, IoT routing

1. Introduction

LoRa is a popular wireless communication technology for IoT applications. The connection of remote IoT sensor nodes to the Internet is often

done using the LoRaWAN architecture [1]. In LoRaWAN, end nodes are always one hop away from at least one gateway node. An end node transmits data to a gateway using a LoRa radio packet in the format of the LoRaWAN Medium Access Control (MAC). Gateways continuously listen for LoRa messages from end nodes. Therefore, uplink messages (from end nodes to gateways) can be sent at any time. On the contrary, downlink messages from the gateway can typically be sent only during two short window frames after an end node's uplink message.

In LoRaWAN, a gateway can acknowledge the reception of a message to the end node through a downlink message. However, the gateway cannot generally send messages to an end node proactively. It has to use the reception window that is opened by the end node after sending an uplink message. This solution applies to end nodes at one hop from the gateway, but not to multi-hop topologies. LoRaWAN applies Cyclic Redundancy Check (CRC) that allows the receiver to assess whether the header and payload of a LoRa packet are correct. Still, this CRC field is available only in the uplink packet, not in the downlink packet. LoRaWAN does not support large data transfers. The payload of a LoRa packet is limited to 255 bytes, excluding those applications that use higher message sizes. Supporting larger data transfers would necessitate the implementation of mechanisms at higher levels of the networking stack, requiring code extensions on top of LoRaWAN at both the gateway and the end node. For traditional IoT monitoring applications, however, where only small payloads are sent and where the end nodes can be placed at a one-hop distance from gateways, LoraWAN's technical features are successfully used [2].

A different approach to connecting sensor nodes is LoRa mesh networks. These have been proposed to overcome some limitations of the LoRaWAN architecture, fostering functionally richer communication in the IoT layer [3]. With multiple hops, mesh networks can increase the distance between end nodes and gateways. In LoRaWAN mesh networks [4], some end nodes act as repeaters of packets from other end nodes and enable a larger geographic spread of the nodes, without needing to increase the number of gateways. However, the nodes are interconnected at the network level and not at the application level. The data destination is always the gateway, not another end node. Differently, with LoRa mesh networks, data can also be exchanged between applications on LoRa IoT nodes, such as presented in the works of Berto et al. [5], Meshtastic [6], and Pueyo et al. [7].

With the possibility of applications on IoT nodes of LoRa mesh networks,

new types of applications arise, including those with large payloads or that must not lose data. This creates the need for a network service that can send large and reliable messages from one node to another. One field in which such a service can be applied is recent Edge AI [8]. There, tiny machine learning is done in microcontroller boards that operate permanently while being interconnected. Such nodes differ from traditional sensor nodes, which only periodically wake up to read sensor values. When these remotely deployed machine-learning nodes use LoRa to transmit classification data [9], higher payloads exceeding a LoRa packet may arise, and the data must not be lost. Another kind of payload that can arise from Edge AI and that contains a large amount of data to be transmitted reliably to remote nodes is over-the-air updates of machine learning models [10]. However, the few LoRa mesh network solutions available today do not offer large payloads and reliable communication services.

In this paper, we present a service that provides large and reliable data transfer for applications on IoT nodes that are interconnected over a LoRa mesh network. The service is implemented on top of LoRaMesher's routing protocol. It is available through the open-source LoRaMesher library[1]. The large and reliable data transfer can be used between any two nodes in a multi-hop LoRa mesh network. A specific case available as well that resembles the LoRaWAN topology is that of two nodes of the LoRaMesh network, where one of the nodes is also a gateway to the Internet. The service implementation in LoRaMesher is operational. The application developer can now choose between the new large and reliable data transfer of the LoRaMesher library and the unreliable data transfer.

We study the service performance with an implementation and real IoT nodes in a LoRa mesh network. We experiment with different message sizes and network topologies to characterize the overhead and the benefits.

The main contributions of this paper are:

- Design considerations and choices for an operational large and reliable data transfer service in LoRa mesh networks.
- Integration of the service by implementing it into the LoRaMesher library.

[1] `https://github.com/LoRaMesher/LoRaMesher`

- Evaluation of the service and its operation by experimenting with large and reliable data transfers between real nodes of a LoRa mesh network.

2. Design

2.1. Design choices

We consider distributed applications in which different instances run on the IoT nodes from a LoRa mesh network. These applications require large and reliable data transfer between the different nodes. Figure 1 illustrates a LoRa mesh network with nodes that function as routers and others that also execute an application.

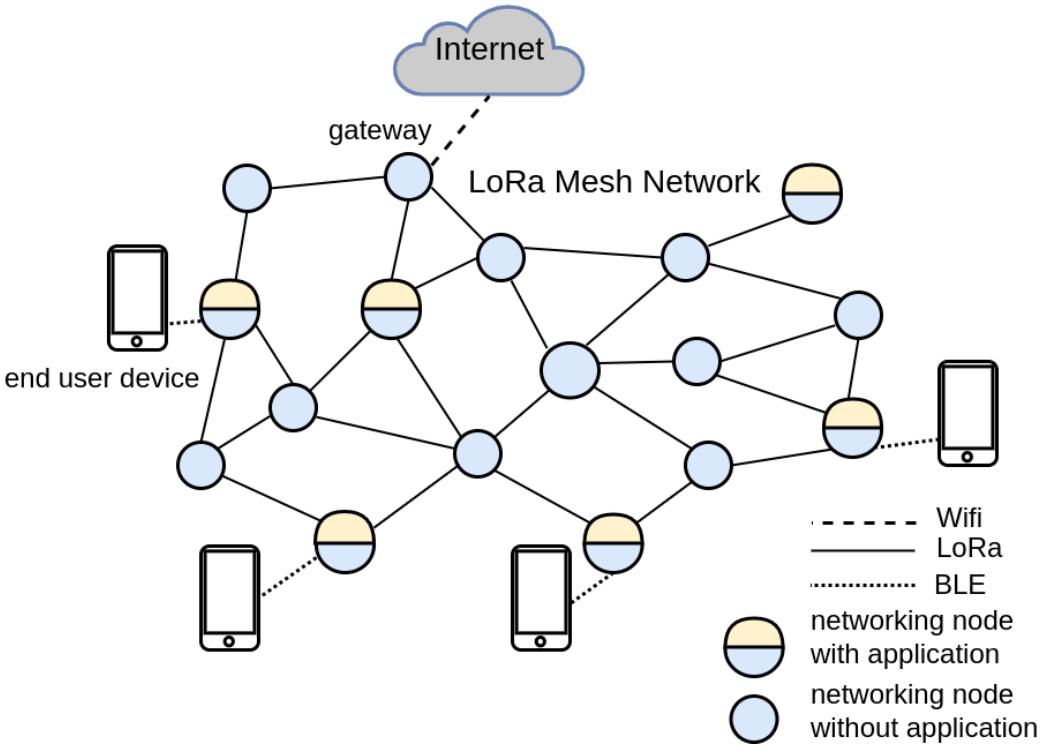

Figure 1: A LoRa mesh network with networking and application nodes.

An application-level connection between two nodes is identified by the source address, destination address, application identifier in the source, and application identifier in the destination. A large message from the application to be transmitted through the LoRa mesh network must be divided into

chunks of data to fit within the payload size of a LoRa packet. The data to be sent has to be stored in a queue.

To ensure reliable delivery, the data chunks to be sent are numbered. The source needs to be informed about the successful transmission of each chunk with a message containing reception acknowledgments issued by the destination. To this end, different acknowledgment strategies exist, including individual, cumulative, and selective. Each strategy has different network-overhead benefits depending on packet loss, traffic volume, and end-to-end latency.

The practical usability objective requires the solution to be implemented to minimize complexity and code footprint. Regarding LoRaMesher, the library is developed as a community effort, with design complexity aimed at broad understanding, so that it is easily extensible by others. Memory is often limited in microcontroller boards. Therefore, the protocol design has to take into account dynamic memory allocation behavior that may arise for the data chunks to be sent under different conditions.

We have chosen a design where two nodes can do bidirectional, full-duplex data transfer at the application level. At the radio layer, however, a node's LoRa transceiver operates in half-duplex mode. For achieving a reliable transfer, the communication design applies a stop-and-wait protocol based on the principles of Automatic Repeat reQuest (ARQ). In our implementation, acknowledgment messages include one chunk number. Each time a chunk is sent, the source starts a timer to wait for the acknowledgment of this specific chunk. The source obtains feedback by ACK and NAK packets. The source retransmits if a packet is lost during a handshake or upon receiving an NAK. The destination detects errors in the packet through CRC.

2.2. Packet structure

2.2.1. LoRa packet format

Figure 2 presents the structure of a regular LoRa packet and the various fields of a LoRaMesher packet. The *Message* field of the LoRaMesher packet can house three distinct types of packets: A data packet that can contain data from a small message whose payload fits into a single LoRa packet, adding a *NextHop* attribute, or data from a large message, for which the *NextHop* and fields to manage large payloads are added (Figure 3), a routing packet with the information detailed in Figure 4, or a control packet to manage large and reliable message transfer. A message is data that an application running

on a node sends to the LoRaMesher network service for transmission to a destination.

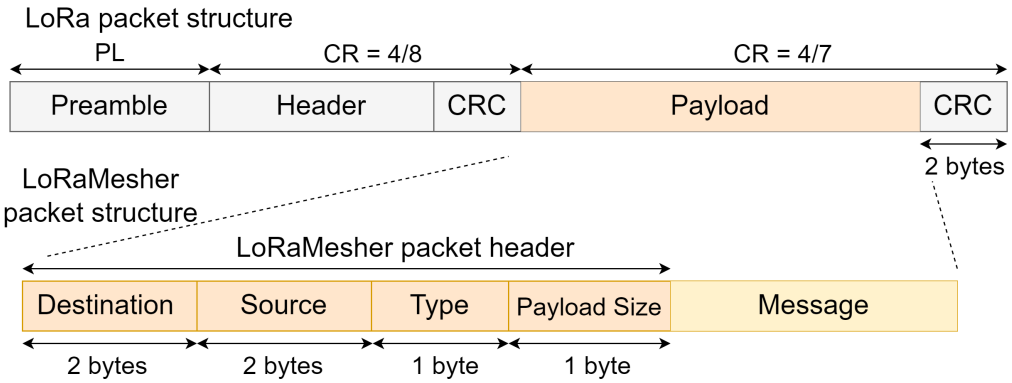

Figure 2: LoRa packet structure and LoRaMesher packet structure.

The size of a LoRa packet can be at most 255 B, but many regulations specify a lower number of bytes for the maximum payload. We adopt the recommendation of a maximum payload size of 222 B, which is accepted by most regulations worldwide. In the LoRa packet format, a 2-byte CRC field can be added, which calculates a checksum and allows the receiver to detect errors in the LoRa payload field. In case an error is detected, the packet is discarded. In the LoRaMesher library configuration, the LoRa CRC is optional and can be enabled.

Note that the specific LoRa Spreading Factor (SF) used affects the recommended payload size for the LoRa packet. For higher SFs that increase the Time on Air (ToA) of the packet and, hence, the likelihood of collisions, a smaller payload size is recommended.

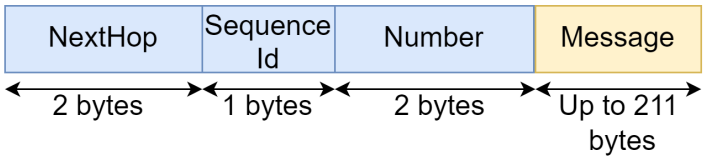

Figure 3: Packet structure for the large and reliable messages. This structure is contained in a LoRaMesher packet *Message* field represented in Figure 2.

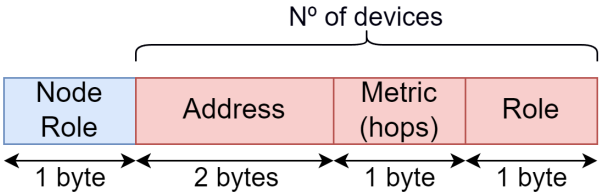

Figure 4: Packet structure for routing data. This structure is contained in a LoRaMesher packet *Message* field represented in Figure 2.

2.2.2. LoRaMesher packet format

A LoRaMesher packet is fitted inside the payload field of a LoRa packet (Figure 2). A LoRaMesher packet has a header of 6 bytes with the following fields: The *Destination* and *Source* fields correspond to the two-byte address of the two LoRaMesher nodes between which the data is sent. The *Payload Size* field of one byte corresponds to the length of all the packets, including the LoRaMesher packet header. The *Type* field of one byte provides information about the specific packet type, which in LoRaMesher are control, routing, and data packets (more details in Section 2.4).

The information about the type of packet is used at the network layer to manage the receiving and sending of LoRaMesher packets.

Control and data packets with a multi-hop destination use the extended LoRaMesher header (Figure 3). The fields used vary depending on the packet type. The extra header size, which can be up to 5 B, is determined by the type of LoRaMesher packet. These fields, however, are not used for routing packets (Figure 4). Unlike data and control packets, the receiving nodes do not forward routing packets.

The *NextHop* field, consisting of two bytes, is used in control and data packets. Whenever a router forwards a packet toward its destination, this field is updated with the address of the next node on the path. This update facilitates the packet's transmission towards the next hop and informs other nodes that they are not part of the ongoing communication.

The *Sequence Id* and *Number* fields are only used when sending in a large and reliable data transfer mode. Regarding packet header size, support for large and reliable data transfer adds three bytes to the header, compared with sending a single data packet without this feature.

The *Sequence Id* field, sized one byte, contains a unique identifier assigned by the source that allows both the source and the destination to identify

this communication. Should two applications hosted at the source initiate a concurrent data transfer toward the same destination, then the *Sequence Id* would allow distinguishing each piece of data. While in our current cases the number of concurrently running applications on a node is small and their number of data flows could be distinguished with a *Sequence Id* field sized less than one byte, the decision to use one byte was taken from an implementation point of view considering easier operational code and maintenance, where using the full-byte field simplifies parsing and reduces code complexity.

The *Number* field has two bytes. When the source starts a large message transmission with a synchronization packet for the handshake, the *Number* field contains the total number of packets to be sent in this data transfer. In subsequent packets from the source, the *Number* field contains the number corresponding to the current data chunk being sent, where this number is incremented by one for each data chunk sent. For each data packet received, the destination sends back a control message containing an acknowledgment with the number of the received data chunk. The *Number* field allows the destination to confirm each data packet of the transfer to the source. The question may arise about using fewer bits, e.g., one byte, for the *Number* field. With two bytes of the *Number* field, approximately 13.8 MB could be sent if the maximum payload size of 211 bytes for a LoRaMesher packet was used (Figure 3). Differently, if only one byte was available for the *Number* field, then approximately 53805 bytes could be sent using a 211-byte payload. However, when a higher SF is used, typically packet sizes smaller than 222 B are chosen. If we consider the case of a one-byte *Number* field and use, for instance, a packet size of 100 bytes where the maximum payload is 89 bytes after the 11-byte header, the amount of data that can be sent would reduce to 22695 bytes. Although 1 byte would suffice for the current use cases and the experiments reported in Section 4, we chose 2 bytes to have flexibility for future extensions and applications.

2.3. LoRaMesher routing packet with roles

Compared to the previous implementation of LoRaMesher [11], the structure of the routing packet was updated with a new parameter, which is the *Role* (Figure 4). The motivation for this new field is that a node in a LoRa mesh network can have a specific capacity different from that of other nodes. For example, a node can have an Internet connection. The other nodes should know such information if their application communicates with a host on the Internet.

The selected method of informing other nodes about each node's role propagates this information to the network through routing table exchanges. A node's role can indicate, for example, whether it is a gateway. Then, any node from its routing table can identify the gateway nodes within the LoRa mesh network. Suppose a node aims to send a message to the Internet. In that case, it can choose the address of the gateway node from the routing table and send a message to that destination without having searched for gateways previously.

The *Role* field is implemented as a one-byte bitset, allowing up to 8 different roles to be represented simultaneously. A developer can configure the roles of a node by calling functions that add or remove specific roles. This design offers flexible usage: for example, in diverse applications and devices, each node can customize the roles it advertises on the network, enabling customized behavior based on the device's functionalities. The one-byte bitset solution was chosen for flexibility and simplicity.

2.4. Usage of packet types in messages

The large and reliable message service requires data, routing, and control packets. Table 1 shows the packet flags that can be set. Data packets are identified by the types DATA_P and XL_DATA_P. They have a payload with application-level data. The HELLO_P type indicates a routing packet. Its payload consists of the source's routing table. A control packet has activated one of the flags ACK_P, LOST_P, NEED_ACK_P, and SYNC_P. These flags, hence control packets, are used only to manage large and reliable data transfers. A controlhas not have a payload. All information is encoded in the header.

2.5. Handshaking and data transfer

LoRaMesher's large and reliable data transfer service is connection-oriented. Before a source can send application data to a destination, they must exchange information about the transfer. For this, a two-way handshake is performed between the source and the destination. The source transmits a packet with the SYNC_P flag set and the number of packets sent. The destination transmits a packet with the ACK_P flag set in the *Type* field and the *Number* field set to 0. Having this information exchanged, both the source and the destination have a unique identification for that connection.

Figure 5 illustrates the reliable transfer of a large data message from the application level that is split into two chunks to be sent in two LoRaMesher

Packet flag	**Usage**	**Message type**
HELLO_P	Identifies a routing packet.	Routing message.
DATA_P	Identifies a data packet (without large and reliable data transfer).	Data message.
XL_DATA_P	Source sends chunks in a large and reliable message transfer.	Data message.
ACK_P	Destination acknowledges a chunk in a large and reliable data transfer.	Control message.
LOST_P	Source obtains a NACK. Destination notifies the number of the missing packet.	Control message.
NEED_ACK_P	Identifies a packet that requests the destination to send an ACK when received.	Control message.
SYNC_P	Source notifies the start of a large and reliable data transfer in a 2-way handshake.	Control message.

Table 1: Usage cases for flags.

data packets. The source sets the XL_DATA_P flag to indicate that the data message belongs to a large data transfer and the NEED_ACK_P flag. Also, the *Sequence Id* is set to a number that uniquely identifies this connection for the source and destination. The value of the *Number* field increases with every data packet for which reception is acknowledged. When the destination obtains the second and last packets, it knows from the exchange of information in the handshake that it is the last packet in this connection. From the *Payload Size* field, the destination also knows the size of this packet, which may be smaller than the previous packet in which data was sent with the maximum available payload. When all the expected data packets have been received, the destination sends a final ACK_P, signaling the termination of the connection. The source will also terminate the connection upon receipt of this packet. If the source does not receive this acknowledgment packet, it will continue to wait until all designated timeouts have occurred before ultimately terminating the connection.

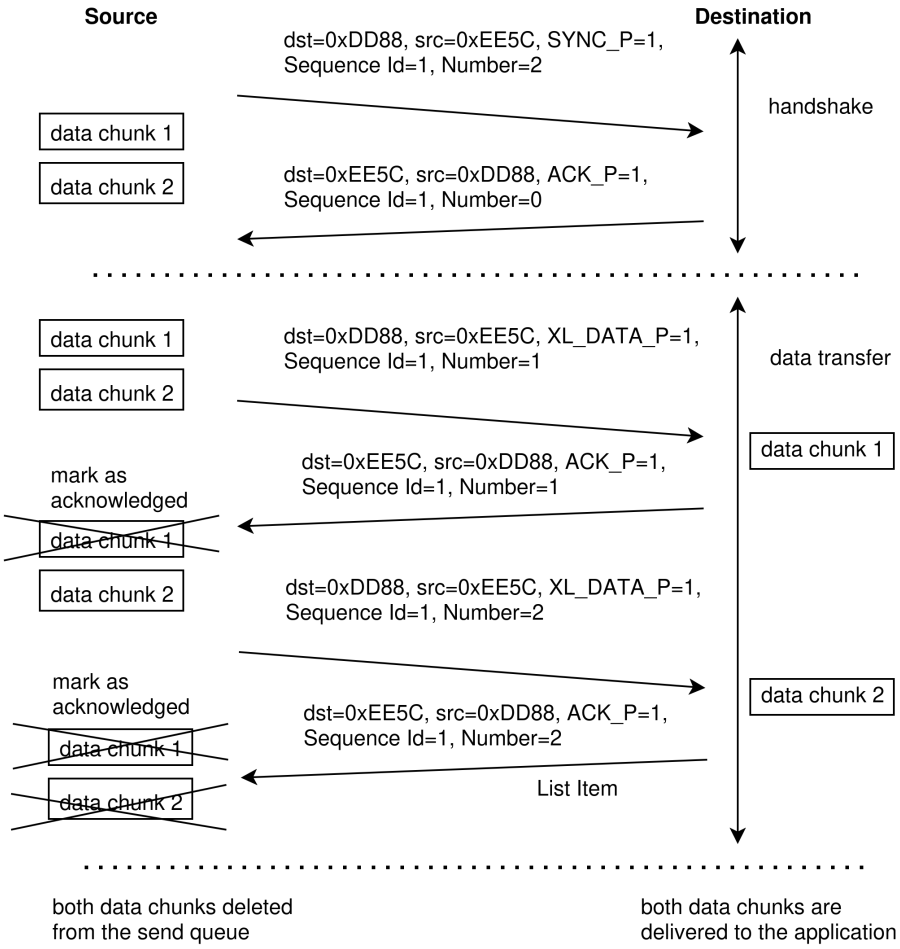

Figure 5: Example of handshaking and transferring two data chunks between source 0xEE5C and destination 0xDD88.

2.6. Operation of timers

A task manages the timeouts of all sequences. It is executed only when a connection state is initiated. Based on an estimated time, it checks for a timeout and increments the counter of timeouts for a connection if one occurs.

We have incorporated a configuration parameter into the LoRaMesher library called `MAX_TIMEOUTS`, which sets the maximum number of timeouts that can occur without receiving any packets from the other end of the con-

nection. After analyzing different values, we have concluded that setting the value of 10 timeouts for `MAX_TIMEOUTS` provides a suitable balance for establishing reliable connections. Therefore, if more than 10 consecutive timeouts occur, indicating a potential issue with the connection, the library terminates it. Then, the connection and all associated packets will be deleted.

When initiating a large and reliable message transfer, once the packets have been initialized, the source sends a packet with the SYNC_P set and initializes a timeout. Upon receiving a SYNC_P from the source, a new connection state will be established at the destination, and the timeout duration to identify a packet loss will be set accordingly. If a timeout occurs and the source has not received an acknowledgment for SYNC_P, it will resend SYNC_P, increasing the number of timeouts. If the timeout occurs at the destination, it will send a LOST_P control packet indicating the next expected packet. If no response is received, the destination will time out again and retransmit the LOST_P packet according to the configured MAX_TIMEOUTS value. We chose receiver-side timeout and LOST_P retransmission to avoid unnecessary full-packet retransmissions. This reduces traffic overhead in bandwidth-constrained environments, like LoRa, and allows the receiver to request only the missing packets.

Figure 6 illustrates the occurrence of two types of timeouts. The first type occurs during the initiation of a connection when attempting to use the SYNC_P control packet. The second type of timeout occurs when the destination fails to receive an expected packet and transmits a LOST_P control packet to indicate the packet it anticipates.

We implemented the Round Trip Time (RTT) calculation method described in RFC 6298 [12] to determine the timeout duration. This calculation is performed each time a specific connection sends a packet and subsequently receives a response.

From experimentation with different values of the minimum timeout duration, we configure the value to 20 seconds to ensure the desired level of reliability. The value can be set by configuration using the `MIN_TIMEOUT` parameter. The maximum timeout duration `MAX_TIMEOUTS` can also be set. Best practices suggest `MIN_TIMEOUT` + number of hops * 5 seconds in environments without duty-cycle restrictions. This calculus takes into account the potential delays introduced by the network.

During our initial experimentation, we encountered many timeouts in the system, resulting from too high variations in the RTT with values beyond the configured maximum duration. Therefore, under dynamic conditions,

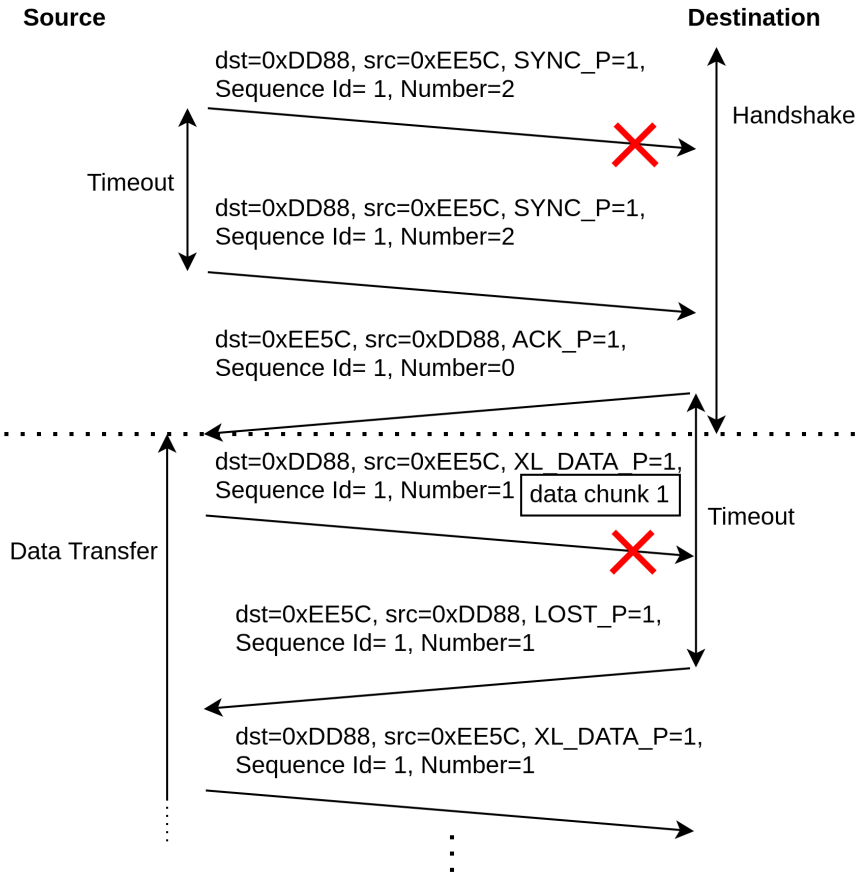

Figure 6: Example of a timeout when sending a SYNC_P and a timeout with a LOST_P between source 0xEE5C and destination 0xDD88.

the maximum timeout duration should be configured with a high value to reduce the number of times a connection is aborted. This finding highlights the need for further refinement and optimization of the RTT calculation to enhance the overall speed and efficiency of the protocol.

2.7. Control overhead analysis

Given a configurable `MAX_PACKET_SIZE` that ranges from 12 to 222 B, we analyze the control overhead for which we select a maximum packet size of 100 B. This leads to a maximum message size of $100 - 11 = 89$ B. The control overhead is then calculated as follows.

Let MS denote the message size, given by the number of bytes of a

message, and let N be the number of packets required to transmit the data. N can be calculated as:

$$N = \lceil \frac{MS}{\texttt{MAX\_PACKET\_SIZE} - 11 \text{ Bytes}} \rceil. \quad (1)$$

Each connection initiates with a SYNC packet, followed by an ACK packet for the handshake. Subsequently, an ACK packet is required for each of the N data packets. Let T represent the total number of packets that must be sent, including the SYNC and ACK packets, which is calculated as:

$$\mathrm{T} = (2\mathrm{N} + 2) \text{ Packets}. \quad (2)$$

Given that each packet has an 11-byte header, the control overhead can be calculated by:

$$\text{Control overhead} = 11\mathrm{T} \text{ Bytes}. \quad (3)$$

This control overhead is obtained under perfect conditions, i.e., without packet loss. In Figure 7, we show, for a given application message of 50, 500, and 10,000 bytes, how the control overhead changes depending on the `MAX_PACKET_SIZE`. It can be seen that the control overhead is higher for small messages, e.g., of 50 B, which fit into a single LoRa packet. In contrast, the difference in control overhead among different large message sizes is small. However, it is essential to note that packets with larger values of the `MAX_PACKET_SIZE` have less overhead and increase the likelihood of collisions. This trade-off between overhead and potential collision risks should be carefully considered in network design and operation.

3. Implementation

3.1. Radio initialization

LoRaMesher uses a configuration file to set the LoRa-specific parameters, including frequency, bandwidth, spreading factor, coding rate, sync word, power, and preamble length. The radio chip obtains these parameters to configure the sending and reception of LoRa messages. It must be pointed out that LoRa radios process only those LoRa packets with the sync word in their configuration. Therefore, the sync word enables the separation of LoRa networks across different private networks.

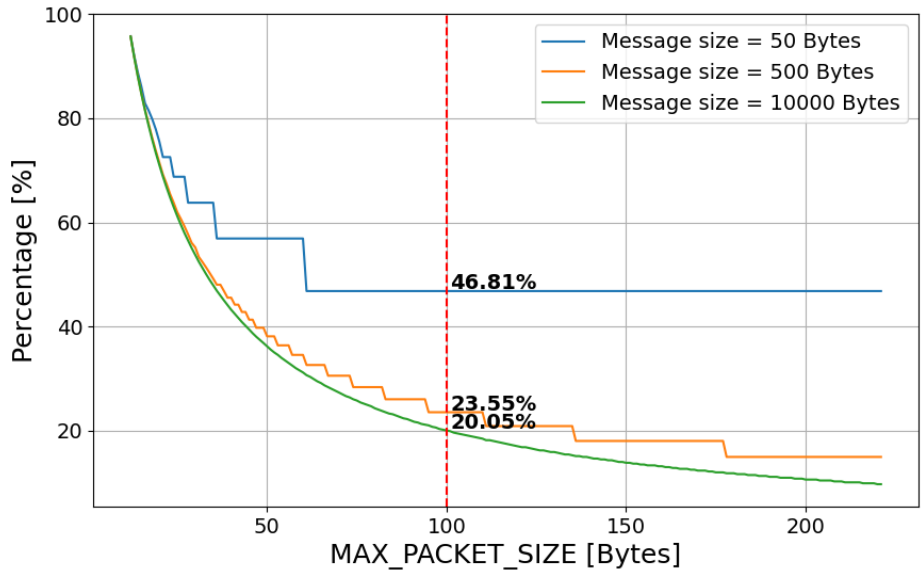

Figure 7: Control overhead versus `MAX_PACKET_SIZE` using 50, 500, and 10,000 bytes of message and a 100% delivery ratio.

3.2. Queues

The LoRaMesher library is implemented with tasks and queues (Figure 8). For the sending and receiving of single LoRa packets without establishing a connection between the sender and the receiver, the queues *Queue Receive Packets* (Q_RP), *Queue Send Packets* (Q_SP), and *Queue User Received Packets* (Q_URP) are used [11].

To support the transmission of large and reliable messages, two additional queues were added to the library: *Queue Waiting Sending Packets* (Q_WSP) and *Queue Waiting Receiving Packets* (Q_WRP).

For each connection established for large and reliable message transmission, the source and destination create a data object written in the queue at the corresponding queue to manage the data transfer of this connection. A node can send and receive data concurrently, creating an element in its Q_WSP and Q_WRP, respectively. The queues implemented at each node enable a node to manage multiple flows that can happen concurrently in the network. The data object for managing a connection contains the following fields:

- Sequence Id (1 byte)

- Source/Destination address (2 bytes)
- Number of packets in the sequence (2 bytes)
- Last Ack number (2 bytes)
- Number of timeouts (1 byte)
- Next timeout in milliseconds (4 bytes)
- A list that contains the packets to be sent/received.

The application notifies the library at the source node for a large and reliable data transfer. The function call includes the destination and the message to be sent. The library creates the object that contains the transfer information and the connection state. It then adds the packet list, which includes the necessary packets for the transfer, according to the `MAX_PACKET_SIZE`. Lastly, it adds the object inside the Q_WSP.

Once a packet of type `SYNC_P` is received at the destination node, an element is created for the Q_WRP. It contains the transfer information extracted from `SYNC_P` and the number of packets to be received. Once the complete data is received, the *User Receive Task* is notified about the availability of the data.

A new task, described in Subsection 2.6, is responsible for connection timeouts. It checks the Q_WRP and Q_WSP states and resends a packet, waits, or removes a connection.

3.3. Tasks in the application-library interaction

The application creates data messages that are delivered to the network layer and receives data messages from the network layer. The following data structures are used at the application layer:

- An instance of the LoRaMesher library, which enables the application layer to interact with the functions of the network layer. A library setup function is run to create the internal data structures and apply configuration parameters for initialization.
- A data packet containing the application message is to be sent to another node. It is delivered to the network layer.

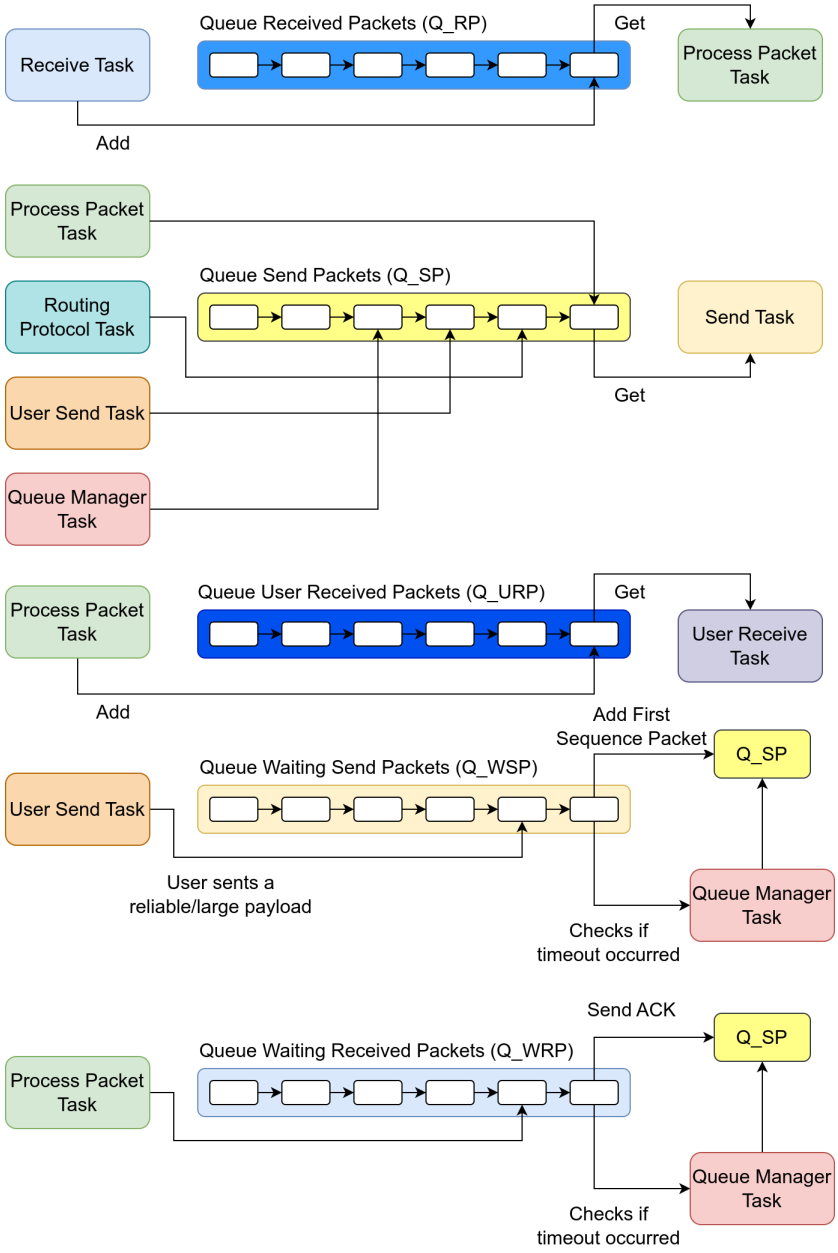

Figure 8: Queues of the LoRaMesher implementation.

- A data packet containing the application message received from another node. It is obtained from the network layer.

Two optional tasks can be created at the application layer: *Receive Task* and *Send Task*. The *Receive Task* activates when the network layer notifies it of an incoming message intended for the application. These messages can be accessed from the *Received App Packets Queue*. This queue, supplied by the network layer, is accessible to the application layer. Without the *Receive Task*, any packets directed to the application layer of this node would be discarded. The *Send Task* facilitates communication with the network layer. To transmit a data message, the application invokes the *radio.sendReliablePacket()* function, specifying both the destination address and the content of the message.

4. Evaluation

4.1. Experiment setup

We define four experiments for different configurations to test and verify the performance and implementation of the large and reliable message service in the LoRaMesher library (Table 2). Each experiment was repeated five times. We study performance under two source profiles on IoT nodes: one sends a sequence of 50-byte messages, and the other sends a sequence of 500-byte messages. The experiment with a message size of 50 bytes (experiments 1 and 3), which fits into a LoRa packet, tests the service's reliable transmission capacity. These two experiments are also run with the baseline communication service, corresponding to the default sending of LoRa packets without any control over message delivery. Experiments 2 and 4, which involve the 500-byte message that does not fit within a single LoRa packet, are conducted exclusively with the reliable and large message transfer service, as it necessitates the protocol's chunking of the message into data segments. This capacity is not available in the baseline service.

For the 50-byte message case, where the application at the node sends a total number of 200 messages, a node transmits a total of 10000 bytes of application data during the experiment. For the 500-byte message, a node sends 20 messages, resulting in 10000 bytes of application data transmitted from a node during the experiment. The application at a node sends its next message every 120 seconds. The chosen cases represent the scenario of an application regularly sending small and large payloads, respectively. We aim to understand the service's behavior under permanent operation by experimenting with transmitting several messages.

The experiments use the two topologies shown in Figures 9 and 10. We configure them with software. We test the sending of messages of 50 and 500 bytes in the two topologies. In the fully connected topology, the nine nodes A to I send messages to a gateway node. When sending 50-byte messages, a total of 90000 bytes are transmitted to the gateway node (10000 bytes from each of the nine nodes). When large messages of 500 bytes are sent, the nine nodes also send a total of 90000 bytes of application data. The gateway node is required to support multiple flows, while the client nodes may have to manage several open connections if packet losses for the sent data happen and retransmission is required. In the linear topology, messages are sent from node A at one end to node J at the opposite end, over nodes B to I. Only node A sends data, which in total is 10000 bytes. For the 500-byte message, a reliable and large message transfer is established between nodes A and J. At node A, the protocol splits the 500-byte message into smaller chunks of data that fit within a LoRa packet. The intermediate nodes B to I receive single data packets of the type XL_DATA_P, which indicates a reliable transfer. But not being the destination of the packet as indicated in the *NextHop* field (Figure 3), these nodes do not conduct a reliable transfer between each other. Reliable transfer is established end-to-end between the sending and destination nodes, but not between the nodes on the path to the destination.

The experiments are carried out with ten real IoT nodes, consisting of T-Beam and ESP32LoRa with ESP32 microcontroller boards, and SF 7, despite the SF being a configurable parameter in LoRaMesher, ranging from 7 to 12. The nodes were positioned close to each other, as illustrated in Figure 11, in an indoor environment to enable experimental control. There was no isolation from third-party traffic. For each experiment, we flashed all the nodes with the specific configuration. We chose to implement the geographic deployment of the nodes represented by the two topologies (Figure 9 and Figure 10; the red bidirectional edges represent end-to-end connections) by software. For this, we determined a set of visible nodes corresponding to the evaluated topology for each node. We leverage the internal *canReceivePacket()* function of LoRaMesher, which can be enabled by configuration and allows specifying which nodes can logically receive packets from other nodes. From an interference point of view, the nodes in the linear topology still face the traffic of all other nodes. When starting an experiment for the linear topology, upon receiving routing messages, a node constructs its routing table from the messages obtained from its visible nodes. In contrast, the routing messages

received from other nodes are ignored.

The library was configured with the LoRa-related settings shown in Table 3. The `MAX_PACKET_SIZE` parameter limits the LoRa packet to 100 bytes, thus forcing the protocol to split the 500-byte message into several data chunks that fit into the LoRa packet. The `MIN_TIMEOUT` parameter is the minimum time in seconds within which the sender expects a packet reception acknowledgment or the next packet of the sequence. The maximum time a message is expected to be received between two nodes integrates this value and a hop count. If that time has expired, then a timeout is counted. The `MAX_TIMEOUTS` parameter indicates how many timeouts are maximally accepted during a reliable and large payload transmission until the protocol aborts the transmission. The LoRa radio was configured with the 869.700 MHz permitted frequency according to the Harmonized European Standard ETSI EN 300 220-2 V3.2.2[2]. This specific band has no duty cycle requirement. Hence, the boards were operated without restriction.

Without duty-cycle restrictions, nodes can send LoRa packets at any time. Sending messages more frequently leads to more packet collisions compared to the situation with an enforced duty cycle, where the same amount of data is sent. Hence, not having a duty cycle is worse for successful data transmission than not having one. Still, it is also where the need for the reliability capacity of the service that we propose can be shown.

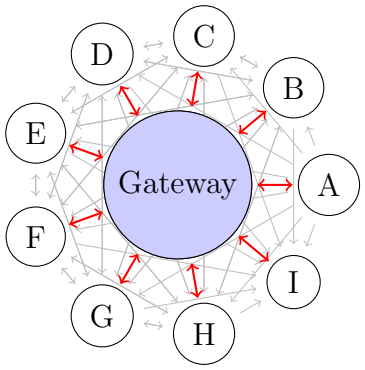

Figure 9: Fully connected topology with a central device emphasis, used in experiments 1 and 2.

[2]`https://www.etsi.org/deliver/etsi_en/300200_300299/30022002/03.02.02_20/en_30022002v030202ev.pdf`

Experiment [#]	**1**	**2**	**3**	**4**
Number of messages from node	200	20	200	20
Message Size	50 bytes	500 bytes	50 bytes	500 bytes
Topology	Fully connected (Figure 9)		Linear (Figure 10)	
Application layer connections	9		1	
Application data	90000 bytes		10000 bytes	
Communication service	BL and PR	PR	BL and PR	PR

Table 2: Settings of the experiments (BL: baseline service, PR: proposed service). Each experiment is repeated five times.

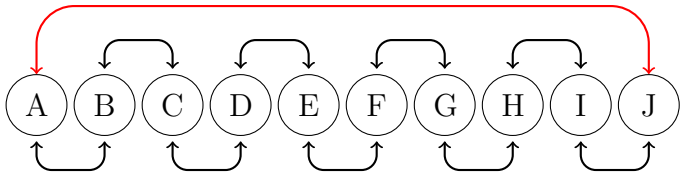

Figure 10: Linear topology with data communication from node A to node J, used in experiments 3 and 4.

4.2. Results

Ten nodes are started simultaneously for experimentation, inducing the worst-case scenario for Packet Delivery Ratio (PDR), in which each node periodically sends messages.

We added a 1-byte identifier attribute to the LoRaMesher packet header to conduct the experimental analysis. This attribute, along with the source address, allows us to identify a packet.

The message size and the header size determine the payload size. Our experiments use two message sizes: 50 and 500 bytes. Both message sizes share the same header size. To determine the complete packet size, we can combine the fields outlined in Figure 3 and those in Figure 2. Together, they contribute an 11-byte header.

Using Equation 2, we calculate the total number of packets needed to transfer the messages reliably. A 50-byte message needs four packets, while

Table 3: LoRa and LoRaMesher configuration parameters for the experiments.

Configuration parameter	**Value**
Band	869.900 MHz
Bandwidth	125.KHz
SpreadingFactor	7
CodingRate	4/7
PreambleLength	8 symbols
TransmitPower	6 dbm
DutyCycle	100 %
MAX_PACKET_SIZE	100 Bytes
HELLO PACKETS DELAY	120 seconds
DEFAULT TIMEOUT	5 * HELLO PACKETS DELAY
MIN TIMEOUT	20 seconds
MAX TIMEOUTS	10 times

a 500-byte message requires 14 packets.

4.2.1. End-to-End Message Delivery Ratio

We measure the end-to-end message delivery ratio, defined as the percentage of application-layer messages successfully received at the destination node from the source node. For chunked messages (experiments 2 and 4), a message is considered delivered only if all data chunks are received and successfully reassembled. For single packet messages (experiments 1 and 3), the message delivery ratio equals the packet delivery ratio.

Figure 12 compares the proposed protocol against the baseline implementation for experiments 1 and 3. The baseline protocol does not apply any reliability control and simply sends LoRa packets with 50 bytes of data to the destination. Since the 50-byte data fits within a single LoRa packet, these experiments specifically test the protocol's reliable transfer capacity without splitting the message.

As shown in Figure 12, the proposed service significantly improves the message delivery ratio compared to the baseline service, achieving 99.97% in experiment 1 and 100% in experiment 3. The 0.03% packet loss in experiment

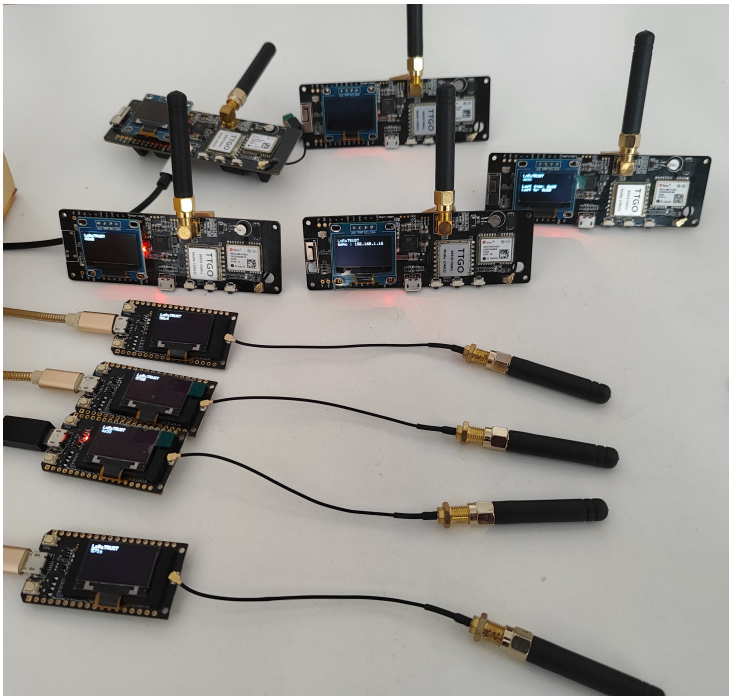

Figure 11: TTGO T-Beam and TTGO ESP32LoRa boards used for the experiments.

1 occurred when specific connections reached the maximum retransmission limit (`MAX_TIMEOUTS` = 10), causing packet transmission to be aborted. By using the default `MAX_TIMEOUTS` value of 10, packet losses in networks with a high number of collisions can still occur since the source can reach the maximum number of packet resending attempts for a message. In practice, depending on the specific reliability requirements, increasing the `MAX_TIMEOUTS` value may allow reaching a 100% message delivery ratio if the network is not persistently congested. Alternative approaches include reducing the sending frequency to lower network load or evaluating the effect of using a smaller packet size. Comparing experiments 1 and 3, packet losses in experiment 3 were lower because less application data was transferred during the same time in the network (see Table 2), resulting in reduced collision probability when a single source node is transmitting.

Experiments 2 and 4 evaluate the protocol's message delivery rates when transmitting 500-byte messages split into multiple 100-byte packets (not shown in Figure 12 as these experiments are not possible with the baseline protocol). Experiment 2, in which multiple devices transmitted simul-

taneously in a fully connected topology, achieved a message delivery ratio of 96.90% (95% CI: [94.36%, 99.44%]). This reduction compared to experiment 1 can be attributed to two factors: larger packet sizes (100 bytes vs 50 bytes) increase collision probability, and large messages require multiple successful packet transmissions per message, amplifying the effect of individual packet losses. The connections for sending 500-byte messages used six packets, which thereby increases the likelihood of retransmissions and collisions, as detailed in Section 4.2.2.

Among the experiments performed with the proposed protocol, experiment 4 achieved the lowest message delivery ratio (89.00%, 95% CI: [79.79%, 98.21%]), despite using the same packet and message sizes as experiment 2. Unlike experiment 2's fully connected topology with one hop between source and destination, experiment 4 required that both packets and their acknowledgments traverse nine hops without loss. Any loss along this multi-hop path triggers a timeout at the source node and increases the packet loss counter. The results indicate that 11% of messages failed because the packet losses exhausted the `MAX_TIMEOUTS` limit. These findings suggest that for large and reliable multi-hop transmissions, the `MAX_TIMEOUTS` parameter must be carefully selected, considering both the number of hops and the number of data packets into which each message is split.

4.2.2. Performance aspects

Figure 13 contrasts the behavior of various packet types across the four experiments. The messages sent in each experiment are displayed, highlighting the number of SYNC_P and their retransmissions, as well as the number of XL_DATA_P and their resends. Comparing results by packet size, we observe higher packet loss in experiments 2 and 4, where 500-byte messages are sent in 100-byte packets. Comparing experiments 1 and 2, experiment 2 shows a smaller absolute number of SYNC_P and resent SYNC_P packets, which corresponds to the smaller number of messages sent in experiment 2 (see Table 3). However, the proportion of the resent SYNC_P and XL_DATA_P is higher in experiment 2. The connections for sending the 500-byte message in experiment 2 used a packet size of 100 bytes, which resulted in more collisions and consequently increased the number of retransmitted XL_DATA_P packets. The higher resend rate of SYNC_P packets in experiment 2 (whose packet size is independent of the message size since it does not carry a payload) can likely be attributed to the increased number of resent XL_DATA_P data packets in the network, as indicated by the elevated XL_DATA_P re-

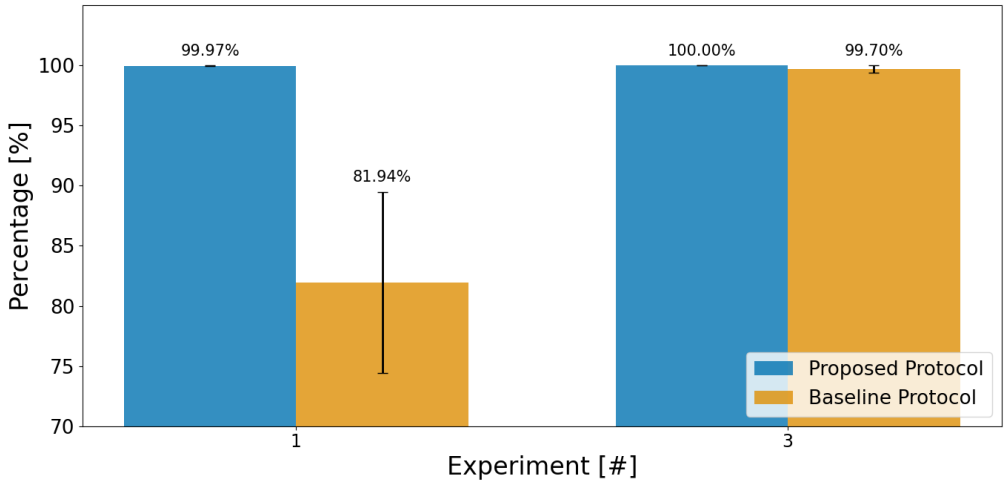

Figure 12: Comparison of end-to-end message delivery rates for 50-byte packets between the baseline and the proposed protocol. Fully connected topology left, linear topology right. Each experimental configuration was repeated 5 times. Error bars indicate 95% confidence intervals calculated using Student's t-distribution.

send value. That resending increased network traffic and likely contributed to higher SYNC_P packet loss.

Figure 14 shows the control overhead percentage for each experiment. The first bar of each experiment displays the theoretical value described in Section 2.7, followed by the experimental percentage overhead in the second bar and the percentage of packet loss in the final bar. Experiments 1 and 3 exhibit higher overhead because data is transmitted in smaller packets, thereby increasing the number of control packets. The second and fourth experiments, which transmit a 500-byte message in 100-byte packets, show a reduced overhead. However, experiments 2 and 4 have a higher packet loss than experiments 1 and 3 (50-byte packets). This illustrates a trade-off: using small packets increases the relative overhead, whereas larger packets used in the experiments are more susceptible to losses under the network conditions. The higher packet loss in experiments 2 and 4 results in more retransmissions and contributes to the higher experimental overhead compared to the theoretical values. Compared with experiment 2, experiment 4 shows a higher packet loss. Considering the message delivery rate of 96.90% in experiment 2 and 89.00% in experiment 4, as indicated in Section 4.2.1, the linear topology used in experiment 4 led to more packet losses and timeouts

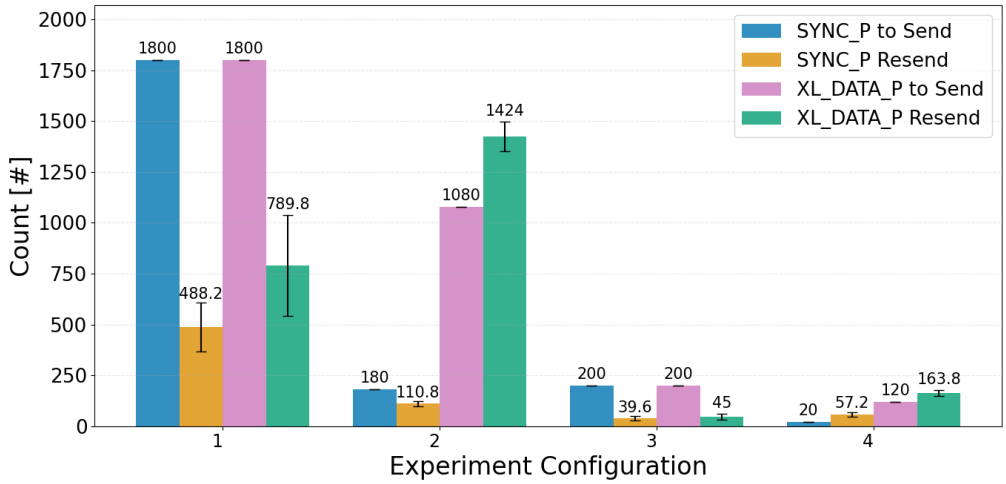

Figure 13: Number of resent SYNC packets, resent DATA packets, and total of messages and data packets per experiment. Each experimental configuration was repeated 5 times. Error bars indicate 95% confidence intervals calculated using Student's t-distribution.

and consequently more retransmissions by the source node. These additional retransmissions contribute to the greater difference between theoretical and experimental overhead in experiment 4. Finally, because the real experiments include packet loss, retransmissions, and timeouts, the experimentally measured overhead consistently exceeds the theoretical value.

4.2.3. RTT

To illustrate the Round-Trip Time (RTT), we conducted experiments in which two devices exchanged messages. Taking into account the parameters specified in Table 3 and the Maximum Time-on-Air (MaxToA) recorded as 236 ms, the delay for a single wait on each device translates to between 472 ms and 2088 ms. The range of these delays arises from two main factors: the packet's airtime and its duration on the device. A significant portion of the delay within the device is attributable to our implementation of the Carrier Sense Multiple Access / Collision Avoidance (CSMA/CA) algorithm, which is used to reduce contention in the wireless network by detecting LoRa preambles before sending. This algorithm sets a random wait time ranging between 1 ToA and $3 \times \text{ToA} + 100 \times$ millisecond routing table size. If the network is not congested, this time largely determines when the packet is stored on a device. Figure 15 shows them in two distinct green lines: one

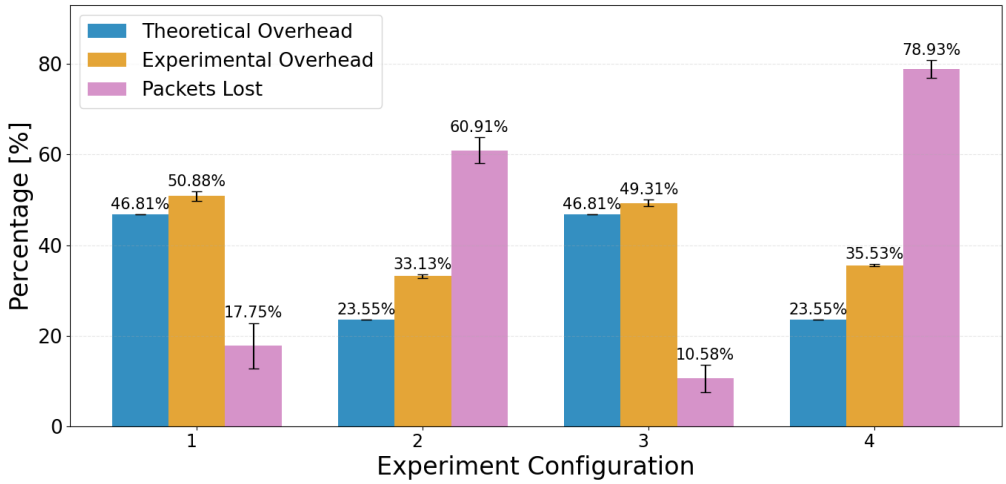

Figure 14: Control Overhead by experiment. Theoretical overhead value, experimental value, and percentage of packet loss. Each experimental configuration was repeated 5 times. Error bars indicate 95% confidence intervals calculated using Student's t-distribution.

at 472 ms and the other at 2088 ms. The lines are identified as *Doubled Delay Min* and *Doubled Delay Max*, respectively, and represent the theoretical minimum and maximum delays expected for a single-hop round trip. Furthermore, Figure 15 shows the actual RTT experienced in each device. Because our timeout configuration is not optimized, it results in longer RTTs than would be experienced with optimized timeout values.

Building on this analysis, we have also compiled the RTT measurements from our previous four experiments to provide a detailed overview of the system's performance trends. In Figure 16, we have marked the maximum doubled delay for nine hops with two red lines. Notably, the RTTs for experiments 1 and 2 show a broad distribution due to the higher number of devices transmitting simultaneously (9 devices sending data and one device sending acknowledgments), which increases collision probability. Nevertheless, these experiments have a lower median RTT than experiments 3 and 4, as they transmit messages over a single hop. In contrast, experiments 3 and 4 exhibit lower RTT variability because only one device sends data at a time. However, because the data is transmitted over nine hops, the RTTs in experiments 3 and 4 are higher, even though fewer collisions occur, thereby reducing RTT variance.

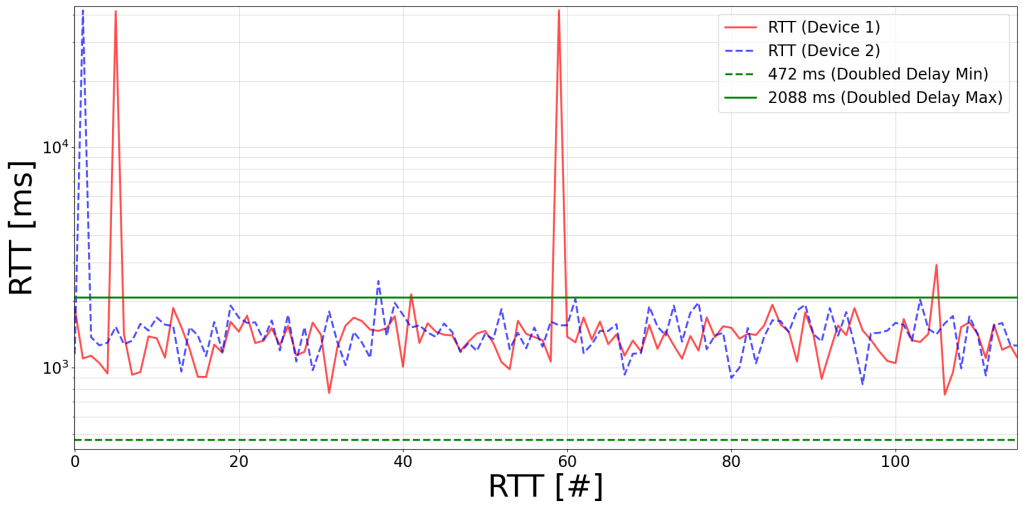

Figure 15: Round Trip Time in milliseconds for two devices, featuring the optimal and theoretical limits.

4.3. Sending large and reliable payloads under different conditions

We conduct a set of exploratory experiments to demonstrate the operation of the large and reliable payload transmission under different conditions. We use T-Beam boards that were introduced previously (Figure 11). The results of this section show tendencies of behavior and usability. For the experiments, the example code for large payloads of LoRaMesher's Git repository was used[3].

In the first experiment, we used a large message of 3500 bytes. The experiment is run with two nodes. One node sends the amount of data with different SFs to the other node. The base configuration of the experimental setup is that of Table 3. However, we changed the `MAX_PACKET_SIZE` parameter that limits the LoRa packet size from 100 bytes to 50 bytes. Smaller packet sizes of up to 51 bytes are recommended for higher SF due to their longer ToA. The LoRaMesher library splits large messages into chunks of data to fit the size of the LoRaMesher packet, as introduced in Section 2.1. To transmit 3500 bytes in 50-byte packets, 91 packets are required, including the SYNC packet. In experiments, we observed that the default `MAX_TIMEOUTS`

[3] `https://github.com/LoRaMesher/LoRaMesher/tree/main/examples/LargePayload`

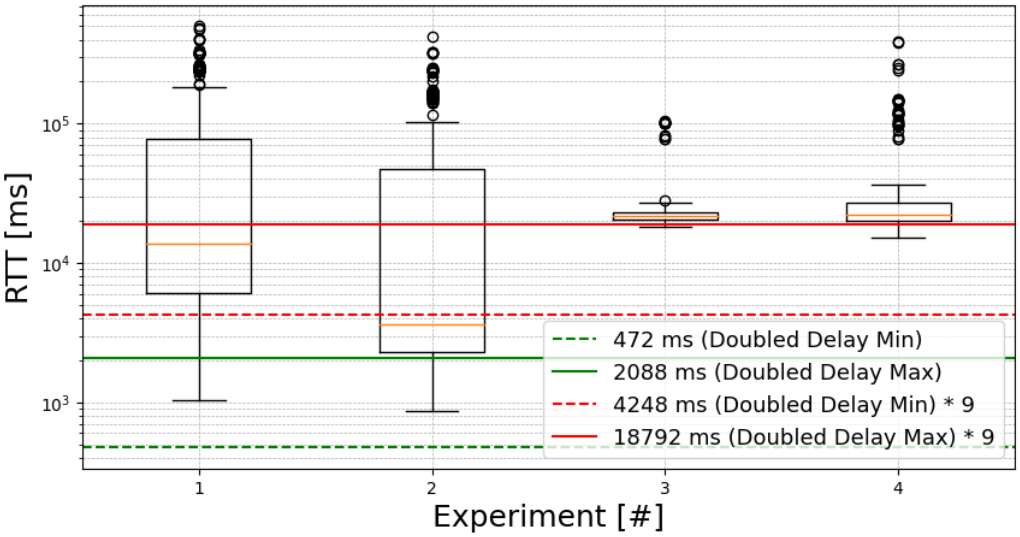

Figure 16: Round Trip Time of a packet by the experiments.

value of 10 was too low for SF 12; thus, we increased it to 20. The transmission time is measured from the time the source sends the first packet of the reliable payload to the time it receives the acknowledgment for the final packet.

Table 4 presents the first experiment's results. The message transmission time increases with higher SF. This is because of the increasing ToA of higher SFs. The results demonstrate the LoRaMesher library's usability for extensive and reliable payload transmission with different spreading factors.

Table 4: Effect of the SF on the message transmission time for a reliable transfer of 3500 bytes from one node to the other, with a `MAX_PACKET_SIZE` of 50 bytes.

Spreading factor	**Message transmission time [ms]**
7	117029
8	171820
9	291100
10	534154
11	1039283
12	1946189

In a second experiment, reliable bidirectional message transfer is conducted between two nodes. Both nodes are started roughly at the same time. The nodes detect one another through the exchange of routing packets. Then, each node sends a 3500-byte message to the other node.

Table 5 shows the results of the second experiment. Compared with the first experiment, the message transmission time is much higher when both nodes aim to send their message at the same time. Due to the similar start times of both nodes, their behavior is initially the same, leading to both nodes aiming to access the medium to send packets simultaneously. LoRaMesher integrates a LoRa preamble detection before sending and introduces a random delay mechanism to separate potentially synchronized node sending times. It can be seen that, finally, the payloads sent from both nodes are received at their destination. Besides the successful message transmission, the result demonstrates the operation of the queues in LoRaMesher used by the different tasks involved in sending and receiving packets at a node (Figure 8).

Table 5: Bidirectional reliable transfer of 3500 bytes between two nodes, with a `MAX_PACKET_SIZE` of 50 bytes.

Spreading factor	**Message transmission time node 1→node 2 [ms]**	**Message transmission time node 2→node 1 [ms]**
7	2519134	2709354

In a third experiment, the payload size is increased to 10080 bytes. The `MAX_PACKET_SIZE` is set to 100 bytes, and SF 7 is used. To transmit 10080 bytes, 115 packets, including the SYNC packet, are created. The payload is sent from one node to the other.

Table 6 shows the time that was needed to transmit the message from one node to the other. The results demonstrate the feasibility of transmitting a large message within a single connection. However, timeouts are tracked per connection, increasing the likelihood of reaching the `MAX_TIMEOUTS` value, which results in connection termination and the loss of packets already received at the destination. To prevent this, either the `MAX_TIMEOUTS` value should be customized, or the large message could be divided at the application level into smaller chunks, each submitted separately for reliable transmission to the library. The experimental setup introduced in Section 4.1 applies a strategy where a node sends 10000 bytes, but the LoRaMesher

library presents the data in 500-byte chunks for reliable transmission.

Table 6: Message transmission time for a reliable transfer of 10080 bytes from one node to the other, with a `MAX_PACKET_SIZE` of 100 bytes.

Spreading factor	**Message transmission time [ms]**
7	214301

In a fourth experiment, we ran four LoRaMesher nodes for 24 hours. The LoRa configuration is that of Table 3. The nodes are connected in the LoRa mesh network with a one-hop distance. One of the four nodes also has a WiFi connection and is used as a gateway. The other three nodes send, each with a different periodicity, a large message of 1014 bytes to the gateway node. When a message is received from the other nodes, the gateway publishes it (and its own message according to the configured periodicity) to an MQTT broker. The message contains a message counter in the *messageId* variable that increases with every message sent from a node. For every 1014-byte message sent from the three nodes to the gateway, the large and reliable message implementation of LoRaMesher is performed. The gateway node that runs the same code as the other nodes also sends a large message. Since the gateway detects the Internet connection and the MQTT broker, its message does not use the LoRa mesh network, but is sent directly over Wifi to the MQTT broker. For this experiment, we customized the LoRaChat implementation's monitor application[4] to send a specific 1014-byte payload and to collect performance information via MQTT subscription, integrating LoRaMesher as a library.

Figure 17 shows the evolution of the messageId value for the four nodes. Given that the messageId is 1 byte, after sending 256 messages, its value resets to 0. For the two nodes with id 46300 and id 56636 that send messages at a higher periodicity, this restart from 0 is performed during the 24-hour experiment. The increase in the messageId of nodes 50884 and 59304 is slower because their message sending periodicity is higher. At time 20h, node 46300's messageId increased not entirely linearly. This means that some messages of node 46300 arrived at the gateway out of order. Due to packet losses in a previous message, the reliability service had to resend packets of this message. In contrast, a later message sent from the same

[4] `https://github.com/Jaimi5/LoRaChat/`

node, which may have faced fewer losses, arrived faster at the gateway. An explanation for the observed out-of-order message arrival at node 46300 is that it sends messages more frequently, every 3 minutes, than the other nodes. Hence, sending an earlier message that has not yet been delivered is more likely to overlap with the following message. To cope with this situation, the SequenceId field of the LoRaMesher packet header allows the destination to distinguish the packets of the two large messages sent concurrently from the same source (Figure 3). The experiment demonstrates that the gateway node can handle large messages periodically transferred from three other nodes during a 24-hour operation. The results also show a stable implementation of the large and reliable payload service in LoRaMesher, which had to handle hundreds of large messages in this experiment.

Figure 17: Large messages of 1014 bytes are sent from three nodes in the LoRa mesh network (id 46300, 50884, 59304) for 24 hours to a gateway node (id 56636). Sending periodicity: node 46300 every 3 minutes, node 50884 every 20 minutes, node 56636 every 5 minutes, node 59304 every 10 minutes. The MessageId values are obtained from a subscription to the MQTT broker.

4.4. Suitability of the communication service

LoRaMesher's baseline communication service, i.e., a single-packet transmission to any node in the LoRa mesh network, may be suitable for applications whose data fits within a single LoRa packet and when packet losses are acceptable. Examples can be environmental monitoring applications, where the loss of samples may be tolerated. Differently, the service for reliable and

large message transfer can be chosen for applications where reliable message delivery is critical. Examples include applications with larger and more complex data objects. The shift from traditional low-capacity IoT nodes that monitor sensor values to more powerful nodes performing tiny machine learning [8], will lead to a change in the required network service, particularly when classification data and the number of model parameters is huge and packet losses cannot be tolerated.

The service implemented in the LoRaMesher library provides the application code for sending large messages without handling packet control at the application level. As the performance analysis shows, this reliability is achieved at the cost of increased message exchange and longer data delivery times.

5. Related Work

In recent years, several proposals have been made regarding multi-hop, mesh, and routing for LoRa and LoRaWAN. They have been thoroughly classified and analyzed by researchers from different points of view: taking the application scenarios into account [3], focusing on the LoRa [13], the LoRaWAN architecture [4], including a recent specification of relay nodes in LoRaWAN [14], or specific implementation aspects like topology and routing [15]. These works are heterogeneous in maturity and technology readiness, ranging from theoretical contributions to experimentally validated proposals in testbeds or real-world deployments.

5.1. Multi-hop and mesh

Numerous proposals exist for multi-hop networks employing LoRa technology, which operate independently of the LoRaWAN architecture. These initiatives establish tree and mesh topologies to construct increasingly decentralized and adaptable networks by using diverse strategies such as routing, Time-Division Multiple Access (TDMA), clustering techniques, and others. In this section, we conduct an in-depth examination of the proposals most pertinent to our research.

Sartori et al. [16] addressed the gateways' coverage issue and designed Routing over Low-Power and Lossy Networks (RPL) [17] + LoRA MAC (RLMAC), a MAC-layer protocol that enables RPL multi-hop communications based on LoRa. They argue that the star topology is convenient for ease of deployment and, from a business perspective, though multi-hop could

mitigate congestion issues. Moreover, multi-hop could be the only option for covering vast areas with few base stations and could increase throughput or reduce time-on-air by using faster SFs. The authors designed a multi-hop solution for single-channel LoRa nodes. They implemented the algorithms to bootstrap and operate a network using RPL by combining a slow reception loop with fast transmission loops, to ensure that nodes can receive messages from any neighbor using any SFs. With the first experimental validations, the authors found that when two nodes choose the same SF to transmit towards a third node, collisions are frequent. This was solved by adding an extra delay, randomly generated from a brief RSSI measurement. The authors state that their solution must be tested with a simulation tool for performance and power consumption before using it in a real-life deployment with a large number of nodes.

Duong and Kim [18] designed and implemented a protocol to enable multi-hop communication in LoRa networks spanning extensive distances. Their solution caters to deployments where every monitoring node is positioned along a linear path. The nodes forward data in the $leaf \rightarrow sink$ direction. Devices synchronize and wake at specific intervals to receive data packets from neighboring nodes, which they can combine with their own data packets and relay further along the line.

Similarly, Abrardo and Pozzebon [19] devised a multi-hop LoRa linear network for subterranean environments, optimizing nodes' sleep/wake cycles to reduce battery consumption. However, the proposals by Duong and Kim [18] and Abrardo and Pozzebon [19] are only suitable for networks with a linear static topology, where all the collected data are targeted towards a single sink node.

Lee and Ke [20] devised and implemented a LoRa mesh networking system to enable indoor nodes to communicate with network servers without requiring additional gateways. Their design involves a data sink broadcasting beacons to invite nodes to join the network. New nodes that hear packets from the data sink, or other nodes, can also join the network, choosing a suitable parent based on multiple factors (namely, Received Signal Strength Indicator (RSSI), hop count). The data sink polls child nodes to request their data and holds a complete view of the network topology, which it can modify based on its comprehensive information. The authors validated their research using a real-world experiment with only six nodes, but they did not comprehensively evaluate system performance.

Mai and Kim [21] proposed a collision-free multi-hop LoRa network pro-

tocol with low latency. The sink node exchanges packets with other nodes to construct a tree topology and allocate a timeslot and channel to each link in its network. This way, communication between the leaf and parent nodes is collision-free with the neighbors, as nodes transmit at their own pace during their assigned timeslot.

Leonardi et al. [22] present MRT-LoRa, a multi-hop communication protocol designed for LoRa networks. This protocol employs a tree topology structured around levels (a different SF per level) and TDMA to achieve the multi-hop objective. To validate their proposal, the authors used the OMNeT++ simulator. Although the simulator utilizes LoRa features, including interference, packet collision, and propagation, it lacks the necessary details to facilitate a practical implementation and enable experiments.

Zhu et al. [23] enhanced the capacity of a multi-hop LoRa network by diverting traffic into multiple subnetworks with varying SFs. This clustering technique results in a multidimensional access network, where each subnetwork is anchored by a sink node with a specific SF. This makes packet transmission in parallel with multiple SFs feasible. Their solution requires a coordinated effort for the clustering decision-making tasks. The authors validated their proposal using a non-realistic simulator without LoRa features, such as interference, packet collisions, and propagation.

Prade et al. [24] introduced a communication architecture employing a multi-radio and multi-hop strategy to extend coverage and improve services for extensive IoT deployments in rural regions. The authors outlined the conceptual design and executed a hardware prototype to validate their approach. Despite its strengths, the paper's focus on architecture alone limits its practical application, as a comprehensive understanding requires delving into network topology, routing, and data transfer specifics.

Berto et al. [5] introduced a preliminary study to enable LoRa-based mesh networks. The prototype of this network is based on the RadioHead library. A message's header includes information necessary for routing and forwarding. This is achieved through automatic route discovery, facilitated by special route discovery request broadcast packets. The source node generates these packets, which are conveyed to the destination node using a reactive routing approach. The experiments are conducted using a payload of 240 bytes, which is not larger than a single LoRa packet. To validate their research, the authors presented a hardware/software prototype that employed low-power-consumption devices. The experimental results presented are based on a limited setup involving only three nodes and do not comprehensively

evaluate system performance.

In conclusion, the proposals for multi-hop networks using LoRa predominantly focus on network topology, routing, clustering techniques, multi-radio capabilities, and slot allocation. Nonetheless, they have not specifically tackled the challenge of ensuring dependable data transmission or the efficient management of substantial data volumes. Consequently, the academic literature lacks comprehensive solutions for efficiently tackling communication reliability and handling large messages within LoRa-based multi-hop networks.

5.2. Software libraries and frameworks

Regarding practical implementations, very few readily available commercial or open-source products offer LoRa multi-hop and mesh capabilities.

RadioHead, developed by McCauley [25], is a library for sending and receiving messages through various IoT communication technologies, including LoRa. The library supports drivers for several radio transceivers. The network addresses in RadioHead are 8 bits long, permitting a maximum of 255 nodes. An additional address is reserved for broadcasting purposes.

Similarities and differences between RadioHead and LoRaMesher [11], considering the extension developed in this paper, are: Both libraries support multi-hop mesh delivery with routing; however, while RadioHead employs reactive routing, LoRaMesher uses proactive routing. Both libraries offer a certain level of reliability. The RadioHeadMesh class achieves reliable hop-to-hop LoRa packet delivery through hop-to-hop acknowledgments, whereas LoRaMesher relies on end-to-end acknowledgments. Therefore, the message sender receives confirmation of the reception at the destination. The RadioHeadMesh class lacks message queuing, allowing it to handle only one LoRa packet at a time. In contrast, LoRaMesher incorporates queues, making it capable of handling multiple packets. By leveraging queues and end-to-end reliability, LoRaMesher enables the transmission of large messages by splitting them into multiple LoRa packets and reassembling them at the destination.

Meshtastic is an open-source text messaging application that integrates a multi-hop LoRa mesh network for interconnecting the nodes [6]. Technically, Meshtastic allows node-to-node communication through LoRa packets as in LoRaMesher. However, LoRaMesher implements a routing protocol where Meshtastic delivers messages through broadcast and repeat. This may congest the network, while in LoRaMesher the routing of messages produces less

traffic in the nodes that are not on the path toward the destination. Meshtastic does not provide a large and reliable message payload network service for the extension developed in this paper. Finally, Meshtastic, rather than being a library, is a concrete application with an integrated LoRa mesh network layer and hence has a narrower technical and social focus. LoRaMesher is a library that provides a networking layer for different types of applications.

5.3. Reliable messaging

Various proposals aim to achieve reliable messaging using LoRa technology. Furthermore, these approaches to enhancing reliability differ conceptually.

Haubro et al. [26] harness the dependable performance of Time-Slotted Channel Hopping (TSCH) and the long-range capabilities of LoRa to establish a reliable multi-hop solution. The focus of reliability in this proposal is centered on reducing interferences. This approach aims to increase the probability of successful delivery of LoRa packets to their destinations, but it does not guarantee delivery or notify the source upon successful reception.

Sakamoto et al. [27] designed and implemented TCP/IP over LoRa, extending IP2LoRa. In their proposal, they leverage the reliability mechanisms of TCP/IP. While this approach achieves a level of reliability similar to TCP/IP, it is limited to a single hop, and it is unclear how to deploy it in a multi-hop or mesh LoRa network. Although the proposal is exciting, it is limited to data transmission between two nodes.

Manzoni et al. [28] proposed a holistic solution to manage the LoRa network, enabling synchronization, routing, and reliability. The reliability is based on acknowledgments and retransmissions to ensure packets are delivered correctly. Their approach achieves reliability within a single hop, while our proposal focuses on end-to-end reliability and includes sender notifications upon successful reception.

Stann et al. [29] proposed RMST (Reliable Multi-Segment Transport) for reliability in the transport layer of sensor networks. The work raises the question of at which layer, hop-to-hop of the MAC layer, end-to-end of the transport layer, or at the application layer, the protocol for reliability should be placed. RMST was designed for use when data communication is based on direct diffusion, a publish/subscribe multipoint-to-multipoint communication. As subscribers or publishers, nodes with the same interests form a group. Simulations evaluate the RMST layer. From the results, the authors

conclude that combining a NACK-based transport-layer protocol with selective ARQ at the MAC layer is a suitable solution for diffusion networks with high error rates.

Ghosh et al. [30] proposed LoRaute, where a LoRa mesh network is conceived as a backhaul network with hosts and mobile nodes. Routing nodes request their neighbor nodes for a list of their neighboring nodes. The obtained list forms the routing table of that node. The protocol implements an optional acknowledgment-based message-sending retry. While the experiments were conducted with real nodes, the scale for evaluating the routing capacity was limited to at most seven nodes and specific scenarios, which restricts the generalizability of the obtained results.

5.4. Large payloads

A few proposals aim to enable the transmission of messages and data larger than the maximum LoRa packet size.

Muñoz et al. [31] utilized the IETF standardized compression and fragmentation scheme known as Static Context Header Compression and Fragmentation (SCHC). This scheme can compress and fragment IPv6 and UDP headers for LoRaWAN, enabling IP-based communications on resource-constrained end devices.

The LoRaWAN standard allows for the fragmentation of data blocks that exceed the maximum size of a LoRa packet [32]. However, the technique is limited to LoRaWAN and therefore unsuitable for multi-hop or mesh LoRa networks.

Arratia et al. [33] introduced a multi-hop monitoring system based on LoRa technology. Their unicast protocol utilizes an ARQ mechanism, similar to the approach presented in [34] by the same authors. Unlike the LoRaMesher library, it lacks routing mechanisms for directing LoRa packets. Instead, it employs the forwarding technique, which is not scalable for large networks or topologies with loops.

Arratia et al. [35] extend the previous work in [33]. The authors present a LoRa network-based solution to solve the data transmission for buoy sensor nodes in a lagoon. A protocol design called AlLoRa, based on a stop-and-wait ARQ scheme, is presented that forwards messages in a multi-hop network without a routing protocol. The evaluation focused on practical aspects of the IoT application case, including throughput across different SFs and energy consumption. In AlLoRa, only the source and sink nodes participate in the transmission of the chunks, and it is closed with a final ACK of the totality

of the data. The source node initiates the transmission in our proposal, and the ACK is per chunk.

In light of these previous works, we conclude that our proposed service is superior to the mechanisms available in the literature in terms of being a real-world implementation of a mesh network over LoRa that incorporates routing, reliability, and the capacity for large data delivery.

6. Discussion

The service we designed for large and reliable data transfer is for amounts of data that are determined at the initialization of a data transfer, i.e., when the application informs the network about the data to be transmitted. The sequence numbers are calculated at the source and communicated to the destination. This design is a solution for the case where the amount of data is known, but it does not allow sending continuously generated streams of an unknown data size. The data chunks the destination obtains are only delivered to its application process once the transfer has been completed. This avoids handling the partial delivery of data at the application level.

The duration of the data transfer with the implemented large and reliable message service is influenced by the stop-and-wait design. In LoRa mesh networks with large diameters, the time for sending each data chunk and the acknowledgment can increase. An alternative to the stop-and-wait protocol is the pipelining of packets, which allows the source to send more than one packet before receiving an acknowledgment from the destination, which may also be cumulative. The TCP protocol uses such a technique, satisfying the requirements of many Internet applications to provide reliable data delivery with a user experience in real-time. Differently, the proposed service supports large and reliable data transfer, where fast data transfer is not critical.

The proposed solution for the large and reliable data transfer service increases the energy consumption due to a higher number of packets that are sent. Even under perfect conditions without packet loss, there is a minimum number of control packets that the source and destination have to send for each data message from the application (as explained in Section 2.7). The number of messages increases under the conditions of packet loss, because data packets have to be resent. It is critical to keep traffic moderate and hence avoid the need for too many nodes to retransmit packets due to packet losses. Special attention must be given to gateway nodes if they are the destinations of data messages from remote nodes. On one hand, for each

connection with a remote node, the total number of control messages to be sent increases, which influences the energy consumption of a gateway node. On the other hand, as explained in Section 2.4, control packets consist of a header but have no payload. Therefore, their energy consumption is less than that of data packets, which may transmit several bytes up to the maximum payload allowed by the `MAX_PACKET_SIZE` configuration.

The implemented large and reliable data transfer service does not include congestion control. However, congestion could happen at the boards, which act as routers in the data path. A higher traffic volume in LoRa mesh networks may require congestion control to improve the network service. For this, when doing a reliable transfer, the source could leverage information that points to congestion issues, such as the RTT of received acknowledgment messages or not receiving them. The source may react to this information, for example, by delaying sending the next data packet. Another option is for the routers to provide network-related details, e.g., the number of packets in their send queue. When considering congestion support in LoRa mesh networks, however, we also have to consider that LoRa networks, compared to the Internet, can be run as relatively private networks, being deployed for one application or by a single operator. This allows the network to be much more controlled in terms of network traffic, and the network activity may be designed beforehand to reduce the risk of congestion. Adding congestion control will be inconvenient due to additional control information, increased complexity, and overhead. Another aspect is the current control messages from routers: routing messages are not forwarded; they are just received by neighboring nodes.

The LoRa radio chip that provides the node's network interface has no computing resources beyond sending and receiving LoRa packets. Therefore, any computing related to the networking protocol has to be done by the microcontroller, hence using the microcontroller's computing and memory resources. Thus, the device's memory is used for the queues of the packets. The resource constraints of LoRa radios differ from the computing capacities of network cards used for hosts on the Internet to perform TCP/IP, which have proper send and receive buffers and take over some computations of the networking stack, relieving load from the CPU.

7. Insights on usability and enhancement of the presented code

7.1. Practical considerations

We have experimentally shown the capacity of the developed large and reliable data transfer service for message sizes of 500 bytes (Section 4.2), of 3500 bytes and 10080 bytes (experiments 1-3 of Section 4.3), and 1014 bytes (experiment 4 of Section 4.3). From a design point of view, the 2-byte *Number* field in the packet header (see Section 2.2) can handle a large number of data chunks of a large message. A configurable parameter of the reliable transfer is the `MAX_TIMEOUTS`, which is counted per connection. For the practical usability of reliable transfer of large messages in environments with high packet loss and when using a high SF, one has to consider the probability of reaching the default `MAX_TIMEOUTS` value of 10 before finalizing the message transfer. A solution could be to increase the `MAX_TIMEOUTS` value, which would avoid prematurely aborting the connection, or to split the application data at the application layer, and thus present it as smaller messages to the large and reliable transfer service of the library, as was done in experiments 2 and 4 of Section 4.2. The `BuildOptions.h` configuration file of LoRaMesher allows these default values to be customized.

Experience that we have obtained from a permanent deployment of a monitoring application within an experimental LoRaMesher network[5] of more than 10 nodes indicates that when large and reliable data transfer is applied, the amount of data and the periodicity of sending packets from the nodes to a gateway node have to be carefully chosen. Packet losses that happen when too much traffic is generated in the network can lead to a critical increase in the gateway's Q_WRP queue due to the growing amount of ongoing and unfinished connections. A solution can be (if compatible with the application requirements), to configure in the implementation of the application's task a longer periodicity for the sending of data packets and thus reduce traffic.

7.2. Optimizing protocol overhead

In Section 2.2 we presented the LoRaMesher packet format. As described, the total size of the header is 11 bytes, of which 3 bytes, consisting of the 2-byte *Sequence Id* and the 1-byte *Number* fields, were introduced to implement the new large and reliable transfer service. The size of these fields was

[5] `https://tomir.ac.upc.edu/loraupc/index.php`

chosen to enable the protocol to support future application scenarios that may be more demanding in terms of message size and the number of concurrent applications. If the specific application case is known and the control overhead is critical, each of these two fields could be customized to the required number of bits. For instance, if the maximum number of packets for data chunks for a message is known, then the *Number* field could be reduced to only the bits needed to represent this maximum. Similarly, if the maximum number of concurrent message transfers between two nodes is known and low, the *Sequence Id* could be reduced from a byte to the minimum necessary number of bits, thereby reducing protocol overhead.

A similar optimization could be done for the 1-byte *Role* field that is part of the routing packet, as described in Section 2.3. Currently, the field is used to identify gateway nodes in a LoRaMesher network deployment, and by periodically exchanging routing tables, any change is regularly propagated and updated. The size of the *Role* field, however, can be optimized according to the specific application. Indeed, for application cases that operate only within the LoRa mesh network without gateways, as in, for instance, Meshtastic [6], there may be no need for the *Role* field in the routing packet.

Since LoRa provides low data-rate communication, optimizing protocol overhead is desirable from a technical perspective, as even small packet-size reductions can improve network performance and energy consumption. From the perspective of a commercial provider, it may be important to determine whether this technical optimization translates into value, and the benefit must be quantified for the specific application scenario.

7.3. Trade-offs between byte efficiency and code complexity

Given that LoRaMesher was conceived as a community effort, code complexity must be kept within reasonable limits to facilitate participation by other developers. While the previous subsection discussed optimizations that could reduce protocol overhead by bit-packing header fields, such optimizations introduce a fundamental trade-off: the gains in byte efficiency must be weighed against the increase in code complexity and reduced maintainability.

A core design principle adopted in LoRaMesher is that the data structures in the code directly map to the packet format transmitted over the network. Specifically, C++ classes are defined with fields that represent packet headers, aligned in bytes, allowing direct casting (using techniques such as *reinterpret_cast*) between received byte streams and structured packet objects.

This design choice means that reading the code provides an immediate understanding of the packet format, without requiring separate documentation or complex bit manipulation routines to extract fields. For instance, when the implementation receives a LoRaMesher packet, the packet type and routing information can be accessed directly through the struct members rather than through bit operations. While operating with bit fields could save a few bytes per packet, it would require additional encoding and decoding logic, increase the likelihood of implementation errors, and make the codebase less accessible to new contributors.

The decision to use byte-aligned fields and straightforward data structures prioritizes code clarity over maximal byte efficiency. This approach has proven practical: new packet types can be easily added through the *createPacket* functionality, and the direct class-to-packet relationship simplifies debugging and testing. When byte efficiency is critical for specific deployments, targeted optimizations can still be applied as discussed in the previous subsection. However, maintaining a readable and accessible codebase by default ensures the library's long-term sustainability and encourages community participation.

8. Conclusion

In this paper, we introduced a new service for the LoRaMesher library that facilitates large and reliable data transfers across a LoRa mesh network. To achieve this, we extended the LoRaMesher packet structure to include new packet types, designed new tasks and a new data delivery service that enables the reliable transmission of large messages between source and destination nodes.

We experimented with the implemented service in a LoRa mesh network built with real hardware nodes. The results evaluate the protocol's performance when transmitting large and reliable messages under various network conditions, achieving in the fully connected topology a 99.97% delivery ratio when the nine nodes sent 200 messages of 50 bytes to a gateway node (Experiment 1), and 96.9% when the nine nodes sent 20 large messages of 500 bytes divided into 100 byte packets to the gateway node (Experiment 2). Perfect delivery (100%) was achieved in the linear topology when the node at one end sent the 50 byte and 89% delivery for 500 byte messages over multiple hops to the node at the opposite end (Experiments 3 and 4). These results show

the protocol's operation even during simultaneous transmissions, increased traffic, and over multiple hops.

Additional experiments demonstrated that LoRaMesher provides the targeted functionality under operational conditions. Large and reliable message sending is not available in any other open-source LoRa mesh network implementation. We expect this novel network service enable new kinds of distributed IoT applications on tiny nodes, such as Edge AI applications in LoRa mesh networks.

Looking ahead, we aim to broaden the service's capabilities by incorporating techniques to increase the library's robustness and security, and designing components for the higher layer of the network stack to ease LoRaMesher interfacing with real applications. LoRaMesher is open-source and available on GitHub.

Acknowledgment

This work was funded by the Spanish Ministerio de Ciencia, Innovación y Universidades through the grants PDC2023-145809-I00/AEI/10.13039/501100011033 and PID2023-146066OB-I00 (AEI 2023 Knowledge Generation Projects) MICIU/AEI/10.13039/501100011033/; the Recovery and Resilience Mechanism of the European Union; and by the European Union NextGenerationEU; and the Generalitat de Catalunya through the grants 2021-SGR-01059; and the Universitat Politècnica de Catalunya through the grant ALECTORS 0BV05-1. The authors gratefully acknowledge the insightful comments and constructive feedback provided by the anonymous reviewers.

References

[1] M. A. M. Almuhaya, W. A. Jabbar, N. Sulaiman, S. Abdulmalek, A Survey on LoRaWAN Technology: Recent Trends, Opportunities, Simulation Tools and Future Directions, Electronics 11 (1) (2022). doi:10.3390/electronics11010164.
URL https://www.mdpi.com/2079-9292/11/1/164

[2] J. Haxhibeqiri, E. De Poorter, I. Moerman, J. Hoebeke, A Survey of LoRaWAN for IoT: From Technology to Application, Sensors 18 (11) (2018). doi:10.3390/s18113995.
URL https://www.mdpi.com/1424-8220/18/11/3995

[3] R. Pueyo Centelles, F. Freitag, R. Meseguer, L. Navarro, Beyond the Star of Stars: An Introduction to Multihop and Mesh for LoRa and LoRaWAN, IEEE Pervasive Computing 20 (2) (2021) 63–72. doi:10.1109/MPRV.2021.3063443.

[4] J. R. Cotrim, J. H. Kleinschmidt, LoRaWAN Mesh Networks: A Review and Classification of Multihop Communication, Sensors 20 (15) (2020). doi:10.3390/s20154273.
URL https://www.mdpi.com/1424-8220/20/15/4273

[5] R. Berto, P. Napoletano, M. Savi, A LoRa-Based Mesh Network for Peer-to-Peer Long-Range Communication, Sensors 21 (13) (2021). doi:10.3390/s21134314.
URL https://www.mdpi.com/1424-8220/21/13/4314

[6] Meshtastic: Open Source hiking, pilot, skiing and secure GPS mesh communicator, https://meshtastic.org/, accessed: Dec. 2025.

[7] R. Pueyo Centelles, R. Meseguer, F. Freitag, R. Baig Viñas, L. Navarro, A minimalistic distance-vector routing protocol for lora mesh networks, IEEE Access 12 (2024) 128941–128962. doi:10.1109/ACCESS.2024.3443605.

[8] R. Kallimani, K. Pai, P. Raghuwanshi, S. Iyer, O. L. A. López, Tinyml: Tools, applications, challenges, and future research directions, Multimedia Tools and Applications 83 (10) (2024) 29015–29045. doi:10.1007/s11042-023-16740-9.
URL https://doi.org/10.1007/s11042-023-16740-9

[9] D. Vasconcelos, M. S. Yin, F. Wetjen, A. Herbst, T. Ziemer, A. Förster, T. Barkowsky, N. Nunes, P. Haddawy, Counting Mosquitoes in the Wild: An Internet of Things Approach, in: Proceedings of the Conference on Information Technology for Social Good, GoodIT '21, Association for Computing Machinery, New York, NY, USA, 2021, p. 43–48. doi:10.1145/3462203.3475914.
URL https://doi.org/10.1145/3462203.3475914

[10] B. Sudharsan, J. G. Breslin, M. Tahir, M. Intizar Ali, O. Rana, S. Dustdar, R. Ranjan, OTA-TinyML: Over the Air Deployment of TinyML

Models and Execution on IoT Devices, IEEE Internet Computing 26 (3) (2022) 69–78. doi:10.1109/MIC.2021.3133552.

[11] J. Miquel Solé, R. Pueyo Centelles, F. Freitag, R. Meseguer, Implementation of a LoRa Mesh Library, IEEE Access 10 (2022) 113158–113171. doi:10.1109/ACCESS.2022.3217215.

[12] M. Sargent, J. Chu, D. V. Paxson, M. Allman, Computing TCP's Retransmission Timer, RFC 6298 (Jun. 2011). doi:10.17487/RFC6298.
URL https://www.rfc-editor.org/info/rfc6298

[13] A. W.-L. Wong, S. L. Goh, M. K. Hasan, S. Fattah, Multi-Hop and Mesh for LoRa Networks: Recent Advancements, Issues, and Recommended Applications, ACM Comput. Surv. 56 (6) (Jan. 2024). doi:10.1145/3638241.
URL https://doi.org/10.1145/3638241

[14] LoRa Alliance Technical Commitee, LoRaWAN ® Relay Specification TS011-1.0.0 , Accessed: Dec. 2025.
URL https://resources.lora-alliance.org/technical-specifications/ts011-1-0-0-relay

[15] A. Osorio, M. Calle, J. D. Soto, J. E. Candelo-Becerra, Routing in LoRaWAN: Overview and Challenges, IEEE Communications Magazine 58 (6) (2020) 72–76. doi:10.1109/MCOM.001.2000053.

[16] B. Sartori, S. Thielemans, M. Bezunartea, A. Braeken, K. Steenhaut, Enabling RPL multihop communications based on LoRa, in: 2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), 2017, pp. 1–8. doi:10.1109/WiMOB.2017.8115756.

[17] R. Alexander, A. Brandt, J. Vasseur, J. Hui, K. Pister, P. Thubert, P. Levis, R. Struik, R. Kelsey, T. Winter, RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks, RFC 6550 (Mar. 2012). doi:10.17487/RFC6550.
URL https://www.rfc-editor.org/info/rfc6550

[18] C. T. Duong, M. K. Kim, Multi-Hop Linear Network Based on LoRa, Advanced Science and Technology Letters 150 (2018) 29–33. doi:10.14257/astl.2018.150.08.

[19] A. Abrardo, A. Pozzebon, A Multi-Hop LoRa Linear Sensor Network for the Monitoring of Underground Environments: The Case of the Medieval Aqueducts in Siena, Italy, Sensors 19 (2) (2019) 402. doi:10.3390/s19020402.

[20] H.-C. Lee, K.-H. Ke, Monitoring of Large-Area IoT Sensors Using a LoRa Wireless Mesh Network System: Design and Evaluation, IEEE Transactions on Instrumentation and Measurement 67 (9) (2018) 2177–2187. doi:10.1109/TIM.2018.2814082.

[21] D. L. Mai, M. K. Kim, Multi-Hop LoRa Network Protocol with Minimized Latency, Energies 13 (6) (2020). doi:10.3390/en13061368.
URL https://www.mdpi.com/1996-1073/13/6/1368

[22] L. Leonardi, L. Lo Bello, G. Patti, Mrt-lora: A multi-hop real-time communication protocol for industrial iot applications over lora networks, Computer Communications 199 (2023) 72–86. doi:https://doi.org/10.1016/j.comcom.2022.12.013.
URL https://www.sciencedirect.com/science/article/pii/S0140366422004637

[23] G. Zhu, C.-H. Liao, T. Sakdejayont, I.-W. Lai, Y. Narusue, H. Morikawa, Improving the Capacity of a Mesh LoRa Network by Spreading-Factor-Based Network Clustering, IEEE Access 7 (2019) 21584–21596. doi:10.1109/ACCESS.2019.2898239.

[24] L. Prade, J. Moraes, E. de Albuquerque, D. Rosário, C. B. Both, Multi-radio and multi-hop LoRa communication architecture for large scale IoT deployment, Computers and Electrical Engineering 102 (2022) 108242. doi:https://doi.org/10.1016/j.compeleceng.2022.108242.
URL https://www.sciencedirect.com/science/article/pii/S0045790622004773

[25] M. McCauley, RadioHead Packet Radio Library for embedded microprocessors (2014).
URL https://www.airspayce.com/mikem/arduino/RadioHead/

[26] M. Haubro, C. Orfanidis, G. Oikonomou, X. Fafoutis, Tsch-over-lora: long range and reliable ipv6 multi-hop networks for the internet

of things, Internet Technology Letters 3 (4) (2020) e165. arXiv: https://onlinelibrary.wiley.com/doi/pdf/10.1002/itl2.165, doi:https://doi.org/10.1002/itl2.165.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/itl2.165

[27] J. Sakamoto, D. Nobayashi, K. Tsukamoto, T. Ikenaga, G. Sato, K. Takizawa, Improvement of TCP Performance based on Characteristics of Private LoRa Interface, in: Proceedings - International Computer Software and Applications Conference, Vol. 2023-June, IEEE Computer Society, 2023, pp. 998–999. doi:10.1109/COMPSAC57700.2023.00145.

[28] P. Manzoni, S. E. Merzougui, C. E. Palazzi, P. Pozzan, A resilient lora-based solution to support pervasive sensing, Electronics 12 (13) (2023). doi:10.3390/electronics12132952.
URL https://www.mdpi.com/2079-9292/12/13/2952

[29] F. Stann, J. Heidemann, Rmst: reliable data transport in sensor networks, in: Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications, 2003, pp. 102–112. doi:10.1109/SNPA.2003.1203361.

[30] A. Ghosh, S. Misra, V. Udutalapally, D. Das, LoRaute: Routing Messages in Backhaul LoRa Networks for Underserved Regions, IEEE Internet of Things Journal 10 (22) (2023) 19964–19971. doi:10.1109/JIOT.2023.3281941.

[31] R. Muñoz, J. S. Hidalgo, F. Canales, D. Dujovne, S. Céspedes, SCHC over LoRaWAN Efficiency: Evaluation and Experimental Performance of Packet Fragmentation, Sensors 22 (4) (2022). doi:10.3390/s22041531.
URL https://www.mdpi.com/1424-8220/22/4/1531

[32] The FUOTA Working Group of the LoRa Alliance Technical Committee, LoRaWAN® Fragmented Data Block Transport Specification TS004-2.0.0 (2022).
URL https://resources.lora-alliance.org/technical-specifications/ts004-2-0-0-fragmented-data-block-transport

[33] B. Arratia, P. García-Guillamón, C. T. Calafate, J.-C. Cano, J. M. Cecilia, P. Manzoni, A modular and mesh-capable LoRa based Content Transfer Protocol for Environmental Sensing, in: IEEE 20th Consumer Communications & Networking Conference (CCNC), 2023, pp. 378–383. doi:10.1109/CCNC51644.2023.10060496.

[34] A. M. Cardenas, M. K. Nakamura Pinto, E. Pietrosemoli, M. Zennaro, M. Rainone, P. Manzoni, A Low-Cost and Low-Power Messaging System Based on the LoRa Wireless Technology, Mobile Networks and Applications 25 (3) (2020) 961–968. doi:10.1007/s11036-019-01235-5.

[35] B. Arratia, E. Rosas, C. T. Calafate, J.-C. Cano, J. M. Cecilia, P. Manzoni, AlLoRa: Empowering environmental intelligence through an advanced LoRa-based IoT solution, Computer Communications 218 (2024) 44–58. doi:https://doi.org/10.1016/j.comcom.2024.02.014.
URL https://www.sciencedirect.com/science/article/pii/S0140366424000641