# Laboratório de processamento digital de sinais Implementação do LoRaWAN

Lucas Givaldo dos Santos Torres
*DES*
*Universidade Federal de Pernambuco*
Recife, Pernambuco
lucas.givaldo@ufpe.br

***Abstract*—Manual do sistema LoRaWAN implementado no Laboratório de Processamento Digital de Sinais.**

***Professor*— José Rodrigues de Oliveira Neto.**

## I. Introdução

O protocolo LoRaWAN [1] foi implementado para estudos posteriores sobre a segurança na comunicação de sistemas embarcados via rádio. O protocolo possui duas versões ainda sob suporte, a v1.1 e a v1.0.x. A versão 1.1 é mais eficiente contra ataques de repetição e separa a responsabilidade do *Network Server* (NS) criando o *Join Server*. A versão 1.0.x possui maior suporte da comunidade e implementação mais simples, por manter a topografia de rede reduzida. Sendo assim, tendo em vista a configuração do *End Device* (ED) disponível e o suporte do fabricante, foi escolhida a versão 1.0.x.

O *sotfware* selecionado para o *Gateway* (GW) foi o *Chirpstack OS* [2] e, para a versão selecionada, foi seguido o guia do fabricante do ED, permitindo a comunicação por meio de uma biblioteca para Arduino. Os dados enviados ao *Application Server* (AS) são visualizados pela integração com *NodeRED*, que está inscrita no tópico MQTT alimentado pelo *Chirpstack*.

Os dispositivos foram selecionados visando à reprodução fiel de um sistema comercial, permitindo ainda, contudo, modificações em futuros estudos. Tendo isso em mente, acessibilidade e suporte foram os outros parâmetros levados em consideração para a escolha dos conjuntos.

## II. *End Device*

### A. Hardware

O dispositivo selecionado foi um pacote de desenvolvimento baseado em Esp32 ligado a um módulo 1276M0 [3] da *Semtech*, o *IoT DevKit - LoRaWAN* da Robocore, Figura 1. Essa opção veio com biblioteca própria, exemplos e suporte para o uso nas versões 1.0.x. Para a versão 1.1, não há suporte do fabricante e não há compatibilidade nas pouquíssimas bibliotecas que implementam esta versão para Arduino.


Fig. 1. Pacote utilizado.

### B. Firmware

Seguindo o guia na documentação do Arduino [4], é possível preparar o ambiente para a execução do código e instalar a biblioteca fornecida pela Robocore [5]. A fabricante fornece uma biblioteca com códigos de exemplo [6], contudo, esses códigos estão desatualizados e não permitiram comunicação durante os testes. O guia da fabricante [7] fornece um código mais atualizado, visto no Tópico VI-A, onde, após definir o pino de *reset*, é chamado o método *lorawan.reset()*. No código, as variáveis *AppEUI* e *Appkey* são definidas pela AS, na configuração do GW.

## III. *Gateway*

### A. Hardware

O GW selecionado foi um *Hat*, Figura 2, placa para acoplamento no topo de uma *Raspberry PI*, da *Waveshare* [8]. A placa liga diretamente alguns pinos do microcontrolador ao SX1302 [9], chip da *Semtech*.

Outra opção seria utilizar outro módulo igual ao ED para servir como GW, contudo, essa configuração não poderia operar no número de canais necessários para um GW de acordo com a especificação *LoRaWAN*, não cumprindo com os objetivos idealizados.

Fig. 2. *Hat* utilizado.

### B. Firmware

Para preparar o GW na *Raspberry Pi*, foi utilizado o *Chirpstack*, um servidor de código aberto com ampla documentação e ferramentas úteis para a comunicação, posteriormente. A documentação do servidor [10] sugere que sua configuração pode ser feita em dispositivos com sistemas operacionais baseados em *Debian*, o que implicaria que é possível utilizar o Sistema Operacional(SO) *Raspberry Pi OS* junto à instalação do GW e do *Network Server* (NS). Apesar dessa configuração permitir maior flexibilidade na configuração do servidor, sendo ideal para implementações futuras no protocolo, o guia de instalação leva a diversos problemas de versionamento e não esclarece métodos de correção, focando a documentação na versão mais atualizada do sistema para a *Raspberry Pi*, o *Chirpstack OS*. Outro problema enfrentado na implementação do sistema em *Pi OS* foi a alimentação do sistema, visto que o módulo anexado à placa necessita de muita corrente, limitando as demais funcionalidades do sistema operacional.

O sistema operacional da *Chirpstack* é feito com suporte para os HATs mais populares no mercado; dentre eles, está o componente da *Waveshare* selecionado para o projeto. O SO dedicado aos serviços da *Chirpstack* resume todos os problemas de instalação, por ser um ambiente de versões compatíveis e reduz o consumo de energia do microcomputador.

### C. Configuração

*1) Instalação do sistema operacional:* Na documentação do sistema operacional, é possível baixar um arquivo de imagem que deve ser escrito no cartão SD onde o sistema estará localizado. Os *softwares* recomendados para essa operação são o *Rufus* para *Windows* e o *Balena Etcher* para *Linux*. Uma vez instalada, a imagem é o SO e está pronta para uso e configuração. Com o *HAT* posto no microcomputador, basta ligar o sistema e começar os ajustes.

*2) Configuração de rede:* Na primeira execução, o GW vai criar um ponto de acesso Wi-Fi com nome *ChirpStackAP-...* e senha "ChirpStackAP". As configurações do dispositivo serão feitas por meio desse acesso, no endereço http://192.168.0.1, acessível a partir de qualquer navegador por um dispositivo conectado ao ponto de acesso. O usuário e a senha são, respectivamente, "root" e "admin".


Fig. 3. Tópico para configuração de rede.

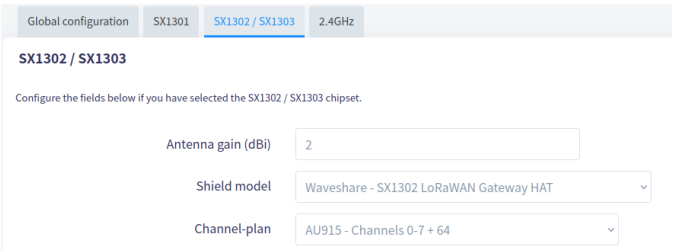


Fig. 4. Configuração do GW

Uma vez acessado, é possível configurar o GW para acessar uma rede e não criar um ponto de acesso, permitindo ao desenvolvedor operar o sistema e acessar uma rede de internet ao mesmo tempo. Para isso, seguindo *Network -> Wireless -> Wireless Overview*, como mostrado na Figura 3, após pressionar o botão de *scan*, o usuário pode inserir as credenciais de uma nova rede, fazendo com que o GW desative o *acess point* e se conecte à rede selecionada.

Por fim, é possível encontrar o novo endereço de rede do GW utilizando o terminal mostrado pelo microcomputador com o comando *ifconfig*, sendo o valor da variável *inet addr*, em *phy0-sta0*.

*3) Configuração do Gateway:* Para o tópico *Concentratord* é marcada a caixa *Enable ChirpStack Concentratord* e selecionado SX1302/SX1303 em *Enabled Chipset*, em *Global Configuration*. No mesmo tópico, em SX1302/SX1303, é selecionado o tipo de *Gateway* utilizado, nesse caso, o fornecido pela *Waveshare*, sua antena e a banda de atuação, conforme a configuração da Figura 4.

*4) Configuração do AppServer:* No tópico *Application*, é possível selecionar o ícone da *Chirpstack*, Figura 5, sendo redirecionado para a página de controle do servidor. O usuário e a senha são, respectivamente, "admin" e "admin". Uma vez no menu principal, é possível acessar o menu de criação de AS selecionando o tópico *Applications* e preenchendo o nome da aplicação.

### D. Configuração do EndDevice

Antes de cadastrar um ED, é necessário criar um perfil de dispositivos. No menu do NS, no tópico *Device Profiles*, em *Add Device Profile* é possível criar um perfil novo. Para o uso no laboratório, o formulário *General* foi preenchido conforme a Figura 6 e a opção *Device supports OTAA* foi marcada no tópico *Join OTAA / ABP*. Após todas as modificações, é necessário apertar o botão *submit*.

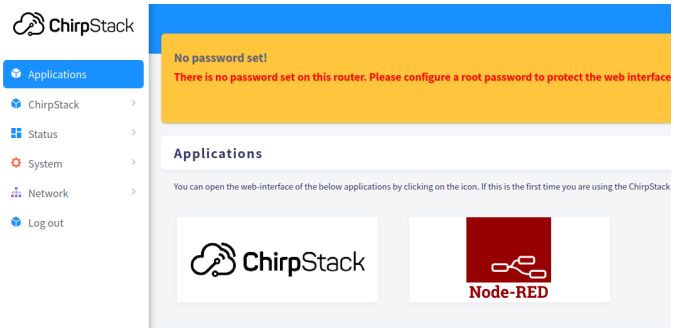


Fig. 5. Aplicações integradas ao GW.

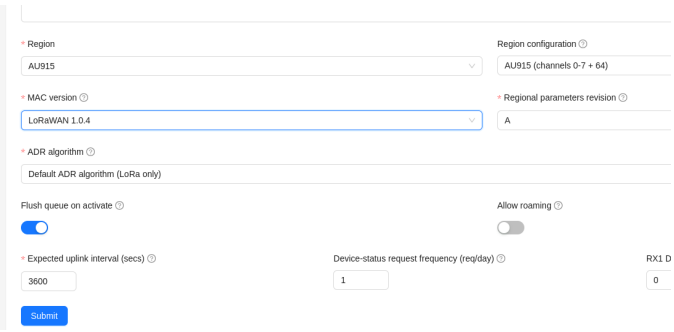


Fig. 6. Configuração do *Device Profile*

Para cadastrar um novo ED na aplicação, no menu da aplicação criada, após selecionar *Add Device*, um formulário deve ser preenchido, Figura 7, selecionando o perfil de dispositivo criado, seu identificador e o *Device EUI*, encontrado no encapsulamento do SX1276.

Ao lado da caixa de texto do *Join EUI*, em *Configuration*, e da caixa *Application Key* em *OTAA keys*, há um ícone para a criação de chave aleatória. Essas chaves serão utilizadas no código do ED como a *App EUI* e a *App key*, respectivamente, durante a criação do *JoinRequest* do dispositivo.

### E. Integração com NodeRed

O *Node-RED* é uma ferramenta de código visual que permite a visualização rápida das mensagens recebidas *upstream*, enviadas do ED ao AS, graças à integração MQTT

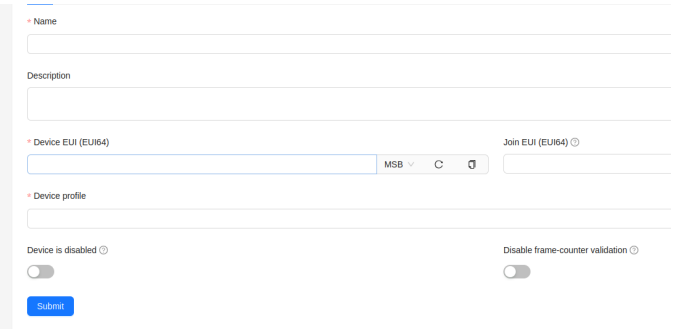


Fig. 7. Cadastro de novo ED

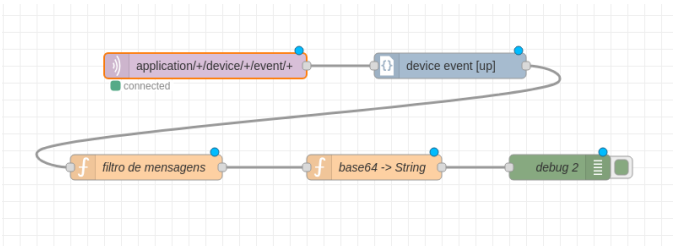


Fig. 8. Esquema no Node-RED

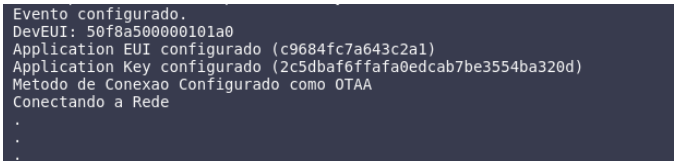


Fig. 9. ED tentando conexão.

do GW. Selecionar a imagem ao lado do ícone do NS mostra o sistema em operação no microcomputador. O esquema do código pode ser visto na Figura 8. 8.

Uma célula *mqtt in* inscrita no tópico *application/+/device/+/event/+* alimenta uma célula fornecida pela *ChirpStack* para filtro de mensagens *upstream*, *device event [up]* que alimenta um formatador proposto na documentação do NS, que alimenta uma célula tradutora de *base64* para *string* e, por fim, é escutada pela célula de *debug*, que exibe seu valor no terminal. Para aplicar qualquer alteração ao diagrama, deve-se pressionar *deploy* no canto superior direito da tela.

## IV. Preparando Comunicação

### A. Envio de mensagem

Afim de comunicar uma mensagem com sucesso pelo sistema implementado, com o ED cadastrado no AS e o código fornecido pela fabricante do dispositivo, foi enviado o texto "teste123". O objetivo é receber essa mensagem de forma legível na interface programada do *Node-RED*.

Utilizando o código em VI-A, o monitor serial descreve o andamento do processo de conexão, como visto na Figura 9. Caso o código tenha sido compilado e "Conectando a Rede" não tenha fim, as credenciais podem estar incorretas, ou a variável *loraSerial* pode não estar sendo reiniciada adequadamente.

Conforme programado, uma vez que o ED tenha resposta positiva do NS, o sucesso será notificado no monitor serial, Figura 10, e a mensagem de teste será periodicamente enviada ao AS.

### B. Tratamento da mensagem

No NS, é possível verificar as mensagens recebidas pelo GW em *LoRa frames*. Durante a tentativa de conexão de um ED, as requisições de entrada podem ser vistas, como na Figura 11, sendo recebidas e respondidas pelo servidor, caso as credenciais estejam corretas.

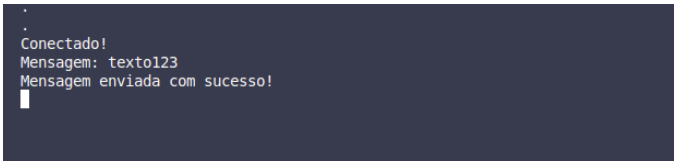


Fig. 10. Comunicação estabelecida


Fig. 11. *Join Frames*.

De maneira similar, uma vez que o ED seja reconhecido e haja histórico de envio de mensagens ao NS, como visto no envio de mensagens *upstream*, Figura 12, dentro do servidor de aplicação o usuário pode acessar apenas as mensagens trocadas com o dispositivo, avaliando características como o *Received Signal Strength Indicator* (RSSI), indicador de força do sinal recebido pelo GW, Figura 13.

Por fim, tendo o ambiente configurado no *Node-RED*, as mensagens chegam conforme a Figura 14, sendo filtradas e traduzidas antes de serem expostas no terminal de *debug*.

## V. Testes

Seguindo o procedimento descrito nos tópicos anteriores e preparando o ED para enviar a mesma mensagem periodicamente com 30 segundos de espera, foi avaliada a distância e a força do sinal entre os dispositivos na universidade. Dois testes foram conduzidos: No primeiro, até o ponto 1 da Figura 15, o dispositivo percorreu o departamento de eletrônica e sistemas e, no segundo, parte da universidade, os demais pontos da figura. Em ambos os testes, o ED tinha sua antena apontada na direção do GW e estava a 1,5 metros do chão. O RSSI é um valor negativo e quanto menor, pior é a qualidade do sinal.


Fig. 12. Mensagens *upstream*.


Fig. 13. RSSI no pacote.

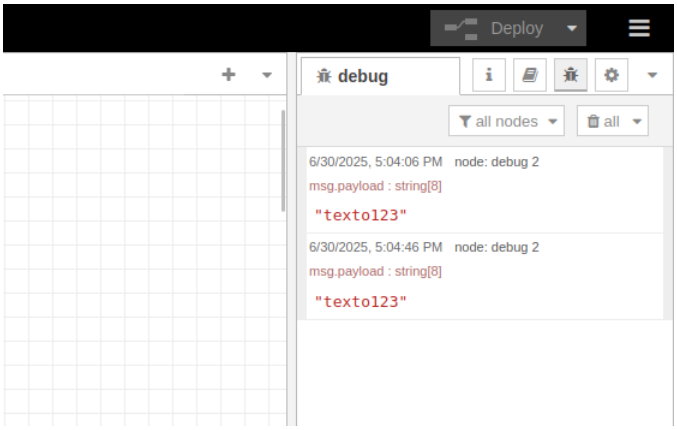


Fig. 14. Mensagens no terminal do *Node-RED*

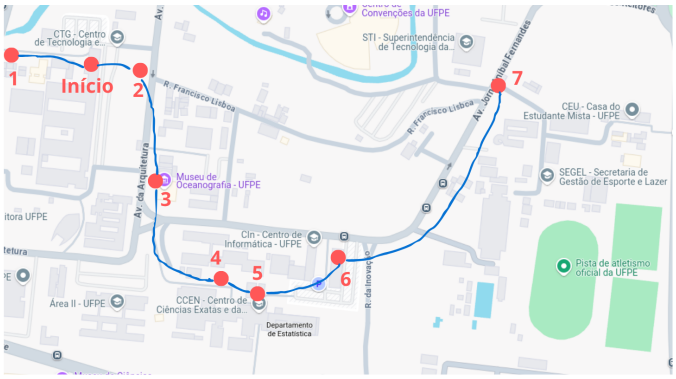


Fig. 15. Percurso dos testes

### A. Primeiro teste

No primeiro teste, o ED estava a 60 metros de distância do GW, na mesma altura, mas entre diversas paredes. O RSSI medido foi de -122 e apenas um pacote estava sendo recebido a cada 3 minutos.

### B. Segundo teste

No segundo teste, o dispositivo seguiu o percurso 2->3->4->5->6->7. O GW foi posto no quarto andar do prédio do Centro de Tecnologia e Geociências, há cerca de 10 metros do chão. A vista entre os pontos selecionados é obstruída por árvores e alguns edifícios.

| Ponto | RSSI (dBm) | Distância (m) |
|---|---|---|
| 2 | -100 | 85 |
| 3 | -105 | 165 |
| 4 | -106 | 325 |
| 5 | -110 | 378 |
| 6 | -118 | 432 |
| 7 | -114 | 600 |

TABLE I
Tabela relacionando os pontos ao RSSI e à distância.

Apesar das medições 4 e 5 terem sido feitas dentro de prédios, o RSSI se manteve bom, mas com aparente perda de pacotes mostrada pelo aumento do tempo entre mensagens.

O teste foi parado por apresentar resultados satisfatórios, mas foi notado que na avenida principal, a 700 metros de distância, o GW ainda recebia mensagens com RSSI próximo ao visto no primeiro teste.

## VI. CÓDIGOS

### *A. End Device*

```
// Verifica se o modelo de
        placa selecionado esta correto
#if !defined(ARDUINO_ESP32_DEV) // ESP32
#error Use this example with the ESP32
#endif

// Inclusao das bibliotecas
#include <RoboCore_SMW_SX1276M0.h>
#include <HardwareSerial.h>
#include <ArduinoJson.h>

// Declaracao dos pinos de comunicacao
        serial do Kit
HardwareSerial LoRaSerial(2);
#define RXD2 16
#define TXD2 17

// Criacao do objeto
        lorawan para a biblioteca SMW_SX1276M0
SMW_SX1276M0 lorawan(LoRaSerial);

// Declaracao da variavel que
        armazena as respostas do modulo
CommandResponse resposta;

// Declaracao das variaveis que
        armazenam as informacoes da rede
const char APPEUI[] = "XXXXXXXXXXXXXXXX";
const char APPKEY[] =
        "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
// As variaveis acima devem ser alteradas
        de acordo com os codigos gerados
        na plataforma Chirpstack

// Declaracao das variaveis de
        intervalo de tempo
const unsigned long TEMPO_ESPERA =
        30000;//30 segundos
unsigned long intervalo;

// Declaracao das variaveis para o LDR
const int PINO_LDR = 15;
int leitura_LDR = 0;

// Declaracao da funcao que verifica a
        conexao do modulo
void event_handler(Event);

void setup()
{

  // Inicia o monitor serial e
        imprime o cabecalho
  Serial.begin(115200);
  Serial.println(F("Envio de Teste
        para o Chirpstack"));

  // Definicao do pino de reset do modulo
  lorawan.setPinReset(5);
  lorawan.reset(); // Realiza um
        reset no modulo

  // Inicia a comunicacao serial
        com o modulo
  LoRaSerial.begin(115200,
        SERIAL_8N1, RXD2, TXD2);

  // Define o pino conectado ao
        sensor como uma entrada
  pinMode(PINO_LDR, INPUT);

  // Associa a funcao que verifica a conexao
        do modulo ao objeto "lorawan"
  lorawan.event_listener = &event_handler;
  Serial.println(F("Evento configurado."));

  // Requisita e imprime o Device EUI do modulo
  char deveui[16];
  resposta = lorawan.get_DevEUI(deveui);
  if (resposta == CommandResponse::OK)
  {
    Serial.print(F("DevEUI: "));
    Serial.write((uint8_t *)deveui, 16);
    Serial.println();
  }
  else
  {
    Serial.println(F("Erro ao
        obter o Device EUI"));
  }

  // Configura o Application EUI no modulo
  resposta = lorawan.set_AppEUI(APPEUI);
  if (resposta == CommandResponse::OK)
  {
    Serial.print(F("Application
        EUI configurado ("));
    Serial.write((uint8_t *)APPEUI, 16);
    Serial.println(')');
  }
  else
  {
    Serial.println(F("Erro ao
        configurar o Application EUI"));
```

```
  }

  // Configura o Application Key no modulo
  resposta = lorawan.set_AppKey(APPKEY);
  if (resposta == CommandResponse::OK)
  {
    Serial.print(F("Application Key
        configurado ("));
    Serial.write((uint8_t *)APPKEY, 32);
    Serial.println(')');
  }
  else
  {
    Serial.println(F("Erro ao configurar
        o Application Key"));
  }

  // Condfigura o metodo de conexao como OTAA
  resposta = lorawan.set_JoinMode(
        SMW_SX1276M0_JOIN_MODE_OTAA);
  if (resposta == CommandResponse::OK)
  {
    Serial.println(F("Metodo de
        Conexao Configurado como OTAA"));
  }
  else
  {
    Serial.println(F("Erro ao
        configurar o metodo OTAA"));
  }

  // Requisita conexao com a rede
  Serial.println(F("Conectando a Rede"));
  lorawan.join();
}


void loop()
{

  // Recebe informacoes do modulo
  lorawan.listen();

  if (lorawan.isConnected())
  { // Verifica se o modulo esta conectado

    // Executa o envio da mensagem
        uma vez a cada 30 segundos
    if (intervalo < millis())
    {
      char textoMes[] = "texto123";
      resposta = lorawan.sendT(1,
        textoMes); // Envio como texto

      Serial.print(F("Mensagem: "));
      Serial.println(textoMes);

      if (resposta == CommandResponse::OK)
      {
        Serial.println(F(
            "Mensagem enviada com sucesso!"));
      }
      else
      {
        Serial.print(F(
            "Erro ao enviar mensagem: "));
        Serial.println((int)resposta);
      }

      intervalo = millis() + TEMPO_ESPERA;
    }
  }
  else
  { // Se o modulo nao estiver conectado

    // Imprime um "." a cada 5 segundos
    if (intervalo < millis())
    {

      Serial.println('.');
      intervalo = millis() + 3000;
    }
  }
}

void event_handler(Event type)
{

  // Verifica se o modulo esta conectado e
    atualiza essa informacao
  if (type == Event::JOINED)
  {
    Serial.println(F("Conectado!"));
  }
}
```

*B. Node-RED*

*1) Formatador proposto pela documentação:*

```
return {
devEui: msg.payload.deviceInfo.devEui,
fPort: msg.payload.fPort,
confirmed: false,
payload: msg.payload.data
};
```

*2) Célula tradutora:*

```
const base64 = msg.payload;
const decoded = Buffer.from(base64,
'base64').toString('utf-8');

msg.payload = decoded;
```

```
return msg;
```

## REFERÊNCIAS


[1] LoRa Alliance, *LoRaWAN - Technical Specifications*.
[2] ChirpStack, *ChirpStack GateWay OS - Introduction*.
[3] Semtech, *SX1276-7-8 - Datasheet*.
[4] Arduino, *Arduino - Getting Started*.
[5] Arduino, *Arduino - Installing Libraries*.
[6] RoboCore, *RoboCore*$_S$MW – SX1276M0.
[7] RoboCore, *IoT DevKit - Guia*.
[8] Waveshare, *SX1302 LoRaWAN Gateway HAT*.
[9] Semtech, *SX1302 - Datasheet*.
[10] ChirpStack, *The ChirpStack project*.