



Data Compression in LoRa Networks: A Compromise between Performance and Energy Consumption

Javan Ataíde de Oliveira Júnior   [Western Parana State University, Campus Cascavel | javanataide10@gmail.com]

Edson Tavares de Camargo  [Federal University of Technology – Paraná, Campus Toledo | edson@utfpr.edu.br]

Márcio Seiji Oyamada  [Western Parana State University, Campus Cascavel | marcio.oyamada@unioeste.br]

 Western Parana State University - Campus Cascavel, Rua Universitaria 2069, Cascavel -PR, 85819-110

Received: 01 November 2022 • Accepted: 15 May 2023 • Published: 18 July 2023

Abstract The Internet of Things (IoT) end devices have major limitations related to hardware and energy autonomy. Generally, the highest energy consumption is related to communication, which accounts for up to 60% of consumption depending on the application. Among the strategies to optimize the energy consumed by communication, data compression methods are one of the most promising. However, most data compression algorithms are designed for personal computers and need to be adapted to the IoT context. This study aims to adapt classical algorithms, such as LZ77, LZ78, LZW, Huffman, and Arithmetic coding, and to analyse their performance and energy metrics in IoT end devices. The evaluation is performed in a device with an ESP32 processor and LoRa modulation. The study makes use of real datasets derived from two IoT applications. The results show compression rates close to 70%, a three-fold increase in the number of messages sent, and a reduction in energy consumption of 22%. An analytical model was also developed to estimate the gain in the battery life of the device using the adapted algorithms.

Keywords: Internet of Things, data compression, energy consumption, LoRa

1 Introduction

Due to technological advances, the Internet of Things (IoT) market is expected to reach trillions of dollars in the coming years (Statista, 2020), which means it will expand its range of applications to include sensors, healthcare applications, industry applications, and smart cities (Perera *et al.*, 2015). However, most IoT end devices have limited power sources due to the characteristics of their applications. This fact imposes restrictions on both the processing capacity and the communication bandwidth (Samie *et al.*, 2017).

In the context of data communication, wireless networks called Low Power Wide Area (LPWAs) have low power consumption, low transmission rates, and long-range characteristics. LPWA networks, such as LoRa and Narrowband IoT (NB-IoT) technology, have been applied both in cities and in rural environments (Gu *et al.*, 2020). Despite these characteristics, data transmission consumes most of the energy due to the high energy of the radio used for transmission. In some cases, the energy cost of communication reaches 60% of the device's energy consumption (Sadler and Martonosi, 2006). It is estimated that the energy cost to send and receive a single bit of information is equivalent to 1,000 instructions performed on a given processor (Marcelloni and Vecchio, 2008). Thus, due to the large proportion of energy consumed, data compression is among the various "edge computing" approaches currently proposed to reduce energy consumption in the process of sending data. The term edge computing refers to using the computational resources of the devices to process the data at the edge instead of transmitting to the

centralized cloud. The approach can improve issues related to response time and energy savings. However, most of the main compression algorithms need to be adapted when used in IoT end devices due to hardware restrictions (Sadler and Martonosi, 2006).

This study aims to adapt the classic algorithms - Arithmetic, Huffman, LZ77, LZ78 and LZW - for IoT devices to analyse metrics that can be critical in determining the compression algorithm to be used in a real application. The metrics evaluated are heap memory, battery, the energy consumption of the method, peak current, energy gain and processing time. The exploration of these measures can serve as a parameter for applications that want to use data compression to increase energy gain or increase network throughput, based on more than the rate of compression as a criterion for choosing the method.

The use of transmission exchange by computation at the edge has been investigated in several studies (Maurya and Singh, 2011), (Sacaleanu *et al.*, 2018), (Sadler and Martonosi, 2006), (Al-kadhim *et al.*, 2021). However, each author evaluates a particular measure of interest, such as compression rate (Sadler and Martonosi, 2006), energy gain (Sacaleanu *et al.*, 2018), processing time or memory usage (Maurya and Singh, 2011). The main contribution of this study is to evaluate a set of metrics, usually absent in most related studies, and perform correlational analysis between the measurements. Another contribution is the analysis of the impact of the number of messages and energy consumed on different LoRa spreading factors.

This research uses as a case study a node composed of

an ESP32 end device with LoRa modulation. Two datasets were used: one refers to the monitoring of the heating of concrete blocks in large buildings, and the other refers to object tracking data containing GPS coordinates (Camargo *et al.*, 2021). The use of two different datasets is due to the facts presented in the works in (Sadler and Martonosi, 2006), (Vander Byl *et al.*, 2009). According to both, the data content impacts measures such as the compression rate. Results obtained demonstrate that the LZW, Huffman and Arithmetic algorithms reached a compression rate of 69%, 61% and 55%, respectively, for the first dataset. In terms of energy consumption, the data compression could result in an energy consumption reduction up to 22%, using the LZW algorithm. Findings for the second dataset indicated that it was possible to reduce the network throughput by 95% with the LZW, Huffman and Arithmetic algorithms.

To present the results in this study, the text is structured as follows: Section 2 presents a background of LoRa modulation and the relative studies involving data compression and IoT devices. Section 3 presents the adopted methodology and details the compression algorithms used in this work. Section 4 presents the results of the research, and Section 5 details the analytic energy model proposed in this work based on the results collected. Section 6 provides the conclusion.

2 Background

This section provides a brief overview of LoRa and LoRaWAN and presents the related work.

2.1 LoRa and LoRaWAN

The LoRa modulation from Semtech is an emerging Low-Power Wide Area Network (LPWAN) for IoT applications. LoRa uses a modulation technique derived from Chirp Spread Spectrum (CSS) modulation (Sinha *et al.*, 2017; Bor *et al.*, 2016), and exhibits characteristics such as long-range, high robustness, and low energy consumption (Bor *et al.*, 2016).

LoRa devices have four main parameters: Spreading Factor (SF), Bandwidth (BW), Carrier Frequency (CF), and Coding Rate (CR) (Bor *et al.*, 2016). One of the main features of LoRa is the ability to exchange the data transmission rate for a higher sensitivity within the bandwidth of the channel, thus achieving a variable data transmission rate just by changing the spreading factor to optimize the network performance (Semtech Corporation, 2015). A higher SF increases the signal-to-noise ratio (SNR) and consequently increases robustness and signal range. However, air time and energy consumption for data transmission increase, and the number of bits transmitted in the message decrease (Bor *et al.*, 2016). The air time required to transmit a packet with a payload of 10 bytes in SF12 is about 25 larger compared to SF7, as shown in Table 1.

The air time refers to the time it takes for a packet sent from a particular transmitter to reach the receiver (Aras *et al.*, 2017). In LoRa communication, this time difference is due to the technique that uses the chirp signal to spread the transmitted signal by varying the frequency. The duration of the

chirp signal is directly related to the spreading factor used by the device. A higher SF means achieving a longer data transmission distance and a lower baud rate, resulting in a longer signal transmission time (Ahmar *et al.*, 2019).

Table 1. Air time according to SF for a payload of 10 bytes (Semtech Corporation, 2019)

SF	Time (ms)
7	41
8	72
9	144
10	288
11	577
12	991

The LoRa physical layer may be used with any Media Access Control (MAC) layer. However, LoRaWAN is the currently proposed MAC, which operates a network in a simple star or star-of-star topology and consists of nodes and one or more gateways used to transfer data between devices to LoRaWAN servers (Semtech Corporation, 2023). One of the advantages of this protocol is that it is optimized for devices with limited power (Sinha *et al.*, 2017).

Devices using the LoRaWAN protocol must follow regional standards established by each country, such as operating frequencies, bandwidth, and maximum message size. The LoRaWAN Alliance provides regional parameters for different regulatory regions worldwide (LoRa Alliance, 2021).

In Brazil, where the experiments were conducted, devices can operate in regions defined by LoRaWAN as AU915-928 or EU433. Devices using these regions have 64 uplink channels with a bandwidth of 125 KHz interleaved at 200 KHz, starting at 902.5 MHz to 927.8 MHz, and can use SF from 12 to 7. The size of the payload depends on the SF and bandwidth chosen. In this work, a bandwidth of 125 KHz and SF from 12 to 7 were used, and the maximum payload is considered in the most restrictive scenario of LoRaWAN. In this case, the payload ranges from 51 bytes for SF12 to 222 bytes for SF7, as shown in Table 2 (LoRa Alliance, 2021).

Table 2. Maximum payload in different SF (LoRa Alliance, 2021)

SF	Payload (Bytes)
12	51
11	51
10	51
9	115
8	222
7	222

2.2 Related Work

The data compression in the IoT enables the use of new techniques focused on the optimization for IoT devices. Due to the limitations of the devices, new algorithms for compression have been proposed or adapted to make them feasible (Sadler and Martonosi, 2006). In particular, exchanging

transmission for edge computation has been applied in several studies. However, each author has focused on a singular measure of interest, be it compression rate, energy gain, computational or memory cost; thus, no study verifies the overall impact in an integrated way of different metrics such as compression rate, execution time, memory usage, and energy consumption. In the literature, there are several adaptations of classical algorithms to IoT devices.

In (Maurya and Singh, 2011), the authors proposed a lossless compression algorithm to reduce the computational cost and the memory usage, considering that the data are stored in the device. They proposed the Median Predictor-based Data Compression (MPDC). The MPDC can be divided into three stages. The first consists of collecting the current sensor reading plus three previous values. The second is the median predictor that predicts the median value, selects the highest and lowest value and calculates the deviation from the current value in relation to the median. The third consists of encoding the deviation value using a static Huffman table. Using the algorithm, compression rates of up to 67% (7500 bits) were obtained. Although the authors reported a reduction in energy consumption, the value of this reduction was not quantified, nor were the memory footprint and execution times provided.

In (Sacaleanu *et al.*, 2018), the authors performed data compression using a LoPy module with LoRa radio. With a focus on reducing energy consumption, they used the delta or residual model (difference between one data collected and another) and coded it with a static Huffman table, similar to that used in (Maurya and Singh, 2011). The objective of this study was to compare the energy cost for the transmission of these compressed data via three transmission networks: ZigBee, Enhanced ShockBurst and LoRa (each modulation used a platform different but with the same compression algorithm). According to (Sacaleanu *et al.*, 2018), when analysing the costs with and without compression, in the LoRa network, up to 31% in energy savings were gained. However, the study did not measure data such as processing time, memory and other compression algorithms.

In (Marcelloni and Vecchio, 2008), the lossless entropy compression (LEC) algorithm was proposed. The LEC explores a modified version of the exponential-Golomb code, which consists of dividing the alphabet and numbers into groups whose sizes increase exponentially. Unlike exponential-Golomb coding, in which codewords are generated as a combination of single and binary codes, for the LEC, codification is performed using the Huffman static table, similar to (Maurya and Singh, 2011) and (Sacaleanu *et al.*, 2018). The compressor was applied to correlated data (data that tend to be similar) of temperature and humidity and reached 70% and 62% compression rates, respectively. The authors applied the algorithm to noncorrelated data, where theoretically it would not have good results, but it was able to achieve compression rates close to 70%, thus showing the validity and reliability of the LEC. However, the study focused on comparing the compression rate and measuring energy consumption analytically, which resulted in a 32% decrease in energy consumption.

In (Samie *et al.*, 2017), a lossy compression algorithm was proposed for biosignal applications. The proposed method

uses Huffman coding and the delta model. To avoid creating a table that is too large, only a few delta values are stored in the Huffman table, allowing the delta to be encoded to search for a value close to its value in the table, within an error limit. According to the results, the proposed methodology obtained a compression rate of 75% in some cases. Aiming to evaluate energy consumption, there was an increase in energy efficiency, increasing the lifespan of the device from 7 days using the state-of-the-art algorithm to 10 days of autonomy. The total energy consumption was calculated by summing the main parts of the device's operation, such as the energy consumed to send messages and the processor in normal state or *deep sleep*.

In (Sadler and Martonosi, 2006), a compression method focused on IoT in delay-tolerant systems, the LZW sensor (S-LZW) and some variations of it, are proposed. According to (Sadler and Martonosi, 2006), the S-LZW alone cannot take advantage of the existing characteristics in sensor data since the data tend to be repetitive at short intervals. Based on the prerogative to optimize these patterns, the authors proposed the S-LZW mini cache (S-LZW-MC) as a variation of the S-LZW. The mini-cache is a hash-indexed dictionary with size N , where N is a power of 2 that stores the recently used and created dictionary entries. According to the results presented in (Sadler and Martonosi, 2006), the dictionary with 32 and 64 entries had the best performance in the tests, with gains of 12.9% and 14.7% in the compression rate when compared to the S-LZW in the SS dataset. Therefore, to evaluate the energy gain using the S-LZW-MC algorithm with 32 inputs, experiments were performed using three radio types (XTend, CC2420 and CC1000) in which four benchmarks were compressed. According to the results, the S-LZW-MC was able to reduce energy consumption by 1.6 to 1.7 times.

In (Al-kadhim *et al.*, 2021), two compression algorithms were used to maximize and evaluate the energy gain. The method is called the adaptive data compression scheme (ADCS). This method consists of two algorithms, the S-LZW and the S-LEC, which are selected according to the interest of the application. The authors also proposed an automatic mode that selects the compression algorithm based on the energy gain of the application. For the proposed study, the best gains were considering the S-LEC. The main purpose of this study is to reduce power consumption using the proposed algorithm. As a result, the authors achieved a reduction of 33% in energy when the algorithm used S-LZW and 40% when S-LEC was used. The number of operations performed by the processor was also evaluated, which indicated greater use in the S-LEC algorithm, estimating the increase of useful life at 50%. Despite the results, measures such as memory were not discussed, and the energy values were estimated using analytical models.

In (Mishra *et al.*, 2022), the authors evaluate the run-length encoding (similar to LZ77) and Huffman encoding. The authors also propose a hybrid approach that combines run-length and Huffman coding to compress the data. To evaluate the compression ratio and power consumption, the authors propose an analytical model based on the MCP TIMSP430 microcontroller. It is important to mention that it is a simple microcontroller and in the analytical model, the authors have considered a fixed number of cycles per instruction (CPI) for

each instruction class. The authors evaluate the compression using the input data of different sizes from 300 to 1500 bits, but they do not describe the content or the symbol distribution. For the energy estimation, they assume a fixed cost for transmitting a bit that is not associated with a specific technology. The authors reported a compression ratio for a data input of 300 bits of approximately 83% for the run-length algorithm, 78% for the hybrid approach using a combination of run-length and Huffman, and 66% with the Huffman algorithm.

This study implemented the classic Huffman, Arithmetic, LZ77, LZ78, and LZW algorithms, adapting them to the LoRa network and the device used, focusing on the algorithm and not on data modeling. To adapt the algorithms, some techniques described in the related studies were used, such as the dictionary limitation proposed in the S-LZW by (Sadler and Martonosi, 2006). It is also noteworthy that this study analysed a set of performance variables absent in most of the related studies. The variables analysed were compression rate (CR), execution time (ET), memory usage (heap, stack) (GM), global energy gain (considering the use in a real scenario) (GE), energy gain between the methods (analysing only the energy spent for compression), peak current and modulation (Mod). Table 3 presents a comparison between the related studies highlighting the variables addressed by each study. Variables that did not appear in any of the studies are omitted from the table.

Table 3. Comparison between the related works

	GE	CR	Mod	Hardware	Compression method
(Sadler and Martonosi, 2006)	40%	70%	-	MSP430x1611	S-LZW
(Marcelloni and Vecchio, 2008)	-	70%	-	MSP430	LEC
(Vander Byl et al., 2009)	-	86%	-	TelosB device	Wavelet-Hybrid
(Maurya and Singh, 2011)	-	67%	-	Simulation	MPDC
(Samic et al., 2017)	48%	75%	BLE	M6d. ECG	Approximation Compressor
(Sacaleanu et al., 2018)	30%	-	LoRa	Board LoPy	Huffmann Static
(Le and Vo, 2018)	-	75%	-	Simulation	D-LZW
(Tasaka et al., 2019)	-	80%	LoRa	-	GPS Compression
(Hanumanthaiah et al., 2019)	-	52%	-	-	Delta + RLE
(Mishra et al., 2022)	-	83%	-	TIMSP430	RLE + Huffman

3 Methodology and Evaluated Scenarios

The study was conducted using a TTGO OLED Display LoRa module, which uses an ESP32 architecture that is composed of two 32-bit Xtensa LX6 processors, an ultralow power (ULP) coprocessor, 4 MB of Flash Memory and 528 kB of SRAM, Wi-Fi, Bluetooth, 18 ADC ports, SPI, I2C and an SX1276 LoRa radio (Lilygo, 2019).

For the measurement of the data compression rate and energy-related metrics, experiments were divided into three scenarios, with a specific focus on the following metrics: compression rate, execution time, memory usage (heap, stack), global energy gain (considering the use in a real scenario), and the energy consumption of the algorithm and peak current.

In the first scenario, only the code is loaded on the end device to perform compression and decompression to prevent any of the metrics from interfering with the normal operating code of the IoT application. In this scenario, real data collected in the field were saved in a text file so that dur-

ing the execution of this scenario, the compressed messages were loaded from this file. The file manipulation time (opening, reading, and closing) is not considered. As a result of this experiment, compression metrics are collected, such as the maximum amounts of compressed messages within the limit of the LoRaWAN packet size.

In the second scenario, the focus is the measurement of the energy consumption of the compressor algorithm. Thus, for a more accurate measurement, the messages to be compressed are statically encoded in the code to prevent the cost of opening and closing the file from distorting the compression cost. Thus, similar to the first scenario, only the compression code is loaded into the device. For the energy measurement, a current monitor INA219 (TI, 2023) was used in an invasive manner that sampled the current at a rate of 250 samples per second.

The third scenario focuses on measuring the energy gain of the compression when sending the data, simulating a real application. In this scenario using the compressor algorithm, the data are collected and grouped until reaching the maximum number of messages allowed in a LoRaWAN packet (obtained in Scenario 1). The transmission was performed using the LoRa modulation, but in the experiments, we consider the packet size limit for each SF determined by the LoRaWAN protocol.

In this work, we evaluate the compression methods Huffman, Arithmetic, LZW, LZ77 and LZ78. All compression methods are lossless and are classified as statistical-based (Huffman and Arithmetic) and dictionary-based (LZW, LZ77, and LZ78).

3.1 Huffman Algorithm

The Huffman algorithm is a probability-based compression technique. This technique uses the probability of occurrence of symbols in a data set to form variable-length codewords with the smallest average length of symbols (McAnlis and Haecky, 2016).

To use the algorithm, it is necessary to create a table of probabilities that can be obtained in two ways: statically or dynamically. The static method uses a fixed probability table that is known to both the encoder and decoder. One advantage of this method is that the probability table does not have to be sent with the encoded message. However, one of the disadvantages is that the table may not represent the data to be encoded, resulting in a degradation of the compression rate (Nelson and Gailly, 1996).

In the adaptive method, the frequency distribution of symbols in the message to be compressed is computed. An advantage of this approach is the possibility of achieving a better compression rate since it obtains the probability table from the message. However, the probability table must be included in the message to allow decoding. Therefore, despite the improvement in compression ratio, the encoding time and computational cost increase (Nelson and Gailly, 1996). For the algorithm used in this work, we used an adaptive table because the data format and application have many variations that could degrade the compression ratio when using a static table.

The classical text compression algorithm uses a probability table with a size of 256 (ASCII symbols). However, for IoT-oriented applications, where the alphabet used in most applications is smaller than the ASCII table, the algorithm has been adapted to reduce the size of the table to be transmitted by limiting its size to N symbols, where N is the number of different symbols in the application's alphabet.

Another adaptation is to store the number of occurrences of each symbol in the table instead of the probability, reducing the size of the table. The elements in the table have a size of one byte, so the same symbol can occur a maximum of 255 times. Beyond this value, the symbol has a fixed value at probability 255. Considering that the maximum payload in LoRaWAN when using SF7 is 222 bytes, the limit of 255 occurrences does not limit the compression ratio that can be achieved by the adaptation performed in this work.

3.2 Arithmetic algorithm

Arithmetic coding is another variable-length coding technique that is particularly suitable for small alphabets and skewed probabilities (Sayood, 2005). Similar to the Huffman algorithm, arithmetic coding also uses the probability of the source to perform data compression. However, unlike Huffman, which generates one codeword per symbol, arithmetic coding can generate one codeword for a sequence of symbols.

In arithmetic coding, a single sequence of symbols is converted into a number (tag), that requires fewer bits to represent it. The method of generating the tag works by reducing the size of the range in which the tag is located when new elements are received for encoding. First, the probabilities of the symbols are calculated and the unit interval is divided according to the value of the cumulative distribution function (*cdf*) of each symbol. The tag starts with the constraint on the unit interval and when a new symbol is received, the probabilities of each symbol are updated. The *cdf* function is used to distribute a new range for each symbol proportional to the probability of the symbol (McAnlis and Haecky, 2016).

The traditional algorithm uses a fixed probability table. However, similar to the Huffman algorithm, the algorithm has been adapted to calculate a probability table for each message. The table must therefore be sent together with the compressed message.

3.3 LZ77

Although variable-length encodings such as Huffman and Arithmetic work well, they are limited by the entropy of the source, which has driven the search for new ways to compress data, and other dictionary-based compression techniques have emerged (McAnlis and Haecky, 2016).

In the LZ77 algorithm, the created dictionary is based on the previously encoded sequence. The encoder works by examining an input string through a sliding window. This window consists of a search buffer, which contains the previously encoded sequence, and a look-ahead buffer, which contains the sequence to be encoded. For each encoded symbol or set of symbols, the encoder performs encoding according to the following format $[o, l, c]$, where o is the displacement,

l is the length, and c is the following symbol in the coincidence character string.

To optimize the number of bytes used, the adaptation proposed in this work limits each index of the triple to a size of 1 byte. Given this limitation on the indexes, the values of the search and look-ahead buffer sizes must be less than 255, since the highest possible match would result in this value. Therefore, in the tests, the search buffer size was set to 200 bytes and 54 for the look-ahead buffer in order to avoid overflow.

3.4 LZ78

The LZ77 algorithm assumes that similar patterns occur at positions close to each other and that the search buffer is able to capture these patterns. However, if the patterns occur outside the search buffer, the method does not achieve good compression rates.

To address these limitations, the LZ78 algorithm discards the search buffer and creates an explicit dictionary. However, one of the limitations is that this dictionary must be created in the encoder and the decoder, and they must be identical. The encoding for this algorithm does not use three terms like LZ77, but the pair i and c , where i is the index corresponding to the position in the dictionary with the largest number of matching symbols, and c is the symbol following the encoded string, similar to LZ77. If the dictionary does not contain the symbol, the index value is zero and c is the symbol to be encoded (Sayood, 2005).

In the classical LZ78 algorithm, there is no limit on the size of the dictionary. However, this kind of approach is not practical for IoT applications because it could cause a large memory requirement for processing and creating the dictionary. Therefore, based on the work of (Sadler and Martonosi, 2006), which proposed a limit on the size of the LZW dictionary, a similar strategy was adopted for LZ78.

As adopted in LZ77 implementation, the values of i and c were restricted to one byte, limiting the dictionary to 256 indexes.

3.5 LZW

The LZ algorithms have a number of variations, such as the LZ77 and LZ78 algorithms, but one of the most widely used is LZW. This was proposed by Terry Welch (Welch, 1984), in which the second element of the pair (i, c) does not need to be encoded, so only the index of the dictionary is sent. However, the dictionary must first be prepared with all symbols of the source alphabet, both in the encoder and the decoder.

For this implementation, all possible symbols of the application's alphabet were defined as the initial dictionary. Additionally, it was assumed that the dictionary has a finite size so that no new combination can be added to the dictionary as assumed by (Sadler and Martonosi, 2006). In this work, the maximum size of the dictionary was limited to 256 indexes (0-255), so each index of the dictionary could be sent with a single byte.

4 Experimental Results

The results were obtained using two datasets. The first dataset is the temperature monitoring data of concrete blocks used in the foundations of large buildings. In this dataset, there is a strong correlation between the data. The second dataset was obtained from GPS data collected from an application in the smart cities domain (Camargo *et al.*, 2021), using data with little temporal correlation. The two datasets have text messages encoded in ASCII.

The temperature messages have a size of 28 bytes with five fields, the first of which refers to the communication channel where the data will be stored in the cloud platform, and the other four are the temperatures measured by the sensors in the concrete block with up to 3 decimal places. The alphabet used consists of 14 symbols, namely: [0 1 2 3 4 5 6 7 8 9 . < > &]. For GPS data, the message has 22 bytes and consists of two fields for latitude and longitude with 6 decimal places and the alphabet consists of 13 symbols, namely: [0 1 2 3 4 5 6 7 8 9 . , -].

4.1 Scenario 1

In this scenario, a set of 300 messages obtained from the running devices have a size of 28 bytes (Temperature) and 22 bytes (GPS). The messages were saved in a text file on the device. In the measurement of the metrics (execution time, heap used, stack and compression rate), groups of 1 to 39 messages were compressed. For each cluster, 100 compressions were performed with different messages, totalling 3,900 compressed messages.

Through the data collected, it was observed that the LZW algorithm required a longer compression time for both sets. According to Figure 1, the compression time difference between the Huffman algorithm and the LZW considering 39 messages is up to 7 times higher for the temperature data and 8 times higher for GPS data.

Figure 2 shows the compression rate of the algorithms. The rate tends to an asymptotic value for all algorithms in the two datasets. For a few messages, the LZ77 and LZ78 algorithms were not able to compress but generated an expansion in the message (negative compression rate) for the two datasets. In both scenarios, the LZW algorithm obtained the highest compression rate, reaching 70% (temperature) and 63% (GPS). Figure 2(b) presents some peaks in the compression rate between 5 and 15 messages for the LZ77 compression algorithm. As the messages are selected randomly, these values are obtained in executions where the messages are similar (for instance, the same position in the GPS), resulting in a high compression rate.

Despite the high compression rates, dictionary-based algorithms used a large amount of heap memory since algorithms such as LZW need to store the dictionary. For 39 messages, the difference between LZW and Huffman is 300% (Temperature) and 400% (GPS). The use of the stack was practically identical, close to 6,000 bytes, regardless of algorithm and the data size.

The consumption of heap memory was linear with the increase in the number of messages and showed a tendency to stabilize when the algorithms used the maximum sizes of

Table 4. Maximum amount of messages

Algorithm	Max. messages		
	SF	Temp.	GPS
Huffman	12	3	3
	7	16	21
Arithmetic	12	3	3
	7	16	21
LZ77	12	NP	NP
	7	16	13
LZ78	12	NP	NP
	7	11	13
LZW	12	3	3
	7	21	21

their data structures. That is, the Huffman algorithm with the binary tree containing all the symbols as in the dictionary and the algorithms based on the dictionary reach the maximum size of the dictionary.

A correlational analysis between the metrics was performed. It was observed that the time and the heap memory are correlated for the two sets. The more the device used the heap memory, the greater the time consumption. However, the rate of compression and use of heap memory have low correlation. Thus, a higher compression rate does not imply greater use of heap memory. Thus, the data that used the most heap indicated that the correlation between these variables exists but is noncausal, so that a greater use of heap has a stronger relationship with the method than with the compression rate.

Despite the average values shown in the figure, the compression algorithms have the indetermination in the compression rate as a characteristic, which is related to the data that are being compressed. Thus, Figure 3 illustrates box plots of the size of the compressed output as a function of the number of compressed messages for the temperature data. The figure also shows the maximum payload in bytes of a LoRaWAN message according to the SF7 and SF12. The horizontal lines represent the maximum payload limit of the packet for SF12 and 7, with sizes of 51 and 222 bytes, respectively. Given the values obtained, it is possible to estimate the maximum number of messages that can be compressed according to the SF, as shown in Table 4. Some algorithms cannot perform compression within SF12 (LZ77 and LZ78) and therefore have NP (not possible).

4.2 Scenario 2

In this scenario, as described in the methodology, the data are no longer taken from a file but from a function saved in the code. For this function, 21 distinct and consecutive messages were saved. As already mentioned, the objective of this experiment is to measure the energy consumed in the compression and to verify the peak current.

The mean energy value between these experiments is shown in Figure 4. It can be observed that the LZW algorithm has the highest energy consumption in compression. LZW consumes five times more than the Arithmetic compression algorithm for one message and five times more than Huffman for 19 messages.

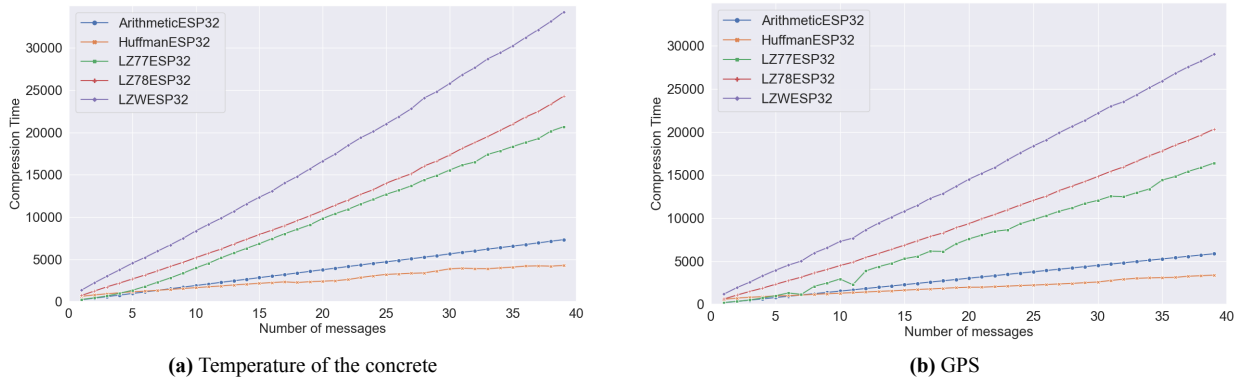


Figure 1. Compression time (µs).

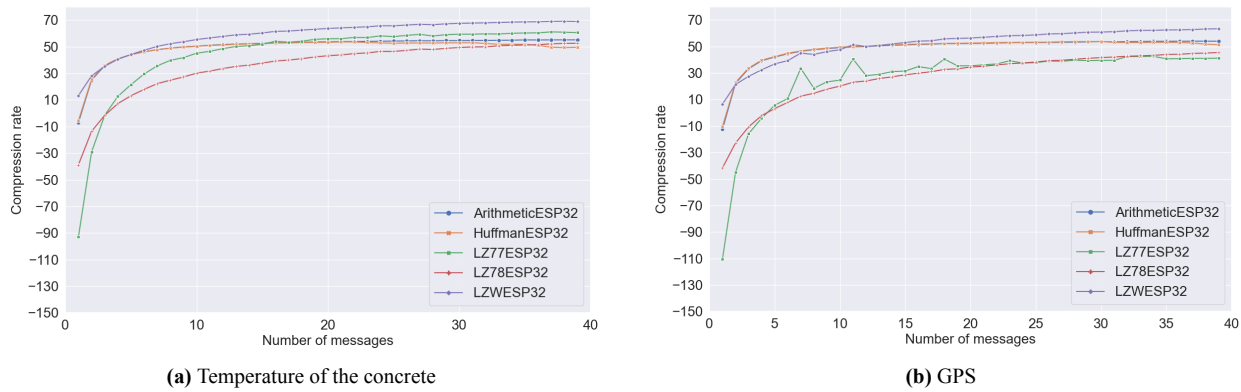


Figure 2. Compression Rate.

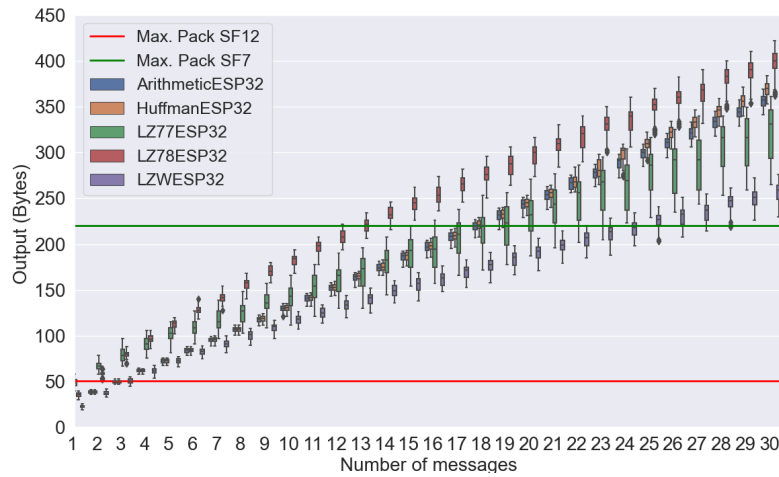


Figure 3. Variability of the compressed output: concrete temperature - SF12 and SF 7 payloads.

There is an important issue that should be addressed. In addition to having its time and energy consumption increase as the messages increase, the Huffman, LZ78 and LZW algorithms increase the current peak considerably during compression.

Figure 5 shows the increase in current amplitude during compression as the number of messages increases. Comparing the current between 3 and 19 messages for the Huffman, LZ78 and LZW algorithms, there was an increase of 10%, 7% and 9%, respectively, for the concrete temperature data.

For example, considering Huffman, for three messages, the average current was 21.68 mA; for 19 messages, the current was 23.92 mA, representing an increase of 10%. For the GPS data, this increase was 9%, 7% and 9%. Although there is no considerable increase in current amplitude, the Arithmetic and LZ77 algorithms have a higher value compared to the other algorithms.

Figure 5 shows that LZ77 and Arithmetic have almost constant amplitudes. It is worth noting that these two algorithms have little memory allocation. LZ77 uses many pointer arith-

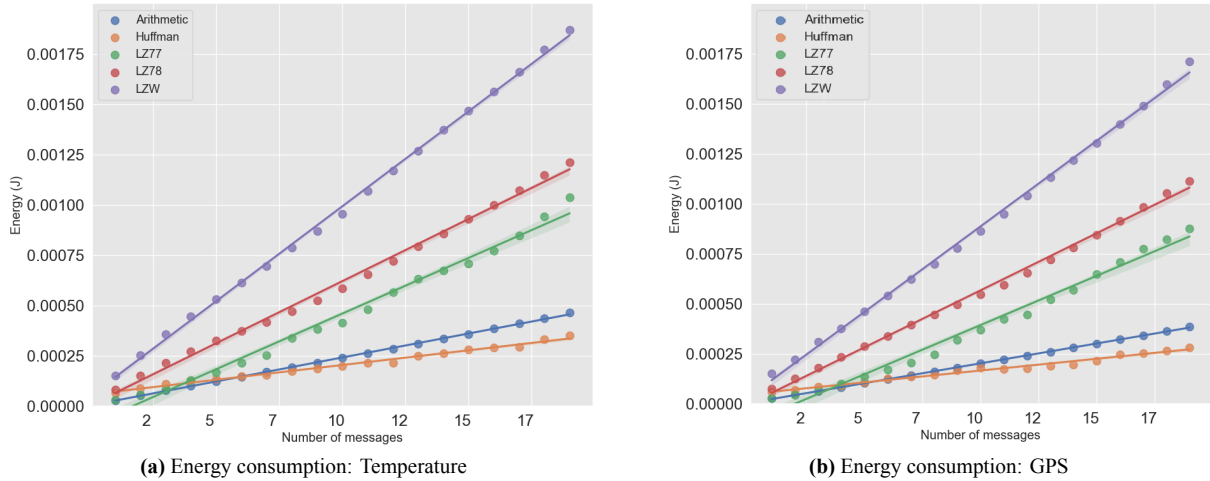


Figure 4. Compression energy consumption.

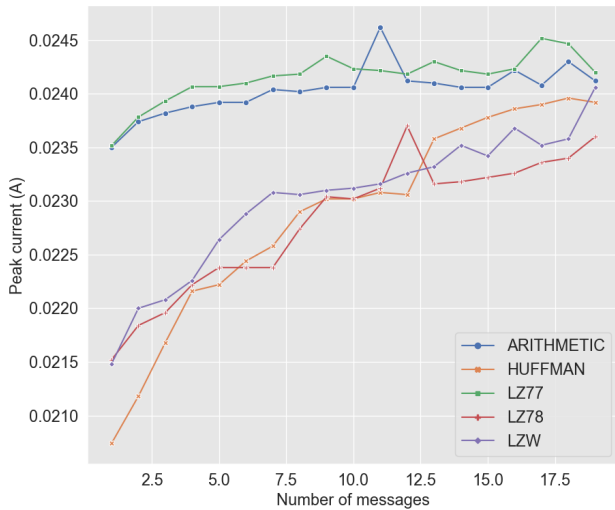


Figure 5. Peak current during compression: Concrete temperature.

metic operations, while Arithmetic uses integer operations. Due to the low memory allocation, the two algorithms mostly use data stored in static memory, while the other algorithms use dynamic memory. This fact may justify the higher current consumption given the memory hierarchy of the ESP32.

Given the linear characteristic of energy consumption according to the number of messages, it is possible to perform a linear regression using the least squares method to find the linear equations for each method, that are presented by Equations 1 and 2. The energy value is given in micro J as a function of the message quantity in Figure 4.

$$\begin{matrix} \text{ARITHMETIC} \\ \text{HUFFMAN} \\ \text{LZ77} \\ \text{LZ78} \\ \text{LZW} \end{matrix} = \begin{bmatrix} 31.19346206 & 240.4389606 \\ 584.6346579 & 147.8401837 \\ -953.0774479 & 559.4875384 \\ 28.56365175 & 624.3464559 \\ 493.016198 & 952.1385793 \end{bmatrix} * \left| \frac{1}{QTY} \right| \quad (1)$$

$$\begin{matrix} \text{ARITHMETIC} \\ \text{HUFFMAN} \\ \text{LZ77} \\ \text{LZ78} \\ \text{LZW} \end{matrix} = \begin{bmatrix} 32.49872122 & 200.3617035 \\ 482.347699 & 118.9338961 \\ -1001.497977 & 498.1676272 \\ -36.23078591 & 577.6657113 \\ 325.3223286 & 861.6462055 \end{bmatrix} * \left| \frac{1}{QTY} \right| \quad (2)$$

4.3 Scenario 3

This scenario, as described in the methodology, is intended to analyse the energy impact of data compression. For this scenario, it was considered that the application sends 21 data packets, each containing a message of 28 and 22 bytes for the concrete temperature and GPS respectively. Each data point is collected at a fixed time interval; for this scenario, two intervals (15 and 60 seconds) were considered. The number of packets used to send the 21 messages is based on the compression algorithm used. For example, considering the Huffman algorithm in SF12, it would be necessary to send 7 packets because the maximum reached in the compression for this SF is 3 messages; thus, 3x7 totals the 21 messages that would be sent in normal mode. Table 4 presents the limits of messages allowed in each packet in a given SF.

Table 5 shows the energy gains using data compression. It can be observed that it was possible to obtain a reduction of approximately 22% of energy for the temperature data and 20% for the GPS data. When considering SF12, it is observed that the best results are for cases where there is deep sleep and a shorter interval time between messages (15 seconds). This behaviour is justifiable because, for this combination, the impact on the compression gain is greater considering the overall energy spent. Similarly, the scenario where the compression had less impact is where the consumption of the device is greater than the transmission (in a continuous 60 seconds).

Considering SF7, there was no energy gain, since most of the values remained close to 1%. Compared to the scenario without compression, this lack of gain is related to the energy consumption of the device when compared to the transmission cost when sending messages in SF12. Since the time to

Table 5. Energy gain (%)

Data	Algorithm	SF 7				SF12			
		DEEPSLEEP		CONTINUOUS		DEEPSLEEP		CONTINUOUS	
		15 s	60 s	15 s	60 s	15 s	60 s	15 s	60 s
GPS	Arithmetic	1.74%	0.65%	0.92%	0.43%	19.73%	9.69%	10.79%	3.39%
	Huffman	1.25%	0.86%	0.91%	0.1%	19.95%	9.66%	10.71%	3.50%
	LZ77	2.74%	1.45%	0.65%	0.24%	-	-	-	-
	LZ78	1.13%	0.35%	0.78%	0.22%	-	-	-	-
	LZW	2.06%	0.15%	0.81%	0.21%	17.48%	8.37%	9.88%	3.13%
	Arithmetic	1.35%	1.54%	0.98%	0.23%	20.58%	11.23%	11.83%	3.61%
	Huffman	1.68%	1.46%	1.23%	0.15%	21.51%	11.51%	11.72%	3.54%
	LZ77	0.93%	1.28%	1.07%	0.13%	-	-	-	-
	LZ78	0.41%	1.54%	0.8%	0.23%	-	-	-	-
	LZW	2.64%	1.29%	0.98%	0.22%	20.1%	11.1%	11.81%	3.64%

send a byte in SF12 is considerably longer than in SF7, it has a low impact on the gain.

Although there are no significant energy gains in SF7, there is a gain in network throughput because, in one of the scenarios, it is possible to send up to 21 messages in a single packet using the compression algorithm. However, it is important that systems that use the compression approach are delay tolerant. That is, it will be necessary to wait for 21 measurements to send the messages in a single LoRaWAN packet. In this sense, considering an interval of 15 seconds to send messages, if using LZW, it would be necessary to wait 315 seconds to send a packet.

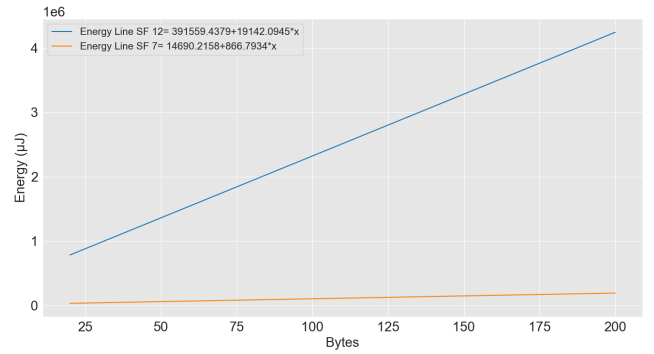
5 Energy Consumption Model

To estimate the gain in the energy autonomy of the device considering the data compression, an analytic model of energy consumption can be proposed. The device has two different modes of operation: continuous and deep sleep. Thus, a model for each mode of operation was developed.

The energy consumption of the device in continuous mode can be estimated considering three items: energy spent in the transmission of a packet, energy spent in the compression and energy spent in the idle mode. The energy values spent by the algorithm were determined in scenario 2 of each case, and the current value during the execution was close to 50.9 mA; with this, it is possible to calculate the energy spent for this item. To determine the cost of sending packets, an experiment was performed to send messages from 20 to 200 bytes in SF12 and 7. Through the experiment, it was observed that the amount spent transmitting is linear to the number of bytes sent, as shown in Figure 6. Given its linear behaviour, it is possible to obtain by first-order linear regression an estimate of the energy consumption as a function of the bytes sent. The estimates for the two SFs are presented in Equation 3 in micro Joule.

$$\begin{vmatrix} SF12 \\ SF7 \end{vmatrix} = \begin{vmatrix} 391559.4379 & 19142.09454 \\ 14690.21583 & 866.7934129 \end{vmatrix} * \begin{vmatrix} 1 \\ Bytes \end{vmatrix} \quad (3)$$

The energy consumption in deep sleep mode can be estimated considering four aspects: energy consumed in deep sleep, energy consumed when turning on the device, transmission energy and compression energy. This is shown in Figure 7, displaying the energy curve of the device in deep

**Figure 6.** Energy consumption curve SF7 and SF12.

sleep. The value in mA during deep sleep is constant, close to 10.9 mA. Using the equations obtained in each situation, the energy model used to estimate the lifetime of a device is represented by Equation 4.

$$E_{Total} = E_{Mode}(T, n) + \sum_{i=1}^n E_{Algo}(size_i) + \sum_{i=1}^n E_{SF}(size_i) \quad (4)$$

E_{mode} represents the energy consumption of the device while waiting for the next transmit window and depends on the operating mode (DeepSleep or Continuous). The energy consumption is a function of the sampling time (T) and the number of messages to be sent (n). Table 6 shows the values that are used to estimate the useful life gain of the device. Some values are in power because they depend on the time (T) that will be used to estimate the energy.

E_{Algo} represents the energy spent to compress the message and depends on the algorithm to be used (Equation 1) and the message *size*. E_{SF} represents the energy spent to send a packet and depends on the SF chosen (Equation 3) and the *size* of the message to be sent.

Table 6. Energy metric

	Value	Unit
Energy to wake up from deep sleep	0.00305	J
Power in deep sleep mode	0.0545	W
Power in normal mode	0.2545	W

To simulate the useful life gain using the models obtained, the use of a Li-Ion 18650 battery with a capacity of 2,200 mAH was considered. However, the battery supply voltage is 3.7 V, and it is necessary to use a boost converter to raise

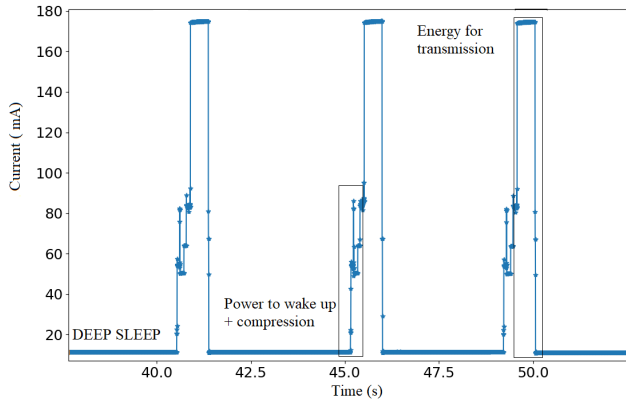


Figure 7. Energy curve in deep sleep mode.

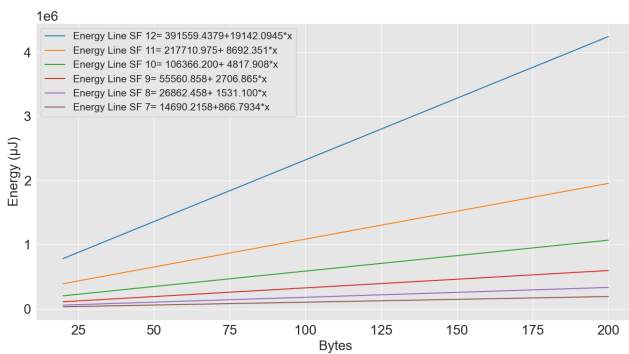


Figure 8. Energy consumption curve from SF7 to SF12.

the operating voltage to 5 V. The considered converter has an efficiency of 93%. Through the model, it was found that with compression, the useful life increased by more than 5 days considering the deep sleep mode and SF12 mode, going from 21 days without compression to 26 days.

Considering the models obtained, the gain values of scenario 3 were compared. As a result, a mean difference between the measurement in the experiment and the analytical model of 0.7% was obtained. It is worth noting that distortions such as noise and variation of the measured values are not included in the model, in addition to distortions caused by the linear approximations of the analytical model.

To analyze the energy gains using compression for the other SF (8, 9, 10, 11), energy consumption curves were generated for each SF and applied to the model (Equation 4). Figure 8 presents the energy consumption curves for all SFs, and the methodology to obtain the curves for the SF8 to SF11 was the same as described at the beginning of this section to obtain the curves for the SF7 and SF12. In the analytical model, for SF 8, 9, 10, and 11, the maximum size allowed per packet for each SF (Table 2) was used as the limit to determine the number of messages transmitted.

Figures 9 and 10 show the gains for the different transmission interval scenarios using the different compression algorithms. In Figures 9, and 10 it can be seen that the gains are less significant for SF (7, 8, 9), but from SF10 the gains are larger. The reason for this behavior is that the energy cost of sending the data from SF10 is greater than that of compressing it.

The analytical model revealed that the algorithms that resulted in a higher compression rate obtained the highest en-

ergy gain. The justification for this finding is that the consumption to send a bit is considerably higher than the compression cost. Thus, considering SF12, the algorithms obtained close gains since they mostly sent the same number of bytes. Considering SF7, the best results were obtained using Huffman and LZW because they had the best compression rates.

6 Conclusion

This article presents the results obtained with data compression applied to LoRa networks. Algorithms based on statistics and dictionaries were used. From the data of real applications, a reduction in energy consumption and traffic on the network was obtained. This study also developed an analytical model for energy consumption when using data compression.

The LZW algorithm had the highest compression rate in both scenarios, with values of 69% (temperature) and 63% (GPS) and an energy reduction of approximately 22%. It was found that the LZW and LZ78 algorithms used more heap memory and that this considerably impacts the compression time and energy consumption of the algorithm. In most cases, the algorithms that used more heap memory had a higher compression rate. However, the rate and use of memory heap are not related. According to the analysis of the compression rate, the compression rate is intrinsically related to the data.

It was also observed that the algorithms based on statistics (Huffman and Arithmetic), have greater stability in the compression rate than those based on a dictionary, and this stability becomes greater as the rate increases. In addition, it was observed that depending on the location of the memory accessed by the algorithm (static or heap), the size of the compressed data can lead to a considerable increase in energy consumption.

As future work, we propose the use of static probabilities on Huffman and Arithmetic algorithms, to avoid the sending of the probability table in the message. For the dictionary algorithms, such as LZ77, LZ78 and LZW, a study can be carried out to evaluate the impact of different dictionary sizes on the compression rate and execution time, and mainly in the energy consumption due to the increase of memory access in bigger dictionaries. Thanks to the compression algorithms, the number of messages in the LoRa network decreases. Thus, we also propose to address the scalability and throughput gain in the LoRa network and the impact in the goodput, obtained with the compression of the messages using simulators.

Acknowledgment

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001

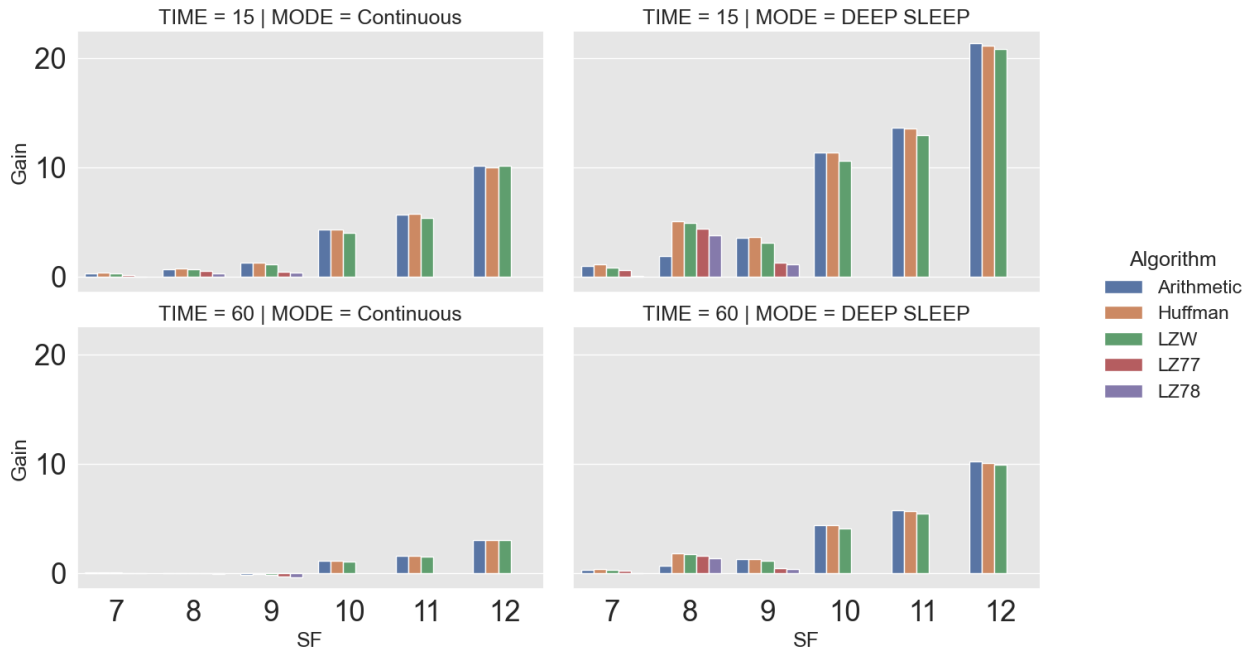


Figure 9. Energy gains (%) - Temperature

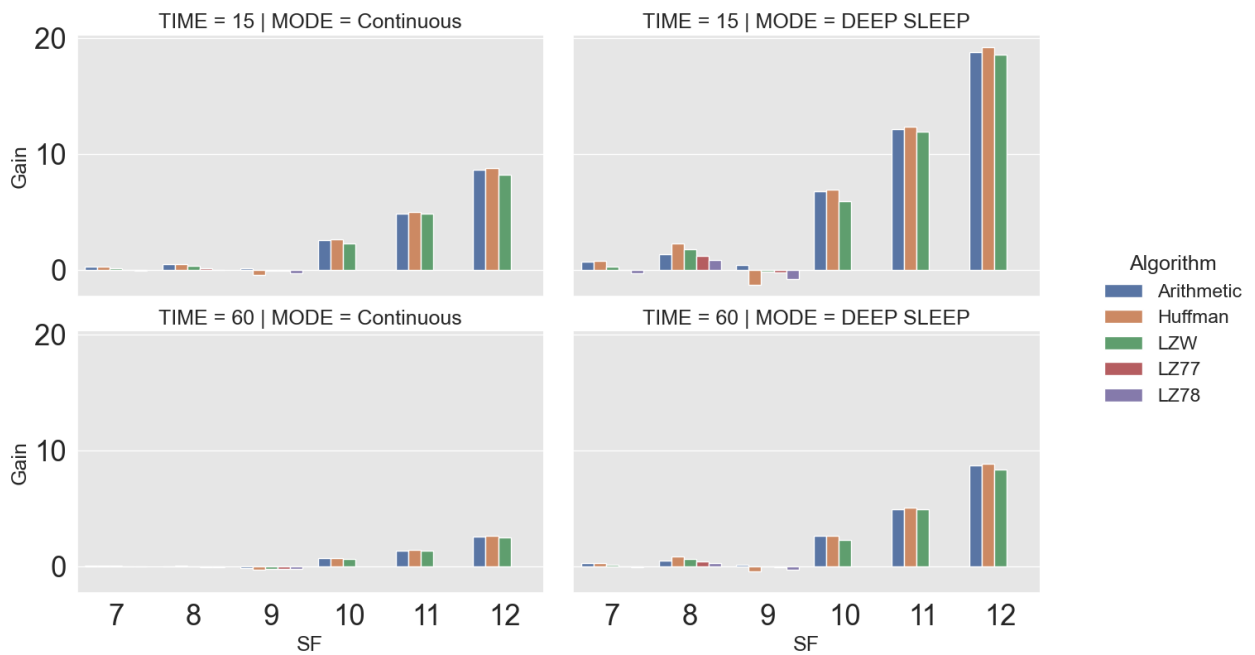


Figure 10. Energy gains (%) - GPS

Declarations

Authors' Contributions

Javan Ataide de Oliveira Junior: Developed the code; Designed the Experiments; Executed the experiments; Analysed the results; and wrote the first draft. Edson Tavares de Camargo: Analysed the data; Revised the manuscript; Rewrote parts of the text. Marcio Seiji Oyama: Supervised the whole work; Designed the research; Revised the manuscript; Rewrote parts of the text and Validate the final results. All authors contributed to the writing of this article, read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Availability of data and materials

Data can be made available upon request.

References

Ahmar, A. U. H., Aras, E., Joosen, W., and Hughes, D. (2019). Towards more scalable and secure lpwan networks

- using cryptographic frequency hopping. *IEEE Computer Society*. DOI: 10.1109/WD.2019.8734249.
- Al-kadhim, H. M., Al-raweshidy, H. S., and Member, S. (2021). in cloud based iot. 21:12212–12219.
- Aras, E., Ramachandran, G. S., Lawrence, P., and Hughes, D. (2017). Exploring the security vulnerabilities of lora. Institute of Electrical and Electronics Engineers Inc.. DOI: 10.1109/CYBConf.2017.7985777.
- Bor, M., Vidler, J., and Roedig, U. (2016). LoRa for the Internet of Things. *Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks*, pages 361–366. Available at: <https://eprints.lanacs.ac.uk/id/eprint/77615/>.
- Camargo, E. T., Spanhol, F. A., and Castro e Souza, A. R. (2021). Deployment of a LoRaWAN network and evaluation of tracking devices in the context of smart cities. *Journal of Internet Services and Applications*, 12(8):1–24. DOI: 10.1186/s13174-021-00138-7.
- Gu, F., Niu, J., Jiang, L., Liu, X., and Atiquzzaman, M. (2020). Survey of the low power wide area network technologies. *Journal of Network and Computer Applications*, 149:102459. DOI: 10.1016/j.jnca.2019.102459.
- Hanumanthaiah, A., Gopinath, A., Arun, C., Hariharan, B., and Murugan, R. (2019). Comparison of Lossless Data Compression Techniques in Low-Cost Low-Power (LCLP) IoT Systems. *Proceedings of the 2019 International Symposium on Embedded Computing and System Design, ISED 2019*, pages 63–67. DOI: 10.1109/ISED48680.2019.9096229.
- Le, T. L. and Vo, M. H. (2018). Lossless data compression algorithm to save energy in wireless sensor network. *Proceedings 2018 4th International Conference on Green Technology and Sustainable Development, GTSD 2018*, pages 597–600. DOI: 10.1109/GTSD.2018.8595614.
- Lilygo (2019). Lilygo TTGO LoRa development board. Available at: <http://www.lilygo.cn/>.
- LoRa Alliance (2021). LoRaWAN Regional Parameters RP002-1.0.3. Available at: https://lora-alliance.org/resource_hub/rp2-1-0-3-lorawan-regional-parameters/.
- Marcelloni, F. and Vecchio, M. (2008). A simple algorithm for data compression in wireless sensor networks. *IEEE Communications Letters*, 12(6):411–413. DOI: 10.1109/LCOMM.2008.080300.
- Maurya, A. K. and Singh, D. (2011). Median predictor based data compression algorithm for wireless sensor network. *International Journal of Computer Applications*, 24. DOI: 10.5120/2961-3940.
- McAnlis, C. and Haecky, A. (2016). *Understanding Compression: Data Compression for Modern Developers*. O’Reilly Media, Inc., 1st edition. Book.
- Mishra, M., Sen Gupta, G., and Gui, X. (2022). Investigation of energy cost of data compression algorithms in WSN for IoT applications. *Sensors*, 22(19). DOI: 10.3390/s22197685.
- Nelson, M. and Gailly, J.-L. (1996). *The Data Compression Book (2Nd Ed.)*. MIS:Press, New York, NY, USA. Available at: <http://www.hti.edu.eg/academic-files/English/2435.pdf>.
- Perera, C., Liu, C. H., Jayawardena, S., and Chen, M. (2015). A Survey on Internet of Things from Industrial Market Perspective. *IEEE Access*, 2:1660–1679. DOI: 10.1109/ACCESS.2015.2389854.
- Sacaleanu, D. I., Popescu, R., Manciu, I. P., and Perișoară, L. A. (2018). Data compression in wireless sensor nodes with LoRa. In *2018 10th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, pages 1–4. DOI: 10.1109/ECAI.2018.8679003.
- Sadler, C. M. and Martonosi, M. (2006). Data compression algorithms for energy-constrained devices in delay tolerant networks. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems, SenSys ’06*, pages 265–278, New York, NY, USA. ACM. DOI: 10.1145/1182807.1182834.
- Samie, F., Tsoutsouras, V., Bauer, L., Xydis, S., Soudris, D., and Henkel, J. (2017). Computation offloading and resource allocation for low-power IoT edge devices. *2016 IEEE 3rd World Forum on Internet of Things, WF-IoT 2016*, pages 7–12. DOI: 10.1109/WF-IoT.2016.7845499.
- Sayood, K. (2005). *Introduction to Data Compression, Third Edition (Morgan Kaufmann Series in Multimedia Information and Systems)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. Book.
- Semtech Corporation (2015). LoRa Modulation Basics. Available online at: <https://www.frugalprototype.com/wp-content/uploads/2016/08/an1200.22.pdf>.
- Semtech Corporation (2019). Understanding The LoRa Adaptive Data Rate. Available at: <https://lora-developers.semtech.com/library/tech-papers-and-guides/understanding-adr/>.
- Semtech Corporation (2023). What are LoRa and LoRaWAN. Available at: <https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan/>.
- Sinha, R. S., Wei, Y., and Hwang, S.-H. (2017). A survey on LPWA technology: LoRa and NB-IoT. *ICT Express*, 3(1):14–21. DOI: 10.1016/j.icte.2017.03.004.
- Statista (2020). Forecast end-user spending on IoT solutions worldwide from 2017 to 2025. Available at: <https://www.statista.com/statistics/976313/global-iot-market-size/>.
- Tasaka, S., Ikari, T., Kaneko, H., Iijima, Y., Yoshino, R., and Tanaka, M. S. (2019). Study of a bus location system with LoRa in Nonochi city. *2019 IEEE 8th Global Conference on Consumer Electronics, GCCE 2019*, pages 58–59. DOI: 10.1109/GCCE46687.2019.9015321.
- TI (2023). INA219 output current/voltage/power monitor. Available at: <https://www.ti.com/product/INA219>.
- Vander Byl, A., Neilson, R., and Wilkinson, R. (2009). An evaluation of compression techniques for wireless sensor networks. pages 1–6. DOI: 10.1109/AFRCON.2009.5308078.
- Welch, T. (1984). A technique for high-performance data compression. *Computer*, 17(6):8–19. DOI: 10.1109/MC.1984.1659158.