Lora Mesh

v1.0.0

Generated by Doxygen 1.10.0

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

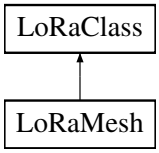
Class Documentation

4.1 LoRaMesh Class Reference

The LoRaMesh class represents a mesh network using LoRa communication.

`#include <LoRaMesh.h>`

Inheritance diagram for LoRaMesh:



Classes

- struct packet

  Structure representing a packet in the LoRaMesh network.

Public Member Functions

- int loraSetup (int myID, int ssPin, int rst, int dio0)
- unsigned short int readInt ()
- void writeInt (unsigned short int num)
- void debug (packet p)
- void meshRead ()
- packet createMessagePacket (String msg, int idToSend, int msgId, int packetType=0)
- packet createAckPacket (int sizeOfMsgToAck, int idToSend, int msgIdToAck)
- void meshSend (packet sendPacket)
- int getmyID ()
- void routingPacket ()
- void updateRoutingTable (int id, unsigned short int size)
- void printRoutingTable ()
- template<typename T >
  void debugPrint (T input)
- template<typename T >
  void debugPrintln (T input)
- void debugPrintln ()
- void readMsgCallback (std::function< void(int, int, String)> callbackData)
- void readAckCallback (std::function< void(int, int)> callbackAck)

Public Attributes

- byte typeDataMessage = 0
- byte typeRoutingMessage = 1
- byte typeAckMessage = 6
- String receivedMsg
- char payload [244]
- bool debugging = true
- std::map< unsigned short int, std::pair< unsigned short int, unsigned short int > > routingTable
- std::map< unsigned short int, unsigned short int > neighbours
- std::vector< unsigned short int > knownRoutes

Private Attributes

- unsigned short int myID
- std::function< void(int, int, String)> callbackDataFunction
- std::function< void(int, int)> callbackAckFunction
- bool callbackDataFlag = false
- bool callbackAckFlag = false

4.1.1 Detailed Description

The LoRaMesh class represents a mesh network using LoRa communication.

This class extends the LoRaClass and provides additional functionality for creating and managing a mesh network. It includes methods for setting up the LoRa module, sending and receiving packets, managing routing tables, and debugging.

The LoRaMesh class also defines a packet structure for sending and receiving messages within the mesh network. It includes fields for the source, destination, next hop, packet size, message ID, type, preamble, and payload.

The class also provides callback functions for handling received data messages and acknowledgment messages.

Note

This class assumes the use of the LoRa library by Sandeep Mistry (https://github.↩ com/sandeepmistry/arduino-LoRa) and requires the LoRa module to be properly configured.

Definition at line 24 of file LoRaMesh.h.

4.1.2 Member Function Documentation

4.1.2.1 createAckPacket()

```
LoRaMesh::packet LoRaMesh::createAckPacket (
            int sizeOfMsgToAck,
            int idToSend,
            int msgIdToAck )
```

A function to create an acknowledgment packet based on the message packet

Parameters

in	*sizeOfMsgToAck*	The size of the message to be acknowledged
in	*idToSend*	The recipient ID
in	*msgIdToAck*	The message ID to be acknowledged

Returns

A packet

Definition at line 480 of file LoRaMesh.cpp.

4.1.2.2 createMessagePacket()

```
LoRaMesh::packet LoRaMesh::createMessagePacket (
            String msg,
            int idToSend,
            int msgId,
            int packetType = 0 )
```

A function create a new packet

Parameters

in	*msg*	A string containing the message
in	*idToSend*	The recipient ID
in	*msgId*	The message ID
in	*packetType*	The type of the packet being sent

Returns

A packet

Definition at line 343 of file LoRaMesh.cpp.

4.1.2.3 debug()

```
void LoRaMesh::debug (
            packet p )
```

Used for debugging the packet received from the LoRa module and printing it to the serial monitor

Parameters

in	*p*	The packet to be debugged

Definition at line 156 of file LoRaMesh.cpp.

4.1.2.4 debugPrint()

```
template<typename T >
void LoRaMesh::debugPrint (
            T print )
```

A function to print debug messages to the serial monitor based on a global debugging flag

Parameters

in	*print*	The string to be printed

Definition at line 19 of file LoRaMesh.cpp.

4.1.2.5 debugPrintln() [1/2]

```
void LoRaMesh::debugPrintln ( )
```

A function to print debug messages to the serial monitor with new line and no input, aka. empty line based on a global debugging flag

Definition at line 44 of file LoRaMesh.cpp.

4.1.2.6 debugPrintln() [2/2]

```
template<typename T >
void LoRaMesh::debugPrintln (
            T print )
```

A function to print debug messages to the serial monitor with new line based on a global debugging flag

Parameters

in	*print*	The string to be printed

Definition at line 33 of file LoRaMesh.cpp.

4.1.2.7 getmyID()

```
int LoRaMesh::getmyID ( )
```

Getter for the ID of this node

Returns

The ID of this node

Definition at line 109 of file LoRaMesh.cpp.

4.1.2.8 loraSetup()

```
int LoRaMesh::loraSetup (
            int myID,
            int ssPin,
            int rst,
            int dio0 )
```

This function extends the default one from the LoRa library adding all of the necessary parameters for the LoRaMesh class and setting the necessary pins for the LoRa module to function

Parameters

in	*myID*	ID to be used by this node
in	*ssPin*	The slave select pin for the LoRa module
in	*rst*	The reset pin for the LoRa module
in	*dio0*	The interrupt pin for the LoRa module

Returns

1 if the LoRa module is successfully started, 0 otherwise

Definition at line 62 of file LoRaMesh.cpp.

4.1.2.9 meshRead()

```
void LoRaMesh::meshRead ( )
```

A function to read and interpret the packet received from the LoRa module, this is where we use the callback functions to interpret the message

Definition at line 273 of file LoRaMesh.cpp.

4.1.2.10 meshSend()

```
void LoRaMesh::meshSend (
            packet sendPacket )
```

Function to send a packet to the LoRa module

Parameters

in	*sendPacket*	The packet to be sent

Definition at line 376 of file LoRaMesh.cpp.

4.1.2.11 printRoutingTable()

```
void LoRaMesh::printRoutingTable ( )
```

Used for printing the routing table to the serial monitor

Definition at line 194 of file LoRaMesh.cpp.

4.1.2.12 readAckCallback()

```
void LoRaMesh::readAckCallback (
            std::function< void(int, int)> callbackAck )
```

Register a callback function to be called when a acknowlegment is received so that any user can interpret the acknowlegment as they want

Parameters

callbackAck	The callback function to register.

Definition at line 97 of file LoRaMesh.cpp.

4.1.2.13 readInt()

```
unsigned short int LoRaMesh::readInt ( )
```

Used for reading an integer received from the LoRa module

Returns

The received integer

Definition at line 119 of file LoRaMesh.cpp.

4.1.2.14 readMsgCallback()

```
void LoRaMesh::readMsgCallback (
            std::function< void(int, int, String)> callbackData )
```

Register a callback function to be called when a message is received so that any user can interpret the message as they want

Parameters

in	*callbackData*	The callback function to register.

Definition at line 84 of file LoRaMesh.cpp.

4.1.2.15 routingPacket()

```
void LoRaMesh::routingPacket ( )
```

Sends a routing packet to the LoRa module. This function constructs a packet with routing information and sends it to the LoRa module. It updates the neighbor count and checks for any neighbors with a count above a threshold. If a neighbor's count exceeds the threshold, it removes the neighbor from the routing table and neighbor list. Finally, it sends the packet to the network.

Definition at line 409 of file LoRaMesh.cpp.

4.1.2.16 updateRoutingTable()

```
void LoRaMesh::updateRoutingTable (
            int id,
            unsigned short int size )
```

A function to update the routing table with a new node

Parameters

in	*id*	The ID of the node to be added to the routing table
in	*size*	The size of the packet received

Definition at line 217 of file LoRaMesh.cpp.

4.1.2.17 writeInt()

```
void LoRaMesh::writeInt (
            unsigned short int num )
```

Used for writing an integer to the LoRa module

Parameters

in	*num*	The integer to be sent

Definition at line 137 of file LoRaMesh.cpp.

4.1.3 Member Data Documentation

4.1.3.1 callbackAckFlag

```
bool LoRaMesh::callbackAckFlag = false  [private]
```

Flag to indicate if a callback function for acknowledgment messages is registered.

Definition at line 30 of file LoRaMesh.h.

4.1.3.2 callbackAckFunction

```
std::function<void(int, int)> LoRaMesh::callbackAckFunction  [private]
```

Registered user function for handling acknolegments.

Definition at line 28 of file LoRaMesh.h.

4.1.3.3 callbackDataFlag

```
bool LoRaMesh::callbackDataFlag = false  [private]
```

Flag to indicate if a callback function for data messages is registered.

Definition at line 29 of file LoRaMesh.h.

4.1.3.4 callbackDataFunction

```
std::function<void(int, int, String)> LoRaMesh::callbackDataFunction  [private]
```

Registered user function for handling messages.

Definition at line 27 of file LoRaMesh.h.

4.1.3.5 debugging

```
bool LoRaMesh::debugging = true
```

Debugging flag.

Definition at line 50 of file LoRaMesh.h.

4.1.3.6 knownRoutes

```
std::vector<unsigned short int> LoRaMesh::knownRoutes
```

A table containing known routes by saving their nodeID.

Definition at line 53 of file LoRaMesh.h.

4.1.3.7 myID

```
unsigned short int LoRaMesh::myID  [private]
```

ID of this node.

Definition at line 26 of file LoRaMesh.h.

4.1.3.8 neighbours

```
std::map<unsigned short int,unsigned short int> LoRaMesh::neighbours
```

Neighbour table containing nodeID and hopCount.

Definition at line 52 of file LoRaMesh.h.

4.1.3.9 payload

```
char LoRaMesh::payload[244]
```

Packet data.

Definition at line 49 of file LoRaMesh.h.

4.1.3.10 receivedMsg

```
String LoRaMesh::receivedMsg
```

Received message.

Definition at line 48 of file LoRaMesh.h.

4.1.3.11 routingTable

```
std::map<unsigned short int,std::pair<unsigned short int,unsigned short int> > LoRaMesh↩
::routingTable
```

Routing table containing nodeID, nextHop, and hopCount.

Definition at line 51 of file LoRaMesh.h.

4.1.3.12 typeAckMessage

```
byte LoRaMesh::typeAckMessage = 6
```

Acknowledgment type packet.

Definition at line 34 of file LoRaMesh.h.

4.1.3.13 typeDataMessage

```
byte LoRaMesh::typeDataMessage = 0
```

Data type packet.

Definition at line 32 of file LoRaMesh.h.

4.1.3.14 typeRoutingMessage

```
byte LoRaMesh::typeRoutingMessage = 1
```

Routing type packet.

Definition at line 33 of file LoRaMesh.h.

The documentation for this class was generated from the following files:

- LoRaMesh.h
- LoRaMesh.cpp

4.2 LoRaMesh::packet Struct Reference

Structure representing a packet in the LoRaMesh network.

`#include <LoRaMesh.h>`

Public Attributes

- unsigned short int source
- unsigned short int destination
- unsigned short int nextHop
- unsigned short int packetSize
- unsigned short int msgId = 2
- byte type
- byte preamble
- char payload [244]

4.2.1 Detailed Description

Structure representing a packet in the LoRaMesh network.

Definition at line 38 of file LoRaMesh.h.

4.2.2 Member Data Documentation

4.2.2.1 destination

```
unsigned short int LoRaMesh::packet::destination
```

Receiver nodeID.

Definition at line 40 of file LoRaMesh.h.

4.2.2.2 msgId

```
unsigned short int LoRaMesh::packet::msgId = 2
```

Message ID.

Definition at line 43 of file LoRaMesh.h.

4.2.2.3 nextHop

```
unsigned short int LoRaMesh::packet::nextHop
```

Next hop nodeID.

Definition at line 41 of file LoRaMesh.h.

4.2.2.4 packetSize

```
unsigned short int LoRaMesh::packet::packetSize
```

Size of the packet.

Definition at line 42 of file LoRaMesh.h.

4.2.2.5 payload

```
char LoRaMesh::packet::payload[244]
```

Packet data.

Definition at line 46 of file LoRaMesh.h.

4.2.2.6 preamble

```
byte LoRaMesh::packet::preamble
```

Preamble.

Definition at line 45 of file LoRaMesh.h.

4.2.2.7 source

```
unsigned short int LoRaMesh::packet::source
```

Sender nodeID.

Definition at line 39 of file LoRaMesh.h.

4.2.2.8 type

```
byte LoRaMesh::packet::type
```

Type of the packet.

Definition at line 44 of file LoRaMesh.h.

The documentation for this struct was generated from the following file:

- LoRaMesh.h

Chapter 5

File Documentation

5.1 LoRaMesh.cpp

```
00001 #include "LoRaMesh.h"
00002 #include <SPI.h>
00003 #include <LoRa.h>
00004 #include <iterator>
00005 #include <vector>
00006 #include <algorithm>
00007 #include <map>
00008 #include <utility>
00009 #include <stdio.h>
00010
00018 template <typename T>
00019 void LoRaMesh::debugPrint(T print)
00020 {
00021     if (debugging)
00022         Serial.print(print);
00023 }
00024
00032 template <typename T>
00033 void LoRaMesh::debugPrintln(T print)
00034 {
00035     if (debugging)
00036         Serial.println(print);
00037 }
00038
00044 void LoRaMesh::debugPrintln()
00045 {
00046     if (debugging)
00047         Serial.println();
00048 }
00049
00062 int LoRaMesh::loraSetup(int myID, int ssPin, int rst, int dio0)
00063 {
00064     debugPrint("loraSetup(");
00065     debugPrint(myID);
00066     debugPrintln(")");
00067
00068     this->myID = myID;
00069     setPins(ssPin, rst, dio0);
00070     if (!begin(868E6)) return 0;
00071     if (!callbackAckFlag || !callbackDataFlag) return 0;
00072     routingTable[this->myID] = {this->myID, 0};
00073     knownRoutes.push_back(this->myID);
00074     return 1;
00075 }
00076
00084 void LoRaMesh::readMsgCallback(std::function<void(int, int, String)> callbackData) //
00085 {
00086     callbackDataFunction = callbackData;
00087     callbackDataFlag = true;
00088     debugPrintln("readMsgCallback");
00089 }
00090
00097 void LoRaMesh::readAckCallback(std::function<void(int, int)> callbackAck)
00098 {
00099     callbackAckFunction = callbackAck;
00100     callbackAckFlag = true;
00101     debugPrintln("readAckCallback");
00102 }
```

```
00103
00109 int LoRaMesh::getmyID()
00110 {
00111     return myID;
00112 }
00113
00119 unsigned short int LoRaMesh::readInt()
00120 {
00121     debugPrint("readInt(");
00122     debugPrint("");
00123     debugPrintln(")");
00124
00125     byte low = (byte)read();
00126     byte high = (byte)read();
00127     delay(10);
00128     return (high << 8 | low);
00129 }
00130
00137 void LoRaMesh::writeInt(unsigned short int num)
00138 {
00139     debugPrint("writeInt(");
00140     debugPrint(num);
00141     debugPrintln(")");
00142
00143     byte high = highByte(num);
00144     byte low = lowByte(num);
00145     write(low);
00146     write(high);
00147 }
00148
00156 void LoRaMesh::debug(packet p)
00157 {
00158     debugPrint("packet(");
00159     debugPrint("p");
00160     debugPrint(") ");
00161     debugPrint("debugging = ");
00162     debugPrintln(debugging);
00163
00164     if (!debugging)
00165     {
00166         return;
00167     }
00168     debugPrint("Preamble: ");
00169     debugPrintln(p.preamble);
00170     debugPrint("Destination: ");
00171     debugPrintln(p.destination);
00172     debugPrint("Source: ");
00173     debugPrintln(p.source);
00174     debugPrint("Next Hop: ");
00175     debugPrintln(p.nextHop);
00176     debugPrint("Type: ");
00177     debugPrintln(p.type);
00178     debugPrint("Packet Size: ");
00179     debugPrintln(p.packetSize);
00180     debugPrint("Payload: ");
00181     for (int i = 0; i < p.packetSize; ++i)
00182     {
00183         debugPrint(p.payload[i]);
00184     }
00185     debugPrint("\nMsgId: ");
00186     debugPrintln(p.msgId);
00187     debugPrintln();
00188 }
00189
00194 void LoRaMesh::printRoutingTable()
00195 {
00196     debugPrint("printRoutingTable(");
00197     debugPrint("");
00198     debugPrintln(")");
00199
00200     for (const auto &x : routingTable)
00201     {
00202         Serial.print(x.first);
00203         Serial.print(" ");
00204         Serial.print(x.second.first);
00205         Serial.print(" ");
00206         Serial.println(x.second.second);
00207     }
00208 }
00209
00217 void LoRaMesh::updateRoutingTable(int id, unsigned short int size)
00218 {
00219     debugPrint("updateRoutingTable");
00220     debugPrint(id);
00221     debugPrint(", ");
00222     debugPrint(size);
00223     debugPrintln(")");
```

```
00224
00225     std::vector<unsigned short int> vNew;
00226     for (int i = 0; i < size / 3; ++i)
00227     {
00228         int nextHop = readInt();
00229         int nodeID = readInt();
00230         int newRssi = readInt();
00231         vNew.push_back(nodeID);
00232         debugPrint(nextHop);
00233         debugPrint(" ");
00234         debugPrint(nodeID);
00235         debugPrint(" ");
00236         debugPrint(newRssi);
00237         debugPrintln();
00238
00239         if (routingTable.find(nodeID) == routingTable.end())
00240             routingTable[nodeID] = {id, newRssi};
00241         else
00242         {
00243             if (routingTable[nodeID].second > newRssi)
00244                 routingTable[nodeID] = {id, newRssi};
00245         }
00246     }
00247
00248     for (const auto &x : routingTable)
00249     {
00250         if (x.second.first == id)
00251         {
00252             if (std::find(vNew.begin(), vNew.end(), x.first) == vNew.end())
00253             {
00254             }
00255             // routingTable.erase(x.first);
00256         }
00257     }
00258
00259     if (neighbours.find(id) == neighbours.end())
00260         neighbours.insert({id, 0});
00261     else
00262         neighbours[id] = 0;
00263
00264     if (debugging) printRoutingTable();
00265 }
00266
00273 void LoRaMesh::meshRead()
00274 {
00275     debugPrint("meshRead(");
00276     debugPrint("");
00277     debugPrintln(")");
00278
00279     packet receivePacket;
00280     receivePacket.preamble = (byte)read();
00281     if (receivePacket.preamble != 170)
00282         return;
00283
00284     receivePacket.destination = readInt();
00285     receivePacket.source = readInt();
00286     receivePacket.nextHop = readInt();
00287     receivePacket.type = (byte)read();
00288     receivePacket.packetSize = readInt();
00289
00290     if (receivePacket.type == typeRoutingMessage)
00291     {
00292         debugPrintln("typeRoutingMessage Receved!");
00293         updateRoutingTable(receivePacket.source, receivePacket.packetSize);
00294         return;
00295     }
00296
00297     for (int i = 0; i < (int)receivePacket.packetSize; ++i)
00298     {
00299         receivePacket.payload[i] = (char)(((byte)read()) - 1);
00300         delay(10);
00301     }
00302     receivePacket.msgId = readInt();
00303     receivePacket.payload[receivePacket.packetSize] = '\0';
00304     receivedMsg = String(receivePacket.payload);
00305     debug(receivePacket);
00306
00307     if (receivePacket.destination == myID)
00308     {
00309         if (receivePacket.type == typeAckMessage)
00310         {
00311             debugPrintln("typeAckMessage Receved!");
00312             callbackAckFunction(receivePacket.msgId, receivePacket.source);
00313             return;
00314         }
00315         if (receivePacket.type == typeDataMessage)
00316         {
```

```
00317             debugPrintln("typeDataMessage Receved!");
00318             callbackDataFunction(receivePacket.source, receivePacket.msgId, receivedMsg);
00319             delay(500);
00320             meshSend(createAckPacket(receivePacket.packetSize, receivePacket.source,
    receivePacket.msgId));
00321         }
00322     }
00323     if (receivePacket.nextHop != myID)
00324         return;
00325     if (receivePacket.type == typeDataMessage)
00326     {
00327         receivePacket.nextHop = routingTable[receivePacket.destination].first;
00328         delay(2500);
00329         meshSend(receivePacket);
00330     }
00331     return;
00332 }
00333
00343 LoRaMesh::packet LoRaMesh::createMessagePacket(String msg, int idToSend, int msgId, int packetType)
00344 {
00345     debugPrint("createMessagePacket(");
00346     debugPrint(msg);
00347     debugPrint(", ");
00348     debugPrint(idToSend);
00349     debugPrint(", ");
00350     debugPrint(msgId);
00351     debugPrintln(")");
00352
00353     packet Packet;
00354     Packet.preamble = (byte)170;
00355     Packet.destination = idToSend;
00356     Packet.source = myID;
00357     Packet.nextHop = routingTable[Packet.destination].first;
00358     Packet.type = (byte)packetType;
00359     Packet.packetSize = msg.length();
00360     msg.toCharArray(Packet.payload, 244);
00361     debugPrint("payload = ");
00362     for (int i = 0; i < Packet.packetSize; ++i)
00363         debugPrint(Packet.payload[i]);
00364     Packet.msgId = msgId;
00365     if (debugging)
00366         Serial.printf("Packet.msgId = %d\n", Packet.msgId);
00367     return Packet;
00368 }
00369
00376 void LoRaMesh::meshSend(packet sendPacket)
00377 {
00378     debugPrint("meshSend(");
00379     debug(sendPacket);
00380     debugPrintln(")");
00381
00382     beginPacket();
00383     write(sendPacket.preamble);
00384     writeInt(sendPacket.destination);
00385     writeInt(sendPacket.source);
00386     writeInt(sendPacket.nextHop);
00387     write(sendPacket.type);
00388     writeInt(sendPacket.packetSize);
00389     for (int i = 0; i < sendPacket.packetSize; ++i)
00390     {
00391         write((byte)((int)sendPacket.payload[i] + 1));
00392     }
00393     writeInt(sendPacket.msgId);
00394     debugPrint("endPacket() = ");
00395     debugPrintln(endPacket(1));
00396     return;
00397 }
00398
00399
00409 void LoRaMesh::routingPacket() //Bivski helloPacket
00410 {
00411     packet sendPacket;
00412     sendPacket.preamble = (byte)170;
00413     sendPacket.destination = 0;
00414     sendPacket.source = myID;
00415     sendPacket.nextHop = 0; // 0 je broadcast
00416     sendPacket.type = (byte)typeRoutingMessage;
00417     sendPacket.packetSize = knownRoutes.size() * 3;
00418     sendPacket.payload[0] = '\0';
00419     debugPrint("helloPacket(");
00420     debug(sendPacket);
00421     debugPrintln(")");
00422
00423     for (const auto &x : neighbours)
00424     {
00425         neighbours[x.first]++;
00426     }
```

```
00427
00428     debugPrint("neighbours = ");
00429     for (const auto &x : neighbours)
00430     {
00431         debugPrint("(");
00432         debugPrint(x.first);
00433         debugPrint(", ");
00434         debugPrint(x.second);
00435         debugPrint("), ");
00436         if (x.second >= 4)
00437         {
00438             for (const auto &y : routingTable)
00439                 if (y.second.first == x.first)
00440                     routingTable.erase(y.first);
00441             neighbours.erase(x.first);
00442             if (debugging) printRoutingTable();
00443         }
00444     }
00445     debugPrintln();
00446
00447     beginPacket();
00448     write(sendPacket.preamble);
00449     writeInt(sendPacket.destination);
00450     writeInt(sendPacket.source);
00451     writeInt(sendPacket.nextHop);
00452     write(sendPacket.type);
00453     writeInt(sendPacket.packetSize);
00454     debugPrintln("knownRoutes = (");
00455
00456     for (const auto &x : routingTable)
00457     {
00458         debugPrint(x.first);
00459         debugPrint(", ");
00460         writeInt(x.first);
00461         writeInt(x.second.first);
00462         writeInt(x.second.second);
00463     }
00464     debugPrintln(")");
00465     debugPrint("endPacket() = ");
00466     debugPrintln(endPacket(1));
00467     return;
00468 }
00469
00480 LoRaMesh::packet LoRaMesh::createAckPacket(int sizeOfMsgToAck, int idToSend, int msgIdToAck)
00481 {
00482     debugPrint("createAckPacket(");
00483     debugPrint(sizeOfMsgToAck);
00484     debugPrint(", ");
00485     debugPrint(idToSend);
00486     debugPrint(", ");
00487     debugPrint(msgIdToAck);
00488     debugPrintln(")");
00489
00490     return createMessagePacket(String(sizeOfMsgToAck), idToSend, msgIdToAck, typeAckMessage);
00491 }
```

5.2 LoRaMesh.h

```
00001 #ifndef LORAMESH_H
00002 #define LORAMESH_H
00003
00004 #include <SPI.h>
00005 #include <LoRa.h>
00006 #include <map>
00007 #include <utility>
00008 #include <vector>
00009
00024 class LoRaMesh : public LoRaClass{
00025 private:
00026   unsigned short int myID;
00027   std::function<void(int, int, String)> callbackDataFunction;
00028   std::function<void(int, int)> callbackAckFunction;
00029   bool callbackDataFlag = false;
00030   bool callbackAckFlag = false;
00031 public:
00032     byte typeDataMessage = 0,
00033     typeRoutingMessage = 1,
00034     typeAckMessage = 6;
00038     struct packet{
00039       unsigned short int source;
00040       unsigned short int destination;
00041       unsigned short int nextHop;
00042       unsigned short int packetSize;
```

```
00043       unsigned short int msgId = 2;
00044       byte type;
00045       byte preamble;
00046       char payload[244];
00047     };
00048     String receivedMsg;
00049     char payload[244];
00050     bool debugging = true;
00051     std::map<unsigned short int,std::pair<unsigned short int,unsigned short int»routingTable;
00052     std::map<unsigned short int,unsigned short int>neighbours;
00053     std::vector<unsigned short int>knownRoutes;
00054     int loraSetup(int myID, int ssPin, int rst, int dio0);
00055     unsigned short int readInt();
00056     void writeInt(unsigned short int num);
00057     void debug(packet p);
00058     void meshRead();
00059     packet createMessagePacket(String msg, int idToSend, int msgId, int packetType = 0);
00060     packet createAckPacket(int sizeOfMsgToAck, int idToSend, int msgIdToAck);
00061     void meshSend(packet sendPacket);
00062     int getmyID();
00063     void routingPacket();
00064     void updateRoutingTable(int id, unsigned short int size);
00065     void printRoutingTable();
00066     template <typename T> void debugPrint(T input);
00067     template <typename T> void debugPrintln(T input);
00068     void debugPrintln();
00069     void readMsgCallback(std::function<void (int, int, String)> callbackData);
00070     void readAckCallback(std::function<void (int, int)> callbackAck);
00071 };
00072
00073 #endif
```

Index