UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Miguel Henrique de Brito Pereira

Implementação de uma Aplicação sobre LoRaWAN para Verificação de Integração Bidirecional entre LoRa e Dojot

Uberlândia, Brasil

2020

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Miguel Henrique de Brito Pereira

Implementação de uma Aplicação sobre LoRaWAN para Verificação de Integração Bidirecional entre LoRa e Dojot

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Rafael Pasquini

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2020

Miguel Henrique de Brito Pereira

Implementação de uma Aplicação sobre LoRaWAN para Verificação de Integração Bidirecional entre LoRa e Dojot

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Ciência da Computação.

Trabalho aprovado. Uberlândia, Brasil, 12 de dezembro de 2020:

Rafael Pasquini
Orientador

Professor

Professor

Uberlândia, Brasil
2020

Resumo

O presente trabalho procura mostrar o estudo, desenvolvimento, questões de segurança e implementação do protocolo LoRaWAN e integrar a uma rede LoRa com a plataforma Dojot. Sendo apresentado o estudo dos seus componentes, bem como sua topologia de rede. Foi desenvolvido um código para o *end device* no ambiente IoT implementando funções de mensagens de *uplink* e *downlink*. Foram feitos testes e validações, no processo de *uplink* com dados aleatórios gerados no *end device*, sensor, e enviados para o servidor, e posteriormente para o Dojot. Para o processo de *downlink*, os dados que chegaram na plataforma Dojot são mostrados no *front end* e retornados para o *end device*, demonstrando um ciclo entre os componentes. Com auxílio de programas como PuTTY, podem ser observados os dados enviados do *end device* e chegando do Dojot e, com isso, comprovar a eficácia do código e componentes da rede implementada.

Palavras-chave: LoRaWAN, protocolo, LoRa, segurança, Internet das Coisas.

Lista de ilustrações

Figura 1 – Camadas de um sistema IoT - Extraída de (VASHI et al., 2017). 12
Figura 2 – Comparativo do alcance e largura de banda - Extraída de (EMBARCADOS, 2017) 13
Figura 3 – Modelo rede LoRaWAN com componentes fisicos e lógicos - Extraída de(LORA ALLIANCE, 2017b) 15
Figura 4 – Disposição do *payload uplink* - Extraida de (MYDEVICES, 2017) . . . 16
Figura 5 – Tabela de tipos de dados *Cayenne LPP* - Extraida de (MYDEVICES, 2017) 17
Figura 6 – Disposição do *payload downlink* - Extraida de (MYDEVICES, 2017) . . 17
Figura 7 – Exemplo JSON 20
Figura 8 – Componentes Dojot - Extraída de (CPQD, 2019) 23
Figura 9 – Dispositivo STM32 usado como *end device* na rede LoRa. 24
Figura 10 – *Gateway* retornando a mensagem ao dispositivo. 26
Figura 11 – Mensagem recebida pelo dispositivo. 27
Figura 12 – Exebição das informações recebidas de forma decimal no PuTTY. . . . 28
Figura 13 – Conteúdo da mensagem *join request* 29
Figura 14 – Modelo do *gateway* usado no projeto. 29
Figura 15 – Conteúdo da mensagem *join accept* 30
Figura 16 – Segurança *payload* LoRaWAN. Extraída de (LORA ALLIANCE, 2017a) 31
Figura 17 – Rede LoRa integrada a plataforma dojot por meio de um *IoT-Agent* Lora. - Extraída de (RIBEIRO, 2019) 32
Figura 18 – Dados que chegaram no dispositivo 32
Figura 19 – Teste e validação do ciclo desenvolvido e implementado 33
Figura 20 – Dados de *uplink* barômetro(pressão), temperatura e umidade no Dojot 34

Lista de tabelas

Tabela 1 – Alfabeto base 64 - Extraída de (THE INTERNET SOCIETY, 2006). . 22

Lista de abreviaturas e siglas

ACK	Acknowledgement
AES	Advanced Encryption Standard
AppEUI	Application Extended Unique Identifier
AppNonce	Application Number Once
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CSS	Chirp Spread Spectrum
CTR	Counter
CEP	Complex Event Processing
CFList	Channel Frame List
DevAddr	Device Address
DevEUI	Developer Extended Unique Identifier
DES	Data Encryption Standard
DevNonce	Device Number Once
FACOM	Faculdade de Computação
FEELT	Faculdade de Engenharia Eletrica
HTTP	Hypertext Transfer Protocol
HDD	Hard Disk Drive
IEEE	Institute of Electrical and Electronic Engineers
IDE	Integrated Development Environment
IOT	Internet das Coisas
IP	Internet Protocol
ISM	Industrial Sientific and Medical

IPSO	Internet Protocol Smart Objects
JSON	JavaScript Object Notation
JoinEUI	Join Extended Unique Identifier
LPWAN	Rede de longa distância e baixa potência (Low Power Wide Area Network)
LoRaWAN	Protocolo de longa distância e baixa potência (Long Range Wide Area Network)
LPP	Low Power Payload
MAC	Medium Access Control
MACH	Medium Access Control Header
MHz	Megahertz
MIC	Message Integrity Code
MQTT	Message Queuing Telemetry Transport
NetID	Network Identifier
NIST	National Institute of Standards and Technology
OUI	Organizationally Unique Identifier
REST	Representational State Transfer
RAM	Random Access Memory
Wi-Fi	Wireless Fidelity

Sumário

1 INTRODUÇÃO 9
1.1 Objetivos 10
1.2 Método 10
1.3 Resultados 11

2 FUNDAMENTAÇÃO TEÓRICA 12
2.1 IoT 12
2.2 LPWAN (Low Power Wide Area Network) 13
2.3 LoRa 13
2.3.1 LoRaWAN 14
2.3.2 Segurança LoRaWAN 15
2.4 Cayenne Low Power Payload 16
2.5 O pré-processador C 17
2.6 *Message Queuing Telemetry Transport* MQTT 18
2.7 Kafka 19
2.8 JSON 20
2.9 Wireshark 21
2.10 *Advanced Encryption Standard*(AES) 21
2.11 *Base64* 21
2.12 Dojot 22

3 DESENVOLVIMENTO 24
3.1 STM32 24
3.2 Gateway 29
3.3 Chirp Stack 30
3.4 Segurança 30
3.5 Conexão entre rede LoRaWAN e plataforma Dojot 31
3.6 Validações e testes 32

4 CONCLUSÃO 35

REFERÊNCIAS 37

1 Introdução

Internet das Coisas(IoT) é um conceito tecnológico amplo que envolve a conexão de dispositivos à rede, análise e inteligência de dados. A redução dos custos dos componentes eletrônicos possibilitou o surgimento de novos produtos e aplicações, tais como; monitoramento de plantações ou gado, de vias públicas, controle de iluminação, segurança de casa e, entre outros. Todas essas aplicações têm a necessidade de tecnologias de comunicação serem mais viáveis economicamente e tecnologicamente em diferentes cenários urbanos e rurais.

Um sistema completo de IoT é composto por 5 camadas, camada física, camada de rede, camada intermediária, camada de aplicação e camada de negócio. Entretanto, neste projeto vamos destacar 3 camadas: a camada física onde se encontra os *end devices*, a camada intermediária onde se situa os *gateways* e a camada de aplicação onde nos deparamos com os servidores (VASHI et al., 2017). Nos *end devices*, dispositivos finais, tem-se os sensores, que são responsáveis por medições e coleta de dados. Um sensor converte o parâmetro físico em um sinal que pode ser medido eletronicamente. A camada seguinte é composta pelos elementos de conexão entre os sensores e os servidores da rede, chamados de *gateways*.

A comunicação entre esses níveis é feita por tecnologias já consolidadas de comunicações sem fio, como Wi-Fi, Bluetooth, Zigbee, ZWave, etc. Porém, o consumo de energia, vida das baterias e o custo de conexões é atualmente um grande problema para as tecnologias convencionais.

Em contrapartida a esses modelos tradicionais, vêm surgindo novas tecnologias com outras características, por exemplo; longo alcance, baixa taxa de dados, maior vida útil para baterias e faixas de frequências de baixa potência não licenciadas, abaixo de 1 GHz. Nesse contexto, tais tecnologias vêm sendo aplicadas em comunicações de longo alcance, as *Low Power Wide Area Network*(LPWAN) (FARRELL, 2018). Atualmente existem três principais tecnologias que podem se destacar no uso de IoT: LoRa, Sigfox e NB-IoT. Cada uma tem sua particularidade para alcançar seus objetivos.

Dentre as três tecnologias, a LoRa é a única com modelo sem fins lucrativos e *open source*, desenvolvida pela empresa SemTech e hoje mantida pela LoRa Alliance. No caso da Sigfox, a empresa e seus operadores detêm os diretos da tecnologia e *servers*, mas abrem espaço para outras entidades produzirem os *end points*, assim mantendo o baixo preço dos equipamentos finais. Sobre a NB-IoT, a tecnologia é administrada por empresas de telefonia móvel, que cobram taxas de utilização.

Para receber essas informações oriundas de uma rede de dispositivos *gateways*

conectados, temos os servidores agregando um grande volume de dados, então se faz necessário uma plataforma que receba esses dados, para que assim sejam trabalhados e administrados. O Dojot é uma plataforma desenvolvida pelo CPqD em parceria com outras instituições de ciência e tecnologia. É uma plataforma brasileira e surgiu com uma proposta *open source*, para facilitar o desenvolvimento de soluções e utilização de tecnologias de IoT voltado às necessidades do país (DOJOT, 2017). O código de todos os componentes que compõem a solução está disponível no repositório do Github (GITHUBDOJOT, 2017).

Os servidores se conectam no Dojot por meio dos módulos denominados *IoT Agent*. Um *IoT Agent* pode ser entendido como um serviço de adaptação entre dispositivos físicos, ou não, e componentes principais da Dojot. Pode-se ter vários *IoT Agents* no Dojot, cada um deles especializado em um protocolo específico, por exemplo, MQTT, HTTP, etc. Bem como a comunicação, o *IoT Agent* também é responsável por garantir que toda a comunicação é feita por meio de canais seguros (CPQD, 2017).

Neste cenário, o modelo de negócios da LoRa, bem como o Dojot, tem como efeito a facilidade de compra e desenvolvimento de sistemas IoT, principal fator que viabiliza este projeto. Este trabalho propõe o estudo e implementação do protocolo LoRaWan para comunicação entre os *end devices*, *gateways* e servidores, analisando os aspectos de segurança, de armazenamento e capacidade de envio de dados dentro deste protocolo.

1.1 Objetivos

O objetivo desse trabalho foi desenvolver uma rede de sensores com a tecnologia LoRa. Portanto, o enfoque, complementar ao apresentado em (RIBEIRO, 2019), é construir a parte física de uma rede LoRa, possibilitando a coleta de informações com os *end devices* e enviando para o *gateway*, e posteriormente aos servidores, utilizando o protocolo LoRaWAN. Com este cenário em mãos, podemos analisar as características do protocolo LoRaWAN, tais como tipos de dados de que podem ser transferidos, criptografia, configurações a serem adotadas e como utilizar os dados coletados de maneira efetiva.

1.2 Método

O método utilizado nos desenvolvimentos para o projeto e realização dos objetivos foram:

- Revisão bibliográfica e estudo de artigos já publicados sobre tais tecnologias.
- Estudo do RFC 8376 sobre LPWAN (FARRELL, 2018)

- Desenvolver e implementar o código no *end device* para enviar informações para o dispositivo LoRa (*uplink*), bem como o caminho contrário, (*downlink*).
- Construção de uma rede LoRa, com os seguintes componentes, *end-devices*, *gateways*, um servidor e a plataforma Dojot.
- Realizar experimentos de comunicação entre todos os níveis, utilizando LoRaWan e Dojot.

1.3 Resultados

No final do projeto foi possivel a implementação de um sistema de IoT utilizando a tecnologia LoRa e o protocolo LoRaWAN, fazendo os processos de *uplink* e *downlink*. Neste contexto, tal sistema apresentou um alcance medio, entre 20 à 40 metros, no cenário dentro da FACOM, bem como segurança para a informação transportada e compreensão do protocolo utilizado.

2 Fundamentação Teórica

Neste capítulo é apresentado os conceitos básicos necessários para compreensão deste trabalho bem como os trabalhos relacionados.

2.1 IoT

A conexão entre objetos é o primeiro passo para uma solução IoT. A conexão entre dispositivos permite otimização de processos e até mesmo automatização sem intervenção humana. O próximo passo será com objetos inteligentes, gerando um ganho massivo de produtividade na economia.

A IoT é vista como a evolução dos processos atuais de comunicação entre máquinas, conforme os processos forem automatizados e os componentes forem se tornando mais inteligentes, e com isso menos intervenção humana será necessária.

A arquitetura de um sistema IoT envolve um grande número de componentes inteligentes conectados e sensores interconectados. É esperado que a comunicação entre esses dispositivos seja a qualquer momento, em qualquer lugar e sem fio, de forma autônoma. Um sistema IoT é descentralizado em relação aos serviços, gerando uma complexidade, em consequência é requerida uma arquitetura bem estruturada. É definida como uma estrutura para a especificação dos componentes físicos, a organização funcional, a configuração, funcionamento, princípios e procedimentos, bem como formato de dados utilizados(VASHI et al., 2017).

A seguir uma imagem que representa os blocos de uma solução IoT:

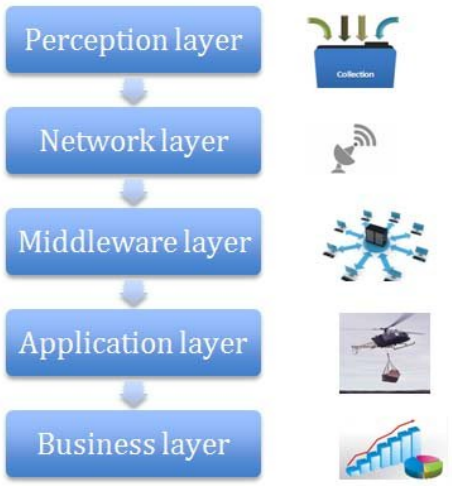

Figura 1 – Camadas de um sistema IoT - Extraída de (VASHI et al., 2017).

2.2 LPWAN (Low Power Wide Area Network)

A rede LPWAN foi projetada para oferecer e suportar uma grande quantidade de dispositivos e aplicações que precisam enviar pequenos pacotes de dados através de longas distâncias, entre 10 quilômetros dependendo do local rural, urbano, etc. Na maioria das tecnologias LPWAN, com exceção de algumas, por exemplo *Weightless-w*, a frequência do sinal é diferente das já conhecidas atualmente, utilizando uma frequência conhecida como *Sub-Ghz*, que é abaixo de 1 Ghz. Apresenta uma robustez e comunicação confiável com um orçamento relativamente baixo. Além desses pontos, vale destacar que frequências *Sub-Ghz* tem menos atenuação e perdas através de obstáculos e superfícies densas como paredes de concreto(RAZA; KULKARNI; SOORIYABANDARA, 2017).

A Figura 2 mostra um comparativo entre frequencias ja conhecidas, indicando o alcance entre elas.

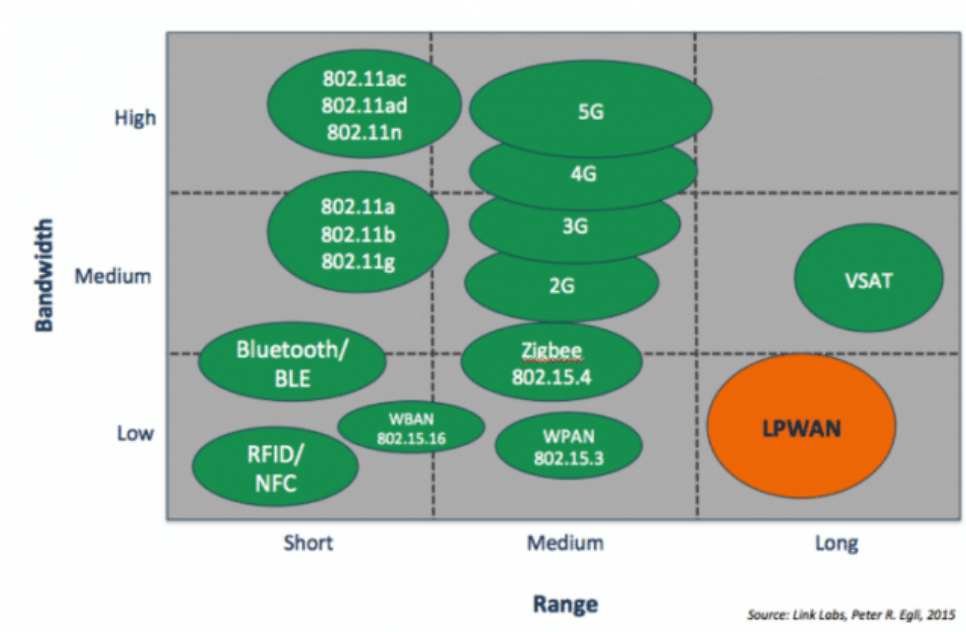

Figura 2 – Comparativo do alcance e largura de banda - Extraída de (EMBARCADOS, 2017)

2.3 LoRa

LoRa, abreviação de *Long Range*, é um sistema de comunicação sem fio, frequentemente refere-se a camada física, e proporciona uma conectividade a longo alcance utilizando uma técnica baseada em modulação CSS (*Chirp Spread Spectrum*), e operando nas bandas entre 868-900 MHz ISM. Originalmente, a modulação CSS foi desenvolvida para radares na década de 1940. Atualmente é usada em comunicações militares e espaciais dado o seu baixo consumo e robustez contra degradação de canal, como interferência e

perdas através do caminho. Enquanto a camada física do LoRa especifica a modulação da banda, o LoRaWAN define a arquitetura e a implementação da camada de protocolo MAC (*Medium Access Control*) da tecnologia LoRa(ARAS et al., 2017).

LoRa é a primeira implementação de baixo custo para uso comercial(LORA ALLIANCE, 2015).

2.3.1 LoRaWAN

Desenvolvido pela SemTech e mantido atualmente pela LoRaAlliance, o LoRaWAN é o protocolo que define a arquitetura do sistema e seus parâmetros para dispositivos baseados no LoRa. Alem disso, também estabelece detalhes de funcionamento, aparelhos, chaves e métodos de criptografia, qualidade de serviço, ajustes de potência, tudo para se ter uma rede sem fio segura e robusta(FARRELL, 2018).

A rede LoRaWAN é normalmente organizada em uma topologia chamada de estrela de estrelas, em que os *gateways* retransmitem mensagens recebidas dos *end devices* a um servidor. Por sua vez, os *gateways* são ligados por uma conexão pelo protocolo IP com o servidor, enquanto os *end devices* utilizam comunicação LoRaWAN que pode ser recebida por um ou mais *gateways*. A comunicação entre todas essas partes citadas anteriormente pode ser bidirecional. Uma rede LoRaWAN é constituída por:

- *End devices* : dispositivos equipados com sensores que captam dados para repassar aos *gateways* utilizando tecnologia LoRa. Conforme as especificações do LoRaWAN, os *end devices* LoRa tem três tipos de canais de comunicação:
  - Classe A : comunicação bidirecional, no qual cada transmissão *uplink* é seguida por duas janelas de recepção *downlink*.
  - Classe B : comunicação bidirecional, mesmo sistema de janelas de recepção citados na classe A, porém recebe janelas de recepções adicionais, que são determinadas por sinais de sincronização com o *gateway*. Também permite que o servidor saiba quando o *end device* está escutando.
  - Classe C : comunicação bidirecional permitindo dados contínuos com o máximo de janelas de recepção(ARAS et al., 2017).
- *Gateways* : dispositivos que recebem os dados dos *end devices* e os entrega para o *Network Server*.
- *Network Server* : componente lógico responsável por receber, processar e analisar os pacotes de dados dos *gateways*. Dentre as outras responsabilidades, podemos citar:
  - checagem de endereço dos *end devices*;

  - envio de ACKs para os *gateways*;
  - direcionamento de dados *uplink* para os respectivos servidores;
  - encaminhamento de dados *downlink* para qualquer *end device* conectado dentro da rede;
  - direcionamento de mensagens de *Join-request* e *Join-accept* entre *end devices* e *Join Servers*.

- *Join Server* : componente lógico que faz o papel do *Network Server*, porém para o lado do servidor, responsável por receber mensagens de *Join request* oriundas dos *end devices* e respectivamente responsável por enviar *Join accept* dos servidores. Dentre outras ações e incumbências, mantém os dados e chaves de sessões dos dispositivos conectados na rede.

- Servidor : servidores de aplicação que recebe os dados da camada anterior, disponibiliza os dados para leitura e aplicações externas. É também responsável pelo *downlink* de dados, processo em que os dados são oriundos do servidor e tem como destino final os *end devices*. Voltado para aplicações de usuário final(LORA ALLIANCE, 2017b).

Na Figura 3 é mostrado uma rede LoRaWAN completa com todos os componentes físicos e lógicos.

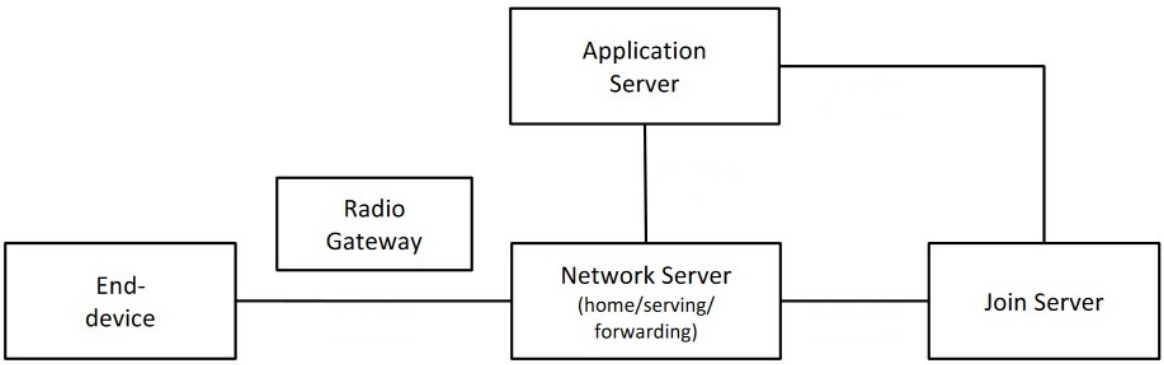

Figura 3 – Modelo rede LoRaWAN com componentes fisicos e lógicos - Extraída de(LORA ALLIANCE, 2017b)

2.3.2 Segurança LoRaWAN

O algoritmo de criptografia usado no LoRaWAN é o AES-128 e operado no modo CRT. Utilizando de chaves diferentes para o *end device*, rede e camada de aplicação, o protocolo garante segurança para os pacotes trafegados entre esses níveis(ARAS et al., 2017).

Para se utilizar o protocolo LoRaWAN, é preciso passar o *DevEUI* do *end device*; todo *end device* LoRa tem um identificador único de 64 bits chamado de *Device Identifier (DevEUI)*, que é definido pelos fornecedores ou desenvolvedores(MYDEVICES, 2017). Além disso, tem-se o *Application Identifier (AppEUI)*, que identifica unicamente um servidor de aplicação para o *end device*(LORA ALLIANCE, 2017b).

Uma chave AES de 128 bits que é dada o nome de *Application Key*, é usada para gerar duas chaves de sessão, *Network Session Key (NwkSKey)* e *Application Session Key (AppSKey)*. A *AppSKey* é utilizada para fazer a cifragem e decifragem dos *payloads* do servidor de aplicação. Usando a *NwkSKey, AppSKey* e o *uplink* ou o *downlink* das mensagens de resposta, o LoRaWAN cria uma chave de fluxo, que posteriormente é utilizada para cifrar cada mensagem aplicando a operação XOR com a chave correspondente da chave do fluxo, e assim gerando a mensagem encriptada. Para garantir a integridade de cada mensagem enviada, o *NwkSKey* é compartilhado entre o *end device* e o *Network Server*, para assim ser gerado o *Message Integrity Code* (MIC). Com todos os procedimentos anteriores, é criado uma assinatura especifica para cada *end device*(ARAS et al., 2017).

2.4 Cayenne Low Power Payload

O *Cayenne Low Power Payload* (LPP) fornece um modo conveniente e fácil de mandar dados através de redes LPWAN, tal como LoRaWAN. Ele utiliza um formato que é de acordo com as restrições de tamanho de *payload*, este pode ser reduzido em até 11 *bytes*, e permite que sensores enviem vários dados de uma só vez. Para se utilizar este formato de *payload*, o *Cayenne* LPP precisará saber o *DevEUI* do *end device*(MYDEVICES, 2017).

O *Cayenne* LPP possibilita enviar dados em vários quadros como ilustrado na Figura 4 . Para tal, cada sensor deve ser antecedido de dois *bytes*:

- *Data Channel* : identifica cada sensor dentro do dispositivo através dos quadros.
- *Data Type* : identifica o tipo de dado dentro do quadro.

1 Byte	1 Byte	N Bytes	1 Byte	1 Byte	M Bytes	...
Data1 Ch.	Data1 Type	Data1	Data2 Ch.	Data2 Type	Data2	...

Figura 4 – Disposição do *payload uplink* - Extraida de (MYDEVICES, 2017)

Cada dado dentro do *payload* utiliza um ou mais *bytes* e os tipos de dados são identificados conforme a IPSO *Alliance Smart Objects Guidelines*, que determina cada

tipo com um *Object ID*, mostrado na Figura 5 . Para se converter o *Object ID* em somente um *byte*, utiliza-se o seguinte metodo:

- CAYENNE_DATA_TYPE = IPSO_OBJECT_ID - 3200.

Type	IPSO	LPP	Hex	Data Size	Data Resolution per bit
Digital Input	3200	0	0	1	1
Digital Output	3201	1	1	1	1
Analog Input	3202	2	2	2	0.01 Signed
Analog Output	3203	3	3	2	0.01 Signed
Illuminance Sensor	3301	101	65	2	1 Lux Unsigned MSB
Presence Sensor	3302	102	66	1	1
Temperature Sensor	3303	103	67	2	0.1 °C Signed MSB
Humidity Sensor	3304	104	68	1	0.5 % Unsigned
Accelerometer	3313	113	71	6	0.001 G Signed MSB per axis
Barometer	3315	115	73	2	0.1 hPa Unsigned MSB
Gyrometer	3334	134	86	6	0.01 °/s Signed MSB per axis
GPS Location	3336	136	88	9	Latitude : 0.0001 ° Signed MSB
					Longitude : 0.0001 ° Signed MSB
					Altitude : 0.01 meter Signed MSB

Figura 5 – Tabela de tipos de dados *Cayenne LPP* - Extraida de (MYDEVICES, 2017)

O *payload downlink* não conta com o campo tipo de dados, tornando a estrutura mais simples que a mencionada anteriormente, Figura 6.

1 Byte	2 Bytes	1 Byte
Data Ch.	Data (.01 Precision)	0xff

Figura 6 – Disposição do *payload downlink* - Extraida de (MYDEVICES, 2017)

2.5 O pré-processador C

A primeira fase de compilação de um programa em linguagem C é a parte que entra em ação o pré-processador. Nesta fase são definido os *macros* que serão usados pelo programa. O pré-processador oferece 4 facilidades:

- Inclusão de arquivos de cabeçalhos..

- Definição de *macros*, que são abreviações para fragmentos do código C, exemplo *define MAXIMO 10*
- Compilação condicional, utilizando de diretivas no código, o pré-processador irá incluir ou excluir partes do programa segundo as condições dessas diretivas.
- Controle de linhas, capacidade de informar ao compilador de onde cada linha originalmente originou.

(FREE SOFTWARE FOUNDATION, 2001)

2.6 *Message Queuing Telemetry Transport* MQTT

O *Message Queuing Telemetry Transport* (MQTT) é um protocolo extremamente simples e leve de troca de mensagens para sensores e pequenos dispositivos que possuem limitação de banda, latência alta e utilizam redes não confiáveis. É fundamentado no modelo *publish/subscribe*, e tem como objetivo minimizar o uso de rede e recurso do dispositivos ao mesmo tempo que tenta manter confiabilidade e garantia de entrega. Esses princípios faz com que o protocolo seja ideal para *machine-to-machine* (M2M), IoT e aplicações móveis onde a largura de banda e vida útil da bateria são essenciais(MQTT, 2019).

Inventado em 1999 por Andy Stanford-Clark da IBM, e Arlen Nipper da Arcom (agora na Eurotech), o MQTT funciona no modelo cliente/servidor, onde o cliente se conecta a um servidor, conhecido como *broker*, e este servidor é responsável por receber todas as mensagens dos clientes e, em seguida, rotear essas mensagens para os clientes de destino relevantes. Seguindo esse padrão, um cliente pode subscrever a um endereço, também conhecido como tópico, para que assim possa receber mensagens dele ou mandar mensagens para outro tópico. Um cliente pode se subscrever a vários tópicos, assim fazendo, passa a receber todas as mensagens enviadas publicadas nesses tópicos. Exemplo de funcionamento do MQTT:

- O cliente conecta-se ao *broker*. Ele pode assinar qualquer tópico de mensagem no *broker*. Lembrando que um cliente pode ser qualquer coisa que possa interagir com o *broker* e receber mensagens.
- O cliente publica as mensagens em um tópico, enviando a mensagem e o tópico ao *broker*.
- Em seguida, o *broker* encaminha a mensagem a todos os clientes que assinam esse tópico.(IBM, 2017)

2.7 Kafka

O Apache Kafka foi originalmente desenvolvido em 2011 pelo LinkedIn para processar grande volume de dados de log em tempo real e conseguir rotea-los a múltiplos consumidores com tempo de atraso bem pequeno. Foi projetado para ser um sistema de mensagem distribuído, escalável, durável e tolerante a falhas.

Atualmente é uma plataforma *open source*, sendo utilizada por diversas empresas de tecnologias voltadas para a internet. O *Twitter* usa a solução como parte da sua estrutura de processamento, e o *Netflix* aplica o Kafka como ferramenta de monitoramento e eventos em tempo real.

A plataforma Apache Kafka proporciona um serviço de mensagem produtor-consumidor, onde o produtor manda uma mensagem para um tópico dentro do *cluster* do Kafka, e o consumidor que subscreveu a este tópico irá processar e ler a mensagem publicada. O tópico pode ser descrito como uma categoria lógica de mensagem.

Um *cluster* do Kafka consiste em vários *brokers*, servidores Kafka, e todas aspartições do mesmo tópico são distribuídas através do *brokers*(WU; ZHIHAO; WOLTER, 2019).

O Apache Kafka segue as seguintes características:

- **Mensageria persistente**: Para lidar com uma abundância de dados, qualquer perda de dados não pode ser aceitável, dessa forma, o Apache Kafka foi desenvolvido com as estruturas de dados O(1), que proporcionam uma performance em tempo constante mesmo com um grande volume de mensagens.
- **Alta taxa de transferência**: O Apache Kafka foi projetado para tratar com centenas de milhares de mensagens por segundo, mesmo com *hardwares* básicos.
- **Distribuído**: Tem sua estrutura baseada em *clusters*, podendo crescer horizontalmente e de forma transparente sem nenhum tempo de inatividade. Suporta o particionamento de mensagens pelos servers e seu consumo através dos *clusters*, podendo ser escalado e particionando entre as máquinas consumidoras e mantendo a a ordem semântica de cada partição
- **Suporte a múltiplos clientes**: O sistema Apache Kafka suporta a integração de clientes de diferentes plataformas como *Java*, *.NET*, *PHP*, *Ruby* e *Python*.
- **Tempo real**: Todas as mensagens produzidas pelos produtores devem ser imediatamente visíveis para os consumidores. Essa característica é comum em sistemas baseado em eventos como *Complex Event Processing* (CEP)(GARG, 2013).

2.8 JSON

Derivado do *JavaScript*, o JSON (*JavaScript Object Notation* - Notação de Objetos *JavaScript*) é uma formatação leve de troca de dados fácil de ler, escrever, interpretar e gerar. O padrão é em formato texto e totalmente independente de linguagens de programação, pois utiliza convenções que são familiares às linguagens C e afins(JSON.ORG, 2002). Dentro do JSON, temos dois tipos estruturados:

- Temos o objeto, constituído por um conjunto desordenado de pares chave/valor. Um objeto começa com *{* e termina com *}*, dentro do objeto cada chave/valor é separado por dois pontos e para dividir os pares de chave/valor é usado a vírgula. Além do tipo objeto, temos também o vetor, que representa uma lista ordenada de valores. O vetor começa com *[* e termina com *]*, e os valores são separados por vírgula.

  Para valores, temos os tipos strings, números e booleanos:

- Uma string dentro do JSON é caracterizado como uma coleção de nenhum ou mais caracteres Unicode, envolvido entre aspas duplas. Os nomes das chaves são considerados string e por isso devem seguir esse padrão. Os números e booleanos podem ser representados por *true* ou *false*, no caso do tipo booleano, e inteiro ou flutuante para números.

A seguir podemos ver um exemplo de JSON, com um vetor com três objetos. E suas chaves *date* e *locations*, sendo esse último um vetor dentro de cada objeto.

```
[
  {
    "date":"2013-11-05",
    "locations":{
      "United States":4,
      "Germany":8
    }
  },
  {
    "date":"2013-11-11",
    "locations":{
      "South Africa":9
    }
  },
  {
    "date":"2013-11-12",
    "locations":{
      "Japan":6
    }
  }
]
```

Figura 7 – Exemplo JSON

2.9 Wireshark

O Wireshark é um programa para análise de tráfego de rede com licença de *software* livre. Com ele é possível capturar todos os dados de entrada e saída do computador de todas as interfaces disponíveis no sistema, bem como organizar e filtrar por diferentes protocolos e portas. É possível utilizar pela interface gráfica ou pelo terminal do Linux, como também é possível salvar em um arquivo os dados capturados pela aplicação(WIRESHARK FOUNDATION, 2020).

2.10 *Advanced Encryption Standard*(AES)

O *Advanced Encryption Standard*(AES) foi criado em 2001 pelo Instituto Nacional de Padrões e Tecnologia (NIST), situado nos Estados Unidos, através de uma competição para encontrar o melhor algoritmo de criptografia. Ele substitui o antigo e defasado *Data Encryption Standard* (DES) publicado em 1977 e utiliza o método de criptografia simétrica, ou seja, a mesma chave é usada para criptografar e descriptografar os dados. O algoritmo utiliza blocos de 128 bits e chaves para encriptar e decriptar de tamanhos 128, 192 e 256, inicialmente proposto como pré-requisitos na competição(NIST, 2001).

2.11 *Base64*

Métodos de codificação são usados geralmente para transferir dados em situações que são restritas a caracteres ASCII ou em aplicações que torna possível manipular objetos com editores de texto. O método de codificação *base64* foi projetado para representar sequências arbitrarias de octetos usando letras maiúsculas, minúsculas e números.(THE INTERNET SOCIETY, 2006)

O processo começa recebendo três octetos, sendo 24 bits, como entrada e retornando uma string de 4 caracteres codificados. Esses 24 bits são então tratados como a concatenação de 4 grupos de 6 bits, cada um dos quais é convertido em um único caractere no alfabeto base 64.(THE INTERNET SOCIETY, 2006)

Para se ter um exemplo prático, a palavra “Meu”, tem os caracteres “M”, “e”, e “u” que são armazenados como *bytes* com valores 77, 101, e 117, cujos valores binários são 01001101, 01100101, e 01110101. Estes 3 valores são concatenados formando um valor de 24 bits, produzindo 010011010110010101110101. Da esquerda para a direita, separando em 4 grupos de 6 bits, temos 010011, 010110, 010101 e 110101 que corresponde aos valores 19, 22, 21 e 53 respectivamente. Após isso, pegando esses valores e traduzindo utilizando a Tabela 1, temos “TWV1” em base 64. Assim temos todo o processo de codificacao e traducao do metodo base 64.

Tabela 1 – Alfabeto base 64 - Extraída de (THE INTERNET SOCIETY, 2006).

Valor	Caractere	Valor	Caractere	Valor	Caractere	Valor	Caractere
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

2.12 Dojot

A Dojot foi projetada com o objetivo de desenvolver tecnologias para as cidades inteligentes, oferecendo uma plataforma fácil de usar, escalável e robusta. A plataforma é *open source* e a princípio tem foco em projetos de segurança pública, mobilidade urbana e saúde. É um projeto brasileiro com conteúdo local e assume o papel habilitador(DOJOT, 2017) por meio de:

- APIs abertas tornando o acesso fácil das aplicações aos recursos da plataforma
- Gerenciamento do ciclo de vida de dispositivos;
- Construção de fluxos de dados e processamento de dados em tempo real;
- Persistência dos dados;
- Interface para acesso aos dados em tempo real.

Na plataforma, os dispositivos de IoT podem ser configurados através da interface gráfica ou pelas APIs REST fornecidas pelo API *Gateway*. Assim como os dispositivos, os fluxos de processamento de dados podem ser configurados, abrindo o leque para varias ações que podem ser executadas por essa entidade, bem como gerar notificações quando um determinado atributo de dispositivo específico atingir um valor, etc. Depois de configurado os itens mencionados, os dispositivos começam a enviar as informações coletadas para o Dojot, podemos ter acesso as informações coletadas pelos dispositivos em tempo

real pelos canais *socket.io*, disponibilizar os dados em dispositivos virtuais e a persistência desses dados(CPQD, 2019).

A arquitetura é composta por vários componentes *open source* e outros projetados e implementados pela equipe Dojot. A figura 8, a seguir, representa os principais componentes dessa arquitetura.

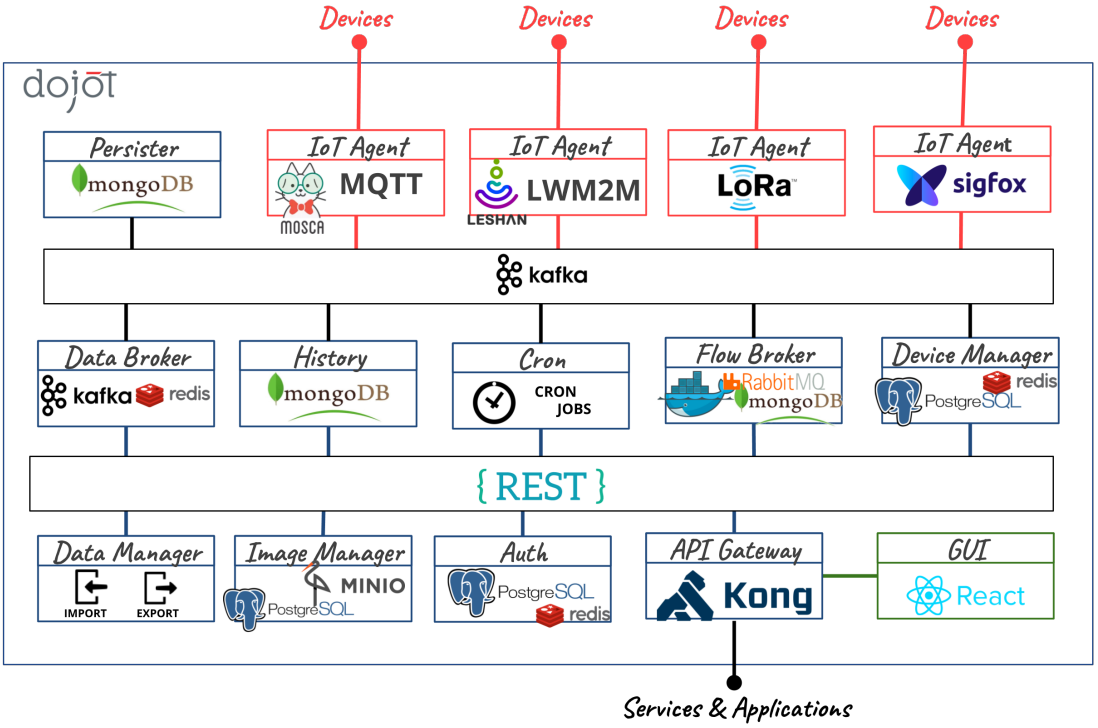

Figura 8 – Componentes Dojot - Extraída de (CPQD, 2019)

3 Desenvolvimento

Este capítulo aborda o desenvolvimento do trabalho, de forma a apresentar a implementação das camadas inferiores, seguindo as definições do protocolo LoRaWAN, para assim executarmos o *uplink* e *downlink* de dados. Mostrando as alterações dentro do código gravado no sensor STM32, bem como suas definições de dados e aspectos de segurança envolvidos. Neste capítulo também são abordadas as camadas superiores, conectividade entre o LoRa e Dojot, bem com a instalação e configuração da plataforma Dojot e também o desenvolvimento e implantação de um *IoT Agent* LoRa, a fim de integrar a rede LoRa ao Dojot(RIBEIRO, 2019).

Com o cenário definido, foi implementado uma rede LoRa seguindo as bases do protocolo LoRaWAN. Para fazer a parte do *end device*, utilizamos o sensor STM32 para enviar e receber informações, também foi necessário um *gateway* para receber e transmitir dados entre o *end device* e o servidor, este últimonecessário para recolher/manipular essas informações.

3.1 STM32

O sensor STM32 (Figura 9) utilizado no projeto foi fornecido pela Faculdade de Engenharia Elétrica, FEELT, e foi usado no fluxo para *downlink*, recebendo os dados vindos do servidor, e no fluxo de *uplink*, gerando os dados e mandando em intervalos de 10 segundos com o *payload* no formato *Cayenne* LPP para o servidor. A escolha desse *payload* se deu por ser um formato suportado pelo servidor LoRa *ChirpStack*.

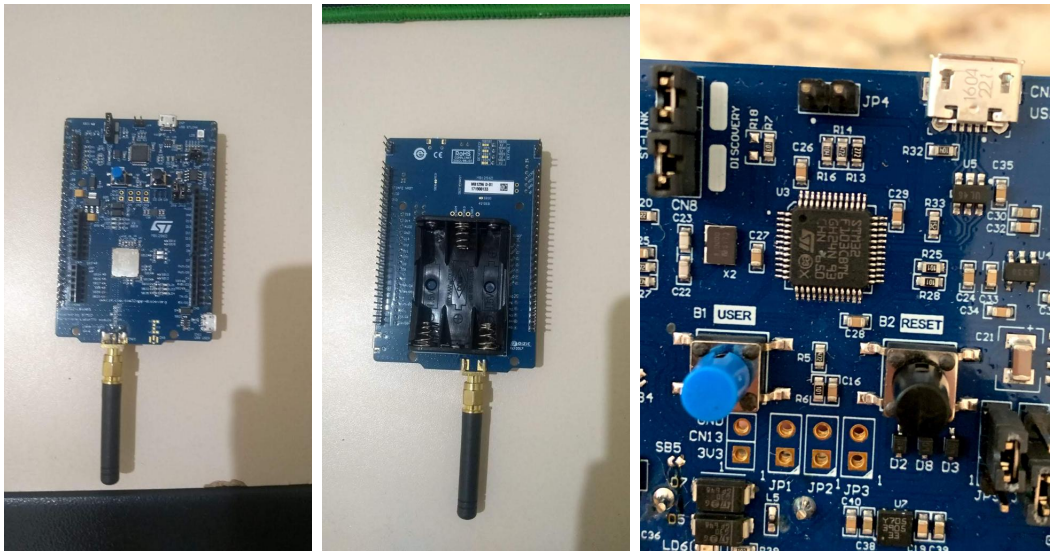

Figura 9 – Dispositivo STM32 usado como *end device* na rede LoRa.

Foi utilizado a IDE *Atollic True STUDIO for STM32*, que é distribuído de forma gratuita para uso não comercial, para desenvolver o código e visualizar e alterar as chaves de acesso entre o dispositivo e o *gateway*. Para se obter comunicação entre o sensor, *gateway* e o servidor, precisamos cadastrar o *DevEUI* e o *JoinEUI* (versões antigas, era chamado de *AppEUI*) manualmente no sensor, no *gateway* e servidor, quanto as demais chaves de segurança, não é preciso tal operação manual. Esses dados ficam armazenados dentro do arquivo *Commissioning.h*, dentro do projeto do software que gravamos no dispositivo. Foram usados os seguintes ID's e chaves:

- *IEEE OUI* : 0x01, 0x01, 0x01
- *LORAWAN DEVICE EUI* : IEEE OUI, 0x01, 0x01, 0x01, 0x01, 0x01
  Obs: usado concatenado com o IEEE OUI seguindo as recomendações da propria semtech
- *LORAWAN JOIN EUI* : 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
- *LORAWAN APP KEY* : 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
- *LORAWAN NWK KEY* : 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
- *LORAWAN DEVICE ADDRESS* : 0x0100000a
- *LORAWAN F NWK S INT KEY* : 0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6, 0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C
- *LORAWAN S NWK S INT KEY* : 0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6, 0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C
- *LORAWAN NWK S ENC KEY* : 0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6, 0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C
- *LORAWAN APP S KEY* : 0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6, 0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C

O código alterado e posteriormente implementado no sensor para as operações de *uplink* e *downlink* teve o propósito de simular um ambiente em que os dados são gerados no dispositivo, enviado para o *gateway*, e posteriormente para o servidor, esse por sua vez, retorna os dados para o dispositivo sem altera-los, mostrado na Figura 10. Após chegar ao sensor, incrementamos os valores dos dados de cada tipo. Desse modo, os dados que trafegam são modificados e, assim, temos uma métrica para cada interação no sistema. Os tipos de dados já disponibilizados no *payload Cayenne* foram temperatura, umidade

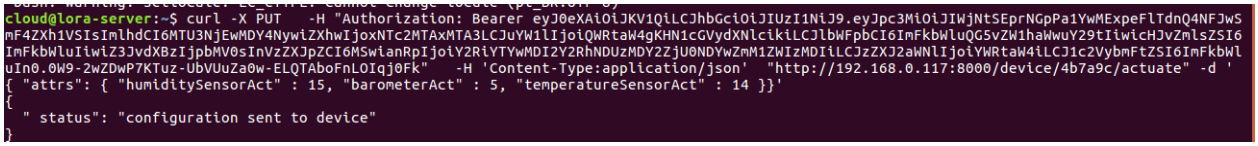

Figura 10 – *Gateway* retornando a mensagem ao dispositivo.

e barômetro (pressão), nos canais 0, 1 e 2 respectivamente. Para habilitarmos o *payload Cayenne*, colocamos o seguinte trecho no código:

```
#define CAYENNE_LPP
```

Tendo o *payload* habilitado, e seguindo as recomendações do *payload*, as variáveis foram criadas e alocados números aleatórios da seguinte maneira, respectivamente:

```
  uint16_t pressure = 0;
  int16_t temperature = 0;
  uint16_t humidity = 0;

  temperature = ( int16_t ) rand() % 50 ;
  pressure    = ( uint16_t ) rand() % 1000 ;
  humidity    = ( uint16_t ) rand() % 100 ;
```

Foi usado o int16_t para a variável **temperature** para contemplar os números positivos e negativos entre -32.768 e 32.767. Para as outras duas variáveis foi utilizado uint16_t para abranger somente os números positivos de 0 a 32.767. Os dados aleatórios foram gerados utilizando a função **rand()**, para tal foi utilizado as seguintes diretrizes no código:

```
#include <time.h>
time_t t;
srand(time(&t));
```

A função **srand()** recebe a *seed* para gerar os números aleatórios baseados no tempo atual do sistema, adquirido pela função **time()** e pela variável **t**.

Após definidas as variáveis, a mensagem é determinada com o vetor do tipo **lora_AppData_t** dessa forma:

```
  AppData.Port = LPP_APP_PORT;

  AppData.Buff[i++] = cchannel++;
  AppData.Buff[i++] = LPP_DATATYPE_BAROMETER;
  AppData.Buff[i++] = ( pressure >> 8 ) & 0xFF;
  AppData.Buff[i++] = pressure & 0xFF;
  AppData.Buff[i++] = cchannel++;
  AppData.Buff[i++] = LPP_DATATYPE_TEMPERATURE;
  AppData.Buff[i++] = ( temperature >> 8 ) & 0xFF;
  AppData.Buff[i++] = temperature & 0xFF;
```

```
11  AppData.Buff[i++] = cchannel++;
12  AppData.Buff[i++] = LPP_DATATYPE_HUMIDITY;
13  AppData.Buff[i++] = humidity & 0xFF;
```

Cada posição do vetor tem a capacidade de oito *bits*, então para os dados temperatura e barômetro (pressão) que podem assumir valores acima de 255, usamos o deslocamento de *bits* para a direita para podermos alocar o valor total sem perder nenhuma informação. Vale ressaltar que foi usada a verificação **AND 0xFF** no código, para se ter certeza do valor que foi mandado. O padrão de passar o canal, o tipo de dado e o valor foi citado nos capítulos anteriores, como padrão do *Cayenne* LPP.

Após estabelecidas as mensagens e as informações, passamos para a função **LORA_send** do seguinte modo:

```
1  PRINTF("REACHED HERE, JUST SENDING\n\r");
2
3  LORA_send( &AppData, LORAWAN_DEFAULT_CONFIRM_MSG_STATE);
```

Na variável **LORAWAN_DEFAULT_CONFIRM_MSG_STATE** é passado o valor **LORAWAN_UNCONFIRMED_MSG** que representa a não confirmação da mensagem enviada. Então, caso a mensagem for perdido ou não chegar no destino, que no nosso caso é o *gateway*, não será feito o reenvio.

Determinado o processo montagem e envio das mensagem para o *gateway*, o processo de *uplink* no dispositivo, podemos partir para o recebimento das mensagens.

Ao colocarmos o canal como classe C, uma janela de transmissão é aberta e fica disponível para receber mensagens logo após o envio. No dispositivo, recebemos a mensagem na função **LORA_RxData** por um ponteiro para um vetor do tipo **lora_AppData_t**, ao receber a mensagem foi adicionado um trecho no código para mostrar os dados recebidos, como é mostrado na Figura 11.

```
PACKET RECEIVED ON PORT 10
 DATA
1
,103
,0
,14
,2
,104
,14
,0
,115
,0
,14
,REACHED HERE
```

Figura 11 – Mensagem recebida pelo dispositivo.

Para visualizar o funcionamento do sensor foi utilizado o aplicativo PuTTY, que permite a emulação de terminal a partir de uma conexão serial com o dispositivo (MIT, 2020).

Em seguida da exibição dos dados, foi adicionado no código um trecho para incrementar o valor de uma unidade ao valor recebido dos dados. O trecho do código é mostrado a seguir:

```
1  PRINTF("SENDING FROM RECEIVE DATA\n\r");
2
3  int16_t temperatureAltered = (AppData->Buff[2] << 8) + (AppData->Buff[3]
    & 0xFF);
4  uint16_t pressureAltered = (AppData->Buff[9] << 8) + (AppData->Buff[10] &
     0xFF);
5
6  PRINTF("HERE IS TEMPERATURE RECEIVED = %d\n\r", temperatureAltered);
7  PRINTF("HERE IS PRESSURE RECEIVED = %d\n\r", pressureAltered);
8  PRINTF("HERE IS HUMIDITY RECEIVED = %d\n\r", AppData->Buff[6]);
9
10 temperatureAltered++;
11 pressureAltered++;
12
13 AppData->Buff[2] = (temperatureAltered >> 8) & 0xFF;
14 AppData->Buff[3] = temperatureAltered & 0xFF;
15 AppData->Buff[6]++;
16 AppData->Buff[9] = (pressureAltered >> 8) & 0xFF;
17 AppData->Buff[10] = pressureAltered &0xFF;
18
19 LORA_send( AppData, LORAWAN_DEFAULT_CONFIRM_MSG_STATE);
```

Ao incrementar o valor de uma unidade decimal aos dados, temos o incremento de 0.1 celsius a temperatura, 0.1 hPa ao barômetro (pressão) e 0.5% a umidade. Então ao final do trecho temos o uso da função de envio da mensagem, e desse modo é formado o ciclo do processo de *uplink* e *downlink*. A Figura 12 a seguir mostra como é a visualização dentro do PuTTY dos dados em forma decimal e logo após o envio.

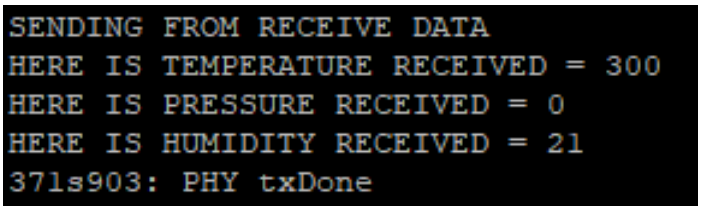

Figura 12 – Exebição das informações recebidas de forma decimal no PuTTY.

A ativação do processo de envio e recebimento de mensagem só é iniciada após a troca das mensagens *join request* e *join accept* entre o sensor e servidor. O processo é sempre iniciado pelo sensor enviando a mensagem de *join request*. A mensagem de *join*

request contém o *DevEUI*, o *AppEUI(JoinEUI)* e 2 octetos chamado de *DevNonce* (Figura 13). O *DevNonce* é um número aleatório gerado e usado unicamente em cada mensagem.

Tamanho Bytes	8	8	2
Join Request	DevEUI	AppEUI(JoinEUI)	DevNonce

Figura 13 – Conteúdo da mensagem *join request*

Após o recebimento da mensagem pelo servidor, o mesmo envia a mensagem de *join accept* para o sensor confirmando a comunicação entre os componentes.

3.2 Gateway

O *gateway* usado no projeto foi disponibilizado pela Faculdade de Engenharia Elétrica, FEELT, na Figura 14 é mostrado um aparelho do mesmo modelo utilizado no projeto. O *gateway* foi configurado para coletar os dados na frequência LoRa do sensor e enviar para o IP do servidor LoRa, funcionando como um intermediário entre os dois.

Após feito as configurações com as chaves *DevEUI* e *JoinEUI(AppEUI)* para permitir a comunicação entre o sensor, *gateway* e servidor, o *gateway* foi colocado no segundo andar da Faculdade de Computação, FACOM, e todos os testes realizados depois das configurações foram feitos em torno de 20 à 40 metros de distância entre os dois componentes.

Figura 14 – Modelo do *gateway* usado no projeto.

3.3 Chirp Stack

O servidor Chirp Stack é uma solução *open source* e oferece uma *stack* LoRa com um LoRa *Network Server* e um LoRa *Application Server*, incluindo uma interface *web* fácil para gerenciamento de dispositivos e APIs para integração. Por esses motivos a plataforma Chirp Stack foi escolhida como servidor LoRa na rede. O servidor foi instalado em uma máquina com 2 vcpus, 4GBs de memória RAM e 100 GBs de disco, e localizado no segundo andar do prédio da Faculdade de Computação, FACOM.

A versão do servidor instalada foi a 3, e foi feita utilizando o Docker Compose, portanto, foi necessário instalar Docker e Docker Compose antes da instalação do servidor Chirp Stack.

Dentro da interface do servidor foi cadastrado o sensor para estabelecer a conexão, usando as chaves *DevEUI*, *JoinEUI(AppEUI)* e *DevAddr(Device Address)*. Após cadastrado o dispositivo, o servidor recebe a mensagem de *request join* e responde com a mensagem de *join accept* permitindo o sensor a se ingressar na rede. A mensagem de *join accept* contém o *AppNonce*, o *NetID*, o *DevAddr* usado no momento do cadastro do sensor, *RxDelay* e a informação opcional *CFList* de lista de canais, mostrado na Figura 15.

Tamanho Bytes	3	3	4	1	1	16 Opcional
Join Accept	AppNonce	NetID	DevAddr	DLSettings	RxDelay	CFList(Não usado)

Figura 15 – Conteúdo da mensagem *join accept*

Caso o servidor não aceite a requisição de entrada na rede do sensor, não será enviada uma resposta de recusa de acesso.

3.4 Segurança

Para garantir a integridade e confidencialidade das informações do protocolo LoRaWAN, o tráfego é protegido usando duas chaves de sessão, *AppSKey* e *NwkSKey*. Todo *payload* é encriptado usando a *AppSKey* e levando consigo um contador de *frame(FCnt)*, para evitar a repetição do pacote. Para a segurança na transmissão dos dados, o pacote inteiro é encriptado com a *NwkSKey* e um *Message Integrity Code* (MIC) é computado a partir dessa chave, para assim evitar a violação do pacote (LORA ALLIANCE, 2017a).

As mensagem de *join request* e *join accept* não são encriptadas, mas para garantir a integridade é acrescentado o *AppKey* e o *MAC header* (MACH) no início das mensagens e computado um *Message Integrity Code* (MIC) e adicionado no final do pacote.

O processo de criptografia do *payload* e pacote já é feito pelo sensor *gateway*, e servidor automaticamente pegando as chaves do arquivo *Commissioning.h*.

Usando as chaves já citadas anteriormente, temos a ilustração do processo de criptografia do *payload* na Figura 16 a seguir.

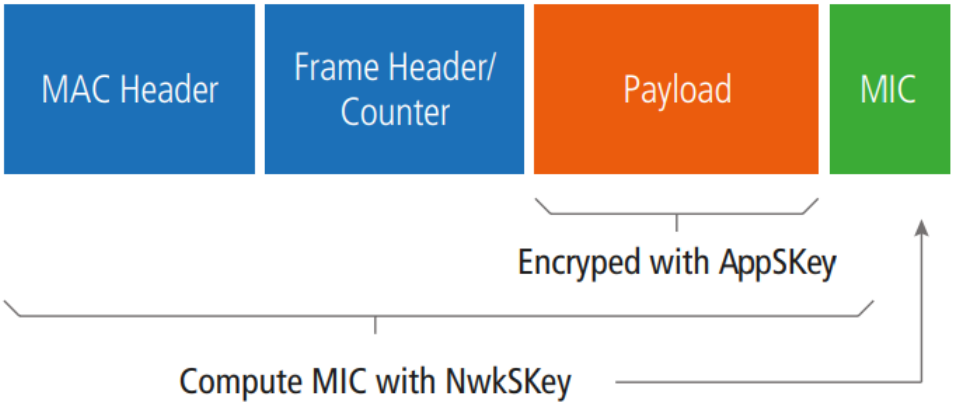

Figura 16 – Segurança *payload* LoRaWAN. Extraída de (LORA ALLIANCE, 2017a)

3.5 Conexão entre rede LoRaWAN e plataforma Dojot

A plataforma dojot foi instalada em uma máquina na Faculdade de Computação, FACOM, com as configurações de 2 vcpus, 4GBs de memória RAM e 100Gbs de HDD e o processo foi feito usando Docker Compose. A versão instalada foi a v0.3.0 e como foi utilizado Docker Compose, a instalação do Docker e Compose teve quer ser feita previamente na máquina.

Na plataforma Dojot foi criado um *template* com os parâmetros barômetro(pressão), temperatura e umidade, que são os mesmos dados gerados no *end device*. O Dojot permite a conexão e coleta de dados de diferentes tecnologias por meio dos *IoT Agents* que se comunicam em diferentes protocolos, dentre eles MQTT/JSON e HTTP/JSON (RIBEIRO, 2019).

A integração entre a rede LoRa e a plataforma Dojot foi feita com a construção de um *IoT Agent*, coletando os dados oriundos do servidor LoRa, no processo de *uplink*, e disponibilizando para o Dojot. Para o processo de *downlink*, o *IoT Agent* coleta os dados do Dojot e disponibiliza para o servidor LoRa. O protocolo utilizado para essa parte foi o MQTT, sendo escolhido por ser suportado entre os componentes (RIBEIRO, 2019).

Deste modo o sistema implementado e colocado em prática, foi o desenvolvimento de um *IoT Agent* LoRa para conectar o servidor da rede LoRa a plataforma Dojot. Então, dessa forma conseguimos implementar os dois fluxos, de *uplink* e *downlink* de dados, sendo iniciado pelo *end device*, e o outro com o processo começando pela plataforma, respectivamente. A topologia do cenário que foi desenvolvido esta ilustrada na Figura 17.

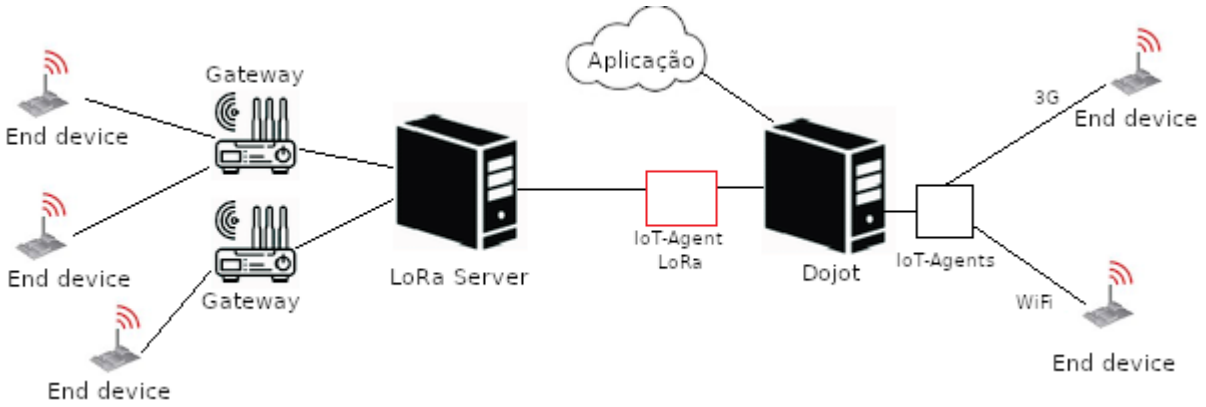

Figura 17 – Rede LoRa integrada a plataforma dojot por meio de um *IoT-Agent* Lora. - Extraída de (RIBEIRO, 2019)

3.6 Validações e testes

Com o sistema desenvolvido e implementado, bem como os processos de *uplink* e *downlink*, a próxima etapa do projeto foram os testes e validações do sistema. Para as validações com o processo começando na plataforma Dojot, *downlink*, utilizamos as exibições no código do sensor, exibindo os dados que chegam no sensor oriundos da plataforma Dojot, Figura 18.

```
PACKET RECEIVED ON PORT 10
 DATA
2
,103
,0
,140
,0
,104
,30
,1
,115
,0
,50
,REACHED HERE
SENDING FROM RECEIVE DATA
HERE IS TEMPERATURE RECEIVED = 140
HERE IS PRESSURE RECEIVED = 50
HERE IS HUMIDITY RECEIVED = 30
```

Figura 18 – Dados que chegaram no dispositivo

Na Figura 19 é mostrado os dados que chegaram no sensor, com todos os dados iniciados em "13" e sendo adicionados uma unidade decimal a cada interação com o dispositivo, isso demonstra o ciclo desenvolvido e implementado.

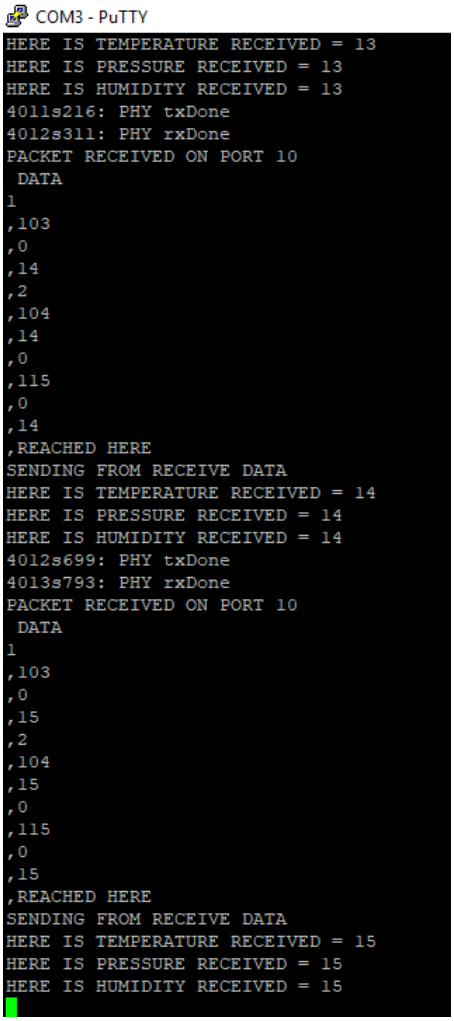

Figura 19 – Teste e validação do ciclo desenvolvido e implementado

Nos testes e validações do processo de *uplink*, o sensor enviava mensagens com dados aleatórios de barômetro (pressão), temperatura e umidade, essas mensagens eram capturadas pelo IoT Agent do servidor LoRa e direcionadas para o Dojot, na Figura 20 são mostrado os gráficos representando as informações vindas do sensor.

Figura 20 – Dados de *uplink* barômetro(pressão), temperatura e umidade no Dojot

4 Conclusão

O desenvolvimento do presente estudo possibilitou entender como é o funcionamento do protocolo LoRaWAN e a rede LoRa. Com isso o objetivo apresentado foi desenvolver e implementar uma rede LoRa com a plataforma Dojot e o protocolo LoRaWAN com todos os seus componentes, a se atentar para as questões de quantidade de dados transmitidos, segurança e integridade das informações com o propósito de fazer o processo de *uplink* e *downlink* entre as camadas do componentes do ambiente implementado.

A disponibilidade dos materiais de estudo da rede e do protocolo apresentado são limitados a poucas fontes e em inglês, bastante completos, porém, com poucas opções, limitados a aliança de empresas que mantém os direitos da tecnologia. A barreira da linguagem não foi problema, porém a falta de opções durante as pesquisas foi um inconveniente. Ao procurar sobre os assuntos na *internet*, quase sempre nos levavam aos mesmos artigos.

Os componentes físicos, o sensor e *gateway*, utilizados no desenvolvimento do projeto foram disponibilizados pela Faculdade de Engenharia Elétrica, FEELT. Por conta de não possuirmos os equipamentos, a disponibilidade ficou a cargo da flexibilidade da FEELT a nos ceder os equipamentos. O sensor foi obtido com facilidade, e com isso o manuseio, configuração e desenvolvimento do código para os processos de *uplink* e *downlink*, foram rapidamente entendidos e implementados no sensor. O código foi desenvolvido em C, utilizando funções do pré-processador do compilador da linguagem e a IDE *TrueSTUDIO for STM32*. Nos estudos do código e do protocolo foram observadas as funcionalidades do *Cayenne* LPP e escolhido como *payload* padrão do projeto. Por outro lado, o *gateway* foi difícil ser obtido para o projeto, e consequentemente as configurações com o servidor foi sendo postergada e, posteriormente, foram encontradas dificuldades nas conexão entre o sensor e o servidor *LoRa The Things Network*(TTN), inicial escolhido para o projeto, porém, devido a essas dificuldades, foi finalmente substituído pelo Chirp Stack.

Após configurado o servidor, foi desenvolvido um *IoT Agent* para fazer a conexão entre o servidor e Chirp Stack e a plataforma Dojot atraves do MQTT. Permitindo assim manusear dados oriundos do sensor, no processo de *uplink*, e mandar dados da camada mais alta da rede, Dojot, para os sensores no processo de *downlink*.

Com a configuração e implementação do Dojot, foi póssivel a realização dos testes e validações dos dados nas mensagens de *uplink*, sendo gerado os dados no sensor e enviados para o *gateway*, repassados para o servidor e posteriormente capturados pelo *IoT Agent* e mandados para o Dojot e assim podendo ser visualizados. Nos dados enviados nas mensagens de *downlink*, sendo iniciado o processo no Dojot e descendo a topologia até

o sensor, as informações foram visualizados no terminal através do programa PuTTY. Assim provando a eficácia do projeto implementado.

Para trabalhos futuros, sugere-se o uso de dados reais no sensor, bem como os testes para a eficácia de conexão e envio de informações a longa distância, assim como propõe a tecnologia LoRa. Neste projeto foi utilizado o *Cayenne* LPP, então, sugere-se usar todos os tipos de dados disponíveis do *payload*, bem como testar as questões envolvendo a segurança. Na parte do código do sensor, sugere-se o desenvolvimento de mais funções, como por exemplo, ascender as luzes de LED do dispositivo ao chegar determinada informação da camada superior.

Referências

ARAS, E. et al. Exploring the security vulnerabilities of lora. In: . [s.n.], 2017. p. 1–6. Disponível em: <https://www.researchgate.net/publication/318575428_Exploring_the_Security_Vulnerabilities_of_LoRa>. Citado 3 vezes nas páginas 14, 15 e 16.

CPQD. *Agente IoT*. 2017. <https://dojotdocs.readthedocs.io/pt_BR/latest/architecture.html#iot-agent>. Acesso em: 2019-11-28. Citado na página 10.

CPQD. *Architecture*. 2019. <https://dojotdocs.readthedocs.io/en/latest/architecture.html>. Acesso em: 2020-04-06. Citado 2 vezes nas páginas 4 e 23.

DOJOT. *Voce conhece a dojot?* 2017. <http://www.dojot.com.br/sobre-a-dojot-iot>. Acesso em: 2019-11-27. Citado 2 vezes nas páginas 10 e 22.

EMBARCADOS. *Introdução ao LPWAN*. 2017. <https://www.embarcados.com.br/introducao-ao-lpwan/>. Citado 2 vezes nas páginas 4 e 13.

FARRELL, E. S. Low-power wide area network (lpwan) overview. *IETF Internet Engineering Task Force*, 2018. ISSN 2070-1721. Disponível em: <https://tools.ietf.org/pdf/rfc8376.pdf>. Citado 3 vezes nas páginas 9, 10 e 14.

FREE SOFTWARE FOUNDATION. *The C Preprocessor*. 2001. <https://gcc.gnu.org/onlinedocs/gcc-2.95.3/cpp_1.html>. Acesso em: 2020-03-12. Citado na página 18.

GARG, N. *Apache Kafka*. [S.l.]: Packt Publishing, 2013. ISBN 1782167935, 9781782167938. Citado na página 19.

GITHUBDOJOT. *Dojot Repository*. 2017. <https://github.com/dojot>. Acesso em: 2019-11-28. Citado na página 10.

IBM. *Conhecendo o MQTT*. 2017. <https://www.ibm.com/developerworks/br/library/iot-mqtt-why-good-for-iot/index.html>. Acesso em: 2020-04-18. Citado na página 18.

JSON.ORG. *Introducing JSON*. 2002. <http://json.org/>. Acesso em: 2020-06-03. Citado na página 20.

LORA ALLIANCE. *A technical overview of LoRa® and LoRaWAN™*. 2015. <https://lora-alliance.org/sites/default/files/2018-04/what-is-lorawan.pdf>. Acesso em: 2020-02-04. Citado na página 14.

LORA ALLIANCE. *Full End-to-End Encryption For IoT Application Providers*. 2017. <https://lora-alliance.org/sites/default/files/2019-05/lorawan_security_whitepaper.pdf>. Acesso em: 2020-10-14. Citado 3 vezes nas páginas 4, 30 e 31.

LORA ALLIANCE. *LoRaWAN™Backend Interfaces1.03Specification*. 2017. <https://lora-alliance.org/sites/default/files/2018-04/lorawantm-backend-interfaces-v1.0.pdf>. Acesso em: 2020-02-11. Citado 3 vezes nas páginas 4, 15 e 16.

MIT. *PuTTY*. 2020. <https://www.putty.org/>. Acesso em: 2020-10-13. Citado na página 28.

MQTT. *FAQ - Frequently Asked Questions*. 2019. <http://mqtt.org/faq>. Acesso em: 2020-04-18. Citado na página 18.

MYDEVICES. *LoRa®*. 2017. <https://developers.mydevices.com/cayenne/docs/lora/#lora-cayenne-low-power-payload-overview>. Acesso em: 2020-02-13. Citado 3 vezes nas páginas 4, 16 e 17.

NIST. *Announcing the ADVANCED ENCRYPTION STANDARD (AES)*. 2001. <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf>. Acesso em: 2020-08-05. Citado na página 21.

RAZA, U.; KULKARNI, P.; SOORIYABANDARA, M. Low power wide area networks: An overview. *IEEE Communications Surveys Tutorials*, v. 19, n. 2, p. 855–873, Secondquarter 2017. ISSN 1553-877X. Citado na página 13.

RIBEIRO, A. F. *Estudo e Desenvolvimento de Mecanismo para Integração do LoRa Server ChirpStack com a Plataforma IoT Dojot*. 39 p. — Faculdade de Computação, Universidade Federal de Uberlândia, Uberlândia, 2019. Citado 5 vezes nas páginas 4, 10, 24, 31 e 32.

THE INTERNET SOCIETY. *The Base16, Base32, and Base64 Data Encodings*. 2006. <https://tools.ietf.org/html/rfc4648>. Acesso em: 2020-08-06. Citado 3 vezes nas páginas 5, 21 e 22.

VASHI, S. et al. Internet of things (iot): A vision, architectural elements, and security issues. In: *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*. [S.l.: s.n.], 2017. p. 492–496. Citado 3 vezes nas páginas 4, 9 e 12.

WIRESHARK FOUNDATION. *wireshark*. 2020. <https://www.wireshark.org/>. Acesso em: 2020-03-12. Citado na página 21.

WU, H.; ZHIHAO, S.; WOLTER, K. Performance prediction for the apache kafka messaging system. In: . [S.l.: s.n.], 2019. p. 154–161. Citado na página 19.