

Jean Michel de Souza Sant'Ana

Redes LoRaWAN: implantação e desenvolvimento de aplicações

São José

2017

Jean Michel de Souza Sant'Ana

Redes LoRaWAN: implantação e desenvolvimento de aplicações

Trabalho de Conclusão de Curso do Bacharelado em Engenharia de Telecomunicações do Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina apresentado à banca avaliadora para obtenção do grau de Engenheiro de Telecomunicações.

Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina

Bacharelado em Engenharia de Telecomunicações

Orientador: Prof. MSc. Arliones Stevert Hoeller Jr

Coorientador: Prof. Dr. Mario de Noronha Neto

São José

2017

Jean Michel de Souza Sant'Ana

Redes LoRaWAN: implantação e desenvolvimento de aplicações

Trabalho de Conclusão de Curso do Bacharelado em Engenharia de Telecomunicações do Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina apresentado à banca avaliadora para obtenção do grau de Engenheiro de Telecomunicações.

Trabalho aprovado. São José, 3 de março de 2017:

Prof. MSc. Arliones Stevert Hoeller Jr
IFSC - Orientador

Prof. Dr. Mario de Noronha Neto
IFSC - Co-orientador

Prof. Dr. Richard Demo Souza
UTFPR - Avaliador

Prof. Dr. Odilson Tadeu Valle
IFSC - Avaliador

Prof. Dr. Marcelo Maia Sobral
IFSC - Avaliador

São José
2017

Agradecimentos

Agradeço primeiramente a Deus, por todas as oportunidades colocadas em minha vida.

A minha família, principalmente aos meus pais, por todo apoio, compreensão e investimento em minha vida acadêmica.

Aos meus professores, em especial meu orientador, pelo incentivo ao aprendizado, ensino de qualidade e total disponibilidade.

Aos amigos e colegas de curso, que sempre proporcionaram boa companhia e discussões dentro e fora do campus.

*“Sempre que te perguntarem se podes fazer um trabalho,
respondas que sim e te ponhas em seguida a aprender como se faz”
(F. Roosevelt)*

Resumo

Com o crescimento do número de dispositivos conectados à rede, cada vez mais a Internet das Coisas entra em foco. Muitas tecnologias são apontadas para a conexão desses dispositivos e as LPWAN (Low Power Wide Area Networks) mostram-se uma alternativa viável para aplicações que necessitam se comunicar a longas distâncias e com baixo consumo de energia. Neste contexto, a tecnologia de comunicação LoRa e sua pilha de protocolos, a LoRaWAN, apresentam-se como uma alternativa promissora. Desta forma, neste trabalho são descritos equipamentos, *software* e protocolos disponíveis para a implantação de uma rede LoRaWAN, que foi implantada com o objetivo de estudar seu funcionamento. Junto à rede, foi desenvolvida um sistema gerenciador de aplicações LoRaWAN, configurável através de mensagens enviadas pelo servidor. Este sistema define um protocolo próprio para configuração remota dos dispositivos LoRaWAN. O sistema mostrou-se promissor, sendo capaz de gerenciar até 52 aplicações simultâneas.

Palavras-chaves: internet das coisas, LoRaWAN, sistemas embarcados.

Abstract

The growing number of devices connected to the Internet-of-Things (IoT) brings several technologies into focus. Many technologies are deployed to connect these devices to the IoT, and the LPWAN (Low Power Wide Area Networks) seems to be a viable option to applications which need to communicate over long distances at low energy consumption. Within this context, the LoRa communication technology and its protocol stack, the LoRaWAN, feature as a promising option. In order to study the LoRaWAN, this work describes equipment, software and protocols available for the network, and deploys a LoRaWAN network. Also, a configurable LoRaWAN application system was developed, allowing users to remotely configure the applications running in the LoRaWAN devices by sending configuration messages from a server using a protocol develop for this end. The system showed to be useful during the IoT application development process and was able to support up to 52 simultaneous applications.

Key-words: internet-of-things, LoRaWAN, embedded systems.

Lista de ilustrações

| | |
|---|----|
| Figura 1 – Uma taxonomia para aplicações de redes de sensores sem-fio (MOT-TOLA; PICCO, 2011). | 20 |
| Figura 2 – Arquitetura LoRaWAN. | 24 |
| Figura 3 – Arquitetura da rede LoRaWAN. | 24 |
| Figura 4 – Janelas de recepção do dispositivo classe A. | 25 |
| Figura 5 – Janelas de transmissão de dispositivos classe B. | 26 |
| Figura 6 – Estrutura do <i>uplink</i> da camada física. | 27 |
| Figura 7 – Estrutura do <i>downlink</i> da camada física. | 27 |
| Figura 8 – Formato dos elementos da mensagem LoRa. | 27 |
| Figura 9 – Estrutura do MACPayload. | 28 |
| Figura 10 – Estrutura do FHDR. | 29 |
| Figura 11 – Estrutura do DevAddr. | 29 |
| Figura 12 – Estrutura do FCtrl. | 29 |
| Figura 13 – Operação de perda de <i>beacon</i> | 32 |
| Figura 14 – Comportamento das janelas de recepção de dispositivos de classe C. | 32 |
| Figura 15 – Arquitetura de rede comum para projetos LoRaWAN. | 33 |
| Figura 16 – Layout do módulo RHF0M301 e conexões com a Raspberry Pi B+. | 38 |
| Figura 17 – Estrutura de uma <i>string</i> JSON. | 40 |
| Figura 18 – Estrutura de um numeral JSON. | 41 |
| Figura 19 – Estrutura de um <i>array</i> JSON. | 41 |
| Figura 20 – Estrutura de um objeto JSON. | 41 |
| Figura 21 – Diagrama de sequência de <i>upstream</i> | 42 |
| Figura 22 – Primeiro diagrama de sequência de <i>downstream</i> | 43 |
| Figura 23 – Segundo diagrama de sequência de <i>downstream</i> | 43 |
| Figura 24 – Interface web do LoRa App Server com as aplicações cadastradas. | 48 |
| Figura 25 – Interface web do LoRa App Server com nós cadastrados em uma aplicação. | 49 |
| Figura 26 – Interface web do LoRa App Server com os detalhes de um nó cadastrado. | 50 |
| Figura 27 – Estrutura da mensagem de controle do protocolo de configuração. | 51 |
| Figura 28 – Diagrama de classe do sistema de gerenciamento de aplicações LoRaWAN configuráveis. | 55 |
| Figura 29 – Diagrama de sequência do funcionamento do ciclo de transmissão do sistema. | 56 |
| Figura 30 – Código-fonte em C++ da declaração dos métodos parametrizados com <i>templates</i> | 57 |
| Figura 31 – Código-fonte em C++ com a implementações de métodos parametrizados com <i>templates</i> | 58 |

Figura 32 – Código-fonte em C++ para configuração da LMIC no Arduino. 59

Figura 33 – Código-fonte em C++ do tratamento de envio-recepção da máquina de estados da LMIC. 59

Lista de tabelas

| | |
|--|----|
| Tabela 1 – Tipos de mensagens MAC | 28 |
| Tabela 2 – Conexão do módulo Modtronix inAir9B ao Arduino Mega. | 36 |
| Tabela 3 – Conexão do módulo HopeRF RFM95W ao Arduino Mega. | 36 |
| Tabela 4 – Conexões do módulo Modtronix inAir9B com a RaspberryPi B+. | 37 |
| Tabela 5 – Conexões do módulo HopeRF RFM95W com a RaspberryPi B+. | 38 |
| Tabela 6 – Estrutura da mensagem PUSH_DATA. | 44 |
| Tabela 7 – Estrutura da mensagem PUSH_ACK. | 44 |
| Tabela 8 – Atributos do objeto com pacote LoRa contidos no vetor rxpk. | 44 |
| Tabela 9 – Atributos do objeto stat. | 45 |
| Tabela 10 – Estrutura da mensagem PULL_DATA. | 45 |
| Tabela 11 – Estrutura da mensagem PULL_ACK. | 45 |
| Tabela 12 – Estrutura da mensagem PULL_RESP. | 46 |
| Tabela 13 – Atributos do objeto txpk. | 46 |
| Tabela 14 – Estrutura da mensagem TX_ACK. | 46 |
| Tabela 15 – Atributos do objeto txpk_ack. | 47 |
| Tabela 16 – Possíveis valores para o campo error. | 47 |
| Tabela 17 – Possíveis valores do campo Type. | 52 |
| Tabela 18 – Possíveis valores para o campo Channel e Ch_Param. | 53 |
| Tabela 19 – Possíveis configurações para a UART. | 53 |
| Tabela 20 – Possíveis dispositivos SPI. | 53 |
| Tabela 21 – Possíveis dispositivos I2C | 54 |
| Tabela 22 – Possíveis dispositivos de interface genérica (GDI). | 54 |
| Tabela 23 – Possíveis métodos de aquisição de dados. | 54 |
| Tabela 24 – Eventos gerados pela biblioteca LMIC. | 55 |

Sumário

| | | |
|------------|--|-----------|
| 1 | INTRODUÇÃO | 17 |
| 1.1 | Objetivos | 18 |
| 2 | FUNDAMENTAÇÃO TEÓRICA | 19 |
| 2.1 | Internet das Coisas | 19 |
| 2.2 | LPWAN | 20 |
| 2.2.1 | LoRa™ | 22 |
| 2.3 | LoRaWAN | 23 |
| 2.3.1 | Mensagens da camada física | 26 |
| 2.3.2 | Mensagens da camada de enlace | 27 |
| 2.3.3 | Ativação na rede | 30 |
| 2.3.4 | Aquisição e monitoramento do <i>beacon</i> | 31 |
| 2.3.5 | Janelas de transmissão na classe C | 32 |
| 2.4 | Topologia da rede | 32 |
| 3 | IMPLANTAÇÃO DE REDE LORAWAN | 35 |
| 3.1 | Equipamentos | 35 |
| 3.1.1 | <i>Endpoints</i> | 35 |
| 3.1.2 | <i>Gateway</i> | 37 |
| 3.1.3 | Servidor | 39 |
| 3.2 | Serviços na rede | 39 |
| 3.2.1 | O formato JSON | 40 |
| 3.2.2 | UDP Packet Forwarder | 41 |
| 3.2.3 | LoRaServer | 45 |
| 3.2.4 | Servidor de Aplicação LoRa | 48 |
| 4 | SISTEMA GERENCIADOR DE APLICAÇÕES LORAWAN CONFIGURÁVEIS | 51 |
| 4.1 | Protocolo de configuração | 51 |
| 4.2 | Aplicação no <i>endpoint</i> | 52 |
| 4.2.1 | LMIC: LoRaMAC-In-C | 53 |
| 4.2.2 | Estrutura da aplicação | 54 |
| 4.3 | Testes | 57 |
| | Conclusão | 61 |

REFERÊNCIAS **63**

1 Introdução

A Internet das Coisas (IoT) é um tema recorrente de pesquisa. Existe atualmente um crescente número de dispositivos conectados à rede, algo que abre possibilidade para novas aplicações. Pode-se encontrar aplicações para esses dispositivos que demandam alta ou baixa taxa de transmissão, grande ou pequena cobertura, ou a necessidade de haver pouca manutenção, para ambientes com difícil acesso, por exemplo. Existe uma série de tecnologias bem difundidas que podem cobrir essas demandas.

As possíveis soluções para tais demandas mais difundidas são o GSM e o IEEE 802.15.4. Uma das vantagens do GSM é sua ampla rede já instalada cobrindo a maior parte do território nacional e mundial. Porém, essas redes trabalham em faixas de frequência licenciadas alocadas a empresas privadas, o que dificulta e encarece as aplicações. Outra limitação é uma baixa eficiência energética comparada a tecnologias concorrentes. Já o IEEE 802.15.4 possui uma eficiência energética melhor, comparado ao GSM, e trabalha em uma faixa não licenciada. Sua maior limitação é o alcance, fazendo necessário o uso nós retransmissores para cobrir áreas com distâncias maiores que quilômetros.

Uma nova categoria de tecnologias sendo explorada se chama LPWAN (Low Power Wide Area Network). Ela inclui tecnologias sem fio de baixo consumo de energia e longo alcance. Suas maiores limitações são as taxas de transmissão, que usualmente não ultrapassam a casa dos 100 kbps. São especialmente projetadas para conectar dispositivos que enviam pequenas quantidades de dados com um longo alcance e com necessidade de ter uma bateria com longa duração, como redes de sensores, por exemplo ([LINK LAB'S, 2016](#)).

Na categoria de LPWAN duas tecnologias destacam-se: Sigfox e LoRa. A Sigfox é uma tecnologia proprietária, de camada física até rede, que atualmente provê cobertura em boa parte da Europa e está em estado de implantação em países da América do Sul, como Brasil e Argentina ([COLLET, 2016](#)). Utiliza um modelo de negócio onde o usuário compra um *end-point* e paga uma mensalidade por uma quantidade de tráfego por *end-point*. Suas mensagens não devem exceder 12 bytes de payload, disponibilizando, no melhor de seus planos, 140 mensagens por dia ([NOLAN; GUIBENE; KELLY, 2016](#)). Devido a estas limitações, a tecnologia promete alcance de até 50 km, o maior dentre as principais tecnologias LPWAN. Outra problemática encontrada no Sigfox é a falta de informação sobre a sua camada física e códigos-fonte livres que implementam o funcionamento da rede para dispositivos e servidores.

Já a LoRa, que é uma derivação da tecnologia de espalhamento espectral (CSS – *Chirp Spread Spectrum*), é propriedade da Semtech e compreende somente a camada física.

Por isso, existe a possibilidade do desenvolvimento de diversas pilhas de protocolos que utilizam a tecnologia LoRa. Dentre essas redes já desenvolvidas, a que mais se destaca é a LoRaWAN, regida pela LoRa Alliance, associação de empresas responsáveis pela sua padronização.

A LoRaWAN caracteriza-se por ser uma pilha de protocolos aberta, logo, é possível encontrar diversas implementações livres do seu funcionamento, bem como a implementação deles com dispositivos de baixo custo disponíveis no mercado. Apresenta uma topologia de rede em estrela (típico das LPWAN) onde nós (*end-points*) conectam-se com um ou mais *gateways*, que por sua vez estão conectados a um *Netserver* responsável pelo tratamento das mensagens recebidas. A rede promete taxa adaptativa de 0,3 a 50 kbps e distâncias de até 40 km com linha de visada e um alcance de 3 a 5 quilômetros em ambientes urbanos (PETAJARVI et al., 2015).

Devido ao fato de somente a camada física LoRa ser patenteada e fechada, possuindo uma rede aberta e com boa documentação, os estudos de redes LoRaWAN tornam-se economicamente mais viáveis que em redes Sigfox, onde é necessário o pagamento de licenças por dispositivos e a cobertura de *gateways* Sigfox na região de estudo. Dessa forma, esse trabalho tem como objetivo um estudo sobre o funcionamento do LoRaWAN, a implantação de uma rede, o desenvolvimento de um sistema de gerenciamento de aplicações LoRaWAN configuráveis para a realização de testes e verificar seu funcionamento.

1.1 Objetivos

O objetivo geral deste trabalho é implantar uma rede LoRaWAN e desenvolver aplicações para a mesma, permitindo a verificação de seu funcionamento. Para atingir este objetivo, os seguintes objetivos foram definidos:

- Estudar e compreender o funcionamento de uma rede LoRaWAN;
- Adquirir, montar e configurar *end-points* LoRaWAN;
- Adquirir, montar e configurar um *gateway* LoRaWAN;
- Instalar e configurar um *Netserver* LoRaWAN;
- Conceber e prototipar um sistema para gerenciamento de aplicações LoRaWAN;
- Implantar aplicações na rede LoRaWAN e verificar seu funcionamento.

2 Fundamentação Teórica

Este capítulo servirá como base teórica para a avaliação de redes LoRaWAN. Nele, são levantadas informações sobre Internet das coisas (IoT), uma das principais aplicações das redes LoRaWAN. Será abordado brevemente o funcionamento básico que as LPWANs apresentam e as características básicas da tecnologia LoRa. Por fim, um estudo mais aprofundado sobre a pilha de protocolo LoRaWAN, os elementos das suas mensagens e suas funcionalidades.

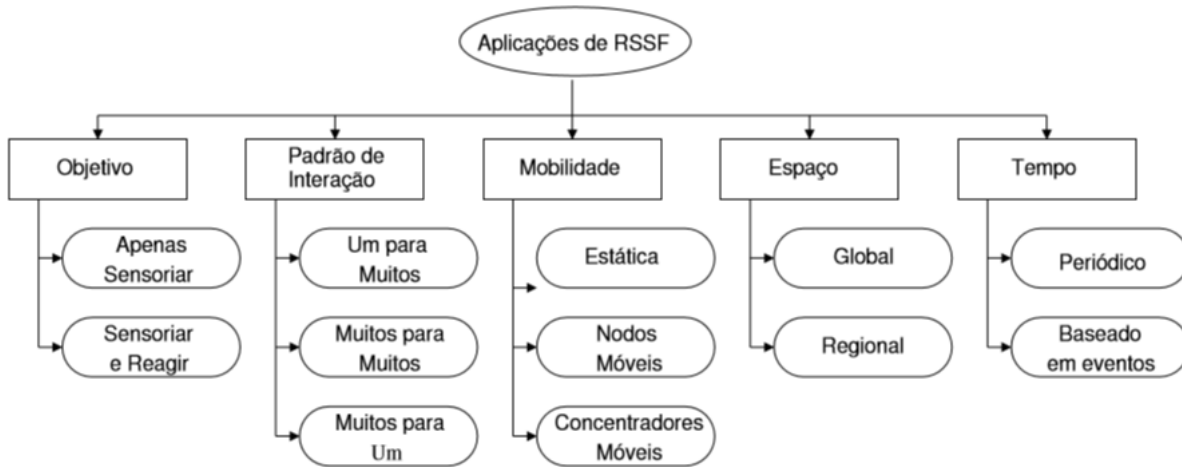
2.1 Internet das Coisas

No início da década de 90 começaram as primeiras ideias de uma Internet ubíqua. Segundo [Weiser \(1991\)](#), “elementos de hardware e software, conectados por cabos, ondas de rádio e infravermelho, serão tão ubíquos que ninguém perceberá sua presença”. Nos dias atuais, fica nítido a presença de tais tecnologias no cotidiano: edifícios com sensores para economizar energia, aplicações em domótica, smartphones com sensores de passos, medicina remota ([STANKOVIC, 2014](#)), monitoramento de qualidade de ar, controle de tráfego, controle de iluminação ([ZANELLA et al., 2014](#)). Segundo [Stankovic \(2014\)](#), teremos um momento de transição com a sofisticação das tecnologias, onde ocorrerá uma mudança tanto em densidade de dispositivos quanto em área de cobertura. “Neste momento, o grau de cobertura deve triplicar ou quadruplicar em relação ao que temos hoje. [...] Assim teremos uma mudança qualitativa no modo de trabalhar e viver”.

Atualmente, uma das grandes aplicações para Internet das Coisas (IoT), se não a que mais se sobressai, são as Redes de Sensores Sem Fio (RSSF). São sistemas distribuídos geralmente compostos de dispositivos embarcados, equipados com uma unidade de processamento, uma interface de comunicação sem fio e sensores. Os cenários apresentados para tais redes comumente requerem que esses dispositivos consumam pouca bateria e gerem pouco tráfego de dados, tornando os dispositivos baratos e de fácil interação com o mundo de forma ubíqua ([MOTTOLA; PICCO, 2011](#)).

O conceito de IoT não define um padrão ou tecnologia específico para conectar os dispositivos à rede. Existem opções bem estabelecidas para realizar este papel. Entretanto, são inúmeras as aplicações para RSSF e muitas continuam a surgir, o que torna impossível generalizá-las em um único grupo. Desta forma, [Mottola e Picco \(2011\)](#) propuseram uma taxonomia para aplicações de RSSF, dividindo-as em categorias baseadas em seus requisitos e propósito. Essa taxonomia pode ser visualizada na [Figura 1](#). Assim, pode-se atribuir a melhor solução tecnológica para cada uma dessas combinações apresentadas na taxonomia de forma que facilite o desenvolvimento de uma nova aplicação.

Figura 1 – Uma taxonomia para aplicações de redes de sensores sem-fio (MOTTOLA; PICCO, 2011).



Fonte: Hoeller e Fröhlich (2010).

Centenaro et al. (2015) encaixa alguns padrões de comunicação para IoT nas categorias enumeradas abaixo:

1. Sistemas alcance extremamente curto, e.g., NFC;
2. Sistemas passivos e ativos de curto alcance, e.g., RFID;
3. Sistemas baseados no padrão IEEE 802.15.4 como ZigBee, 6LoWPAN;
4. Sistemas baseados em Bluetooth, incluindo o *Bluetooth Low Energy* (BLE);
5. Sistemas proprietários como Z-Wave, CSRMesh;
6. Sistemas baseados no IEEE 802.11, principalmente o Wi-Fi;

Neste contexto de tecnologias, surge como nova categoria para comunicações de longo alcance, com principal uso aplicações para IoT, as LPWANs.

2.2 LPWAN

Uma categoria de tecnologias emergentes para redes de baixa potência é a LPWAN (*Low Power Wide Area Network*). Projetadas para trabalharem em redes M2M (*Machine to Machine*), “conectando dispositivos que necessitam enviar uma pequena quantidade de dados por uma longa distância, sendo capazes de manter uma bateria por um longo período” (LINK LAB’S, 2016).

Atualmente, uma das principais características das LPWANs é a operação em bandas não licenciadas, principalmente nas bandas ISM em 2,4 GHz, 868/915 MHz, 433 MHz e 169 MHz. Suas camadas físicas são projetadas para terem alta sensibilidade no receptor, em torno de -130 dBm, quando comparada a outras tecnologias sem fio, que ficam em torno de -90 dBm até -110 dBm. Esta sensibilidade só é possível ser alcançada utilizando taxas de modulações mais lentas, aumentando a quantidade de energia por símbolo. Em contrapartida, baixas taxas de modulação implicam em baixas taxas de transmissão, fazendo com que não ultrapassem 100 kbps de modo geral. Combinando uma alta sensibilidade com a utilização de frequências abaixo da casa dos GHz, as LPWANs são capazes de atingir distâncias de até 10 km em áreas urbanas e 30 km ou mais em áreas rurais, caso haja linha de visada (CENTENARO et al., 2015; LINK LAB'S, 2016). Estas taxas pequenas e longas distâncias assemelham-se em muito à aplicações e serviços para IoT, com “[...] transmissões intermitentes e esporádicas de pequenos pacotes de dados, tolerantes a atrasos, perda de pacotes[...]” (CENTENARO et al., 2015).

Devido às grandes distâncias alcançadas, a topologia de rede das LPWAN consiste em redes estrela, onde cada nó se comunica com a estação base (*gateway*) diretamente, sem a necessidade de comunicação entre nós (*multi-hop*). A topologia estrela é simples de implementar e reduz a quantidade de tráfego na rede. Por exemplo, para qualquer comunicação entre 2 nós, somente 3 dispositivos (2 nós e 1 *gateway*) e 2 enlaces (nó1-*gateway* e *gateway*-nó2) são envolvidos. No caso das LPWAN, a maioria das trocas de mensagens seguem o fluxo nó-*gateway*-servidor, contendo apenas um enlace sem fio (geralmente a comunicação entre *gateways* e servidores utiliza uma outra tecnologia de rede, principalmente redes IP). Desta forma, os nós são isolados uns dos outros, facilitando a reposição desses nós. A centralização da rede também facilita a inspeção do tráfego da rede em um único ponto. A grande desvantagem desse tipo de rede é o ponto único de falha localizado no *gateway*, que pode desabilitar uma rede inteira caso haja algum problema nele (SEMTECH, 2015). Este problema, contudo, pode ser facilmente contornado com o uso de *gateways* redundantes.

Segundo Petajarvi et al. (2016) “o uso do enlace de comunicação é muito assimétrico entre conexões *uplink* e *downlink*. O *uplink* é mais utilizado para transmitir dados medidos, enquanto o *downlink* é usado praticamente para confirmações (ACK) na maioria das aplicações”. Assim é perceptível que o tráfego *uplink* é maior tanto em quantidade de mensagens quanto em conteúdo carregado. Mensagens *downlink* ainda podem ser utilizadas para enviar comandos de configurações do servidor para a rede em alguns casos.

Esta forma de topologia das LPWAN assemelha-se com as redes celulares tradicionais. Isto simplifica em muito a cobertura de grandes áreas, inclusive de nações inteiras, devido à possibilidade de reutilização da infraestrutura celular atual (torres, por exemplo) para implantação de *gateways* LPWAN (CENTENARO et al., 2015). O que também

auxilia na cobertura é o grande alcance dos enlaces das LPWAN, algo que diminui a quantidade de *gateways* para cobrir uma determinada área.

2.2.1 LoRa™

LoRa é uma tecnologia, de propriedade da Semtech, de espalhamento espectral derivado da modulação CSS (*Chirp Spread Spectrum*), otimizada para aplicações de longo alcance, baixo consumo de energia e baixa taxa de transmissão. Utilizando esta tecnologia, se faz possível a troca de taxa de transmissão por sensibilidade nos receptores, utilizando uma mesma largura de banda por canal, utilizando os três parâmetros chave da modulação (explicado mais abaixo). Ela implementa taxa de transmissão variável utilizando fatores de espalhamento ortogonais, fazendo possível projetar troca de taxa por alcance ou potência, otimizando a performance da rede, considerando uma largura de banda constante (SEMTECH, 2015). Atualmente, o circuito integrado produzido pela Semtech está sendo projetado para funcionar nas frequências 169 MHz, 433 MHz e 915 MHz nos Estados Unidos e 868 MHz na Europa (VANGELISTA; ZANELLA; ZORZI, 2015).

O manual da Semtech sobre a modulação da LoRa (SEMTECH, 2015) cita alguns fatores chave de sua modulação:

1. **Largura de banda escalável:** Modulação LoRa é escalável em largura de banda e frequência, sendo possível utilizar em aplicações de *frequency hopping*¹ de banda estreita e sequência direta de banda larga com apenas alguns ajustes de configuração nos dispositivos.
2. **Encapsulamento e consumo de energia constante:** Os mesmos estágios de amplificação de potência de pouco consumo e alta eficiência podem ser re-utilizados sem modificação. Ainda, devido ao ganho de processamento associado ao LoRa, a potência de saída do transmissor pode ser reduzida, comparado ao FSK.
3. **Alta robustez:** Seletividade *out-of-channel* de 90 dB e rejeição co-canal melhor que 20 dB podem ser conseguidas, comparados a 50 dB e -6 dB, respectivamente, obtidos com o FSK. Isto se dá devido ao alto produto largura de banda e tempo (> 1) e a natureza assíncrona do sinal LoRa.
4. **Resistência a multipercurso e desvanecimento:** Ideal para uso urbano e sub-urbano devido à característica banda larga do pulso LoRa, fazendo com que seja imune a multipercurso e desvanecimento.

¹ Técnica de espalhamento espectral consistindo na mudança da portadora utilizando uma sequência pseudoaleatória conhecida pelo transmissor e receptor.

5. **Resistência a efeito Doppler:** Efeito Doppler causa um pequeno desvio de frequência no pulso LoRa que introduz um desvio desprezível no eixo do tempo do sinal de banda base. Isto mitiga o requisito de fontes de *clock* com alta precisão.
6. **Capacidade de longo alcance:** Com potência de saída e vazão fixas, o cálculo de enlace do LoRa excede o do FSK. Quando levado em consideração com mecanismos de robustez à interferência e desvanecimento, isto pode facilmente ser traduzido em uma melhoria no alcance de quatro vezes ou mais.
7. **Capacidade de rede melhorada:** A modulação LoRa aplica fatores de espalhamentos ortogonais que permite múltiplos sinais espalhados serem transmitidos ao mesmo tempo e no mesmo canal com uma mínima degradação da sensibilidade do receptor. Os sinais com fatores de espalhamento diferentes aparecem como ruído no receptor alvo e podem ser tratados como tal.

A modulação pode ser especificada por três parâmetros: a largura de banda (BW), geralmente 125 kHz ou 250 kHz (podendo ser de 500 kHz também), o fator de espalhamento (FP), que vai de 7 a 12, e o parâmetro CR, que vai de 0 a 4 e determina a taxa do código corretor de erro, sendo $Codigo = \frac{4}{4+CR}$. Assim a taxa efetiva de dados com carga útil infinita é dada por:

$$R_b = FP \frac{Codigo}{\frac{2^{FP}}{BW}} [b/s]$$

Considerando o preâmbulo de cada pacote que permite sincronização de frequência e tempo no receptor, as taxas de transmissão variam de 0.3 kb/s até 11 kb/s, com largura de banda de 250 kHz e levando em consideração um único canal (VANGELISTA; ZANELLA; ZORZI, 2015). Este valor acaba não sendo o real, pois os *gateways* LoRa são capazes de realizar um processamento paralelo para cada fator de espalhamento, podendo aumentar esta taxa.

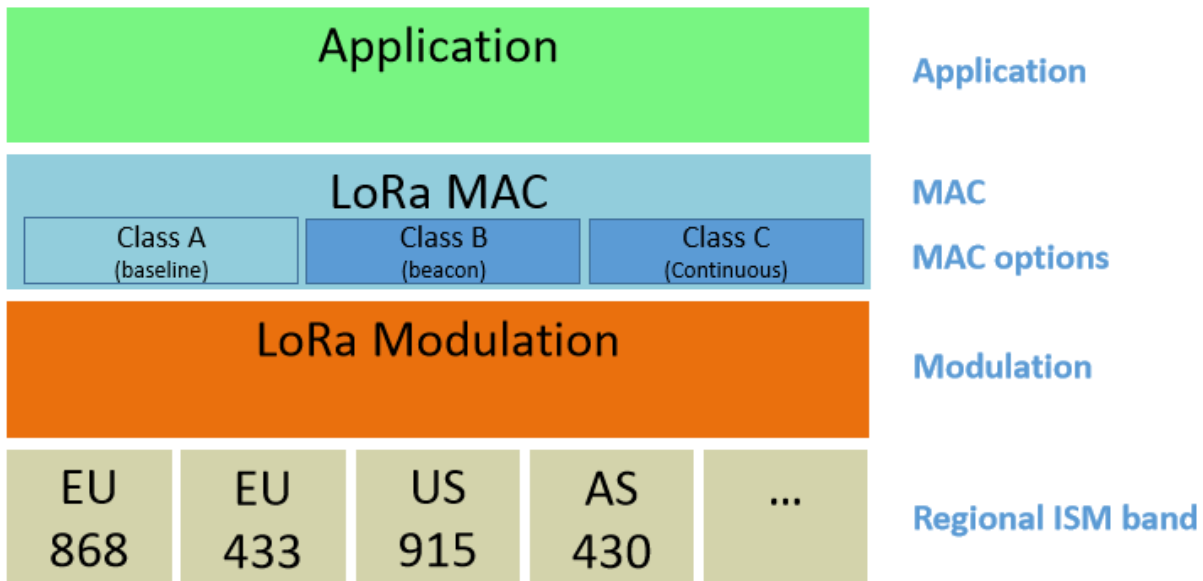
Segundo Semtech (2015) “LoRa é uma implementação de camada física e é agnóstica das implementações de camadas de nível mais alto. Isto permite que a LoRa coexista e opere com arquiteturas de rede existentes”. Desta forma, é possível existir uma gama de pilhas de protocolos que funcionem com a tecnologia LoRa, uma das características que fazem com que esta seja umas das tecnologias LPWAN com mais destaque. A pilha de protocolos LoRa mais difundida atualmente é a LoRaWAN.

2.3 LoRaWAN

LoRaWAN é uma arquitetura de rede aberta, cuja arquitetura pode ser vista na Figura 2. Suas regras são regidas pela LoRa Alliance, grupo formado por diversas empresas do ramo como IBM, Actility, Semtech e Microchip. Normalmente utilizado em topologias

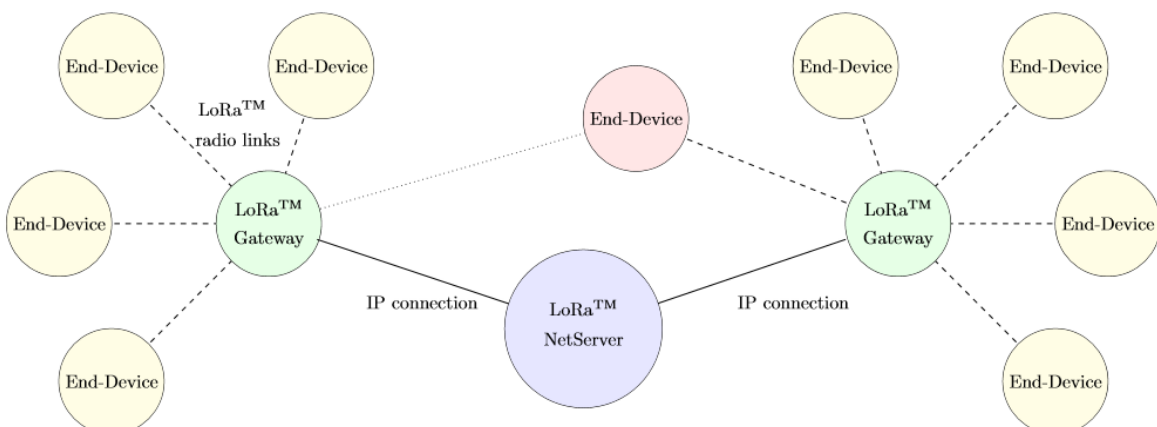
de rede do tipo estrela. Utiliza um modelo com *duty cycle*, ou seja, os dispositivos passam mais tempo em modo de espera do que transmitindo. Possui três tipos de dispositivos, os Nós (*end-devices, motes*), conectados por um link de único salto a um ou mais *gateways* que por sua vez possuem conexão, via rede IP, a um *Netserver* conforme a [Figura 3](#) ([LORA-ALLIANCE, 2016](#)).

Figura 2 – Arquitetura LoRaWAN.



Fonte: [Centenaro et al. \(2015\)](#)

Figura 3 – Arquitetura da rede LoRaWAN.



Fonte: [Centenaro et al. \(2015\)](#)

Os *gateways* chaveiam a troca de mensagens entre os nós e o *Netserver*. Os nós não necessitam assossiar-se com um *gateway* para ter acesso à rede (sendo os *gateways* transpa-

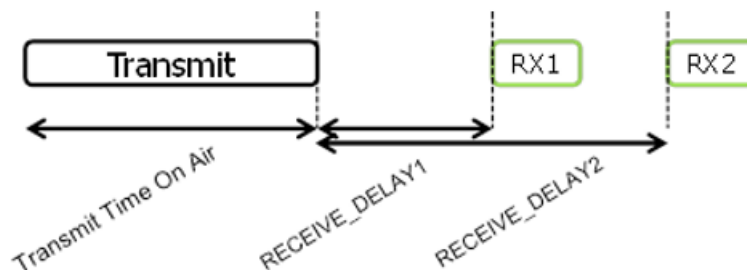
rentes aos nós), mas sim a um *Netserver*. O *gateway* atua somente como um encaminhador de mensagens decodificadas corretamente a um *Netserver* associado, adicionando algumas informações extra, como relação sinal-ruído, RSSI, etc. Com estas associações, o *Netserver* é capaz de filtrar mensagens duplicadas, quando por exemplo, um nó é visto por mais de um *gateway*. Os *gateways* também possuem a capacidade de realizar um processamento paralelo de até nove canais. Cada nó utiliza um canal combinado com um espalhamento espectral (CENTENARO et al., 2015).

Esta configuração move toda a complexidade dos nós para o *Netserver*, proporcionando a criação de dispositivos simples. Segundo Centenaro et al. (2015) “os nós podem se mover livremente por células servidas por *gateways* diferentes sem gerar tráfego de sinalização extra. Por fim, percebemos que o aumento do número de *gateways* que servem um certo nó aumentará a confiabilidade da sua conexão com o *Netserver*, que pode ser útil para algumas aplicações críticas”.

Os nós LoRaWAN podem implementar três classes de operação: Classe A, B e C. Todos devem implementar a classe A, e dispositivos que implementam mais de uma classe são chamados de “*higher Class end-devices*” (LORA-ALLIANCE, 2016).

- **Nós classe A:** Dispositivos classe A permitem transmissões bidirecionais, onde cada transmissão *uplink* é seguida de duas curtas janelas de recepção. A transmissão é baseada nas necessidades de comunicação do dispositivo com uma pequena variação baseada em um tempo aleatório (protocolo do tipo ALOHA). Esta classe consome a menor quantidade de bateria para aplicações que só necessitam de transmissões *downlink* logo após que o dispositivo faz uma transmissão *uplink* (e.g. ACK). Um exemplo de comunicação de um dispositivo classe A pode ser vista na Figura 4.

Figura 4 – Janelas de recepção do dispositivo classe A.

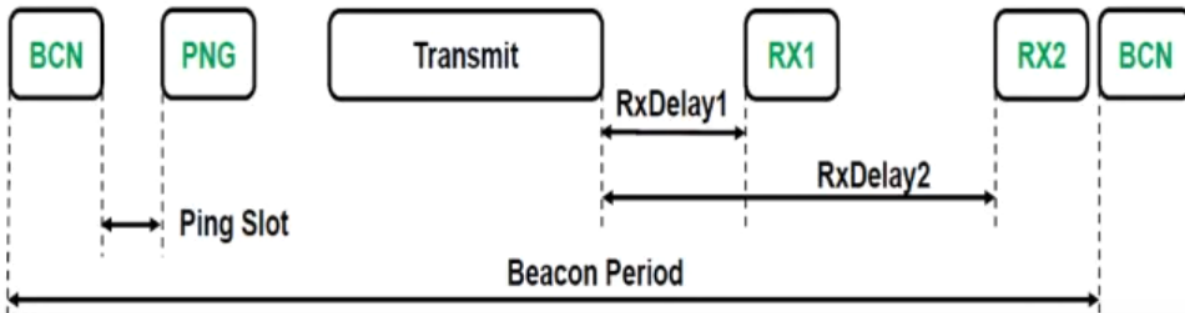


Fonte: LoRa-Alliance (2016)

- **Nós classe B:** Dispositivos classe B possuem uma janela de recepção extra, em relação a classe A, tempos fixos, baseadas em uma sincronização com o *gateway*. Todos os *gateways* são sincronizados e enviam *beacons* periódicos para manter a

célula sincronizada (protocolo do tipo ALOHA-slotted). Após receber um *beacon*, o dispositivo abre uma janela extra, chamada de *ping-slot*. A principal ideia desta classe é ter o dispositivo disponível para uma recepção em um instante previsível. Um exemplo de comunicação de um dispositivo classe B pode ser vista na [Figura 5](#).

Figura 5 – Janelas de transmissão de dispositivos classe B.



Fonte: [LoRa-Alliance \(2016\)](#)

- **Nós classe C:** Dispositivos classe C possuem praticamente uma janela de recepção aberta, estando fechada somente quando está transmitindo. Essa classe de operação é indicada para dispositivos conectados diretamente à rede elétrica devido seu alto consumo de energia e para aplicações que necessitam uma menor latência no *downlink*. Dispositivos classe C não podem implementar a classe B. Mais detalhes sobre as janelas de transmissão da classe C podem ser vistos em [subseção 2.3.5](#).

Todas as terminologias e formatos das mensagens a seguir são utilizados em todas as classes de dispositivos, a não ser quando explicitamente indicados.

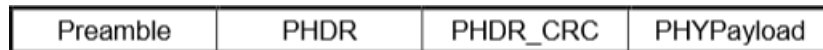
2.3.1 Mensagens da camada física

As mensagens LoRa seguem esta terminologia:

- **Uplink:** Mensagens enviadas pelos nós ao *Netserver*, chaveadas por um ou mais *gateways*. Estas mensagens utilizam um cabeçalho da camada física do LoRa (PHDR) mais um CRC deste cabeçalho (PHDR_CRC). O *payload* é protegido por outro CRC. Todos estes campos são inseridos pelo *transceiver* conforme a [Figura 6](#).
- **Downlink:** Mensagens enviadas pelo *Netserver* para somente um nó e chaveada por um único *gateway*. Estas mensagens utilizam um cabeçalho da camada física do LoRa (PHDR) mais um CRC deste cabeçalho (PDHR_CRC) conforme a [Figura 7](#).

Figura 6 – Estrutura do *uplink* da camada física.

Fonte: [LoRa-Alliance \(2016\)](#)

Figura 7 – Estrutura do *downlink* da camada física.

Fonte: [LoRa-Alliance \(2016\)](#)

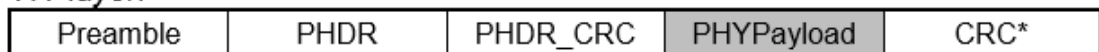
Para cada transmissão *uplink*, o dispositivo abre duas pequenas janelas de recepção (RX1 e RX2). Os temporizadores de início da janela de recepção possuem um período configurado e iniciam suas contagens no final da transmissão do último bit de *uplink*, conforme a [Figura 4](#). As janelas RX1 e RX2 ainda podem receber configurações de canal diferentes entre si, como frequência central e espalhamento espectral.

2.3.2 Mensagens da camada de enlace

Todas as mensagens LoRa carregam um *payload* da camada física, que começa com um octeto de cabeçalho MAC (MHDR), seguido pelo *payload* MAC (MACPayload) e terminando com um código de integridade da mensagem (MIC) com quatro octetos de tamanho. A [Figura 8](#) mostra todos os elementos da mensagem LoRa.

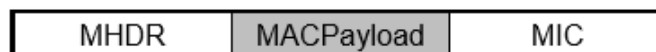
Figura 8 – Formato dos elementos da mensagem LoRa.

Radio PHY layer:

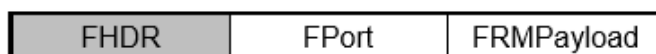


CRC* somente em mensagens de *uplink*

PHYPayload:



MACPayload:



FHDR:



Fonte: [LoRa-Alliance \(2016\)](#)

O MHDR especifica o tipo da mensagem (MType) e a versão principal (Major) do

LoRaWAN do frame. O LoRaWAN especifica seis tipos de mensagens MAC. A [Tabela 1](#) relaciona o valor do MType com o tipo de mensagem correspondente. Mensagens com confirmação necessitam de um ACK pelo receptor enquanto mensagens sem confirmação não necessitam de um ACK. Mensagens proprietárias podem ser implementadas fora do formato padrão, não sendo compatíveis com mensagens LoRaWAN, havendo comunicação somente com dispositivos específicos.

| MType | Descrição |
|-------|--------------------------------------|
| 000 | Join Request |
| 001 | Join Accept |
| 010 | Dado <i>Uplink</i> Sem Confirmação |
| 011 | Dado <i>Downlink</i> Sem Confirmação |
| 100 | Dado <i>Uplink</i> Com Confirmação |
| 101 | Dado <i>Downlink</i> Com Confirmação |
| 110 | Reservado para uso futuro |
| 111 | Formato Proprietário |

Tabela 1 – Tipos de mensagens MAC

O MACPayload, ou “quadro de dados”, contém o cabeçalho do quadro (FHDR), seguido pela porta (FPort) e pelo *payload* do quadro (FRMPayload). Os tamanhos dos elementos do MACPayload são descritos na [Figura 9](#).

Figura 9 – Estrutura do MACPayload.

| Size (bytes) | 7..23 | 0..1 | 0..N |
|--------------|-------|-------|------------|
| MACPayload | FHDR | FPort | FRMPayload |

Fonte: [LoRa-Alliance \(2016\)](#)

Na [Figura 9](#), o tamanho máximo do payload (N) varia baseado na região específica de atuação conforme [LoRa-Alliance \(2016, Sessão 7\)](#) e deve seguir a seguinte regra:

$$N \leq M - 1 - (FHDR_em_octetos)$$

onde M é o tamanho máximo do *payload* do MAC.

O FHDR contém o endereço curto do dispositivo (DevAddr), um octeto de controle de quadro (FCtrl), um contador de quadros de dois octetos (FCnt) e um campo de até 15 octetos usado para transportar comandos MAC (FOpts), mostrados na [Figura 10](#).

- **DevAddr:** Endereço de 32 bits separados em dois campos: Identificador de rede (NwkID), que é utilizado para evitar endereços iguais em redes próximas no caso de

Figura 10 – Estrutura do FHDR.

| | | | | |
|---------------------|---------|-------|------|-------|
| Size (bytes) | 4 | 1 | 2 | 0..15 |
| FHDR | DevAddr | FCtrl | FCnt | FOpts |

Fonte: [LoRa-Alliance \(2016\)](#)

Figura 11 – Estrutura do DevAddr.

| | | |
|---------------------|----------|---------|
| Bit# | [31..25] | [24..0] |
| DevAddr bits | NwkID | NwkAddr |

Fonte: [LoRa-Alliance \(2016\)](#)

roaming, e o Endereço de rede (NwkAddr) do nó, que pode ser atribuído arbitrariamente, como na [Figura 11](#).

- **FCtrl**: O quadro de controle possui cinco campos, conforme a [Figura 12](#).

Figura 12 – Estrutura do FCtrl.

| | | | | | |
|-------------------|-----|-----------|-----|----------|----------|
| <i>Downlink:</i> | | | | | |
| Bit# | 7 | 6 | 5 | 4 | [3..0] |
| FCtrl bits | ADR | ADRACKReq | ACK | FPending | FOptsLen |
| <i>Uplink:</i> | | | | | |
| Bit# | 7 | 6 | 5 | 4 | [3..0] |
| FCtrl bits | ADR | ADRACKReq | ACK | RFU | FOptsLen |

Fonte: [LoRa-Alliance \(2016\)](#)

- ADR (Adaptative Data Rate): Campo responsável por ativar o controle adaptativo de taxa de transmissão do LoRa.
- ADRACKReq: Bit responsável por validar se o ADR está funcionando normalmente caso a taxa otimizada pelo ADR seja maior que a taxa padrão do dispositivo. O dispositivo conta um certo número de mensagens não respondidas pelo servidor até um limite, e então configura o ADRACKReq para 1.
- ACK: Resposta ao ADRACKReq.
- FPending: Somente usado no *downlink*, indica que o *gateway* possui mais dados prontos para serem transmitidos para que o dispositivo abra mais uma janela de transmissão o mais rápido possível.

- **FOptsLen**: Indica o tamanho do FOpts. Se for 0, indica que não há o campo FOpts. Se for diferente de 0, ou seja, existem comandos MAC no FOpts, FPort deve ser ausente ou diferente de 0, pois a porta 0 não pode ser utilizada.
 - **RFU**: Reservado para uso futuro. Utilizado no classe B para indicar que o dispositivo está com a classe B ativa.
- **Fcnt**: Cada nó possui dois contadores: um para *uplink* (FCntUp), incrementado pelo dispositivo e um para *downlink* (FCntDown), incrementado pelo *Netserver*. O *Netserver* usa o FCntUp para gerar o próximo FCntDown. Os contadores são reiniciados para 0 após um *Join Accept*.
 - **FOpts**: Transmite comandos MAC com tamanho máximo de quinze octetos acoplados (*piggyback*) no frame de dados. Comandos MAC não podem estar presentes no campo *payload* e no campo de opções simultaneamente.

O FPort define uma porta para a aplicação. Desta forma, é possível executar diversas aplicações em um único nó e ser capaz de diferenciá-las. Se o campo FRMPayload não é vazio, isto indica que o campo FPort deve estar presente. Se presente, o valor 0 no FPort indica que o FRMPayload contém somente comandos MAC. FPort com valores de 1 até 233 (0x01 até 0xDF) estão disponíveis para as aplicações. Valores de 223 até 255 (0xE0 até 0xFF) são reservados para extensão de aplicações padrões futuras.

As mensagens LoRaWAN passam por uma criptografias AES com chaves de tamanho de 128 bits conforme algoritmo descrito no [IEEE... \(2016, Anexo B\)](#). A criptografia é utilizada no MACPayload antes do cálculo do MIC caso o quadro carregue um *payload*. Esta é a criptografia de rede que utiliza a chave de sessão de rede (NwkSKey). Cada dispositivo possui sua própria NwkSKey. Já o FRMPayload é criptografado com a chave de sessão de aplicação (AppSKey). As duas chaves utilizadas ao mesmo tempo impossibilita que o provedor da rede acesse os dados da aplicação. Dessa forma, a NwkSKey é obrigatória, enquanto a AppSKey é facultativa à aplicação.

2.3.3 Ativação na rede

Existem duas formas de um dispositivo (nó) participar de uma rede: ativação pelo ar (OTAA) e ativação personalizada (ABP). Na **ativação pelo ar** o dispositivo realiza um procedimento de entrada na rede sempre que perder informações sobre o contexto da sessão. Este procedimento requer um identificador único (DevEUI) no formato IEEE EUI64, um identificador de aplicação (AppEUI) no formato IEEE EUI64 e uma chave AES-128 (AppKey) que irá derivar-se nas chaves NwkSKey e AppSKey. O procedimento consiste em duas mensagens MAC:

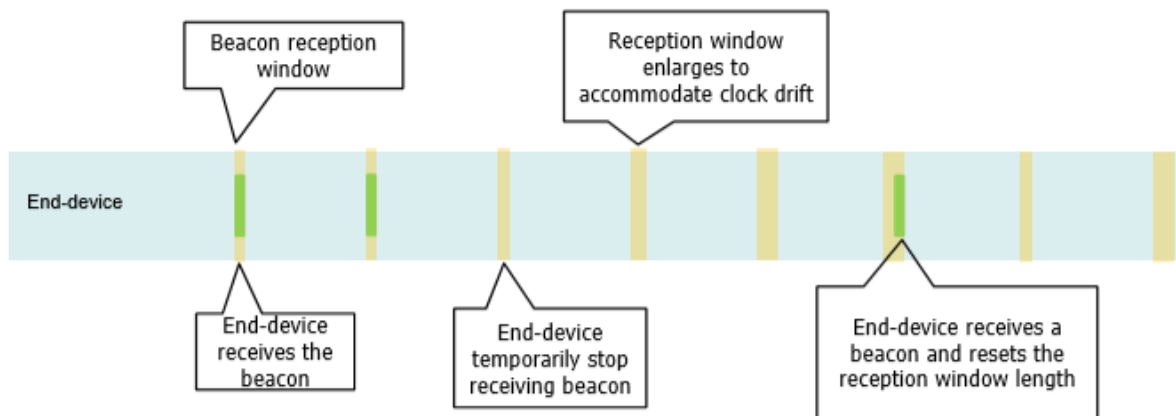
- *Join-request*: Gerada pelo dispositivo. Contém o AppEUI, o DevEUI e um valor aleatório de dois bytes (DevNonce). O *Netserver* mantém os últimos DevNonce de cada dispositivo afim de ignorar requisições repetidas com os mesmos valores. Este mecanismo previne ataques de repetição com intenção de desconectar o dispositivo da rede. Esta mensagem não é criptografada.
- *Join-accept*: Gerada pelo *gateway*. Contém um valor aleatório de três bytes (AppNonce), um identificador de rede (NetID), um endereço de dispositivo (DevAddr), um atraso entre TX e RX (RxDelay), o DLsettings e uma lista opcional de canais (CFList). O AppNonce é utilizado somente na derivação do AppKey em NwkSKey e AppSKey. O NetID é usado na derivação das chaves de sessão e se divide da seguinte maneira: os sete bits menos significativos são iguais ao endereço curto do dispositivo e os outros são arbitrariamente escolhidos. O DLsetting contém 1 byte e é responsável pela configuração do *downlink*. O RxDelay configura o atraso entre o fim de uma transmissão de *uplink* e o início de uma nova abertura de recepção. O CFList é uma lista de frequências utilizadas pelo *gateway* e varia com a região.

Na **ativação personalizada** não há procedimento de requisição. O DevAddr, NwkSKey e AppSKey são armazenados previamente no dispositivo ao invés do DevEUI, AppEUI e AppKey. Cada dispositivo deve ter uma combinação única de NwkSKey e AppSKey para não comprometer a segurança do sistema. Nesta forma de ativação, o dispositivo está pronto para participar da rede assim que iniciado.

2.3.4 Aquisição e monitoramento do *beacon*

Antes de trocar de modo de funcionamento de classe A para classe B, o dispositivo deve receber pelo menos um *beacon* da rede para alinhar sua referência temporal interna com a da rede. Uma vez na rede, o dispositivo deve receber periodicamente um *beacon* para realizar um ajuste no seu *clock* interno. Uma vez sem receber um *beacon*, o dispositivo pode manter sua operação em classe B por até duas horas. Para garantir a recepção sem o ajuste de sincronismo, o dispositivo aumenta o tempo de recepção com intenção de alargar o tamanho da janela para acomodar o desvio de tempo entre o dispositivo e a rede. Esta operação pode ser vista na [Figura 13](#). A recepção de qualquer *beacon* estende a duração da operação por mais duas horas.

Todo *beacon* é composto de um preâmbulo um pouco maior que o padrão (dez símbolos não modulados) e um campo de *payload* que varia conforme a região de operação. Este *payload* contém, entre outras informações, o período do *beacon*, que pode variar de 1 a 4294967295 segundos. Todos os *beacons* não possuem cabeçalho e CRC atribuídos pela camada física do LoRa. Informações específicas sobre o funcionamento da classe B podem ser encontradas em [LoRa-Alliance \(2016, Class B - Beacon\)](#).

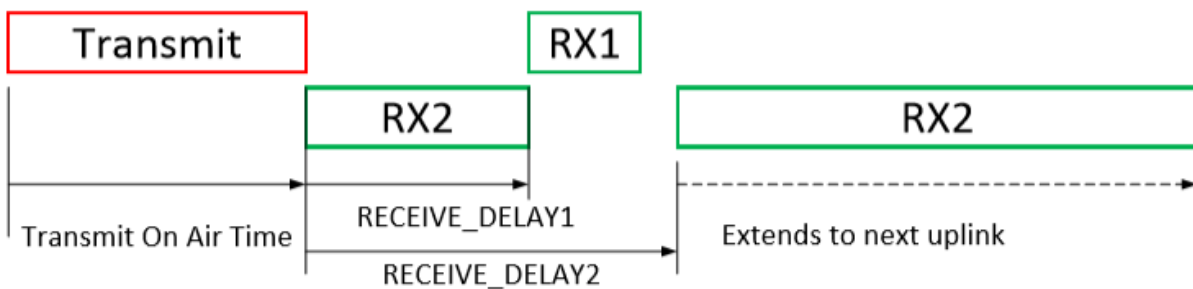
Figura 13 – Operação de perda de *beacon*.

Fonte: LoRa-Alliance (2016)

2.3.5 Janelas de transmissão na classe C

Dispositivos classe C estão sempre com a janela de recepção 2 aberta a não ser que esteja transmitindo ou recebendo na janela 1. Não existe nenhuma mensagem específica para identificar que um dispositivo esteja operando em classe C. Isto deve ser passado como informação no procedimento de entrada na rede. O comportamento das janelas na classe C é mostrado na Figura 14.

Figura 14 – Comportamento das janelas de recepção de dispositivos de classe C.



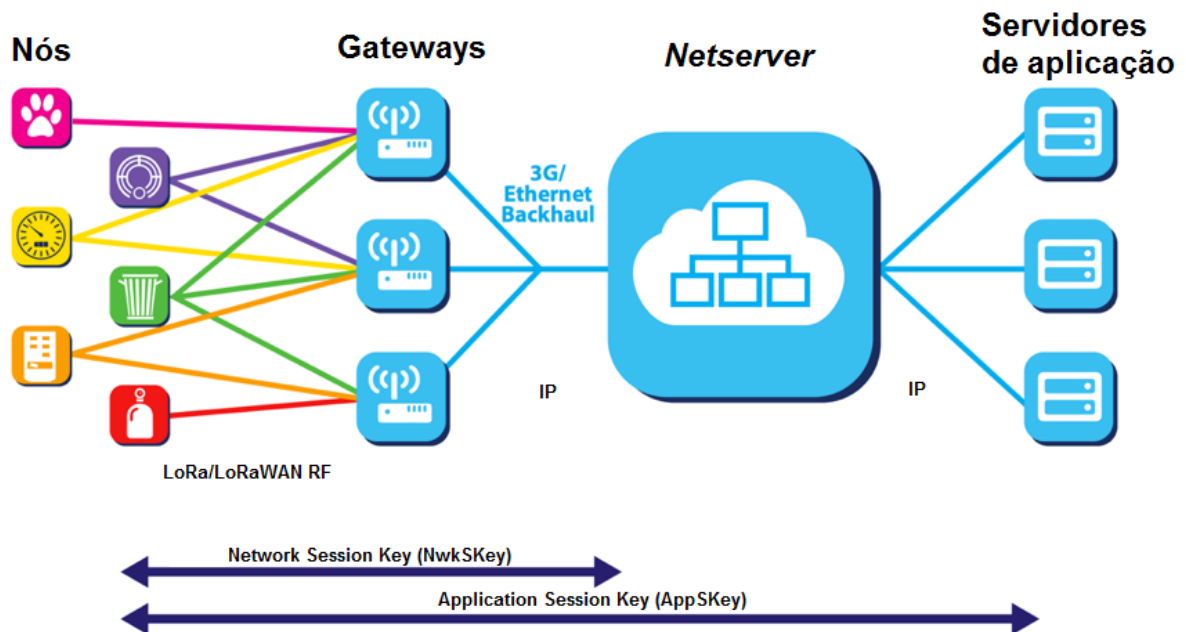
Fonte: LoRa-Alliance (2016)

2.4 Topologia da rede

A organização eficiente de grandes volumes de mensagens pequenas, como o gerado por aplicações da IoT, é tema de vários trabalhos em andamento. Há plataformas, tanto livres quanto pagas, para a implementação destas aplicações. No caso de *Netservers* para LoRaWAN, o mesmo se repete. Pode-se citar o próprio *Netserver* pago da Semtech,

projetos já implementados para uso público, com interface web para configurações, e alguns projetos livres disponíveis em repositórios públicos na rede. A arquitetura majoritariamente utilizada em projetos LoRaWAN pode ser vista na [Figura 15](#).

Figura 15 – Arquitetura de rede comum para projetos LoRaWAN.



Fonte: [LoRa-Alliance \(2017\)](#)

Para o funcionamento de uma aplicação em uma rede LoRaWAN, os seguintes passos devem acontecer:

1. A rede LoRaWAN deve estar funcionando, ou seja, os endpoints devem ser capazes de comunicar-se com ao menos 1 *gateway* e esse *gateway* deve ser capaz de comunicar-se com o *Netserver*;
2. O *endpoint* transmite periodicamente uma ou mais informações em suas respectivas portas, cada porta representando uma aplicação;
3. Após passar pelo *gateway* e *Netserver*, o servidor de aplicação recebe as mensagens e pode agrupá-las da forma que for desejado: por dispositivo, utilizando o DevEUI, ou por proprietário do dispositivo, utilizando o AppEUI. O modo de agrupamento pode variar de acordo com as intenções do proprietário da rede e dos dispositivos;
4. Após isso, esses grupos de mensagens podem ser divididos pela porta utilizada pelo dispositivo, que representa de qual aplicação é a informação contida na mensagem;
5. Finalmente, os dados podem ser tratados e distribuídos aos usuários interessados.

3 Implantação de rede LoRaWAN

Antes de realizar qualquer tipo de teste com a tecnologia LoRa, foi necessário implantar uma rede LoRaWAN. Foram pesquisados diversos equipamentos como transceivers LoRa e *gateways*, máquinas para hospedar serviços e softwares para servidores de rede (*Netserver*) e aplicações. Tais opções foram avaliadas e escolhidas com base em custo, facilidade para montagem do cenário e utilização geral pela comunidade. Neste capítulo é apresentado o desenvolvimento e a implantação da rede LoRaWAN utilizada nos testes, desde a escolha e aquisição do hardware até a escolha dos softwares.

3.1 Equipamentos

Nesta sessão são apresentados todos os equipamentos utilizados na montagem da rede LoRaWAN utilizada no projeto: *endpoints*, *gateways* e *servidores*.

3.1.1 *Endpoints*

Para a construção dos endpoints, baseando-se na alta disseminação e a existência de uma base de código-fonte disponível em comunidades de software livre e fóruns sobre LoRaWAN, a plataforma Arduino Uno foi escolhida. Após alguns testes com o código do sistema de gerenciamento de aplicações LoRaWAN configuráveis, resolveu-se por utilizar outra versão da plataforma, a Arduino Mega, devido à sua maior disponibilidade de memória, pois o sistema ocupava inicialmente aproximadamente 50% da memória dinâmica (1030 bytes). Como o sistema aloca muitas variáveis que escalam de tamanho baseado no número de aplicações configuradas, preferiu-se utilizar uma plataforma com mais memória. É importante ressaltar que futuras versões podem ser otimizadas para a utilização de placas ArduinoUno.

Após uma busca nos fornecedores de soluções LoRa, dois modelos de interfaces LoRa para testes foram encontrados:

- **RFM95W:** Módulo da HopeRF, integra o transceiver SX1276 da Semtech e *front-end* RF para faixa de 915MHz, formando um módulo de 16x16 mm com terminação de 50 Ω para conexão de antena e interfaces digitais para integração com o processador de aplicação ([HOPERF, 2014](#)). O módulo possui até -148dBm de sensibilidade e, devido a um PA de +14 dBm, +20 dBm de potência de saída RF, gerando um *link budget* de 168dB. Sua pequena dimensão facilita a integração a novos projetos de hardware. Para integração com o Arduino, foi necessária a aquisição de uma placa adaptadora.

- **inAir9B:** Módulo da Modtronix que também integra o transceiver SX1276 da Semtech e *front-end* RF para faixa de 915MHz, porém implementam um módulo de maior dimensão (27,3x24,51mm) (MODTRONIX, 2014). Assim como o RFM95W, o inAir9B também possui terminação de 50Ω para conexão de antena e interfaces digitais para integração com o Arduino. Possui -139 dBm de sensibilidade e até +20 dBm de potência de saída RF, com cálculo de enlace de 159 dB. Estes módulos permitem fácil integração para prototipagem rápida, não necessitando adaptações ou placas extra. As características de sensibilidade e potência são as mesmas do RFM95W.

As conexões dos módulos ao Arduino se dão pela ligação de sinais de sinalização digital, uma interface serial SPI, alimentação de 3,3 V e o terra, como apresentado nas Tabela 2 e Tabela 3.

Tabela 2 – Conexão do módulo Modtronix inAir9B ao Arduino Mega.

| inAir9B | Arduino Mega |
|---------|--------------|
| D0 | 2 |
| D1 | 3 |
| D2 | 4 |
| CS | 10 |
| S0 | 50 |
| S1 | 51 |
| CK | 52 |
| 3V3 | 3V |
| 0V | GND |

Tabela 3 – Conexão do módulo HopeRF RFM95W ao Arduino Mega.

| RFM95W | Arduino Mega |
|--------|--------------|
| MISO | 50 |
| MOSI | 51 |
| SCK | 52 |
| NSS | 10 |
| RES | 9 |
| DIO0 | 2 |
| DIO1 | 3 |
| DIO2 | 4 |
| 3.3V | 3V |
| GND | GND |

3.1.2 Gateway

O hardware do *gateway* é um pouco mais complexo e caro que os endpoints, já que um *gateway* deve ser capaz de processar até 9 canais paralelamente. Este hardware é disponibilizado pela Semtech apenas para integradores de soluções LoRa. Devido à dificuldade em adquirir um *gateway* LoRa, os primeiros testes da rede LoRaWAN usaram um protótipo de *gateway* operando em um único canal, que é conhecido na comunidade como Single Channel Gateway ([TELKAMP, 2016](#)).

Dada a boa relação custo benefício e o fato de que o software disponível para o *gateway* executa primariamente em sistemas Linux, a plataforma base selecionada para os gateways foi a RaspberryPi. Foram utilizadas placas do modelo RaspberryPi B+ devido à disponibilidade. De modo similar aos endpoints, os módulos RF foram conectados à plataforma, conforme as [Tabela 4](#) e [Tabela 5](#). Como software foi utilizado o Single Channel Packet Forwarder, responsável por realizar as funções de *gateway*, adicionando campos de informação sobre o enlace LoRa, em um dispositivo capaz de utilizar um único canal por vez e encaminhar o pacote para um ou mais servidores. Para alterar o canal utilizado por este *gateway*, é necessário recompilar o software e reiniciar o processo no sistema operacional. O Single Channel Gateway, devido à simplicidade, torna-se muito barato e ideal para testes de enlace com redes simplificadas. Em contrapartida, funcionalidades como a taxa de transmissão adaptativa (ADR), mensagens de *downlink*, confirmações e capacidade de multi-canais ficam impossibilitadas.

Tabela 4 – Conexões do módulo Modtronix inAir9B com a RaspberryPi B+.

| inAir9B | RaspberryPi |
|---------|--------------|
| D0 | 7 (GPIO 4) |
| RT | 11 (GPIO 17) |
| CS | 24 (GPIO 8) |
| 0V | 25 (GND) |
| SI | 19 (GPIO 10) |
| SO | 21 (GPIO 9) |
| CK | 23 (GPIO 11) |
| 3V3 | 17 (3.3V) |

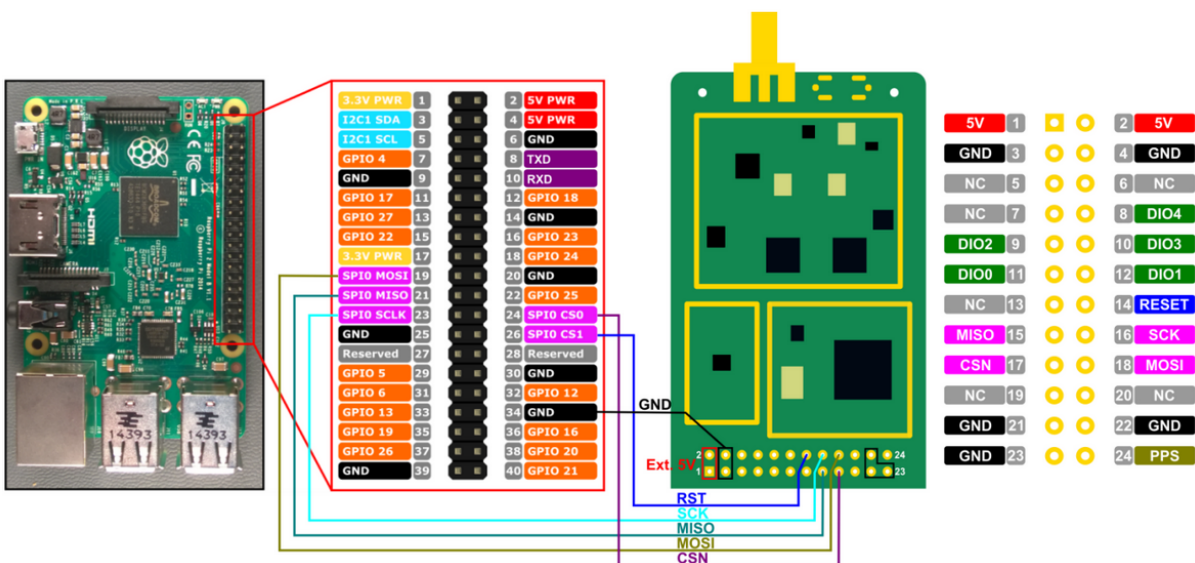
Em uma segunda etapa do trabalho, foi possível a aquisição de um *gateway* LoRa multicanal. O RHF0M301 da RisingHF ([RISINGHF, 2015](#)) é um módulo *gateway* LoRa com 10 canais (8 multicanais com espalhamento espectral, 1 canal LoRa padrão e 1 canal FSK). Com dimensão de 15x8x5 cm, possui um processador multicanal SX1301 da Semtech e front-end SX1255/7 da Semtech TX e RX. O *gateway* ainda vem com 24 pinos para interface digital com o controlador principal e terminação de 50Ω para conexão de antena com potência máxima de saída de 27 dBm e sensibilidade de até -141,5 dBm. O layout do módulo e suas conexões com a Raspberry podem ser vistos na [Figura 16](#). O

Tabela 5 – Conexões do módulo HopeRF RFM95W com a RaspberryPi B+.

| RFM95W | RaspberryPi |
|--------|--------------|
| MISO | 21 (GPIO 9) |
| MOSI | 19 (GPIO 10) |
| SCK | 23 (GPIO 11) |
| NSS | 22 (GPIO 25) |
| RES | 11 (GPIO 17) |
| GND | 6 (GND) |
| DIO0 | 7 (GPIO 4) |
| 3V3 | 1 (3V3) |

gateway vem acompanhado de um guia para instalação rápida em plataformas Raspberry Pi. Seguindo o guia, foram utilizados os softwares da própria Semtech: *lora_gateway* e *packet_forwarder*. O software *lora_gateway* é o responsável por realizar testes para verificar se há alguma irregularidade nas conexões eletrônicas do módulo com a plataforma. O software *packet_forwarder* adiciona as informações sobre o enlace LoRa e encaminha o pacote para um ou mais servidores utilizando o protocolo UDP Packet Forwarder, mais detalhado na [subseção 3.2.2](#). Nele que configuramos quais canais serão utilizados pelo *gateway*. Durante a utilização da rede, o *gateway* apresentou um problema no *software* embarcado, parando de encaminhar pacotes após aproximadamente 3 dias funcionando ininterruptamente. Como este *software* não foi desenvolvido neste trabalho, optou-se por não investigar a origem deste problema e por reiniciar o *gateway* periodicamente.

Figura 16 – Layout do módulo RHF0M301 e conexões com a Raspberry Pi B+.



Fonte: RisingHF (2015)

3.1.3 Servidor

Ao longo do projeto foram utilizados alguns modelos diferentes de *Netserver*. Uma das primeiras alternativas foi a utilização dos servidores da TTN (The Things Network), empresa que investe na tecnologia atualmente. Desta forma, o *gateway* envia os pacotes aos servidores da TTN e este cuida de todo o processo de *Netserver*. Utilizando uma interface web, são feitos os processos de registro de sessão, para o caso de autenticação personalizada, e de manipulação das informações adquiridas.

Visando ter maior controle sobre os processos servidores de rede e aplicação, optou-se por instalar estes servidores localmente em uma máquina virtual Ubuntu dentro da rede interna do IFSC (OpenStack). Nesta máquina foi configurado um broker do protocolo `packet_forwarder` chamado LoRa Gateway Bridge (BROCAAR, 2017b). Este broker funciona como uma ponte de comunicação entre o *gateway* e o *Netserver*, extraindo os objetos no formato JSON (ver subseção 3.2.1) destas mensagens e publicando-os como tópicos MQTT¹, disponibilizando estas mensagens aos servidores de rede e aplicação e outros serviços que se registrarem para recebê-las. Utilizando o Node-Red, ferramenta visual desenvolvida em `node.js` para tratar fluxos de dados, as mensagens MQTT são lidas e registradas. Este protótipo de servidor não atendia às especificações LoRaWAN e servia apenas para adquirir os dados recebidos. Entre os problemas deste protótipo destaca-se a dificuldade em decodificar o payload LoRa, sendo que, durante esta etapa, somente eram visíveis as informações sobre o enlace LoRa geradas pelo gateway.

Devido a problemas de instabilidade e ao desligamento da rede OpenStack prevista para o fim do ano de 2016, o *Netserver* foi migrado para outra rede. Desta forma, optou-se por uma máquina virtual Ubuntu na rede de serviços AWS da Amazon, disponibilizada pela empresa Teltec Solutions. Simultaneamente, foram estudados projetos de *Netserver* gratuitos para substituir o atual protótipo de servidor. Assim, foi optado por utilizar o LoraServer, projeto gratuito de código aberto de um *Netserver* para redes LoRaWAN. Este projeto foi testado ainda no servidor dentro do IFSC, testando suas configurações e atualizações que aconteciam com o tempo e finalmente implantado com a migração para o servidor na Amazon. Informações mais detalhadas do LoraServer são encontradas na subseção 3.2.3. Junto com o LoraServer, é disponibilizado um servidor de aplicação compatível, que é visto em mais detalhes na subseção 3.2.4.

3.2 Serviços na rede

Nesta sessão, encontram-se os principais serviços e softwares utilizados na rede LoRaWAN implantada.

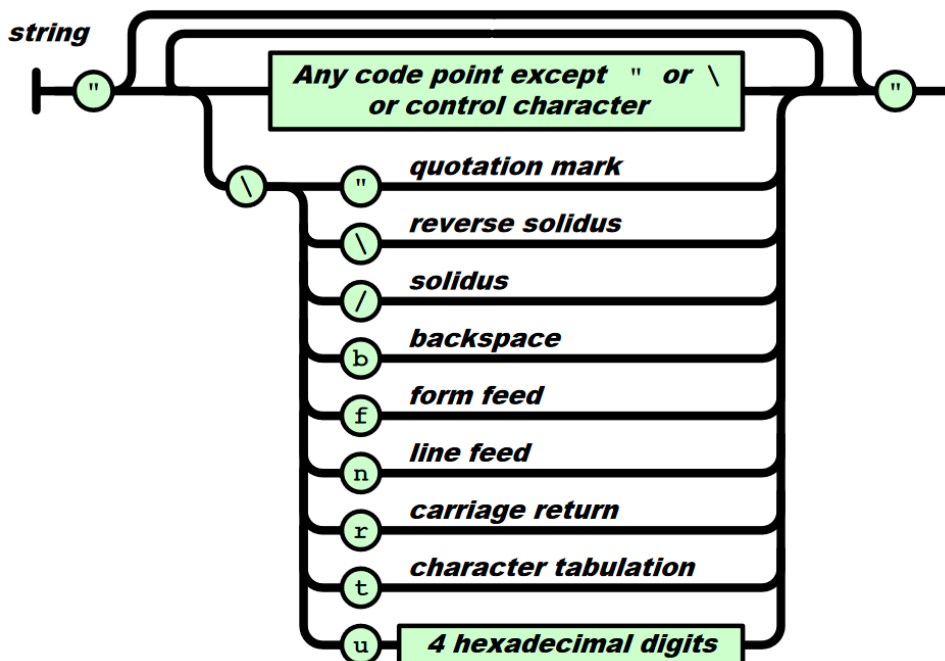
¹ O Message Queuing Telemetry Transport

3.2.1 O formato JSON

JSON (JavaScript Object Notation) (BRAY, 2014) é um formato para troca de dados na forma de texto UNICODE que independe da linguagem utilizada. Uma mensagem JSON pode ser constituída com os seguintes valores (ECMA, 2013):

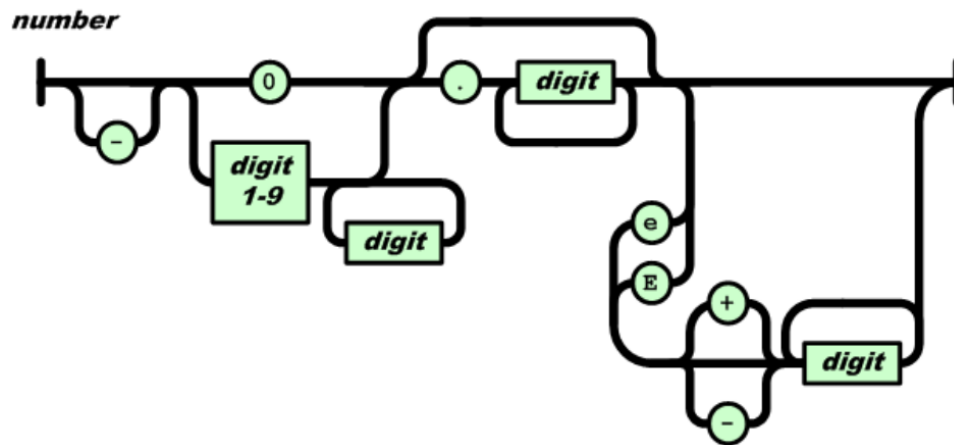
- String: Caracteres entre aspas duplas, salvo exceções vistas na Figura 17;
- Numeral: Numerais seguindo o formato mostrado na Figura 18;
- *Array*: Coleção de valores ordenados, iniciado com [, separados por , e terminado com], conforme Figura 19. Representa na maioria das linguagens vetores e listas;
- Objeto: Conjunto desordenado de pares com um nome (*string*) e um valor respectivo. Assemelha-se a estruturas encontradas em diversas linguagens como **struct**, **object**, dicionário, etc. É iniciado com {, terminado com }, nome e valor são separados por : e pares separados por ,, conforme Figura 20;
- Uma das 3 palavras reservadas: **null**, **true** ou **false**.

Figura 17 – Estrutura de uma *string* JSON.

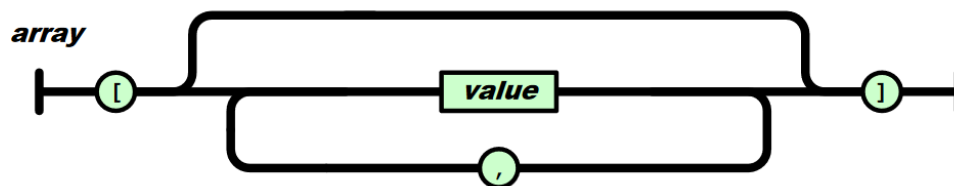


Fonte: ECMA (2013)

Figura 18 – Estrutura de um numeral JSON.

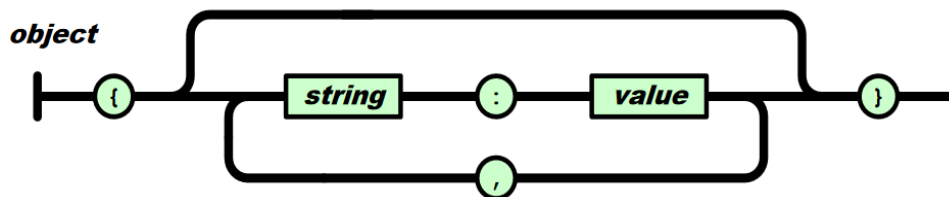


Fonte: ECMA (2013)

Figura 19 – Estrutura de um *array* JSON.

Fonte: ECMA (2013)

Figura 20 – Estrutura de um objeto JSON.



Fonte: ECMA (2013)

3.2.2 UDP Packet Forwarder

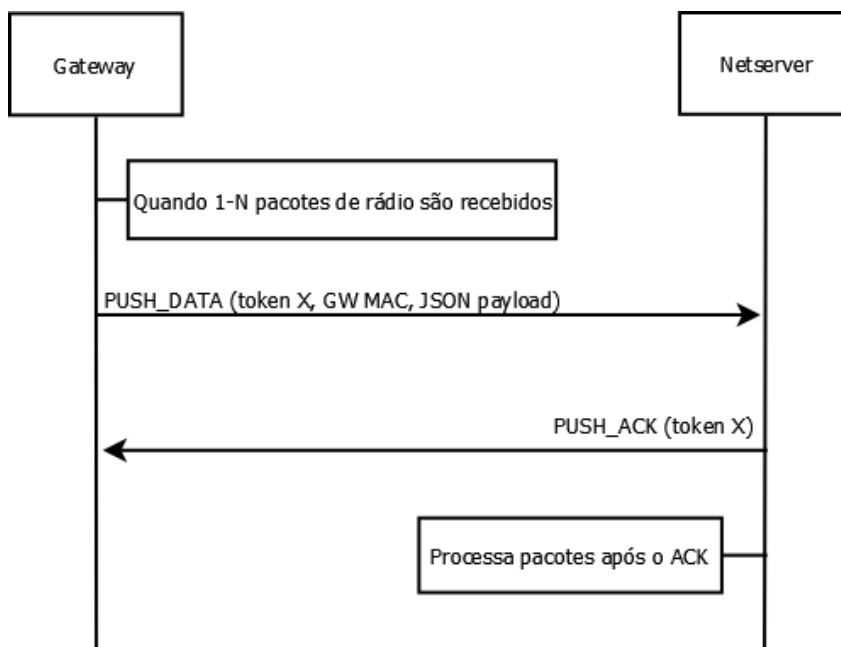
O UDP Packet Forwarder², especificado pela Semtech, é um protocolo de encaminhamento de mensagens entre *gateway* LoRa e *Netserver*, utilizando datagramas UDP

² Todo o conteúdo desta sessão é baseado na documentação oficial da Semtech, encontrada em https://github.com/Lora-net/packet_forwarder, e na experiência adquirida na utilização deste protocolo.

como transporte das mensagens. Não há autenticação entre *gateway* e *Netserver* e as mensagens de confirmação são utilizadas apenas para avaliação da qualidade da rede. Desta forma, esse protocolo não exige recursos avançados encontrados no TCP, por exemplo, diminuindo o tráfego entre *gateway* e *Netserver*. Por sua simplicidade é indicado que o protocolo seja utilizado somente para demonstrações ou em redes privadas e confiáveis. O protocolo é dividido em dois fluxos:

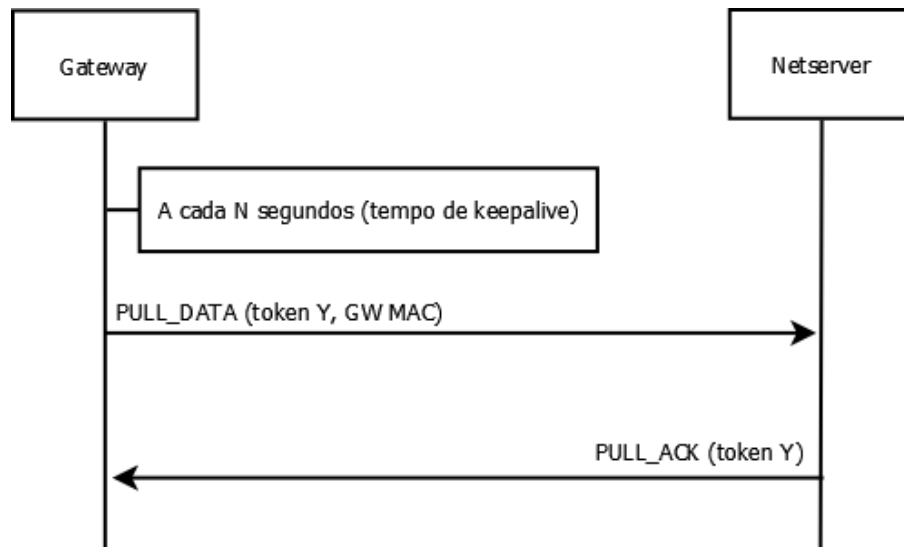
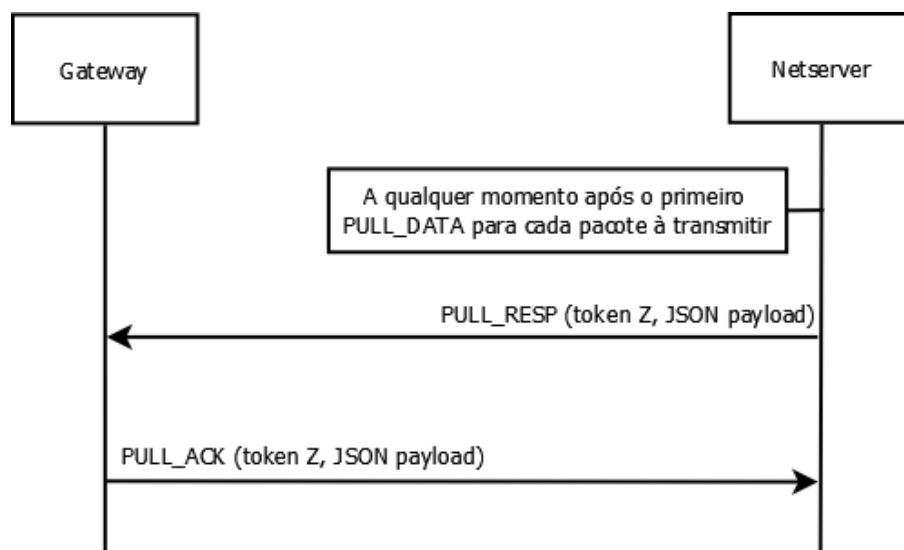
- **Upstream:** Pacotes de rádio recebidos pelo *gateway*, meta-dados adicionados pelo *gateway* e encaminhamento para o *Netserver*. Essas mensagens podem também conter o estado do *gateway*. O fluxo upstream é representado com um diagrama de sequência na [Figura 21](#).

Figura 21 – Diagrama de sequência de *upstream*.



- **Downstream:** Pacotes gerados pelo *Netserver*, com meta-dados inclusos, a serem transmitidos pelo *gateway* em um canal de rádio. Essas mensagens podem também conter informações de configuração para o *gateway*. O fluxo downstream é representado com dois diagramas de sequência na [Figura 22](#) e [Figura 23](#).

No upstream, o *gateway* encaminha ao *Netserver* os pacotes recebidos dos endpoints junto de meta-dados do pacote. Esta mensagem é chamada de PUSH_DATA, e sua estrutura é apresentada na [Tabela 6](#). Ao receber esta mensagem, o *Netserver* responde com uma mensagem de reconhecimento, a PUSH_ACK, cuja estrutura está na [Tabela 7](#). O objeto JSON raiz contido na PUSH_DATA pode conter um vetor chamado rxpk ou um vetor stat, podendo conter os dois tipos simultaneamente.

Figura 22 – Primeiro diagrama de sequência de *downstream*.Figura 23 – Segundo diagrama de sequência de *downstream*.

O vetor `rxpk` contém ao menos um novo objeto JSON com o pacote LoRa e seus metadados. Os campos destes objetos podem ser vistos na [Tabela 8](#). O objeto `stat` contém informações estatísticas do *gateway* e é descrito na [Tabela 9](#). Estes pacotes com objetos `stat` são enviados periodicamente, sendo que as suas informações são incrementais, ou seja, os contadores que geram estes dados são zerados no *gateway* após cada reinício, e as informações refletem as operações realizadas pelo *gateway* a partir da última vez em que ele foi iniciado.

No *downstream*, existem duas sequências de mensagens. A primeira acontece a cada tempo fixo de segundos (configurado no *gateway*), onde o *gateway* envia uma mensagem de keepalive para o *Netserver*, chamada `PULL_DATA`, respondido pelo *Netserver* por

Tabela 6 – Estrutura da mensagem PUSH_DATA.

| Bytes | Função |
|----------|--|
| 0 | Versão do protocolo (padrão: 2) |
| 1-2 | Token aleatório |
| 3 | Identificador do PUSH_DATA (0x00) |
| 4-11 | Identificador único do <i>gateway</i> (endereço MAC) |
| 12-final | Objeto JSON com informações do pacote |

Tabela 7 – Estrutura da mensagem PUSH_ACK.

| Bytes | Função |
|-------|---|
| 0 | Versão do protocolo (padrão: 2) |
| 1-2 | Mesmo token gerado para o PUSH_DATA equivalente |
| 3 | Identificador do PUSH_ACK (0x01) |

Tabela 8 – Atributos do objeto com pacote LoRa contidos no vetor rxpk.

| Atributo | Tipo | Descrição |
|----------|--------|---|
| time | string | Instante do envio da mensagem no formato ISO 8601 compacto |
| tmst | número | Timestamp do recebimento do pacote no <i>gateway</i> (32b unsigned) |
| freq | número | Frequência central do canal MHz (float, precisão em Hz) |
| chan | número | Canal IF usado pelo RX (unsigned integer) |
| rfch | número | “RF chain” usado pelo RX (unsigned integer) |
| stat | número | Status do CRC: 1 = OK, -1 = falha, 0 = sem CRC |
| modu | string | Identificador de modulação “LORA” ou “FSK” |
| dart | string | Identificador de taxa de dados LoRa (SF e BW) |
| datr | número | Taxa de dados FSK (unsigned, em bits por segundo) |
| codr | string | Identificador de taxa de código LoRa em fração |
| rsi | número | RSSI em dBm (signed integer, precisão de 1 dB) |
| lsnr | número | SNR LoRa em dB (signed float, precisão de 0,1 dB) |
| size | número | Tamanho do pacote RF em bytes (unsigned integer) |
| data | string | Payload PHY do pacote RF criptografado e codificado em Base64 |

uma mensagem chamada PULL_ACK. Esta troca inicial de mensagens se dá pelo fato de ser impossível ao servidor enviar pacotes para o *gateway* se o *gateway* estiver atrás de um NAT ou firewall. Quando o *gateway* inicia a negociação, a rota até o *Netserver* é aberta e permitirá um fluxo de pacotes em ambas as direções. Assim, o *gateway* deve periodicamente enviar pacotes PULL_DATA para garantir que a rota na rede permaneça aberta para o servidor utilizá-la a qualquer momento. As estruturas das mensagens PULL_DATA e PULL_ACK podem ser vistas na [Tabela 10](#) e [Tabela 11](#), respectivamente.

A segunda sequência de mensagens acontece após um PULL_ACK do *Netserver*. O *Netserver* envia pacotes de rádio e meta-dados associados que deverão ser emitidos pelo *gateway*. Esta mensagem chama-se PULL_RESP e sua estrutura é representada

Tabela 9 – Atributos do objeto stat.

| Atributo | Tipo | Descrição |
|----------|--------|---|
| time | string | Instante do envio da mensagem no formato ISO 8601 compacto |
| lati | número | Latitude do <i>gateway</i> em graus (float, N +) |
| long | número | Longitude do <i>gateway</i> em graus (float, E +) |
| alti | número | Altitude do <i>gateway</i> em metros (integer) |
| rxnb | número | Pacotes RF recebidos (unsigned integer) |
| rxok | número | Pacotes RF recebidos com CRC PHY válido (unsigned integer) |
| rxfw | número | Pacotes RF encaminhados (unsigned integer) |
| ackr | número | % de datagramas de <i>uplink</i> confirmados (unsigned integer) |
| dwnb | número | Datagramas de <i>downlink</i> recebidos (unsigned integer) |
| txnb | número | Pacotes emitidos (unsigned integer) |

Tabela 10 – Estrutura da mensagem PULL_DATA.

| Bytes | Função |
|-------|--|
| 0 | Versão do protocolo (padrão: 2) |
| 1-2 | Token aleatório |
| 3 | Identificados do PULL_DATA (0x02) |
| 4-11 | Identificador único do <i>gateway</i> (endereço MAC) |

Tabela 11 – Estrutura da mensagem PULL_ACK.

| Bytes | Função |
|-------|---|
| 0 | Versão do protocolo (padrão: 2) |
| 1-2 | Mesmo token gerado pelo PULL_DATA equivalente |
| 3 | Identificador do PULL_ACK (0x04) |

pela [Tabela 12](#). O objeto JSON raiz contido na PULL_RESP deve conter um objeto chamado txpk, que contém o pacote de rádio e seus meta-dados associados. Os campos desse objeto podem ser vistos na [Tabela 13](#). Em contrapartida, o *gateway* responde com uma mensagem chamada TX_ACK., cuja estrutura pode ser vista na [Tabela 14](#). Esta mensagem é utilizada para o *gateway* informar o *Netserver* se a requisição de *downlink* foi aceita ou rejeitada pelo *gateway*. O datagrama pode opcionalmente conter um objeto JSON chamado txpk_ack, com mais detalhes da requisição em um campo chamado error. A estrutura do objeto txpk_ack e a lista de possíveis valores do campo error são apresentadas na [Tabela 15](#) e [Tabela 16](#), respectivamente.

3.2.3 LoRaServer

O LoRa Server ([BROCAAR, 2017c](#)) é um *Netserver* LoRaWAN de código aberto escrito principalmente na linguagem Go. Faz parte de um projeto maior que é constituído de um broker do protocolo packet forwarder para MQTT (LoRa Gateway Bridge), do

Tabela 12 – Estrutura da mensagem PULL_RESP.

| Bytes | Função |
|---------|---------------------------------------|
| 0 | Versão do protocolo (padrão: 2) |
| 1-2 | Token aleatório |
| 3 | Identificador do PULL_RESP (0x03) |
| 4-final | Objeto JSON com informações do pacote |

Tabela 13 – Atributos do objeto txpk.

| Nome | Tipo | Descrição |
|------|----------|---|
| imme | booleano | Envia o pacote imediatamente (ignoraré tmst e time) |
| tmst | número | Envia o pacote em um certo valor de timestamp (ignoraré time) |
| time | string | Envia o pacote em um certo tempo (necessita sincronização) |
| freq | número | Frequência central TX em MHz (unsigned float, precisão em Hz) |
| rfch | número | "RF chain"do <i>gateway</i> usada para TX (unsigned integer) |
| powe | número | Potência de saída TX em dBm (unsigned integer) |
| modu | string | Identificador da modulação "LORA"ou "FSK" |
| datr | string | Identificador da taxa de dados LoRa (SF e BW) |
| datr | número | Taxa de dados FSK (unsigned, em bits por segundo) |
| codr | string | Identificador de taxa de código LoRa em fração |
| fdev | número | Desvio de frequência FSK (unsigned integer, em Hz) |
| ipol | booleano | Inversão de polarização da modulação LoRa |
| prea | número | Tamanho do preâmbulo RF (unsigned integer) |
| size | número | Tamanho do payload do pacote RF em bytes (unsigned integer) |
| data | string | Payload PHY do pacote RF criptografado e codificado em Base64 |
| ncrc | booleano | Se verdadeiro, desabilita o CRC da camada física (opcional) |

Tabela 14 – Estrutura da mensagem TX_ACK.

| Bytes | Função |
|----------|--|
| 0 | Versão do protocolo (padrão: 2) |
| 1-2 | Mesmo token gerado pelo PULL_RESP equivalente |
| 3 | Identificador do TX_ACK (0x05) |
| 4-11 | Identificador único do <i>gateway</i> (endereço MAC) |
| 12-final | Objeto JSON com informações do pacote (opcional) |

LoRa Server propriamente dito é um servidor de aplicação compatível. A comunicação entre o LoRa Server e o servidor de aplicação deve ser feita via RPC (Remote Procedure Call) (THURLOW, 2009), utilizando a implementação gRPC. É responsável por tratar as mensagens de *uplink* recebidas pelos gateways e escalonamento das transmissões *downlink*. Atualmente é um projeto em desenvolvimento, logo não possui suporte a todos os mecanismos descritos pela especificação do LoRaWAN. O LoRa Server possui suporte integral para dispositivos classe A. Mensagens duplicadas são tratadas e encaminhadas para o servidor de aplicação. Ao ocorrer uma janela de recepção, o servidor elege uma

Tabela 15 – Atributos do objeto txpk_ack.

| Nome | Tipo | Descrição |
|-------|--------|---|
| error | string | Indicação sobre o sucesso ou tipo de falha na requisição de <i>downlink</i> |

Tabela 16 – Possíveis valores para o campo error.

| Valor | Definição |
|------------------|---|
| Nenhum | Pacote foi escalonado para o <i>downlink</i> |
| TOO_LATE | Rejeitado: tarde demais para programar o envio |
| TOO_EARLY | Rejeitado: cedo demais para programar o envio |
| COLLISION_PACKET | Rejeitado: instante solicitado está ocupado por um pacote |
| COLLISION_BEACON | Rejeitado: instante solicitado está ocupado por um beacon |
| TX_FREQ | Rejeitado: frequência requisitada não é suportada |
| TX_POWER | Rejeitado: potência requisitada não é suportada |
| GPS_UNLOCKED | Rejeitado: GPS não está disponível para sincronizar relógio |

mensagem de *downlink* do servidor de aplicação, avaliando se a mensagem se adequa em alguma taxa de transmissão disponível baseando-se no tamanho do payload da mensagem. Os mecanismos disponíveis no LoRa Server são:

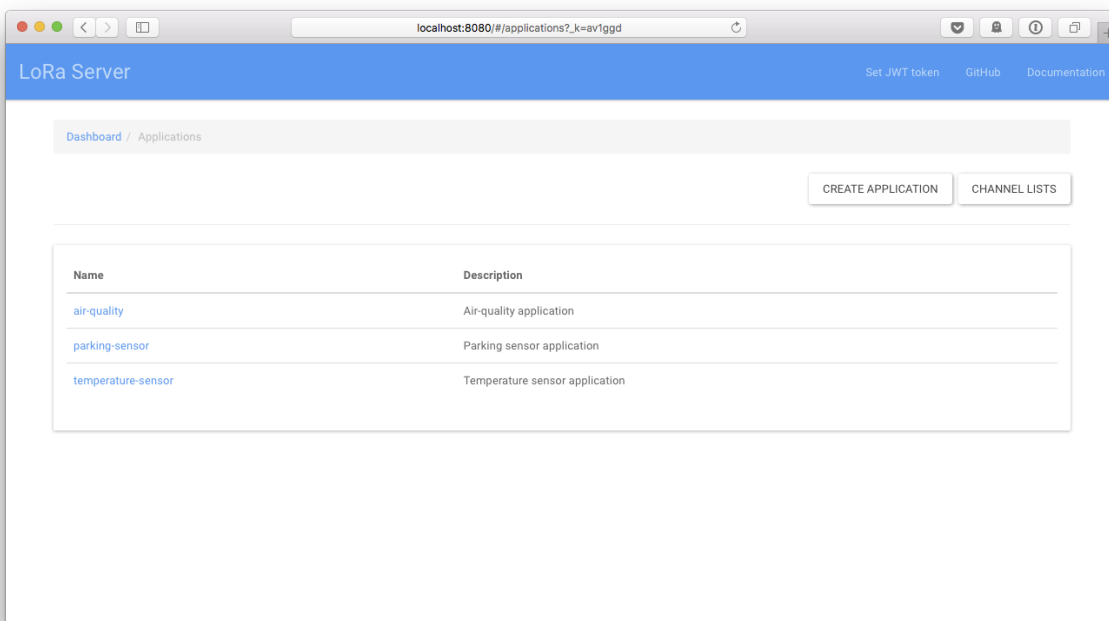
- **Mensagens com confirmação:** Ambas mensagens *downlink* e *uplink* são tratadas pelo LoRa Server. No caso de mensagens de *downlink* com confirmação, o LoRa Server mantém a mensagem enfileirada até receber a confirmação do *endpoint*.
- **Ativação do *endpoint*:** Ambos métodos de ativação (Ativação personalizada e Ativação pelo ar) são suportados. No caso de ABP, o servidor de aplicação entrega ao LoRa Server as informações da sessão. No caso de OTAA, o LoRa Server chama o servidor de aplicação com a mensagem join-request e, no caso de resposta positiva, transmite a mensagem join-accept ao *endpoint*.
- **ADR (Taxa de transmissão adaptativa):** O LoRa server atualmente possui suporte para ADR somente na banda EU 863-870. Por isso e por ter sido implementado após o início dos testes, essa função não foi testada e utilizada no projeto. Para ativar o ADR, devem estar configurados no *endpoint* o intervalo ADR, número de frames para recalculer a taxa de transmissão, e a potência de transmissão do *endpoint*. Para iniciar o ADR, as transmissões *uplink* devem conter a flag ADR ativa.
- **Janelas de recepção:** Em ambos modos de ativação é possível configurar qual janela de transmissão será utilizada nas transmissões *downlink*. Isso inclui parâmetros como taxa de transmissão (para a janela 2) e o atraso.

- **Bandas ISM³**: Ao iniciar o LoRa Server, deve estar configurado qual banda ISM será utilizada. Atualmente o Loraserver suporta faixas de países situados na Europa, sudeste asiático, Estados Unidos, Coreia do Sul, Rússia e China, seguindo a especificação da LoRaWAN.

3.2.4 Servidor de Aplicação LoRa

O LoRa App Server ([BROCAAR, 2017a](#)) é um servidor de aplicação LoRa, compatível com o LoRa Server, de código aberto e escrito principalmente nas linguagens Go e Javascript. É responsável por cuidar de todos os registros dos endpoints cadastrados na rede e suas sessões, tratamento de mensagens das aplicações recebidas e enfileiramento de mensagens de *downlink*. Possui uma interface web, mostrada nas figuras [Figura 24](#), [Figura 25](#) e [Figura 26](#), para administrar todos os endpoints e suas configurações, podendo utilizar JWT (JSON Web Tokens) ([JONES; BRADLEY; SAKIMURA, 2015](#)) como método de autenticação. Por fim, ainda possui uma API gRPC e RESTful ([FIELDING, 2000](#)) para integração com aplicações.

Figura 24 – Interface web do LoRa App Server com as aplicações cadastradas.

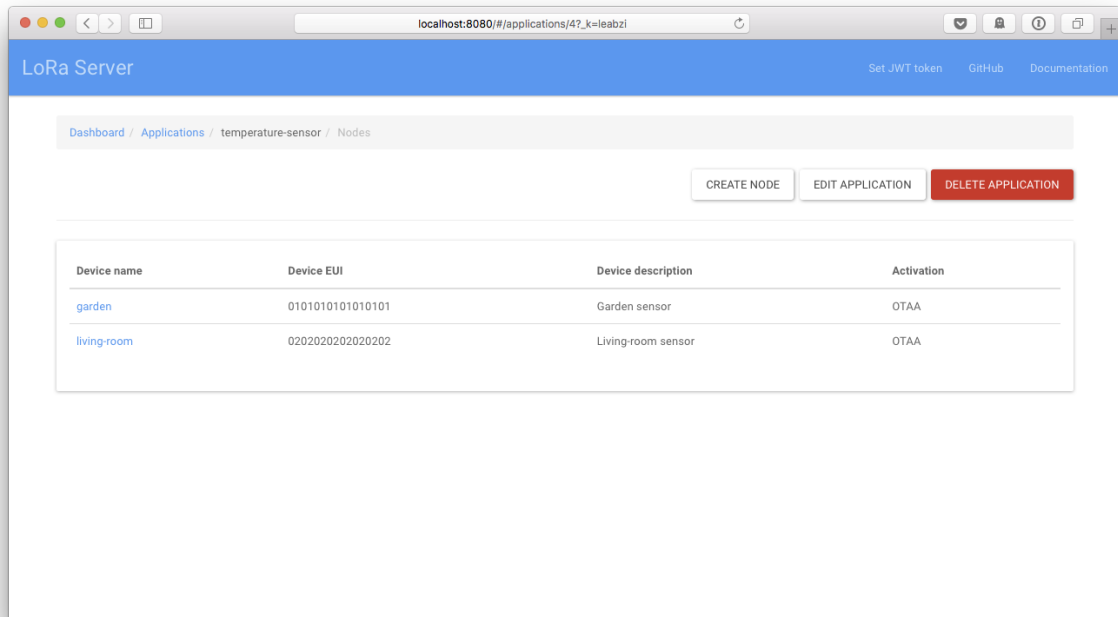


Fonte: ([BROCAAR, 2017a](#))

Todas as mensagens de *uplink* são publicadas em tópicos MQTT depois de serem descritografadas. As mensagens de *downlink* podem ser enfileiradas publicando em um

³ Bandas reservadas internacionalmente para o desenvolvimento industrial, científico e médico.

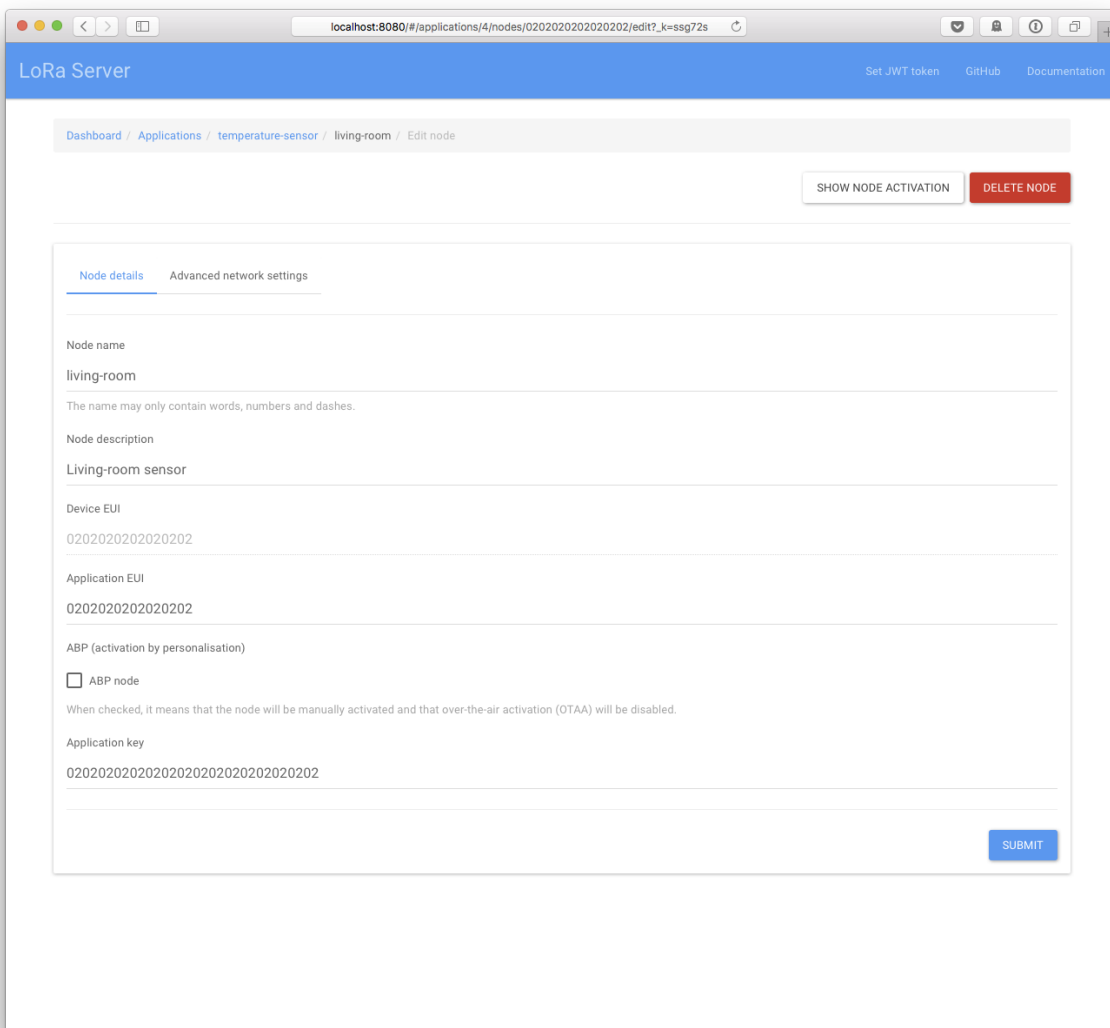
Figura 25 – Interface web do LoRa App Server com nós cadastrados em uma aplicação.



Fonte: (BROCAAR, 2017a)

tópico MQTT respectivo ou utilizando a API. Além dessas mensagens, o LoRa App Server ainda pode publicar eventos em tópicos MQTT, como uma ativação de um *endpoint*, uma mensagem confirmada por um *endpoint* ou um erro, como, por exemplo, um payload de *downlink* que excedeu o tamanho máximo permitido.

Figura 26 – Interface web do LoRa App Server com os detalhes de um nó cadastrado.



The screenshot displays the LoRa Server web interface in a browser window. The address bar shows the URL: localhost:8080/#/applications/4/nodes/0202020202020202/edit?k=ssg72s. The page title is "LoRa Server" and the breadcrumb navigation is "Dashboard / Applications / temperature-sensor / living-room / Edit node".

At the top right, there are links for "Set JWT token", "GitHub", and "Documentation". Below the breadcrumb, there are two buttons: "SHOW NODE ACTIVATION" and "DELETE NODE".

The main content area is titled "Node details" and "Advanced network settings". The "Node details" section includes the following fields:

- Node name:** living-room
- Node description:** Living-room sensor
- Device EUI:** 0202020202020202
- Application EUI:** 0202020202020202
- ABP (activation by personalisation):** ABP node
- Application key:** 02020202020202020202020202020202

A "SUBMIT" button is located at the bottom right of the form.

Fonte: (BROCAAR, 2017a)

4 Sistema gerenciador de aplicações LoRaWAN configuráveis

Neste capítulo são tratados todos os tópicos do desenvolvimento do sistema gerenciador de aplicações LoRaWAN configuráveis. Este sistema tem o propósito de utilizar todos os recursos (portas analógicas, digitais, serial, entre outras) da plataforma utilizada para o *endpoint* (ArduinoUno, Raspberry Pi ou uma placa personalizada), podendo ser configurada e atualizada por mensagens de controle enviadas pelo *Netserver*. Como proposta, esta aplicação deve ser capaz de:

- Receber mensagens *downlink* com as configurações de seu funcionamento, interpretá-las e configurar o software do *endpoint* de acordo com o conteúdo recebido;
- Trabalhar com múltiplas aplicações simultâneas, ou seja, lendo valores de diferentes entradas com diferentes portas, períodos, etc;
- Ser facilmente adaptável para plataformas diferentes;
- Facilitar a criação de aplicações LoRaWAN, expondo os seus principais parâmetros de configuração.

4.1 Protocolo de configuração

Para ser possível a configuração remota das aplicações no *endpoint*, foi necessário criar um protocolo de configuração que carregue as informações básicas necessárias para uma aplicação LoRa. Este protocolo foi projetado pensando nos recursos que uma placa ArduinoUno possui, entretanto, diversos campos foram reservados para possíveis adaptações. Em sua primeira versão, o protocolo de configuração possui apenas uma mensagem, chamada de mensagem de controle. Essa mensagem de controle é um pacote *downlink* LoRaWAN que utiliza a porta 1 como padrão e seu conteúdo no payload deste pacote. A estrutura da mensagem de controle pode ser vista na [Figura 27](#). Seus campos são:

Figura 27 – Estrutura da mensagem de controle do protocolo de configuração.

| | | | |
|-----------------|-------|--------------|--------------|
| Size (bits) | 8 | 40..320 | 48..328 |
| Control Message | nApps | Applications | Total Length |

| | | | | | | | | |
|-------------|-------|------|---------|----------|-------------|-----|--------|--------------|
| Size (bits) | 8 | 3 | 4 | 4 | 3 | 1 | 17 | 40 |
| Application | FPort | Type | Channel | Ch_Param | Acquisition | Ack | Period | Total Length |

- **nApps:** Número de aplicações a serem configuradas. Devido ao tamanho máximo de uma mensagem de *downlink* LoRaWAN, este valor não deve ultrapassar o valor 8 e deve sempre ser maior que 0.
- **Applications:** Parâmetros utilizados para configurar as aplicações:
 - **FPort:** Define qual porta LoRaWAN será utilizada na transmissão. Este valor não pode ser 0 (porta para comandos MAC) e 1 (porta para mensagens de controle deste protocolo);
 - **Type:** Define o tipo do valor que a aplicação irá transportar, listados na [Tabela 17](#);
 - **Channel:** Define o canal de informação no qual o *endpoint* retirará os dados. As opções válidas podem ser vistas na [Tabela 18](#);
 - **CH_Param:** Parâmetro de configuração do canal definido anteriormente.
 - **Acquisition:** Define o método de aquisição dos dados. Opções válidas podem ser vistas na [Tabela 23](#);
 - **Ack:** Define se a aplicação deve enviar mensagens com confirmação ou não. Valor 0 para mensagens não confirmadas e 1 para mensagens confirmadas;
 - **Period:** Define a periodicidade das mensagens, em segundos. Este valor está sujeito a desvios de poucos segundos devido ao tempo de processamento.

Tabela 17 – Possíveis valores do campo Type.

| Type | Valor | Nota |
|------|---------|-----------------------------|
| 0 | int | Inteiro de 16 bits |
| 1 | string | Sequência de caracteres |
| 2 | boolean | Verdadeiro (1) ou Falso (0) |
| 3 | float | Ponto flutuante de 32 bits |
| 4-7 | RFU | Reservado para uso futuro |

Os dispositivos inseridos nas tabelas de configurações foram escolhidos com base na disponibilidade de sensores e equipamentos no campus para testes.

4.2 Aplicação no *endpoint*

O sistema desenvolvido para os endpoints foi codificado em C++ e utiliza alguns recursos disponibilizados pela biblioteca do Arduino, como a leitura de suas portas, por exemplo. Toda a parte que se refere ao funcionamento do protocolo LoRaWAN é feita pela biblioteca LMIC (LoRa Mac-In-C).

Tabela 18 – Possíveis valores para o campo Channel e Ch_Param.

| Channel | Valor | Ch_Param |
|---------|------------------|--|
| 0 | Digital | Número do pino |
| 1 | Analógico | Canal analógico |
| 2 | Interrupção | Número da interrupção |
| 3 | UART | Baudrate (veja Tabela 19) |
| 4 | SPI | Configuração SPI (veja Tabela 20) |
| 5 | I2C | Configuração I2C (veja Tabela 21) |
| 6 | GDI ¹ | Configuração GDI (veja Tabela 22) |
| 7-F | RFU | Reservado para uso futuro |

Tabela 19 – Possíveis configurações para a UART.

| Ch_Param | Configuração |
|----------|--------------|
| 0 | 300 8N1 |
| 1 | 600 8N1 |
| 2 | 1200 8N1 |
| 3 | 2400 8N1 |
| 4 | 4800 8N1 |
| 5 | 9600 8N1 |
| 6 | 14400 8N1 |
| 7 | 19200 8N1 |
| 8 | 28800 8N1 |
| 9 | 38400 8N1 |
| A | 57600 8N1 |
| B | 115200 8N1 |
| C-F | RFU |

Tabela 20 – Possíveis dispositivos SPI.

| Ch_Param | Dispositivo | Tipo |
|----------|-------------|---------------------------|
| 0 | SCP1000 | Sensor de pressão |
| 1 | BME280 | Multisensor Adafruit |
| 2 | DS3234 | Relógio de tempo real |
| 3 | DS1306 | Relógio de tempo real |
| 4-F | RFU | Reservado para uso futuro |

4.2.1 LMIC: LoRaMAC-In-C

A LMIC é uma biblioteca inicialmente criada pela IBM, posteriormente abandonada pela empresa e disponibilizada em um repositório público. Atualmente ela é mantida pela comunidade de software livre. Esta biblioteca possui suporte para OTAA (ativação pelo ar) e ABP (ativação personalizada), bandas europeia (868 MHz) e americana (915 MHz). Trata os eventos de recepção e transmissão com uma máquina de estados e utiliza de uma fila temporizada interna para escalonar os envios.

Tabela 21 – Possíveis dispositivos I2C

| Ch_Param | Dispositivo | Tipo |
|----------|-------------|--|
| 0 | MPU-6050 | Acelerômetro e giroscópio |
| 1 | GY-302 | Sensor de luminescência baseado no BH1750FVI |
| 2 | DS1307 | Relógio de tempo real |
| 3-F | RFU | Reservado para uso futuro |

Tabela 22 – Possíveis dispositivos de interface genérica (GDI).

| Ch_Param | Dispositivo | Tipo |
|----------|-------------|---|
| 0 | DHT-11 | Sensor de temperatura e humidade |
| 1 | DHT22 | Sensor de temperatura e humidade (AM2303) |
| 2 | DS18B20 | Sensor de temperatura digital (1 fio) |
| 3-F | RFU | Reservado para uso futuro |

Tabela 23 – Possíveis métodos de aquisição de dados.

| Acquisition | Método | Nota |
|-------------|--------|-------------------------------|
| 0 | Único | Única aquisição |
| 1 | Média | Valor médio de 'x' aquisições |
| 2 | RMS | Valor RMS de 'x' aquisições |
| 3-F | RFU | Reservado para uso futuro |

O tratamento dos eventos gerados pela máquina de estados da LMIC não é implementado pela biblioteca, mas sim, por quem estiver utilizando a máquina. Os eventos gerados por ela são apresentados na [Tabela 24](#).

No sistema no *endpoint*, a fila de envios foi o único componente utilizado da biblioteca LMIC. Toda a parte de ativação e seleção de banda deve ser feita em uma outra etapa do *software* que estiver utilizando o sistema. Para agendar eventos, a LMIC recebe uma função callback de envio (contendo uma transmissão de rádio LoRa) juntamente com um valor de tempo. Esta callback é enfileirada ordenadamente segundo o valor de tempo e um timer é configurado. Assim que um tempo se esgotar, o sistema liga uma interrupção, desenfileira a função callback e a executa.

4.2.2 Estrutura da aplicação

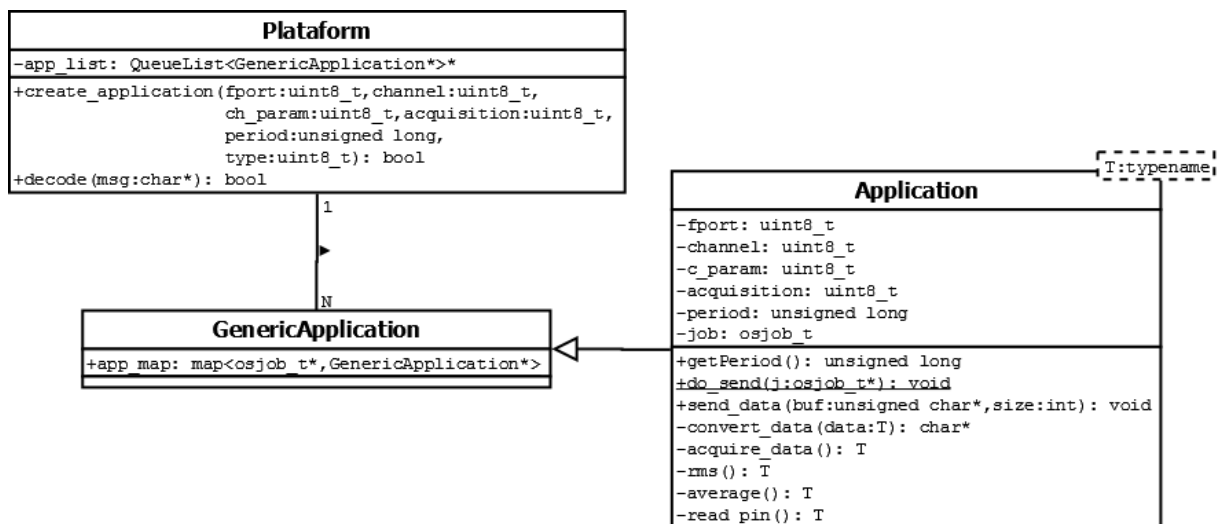
A estrutura da aplicação pode ser vista na [Figura 28](#). Ela consiste em duas classes principais, `Application` e `Plataform`.

A classe `Application` é a representação de uma aplicação contendo atributos como canal de dados da plataforma a ser utilizado, periodicidade de aquisição, porta utilizada, mensagens com confirmação, etc. Esta classe fica invisível ao usuário do sistema, sendo utilizada somente pela classe `Plataform`. Ao ser criado um objeto desta classe, sua

Tabela 24 – Eventos gerados pela biblioteca LMIC.

| Evento | Descrição |
|-------------------|--|
| EV_JOINING | Nó começou o processo de entrada na rede |
| EV_JOINED | Nó entrou na rede com sucesso |
| EV_JOIN_FAILED | Nó não pode entrar na rede (após segunda tentativa) |
| EV_REJOIN_FAILED | Nó não pode entrar em uma nova rede, mas continua conectado na rede anterior |
| EV_TXCOMPLETE | Dados foram transmitidos e possíveis dados de <i>downlink</i> foram recebidos |
| EV_RXCOMPLETE | Dados de <i>downlink</i> foram recebidos |
| EV_SCAN_TIMEOUT | Nenhum <i>beacon</i> foi encontrado no intervalo de <i>beacon</i> |
| EV_BEACON_FOUND | Primeiro <i>beacon</i> foi encontrado no intervalo de <i>beacon</i> |
| EV_BEACON_TRACKED | Próximo <i>beacon</i> foi encontrado dentro do tempo esperado |
| EV_BEACON_MISSED | Nenhum <i>beacon</i> foi encontrado no tempo esperado |
| EV_LOST_TSYNC | <i>Beacon</i> foi perdido repetidas vezes e sincronização foi perdida |
| EV_RESET | Sessão foi reiniciada |
| EV_LINK_DEAD | Nenhuma confirmação foi recebida do <i>Netserver</i> por um longo período de tempo |

Figura 28 – Diagrama de classe do sistema de gerenciamento de aplicações LoRaWAN configuráveis.

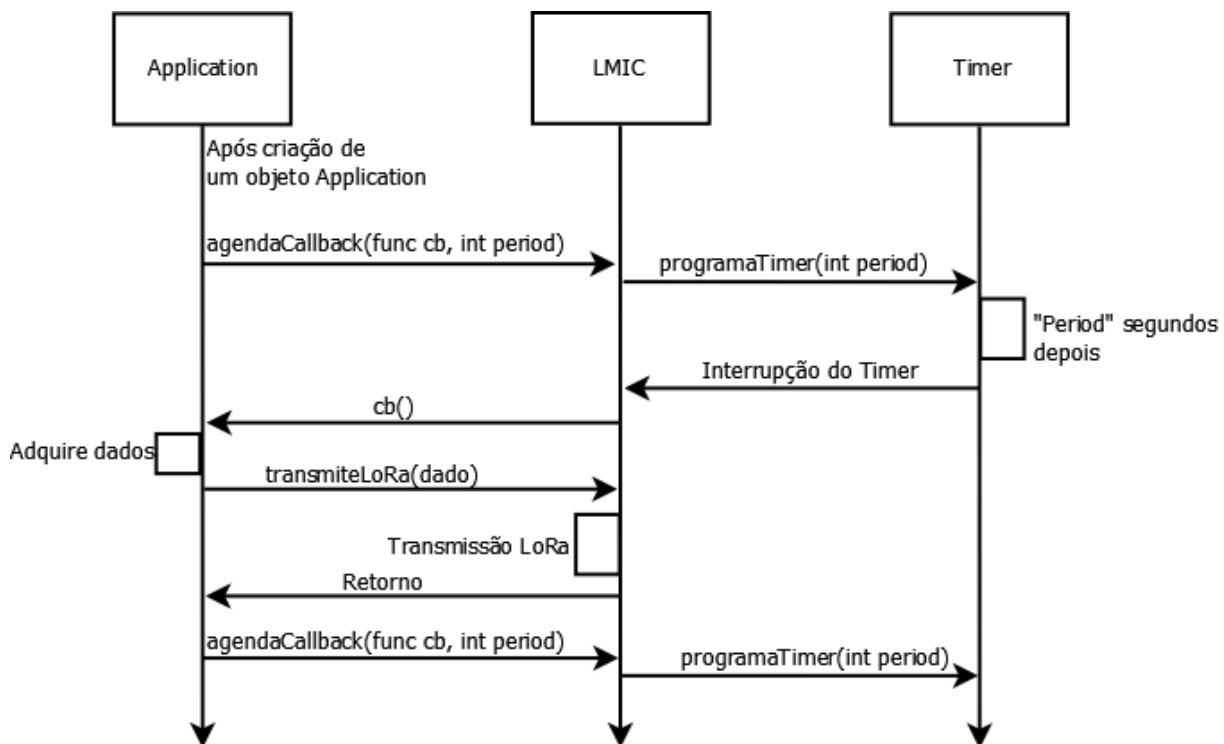


função callback de envio é enfileirada. Quando a callback é executada, após realizar a transmissão, ela se enfileira novamente, gerando um ciclo periódico de envios, segundo a Figura 29. Como a função de envio deve ser estática, foi necessário criar um mapeamento dos objetos com o argumento da função de envio. Isto foi realizado para que a função de envio saiba de qual instância de Application ela foi invocada e possa acessar os atributos do objeto Application correto. Este mapeamento foi feito utilizando uma biblioteca de terceiros (MANIACBUG, 2013). Para poder realizar diferentes tipos de aquisição de dados,

são utilizadas classes parametrizadas (*templates*), onde a função de aquisição de dados e derivadas se adequam a cada tipo de aplicação.

A **Plataform** é a classe responsável por interpretar as mensagens de controle recebidas e criar as aplicações correspondentes. Possui apenas um único método como interface para sua utilização, chamado `decode`, que recebe uma *string* seguindo a especificação do protocolo de configuração. Após invocar o método, o objeto trata de deletar possíveis aplicações existentes, limpar toda a fila de envios da biblioteca LMIC, interpretar a mensagem de controle e criar e armazenar em uma lista as aplicações desejadas contidas na mensagem. A lista encadeada utilizada vem de uma biblioteca de terceiros ([CHATZIKYRIAKIDIS, 2010](#)).

Figura 29 – Diagrama de sequência do funcionamento do ciclo de transmissão do sistema.



Uma das partes mais importantes do código encontra-se na classe **Application**, que é o uso de *templates* para generalizar os tipos de aplicações. Aplicações que extraem dados do tipo inteiro possuem formas de aquisição desses dados diferentes de aplicações que extraem *strings*, por exemplo. Dessa forma, métodos diferentes devem ser implementados para cada tipo de aplicação. Entretanto, uma aplicação extrai inteiros não necessita ter os métodos para extrair uma string. Desta forma, o uso de *templates* permite que somente os métodos necessários para um determinado tipo de aplicação sejam alocados, reduzindo o uso de memória do dispositivo. Um exemplo da codificação dos *templates* no *software* pode visto na [Figura 30](#) e [Figura 31](#)

Note que a única parte do código da aplicação dependente da plataforma Arduino

Figura 30 – Código-fonte em C++ da declaração dos métodos parametrizados com *templates*.

```
template<typename T>
class Application : public GenericApplication {

private:
...
    T rms();
    T average();
    T read_pin();
    T acquire_data();
...
}
```

é a utilização dos métodos de aquisição de dados das portas digitais, analógicas, seriais da biblioteca do Arduino. Assim, essa é a única parte necessária adaptar caso seja desejado utilizar o código em outras plataformas.

4.3 Testes

Os testes a seguir foram realizados visando validar as propostas do sistema de gerenciamento de aplicações LoRaWAN configuráveis desenvolvida. Os testes foram iniciados após ser obtida uma versão minimamente estável da aplicação e da rede LoRaWAN implantada.

Para a utilização do sistema, foram criados programas com a IDE do Arduino contendo as configurações básicas necessárias para o funcionamento da biblioteca LMIC. Em todos os testes foi utilizado o método e ativação personalizado (ABP). As chaves de sessão utilizadas foram as chaves padrão disponibilizadas pela Semtech para testes, também utilizadas nos exemplos que a The Things Network disponibiliza. O endereço do dispositivo utilizado também foi o encontrado nos exemplos da The Things Network. Como os dois módulos adquiridos possuem algumas diferenças, a configuração da pinagem para o uso da biblioteca é levemente diferente, uma definição foi criada para facilitar a troca dos módulos nos testes. Todas essas configurações iniciais podem ser vistas na [Figura 32](#).

A máquina de estados da biblioteca LMIC aparece no código como a função `onEvent`, que trata os eventos como recepção, transmissão, *joining*, entre outros. Após uma transmissão, o sistema verifica se existe algum dado no buffer de recepção (lembrando que um dado só pode ser recebido após uma transmissão, utilizando as janelas de recepção). Caso exista, ele verifica se a porta 1 foi utilizada para encaminhar a mensagem recebida para a classe `Plataform` interpretá-la. Este processo é mostrado na [Figura 33](#).

Os primeiros testes verificaram a funcionalidade da classe `Application` sozinha.

Figura 31 – Código-fonte em C++ com a implementações de métodos parametrizados com *templates*.

```

//Aplicacoes de int
template<>
int Application<int>::read_pin(){
    switch (this->channel){
        case 1: //analog
            return analogRead(this->ch_param);
        default:
            return 0;
            break;
    }
}

//Aplicacoes de char*
template<>
char* Application<char*>::read_pin(){
    switch(this->channel){
        case 3:
            if(this->ch_param>SERIAL_BR_SIZE-1) return NULL;
            Serial.end();
            Serial.begin(serial_br[this->ch_param]);
            if(Serial.available()>0)
                return (char*)(Serial.readString().c_str());
            break;
        default:
            return NULL;
    }
}

//Aplicacoes de bool
template<>
bool Application<bool>::read_pin(){
    switch (this->channel){
        case 0: // digital
            if(digitalRead(this->ch_param)==HIGH){return true;}
            else {return false;}
            break;
        default:
            return 0;
            break;
    }
}

```

Figura 32 – Código-fonte em C++ para configuração da LMIC no Arduino.

```

static const PROGMEM u1_t NWKSKEY[16]
    = { 0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6,
        0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C };

static const u1_t PROGMEM APPSKEY[16]
    = { 0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6,
        0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C };

static const u4_t DEVADDR = 0x03FF0001 ;

#if LoRaTransceiver == HopeRF_Transceiver
const lmic_pinmap lmic_pins = {
    .nss = 10,
    .rxtx = LMIC_UNUSED_PIN,
    .rst = 9,
    .dio = {2, 3, LMIC_UNUSED_PIN},
};
#elif LoRaTransceiver == Modtronix_Transceiver
const lmic_pinmap lmic_pins = {
    .nss = 10,
    .rxtx = LMIC_UNUSED_PIN,
    .rst = 9,
    .dio = {2, 3, 4},
};

```

Figura 33 – Código-fonte em C++ do tratamento de envio-recepção da máquina de estados da LMIC.

```

case EV_TXCOMPLETE:
    if (LMIC.dataLen) {
        if (LMIC.frame[LMIC.dataBeg-1] == 1) { //if fport == 1
            plat->decode((char*)&LMIC.frame[LMIC.dataBeg]);
        }
    }
break;

```

Dentro dos arquivos da IDE do Arduino, objetos dessa classe foram criados, sendo parametrizados manualmente. Após validado o funcionamento isolado, foram criados objetos simultâneos, verificado e validado o funcionamento de poucas aplicações (até 5) ao mesmo tempo.

Após isso, foi testada a implementação do decodificador do protocolo de configuração, juntamente com a criação das aplicações a partir do recebimento das mensagens de controle. Para isso, um pequeno circuito com um termistor foi adicionado ao *endpoint* afim de criar uma aplicação para monitorar temperatura. Este dispositivo foi deixado ligado por alguns dias monitorando a temperatura de um ambiente. Durante esse tempo, diversas mensagens de configuração foram enviadas, alterando a periodicidade do envio

das mensagens, porta utilizada e aumentando o número de aplicações utilizando a mesma informação gerada pelo termistor. Após algumas verificações do funcionamento e código, percebeu-se um problema na utilização das interrupções por causa da biblioteca LMIC. Este problema foi corrigido alterando as funções de ligar e desligar interrupções pelas utilizadas pela biblioteca LMIC.

O próximo teste teve a intenção de forçar o maior número de aplicações simultâneas possíveis utilizando o protocolo de configuração. Os limitantes quanto ao número de aplicações simultâneas são a quantidade de memória que a plataforma dispõe, devido a constante alocação dinâmica do sistema, e ao número de processos simultâneos, podendo descartar pacotes antes de serem transmitidos. Atualmente, como o protocolo não prevê atualização das aplicações mas um reinício, apagando todas as aplicações existentes previamente, o máximo de aplicações simultâneas é oito, devido ao tamanho máximo de uma mensagem *downlink*. Neste teste, o servidor enviou uma mensagem configurando oito aplicações com período de 16 segundos, cada uma utilizando uma porta diferente com a intenção de identificá-las. Por serem configuradas quase que simultaneamente pelo sistema e por terem os mesmos períodos de envio, os envios foram escalonadas para instantes de tempo semelhantes. Assim, houve um atraso no envio de algumas mensagens, entretanto, não houveram perdas devido ao software. Após isso, foi realizado o mesmo teste com as aplicações com período de 255 segundos e o atraso não foi detectado.

Com a tentativa de forçar o sistema, foram configuradas 222 aplicações simultâneas (fPort, seguindo a regra do protocolo de configuração, fica entre 2 e 223) na inicialização do programa, sem utilizar uma mensagem de controle. Ao fazer isso, a plataforma Arduino apresentou diversos reinícios contínuos, característicos de estouro de memória. Ao diminuir o número de aplicação, o mesmo comportamento continuou a apresentar-se, até chegar-se ao número de 52 aplicações simultâneas, sem apresentar nenhum descarte de envio e nenhum estouro de memória.

Conclusão

Neste trabalho foi implementada uma rede LoRaWAN e desenvolvido um sistema gerenciador de aplicações LoRaWAN configuráveis. Para isso, foi realizado um estudo do funcionamento do protocolo LoRaWAN, servidores de rede de código aberto e bibliotecas que implementam o protocolo nos endpoints.

A implantação da rede LoRaWAN foi realizada com êxito. Com ela foram realizados todos os testes que envolviam a comunicação fim-a-fim entre os dispositivos e o servidor de aplicação. O sistema gerenciador de aplicações configuráveis foi desenvolvido e apresentou resultados satisfatórios, cumprindo os requisitos propostos. Ele foi capaz de receber mensagens *downlink* de configuração através do protocolo de configuração e interpretá-las corretamente. O sistema também suportou a existência de múltiplas configurações simultâneas, ultrapassando o limite imposto inicialmente pelo protocolo de configuração de 8 aplicações, para até 52 (limite de memória da plataforma utilizada). Por fim, o *software* desenvolvido possui um código que pode ser facilmente portado para outras plataformas. Uma limitação do sistema é a necessidade de, caso esteja sendo utilizada ativação personalizada e nós classe A, o nó ser iniciado com alguma aplicação enviando mensagens com o intuito de abrir janelas de recepção para que o servidor envie uma mensagem de controle.

Devido aos resultados obtidos, as seguintes propostas são feitas para novos testes na rede LoRaWAN e melhorias no sistema gerenciador de aplicações LoRaWAN configuráveis:

- Estender o protocolo de configuração para adicionar novas aplicações em um dispositivo sem remover todas as aplicações existentes;
- Criar alguma forma de abrir janelas de recepção para nós classe A utilizando ABP, possibilitando a recepção de uma mensagem de controle;
- Continuar atualizando o *Netserver* para as versões mais recentes e utilizar possíveis novas funcionalidades;
- Criar um código compatível com o sistema que trate de toda a etapa da ativação dos dispositivos na rede, abstraindo ainda mais a biblioteca LMIC;

Referências

- BRAY, T. *The JavaScript Object Notation (JSON) Data Interchange Format*. [S.l.], 2014. <<http://www.rfc-editor.org/rfc/rfc7159.txt>>. Disponível em: <<http://www.rfc-editor.org/rfc/rfc7159.txt>>. 40
- BROCAAR, O. *LoRa App Server*. [S.l.]: GitHub, 2017. <<https://github.com/brocaar/lora-app-server>>. 48, 49, 50
- BROCAAR, O. *LoRa Gateway Bridge*. [S.l.]: GitHub, 2017. <<https://github.com/brocaar/lora-gateway-bridge>>. 39
- BROCAAR, O. *LoRa Server*. [S.l.]: GitHub, 2017. <<https://github.com/brocaar/loraserver>>. 45
- CENTENARO, M. et al. Long-range communications in unlicensed bands: the rising stars in the iot and smart city scenarios. *CoRR*, abs/1510.00620, 2015. Disponível em: <<http://arxiv.org/abs/1510.00620>>. 20, 21, 24, 25
- CHATZIKYRIAKIDIS, E. *QueueList Library For Arduino*. [S.l.]: Arduino Playground, 2010. <<http://playground.arduino.cc/Code/QueueList>>. 56
- COLLET, L. *WND and SIGFOX To Extend Global Internet of Things Network to Brazil*. [S.l.]: Sigfox, 2016. <<http://www.sigfox.com/en/press/wnd-and-sigfox-to-extend-global-internet-of-things-network-to-brazil>>. 17
- ECMA. *The JSON Data Interchange Format*. 1st edition. ed. [S.l.], 2013. Disponível em: <<http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>>. 40, 41
- FIELDING, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. Tese (Doutorado) — University of California, Irvine, Estados Unidos, 2000. Disponível em: <<http://www.loa.istc.cnr.it/Guizzardi/SELMAS-CR.pdf>>. Acesso em: 22 fev. 2017. 48
- HOELLER, A. J.; FRÖHLICH, A. A. *Redes de sensores sem-fio sob a perspectiva do EPOS*. Petrópolis, Brasil: [s.n.], 2010. Proceedings of the 11th Symposium on Computing. Disponível em: <http://www.lisha.ufsc.br/pub/Hoeller_WSCAD_2010.pdf>. Acesso em: 14 jul. 2016. 20
- HOPERF. *RFM95/96/97/98(W) - Low Power Long Range Transceiver Module*. [S.l.], 2014. V1.0. Disponível em: <http://www.hoperf.com/upload/rf/RFM95_96_97_98W.pdf>. Acesso em: 22 fev. 2017. 35
- IEEE Standard for Low-Rate Wireless Networks. *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)*, p. 1–709, abr. 2016. 30
- JONES, M.; BRADLEY, J.; SAKIMURA, N. *JSON Web Token (JWT)*. [S.l.], 2015. <<http://www.rfc-editor.org/rfc/rfc7519.txt>>. Disponível em: <<http://www.rfc-editor.org/rfc/rfc7519.txt>>. 48

- LINK LAB'S. *A Comprehensive Look At Low Power, Wide Area Networks*. White Papper, 2016. 17, 20, 21
- LORA-ALLIANCE. Lora™ specification. In: . [S.l.: s.n.], 2016. 24, 25, 26, 27, 28, 29, 31, 32
- LORA-ALLIANCE. 2017. <<https://www.lora-alliance.org/>>. Acessado em 23/02/2017. 33
- MANIACBUG. 2013. <<https://maniacbug.wordpress.com/>>. Acessado em 23/02/2017. 55
- MODTRONIX. *Wireless SX1276 LoRa Module, 868MHz and 915MHz, 3.3V, SMA Connector*. [S.l.], 2014. Disponível em: <<http://modtronix.com/inair9.html>>. Acesso em: 22 feb. 2017. 36
- MOTTOLA, L.; PICCO, G. P. Programming wireless sensor networks: Fundamental concepts and state of the art. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 43, n. 3, p. 19:1–19:51, abr. 2011. ISSN 0360-0300. Disponível em: <<http://doi.acm.org/10.1145/1922649.1922656>>. 11, 19, 20
- NOLAN, K. E.; GUIBENE, W.; KELLY, M. Y. An evaluation of low power wide area network technologies for the internet of things. In: *2016 International Wireless Communications and Mobile Computing Conference (IWCMC)*. [S.l.: s.n.], 2016. p. 439–444. 17
- PETAJAJARVI, J. et al. Evaluation of lora lpwan technology for remote health and wellbeing monitoring. *2016 10th International Symposium on Medical Information and Communication Technology (ISMICT)*, IEEE, Worcester, MA, USA, v. 1, n. 1, p. 1–5, mar. 2016. 21
- PETAJAJARVI, J. et al. On the coverage of lpwans: range evaluation and channel attenuation model for lora technology. In: *2015 14th International Conference on ITS Telecommunications (ITST)*. [S.l.: s.n.], 2015. p. 55–59. 18
- RISINGHF. *Set up LoRaWAN GW with RHF0M301*. [S.l.], 2015. V1.8. 37, 38
- SEMTECH. *AN120.22 LoRa Modulation Basics*. [S.l.], 2015. 21, 22, 23
- STANKOVIC, J. A. Research directions for the internet of things. *IEEE Internet of Things Journal*, IEEE, v. 1, n. 1, p. 3–9, fev. 2014. ISSN 2327-4662. 19
- TELKAMP, T. *Single Channel LoRaWAN Gateway*. [S.l.]: GitHub, 2016. <https://github.com/tftelkamp/single_chan_pkt_fwd>. 37
- THURLOW, R. *RPC: Remote Procedure Call Protocol Specification Version 2*. [S.l.], 2009. <<http://www.rfc-editor.org/rfc/rfc5531.txt>>. Disponível em: <<http://www.rfc-editor.org/rfc/rfc5531.txt>>. 46
- VANGELISTA, L.; ZANELLA, A.; ZORZI, M. Long-range iot technologies: The dawn of lora™. jan. 2015. 22, 23
- WEISER, M. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 3, n. 3, p. 3–11, jul. 1991. ISSN 1559-1662. Disponível em: <<http://doi.acm.org/10.1145/329124.329126>>. 19

ZANELLA, A. et al. Internet of things for smart cities. *IEEE Internet of Things Journal*, v. 1, n. 1, p. 22–32, fev. 2014. ISSN 2327-4662. [19](#)