# Avaliando o impacto da compressão de dados no desempenho e energia em redes LoRa

Javan Ataíde de Oliveira Júnior
*Programa de Pós-graduação em Ciência da Computação*
*Universidade Estadual do Oeste do Paraná*
Cascavel, Brasil
javanataide10@gmail.com

Marcio Seiji Oyamada
*Programa de Pós-graduação em Ciência da Computação*
*Universidade Estadual do Oeste do Paraná*
Cascavel, Brasil
marcio.oyamada@unioeste.br

***Resumo*—Os dispositivos utilizados em soluções de Internet of things (IoT), em sua maioria possuem grandes limitações principalmente relacionadas ao hardware e sua autonomia energética. Em grande parte dos casos o maior gasto energético está relacionado à comunicação. Na literatura são encontrados vários métodos para otimizar a energia consumida com a comunicação, entre estes métodos estão as abordagens de compressão de dados. A maioria dos algoritmos de compressão são projetados para serem executados em computadores pessoais e por isso muitas vezes necessitam ser adaptados ao contexto IoT, tendo que lidar com limitações como memória e tempo de execução. Este trabalho analisa métricas de desempenho e energia de dois algoritmos clássicos de compressão de dados, Huffman e LZ77, que foram adaptados para execução em dispositivos IoT. A compressão foi utilizada na transmissão de pacotes via modulação LoRa, avaliando o compromisso entre taxa de compressão e energia consumida. Os resultados obtidos no estudo de caso utilizando dados reais de uma aplicação IoT na área da agricultura mostraram que a compressão Huffman resultou em um melhor compromisso, entre taxa de compressão e energia consumida, reduzindo em até 17% o consumo de energia do dispositivo.**

***Index Terms*—Internet of things, modulação LoRa, compressão de dados**

## I. Introdução

Devido ao avanço dos sistemas de comunicação, nanociência, eletrônica e tecnologia de sensores a *Internet of Things* (IoT) vem ganhando destaque, tornando-se cada vez mais utilizada [1], [2]. A IoT consiste em um conjunto de ”coisas”conectadas a uma rede heterogênea [2]; essa rede é o que denomina-se de *wireless sensor network* (WSN).

O fator energia é um dos principais limitantes na IoT e na maioria dos casos, o rádio utilizado para comunicação é um dos principais consumidores. Em alguns dispositivos o custo energético da comunicação chega a $60\%$ do gasto energético do dispositivo [3]. Estima-se que o custo energético para enviar um único bit equivale a mil operações realizadas no processador [4]. Devido a essas grandes proporções de energia, algumas abordagens são propostas para diminuir o fluxo de dados enviados, entre elas está a compressão de dados. No entanto, a maioria dos algoritmos de compressão são voltados à taxa de compressão, não ao gasto de energia ou memória [3], que são fatores limitados em dispositivos IoT. Devido a essas limitações, boa parte dos algoritmos de compressão precisam ser adaptados, sendo necessário na maioria dos casos desenvolver algoritmos especificamente para o dispositivo IoT [3] [5].

O objetivo desse trabalho é verificar a possibilidade de trocar transmissão por computação na borda[1], de modo à otimizar a eficiência energética e redução no número de pacotes transmitidos utilizando compressão de dados. Um estudo de caso é realizado utilizando um nó ESP32 e rede LoRa para enviar mensagens de texto de uma aplicação de monitoramento no domínio da agricultura. Os métodos de compressão utilizados são os algoritmos de Huffman e LZ77 que foram adaptados para execução em dispositivos com pouca memória e também considerando o tamanho da mensagem suportada pelo protocolo LoRaWan.

Este trabalho está organizado da seguinte forma: A Seção II apresenta os trabalhos relacionados, Seção III descreve a modulação LoRa, Seção IV descreve os métodos de compressão Huffman e LZ77, enquanto que a Seção V apresenta a metodologia. A Seção VI e VII apresentam os resultados, as conclusões e trabalhos futuros respectivamente.

## II. Trabalhos relacionados

Apesar da compressão de dados ser um campo de pesquisa no estado da arte, quando direcionada a IoT, abre espaço para novas técnicas voltadas as métricas de interesse da mesma [3]. Devido as limitações dos dispositivos, novos algoritmos para compressão foram propostos ou adaptados para tornar viável sua utilização [3].

Maurya et al. [5] propõem um algoritmo de compressão sem perdas, visando reduzir o custo computacional e o gasto de armazenamento de memoria, considerando que os dados sejam armazenados no dispositivo. Segundo Maurya et al. [5], parte da literatura é voltada a diminuir o tempo de processamento e a ocupação do canal. Eles propõem o *Median Predictor based Data Compression* (MPDC), que utiliza como codificador a codificação de Huffman estática. O MPDC visa ter uma alta taxa de compressão para dados correlacionados e uma boa taxa para não correlacionados. O mesmo consegue taxas de até $67\%$ de compressão. Apesar dos autores informarem que há

[1]Computação na borda refere-se a utilizar recursos computacionais dos dispositivos para processar os dados na borda em vez de transmiti-los massivamente, a mesma pode melhorar questões como tempo de resposta e economia de energia [6], [7]

uma redução do gasto energético, o valor dessa redução não é quantificado, bem como o gasto de memória. Adicionalmente, os dados se restringem à simulações.

Sacaleanu et al. [8] realizam a compressão de dados utilizando uma adaptação da codificação de Huffman estática, além de explorar a correlação temporal e agregação de dados. Os autores tem como objetivo realizar a comparação do custo energético para a transmissão desses dados comprimidos em três diferentes redes de transmissão ZigBee, Enhanced ShockBurst e LoRa. O mesmo desconsidera o custo energético do processador, quando comparado ao custo de transmissão. De acordo com Sacaleanu et al. [8], analisando os custos com e sem compressão, na rede Lora, os ganhos foram de até $31\%$ de economia de energia. Todavia, o trabalho não mensurou dados como tempo de processamento ou cenários com dados não correlacionados, bem como o gasto de memória.

Marceloni e Vecchio. [4] propõem um esquema de compressão sem perdas (*lossless entropy compression*- LEC), que explora a correlação existente entre dados de um determinado dispositivo, considerando que a variação entre os sinais são pequenas. A proposta do algoritmo é que o mesmo possua baixo custo computacional e que o dicionário seja pequeno e com o tamanho fixo de acordo com a resolução do ADC. Os autores argumentam que os algoritmos clássicos são praticamente inviáveis de reproduzir na maioria dos dispositivos IoT, além disso, os algoritmos sem perda propostos recentemente apesar de possuir boas taxas de compressão, requerem recursos computacionais consideráveis. O compressor LEC foi aplicado a dados correlacionados de temperatura e umidade e comparado a alguns algoritmos de compressão (Gzip, Bzip2, Rar, Huffman e Aritimético) chegando a atingir $70\%$ e $62\%$ de taxa de compressão. O mesmo também aplicou o algoritmo a dados que possuem grandes variações onde teoricamente não teria bons resultados, porém, conseguiu atingir taxas de compressão próxima ou superiores aos outros métodos, com taxas próximas a $70\%$, mostrando desse modo a validade e confiabilidade do LEC. Apesar dos bons resultados, o trabalho focou mais na comparação da taxa de compressão, medindo o consumo energético por meio de fórmulas, bem como não se aprofundando em métricas como memória.

De acordo com Byl et al. [9] existe uma relação entre a energia computacional usada para compressão e a economia de energia associada a redução de pacotes enviados. Segundo os autores, o tempo de processamento da compressão de dados deve ser levado em conta, visto que um tempo de processamento relativamente alto, pode cancelar quaisquer ganhos provindos da compressão. Baseado nessa prerrogativa, foi analisado o comportamento de métodos de compressão sem perdas (Huffman e RLE) e com perdas (Transformada discreta de Wavelet) utilizando tanto dados correlacionado, quanto dados não correlacionados, além de proporem um modelo híbrido de compressão com perda. Segundo os autores, uma das desvantagens de aplicar compressão sem perdas é a imprecisão sobre a taxa de compressão, com poucas variações entre os dados, a compressão sem perda mantém uma taxa quase constante, porém para grandes variações a taxa varia significativamente. De acordo com o trabalho, os modelos de compressão sem perdas apresentam melhores resultados quando os dados são correlacionados e os sem perdas quando os dados possuem grandes flutuações. O modelo hibrido proposto atingiu taxa de compressão fixa de $6,5:1$, além disso, o dispositivo conseguiu aumentar o tempo de duração de bateria 3,4 vezes quando comparada a operação sem compressão. Apesar do trabalho avaliar a questão energética, o mesmo focou a questão apenas nos algoritmos com perda, além de não explorar valores como consumo de memória.

Sadle e Martonosi. [3] propuseram um método de compressão voltada a IoT em sistemas tolerantes a atraso, o *Sensor LZW* (S-LZW) e algumas variações do mesmo. O foco principal do trabalho foi compressão de dados para dispositivos com energia limitada. O autor analisou o custo de memória RAM utilizando o algoritmo proposto com outros algoritmos de compressão de dados utilizados normalmente em computadores pessoais. Tais algoritmos em sua maioria não são possíveis de serem utilizados diretamente em alguns dispositivos de IoT. Os autores também avaliaram a taxa e tempo de compressão para o algoritmo proposto bem como para suas variações. Os ganhos de eficiência energética foram mensurados, porém apenas para o algoritmo proposto e suas derivações, não comparando com os métodos clássicos de compressão. O experimento foi realizado com três rádios diferentes e quatro *benchmarks*. Segundo os autores chegou-se a ter a redução de um fator de 1,5 de energia utilizando o método proposto.

O presente trabalho é similar ao proposto por [8], utilizando uma rede LoRa. No entanto, ao invés de comparar o ganho energético entre diferentes tecnologias de redes de sensores, será voltado ao ganho obtido pela utilização de compressão de dados. Neste trabalho será utilizado a compressão de Huffman, que foi base para os trabalhos [4], [5] e [8]. No entanto, outras métricas além da taxa de compressão são exploradas como: tempo de execução, gasto de memória (heap), ganho energético global (considerando o uso em um cenário real), ganho energético entre o métodos (analisando apenas a energia gasta para a compressão). Adicionalmente, neste trabalho será analisado o LZ77, um método de compressão por dicionário.

## III. Modulação LoRa

A modulação LoRa é uma tecnologia emergente, pertencente a Semtech e vem ganhando espaço na IoT. A mesma possui como técnica de modulação uma tecnologia derivada da modulação *Chirp Spread Spectrum*(CSS) [10], [11], possibilitando vantagens como: longo alcance, robustez e baixo consumo [11].

Os dispositivos que utilizam a modulação LoRa possuem quatro parâmetros principais, que são: fator de espalhamento (*spreading factors*, SF), largura de banda (BW), frequência de portadora e taxa de codificação (*coding rate*) [11]. A modulação LoRa possibilita a troca de *data rate* por uma maior sensibilidade dentro da largura de banda fixa do canal, fornecendo uma taxa de dados variável apenas alterando o fator de espalhamento [12]. Um SF mais alto aumenta a

relação sinal ruído (SNR), com isso aumenta-se a robustez e o alcance do sinal, todavia, aumenta-se o tempo no ar e consequentemente gasto energético para a transmissão [11].

### A. LoRaWAN: Protocolo

A modulação LoRa (camada física) pode ser utilizada com outras camadas MAC. No entanto, a camada MAC mais utilizada é a LoRaWAN, que opera em topologia estrela simples, composta por nós e *gateways*. A comunicação entre os nós e *gateway* na maioria das vezes é bi-direcional, a mesma opera em frequências distintas, além de possuir um *data rate* de $0.3$ kbps à $50$ kbps, o qual é escolhido conforme a relação entre distância e tempo de duração das mensagens [13].

A modulação LoRa opera em frequências da ISM ( *industrial, scientific and medical*) *band*, que são frequências destinadas a fins científicos, industriais e médico [11]. A LoRaWAN Alliance fornece tabelas com os valores dos parâmetros dos dispositivos LoRa de acordo com as normas regionais pré-estabelecidas do país a ser utilizada, a mesma fornece os planos de canais (frequência de operações) e valores como as janelas de atraso e tamanho da mensagem de acordo com os parâmetros escolhidos. No Brasil, os dispositivos podem operar nas frequências entre 902 à 907.5 MHz, 915 à 928 MHz e 433 à 435 MHz; o Brasil portanto pode operar nas regiões definidas como: AU915-928 ou EU433 [14].

Os dispositivos nesse trabalho, seguem os parâmetros da região AU915, utilizando 64 canais de *UpLink* com largura de banda de 125 KHz, intercalados de 200 KHz, iniciando em 902.5 MHz a 927.8 MHz, operando com CR de 4/5 e podendo utilizar SF de 12 a 7 com um *payload* de 51 à 222 *bytes* respectivamente.

## IV. Compressão de dados

A compressão pode ser definida como a ciência de representar informação de forma compacta e são criadas reconhecendo e identificando estruturas existentes nos dados. Devido ao modo o qual as informações são, em sua maioria, atualmente geradas em formato digital [15], o objetivo principal da compressão tornou-se codificar dados usando a menor quantidade possível de símbolos no menor número de bits possível [16]. Deste modo as técnicas de compressão ou algoritmos de compressão podem ser divididas em duas classes, com perdas (*lossy*) e sem perdas (*lossless*).

As técnicas de compressão com perda admitem uma certa quantidade de perdas em troca de uma compressão maior e são mais utilizadas em imagens gráficas e voz digitalizada. Nessas técnicas, o dado a ser comprimido será diferente depois da descompressão. Devido a isto, os métodos existentes tentam equilibrar a distorção provinda da compressão com a capacidade de comprimir, possibilitando obter uma maior taxa de compressão se comparada com as técnicas sem perda. Já as técnica de compressão sem perdas, consistem em gerar um fluxo de dados idênticos na descompressão ao dado comprimido; são geralmente utilizadas em registros de banco de dados, planilhas e arquivos de processamento de textos, onde perdas mesmo de um bit, pode gerar grandes erros [17]. Como este trabalho está voltado à compressão de mensagens de texto, apenas os compressores sem perda serão discutidos.

### A. Codificação de Huffman

Os codificadores tem como objetivo representar uma determinada quantidade de informação de maneira mais otimizada possível, umas das primeiras abordagens de codificadores foram as de Shannon-Fano e Huffman, ambas são maneiras diferentes de gerar códigos de comprimento variável a partir de uma tabela de probabilidades, portanto as mesmas são baseadas em modelos estatísticos [17].

A técnica de codificação de Huffman baseia-se nas frequências de ocorrência dos símbolos em um conjunto de dados para construir as *codewords* de tamanho variável e com o menor comprimento médio para os símbolos, além disso, as *codewords* geradas são do tipo prefixadas o que garante a decodibilidade dos dados [15], [18]. A codificação de Huffman define que os símbolos que tiverem as maiores probabilidades de ocorrer deverão ter *codewords* de tamanho menor do que as com menor probabilidade. Como exemplo, considerando uma fonte de dados a ser codificada composta pelo alfabeto: $A = [a_1, a_2, a_3, a_4, a_5]$ com as seguintes probabilidades $P(a_1) = P(a_3) = 0,2$ , $P(a_2) = 0,4$ e $P(a_4) = P(a_5) = 0,1$. A primeira etapa para realizar a codificação é ordenar os símbolos de acordo com a probabilidade, como ilustra a Tabela I [15].

Tabela I: Fonte de dados:$A$

| Símbolo | Probabilidade | *Codeword* |
|---|---|---|
| $a_2$ | 0.4 | c($a_2$) |
| $a_1$ | 0.2 | c($a_1$) |
| $a_3$ | 0.2 | c($a_3$) |
| $a_4$ | 0.1 | c($a_4$) |
| $a_5$ | 0.1 | c($a_5$) |

Fonte: Adaptado de [15]

Para este alfabeto os símbolos com menor probabilidade são $a_4$ e $a_5$, portanto para eles pode ser atribuído as seguintes *codewords*:

$$c(a_4) = \alpha_1 * 0$$

$$c(a_5) = \alpha_1 * 1$$

Deste modo um novo alfabeto A' é definido composto por $[a_1, a_2, a_3, a'_4]$ onde $a_4$ é composto por $a_4$ e $a_5$ e portanto a probabilidade $P(a'_4) = P(a_4) + P(a_5) = 0.2$, novamente os símbolos são ordenados como mostra a Tabela II.

Tabela II: Fonte de dados:$A'$

| Símbolo | Probabilidade | *Codeword* |
|---|---|---|
| $a_2$ | 0.4 | c($a_2$) |
| $a_1$ | 0.2 | c($a_1$) |
| $a_3$ | 0.2 | c($a_3$) |
| $a'_4$ | 0.2 | $\alpha_1$ |

Fonte: Adaptado de [15]

Neste alfabeto agora os símbolos são $a_3$ e $a'_4$, portanto para eles pode ser atribuído as seguintes *codewords*:

$$c(a_3) = \alpha_2 * 0$$

$$c(a'_4) = \alpha_2 * 1$$

porém como $c(a'_4) = \alpha_1$, $\alpha_1 = \alpha_2 * 1$, portanto:

$$c(a_4) = \alpha_2 * 10$$

$$c(a_5) = \alpha_2 * 11$$

Novamente um novo alfabeto $A''$ é gerado composto por três símbolos $[a_1, a_2, a'_3]$ onde $a'_3$ é composto por $a_3$ e $a'_4$ e tem como probabilidade $P(a'_3) = P(a_3) + P(a'_4) = 0.4$ e é novamente ordenado como apresenta a Tabela III.

Tabela III: Fonte de dados: $A''$

| Símbolo | Probabilidade | *Codeword* |
|---|---|---|
| $a_2$ | 0.4 | c($a_2$) |
| $a'_3$ | 0.4 | $\alpha_2$ |
| $a_1$ | 0.2 | c($a_1$) |

Fonte: Adaptado de [15]

Para este alfabeto os símbolos com menor probabilidade são $a_1$ e $a'_3$, portanto para eles pode ser atribuído as seguintes *codewords*:

$$c(a'_3) = \alpha_3 * 0$$

$$c(a_1) = \alpha_3 * 1$$

Portanto:

$$c(a_3) = \alpha_3 * 00$$

$$c(a_4) = \alpha_3 * 010$$

$$c(a_5) = \alpha_3 * 011$$

Novamente um novo alfabeto $A'''$ é gerado composto por dois símbolos $[a''_3, a_2]$ onde $a''_3$ é composto por $a'_3$ e $a'_1$ e tem como probabilidade $P(a''_3) = P(a'_3) + P(a_1) = 0.6$ e é novamente ordenado como apresenta a Tabela IV.

Tabela IV: Fonte de dados: $A'''$

| Símbolo | Probabilidade | *Codeword* |
|---|---|---|
| $a''_3$ | 0.6 | $\alpha_3$ |
| $a_2$ | 0.4 | c($a_4$) |

Fonte:Adaptado de [15]

Como restam apenas dois símbolos, as *codewords* podem ser atribuídas de forma direta:

$$c(a''_3) = 0$$

$$c(a_2) = 1$$

o que significa que $\alpha_3 = 0$, logo :

$$c(a_1) = 01$$

$$c(a_3) = 000$$

$$c(a_4) = 0010$$

$$c(a_5) = 0011$$

Tabela V: Codificação de Huffman para o alfabeto : $A$

| Símbolo | Probabilidade | *Codeword* |
|---|---|---|
| $a_2$ | 0.4 | 1 |
| $a_1$ | 0.2 | 01 |
| $a_3$ | 0.2 | 000 |
| $a_4$ | 0.1 | 0010 |
| $a_5$ | 0.1 | 0011 |

Fonte: Adaptado de [15]

A Tabela V representa a codificação de Huffman para o alfabeto original com os cinco símbolos, porém a mesma também pode ser representada por um árvore binária.

A codificação de Huffman requer o conhecimento das probabilidades da fonte para seu processo. A forma na qual este conhecimento é disponibilizado pode ser dividido em duas abordagens: estática ou adaptativa.

A estática tem como característica utilizar uma tabela de probabilidade da fonte fixa, onde tanto o codificador e o decodificador possuem conhecimento dela. Uma das vantagens é a necessidade de construir apenas uma vez a tabela/árvore de Huffman, além de não ser necessário enviar a tabela de probabilidades junto à mensagem a codificada. Porém uma das desvantagens é que a mesma pode não representar bem o dado a ser codificado levando a degradação na taxa de compressão [17].

A adaptativa calcula a distribuição de frequências da mensagem a ser comprimida, de modo a calcular as probabilidades dos símbolos ocorrerem. Uma das vantagens é possibilidade de se obter uma melhor taxa de compressão. No entanto, é necessário calcular às probabilidades, construir a tabela/árvore de Huffman, além de ser necessário enviar a tabela de probabilidades na mensagem, de modo a possibilitar a decodificação. Portanto, apesar de melhorar a taxa de compressão, aumenta-se o tempo para a codificação, custo computacional e adiciona um custo na mensagem a ser transmitida [17].

### *B. Codificação Lempel-Ziv*

Apesar da codificação de comprimento variável como Huffman possuir um bom desempenho, a mesma está limitada à entropia da fonte o que impulsionou a busca por novas maneiras comprimir os dados, dando origem as técnicas compressão por dicionário, que podem ser estáticas ou dinâmicas. Essas técnicas são mais eficientes em fontes que possuem um número pequeno de padrões com bastante frequência, como fontes de texto e comandos de computador [18].

As técnicas estáticas são mais adequadas quando possui-se um conhecimento prévio da fonte à ser comprimida, nesse tipo de situação é altamente eficiente projetar um esquema de compressão baseado em um dicionário estático contendo os padrões recorrentes, também conhecidos como *tokens*. Para aplicações genéricas as técnicas de dicionário adaptativas são mais adequadas e são caracterizadas por se adaptarem a característica da fonte para gerar o dicionário [15].

Uma das principais questões das técnicas baseadas em dicionário é como determinar quais são os melhores *tokens*, aos quais a combinação resulta na menor entropia da fonte.

Um dos métodos existentes de *tokenization* é o de força bruta o qual tenta combinar todos os símbolos e encontrar padrões de modo a reduzir a entropia, no entanto, além de exigir muita memória, leva muito tempo para realizar a operação, não sendo recomendado para nenhum tipo de processamento em tempo real [18].

Visando resolver o problema da *tokenization* ideal, Jacob Ziv e Abraham Lempel criaram duas soluções para o problema, essas soluções foram publicadas em 1977 e 1978 e atualmente são base para a maioria das abordagens de dicionário adaptativo [18]. As abordagens baseadas no trabalho de 1977 são pertencentes a família $LZ77$, enquanto as de 1978 são pertencentes a família $LZ78$ [15]. O $LZ77$ e $LZ78$ levaram a criação de outros algorimtos como o Lempel–Ziv–Welch (LZW) que é utilizado no formato GIF e também foram base para outro formatos como PNG, PKZIP, GZIP e ZLIB [18].

*1) LZ77:* Na abordagem do algoritmo LZ77, o codificador funciona examinando uma sequência de entrada por meio de uma janela deslizante. Essa janela é composta por um *buffer* de pesquisa (BP), a qual contém a sequência já codificada e um *buffer* de antecipação (BA), que contém a sequência a ser codificada como ilustra a Figura 1.

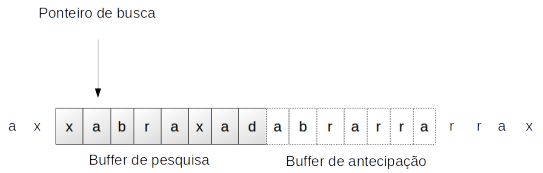


Figura 1: Codificação utilizando a abordagem do LZ77

Fonte:Adaptado de [15]

Para realizar a codificação da sequência que está no *buffer* de antecipação, há um ponteiro de busca que é movido no *buffer* de pesquisa até encontrar o primeiro símbolo que coincida com o primeiro símbolo do *buffer* de antecipação, a distância entre o ponteiro e o *buffer* de antecipação é conhecida como deslocamento. Após encontrar o símbolo coincidente, o codificador examina se o símbolo a seguir do local do ponteiro corresponde ao símbolo consecutivo do *buffer* de antecipação. A quantidade de símbolos consecutivos no *buffer* de pesquisa que coincide como os símbolos *buffer* de antecipação é conhecido como comprimento de coincidência. O codificador refaz o processo de modo a encontrar o maior comprimento. Após encontrar o maior comprimento, o codificador realiza a codificação com o seguinte formato $[o, l, c]$, onde $o$ é o deslocamento, $l$ o comprimento e $c$ o símbolo subsequente à *string* de coincidência. Por exemplo, na Figura 1 o valor do deslocamento seria 7, o comprimento 4 e o símbolo subsequente a *string* seria r. Quando não há nenhuma correspondência do símbolo no *buffer* de pesquisa, o valor de deslocamento e comprimento são zero [15].

Pode-se observar que o LZ77 é um esquema adaptativo simples e que não necessita de conhecimento prévio da fonte para realizar a compressão. Segundo os autores do algoritmo, o desempenho do mesmo se aproxima da melhor forma possível dos que possuem conhecimento pleno sobre as estatísticas da fonte de modo assintótico, todavia há modos de melhorar o LZ77 que derivam outros algoritmos. Entre as melhorias possíveis seria assumir que a tríplice de codificação seja codificada de tamanho variável ou variar o tamanho dos *buffers* possibilitando encontrar a *string* no dicionário. No entanto, realizar esse aumento requer o desenvolvimento de metodologias para otimizar a busca no buffer de pesquisa [15].

## V. METODOLOGIA

O estudo foi realizado utilizando um dispositivo com a plataforma ESP32 conectado a um rádio LoRa via SPI, disponibilizado pela empresa Trinovati Tecnologia, que também disponibilizou os dados a serem transmitidos. Tais dados foram obtidos de um cenário real aplicado na agricultura. Os métodos de compressão adotados foram o de Huffman adaptativo e LZ77.

Na compressão baseada no algoritmo de Huffman a mensagem codificada é agrupada em *bytes*, uma vez uma *codeword* pode ser menor ou maior que um *byte*, baseado nisso, adiciona-se um *byte* N que indica o número de bits que devem ser considerados no último *byte* da codificação. Considerando o número pequeno de mensagens que poderiam ser codificadas, a tabela de probabilidade é do tipo byte, o que reduz o tamanho ocupado na mensagem. As mensagens possuem 13 caracteres diferentes nos dados, portanto a tabela de probabilidades ficou restrita à 13 *bytes*. A Figura 2 apresenta a estrutura da mensagem a ser enviada.

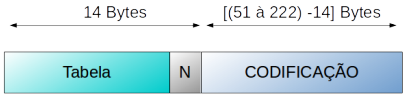


Figura 2: Estrutura da mensagem de Huffman

Para a compressão LZ77 utilizou-se como tamanho do BP (buffer de procura) e BA(buffer de antecipação) de 200 e 80 *bytes* respectivamente. A Figura 3 apresenta a estrutura resultante da compressão. Para o LZ77, destinam-se 3 *bytes* para tríplice de codificação, os *bytes* O e L colocam-se os valores do deslocamento e coincidência; devido ao tamanho da janela (200), torna-se possível o armazenamento dessas informações em um byte e C representa o *byte* de caracter. O byte N é utilizado como verdadeiro ou falso para indicar se o último caracter é válido.

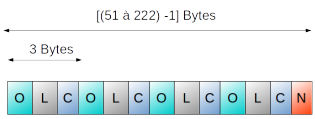


Figura 3: Estrutura da mensagem de LZ77

## VI. Resultados

Visando coletar os dados sem a interferência de outras partes da plataforma utilizada nos algoritmos, foram adotados 3 cenários distintos.

O cenário 1 é dedicado a obter o tempo de execução, taxa de compressão e heap utilizado. Neste cenário todo processador está dedicado apenas a compressão. Para realizar a compressão, um conjunto de 2000 mensagens reais obtidas dos dispositivos em campo de tamanho $\sim$ 30 *bytes* foram salvas em um arquivo de texto. Ao realizar o processo de compressão, um número $x$ de mensagens são escolhidas de forma aleatória de modo avaliar o comportamento da taxa de compressão com mensagens de conteúdo correlacionados e não correlacionados. Vale ressaltar que o tempo para obtenção da mensagem no arquivo, abertura e fechamento é removido do tempo de execução para evitar quaisquer distorções. Funções específicas da plataforma são utilizadas para se obter o tempo de execução.

O cenário 2 é voltado à obter a métrica de energia gasta pelo método de compressão, com o processador dedicado apenas à compressão. O código do cenário 1 foi alterado para que a mensagem a ser comprimida seja retornada de uma função sem a necessidade de abertura de um arquivo, pois tal operação aumenta sensivelmente o consumo de energia, sendo difícil separar precisamente os dados no *datalogger*. Para coleta do consumo de energia foi construído um circuito para monitoramento da energia consumida pelo dispositivo conforme mostra a Figura 4. O mesmo pode ser subdividido em estágios, sendo o primeiro o circuito utilizado para medição de corrente, neste trabalho adotamos o *low-side sensing*, o segundo estágio é um circuito de *buffer* para aumentar a impedância de entrada do circuito e por último o valor analógico é convertido por um ADC, neste caso o ADS1115.

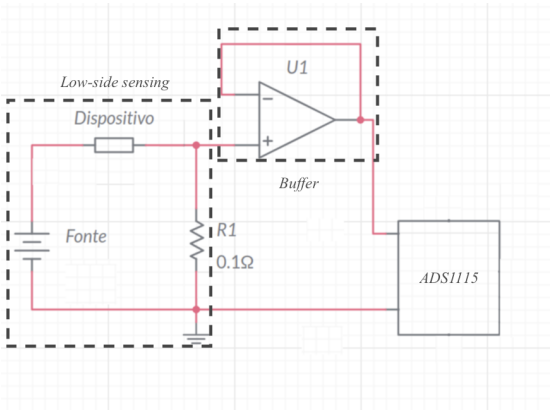


Figura 4: Circuito para aferição de energia.

No último cenário, é avaliado qual seria o ganho na aplicação real, onde o dado é comprimido e enviado via rede LoRaWan. Todos os pacotes enviados são considerados como recebidos, não considerando cenário com perda de pacotes. Assim, deve-se considerar o gasto energético do dispositivo executando a compressão e suas demais aplicações, além do custo do envio do pacote. Neste cenário o dispositivo permanece sempre ligado até o envio completo das mensagens.

### A. Cenário 1

Para a coleta dos dados de tempo de execução, tamanho do *heap* e taxa de compressão foram comprimidos grupos de 1 a 19 mensagens, onde cada agrupamento foi executado 100 vezes com mensagens distintas entre si, totalizando 19000 mensagens comprimidas. Essa quantidade máxima de mensagens(19), foi escolhida considerando o tamanho máximo suportado no pacote LoRaWan de 222 bytes, demonstrando a característica diferenciada do problema em relação a aplicação tradicional da compressão de dados em computadores de propósito geral. Os resultados de *heap*, taxa de compressão, tempo e tamanho de saída da mensagem são apresentados nas Figuras 5, 6, 7 e 8.

Na Figura 5, nota-se que o Huffman tem um uso maior da *heap* devido a necessidade de criações de árvore e filas para a execução do algoritmo. É possível notar que a partir de duas mensagens, a diferença do uso do *heap* entre os métodos torna-se praticamente constante, visto que já se atingiu o máximo comprimento da árvore e fila utilizadas em Huffman, como mostra a Figura 5. Além disso, é possível notar que o comportamento dos métodos é linear, portanto adequado para sistemas IoT.

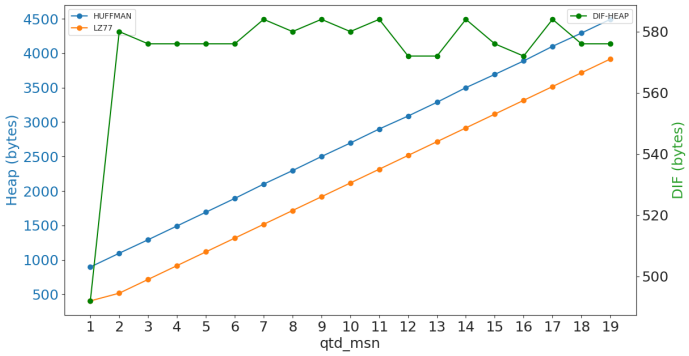


Figura 5: *Heap* utilizado

Na Figura 6 nota-se que as taxas de compressão para esse estudo de caso possuem um comportamento logarítmico que tendem a estabilizar, enquanto a diferença entre as taxas possui um comportamento exponencial decrescente em que o LZ77 tende à uma taxa de compressão 20% menor que a de Huffman. A taxa de compressão negativa para o LZ77 indica que houve uma expansão dos dados ao invés de compressão, sendo que o aumento da taxa de compressão é proporcional ao volume de dados.

Na Figura 7 pode-se perceber que o tempo de execução para o Huffman é superior ao LZ77. No caso para uma mensagem, o LZ77 foi cerca de 6,5 vezes mais rápido e para 19 1,7. No entanto, é importante frisar que para pequenas mensagens o LZ77 não tem taxa de compressão adequada, portanto inviabilizando os seu uso.

Na Figura 8 pode-se observar que utilizando a compressão de Huffman foi possível comprimir até 3 mensagens dentro

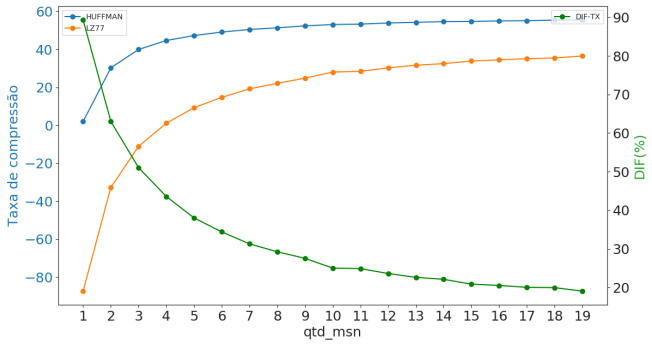


Figura 6: Taxa de compressão

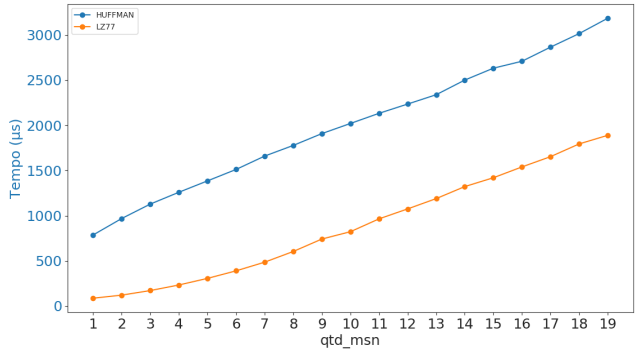


Figura 7: Tempo de execução

do limite do SF 12, enquanto para o LZ77 apenas uma, não havendo sentido de realizar compressão. Considerando o cenário de SF 7 é possível comprimir em média até 12 mensagens utilizando LZ77 e 19 utilizando Huffman.

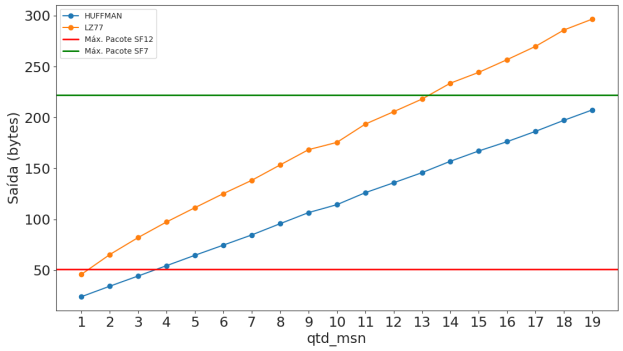


Figura 8: Tamanho da mensagem de saída

## B. Cenário 2

No Cenário 2 foi realizado o experimento similar do cenário 1, porém o tempo para obtenção da mensagem a ser comprimida por meio da função é considerado. A Figura 9 mostra a energia gasta apenas na compressão, o gasto da plataforma não está inserido, ou seja, o gasto fixo de manter o dispositivo ligado não é considerado, uma vez que o mesmo tende a ser muito maior que o consumo da compressão.

A Figura 10 mostra a diferença de duração entre os métodos e a amplitude da tensão medida no resistor shunt, onde é

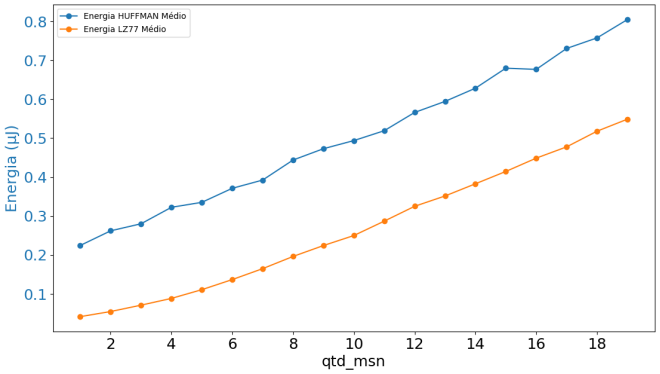


Figura 9: Consumo Energético Huffman vs LZ77

possível notar que a compressão LZ77 tem um menor tempo de execução e um menor consumo de energia. Além disso por meio da figura, percebe-se que a amplitude da tensão aumenta a medida que aumenta-se o número de mensagens. Em termos percentuais, no o LZ77 há um aumento de 8% e para o Huffman de 18%.

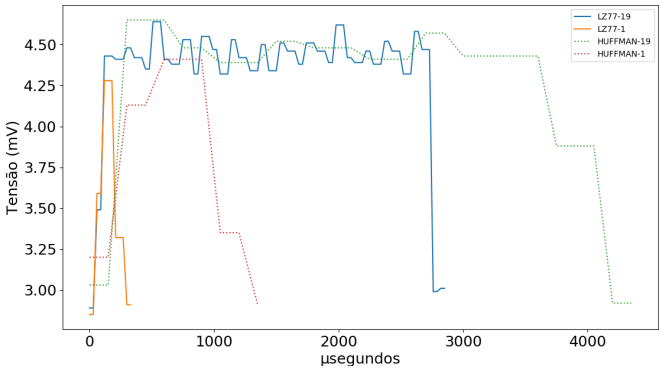


Figura 10: Tempo de Execução

## C. Cenário 3

O cenário avaliado considera o envio de 20 mensagens de dados com envio a cada 15 segundos quando os dados não são comprimidos. Quando a compressão é utilizada, inicialmente as mensagens são agrupadas e comprimidas, somente após essa etapa são enviadas. Por exemplo, de acordo com a Tabela VI no experimento Normal SF 12, foram enviados 20 pacotes, onde cada pacote foi enviado contendo apenas uma mensagem a cada 15s. Já no Huffman SF 12, foram enviados ao todo 7 pacotes, sendo que 6 possuem 3 mensagens comprimidas e um contém 2 mensagens comprimidas, totalizando 20 mensagens. Assim, a taxa de envio com 3 mensagens passa a ser a cada 45 segundos, pois as mensagens são colocadas em um buffer e enviadas em um único pacote. O valor de 3 mensagens por pacote foi utilizado baseado nos dados coletados do Cenário 1 e apresentados na Figura 8. No caso do fator de espalhamento SF 7 ser possível em média comprimir até 19, utilizou-se 16 mensagens como segurança devido a variabilidade da taxa de compressão.

A Figura 11 mostra o gasto médio de 5 execuções em cada experimento. É possível notar que no SF 7 não há ganho em energia, pois o custo de transmissão é menor neste fator de espalhamento. No entanto, mesmo que não se obtenha ganhos energéticos, a compressão neste contexto poderia ser utilizada para garantir a escalabilidade da rede, reduzindo o uso do canal devido a redução no número de mensagens enviadas. Essa redução no uso do canal também é perceptível se analisarmos o fator de espalhamento SF 12, onde é possível reduzir em quase 3 vezes o número de pacotes transmitidos. A Tabela VI mostra a quantidade de mensagens enviadas por pacotes e quantos pacotes foram enviados, bem como a economia de energia, onde $*$ é o gasto de referência no cenário normal sem compressão. Não houve experimento para o LZ77 no SF 12 pois o mesmo ultrapassa o limite de 51 *bytes* além de expandir mensagem como mostrado na Figura 6 e 8. O ganho energético com o uso da compressão na fator de espalhamento SF 12 chegam a 17%, o que é benéfico para dispositivos alimentados por bateria.

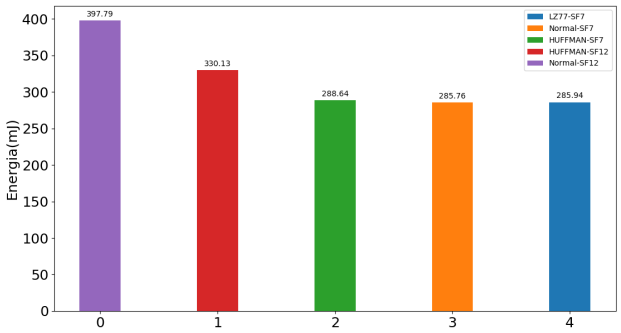


Figura 11: Consumo Energético- Algoritmos + Plataforma

Tabela VI: Tabela: Ganho Energético

| MODO | SF | Mensagens | Pacotes | Ganho Energético |
|---|---|---|---|---|
| NORMAL | 12 | [1x20] | 20 | * |
| HUFFMAN | 12 | [3x6 +2] | 7 | +17,01% |
| NORMAL | 7 | [7,7,6] | 3 | * |
| LZ77 | 7 | [11,9] | 2 | -0.06% |
| HUFFMAN | 7 | [16,4] | 2 | -1.01% |

## VII. Conclusões e Trabalhos futuros

Este trabalho analisou o impacto no consumo energético do uso de métodos de compressão de dados na transmissão de dados em uma rede LoRa. Os métodos Huffman e LZ77 foram adaptados ao contexto IoT e analisados. Os dados mensurados ao realizar a compressão utilizando SF 12, obtiveram uma economia de energia de 17,01%, mostrando que apesar de aumentar o custo computacional tem-se uma economia significativa de energia.

Pode-se observar também o Huffman resulta em taxas melhores de compressão que o LZ77. No entanto, o LZ77 é mais rápido e consome menos energia e memória *heap*. Assim, caso a aplicação tenha como requisito não somente a taxa de compressão, o LZ77 torna-se uma escolha viável caso seja necessário comprimir um volume maior de dados, visando reduzir o tempo de execução e o consumo de energia.

Como trabalhos futuros espera-se analisar mais algoritmos de compressão, além dos apresentados neste trabalho, que possam se adaptar melhor as características do tamanho de pacote da rede LoRa, bem como o desenvolvimento de um dispositivo de medição mais robusto. Adicionalmente, será analisado o uso do deep sleep como forma de reduzir o consumo de energia entre os envios das mensagens, aumentando ainda mais o impacto da compressão de dados.

## Agradecimentos

À Trinovati Tecnologia LTDA pela colaboração no desenvolvimento do trabalho.

## Referências

[1] A. Lele, *Disruptive Technologies for the Militaries and Security*, 1st ed., ser. Smart Innovation, Systems and Technologies. Singapore: Springer Singapore, 2019, vol. 132.

[2] B. Risteska Stojkoska and K. Trivodaliev, "A review of internet of things for smart home: Challenges and solutions," *Journal of Cleaner Production*, vol. 140, p. 1454–1464, 01 2017.

[3] C. M. Sadler and M. Martonosi, "Data compression algorithms for energy-constrained devices in delay tolerant networks," in *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, ser. SenSys '06. New York, NY, USA: ACM, 2006, pp. 265–278.

[4] F. Marcelloni and M. Vecchio, "A simple algorithm for data compression in wireless sensor networks," *IEEE Communications Letters*, vol. 12, no. 6, pp. 411–413, June 2008.

[5] A. K. Maurya and D. Singh, "Median predictor based data compression algorithm for wireless sensor network," *International Journal of Computer Applications*, vol. 24, 06 2011.

[6] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," *IEEE Internet of Things Journal*, 2016.

[7] Y. Ai, M. Peng, and K. Zhang, "Edge computing technologies for Internet of Things: a primer," *Digital Communications and Networks*, vol. 4, no. 2, pp. 77–86, 2018.

[8] D. Sacaleanu, P. Razvan, I. Manciu, and L. Perisoara, "Data compression in wireless sensor nodes with lora," pp. 1–4, 07 2018.

[9] A. Van der Byl, R. Neilson, and R. Wilkinson, "An evaluation of compression techniques for wireless sensor networks," pp. 1–6, 09 2009.

[10] R. S. Sinha, Y. Wei, and S.-H. Hwang, "A survey on LPWA technology: LoRa and NB-IoT," *ICT Express*, vol. 3, no. 1, pp. 14–21, mar 2017.

[11] M. Bor, J. Vidler, and U. Roedig, "LoRa for the Internet of Things," *Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks*, pp. 361–366, 2016.

[12] Semtech Corporation, "LoRa Modulation Basics," pp. 1–26, 2015. [Online]. Available: 1 [Online], https://www.frugalprototype.com/wp-content/uploads/2016/08/an1200.22.pdf

[13] LoRa Alliance, "LoRaWAN 1.0.3 specification," *Lora-Alliance.Org*, pp. 1 –72, 2018. [Online]. Available: [Online], https://lora-alliance.org/sites/default/files/2018-07/lorawan1.0.3.pdf

[14] ——, "LoRaWAN Regional Parameters," *Lora-Alliance*, pp. 1 –88.

[15] K. Sayood, *Introduction to Data Compression, Third Edition (Morgan Kaufmann Series in Multimedia Information and Systems)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.

[16] C. McAnlis and A. Haecky, *Understanding Compression: Data Compression for Modern Developers*, 1st ed. O'Reilly Media, Inc., 2016.

[17] M. Nelson and J.-L. Gailly, *The Data Compression Book (2Nd Ed.)*. New York, NY, USA: MIS:Press, 1996.

[18] C. McAnlis and A. Haecky, *Understanding Compression: Data Compression for Modern Developers*, 1st ed. O'Reilly Media, Inc., 2016.