# PROTÓTIPO DO SISTEMA DE TELEMETRIA PARA APLICAÇÃO NO PROJETO DO FÓRMULA SAE ELÉTRICO

# TELEMETRY SYSTEM PROTOTYPE FOR APPLICATION IN THE ELECTRIC FORMULA SAE PROJECT

Erick Flávio de Arruda Galindo[1]
Roberto Kenji Hiramatsu[2]

RESUMO

A fórmula SAE elétrico é uma competição com o objetivo de proporcionar aos estudantes de engenharia a oportunidade de desenvolver na prática um projeto completo de um carro tipo fórmula. Um dos pilares importantes da competição está relacionado com o desempenho do carro em provas dinâmicas onde são observados aspectos referentes ao funcionamento total dos sistemas como aceleração, frenagem, aerodinâmica, segurança e partida do veículo. Com um papel importante, nota-se a necessidade da telemetria aplicada ao carro elétrico com o objetivo de extrair e monitorar dados em tempo real possibilitando à equipe maneiras de estudar e aprimorar os aspectos referentes ao funcionamento dos subsistemas. Neste contexto, o presente trabalho desenvolve um protótipo de sistema de telemetria focado na transmissão, armazenamento e apresentação de dados em tempo real utilizando os recursos disponíveis pela *internet of things* (IOT). Como resultado, obteve-se a comunicação entre dois dispositivos construídos com a interface de comunicação Lora e o micro controlador ESP 32, sendo, respectivamente, o emissor que será instalado no carro e receptor que ficará no box da competição. Os dados coletados e transmitidos pelo sistema de telemetria são armazenados em um banco de dados MongoDB e apresentados em tempo real em um dashboard construído no Node-Red. Os resultados obtidos atendem aos requisitos mínimos do sistema de telemetria que se resume na transmissão de informação entre o carro e um servidor de dados local abrindo um leque de possibilidades para novos estudos e melhorias do sistema de telemetria.

**Palavras-chave:** IOT; Telemetria; ESP 32; LoRa; Fórmula SAE.

SAE elétrico is a formula with the aim of providing engineering students with the opportunity to develop in practice a complete design of a formula car. One of the important pillars of the competition is related to the car's performance in dynamic tests where aspects related to the total functioning of the systems are observed, such as acceleration, braking, aerodynamics, safety and vehicle departure. With an important role, there is the need for telemetry applied to the electric car in order to extract and monitor data in real time, allowing the team ways to study and improve aspects related to the operation of the subsystems. In this context, the present work develops a prototype of a telemetry system focused on the transmission, storage and presentation of data in real time using the resources available through the internet of things (IOT). As a result, communication was obtained between two devices built with the Lora communication interface and the ESP 32 microcontroller, being, respectively, the transmitter that will be installed in the car and the receiver that will be in the competition box. The data collected and transmitted by the telemetry system is stored in a MongoDB database and presented in real time on a dashboard built in Node-Red. The results obtained meet the minimum requirements of the telemetry system, which is summarized in the transmission of

information between the car and a local data server, opening a range of possibilities for new studies and improvements of the telemetry system.

**Keywords:** IOT; Telemetry; ESP 32; LoRa; Fórmula SAE.

---

[1] Bacharelando em Engenharia Eletrônica pela Universidade Federal Rural de Pernambuco - Unidade Acadêmica do Cabo de Santo Agostinho.
[2] Doutor em Engenharia Elétrica pela Universidade de São Paulo.

## INTRODUÇÃO

Com o avanço da tecnologia e o crescente volume de dados gerados pelos sistemas digitais, o processo remoto de coleta e transmissão de informação se tornou cada vez mais importante devido aos benefícios gerados pela tomada de decisão em tempo real baseada na análise de dados. A telemetria, termo usado para o conjunto de recursos tecnológicos para medição remota, é utilizada por diversos segmentos como o industrial, médico, agrícola, aeroespacial, veicular, automobilístico e energético. Em cada um deles a telemetria provê benefícios na segurança, performance e redução de custos operacionais. Um dos segmentos que vem ganhando destaque é o automobilístico, que nos meados da década de 80 com a Fórmula 1 adotou sistemas de telemetria para otimizar o desempenho dos carros de competição[1]. Antes do uso da telemetria, as informações do veículo eram limitadas, resumidas e transmitidas para os operadores técnicos por comandos de voz do piloto via rádio, gerando assim diversos problemas como atenção do condutor dividida, latência na transmissão das informações, redução da confiabilidade das medidas manuais e interferência na comunicação devido ao compartilhamento do mesmo canal de comunicação[2,3]. Com a utilização da telemetria vinculada aos novos recursos da *internet of things* (IOT), surgiram diversas maneiras de construir sistemas de medição remota baseados, por exemplo, em interfaces de comunicação de baixo consumo de energia e de alto alcance. Este é o caso do LoRa que permite, segundo os dados de fabricação, uma transmissão em distâncias de até 15km de acordo com um conjunto de fatores locais e de configuração do dispositivo[4,5,6].

O LoRa é uma tecnologia de camada de rede física de radiofrequência que permite a comunicação em longas distâncias com um consumo mínimo de energia utilizando uma técnica de modulação conhecida como *chirp spread spectrum modulation*. A camada lógica de rede do Lora é conhecida como LoRaWAN que implementa todos os detalhes de funcionamento, segurança, qualidade e ajuste de potência visando maximizar a duração da bateria dos módulos. A arquitetura de rede básica utilizada por essa tecnologia se resume em quatro camadas bem definidas: *end-devices*, *gateways*, *network server* e *application server;* sendo cada uma delas responsáveis por uma parte do sistema. A camada dos *end-devices* representa os dispositivos básicos da rede responsáveis pela medição e interação com o mundo físico, suportando atuadores e transdutores em sua implementação. Já os elementos responsáveis pela comunicação entre os módulos e *network servers* são chamados de *gateways*, que podem manipular milhares de dispositivos de uma só vez e encaminhá-los para os servidores de rede. Os *network servers* são responsáveis pelo gerenciamento das informações enviadas pelo *gateway* fornecendo recursos de *hardware*, *software* e rede, servindo como um repositório central. Por fim, temos os *application servers* que são responsáveis por receber os dados dos *network servers* e executar

diversas ações específicas baseadas em um conjunto de configurações[6].

Para conseguir implementar projetos utilizando a rede LoRa existem algumas placas disponíveis no mercado com uma série de componentes integrados, como é o caso do LoRa Esp 32, que disponibiliza uma interface de comunicação LoRa integrada a um microcontrolador ESP 32 e uma tela de Oled, sendo uma opção ideal para novos protótipos com a rede LoRa. O ESP 32 é um microcontrolador rico em recursos de conectividade *Wi-Fi* e *Bluetooth* integrado a uma ampla gama de aplicativos capaz de funcionar de forma confiável em ambientes industriais com um consumo de energia ultra baixo[7] .

Pensando no contexto de conectividade entre microcontroladores com um baixo consumo de energia e uma taxa de comunicação leve entre máquinas e sensores, a IBM desenvolveu nos anos 90 o *Message Queuing Telemetry Transpor*t (MQTT) um protocolo de comunicação de mensagens para IOT. Ele possui uma arquitetura baseada em eventos com um transporte de mensagem no formato de publicações e assinaturas, tornando-o ideal para conectar dispositivos remotos com pouco espaço de código e banda de rede mínima. Sua aplicação envolve amplas áreas como indústrias, automotiva, manufatura, telecomunicações, petróleo e gás, entre outras[8]. Para conseguir implementar esse tipo de tecnologia, é necessário o uso de um servidor local ou remoto que servirá como *broker* para gerenciar os diversos dispositivos nomeados de clientes da aplicação. Os dispositivos clientes se escrevem em tópicos no *broker* e recebem todos os eventos referentes a ele, já nos atos de publicação, os clientes disparam eventos para a modificação de um tópico interno ao servidor *broker* e todos os clientes que assinaram esse tópico recebem a informação de mudança. Uma das grandes vantagens de utilizar os serviços do MQTT é a quantidade de programas que conseguem se conectar ao servidor e participar das interações, grande parte das linguagens e ferramentas hoje tem suporte a *application programming interface* (APIs) que permitem criar interações com um servidor MQTT. Este é o caso da ferramenta Node-RED que está sendo bastante utilizada na criação de aplicativos para redes IOTs[9] .

O Node-RED é uma ferramenta de programação para conectar dispositivos de *hardware*, APIs e serviços online de maneira inovadora[10]. A ferramenta fornece um editor de fluxo no navegador com uma paleta que oferece diversas funções para serem utilizadas na criação dos programas. Além disso, o aplicativo também fornece uma ampla biblioteca de funções que podem ser baixadas e utilizadas nos projetos de forma gratuita.

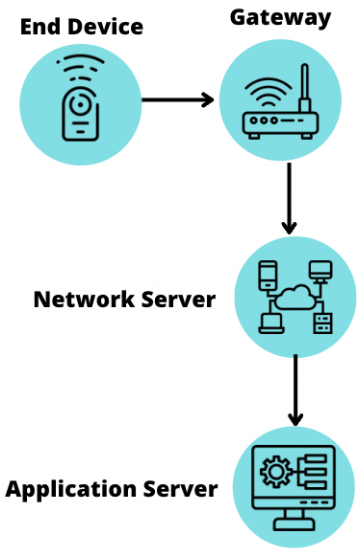
Nesse contexto, o presente trabalho realiza um estudo da telemetria aplicada ao projeto do fórmula SAE elétrico e descreve as etapas de implementação de um protótipo do sistema focado na transmissão, apresentação e armazenamento de dados.

O artigo está organizado em sete tópicos, sendo o primeiro uma breve abordagem das tecnologias utilizadas na construção do projeto, o segundo apresenta o modelo de arquitetura do sistema e as definições dos protocolos e padrões de mensagens, o terceiro e quarto abodam a construçãos dos dispositivos *end-device* e *gateway,* o quinto define o funcionamento do *network serve* e sua construção, o sexto apresentação a construção dos dashboards e as tecnologias envolvidas em seu funcionamento, o sétimo ultimo mostra um modelo de teste desenvolvido para avaliar a eficiência do sistema em um cenário específico. Por fim, é apresentado uma discussão sobre os resultados obtidos e futuras ideias para novas implementações.

## ARQUITETURA DO SISTEMA E DEFINIÇÕES DOS PROTOCOLOS E PADRÕES DE MENSAGENS

A construção do sistema de telemetria para o fórmula SAE elétrico se dividiu em quatro partes, seguindo a arquitetura de rede LoRa. Como primeira e segunda parte tem-se o desenvolvimento do módulo *end-Device* e *gateway*, com o primeiro sendo responsável pela transmissão das informações do carro e o segundo focado no recebimento e tratamento dessas informações. A terceira parte se refere à configuração do *network server* baseado no servidor *broker* (MQTT Server), responsável pela arquitetura baseada em eventos que é utilizada para integrar o módulo *gateway* ao servidor de aplicação. A quarta e última parte é responsável pela implementação do *application server,* que manipula a integração com o banco de dados MongoDB e a construção do *dashboard* para o acompanhamento das variáveis do sistema em tempo real. A Figura 1 apresenta uma visão geral da arquitetura do sistema de telemetria.

Figura 1 - Arquitetura do sistema de telemetria.




Fonte: Autor, 2022.

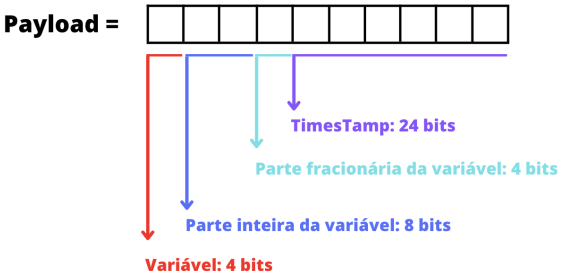
O dispositivo *end-device* será instalado no carro elétrico junto com o sistema de controle, já o *gateway*, *network server* e o *application server* ficaram no box da competição.

Como dispositivo base para a construção dos módulos *end-devices* e *gateway*, foi utilizado a placa LoRa 32 que possui as interfaces de comunicação *Wi-fi* e LoRa integradas diretamente ao módulo ESP 32. Esse dispositivo foi escolhido com base nos estudos realizados pela equipe do Fórmula SAE elétrico, que dentro do cenário atual de desenvolvimento do projeto, o tomou como sendo um dispositivo ideal para os novos protótipos de telemetria.

### Padrões de mensagens para o envio de dados

A estrutura do *payload,* que representa a carga de dados de transmissão entre os dispositivos, foi desenvolvida contendo três partes representadas com valores em hexadecimal, como apresentada na Figura 2.

Figura 2 - Estrutura do payload

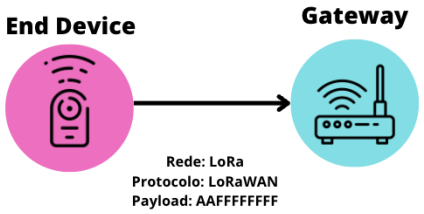



Fonte: Autor, 2022.

A primeira parte (destacada em vermelho na Figura 2) é responsável por indicar qual variável está sendo transmitida tendo um identificador único definido na estrutura do sistema, a segunda (destacada em azul cobalto na Figura 2) representa o valor da variável e a terceira contém um *timestamp* construído utilizando a função millis do ESP 32. Dessa forma, envia-se no *payload* um valor com 12 bits de dados,

sendo 8 bits representando a parte inteira e 4 a fracionária da variável. Para o *timestamp*, foi definido um tamanho de 24 bits, pois desta forma ele consegue representar um valor em milisegundos com tempo máximo de aproximadamente 297 minutos. A Figura 3 ilustra o padrão de comunicação, a rede e o protocolo de comunicação da interação entre o *end-device* e o *gateway.*

Figura 3 - Rede LoRa e o padrão de comunicação.




Fonte: Autor, 2022.

Os identificadores únicos para as variáveis que serão enviadas no sistema de telemetria foram padronizados conforme os seguintes dados: (A) Aceleração;(B) Velocidade; (C) Temperatura do motor; (D) Temperatura do pack de baterias.

## END-DEVICE

Esse módulo é responsável por receber e manipular um conjunto de informações que serão padronizadas e enviadas para o *gateway* seguindo um padrão de mensagem definido para o sistema.

### Materiais e programas usados para a construção do módulo

Para a construção e testes do módulo foi utilizada uma placa LoRa 32 e um cabo USB tipo C para a alimentação. A programação do dispositivo foi desenvolvida na IDE do Arduino utilizando os recursos das bibliotecas LoRa, SPI e Wire disponíveis para *download* na própria IDE do Arduino. Além disso, foi utilizada uma interrupção do ESP 32 associada ao Timer para o controle da frequência no envio de mensagens entre o *end-device* e o *gateway*.

### Método e configurações da rede LoRa

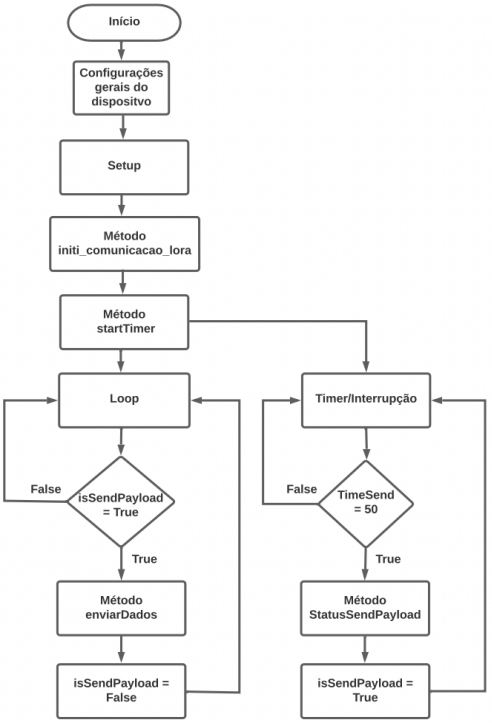
Como primeira parte do trabalho, foi definido um conjunto de configurações para o funcionamento da rede LoRa seguindo uma banda de frequência de 915 MHz e um ganho de 20 dBm. Logo após, foram desenvolvidos quatro métodos para o controle do dispositivo:

- *initi_comunicacao_lora:* Responsável por configurar e iniciar a comunicação de rede;
- *startTime:* Configura o timer que aciona uma interrupção a cada 1 segundo;
- *enviarDados:* Utiliza a informação de uma variável global e constrói um *payload* para ser enviado a o módulo *gateway*;
- *statusSendPayload*: Muda o valor de uma variável global chamada isSendPaylod habilitando o envio de informações para o *gateway*.

Quando o dispositivo é iniciado, conforme apresentado no Fluxograma 1, as configurações gerais são realizadas para definir os pinos do microcontrolador usados na rede LoRa, o ganho e a frequência de comunicação do dispositivo. Logo após, o método *initi_comunicacao_lora* é chamado para iniciar a comunicação da rede LoRaWAN com o dispositivo *gateway*. Em seguida, o método *startTime* é inicializado para configurar o *timer* e a interrupção que será usada para definir o tempo de envio de dados entre o *end-device* e o *gateway*. O intervalo de tempo mínimo entre envios foi definido em 50 milissegundos, devido a perda de pacotes medidos experimentalmente em intevalos de tempo inferiores. Quando o *timer* identifica o tempo configurado na variável global timeSend, ele ativa uma interrupção que

chama o método *statusSendPayload* mudando o valor da variável global isSendPayload para *true*. No *loop* uma estrutura de decisão ativa o método *enviarDados* caso o valor da variável isSendPayload seja *true*, e obtém os dados armazenados em uma variável global chamada dadosLoRaWAN construindo um *payload* para ser enviado a o dispositivo *gateway*.

Fluxograma 1: Funcionamento do End-Device

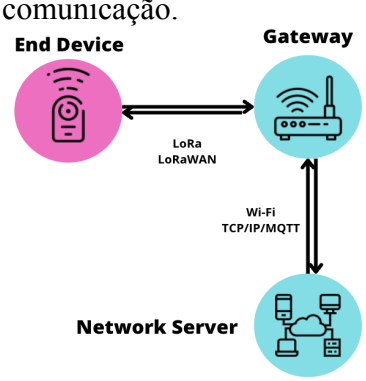



Fonte: Autor, 2022.

## GATEWAY

O *gateway* tem como função principal receber, tratar e enviar para o *network server* as informações transmitidas pelo módulo *end-device*. A conexão entre o *gateway* e o servidor de rede é feita utilizando o protocolo de comunicação MQTT, por meio de uma rede *Wi-Fi*. Já a comunicação com o módulo *end-device* é feita utilizando a rede LoRa com o protocolo LoRaWAN. Essa arquitetura é apresentada na Figura 4.

Figura 4 - Arquitetura e protocolos de comunicação.




Fonte: Autor, 2022.

A estrutura do sistema foi dividida em duas partes, configurações e métodos da rede LoRa, para o recebimento das mensagens do *end-device,* e configurações e métodos para comunicação *Wi-Fi* utilizando o protocolo de comunicação MQTT.

### Matérias e programas usados para a construção do módulo

Para a construção e testes do módulo foi utilizada uma placa LoRa 32 e um cabo USB tipo C para a alimentação. A programação do dispositivo foi desenvolvida na IDE do Arduino utilizando os recursos das bibliotecas Wifi, PubSubClient, Lora, SPI e Wire.

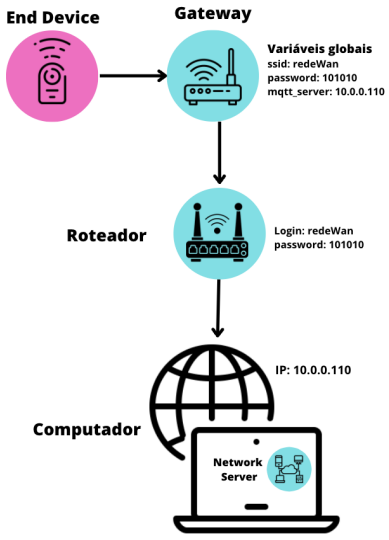
### Configurações da rede wi-fi e protocolo de comunicação MQTT

Para fazer a conexão com o *network server*, foi utilizado o módulo *Wi-Fi* do ESP 32 com o protocolo de comunicação MQTT. Para isso, foi usado um roteador que servirá de rede *Wi-Fi* para a comunicação entre o *gateway* e o *network server.*

Como entrada parametrizada para a configuração das credenciais do roteador, foi deixado na estrutura do *gateway* duas variáveis globais chamadas *ssid* e

*password*. Além disso, também foram definidas as configurações para o funcionamento do protocolo MQTT e a ligação com o servidor *broker* no *network server*. Para conseguir se conectar com o servidor *broker*, foi definido no *gateway* uma variável global chamada mqtt_server que armazena o IP do computador na rede *Wi-Fi* aonde o *network server* está configurado. Também foram configuradas quatro variáveis globais nomeadas de topicAceleracao, topicVelociade, topicTempMotor e topicTempBateira que armazenam o nome dos tópicos que o *gateway* irá se inscrever para conseguir enviar as mensagens ao servidor *broker*. A Figura 5 apresenta uma visão macro dos dispositivos e as configurações definidas.

Figura 5 - Dispositivos e configurações de rede.

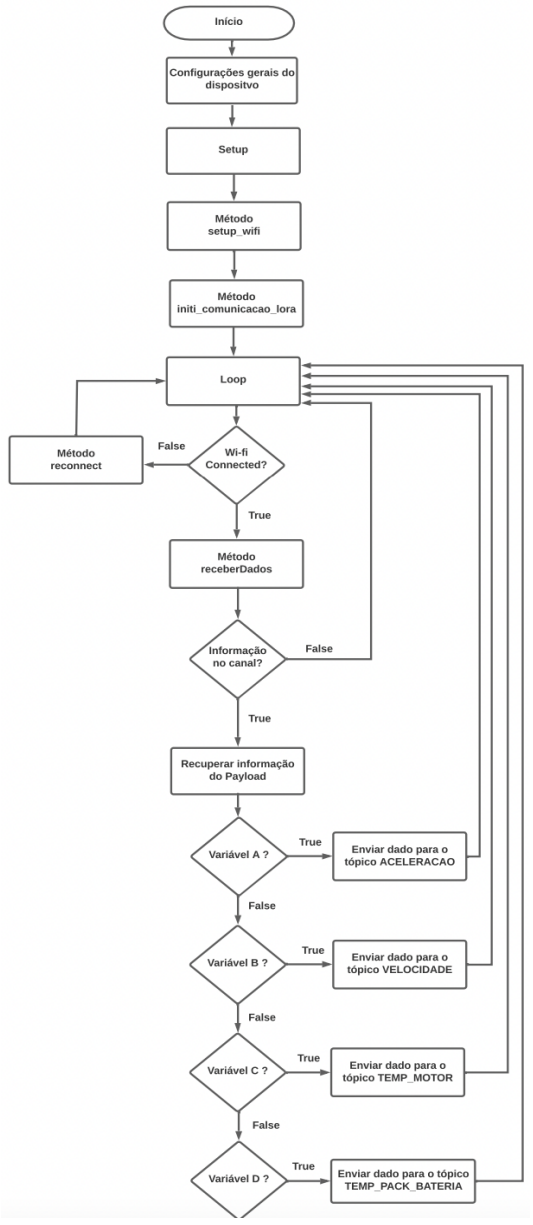



Fonte: Autor, 2022.

**Método da rede wi-fi**

Para conseguir se conectar com o roteador, foram construídos dois métodos chamados *setup_wifi* e *reconnect*. O primeiro é responsável por iniciar a conexão com o roteador passando o login e a senha armazenados nas variáveis globais ssid e password. O segundo é responsável por restabelecer a conexão com a rede caso ela seja perdida. O fluxo de funcionamento dos métodos da rede *Wi-fi* está sendo apresentado no Fluxograma 2.

Fluxograma 2 - Funcionamento do *gateway*.




Fonte: Autor, 2022.

**Método e configurações da rede LoRa**

Nas configurações da rede Lora, foram definidos uma banda de frequência de 915 MHz e um ganho de 20 dBm, seguindo o que foi definido para o *end-device*, já para o controle da rede

foram criados os métodos *initi_comunicacao_lora* responsável por configurar e iniciar a comunicação da rede LoRaWAN e o método *recebeDados*, que analisa o canal de comunicação esperando o envio de mensagens do *end-device.*

Conforme apresentado no Fluxograma 2, quando um dado é identificado pelo *gateway* no canal da rede LoRaWAN, o método *recebeDados* recupera as informações que estão na mensagem do canal e monta uma mensagem no formato J*avaScript Object Notation (*JSON) que é encaminhada ao tópico específico do servidor *broker* conforme o protocolo MQTT. A escolha do tópico é realizada pelo método observando a variável nos primeiros 4 bits representados em hexadecimal no *payload* enviado pelo *end-device*, conforme apresentado na Figura 2. A Figura 6 apresenta o formato do JSON que é montado e enviado para o *network server* pelo método *recebeDados*.

Figura 6 - Formato da mensagem de saída do gateway.

```
{"Valor":"","TimesTamp":"","RSSI":""}
```

Fonte: Autor, 2022.

As informações do campo Valor e Timestamp do JSON são preenchidas com os dados do *payload* recebido no canal da rede LoRaWAN enviados pelo *end-device*, conforme apresentado na figura 2. Sendo assim, é concatenado no campo Valor o trecho do *payload* que representa a parte inteira e decimal da variável e no campo TimesTamp, são colocados os 24 últimos bits que representam o *timestamp* do *payload*. No entanto, o valor do campo *Signal Strength Indicatio*n (RSSI) é preenchido utilizando o retorno da função packetRssi da biblioteca LoRa que indica a potência do sinal recebido em miliwatts.

## NETWORK SERVER

Esse módulo é composto por um computador com acesso à rede *Wi-Fi* que fornece recursos de *hardware* para suportar as conexões de rede e a execução do servidor *broker* MQTT. Para isso, foi escolhido uma máquina linux com o sistema operacional ubuntu na versão 18.04.5 LTS, no entanto, qualquer máquina que suporte alguma versão do linux fornecerá o mesmo desempenho para o projeto proposto. Além disso, foi instalado o servidor *broker* MQTT na versão 5.0/3.1.1/3.1 com as configurações padrões de usabilidade dos recursos.

## APPLICATION SERVER

Esse módulo é responsável por transformar, armazenar e apresentar em tempo real os dados recebidos do *network server* que foram enviados pelo *end-device*. O módulo foi dividido em quatro partes: configuração e instalação das ferramentas do Node-RED, node-red-dashboard e banco de dado MongoDB; construção do fluxo para manipulação e tratamento dos dados de evento do MQTT server; desenvolvimento dos *dashboards* para a apresentação das variáveis; modelagem do banco de dados do Fórmula SAE elétrico para armazenar os dados das variáveis transmitidas pelo sistema.

### Node-Red e banco de dados MongoDB

Como primeira parte do trabalho foi necessário instalar no computador o Node-RED versão v3.0.2, seguindo com a instalação da biblioteca node-red-dashboard na versão 3.1.7 e finalizando com a instalação do MongoDB versão v3.6.3.

### Nodes MQTT IN para entrada de dados node Node-Red

Para conseguir receber e tratar as informações enviadas pelo servidor MQTT

Broker, foi construído um fluxo no Node-Red com quatro *nodes input* chamados ACELERACAO, VELOCIDADE, TEMP_MOTOR e TEMP_PACK_BATERIA. Esses *nodes* são do tipo *mqtt in* que serve para conectar um node do fluxo a um MQTT *broker* e assinar mensagens de um tópico especificado. Dessa forma, o fluxo se conecta ao MQTT *broker* e se escreve em quatro tópicos, ficando ativo para receber qualquer tipo de evento associado aos tópicos assinados. Caso algum dado seja publicado em um desses quatro tópicos assinados pelo fluxo, ele receberá o evento com a informação publicada.

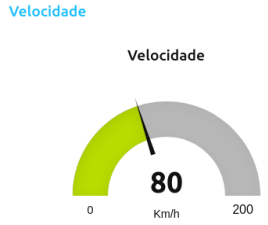
Nas propriedades de cada node, o server foi especificado como sendo a porta local do MQTT *broker* que por padrão está no localhost:1883, o action foi definido como *Subscribe to single topic* e o *output* como um tipo *String*.

**Construção dos dashboards**

Para a construção do *dashboard*, foi utilizada a biblioteca node-red-dashboard do Node-RED que possui um conjunto de *nodes* com diversos componentes para a montagem de painéis.

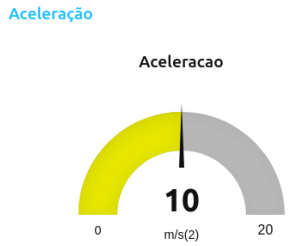
O *dashboard* foi desenvolvido para apresentar seis informações referentes ao sistema de telemetria, sendo quatro gráficos e duas informações de saída de texto. Para os gráficos foram utilizados quatro *nodes* do tipo gauge que montam um gráfico de medidor apresentando o valor de entrada dentro de um range mínimo e máximo. Os quatro gráficos se referem aos valores de aceleração, velocidade, temperatura do motor e *pack* de baterias enviados pelo módulo *end-device*. Já os dados de saída de texto, foram montados usando *nodes* do tipo text sendo um responsável por indicar o status do sistema mostrando conectado ou desconectado e o outro indicando a força do sinal recebido na comunicação LoRa entre o *end-device* e o *gateway*. As Figuras de 7 a 12 apresentam as partes do painel.

Figura 7 - Gráfico de velocidade



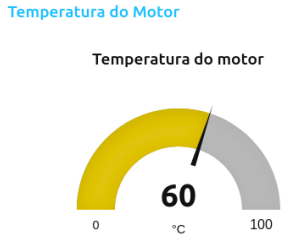

Autor, 2022.

Figura 8 - Gráfico da aceleração



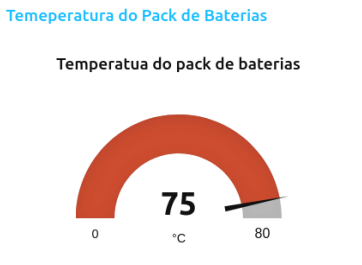

Autor, 2022.

Figura 9 - Gráfico de temperatura do motor




Autor, 2022.

Figura 10 - Gráfico de temperatura das baterias




Autor, 2022.

Figura 11 - Status de conexão

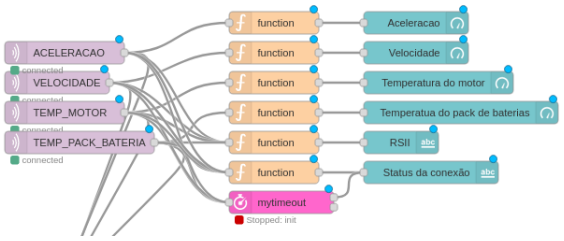

Autor, 2022.

Figura 12 - Força do sinal da rede Lora


Autor, 2022.

As informações dos gráficos gauge e da saída de texto *received signal strength indication* (RSSI), são recebidas nos nodes de entrada e encaminhadas para um node função aonde ocorre a conversão e junção da parte inteira e fracionária do valor da variável recebida na mensagem de entrada. Após essas operações a função manda um valor decimal para os nodes do dashboard conforme apresentado na Figura 13. O RSSI indica potência do sinal recebido em miliwatts e é medida em dBm indicando quão bem um receptor recebe a transmissão do emissor. Quanto mais próximo de zero o sinal é forte e quanto mais se aproxima de cento e vinte, o sinal fica fraco.

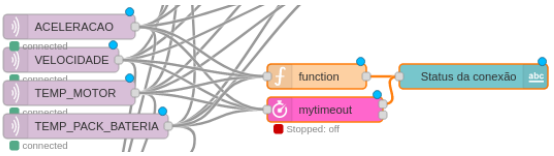
Figura 13 - Fluxo dashboard




Autor, 2022.

Já a saída de texto status de conexão ela é construída utilizando o node mytimeout da biblioteca node-red-contrib-mytimeout do Node-Red. Esse node funciona como uma espécie cronômetro que conta um tempo determinado em suas configurações e envia uma mensagem de saída no fim dessa contagem. Para conseguir identificar que a conexão com o gateway foi perdida, foi configurado um node *mytimeout* com um cronômetro regressivo de 5 segundos interligado aos nodes de entrada ACELERACAO, VELOCIDADE, TEMP_MOTOR e TEMP_PACK_BATERIA que reseta a contagem do cronômetro sempre que uma informação é publicada nesses nodes. Assim, todas as vezes que uma informação chega nos nodes de entrada o cronômetro regressivo é reiniciado, no entanto, caso tenha se passado 5 segundos da última informação publicadas nos nodes de entrada o mytimeout escreve desconectado na saída de texto status de conexão. A figura 14 apresenta o trecho do fluxo do Node-Red com as informações do tratamento do status de conexão.

Figura 14 - Fluxo com o cronômetro do status de conexão




Autor, 2022.

## Modelagem do banco de dados

Para o armazenamento das informações recebidas pelo sistema de telemetria foi utilizado o banco de dados MongoDB devido ao seu formato de armazenamento em documentos e a sua facilidade na conexão com o Node-Red. O banco de dados foi construído em uma estrutura com quatro coleções de tabelas cada uma contendo três colunas nomeadas

de variável, valor e *timestamp*. A coluna variável corresponde a o nome da variável do sistema que foi transmitida, o valor corresponde a informação transmitida e o *timestamp* representa o instante de tempo único que descreve o momento exato da medição da variável no sistema. Sendo assim, como o nosso sistema está trabalhando com as variáveis de aceleração, velocidade, temperatura do motor e pack de baterias, cada uma possui uma tabela para as suas coleções de valores. A Figura 15 mostra as quatro coleções do banco de dados BancoSAE nomeadas de ACELERACAO, TEMP_MOTOR, PEMT_PACK_BATERIAS e VELOCIDADE.

Figura 15 - Banco de dados do SAE e suas coleções.

```
> use BancoSAE
switched to db BancoSAE
> show collections
ACELERACAO
TEMP_MOTOR
TEMP_PACK_BATERIA
VELOCIDADE
```

Autor, 2022.

Além disso, na figura 16 é apresentado o formato dos registros armazenados nas coleções, tomando como exemplo os registros da tabela TEMP_MOTOR. O coluna _id é criada automaticamente pelo gerenciador do banco e armazena um tipo de *timestamp* para cada registro inserido no sistema.

Figura 16 - Formato dos registros armazenados nas coleções

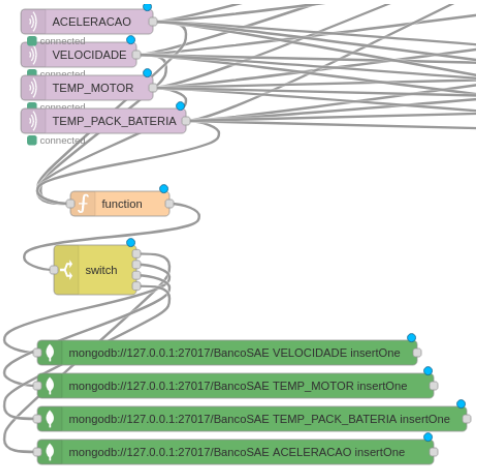
```
{ "_id" : ObjectId("62f007252e4031721ce6531a"),
 "Variavel" : "TEMP_MOTOR", "Valor" : "fff", "T
imestamp" : "fffff" }
```

Autor, 2022.

**Fluxo de insert no MongoDB**

Quando os dados são recebidos pelos *nodes* de entrada *mqqt in*, as informações são enviadas para um outro node de função responsável por formatar os dados de entrada para poderem ser armazenados no banco de dados. A função que formata os dados tem por objetivo converter e juntar na representação decimal os valores hexadecimais da parte inteira e fracionária da variável recebida na mensagem de entrada. Logo após, os dados formatados são passados para um node tipo *switch* que determina com base no nome da variável de entrada em qual tabela do banco de dados a informação será inserida. O trecho do fluxo com as ações de conversão, escolha da tabela e inserção no MongoDB está sendo apresentado na Figura 17.

Figura 17 - Fluxo inserte MongoDB




Autor, 2022.

## MODELO DE TESTE

Para conseguir testar o sistema de telemetria, foi desenvolvido um modelo que identifica a perda de pacotes na transmissão de dados entre o *end-device* e o *gateway*. O modelo consiste de um contador de pacotes no lado do *gateway* e um gerador de pacotes sequenciais no lado

do *end-device*. Os dados no gerador de pacotes são enviados sequencialmente com um valor começando de um, recebendo um incremento unitário a cada nova transmissão. No entanto, o contador de pacotes verifica se existe uma diferença entre o último valor recebido e o valor esperado a receber e soma a diferença da discrepância de contagem em uma variável de pacotes perdidos. Com isso, é determinada uma base da quantidade de pacotes perdidos na transmissão em relação a um total de pacotes enviados pelo sistema.

Como cenário de medição, foi escolhido o sexto andar do prédio da Unidade Acadêmica do Cabo de Santo Agostinho (UACSA) colocando os dispositivos a uma distância de 35m e uma altura 80cm do piso contendo uma barreira fïsica de bloco entre eles. Foram realizadas três medidas de 5 minutos obtendo uma taxa de perda de pacotes de 0,08%. O cálculo da perda de pacotes foi realizado dividindo o total de pacotes perdidos pelo total de pacotes enviados.

## DISCUSSÕES E RESULTADOS

Observando os aspectos da competição do fórmula SAE elétrico, é percebido a grande importância de um sistema de telemetria para o estudo das variáveis do sistema e a tomada de decisão baseada na análise de dados em tempo real, garantindo uma melhor performance do carro e um aumento na segurança do piloto nas competições. Diante disso, o presente trabalho realiza um estudo da telemetria aplicada ao projeto do fórmula e descreve as etapas de implementação de um protótipo do sistema focado na transmissão, apresentação e armazenamento de dados.

O projeto teve seu desenvolvimento concluído com êxito resultando em um protótipo de telemetria funcional, que permite o acompanhamento e análise das variáveis do carro em tempo real. Essa análise só poderá ser feita pela equipe do *box* em tempo real nas competições ou testes, ou posteriormente com o uso da base de dados do sistema. Além disso, foi implementado no projeto algumas tecnologias atuais da IoT que auxiliam a integração do sistema, proporcionando avanços e melhorias em diversos contextos do projeto, como o protocolo de comunicação MQTT para o desenvolvimento de novos aplicativos de controle para o sistema de telemetria.

Com a utilização do protocolo de comunicação MQTT vinculado ao MQTT server, é possível construir e integrar novos programas de forma mais fácil e eficiente, abrindo um leque de oportunidades para o desenvolvimento de novos programas utilizando os mesmos recursos atualmente desenvolvidos. Além disso, com o uso da ferramenta Node-Red , em que foram desenvolvidos os dashboards, pode-se explorar a construção de novos componentes gráficos para o sistema sem impactar no funcionamento dos que já existem, contando com uma grande diversidade de funções prontas nativas da ferramenta.

Atualmente os dados estão sendo armazenados de forma bruta no banco de dados e não foi desenvolvido nenhuma interface para a extração e análise de dados. Porém, como as informações estão sendo inseridas em uma base de dados local do MongoDB, é possível utilizar diversas ferramentas para fazer a análise de dados e construir modelos preditivos para o sistema e apresentar os resultados no mesmo *dashboard* desenvolvido para o projeto. Uma ferramenta de software livre acessível para essa análise é a biblioteca criada para a linguagem Python chamada Pandas, que oferece estruturas e operações para manipular tabelas numéricas e séries temporais.

O teste do sistema de telemetria foi realizado em um cenário simples para a obtenção dos resultados parciais de funcionamento. No entanto, é necessário a realização de novos testes aplicando

cenários mais reais ao da competição para uma análise mais detalhada e a obtenção dos possíveis problemas que possam ocorrer nesses cenários. Além disso, outras características funcionais podem ser implementadas no sistema em trabalhos futuros, como é o caso dos mecanismos de verificação de erros nas mensagens, controle de identificação das informações recebidas no *gateway* e um mecanismo de recuperação de perdas de pacotes de dados na transmissão.

## AGRADECIMENTOS

Agradeço primeiramente a Deus pela paz e oportunidade de poder estudar, aos meus pais Lucineide Silva de Arruda e Flávio Gomes Galindo, por todo apoio e cuidado nessa minha jornada.

Meus agradecimentos aos meus amigos Mário Lira, Paulo Carneiro e Robson Luiz por tornar meus dias na universidade mais agradáveis e felizes.

Agradeço aos diversos professores(as) que contribuíram para a minha formação, em especial aqueles com quem eu mais me identifiquei e cujos os ensinamentos se retiveram com mais clareza em minha mente, que são: Roberto Kenji, Rafael Alves, Elias Marques, Weliton Martins, Serginei Liberato.

Em especial, quero deixar meus sinceros agradecimentos ao meu amigo Robson Luiz, por todo incentivo, paciência e companheirismo nessa minha trajetória na universidade. Sem dúvidas, meu caminho teria sido bem mais difícil sem a sua ajuda diária em todos os momentos.

Por último, e não menos importante, agradeço às pessoas que acreditaram no meu potencial e me preparam para o ingresso na universidade, que são: Rosemberg Nascimento, Robson Lima e Rogério Lima. Essas pessoas tiveram uma grande influência em minha vida tornando real o meu sonho de se formar em uma universidade pública.

## REFERÊNCIAS

[1] MCLAREN. **A brief history of computing in Formula 1**. [*S. l.*]: Mclaren 2016. Disponível em: https://www.mclaren.com/racing/team/a-brief-history-of-computing-in-f1-1052199/. Acesso em: 2 set. 2022.

[2] CALDERÓN, Alfonso Gago; RUIZ, German Galbeño Galbeño; BOHÓRQUEZ, Alfonso Carlos Gago. GPRS telemetry system for high-efficiency electric competition vehicles. *In*: 2013 WORLD ELECTRIC VEHICLE SYMPOSIUM AND EXHIBITION (EVS27), 2013, Barcelona. **Anais [...]**. [*S. l.*]: IEEE, 2013. p. 1-7. Disponível em: https://www.doi.org/10.1109/EVS.2013.6914788. Acesso em: 2 set. 2022.

[3] SANTOS, Arthur Luis Viña dos. **Economia de combustível com o uso de telemetria para veículos de passeio.** 2016. Monografia (Bacharelado em Engenharia de Computação) – Universidade Federal do Rio Grande do Sul, Porto Alegre, 2016. Disponível em: https://www.lume.ufrgs.br/handle/10183/147672. Acesso em: 2 set. 2022.

[4] LOPES, Amanda Ingrid. **Estudo de área de cobertura de dispositivo LORA para ambientes de Smart Campus.** 2019. Monografia (Bacharelado em Engenharia de Computação e Automação) – Universidade Federal do Rio Grande do Norte, Natal, 2019. Disponível em: https://repositorio.ufrn.br/handle/123456789/43652. Acesso em: 2 set. 2022.

[5] SILVEIRA, Samir Michel Emanuel da. **Sistema de Sensores com Transmissão de Dados Utilizando Tecnologia de Radiofrequência LORA.** 2018. Trabalho de Conclusão de Curso (Bacharelado em Engenharia Elétrica) – Universidade Federal do Paraná, Curitiba, 2018. Disponível em: http://www.eletrica.ufpr.br/tcc/2018/2s/Sa

mir%20Michel%20Emanuel%20da%20Silveira/TCC_SAMIR.pdf. Acesso em: 2 set. 2022.

[6] SANTOS, Alexandro Vanderley dos. **Integração entre Dispositivos LoRa e Servidores de Aplicação Utilizando o Protocolo LoRaWAN.** 2021. Trabalho de Conclusão de Curso (Bacharelado em Engenharia Eletrônica) – Universidade Federal de Santa Catarina, Florianópolis, 2021. Disponível em: https://repositorio.ufsc.br/handle/123456789/223220. Acesso em: 2 set. 2022.

[7] ESP32-S2 Series Datasheet**:** SoC with Xtensa® SingleCore 32-bit LX7 Microprocessor
Supporting IEEE 802.11b/g/n (2.4 GHz Wi-Fi). **Espressif Systems**, [*s. l.*], 2023. Disponível em: https://www.espressif.com/sites/default/files/documentation/esp32-s2_datasheet_en.pdf. Acesso em: 2 set. 2022.

[8] STANFORD-CLARK, Andy; TRUONG, Hong Linh. **MQTT for sensor networks (MQTT-SN) protocol specification**: Version 1.2. International Business Machines Corporation, 2013. Disponível em: https://www.oasis-open.org/committees/download.php/66091/MQTT-SN_spec_v1.2.pdf. Acesso em: 2 set. 2022.

[9] NODE-RED: Low-code programming for event-driven applications. **OpenJS Foundation**, [*s. l.*], [2022]. Disponível em: https://nodered.org/. Acesso em: 2 set. 2022.

[10] SONI, Dipa; MAKWANA, Ashwin. A survey on mqtt: a protocol of internet of things (iot). *In*: INTERNATIONAL CONFERENCE ON TELECOMMUNICATION, POWER ANALYSIS AND COMPUTING TECHNIQUES, 2017, Chennai. **Anais [...]**. Chennai: Bharath Institute of Higher Education and Research, 2017, p. 173-177. Disponível em: https://www.researchgate.net/publication/316018571_A_SURVEY_ON_MQTT_A_PROTOCOL_OF_INTERNET_OF_THINGSIOT. Acesso em: 2 set. 2022.