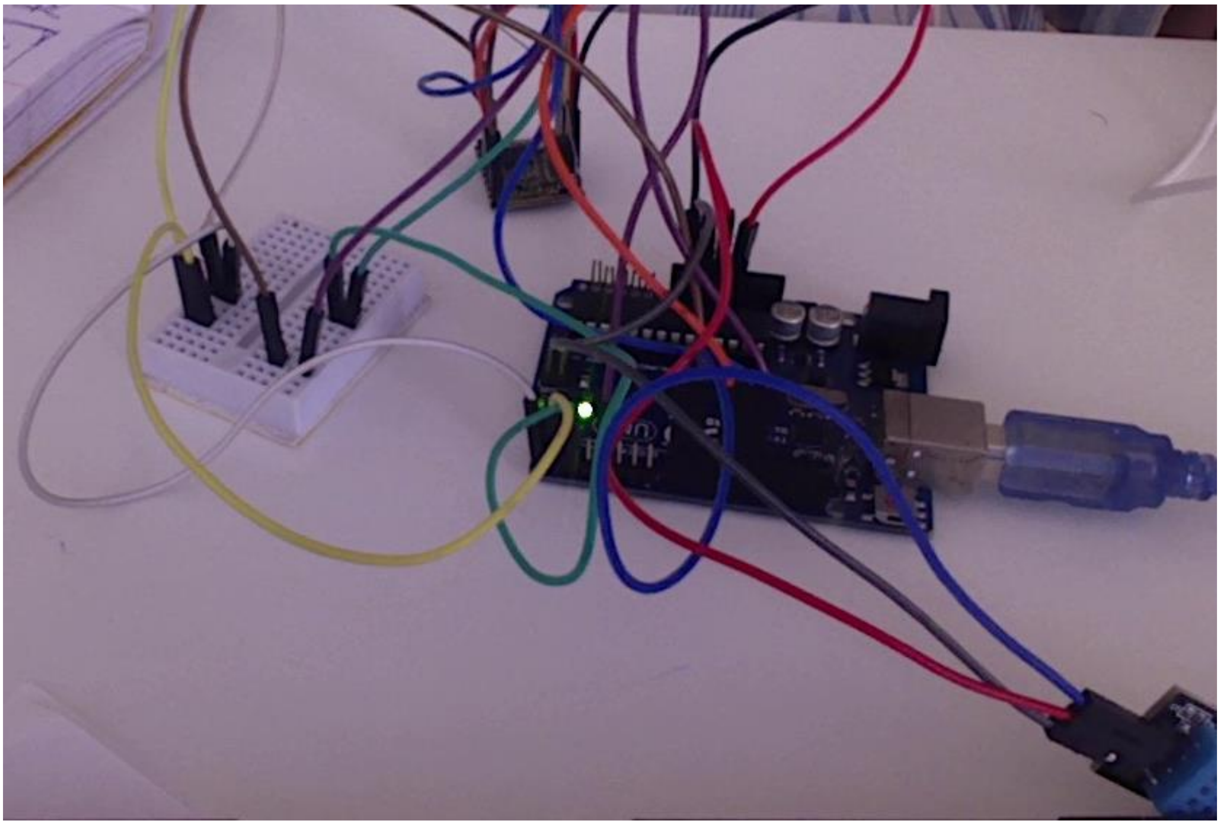

Figure 1: Schéma électrique

Table 1: Pin Mapping

Pin Module LoRa	Pin Arduino Uno	Pin Arduino Mega
ANT	[NC]	[NC]
GND	GND	GND
VSS	3.3V	3.3V
RX_Switch	1_TX	
TX_Switch	0_RX	
DIO0	2	
DIO[1…5]	[NC]	[NC]
RESET	9	
SPI SCKL	SPI 13	
MOSI	12	
MISO	11	
nSEL	4	

DHT11*	Pin Arduino Uno
(1)GND	GND
(2)VSS	5V
(3)Data	8

**Lecture de droit à gauche*

Le code Arduino correspondant : (Dht11_Client)

```
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>
#include <dht.h>
#define DHT11_PIN 8
dht DHT;


// LoRaWAN NwkSKey, network session key
// This is the default Semtech key, which is used by the early prototype TTN
// network.
static const PROGMEM u1_t NWKSKEY[16] = { 0x2B, 0x7E, 0x15, 0x16, 0x28,
0xAE, 0xD2, 0xA6, 0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C };

// LoRaWAN AppSKey, application session key
// This is the default Semtech key, which is used by the prototype TTN
// network initially.
static const u1_t PROGMEM APPSKEY[16] = { 0x2B, 0x7E, 0x15, 0x16, 0x28,
0xAE, 0xD2, 0xA6, 0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C };

// LoRaWAN end-device address (DevAddr)
// See http://thethingsnetwork.org/wiki/AddressSpace
static const u4_t DEVADDR = 0x03FF0001; // <-- Change this address for every
node!

// These callbacks are only used in over-the-air activation, so they are
// left empty here (we cannot leave them out completely unless
// DISABLE_JOIN is set in config.h, otherwise the linker will complain).
void os_getArtEui (u1_t* buf) { }
void os_getDevEui (u1_t* buf) { }
void os_getDevKey (u1_t* buf) { }

// static uint8_t mydata[] = "Hello, world!";
static osjob_t sendjob;

// Schedule TX every this many seconds (might become longer due to duty
// cycle limitations).
const unsigned TX_INTERVAL = 60;
```

```
// Pin mapping
const lmic_pinmap lmic_pins = {
    .nss = 4,
    .rxtx =1,
    .rst = 9,
    .dio = {2,LMIC_UNUSED_PIN ,LMIC_UNUSED_PIN},
};

void onEvent (ev_t ev) {
    Serial.print(os_getTime());
    Serial.print(": ");
    switch(ev) {
        case EV_SCAN_TIMEOUT:
            Serial.println(F("EV_SCAN_TIMEOUT"));
            break;
        case EV_BEACON_FOUND:
            Serial.println(F("EV_BEACON_FOUND"));
            break;
        case EV_BEACON_MISSED:
            Serial.println(F("EV_BEACON_MISSED"));
            break;
        case EV_BEACON_TRACKED:
            Serial.println(F("EV_BEACON_TRACKED"));
            break;
        case EV_JOINING:
            Serial.println(F("EV_JOINING"));
            break;
        case EV_JOINED:
            Serial.println(F("EV_JOINED"));
            break;
        case EV_RFU1:
            Serial.println(F("EV_RFU1"));
            break;
        case EV_JOIN_FAILED:
            Serial.println(F("EV_JOIN_FAILED"));
            break;
        case EV_REJOIN_FAILED:
            Serial.println(F("EV_REJOIN_FAILED"));
            break;
        case EV_TXCOMPLETE:
```

```
      Serial.println(F("EV_TXCOMPLETE (includes waiting for RX windows)"));
      if (LMIC.txrxFlags & TXRX_ACK)
       Serial.println(F("Received ack"));
      if (LMIC.dataLen) {
       Serial.println(F("Received "));
       Serial.println(LMIC.dataLen);
       Serial.println(F(" bytes of payload"));
      }
      // Schedule next transmission
      os_setTimedCallback(&sendjob,
os_getTime()+sec2osticks(TX_INTERVAL), do_send);
      break;
    case EV_LOST_TSYNC:
      Serial.println(F("EV_LOST_TSYNC"));
      break;
    case EV_RESET:
      Serial.println(F("EV_RESET"));
      break;
    case EV_RXCOMPLETE:
      // data received in ping slot
      Serial.println(F("EV_RXCOMPLETE"));
      break;
    case EV_LINK_DEAD:
      Serial.println(F("EV_LINK_DEAD"));
      break;
    case EV_LINK_ALIVE:
      Serial.println(F("EV_LINK_ALIVE"));
      break;
     default:
      Serial.println(F("Unknown event"));
      break;
  }
}

void do_send(osjob_t* j){
  byte buffer[8];
  int chk=DHT.read11(DHT11_PIN);
  float t = DHT.temperature;
  float h = DHT.humidity;
  dtostrf(t, 2, 1, buffer);
```

```
    dtostrf(h, 2, 1, &buffer[4]);
    String res = buffer;
    res.getBytes(buffer, res.length() + 1);
    Serial.println("");
    Serial.print("Sending - Temperature: ");
    Serial.print(t);
    Serial.print("°C");
    Serial.print(", Humidity: ");
    Serial.print(h);
    Serial.print("%");
    Serial.println("");

    // Check if there is not a current TX/RX job running
    if (LMIC.opmode & OP_TXRXPEND) {
        Serial.println(F("OP_TXRXPEND, not sending"));
    } else {
        // Prepare upstream data transmission at the next possible time.
        LMIC_setTxData2(1, (uint8_t*) buffer, res.length(), 0);
        Serial.println(F("Packet queued"));
    }
    // Next TX is scheduled after TX_COMPLETE event.
}

void setup() {
    Serial.begin(115200);
    Serial.println(F("Starting"));
//    dht.begin();

    #ifdef VCC_ENABLE
    // For Pinoccio Scout boards
    pinMode(VCC_ENABLE, OUTPUT);
    digitalWrite(VCC_ENABLE, HIGH);
    delay(1000);
    #endif

    // LMIC init
    os_init();
    // Reset the MAC state. Session and pending data transfers will be discarded.
    LMIC_reset();
```

```
   // Set static session parameters. Instead of dynamically establishing a session
   // by joining the network, precomputed session parameters are be provided.
   #ifdef PROGMEM
   // On AVR, these values are stored in flash and only copied to RAM
   // once. Copy them to a temporary buffer here, LMIC_setSession will
   // copy them into a buffer of its own again.
   uint8_t appskey[sizeof(APPSKEY)];
   uint8_t nwkskey[sizeof(NWKSKEY)];
   memcpy_P(appskey, APPSKEY, sizeof(APPSKEY));
   memcpy_P(nwkskey, NWKSKEY, sizeof(NWKSKEY));
   LMIC_setSession (0x1, DEVADDR, nwkskey, appskey);
   #else
   // If not running an AVR with PROGMEM, just use the arrays directly
   LMIC_setSession (0x1, DEVADDR, NWKSKEY, APPSKEY);
   #endif

   #if defined(CFG_eu868)
   // Set up the channels used by the Things Network, which corresponds
   // to the defaults of most gateways. Without this, only three base
   // channels from the LoRaWAN specification are used, which certainly
   // works, so it is good for debugging, but can overload those
   // frequencies, so be sure to configure the full frequency range of
   // your network here (unless your network autoconfigures them).
   // Setting up channels should happen after LMIC_setSession, as that
   // configures the minimal channel set.
   // NA-US channels 0-71 are configured automatically
   LMIC_setupChannel(0, 868100000, DR_RANGE_MAP(DR_SF12, DR_SF7),
BAND_CENTI);      // g-band
   //LMIC_setupChannel(1, 868300000, DR_RANGE_MAP(DR_SF12, DR_SF7B),
BAND_CENTI);      // g-band
   //LMIC_setupChannel(2, 868500000, DR_RANGE_MAP(DR_SF12, DR_SF7),
BAND_CENTI);      // g-band
   //LMIC_setupChannel(3, 867100000, DR_RANGE_MAP(DR_SF12, DR_SF7),
BAND_CENTI);      // g-band
   //LMIC_setupChannel(4, 867300000, DR_RANGE_MAP(DR_SF12, DR_SF7),
BAND_CENTI);      // g-band
   //LMIC_setupChannel(5, 867500000, DR_RANGE_MAP(DR_SF12, DR_SF7),
BAND_CENTI);      // g-band
   //LMIC_setupChannel(6, 867700000, DR_RANGE_MAP(DR_SF12, DR_SF7),
BAND_CENTI);      // g-band
```

```
  //LMIC_setupChannel(7, 867900000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI);      // g-band
  //LMIC_setupChannel(8, 868800000, DR_RANGE_MAP(DR_FSK,  DR_FSK), BAND_MILLI);      // g2-band
  // TTN defines an additional channel at 869.525Mhz using SF9 for class B
  // devices' ping slots. LMIC does not have an easy way to define set this
  // frequency and support for class B is spotty and untested, so this
  // frequency is not configured here.
  #elif defined(CFG_us915)
  // NA-US channels 0-71 are configured automatically
  // but only one group of 8 should (a subband) should be active
  // TTN recommends the second sub band, 1 in a zero based count.
  // https://github.com/TheThingsNetwork/gateway-conf/blob/master/US-global_conf.json
  LMIC_selectSubBand(1);
  #endif

  // Disable link check validation
  LMIC_setLinkCheckMode(0);

  // TTN uses SF9 for its RX2 window.
  LMIC.dn2Dr = DR_SF9;

  // Set data rate and transmit power for uplink (note: txpow seems to be ignored by the library)
  LMIC_setDrTxpow(DR_SF7,14);

  // Start job
  do_send(&sendjob);
}

void loop() {
  os_runloop_once();
}
```