IoT Device Localization with ESP8266 and RFM69 Code Explanation_v1

The code was developed using the Arduino IDE platform, comprising three firmware programs: two for the reference node and one for the target node. The reference node includes two separate codes, one for the ATmega328P microcontroller and another for the ESP8266 (ESP12F) module, ensuring robust communication and data processing capabilities. The target node utilizes a single code designed for the ATmega328P microcontroller. Below is a detailed description of each code, followed by the complete code for all three components.

A. Reference node code

❖ Atmega328p

This Arduino code sets up an RFM69 wireless transceiver module for secure communication in a network. It uses the RFM69 and SPI libraries to control the transceiver and establish SPI communication. A SoftwareSerial instance is created to enable additional serial communication on pins 8 (RX) and 9 (TX). The **network configuration** is defined with unique NETWORKID, MYNODEID, and TONODEID values to identify the nodes. The module operates at the frequency specified by the FREQUENCY definition (e.g., 433 MHz), and **AES encryption** is enabled using a 16-character key, TOPSECRETPASSWRD, for secure message transmission. LED indicators are configured on pins 5 and 6 to provide visual feedback for communication activities. Additionally, the code initializes timing variables to handle LED state updates without blocking the main program. Below show the step-by-step development with a detailed description:

```
#include <RFM69.h>

#include <SPI.h>

#include <SoftwareSerial.h>
SoftwareSerial mySerial(8, 9); // RX, TX

// Addresses for this node. CHANGE THESE FOR EACH NODE!
#define NETWORKID     0   // Must be the same for all nodes
#define MYNODEID      1   // My node ID
#define TONODEID      4   // Destination node ID

// RFM69 frequency, uncomment the frequency of your module:
#define FREQUENCY   RF69_433MHZ
//#define FREQUENCY     RF69_915MHZ

// AES encryption (or not):
#define ENCRYPT       true // Set to "true" to use encryption
//#define ENCRYPT       false // Set to "true" to use encryption
#define ENCRYPTKEY    "TOPSECRETPASSWRD" // Use the same 16-byte key on all
nodes

// Use ACKnowledge when sending messages (or not):
#define USEACK        true // Request ACKs or not

// Packet sent/received indicator LED (optional):
#define LED_GREEN            5 // LED Green
```

```
#define LED_BLUE             6 // LED Blue

// Create a library object for our RFM69HCW module:
RFM69 radio;

int ledState = LOW;                // ledState used to set the LED
unsigned long previousMillis = 0; // will store last time LED was updated
// constants won't change:
const long interval = 10;
```

The setup function initializes the hardware and software components necessary for the Arduino to communicate using the RFM69 module and LEDs for status indication. Here's a breakdown:

1. **Serial Communication Setup**:

   The Serial port is initialized at a baud rate of 9600 for debugging and status updates, displaying the node's ID and readiness status. Additionally, mySerial is initialized to facilitate communication over software-based serial pins.

2. **LED Configuration**:

   Two LEDs are configured on pins defined by LED_GREEN and LED_BLUE. These are set as outputs and initialized to the LOW state, ensuring they are off at startup.

3. **RFM69 Module Initialization**:

   The radio.initialize function sets up the RFM69 module with the defined frequency, node ID, and network ID. For RFM69HCW modules, the radio.setHighPower() method is called to enable high-power mode for extended range.

4. **Encryption Activation**:

   If encryption is enabled (ENCRYPT is true), the AES encryption key (ENCRYPTKEY) is applied to secure communication.

This setup ensures the system is ready for secure wireless communication and provides visual indicators for system status.

```
void setup()
{
  Serial.begin(9600);
  Serial.print("Node ");
  Serial.print(MYNODEID, DEC);
  Serial.println(" ready");

  mySerial.begin(9600);
  mySerial.println("0");


  // Set up the indicator LED (optional):

  pinMode(LED_GREEN, OUTPUT);
  digitalWrite(LED_GREEN, LOW);
  pinMode(LED_BLUE, OUTPUT);
  digitalWrite(LED_BLUE, LOW);


  // Initialize the RFM69HCW:
  radio.initialize(FREQUENCY, MYNODEID, NETWORKID);
  radio.setHighPower(); // Always use this for RFM69HCW

  // Turn on encryption if desired:
  if (ENCRYPT)
    radio.encrypt(ENCRYPTKEY);
}
```

The loop function handles sending and receiving messages via the RFM69 module, as well as blinking an indicator LED. Below is the detailed breakdown and explanation:

1. Buffer Initialization

The code begins by initializing a static character array sendbuffer with a size of 62, and an integer sendlength set to 0. These variables serve as a temporary storage area for characters that are read from the serial port. Since the buffer is declared as static, it retains its value across iterations of the loop, ensuring no data is lost between executions of the function.

```
  static char sendbuffer[62];
  static int sendlength = 0;
```

2. Sending Messages

a. Gathering Input

This section handles the input from the serial port. When a character is received, it is checked to determine if it is a carriage return ('\r'). If not, the character is stored in the sendbuffer, and the length counter (sendlength) is incremented. This process allows the program to collect a stream of characters into the buffer.

b. Triggering the Message Send

The buffer contents are sent as a message when either a carriage return is detected or the buffer becomes full (containing 61 characters). These triggers ensure that the message is transmitted efficiently without overloading the buffer.

c. Sending Process

The message is transmitted to the destination node (TONODEID). The code uses two different methods to send the message based on the USEACK setting:

- **With Acknowledgment**: If USEACK is enabled, the sendWithRetry function attempts to send the message and waits for an acknowledgment. If successful, it displays "ACK received"; otherwise, it notifies the user of the failure.

- **Without Acknowledgment**: If acknowledgment is not required, the send function transmits the message directly. This method is faster but less reliable.

d. Resetting and Feedback

Once the message is sent, the buffer (sendlength) is reset to 0, preparing it for new input. Additionally, a visual confirmation is provided by blinking the green LED (LED_GREEN), signaling that the message has been sent successfully.

```
// SENDING
// In this section, we'll gather serial characters and
// send them to the other node if we (1) get a carriage return,
// or (2) the buffer is full (61 characters).

// If there is any serial input, add it to the buffer:

if (Serial.available() > 0)
{
  char input = Serial.read();
  if (input != '\r') // not a carriage return
  {
    sendbuffer[sendlength] = input;
    sendlength++;
  }

  // If the input is a carriage return, or the buffer is full:

  if ((input == '\r') || (sendlength == 61)) // CR or buffer full
  {
    // Send the packet!
    Serial.print("sending to node ");
    Serial.print(TONODEID, DEC);
    Serial.print(", message [");
    for (byte i = 0; i < sendlength; i++)
    Serial.print(sendbuffer[i]);
    Serial.println("]");

    // There are two ways to send packets. If you want
    // acknowledgements, use sendWithRetry():

    if (USEACK)
    {
      if (radio.sendWithRetry(TONODEID, sendbuffer, sendlength))
        Serial.println("ACK received!");
      else
        Serial.println("no ACK received");
    }

    // If you don't need acknowledgements, just use send():

    else // don't use ACK
    {
      radio.send(TONODEID, sendbuffer, sendlength);
    }

    sendlength = 0; // reset the packet
```

```
      Blink(LED_GREEN, 10);
    }
  }
```

3. Receiving Messages

The program continuously checks if the RFM69 module has received a new message using the radio.receiveDone() function. If a message is available, the following actions are performed:

- **Displaying Message Details**: The sender's ID and the message content are extracted and displayed on the serial monitor. The data is retrieved from the DATA array, and its length is determined by DATALEN.
- **Displaying Signal Strength**: The RSSI (Receive Signal Strength Indicator) value is printed. This value provides an indication of the signal's power, with smaller numbers indicating stronger signals.
- **Acknowledging the Sender**: If the received message requests acknowledgment, the code sends a response using the sendACK function. This ensures reliable communication and notifies the sender that their message was successfully received.
  Finally, the green LED blinks to confirm the successful receipt of the message.

```
 // RECEIVING

  // In this section, we'll check with the RFM69HCW to see
  // if it has received any packets:

  if (radio.receiveDone()) // Got one!
  {
    // Print out the information:

    Serial.print("received from node ");
    Serial.print(radio.SENDERID, DEC);
    Serial.print(", message [");

    // The actual message is contained in the DATA array,
    // and is DATALEN bytes in size:

    for (byte i = 0; i < radio.DATALEN; i++)
      Serial.print((char)radio.DATA[i]);

    // RSSI is the "Receive Signal Strength Indicator",
    // smaller numbers mean higher power.

    Serial.print("], RSSI ");
    Serial.println(radio.RSSI);
    mySerial.println(radio.RSSI);


    // Send an ACK if requested.
    // (You don't need this code if you're not using ACKs.)

    if (radio.ACKRequested())
    {
      radio.sendACK();
      Serial.println("ACK sent");
    }
    Blink(LED_GREEN, 10);
  }
```

4. LED Blinking Indicator

This section implements a blinking mechanism for the blue LED (LED_BLUE) as a heartbeat indicator. Using the millis() function, the program calculates the time elapsed since the last blink. If the elapsed time exceeds the predefined interval (10 ms), the LED's state is toggled between ON and OFF. This functionality ensures the LED blinks at consistent intervals, providing a visual cue that the program is running as expected.

```
 unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval)
  {
    previousMillis = currentMillis; // save the last time you blinked the LED
    // if the LED is off turn it on and vice-versa:
    if (ledState == LOW) {
      ledState = HIGH;
    } else {
      ledState = LOW;
    }
    // set the LED with the ledState of the variable:
    digitalWrite(LED_BLUE, ledState);
  }
```

The Blink function is a utility designed to create a simple visual indicator using an LED. It is especially useful for signaling specific events, such as successful message transmissions or receptions, in the main loop.

In the main program, the Blink function is called whenever:

- A message is successfully sent or received.
- An acknowledgment (ACK) is sent or received.

By providing visual confirmation, the Blink function helps in monitoring the system's operation without relying solely on the serial monitor, making debugging and system validation easier in real-world scenarios.

```
void Blink(byte PIN, int DELAY_MS)
// Blink an LED for a given number of ms
{
  digitalWrite(PIN, HIGH);
  delay(DELAY_MS);
  digitalWrite(PIN, LOW);
}
```

The complete code is provided below: **(Atmeg328p reference nodes)**

```
// Include the RFM69 and SPI libraries:
#include <RFM69.h>
#include <SPI.h>

#include <SoftwareSerial.h>
SoftwareSerial mySerial(8, 9); // RX, TX

// Addresses for this node. CHANGE THESE FOR EACH NODE!
#define NETWORKID     0   // Must be the same for all nodes
#define MYNODEID      1   // My node ID
#define TONODEID      4   // Destination node ID

// RFM69 frequency, uncomment the frequency of your module:
#define FREQUENCY   RF69_433MHZ
//#define FREQUENCY     RF69_915MHZ

// AES encryption (or not):
#define ENCRYPT       true // Set to "true" to use encryption
//#define ENCRYPT       false // Set to "true" to use encryption
#define ENCRYPTKEY    "TOPSECRETPASSWRD" // Use the same 16-byte key on all
nodes

// Use ACKnowledge when sending messages (or not):
#define USEACK        true // Request ACKs or not

// Packet sent/received indicator LED (optional):
#define LED_GREEN           5 // LED Green
#define LED_BLUE            6 // LED Blue

// Create a library object for our RFM69HCW module:
RFM69 radio;

int ledState = LOW;               // ledState used to set the LED
unsigned long previousMillis = 0; // will store last time LED was updated
// constants won't change:
const long interval = 10;



void setup()
{
  // Open a serial port so we can send keystrokes to the module:

  Serial.begin(9600);
  Serial.print("Node ");
  Serial.print(MYNODEID, DEC);
  Serial.println(" ready");


  mySerial.begin(9600);
  mySerial.println("0");


  // Set up the indicator LED (optional):
```

```
  pinMode(LED_GREEN, OUTPUT);
  digitalWrite(LED_GREEN, LOW);
  pinMode(LED_BLUE, OUTPUT);
  digitalWrite(LED_BLUE, LOW);


  // Initialize the RFM69HCW:
  radio.initialize(FREQUENCY, MYNODEID, NETWORKID);
  radio.setHighPower(); // Always use this for RFM69HCW

  // Turn on encryption if desired:
  if (ENCRYPT)
    radio.encrypt(ENCRYPTKEY);
}

void loop()
{
  // Set up a "buffer" for characters that we'll send:
  static char sendbuffer[62];
  static int sendlength = 0;

  // SENDING
  // In this section, we'll gather serial characters and
  // send them to the other node if we (1) get a carriage return,
  // or (2) the buffer is full (61 characters).

  // If there is any serial input, add it to the buffer:

  if (Serial.available() > 0)
  {
    char input = Serial.read();
    if (input != '\r') // not a carriage return
    {
      sendbuffer[sendlength] = input;
      sendlength++;
    }

    // If the input is a carriage return, or the buffer is full:

    if ((input == '\r') || (sendlength == 61)) // CR or buffer full
    {
      // Send the packet!
      Serial.print("sending to node ");
      Serial.print(TONODEID, DEC);
      Serial.print(", message [");
      for (byte i = 0; i < sendlength; i++)
      Serial.print(sendbuffer[i]);
      Serial.println("]");

      // There are two ways to send packets. If you want
      // acknowledgements, use sendWithRetry():

      if (USEACK)
      {
        if (radio.sendWithRetry(TONODEID, sendbuffer, sendlength))
          Serial.println("ACK received!");
        else
          Serial.println("no ACK received");
      }

      // If you don't need acknowledgements, just use send():

      else // don't use ACK
      {
        radio.send(TONODEID, sendbuffer, sendlength);
      }

      sendlength = 0; // reset the packet
      Blink(LED_GREEN, 10);
    }
  }

  // RECEIVING

  // In this section, we'll check with the RFM69HCW to see
  // if it has received any packets:

  if (radio.receiveDone()) // Got one!
  {
    // Print out the information:

    Serial.print("received from node ");
    Serial.print(radio.SENDERID, DEC);
    Serial.print(", message [");

    // The actual message is contained in the DATA array,
    // and is DATALEN bytes in size:

    for (byte i = 0; i < radio.DATALEN; i++)
      Serial.print((char)radio.DATA[i]);

    // RSSI is the "Receive Signal Strength Indicator",
    // smaller numbers mean higher power.

    Serial.print("], RSSI ");
    Serial.println(radio.RSSI);
    mySerial.println(radio.RSSI);


    // Send an ACK if requested.
    // (You don't need this code if you're not using ACKs.)

    if (radio.ACKRequested())
    {
      radio.sendACK();
      Serial.println("ACK sent");
    }
    Blink(LED_GREEN, 10);
  }
  //-----------------------------------------------------------
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval)
  {
    previousMillis = currentMillis; // save the last time you blinked the LED
```

```
    // if the LED is off turn it on and vice-versa:
    if (ledState == LOW) {
      ledState = HIGH;
    } else {
      ledState = LOW;
    }
    // set the LED with the ledState of the variable:
    digitalWrite(LED_BLUE, ledState);
  }
}

void Blink(byte PIN, int DELAY_MS)
// Blink an LED for a given number of ms
{
  digitalWrite(PIN, HIGH);
  delay(DELAY_MS);
  digitalWrite(PIN, LOW);
}
```

❖ **Esp8266 code**

This code show the setup of an ESP8266 microcontroller to connect to a WiFi network and send data to a ThingSpeak channel. ThingSpeak is an IoT analytics platform that allows users to collect, visualize, and analyze data. The key parts of this code are:

1. **WiFi Connection Setup**: The #include <ESP8266WiFi.h> library allows the ESP8266 to connect to a WiFi network. The ssid and pass variables store the WiFi network name and password, respectively. These values must be replaced with the actual network credentials.

2. **ThingSpeak Integration**: The ThingSpeak.h library is included to enable communication with ThingSpeak. The myChannelNumber and myWriteAPIKey variables define the target channel and its API key, which are essential for authenticating and sending data.

3. **LED State and Pin Configuration**: The ledPin is assigned to the built-in LED of the ESP8266. The ledState variable keeps track of whether the LED is ON or OFF. This will help in signaling specific states visually.

4. **Timer Variables**: The lastTime and timerDelay variables are used to implement periodic updates. The timerDelay is set to 30,000 milliseconds (30 seconds), meaning the ESP8266 will attempt to send data to ThingSpeak at this interval.

5. **Random Value Generation**: The code initializes placeholders (Value_1, Value_2, randNumber_1, randNumber_2) for generating random data, which can later be sent to ThingSpeak.

6. **Input Handling**: The inString variable stores incoming data from the serial interface, which can be processed or sent to ThingSpeak.

This is a foundational setup code that ensures your ESP8266 is ready to collect and send data. The section before the **setup()** function is called the **global declarations section**. It is where you include the required libraries, define constants and macros, and declare global variables that will be used throughout the program. This section ensures that all necessary components, such as pin numbers, configuration settings, and objects for hardware or libraries, are prepared before the main program starts executing. It acts as the foundation for the rest of the code.

```
#include <ESP8266WiFi.h>
#include "ThingSpeak.h"

const char *ssid =  "Your wifi Network name";     // replace with your wifi
ssid and wpa2 key
const char *pass =  "Network password";
WiFiClient client;

unsigned long lastTime = 0;
unsigned long timerDelay = 30000;

unsigned long myChannelNumber = 1;
const char *myWriteAPIKey = "1GR99RDKT33TKGS0";

const int ledPin =  LED_BUILTIN;// the number of the LED pin
int ledState = LOW;             // ledState used to set the LED
float Value_1;
float Value_2;
long randNumber_1;
long randNumber_2;

String inString = "";    // string to hold input
```

The setup() function initializes the ESP8266 module, establishes a WiFi connection, prepares the ThingSpeak library, and provides visual and textual feedback to indicate the status of the operations. Here's a breakdown of the key sections with their explanation:

1. **Configuring the LED Pin and Serial Communication**

```
pinMode(ledPin, OUTPUT);   // Set LED pin as output
Serial.begin(9600);        // Start serial communication
delay(10);
```

   - The LED pin is set as an output to control the onboard LED.
   - The Serial.begin(9600) initializes serial communication at a baud rate of 9600, which is used to send and receive data via the serial monitor.
   - A short delay (delay(10)) allows the microcontroller to stabilize before proceeding with other operations.

2. **Printing WiFi Connection Details**

```
Serial.println("Connecting to ");  // Print WiFi connection status
Serial.println(ssid);             // Print SSID
```

- These lines print a message to the serial monitor indicating the start of the WiFi connection process.
- The SSID (network name) is displayed, allowing the user to confirm the network being connected to.

3. **Initiating WiFi and ThingSpeak**

```
WiFi.begin(ssid, pass);     // Start WiFi connection

ThingSpeak.begin(client);   // Initialize ThingSpeak
```

- The WiFi.begin() function attempts to connect to the WiFi network using the provided SSID and password.
- The ThingSpeak.begin(client) initializes the ThingSpeak library, preparing it for data transmission to the ThingSpeak server.

4. **WiFi Connection Loop with Visual Feedback**

```
while (WiFi.status() != WL_CONNECTED)    // Wait for WiFi to connect
  {
   ledState = ~ledState;                // Toggle LED state
   digitalWrite(ledPin, ledState);      // Blink LED
   delay(500);                          // Wait for 500ms
   Serial.print(".");                   // Print progress dot
  }
```

- The while loop continuously checks the WiFi connection status using WiFi.status(). If not connected, it remains in the loop.
- Inside the loop, the LED toggles between ON and OFF to indicate that the ESP8266 is attempting to connect.
- A delay of 500 ms provides a visible blinking effect for the LED, while a dot (".") is printed to the serial monitor to show progress.

5. **Connection Confirmation**

```
Serial.println("WiFi connected");   // Print WiFi connected message
```

- Once connected, a message confirming the connection is printed to the serial monitor.

6. **Swapping TX/RX Pins**

```
Serial.swap();                  // Swap TX/RX pins for serial
```

- The Serial.swap() function swaps the default transmit (TX) and receive (RX) pins to alternate pins, useful in some hardware configurations.

The complete code is provided below for the setup.

```
void setup()
{
  pinMode(ledPin, OUTPUT);    // Set LED pin as output
  Serial.begin(9600);         // Start serial communication
  delay(10);                  // Short delay for stabilization

  Serial.println("Connecting to ");   // Print WiFi connection status
  Serial.println(ssid);              // Print SSID

  WiFi.begin(ssid, pass);     // Start WiFi connection

  ThingSpeak.begin(client);   // Initialize ThingSpeak

  while (WiFi.status() != WL_CONNECTED)    // Wait for WiFi to connect
   {
    ledState = ~ledState;              // Toggle LED state
    digitalWrite(ledPin, ledState);    // Blink LED
    delay(500);                        // Wait for 500ms
    Serial.print(".");                 // Print progress dot
   }
  Serial.println("");             // New line after connection
  Serial.println("WiFi connected");   // Print WiFi connected message

  Serial.swap();                // Swap TX/RX pins for serial
}
```

The loop() function contains the main program logic that continuously executes, performing tasks such as controlling the onboard LED, maintaining the WiFi connection, sending data to ThingSpeak, and handling serial communication. Let's break it into sections for better understanding:

1. **LED Blinking for Visual Feedback**

```
  digitalWrite(ledPin, HIGH);   // Turn the LED on
  delay(1000);                  // Wait for 1 second
  digitalWrite(ledPin, LOW);    // Turn the LED off
  delay(1000);                  // Wait for 1 second
```

- The LED is turned ON and OFF alternately with a 1-second delay between each state. This provides a simple visual indicator that the system is running.

2. **Checking Timer for Periodic Execution**

```
 if ((millis() - lastTime) > timerDelay)  // Check if it's time to execute
the next cycle
  {
    Serial.println("#");
```

- The millis() function is used to measure elapsed time since the program started. When the difference between the current time and lastTime exceeds timerDelay (30 seconds), the following operations are executed.
- The Serial.println("#"); provides a marker in the serial monitor for tracking periodic activity.

3. **WiFi Connection Maintenance**

```
   if(WiFi.status() != WL_CONNECTED)   // Check and attempt to reconnect to
WiFi if disconnected
    {
      Serial.print("Attempting to connect");
      WiFi.begin(ssid, pass);
      while(WiFi.status() != WL_CONNECTED)
      {
        ledState = ~ledState;
        digitalWrite(ledPin, ledState);
        delay(5000);
        Serial.print("+");
      }
      delay(500);
      Serial.println("\nConnected."); // Indicate successful connection
      ThingSpeak.begin(client);       // Initialize ThingSpeak
    }
```

- If the WiFi connection is lost, the system attempts to reconnect using WiFi.begin().
- During reconnection, the LED blinks rapidly (every 5 seconds) to indicate the connection attempt, and "+" is printed to the serial monitor to track the progress.
- Once reconnected, ThingSpeak.begin(client) is called again to ensure communication with ThingSpeak is re-established.

4. **Updating the Last Execution Time**

```
lastTime = millis();  // Update the last time marker
```

- This line updates the lastTime variable to the current time, ensuring the timer cycle restarts.

5. **Reading and Sending Data**

```
 if (Serial.available() > 0)
  {
      inString = Serial.readStringUntil('\n');  // Read the incoming string 
from Serial until a newline character

      // Write to ThingSpeak. There are up to 8 fields in a channel, allowing 
you to store up to 8 different
      // pieces of information in a channel.
      ThingSpeak.setField(1, inString);  //Here, we write to field 1.
      int x = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey); // Send 
data to the ThingSpeak channel
```

- When data is sent via the serial port, it is read using Serial.readStringUntil('\n') until a newline character.
- This data is then written to Field 1 of the ThingSpeak channel using ThingSpeak.setField().

6. **Handling ThingSpeak Response**

```
     if(x == 200){  // checks if the HTTP response code is 200, which 
indicates the request was successful (standard HTTP success status)
        Serial.println("Channel update successful.");  // Confirm successful 
data upload
      }
      else{
        Serial.println("Problem updating channel. HTTP error code " + 
String(x)); // Print error if upload fails
    }
```

- The variable x stores the HTTP response code from ThingSpeak. A code of 200 indicates success.
- If the update fails, the error code is printed to the serial monitor for debugging.

The complete code is provided below: (ESP8266)

```
void loop()
{
  digitalWrite(ledPin, HIGH);  // Turn the LED on
  delay(1000);                 // Wait for 1 second
  digitalWrite(ledPin, LOW);   // Turn the LED off
  delay(1000);                 // Wait for 1 second
  if ((millis() - lastTime) > timerDelay)  // Check if it's time to execute
the next cycle
  {
    Serial.println("#");
    // Connect or reconnect to WiFi
    if(WiFi.status() != WL_CONNECTED)   // Check and attempt to reconnect to
WiFi if disconnected
    {
      Serial.print("Attempting to connect");
      WiFi.begin(ssid, pass);
      while(WiFi.status() != WL_CONNECTED)
      {
        ledState = ~ledState;
        digitalWrite(ledPin, ledState);
        delay(5000);
        Serial.print("+");
      }
      delay(500);
      Serial.println("\nConnected."); // Indicate successful connection
      ThingSpeak.begin(client);       // Initialize ThingSpeak
    }
    lastTime = millis();  // Update the last time marker
  }

  if (Serial.available() > 0)
  {
      inString = Serial.readStringUntil('\n');  // Read the incoming string
from Serial until a newline character

      // Write to ThingSpeak. There are up to 8 fields in a channel, allowing
you to store up to 8 different
      // pieces of information in a channel.
      ThingSpeak.setField(1, inString);  //Here, we write to field 1.
      int x = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey); // Send
data to the ThingSpeak channel

      if(x == 200){  // checks if the HTTP response code is 200, which
indicates the request was successful (standard HTTP success status)
        Serial.println("Channel update successful.");  // Confirm successful
data upload
      }
      else{
        Serial.println("Problem updating channel. HTTP error code " +
String(x)); // Print error if upload fails
    }
      }
}
```

B. Target node code

This code sets up a basic communication framework using the **RFM69HCW** radio module as the target node, which is used for communication in IoT and sensor network applications. The program includes key configurations and features to enable effective data transmission between nodes in a network.

1. **Library Inclusion and Setup**

   The code starts by including the **RFM69** and **SPI** libraries, essential for interfacing with the RFM69 radio module. These libraries handle the communication protocols needed for data transmission.

```
#include <RFM69.h>
#include <SPI.h>
```

2. **Network and Node Configuration**

   Each node in the network is assigned a unique ID and network ID to differentiate between devices.

```
#define NETWORKID     0   // Must be the same for all nodes
#define MYNODEID      4   // My node ID
byte TONODEID    =  0;   // Destination node ID
```

3. **Frequency Selection**

   The RFM69 module can operate on different frequencies (433 MHz or 915 MHz). The correct frequency must be chosen based on the hardware.

```
#define FREQUENCY   RF69_433MHZ
```

4. **Encryption**

   Data security is enhanced using AES encryption. The same encryption key (ENCRYPTKEY) must be set across all nodes for successful communication.

```
#define ENCRYPT       true // Set to "true" to use encryption
//#define ENCRYPT       false // Set to "true" to use encryption
#define ENCRYPTKEY    "TOPSECRETPASSWRD" // Use the same 16-byte key on all
nodes
```

5. **Acknowledgment Setting**

   The code includes an option to enable or disable acknowledgment (ACK) for sent messages, ensuring reliable communication by verifying message delivery.

```
#define USEACK        true // Request ACKs or not
```

6. **LED Indicators**

   LEDs are configured to provide visual feedback for communication status. For example, red, green, and blue LEDs can indicate errors, successful transmissions, or standby mode.

```
#define LED_RED             4 // LED positive pin
#define LED_GREEN           5 // LED positive pin
```

```
#define LED_BLUE             6 // LED ground pin
```

7. **Radio Object Creation**

   An RFM69 object is created to manage communication with the radio module. Timing variables (previousMillis, interval) help manage periodic tasks like sending data or toggling LEDs.

```
RFM69 radio;
```

8. **Send Data Flag**

   The Send_Data_Flag variable is used to signal when data should be sent, supporting efficient message handling in the loop.

```
byte Send_Data_Flag = 0;
```

The complete code is provided below for the global declarations section:

```
// Include the RFM69 and SPI libraries:
#include <RFM69.h>
#include <SPI.h>

// Addresses for this node. CHANGE THESE FOR EACH NODE!

#define NETWORKID     0   // Must be the same for all nodes
#define MYNODEID      4   // My node ID
byte TONODEID    =  0;   // Destination node ID

// RFM69 frequency, uncomment the frequency of your module:
#define FREQUENCY   RF69_433MHZ
//#define FREQUENCY     RF69_915MHZ

// AES encryption (or not):

#define ENCRYPT       true // Set to "true" to use encryption
//#define ENCRYPT       false // Set to "true" to use encryption
#define ENCRYPTKEY    "TOPSECRETPASSWRD" // Use the same 16-byte key on all
nodes

// Use ACKnowledge when sending messages (or not):
#define USEACK        true // Request ACKs or not

// Packet sent/received indicator LED (optional):
#define LED_RED              4 // LED positive pin
#define LED_GREEN            5 // LED positive pin
#define LED_BLUE             6 // LED ground pin

// Create a library object for our RFM69HCW module:

RFM69 radio;

int ledState = LOW;             // ledState used to set the LED
unsigned long previousMillis = 0;        // will store last time LED was
updated
```

```
unsigned long previousMillis_data = 0;
// constants won't change:
const long interval = 3000;
const long interval_data = 100;
byte Send_Data_Flag = 0;
```

The setup function initializes the system components and prepares the RFM69HCW module for communication. This is a critical part of the program where the hardware and software configurations are established.

1. **Serial Communication Setup**

   The code begins by initializing serial communication at a baud rate of 9600. This allows the node to communicate with a computer for debugging or monitoring purposes. The node ID is displayed using the Serial.print() function for identification.

```
  Serial.begin(9600);
  Serial.print("Node ");
  Serial.print(MYNODEID,DEC);
  Serial.println(" ready");
```

2. **LED Configuration**

   Two LEDs (green and blue) are configured as output indicators. Initially, they are set to LOW, ensuring they remain off until triggered. These LEDs can visually signal the system's status or activities like data transmission or errors.

```
  pinMode(LED_GREEN,OUTPUT);
  digitalWrite(LED_GREEN,LOW);
  pinMode(LED_BLUE,OUTPUT);
  digitalWrite(LED_BLUE,LOW);
```

3. **RFM69HCW Initialization**

   The RFM69HCW module is initialized with the specified frequency, node ID, and network ID. This setup ensures the node is correctly configured for communication within the designated network. The setHighPower() function is called to enable high-power mode, which is necessary for the RFM69HCW module to operate at its full transmission capacity.

```
 radio.initialize(FREQUENCY, MYNODEID, NETWORKID);
  radio.setHighPower(); // Always use this for RFM69HCW
```

4. **Encryption Activation**

   If encryption is enabled (ENCRYPT is set to true), the specified encryption key is applied to secure communication between nodes. Encryption ensures that transmitted data is protected from unauthorized access.

```
 if (ENCRYPT)
    radio.encrypt(ENCRYPTKEY);
```

This setup ensures that the node is ready for secure and efficient communication within the RFM69HCW network. The use of LEDs provides visual feedback, while the configuration of high-power mode and encryption optimizes performance and security.

The complete code is provided below for setup

```
void setup()
{
  Serial.begin(9600);
  Serial.print("Node ");
  Serial.print(MYNODEID,DEC);
  Serial.println(" ready");

  // Set up the indicator LED (optional):
  pinMode(LED_GREEN,OUTPUT);
  digitalWrite(LED_GREEN,LOW);
  pinMode(LED_BLUE,OUTPUT);
  digitalWrite(LED_BLUE,LOW);

  // Initialize the RFM69HCW:
  radio.initialize(FREQUENCY, MYNODEID, NETWORKID);
  radio.setHighPower(); // Always use this for RFM69HCW

  // Turn on encryption if desired:
  if (ENCRYPT)
    radio.encrypt(ENCRYPTKEY);
}
```

The loop() function in this program manages the continuous operation of sending and receiving messages between nodes using the RFM69HCW module. It also includes a heartbeat indicator for system status monitoring. This function ensures seamless communication by handling data transmission and reception while maintaining network stability.

1. **Sending Messages**

   The **SENDING** section constructs and sends packets to another node. A buffer (sendbuffer) holds characters to be sent. When the Send_Data_Flag is set to 1, a message is prepared and sent to the target node (TONODEID).

   - **Sending with Acknowledgements:** The sendWithRetry() function ensures the message is delivered, retrying if no acknowledgment is received. If acknowledgments are not required, the send() function directly transmits the data.

```
if (USEACK)
 {
   if (radio.sendWithRetry(TONODEID, sendbuffer, sendlength))
     Serial.println("ACK received!");
```

```
      else
        Serial.println("no ACK received");
    }
```

After sending, the buffer is reset, and an LED blinks briefly to indicate a successful transmission.

2. **Receiving Messages**

The radio.receiveDone() function is the core of the receiving mechanism in the RFM69 communication module. When a packet is successfully received, the following operations take place:

Printing Sender Information

The sender node's ID (radio.SENDERID) is identified and printed on the serial monitor. This allows users to know the source of the received message.

```
Serial.print("received from node ");
 Serial.print(radio.SENDERID, DEC);
 Serial.print(", message [");
```

Extracting and Printing the Message

The received message is stored in the radio.DATA array, with its size indicated by radio.DATALEN. Each byte of the data is converted to a character and displayed, showing the full message content in the serial monitor.

```
for (byte i = 0; i < radio.DATALEN; i++)
      Serial.print((char)radio.DATA[i]);
```

RSSI Value Reporting

The Received Signal Strength Indicator (RSSI) is also displayed. This value provides insight into the signal's power: lower RSSI values indicate stronger signals. Monitoring the RSSI can help evaluate the quality of communication between nodes.

```
Serial.print("], RSSI ");
 Serial.println(radio.RSSI);
```

Acknowledgment Handling

If the sender node has requested an acknowledgment (ACK), the receiver responds by sending an ACK using radio.sendACK(). This ensures reliable data exchange, especially in applications requiring confirmation of message receipt.

```
if (radio.ACKRequested())
 {
   radio.sendACK();
   Serial.println("ACK sent");
 }
```

Visual Indicator

Upon successful receipt and processing of a message, a green LED blinks briefly to indicate successful communication, providing visual feedback for debugging or system monitoring.

```
Blink(LED_GREEN,300);
```

This segment of code ensures that incoming data is efficiently processed, relevant information is displayed, and acknowledgments are handled for reliable communication. The use of RSSI reporting and visual indicators enhances the system's usability for real-time monitoring and debugging.

3. **Heartbeat LED Indicator**

   A blue LED blinks periodically as a "heartbeat" indicator, signifying the system is operational. The timing for the blink is controlled using millis() to ensure non-blocking operation.

```
if (currentMillis - previousMillis >= interval)
 {
   // save the last time you blinked the LED
   previousMillis = currentMillis;
   digitalWrite(LED_BLUE, HIGH);
   delay(10);
   digitalWrite(LED_BLUE, LOW);
 }
```

4. **Periodic Data Transmission**

   A flag (Send_Data_Flag) is toggled periodically to initiate data transmission. The TONODEID cycles through available nodes to ensure each one receives data in turn.

```
if (currentMillis - previousMillis_data >= interval_data)
 {
   previousMillis_data = currentMillis;
   Send_Data_Flag = 1;
   if (TONODEID == 3)
   {
     TONODEID = 0;
   }
   TONODEID++;
 }
```

This structure ensures robust and efficient communication between nodes. It manages data flow, acknowledges successful transmission, and provides visual feedback for operational status.

The complete code is provided below: (Atmeg328p target node)

```
// Include the RFM69 and SPI libraries:
#include <RFM69.h>
#include <SPI.h>
```

```
// Addresses for this node. CHANGE THESE FOR EACH NODE!

#define NETWORKID     0   // Must be the same for all nodes
#define MYNODEID      4   // My node ID
byte TONODEID    =  0;   // Destination node ID

// RFM69 frequency, uncomment the frequency of your module:
#define FREQUENCY   RF69_433MHZ
//#define FREQUENCY     RF69_915MHZ

// AES encryption (or not):

#define ENCRYPT       true // Set to "true" to use encryption
//#define ENCRYPT       false // Set to "true" to use encryption
#define ENCRYPTKEY    "TOPSECRETPASSWRD" // Use the same 16-byte key on all
nodes

// Use ACKnowledge when sending messages (or not):
#define USEACK        true // Request ACKs or not

// Packet sent/received indicator LED (optional):
#define LED_RED             4 // LED positive pin
#define LED_GREEN           5 // LED positive pin
#define LED_BLUE            6 // LED ground pin

// Create a library object for our RFM69HCW module:

RFM69 radio;

int ledState = LOW;             // ledState used to set the LED
unsigned long previousMillis = 0;        // will store last time LED was
updated
unsigned long previousMillis_data = 0;
// constants won't change:
const long interval = 3000;
const long interval_data = 100;
byte Send_Data_Flag = 0;

void setup()
{
  Serial.begin(9600);
  Serial.print("Node ");
  Serial.print(MYNODEID,DEC);
  Serial.println(" ready");

  // Set up the indicator LED (optional):
  pinMode(LED_GREEN,OUTPUT);
  digitalWrite(LED_GREEN,LOW);
  pinMode(LED_BLUE,OUTPUT);
  digitalWrite(LED_BLUE,LOW);

  // Initialize the RFM69HCW:
  radio.initialize(FREQUENCY, MYNODEID, NETWORKID);
  radio.setHighPower(); // Always use this for RFM69HCW

  // Turn on encryption if desired:
```

```
  if (ENCRYPT)
    radio.encrypt(ENCRYPTKEY);
}

void loop()
{
  // Set up a "buffer" for characters that we'll send:
  static char sendbuffer[62];
  static int sendlength = 0;

  // SENDING
  // In this section, we'll gather serial characters and
  // send them to the other node if we (1) get a carriage return,
  // or (2) the buffer is full (61 characters).

  // If there is any serial input, add it to the buffer:

  if (Send_Data_Flag == 1)
  {
    // Send the packet!
    sendlength = 1;
    sendbuffer[0] = 0x41 ;

    Serial.print("sending to node ");
    Serial.print(TONODEID, DEC);
    Serial.print(", message [");
    for (byte i = 0; i < sendlength; i++)
      Serial.print(sendbuffer[i]);
    Serial.println("]");

    // There are two ways to send packets. If you want
    // acknowledgements, use sendWithRetry():

    if (USEACK)
    {
      if (radio.sendWithRetry(TONODEID, sendbuffer, sendlength))
        Serial.println("ACK received!");
      else
        Serial.println("no ACK received");
    }

    // If you don't need acknowledgements, just use send():

    else // don't use ACK
    {
      radio.send(TONODEID, sendbuffer, sendlength);
    }

    sendlength = 0; // reset the packet
    Send_Data_Flag = 0;
    Blink(LED_RED,10);
  }


  // RECEIVING
  // In this section, we'll check with the RFM69HCW to see
  // if it has received any packets:
```

```
  if (radio.receiveDone()) // Got one!
  {
    // Print out the information:
    Serial.print("received from node ");
    Serial.print(radio.SENDERID, DEC);
    Serial.print(", message [");

    // The actual message is contained in the DATA array,
    // and is DATALEN bytes in size:
    for (byte i = 0; i < radio.DATALEN; i++)
      Serial.print((char)radio.DATA[i]);

    // RSSI is the "Receive Signal Strength Indicator",
    // smaller numbers mean higher power.
    Serial.print("], RSSI ");
    Serial.println(radio.RSSI);

    // Send an ACK if requested.
    // (You don't need this code if you're not using ACKs.)
    if (radio.ACKRequested())
    {
      radio.sendACK();
      Serial.println("ACK sent");
    }
    Blink(LED_GREEN,300);
  }

unsigned long currentMillis = millis(); // HB LED
  if (currentMillis - previousMillis >= interval)
  {
    // save the last time you blinked the LED
    previousMillis = currentMillis;
    digitalWrite(LED_BLUE, HIGH);
    delay(10);
    digitalWrite(LED_BLUE, LOW);
  }

  if (currentMillis - previousMillis_data >= interval_data)  // DATA Tx
  {
    previousMillis_data = currentMillis;
    Send_Data_Flag = 1;
//    Serial.print("Send_Data_Flag = 1");
    if (TONODEID == 3)
    {
      TONODEID = 0;
    }
    TONODEID++;
  }
}

void Blink(byte PIN, int DELAY_MS)
// Blink an LED for a given number of ms
{
  digitalWrite(PIN,HIGH);
  delay(DELAY_MS);
  digitalWrite(PIN,LOW);
}
```