# A Suggested Lightweight Lossless Compression Approach for Internet of Everything Devices

**Mohammed Salih Mahdi**
*BIT Dept, Business Information College, University of Information Technology and Communications, Baghdad, Iraq*
mohammed.salih@uoitc.edu.iq
**Nidaa Flaih Hassan**
*Computer science Dept. University of Technology, Baghdad, Iraq*
110020@uotechnology.edu.iq

**ARTICLE INFO**
**Submission date**:27/10/2017
**Acceptance date**:4/3/2018
**Publication date**:14/10/2018

## Abstract

Limit storage space, high traffic sensor data transfer and power efficient transmission are samples of the challenging issues in the development of Internet of Everything (IoE) apps. This paper tackles these issues by presenting a suggested lossless compression approach according to lightweight operations. The suggested approach is working efficiently even with a low-performance equipment. Furthermore, enhancing the sensor node effectively of IoE by minimizing energy exhaustion and resource utilizing. Hence, provision power and expanding the age of IoE devices. The suggested approach is evaluated by using two datasets as a benchmark by calculating compression ratio firstly, on messages between person to person and secondly, on healthcare sensors (HeartRate and Body Temperature) between machine to person pattern of IoE. In two tests, the suggested approach may obtain a significant compression ratio.

**Keywords:** lossless compression, dictionary table, IoT, IoE

## 1. Introduction

Internet of Things (IoT) is a modern standard, which is swiftly obtained scope by combining sets of models and communication. The main concept of the IoT is the exist a diversity of objects like emergency mechanisms (Bluetooth, RIFD, sensors) and mobile phones, etc. IoT has eligible to communicate with another mechanism to attain the aims during transmission across the Internet. The native IoT sight surrounds a structure which is utilizing Internet to the specified and monitored objects[1][2]. Wireless Sensor Network (WSN) is an essential coring mechanism of IoT combines a number of spatially disseminated separated sensors into net and exchange their data during links [3].

Intranet has been expanded into the Internet. This expansion is referred to access IoT[4]. Things are 1st layer in Internet, IoT pushes across combining everything into Internet root, and this is referred to Internet of Everything (IoE). Whole processes far reachable during internet, hence, Combining of everything is encouraged by the business desires[5]. Everything indicates to refrigerator, vehicle, persons, sensors and fundamentally anything, which has an ability Internet link. IoE is a novel Internet idea that attempts to link everything which may be linked to the

Internet [6]. When utilizing IoE, Numerous firms have highly incomes because IoE has an effect on growth of job capacity. IoE consists of three patterns of links:

- The 1st pattern is machine to machine which is regarded to security. Notable instances of this pattern are catastrophe warning and martial security.
- The 2nd pattern is person to machine which fetches growing house automation model. Notable instances of this pattern are catastrophe reply, patient surveillance, smart garage, etc.
- The 3rd pattern is person to person which is regarded to people relations. Notable instances of this pattern are social networks, network education, chat, and telemedicine.

In this paper, the 2nd pattern and the 3rd pattern will be used.

IoE acts many orientation like cloud computing, mobile computing, data analysis [7]. At the moment, Sensors are all over the places of IoE apps. IoE apps infiltrate into people body, vehicles, healthcare, Agriculture, etc. Mammoth size of sensors produce high size of information with possibility subversive achievement and hazard in information storing and transition. These criteria impacts minifying duration of battery life and large procedure power[8], [9].

With a view to reaper IoE apps is to be expanded compression procedure and suggested lightweight data compression for IoE sensor.

Data Compression is expressed as transformation of data into a small number of individual bits as an alternative of native data representation [10] Data compression is an approach utilized to minify the recurrence in the appearance of the data. It assists to adequate more files inside a restricted amount of size of the storage medium by minimizing files.

Furthermore, in the data transference, it displays an excellent way to use the bandwidth of the connection and thus minify the communication cost[11], [12]. Data compression is mainly labeled into two kinds, if compressed data may be coming back to the proper data form precisely, this kind recognized as lossless compression. Otherwise, the other kind recognized as lossy compression due to compressed data may not come back to the proper data form [13]. The scope of this paper only focuses on lossless compression.

The lossless compression utilizes particular manner for a rebuilding the precise shape of data from the result of compressed data. It applied in sensitive applications where missing an individual bit may lead to a harmful failure like text, data of sensor record in database and execution files [14], [15]. In lossless compression, not all text is potential to be briefed. Hence, the execution of lossless compression relies on the features of native data[16]. Lossless compression has been acted on the claim that the data has to be storage completely [17].

In the lossless compression, many various theoretical procedures are existed. The 1st procedure points to the likelihood distribution of the input stream are not importantly calculated. It is called a dictionary depended algorithms. Examples of this procedure are LZW and LZ77 [18]. It works by finding corresponds between input stream that is needed to compress and a group of words that comprised in the dictionary. The compressor discovers like to correspond, it replaces a reference to word's indexing in dictionary[19].

In contrast, the 2nd procedure is relying on the likelihood distribution of the input stream prior compression process, it is called entropy depended algorithms. Examples of this procedure are Run Length Encoding and Huffman Coding [18] .

For checking the performance of suggested approach ,Compression Ratio (CR) is utilized, which indicates the volume of the distinction between Compressed Data (CD) and Native Data (ND) as Eq.1 [20], [21].

$$\mathrm{CR} = (1 - \frac{CD}{ND}) * 100\% \qquad \text{(Eq.1)}$$

There are various suggested procedures have been implemented to enhance the efficiency of the sensor nodes of WSN, IoT by minify the sending input stream volume utilizing the lossless compression procedures.

[22] suggested a lossless compression procedure for small input stream on embedded devices depend on context and arithmetic coding. To implement a compression procedure, the suggested procedure needs former knowing of English input features. In addition, the suggested procedure is intended for input stream which are larger than a fifty - bytes. In [23] get a superior compression ratio by offering lossless compression procedure by adjusting Adaptive Huffman coding. Furthermore, the adjusting algorithm is effective and appropriate to job with different WSN nodes. In spite of, the adjusting algorithm is not eligible to compress input data which has minimum correlation. In [24] suggested lossless compression procedure to specialize issues of energy effective transportation in IoT equipments with two modification structure depend on an idea of the shaft the tables. These modifications make improvements in performance of IoT equipment statistically important. The suggested procedure is checked on air and surface temperatures, solar radiation and relative humidity datasets which are computed by the sensor. Hence, preventing waste in power of IoT equipments.

Concerning, the former literature review describes above, the contribution of this research displays a suggested algorithm for lossless compression of IoE devices to allow preventing waste of power, and thus extending the age of the sensor nodes in IoT Devices.

## 2. Material and method

The suggested approach is a lossless compression according to dynamic dictionary table work with IOE devices. The basic thought of the suggested approach is compressing not only the text (messages, health care sensors), but also produces the dynamic dictionary table. The major benefit of the suggested approach are capable to compress the data without depending on the idea of character repetition, in addition, it works adeptly even with a small size of the text. The suggested approach is formed by a simple process in both dynamic dictionary constructing and data compression.

The following steps to describe the suggested approach compression process:

- ***Preprocessing:*** The first step in the suggested approach is to process the text, specially in person to person pattern of IoE pattern. Usually in a text file, approximately 5 % to 15 % of symbols are blanks (spaces). In this step, minimizing these blanks will consequence a text file with the compression ratio. So, these blanks are compressed by analyzing phrases in a text file and finding is scarce to have a word that begins with a small character and ends with a capital character. Using this concept, the blanks between the phrases are compressed by changing the last small character of a word and a followed blank with a respective capital character. This changing is ignored, either the whole word is in capital characters or the length of the word is two characters and begins with a capital character. In addition, the changing is ignored if the word contains one character. For example, the text file contains two statements "**My name is Mohamed and I am an Iraqi student"** and **"A dog is on the street**". In the compress operation of text preprocessing step minimizing blanks in the text file by changing it to **"My namEis MohameDand I aMaNiraqIstudent"** and **"A doGiSoNthEstreet"**. In the decompression operation of text preprocessing step, it is done by encounters a capital character, if a former character is small. Consequently, it is increased in size into two letters, the first is the respective small character and the second is blank. Finally, the preprocessing step is optional, especially when there are blanks in the input text.
- ***Splitting:*** Read the data letter by letter from the output of text preprocessing step. These letters converted to ASCII codes, multiplied each code by 0.1 (divided each code by ten). Taking the integer number of output of the division code to create a dictionary table as first part and taking the remainder from the output of the division code as a second part
- ***Dynamic dictionary table:*** Creating a dynamic dictionary table by converting number digit part (first part) of each code that are extracted in splitting step in a decimal form and save it in the dictionary table (DT) without repetition. Order the value of DT in ascending order. For each value in DT, allocating an index value starting from 0. Change the decimal form with its particular index in dictionary table. Dictionary table is dynamic due to the altered of the size and the value of elements of the dictionary table in each session of compression. Figure (1) shows creating dictionary table**.** The maximum dynamic dictionary table decimal form ranged from 0 to 25 and cannot exceed 25, because the maximum value of the byte is (255). Therefore, the maximum value will be 25.

**Algorithm (1):** Create Dictionary Table

**Input:** Text file

**Output:** Dictionary Table (DT) , Bytes array of scanned text file (S)

**Begin**
**Step$_1$:** Read text file and store its bytes in array S
**Step$_2$:** i=0
**Step$_3$:** item= ASCII(S[i]) // Tacking ASCII code for bytes in array S
**Step$_4$:** m= item *0.1
**Step$_5$:** if m is not exist in in DT Then Add m to DT
**Step$_6$:** i=i+1
**Step$_7$:** if i less than or equal to the length of array S then Goto **Step$_3$**
**Step$_8$:** Sort DT ascending
**End**

**Figure (1): Suggested Approach (Create Dictionary Table Part)**

- ***Compressed Data:*** Integrating the indexed of first part with the second part of the splitting step to create modified code form (MCF). Discovering largest number in these MCF numbers and calculating the number of bits required to represent the max value of it. Computing the minimum value in MCF, called it min- MCF, and subtract each value of MCF with min- MCF. Figure (2) shows the pseudo code of compressed data**.** The compressed data include the following three fields:
  1. Five bits act the number of bits used for min- MCF
  2. Three bits act the maximum number of bits used for each character in MCF
  3. Binary representation for each MCF numbers.

**Algorithm (2):** Compressed Data

**Input:** Dictionary Table (DT) , Bytes array of scanned text file (S)

**Output:** Compressed Data

**Begin**
**Step$_1$:** i=0
**Step$_2$:** v= ASCII(S[i]) // Tacking ASCII code for bytes in array S
**Step$_3$:** r = v mod 10
**Step$_4$:** z = index of ( v * 0.1 ) in DT
**Step$_5$:** S[i] = (z *10)+r
**Step$_6$:** i=i+1
**Step$_7$:** if i less than or equal to the length of array S then Goto **Step$_2$**
**Step$_8$:** calculate the minimum value (Min) and maximum (Max) in S
**Step$_9$:** Subtract each value in S by Min
**Step$_{10}$:** Compute the length of bits for max (Bitn_Max) and the length of bits for min (Bitn_Min)
**Step$_{11}$:** Compress=""
**Step$_{12}$:** i=0
**Step$_{13}$:** Compress= Compress+ S[i] $_{2,\ Bitn\ Max}$
**Step$_{14}$:** i=i+1
**Step$_{15}$:** if j less than or equal to the length of array S then Goto **Step$_{13}$**
**Step$_{16}$:** Compressed Data= Bitn_Min + Bitn_Max + Compress
**End**

**Figure (2): Suggested Approach (Compressed Data Part)**

- **Compress Dictionary Table:** The dictionary table is compressed by subtracting the particular item from $TD_i$ by its direct predecessor such as the followings:

$$CDT[0] = DT[0] \quad \text{(Eq.2)}$$

$$CDT[i] = DT[i] - DT[i-1] \text{ for } i>=1 \quad \text{(Eq.3)}$$

Discover the maximum value in Compressed Dictionary Table (CDT), called it Max value, and for representation max value in bits called it bit-max-value. The Max value cannot override 25. Thus, for representation requiring at the most five bits. The bit-max-value is less or equal to five. In the subsequent, if the number of bits to represents CDT [i] is lower than or same to (bit-max-value / 2). Placing '0' at the starting then putting the binary representation of CDT [i] in (bit-max-value / 2) bits. Otherwise, placing '1' then putting the binary representation of CDT [i] in (bit-max-value). Figure (3) shows the pseudo code of the compressed dictionary table**.** The compressed dictionary table comprises the following three points:

1. Five bits to act the number of values in CDT.
2. Five bits to act first value in CDT
3. Binary representation for each value from CDT [i]) bits.

**Algorithm (3):** Compressed Dictionary

**Input:** Dictionary Table (DT) from Algorithm 1

**Output:** Compressed Dictionary

**Begin**
**Step$_1$**: CDT [0] = DT[0] // CDT: Compressed Dictionary Table
**Step$_2$**: i=0
**Step$_3$**: CDT [i] = DT [i] - DT [ i -1]
**Step$_4$**: i=i+1
**Step$_5$**: if i less than or equal to the length of (DT) then Goto **Step$_3$**
**Step$_6$**: CD=""
**Step$_7$**: i=0
**Step$_8$**: Calculate the maximum (Max) in CDT
**Step$_9$**: Compute length of bits for max (Bitn_Max)
**Step$_{10}$**: if Bitn_CDT[ i] less than or equal to (Bitn_Max_CDT /2) then
CD = CD + "0" + CDT [ j]$_{2,\ (Bitn\_Max\_CDT\ /2)}$
else
CD = CD + "1" + CDT [ j]$_{2,\ (Bitn\_Max\_CDT\ /2)}$
**Step$_{11}$**: i=i+1
**Step$_{12}$**: if i less than or equal to the length of (CDT) then Goto **Step$_{10}$**
**Step$_{13}$**: Compressed Dictionary= length of CDT + CDT [0] + CD
**End**

**Figure (3): Suggested Approach (Compressed Dictionary Table Part)**

The decompressed package is identical steps of compression text but in reverse way. Figure (4) shows the pseudo code of the decompressed package (dictionary table and original data) .

**Algorithm (4): Suggested Decompression Algorithm**

**Input:** Compressed Package = compressed data + compressed dictionary

**Output:** Dictionary Table (DT),Original data

**Begin**

//Extracting the Dictionary Table (DT)

**Step$_1$**: Taking the first five bits from Compressed Package and convert it to decimal form, which is acting the number elements in the Dictionary Table.

**Step$_2$**: Create Dictionary Table then go to the next 5 bits and convert it to decimal format (max-element) and add it to the Dictionary Table. Compute length of bits for max-element (Bitn_Max).

**Step$_3$:** Check if the next binary value starts with (0) that means we need to take (Bitn_Max/2) bit after it, and if it starts with (1), we need to take (Bitn_Max) bits and convert it to decimal form and Add it to the Dictionary Table.

**Step$_4$:** Repeat steps (1 to 3) based on the number elements which is determined in **Step$_1$**

**Step$_5$:** Add second value in Dictionary Table to the first value, Add third value in Dictionary Table to the second value and so on.

//Extracting the Original data

**Step$_6$:** Now, take the next five bits and convert them to decimal form, which is acting max value (max value length of bits), Take the next max value bits and convert them to decimal form that is acting min value .Then added it to all next data after converting to decimal form.

**Step$_7$:** multiply each item by 0.1 and replace the integer part with its value in the Dictionary table. After using the integer part as an index, we need to multiply the value by 10 to return as integer(ASCII) and that equal to original data

**End**

**Figure (4): Suggested Approach (Decompressed package Part)**

## 3. Case Study

Suppose, the input data is a text file, which contains "hgfedcba". The following steps illustrated the suggested algorithm for compressing text files.

$S_1$: There are no blanks in the text file. Therefore, it is still the same.

$S_2$: Transform the output of step1 to its ASCII code form. Each item of ASCII code is divided by ten and partition to first part (FP) and second part (SP):

| Letter: | h | g | f | e | d | c | b | a |
|---|---|---|---|---|---|---|---|---|
| ASCII | 104 | 103 | 102 | 101 | 100 | 99 | 98 | 97 |
| ASCII*0.1 | 10.4 | 10.3 | 10.2 | 10.1 | 10.0 | 9.9 | 9.8 | 9.7 |
| FP | 10 | 10 | 10 | 10 | 10 | 9 | 9 | 9 |
| SP | 4 | 3 | 2 | 1 | 0 | 9 | 8 | 7 |

$S_3$: Taking the first part of step 2 without recurrence. Sorting these items and allocating each item an index begin from 0.

| FP | 9 | 10 |
|---|---|---|
| index | 0 | 1 |

$S_4$: Change the first part of step 2 with its index, and then combine with the values of the second part from step. Therefore, that means modified code form (MCF).

| FP | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|

| MCF | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 |
|---|---|---|---|---|---|---|---|---|

$S_5$: Calculate the lowest item in MCF, called it min-MCF that is represented by 7 and subtract each item of MCF with 7.

| MCF | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 |
|---|---|---|---|---|---|---|---|---|
| MCF | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

$S_6$: The maximum value from step 5 is 7, so items requiring three bits to represent it. Thus, at starting adding five bits (00111) that acts the number of bits, which used for min- MCF. Also, adding three bits (111) that acts the maximum number of bits that used for letters in MCF. The whole quantity of bits would be 5+3+ (3 * 8) = 32 bits. Later, binary form of bits that are used to act the compressed data.

Binary 00111 111 111 110 101 100 011 010 001

$S_7$: Compressing the dictionary table according to step 3. Occupying the 1st part as it and place in new table, then subtract each item from the former ancient item (as eq. 1)

| 9 | 10-9 |
|---|---|
| 9 | 1 |

$S_8$: Putting five-bits at the starting of the dictionary table to act the number of item in the dictionary (in this case study 2 items so it is 00010 in binary). Act the first item in five bits to be (01001). The real quantity of bits which is used for first item (9) is 4 bits so divided 4 by 2 and neglected the floating so will be (2). that's mean needing either five-bits for each item more than (2) or two-bits for each item less or equal than (2) plus one bit (0,1) as indicator

$S_9$: Now, act each next item as binary: $1= (1)_2$ less than (2) which is the integer value of step8. So it will be $(001)_2$. Repeat this step with all item and the definitive outcome of a dictionary table in binary be:

Binary 00010 01001 001 The total number of bits is 16 bits.

$S_{10}$: The compressed package is a mixture of bits, which it contains bits that are represent the dictionary table and data compressed. The Final compressed package bits are:

00010 01001 001 00111 111 111 110 101 100 011 010 001

$S_{11}$: So, the entire quantities of bits for text file and compression ratio.

Compressed Package = compressed data + compressed dictionary

Compressed Package =32+ 13= 45 bits

Compression ratio = 100 * (1 - (Compressed Package / original text file))

Compression ratio = 100 * (1- (45/64)) = 30%

The decompressed package is identical steps of compression text but in reverse way as following steps

$S_1$: Assume, the input for decompressed package is 00010 01001 001 00111 111 111 110 101 100 011 010 001 000 from above steps.

$S_2$: Bring the initial five bits (00010) and transform it to decimal format. 00010 → 2, so, it refer to number of elements in dictionary table.

$S_3$: Create a dictionary table, bring the subsequent five bits (01001) and transform it to decimal format. 01001 → 9. Add 9 to the dictionary table.

9 require 4bits to act it. Check the following bit if it begins with (0) led to get (4 / 2 ) bits after it, however, if it begins with (1) led to get (4) bits.

$S_4$: Discover '0' led to bring two bits (01). So, 01→1 and add 1 to the dictionary table. the final dictionary table values are: 9 , 1

$S_5$: Modify the second element of dictionary table by following formula second element = second element +first element
9 , 1+9 → 9 , 10

$S_6$: Bring the subsequent 5 bits (00111) and transform them to decimal format. 00111 --> 7 which is acting max-element (max-element length of bits is 3 bits). Bring the following 3 bits 111 and 111 --> 7 , which is acting min-element .Then added min-element to all following data stream after transformation to decimal format.
111 110 101 100 011 010 001 000 → 7 6 5 4 3 2 1 0 → 7+7 6+7 5+7 4+7 3+7 2+7 1+7 0+7 → 14 13 12 11 10 9 8 7

$S_7$: Multiply each element by 0.1 and substitute the integer portion with its element in dictionary table.in accordance with utilizing the integer portion as an indicator(index), Getting ASCII by multiply the elements by 10
14 → 14/10 → 1.4 →(table[1] = 10) → 10.4 * 10 = 104=h
13 → 13/10 → 1.3 →(table[1] = 10) → 10.3 * 10 = 103=g
12 → 12/10 → 1.2 →(table[1] = 10) → 10.2 * 10 = 102=f
11 → 111/10 → 1.1 →(table[1] = 10) → 10.1 * 10 = 101=e
10 → 10/10 → 1.0 →(table[1] = 10) → 10.0 * 10 = 100=d
9 → 9/10 → 0.9 →(table[0] = 9) → 9.9 * 10 = 99=c
8 → 8/10 → 0.8 →(table[0] = 9) → 9.8 * 10 = 98=b
7 → 7/10 → 0.7 →(table[0] = 9) → 9.7 * 10 = 97=a
And its equal to **hgfedcba** which is refers to the original data

## 4. Results of Suggested Approach

The suggested approach is implemented utilizing C++ Arduino microcontroller; two tests are utilized as a benchmark. The 1st test measures compression ratio of messages, which are between person to person patterns of IoE. The 2nd test measures compression ratio of healthcare sensors (HeartRate and Body Temperature) which are between machines to person pattern of IoE. Actually, with the two tests, messages test includes higher recurring values than healthcare sensors test. For suggested approach, Table (1) and Table (2) show the effect of compression for person to person and machine to person pattern of IoE respectively.

In these tables, notice that, suggested approach at 8 bytes begins with a 14.06 % reduction for person to person pattern and 10.39 % reduction in machine to person pattern. Then it illustrates a sudden enhancement of its evaluation and its value appears to settle outputs.

**Table 1: The Effect of Compression for person to person pattern of IoE**

| Input Data Size | Compression Ratio (%) |
|---|---|
| 8 bytes | 14.06 |
| 16 bytes | 32.04 |
| 32 bytes | 41.01 |
| 64 bytes | 45.05 |
| 128 byte | 47.75 |
| 256 bytes | 48.85 |
| 512 bytes | 49.43 |

**Table 2: The Effect of Compression for machine to person pattern of IoE**

| Input Data Size | Compression Ratio (%) |
|---|---|
| 8 bytes | 10.39 |
| 16 bytes | 24.21 |
| 32 bytes | 30.85 |
| 64 bytes | 34.17 |
| 128 byte | 35.39 |
| 256 bytes | 36.66 |
| 512 bytes | 37.08 |

## 5. Conclusion

This paper suggested lossless compression approach, according to dynamic dictionary table and lightweight procedure. From the concluding outcomes of implementing the suggested approach, which is improving the IoE efficiency in three layers. The 1st layer, enhanced the sensor node effectively of IoE by minify energy exhaustion and resource utilizing. Hence, provision power and expanding the age of IoE sensor. The 2nd layer, enhanced the net effectively by minifying the size of transmission of input stream without impacting the accuracy of the input stream when using suggested lossless compression. The 3rd layer, enhanced the processing effectively by minifying the calculation overhead on input stream. Hence, it works effectively even with a low-performance equipment.

**CONFLICT OF INTERESTS.**
There are non-conflicts of interest.

## References

[1] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," Comput. networks, vol. 54, no. 15, pp. 2787–2805, 2010.
[2] D. Giusto, A. Iera, G. Morabito, and L. Atzori, The internet of things: 20th Tyrrhenian workshop on digital communications. Springer Science & Business Media, 2010.
[3] ITU-T, ", 'Ubiquitous Sensor Networks (USN),'" ITU-T Technol. Watch Brief. Rep. Ser., vol. 4, 2008.
[4] D. Fitton, G. Kortuem, F. Kawsar, and V. Sundramoorthy, "Smart Objects as Building Blocks for the Internet of Things," Internet Comput. IEEE 14, pp. 44 – 51, 2010.
[5] A. J. Jara, L. Ladid, A. Skarmeta, I. Comsoc, and I. Etc, "The Internet of Everything

through IPv6 : An Analysis of Challenges , Solutions and Opportunities," Networks Ubiquitous Comput., pp. 97–118, 2013.
[6] A. Abdelwahab, S., Hamdaoui, B., Guizani, M., Rayes, "Enabling smart cloud services through remote sensing: An internet of everything enabler.," Internet Things Journal, IEEE, vol. 1, no. 3, pp. 276 – 288, 2014.
[7] X. Jordi, M. , George, M. and Constandinos, ",'Beyond the Internet of Things: Everything Interconnected', -," Springer Int. Publ. Jan 9, Technol. Eng., p. 408 pages., 2017.
[8] K. Hung, N. Jeung, H. and Aberer, "'An evaluation of model- based approaches to sensor data compression,'" IEEE Trans. Knowl. Data Eng., pp. 2434 – 2447, 2013.
[9] L. Misra, V. Goyal, V. and Varshney, "'Distributed scalar quantization for computing: High-resolution analysis and extensions,'" IEEE Trans. Inf. Theory, vol. 57, no. Aug, pp. 5298–5325, 2011.
[10] K. Ekhlas, "Text Compression & Encryption Method Based on RNA and MTF," Iraqi J. Sci., vol. 58, no. 2, pp. 1149–1158, 2017.
[11] S. Khalid, "Introduction to Data Compression," Third Edit. Elsevier, 2000.
[12] R. Waghulde, H. Gurjar, V. Dholakia, and G. P. Bhole, "New Data Compression Algorithm and its Comparative Study with Existing Techniques," Int. J. Comput. Appl., vol. 102, no. 7, 2014.
[13] S. Porwal, Y. Chaudhary, J. Joshi, and M. Jain, "Data compression methodologies for lossless data and comparison between algorithms," Int. J. Eng. Sci. Innov. Technol. Vol., vol. 2, pp. 142–147, 2013.
[14] N. Sharma, J. Kaur, and N. Kaur, "A review on various Lossless text data compression techniques," Res. Cell An Int. J. Eng. Sci., vol. 12, no. 2, pp. 58–63, 2014.
[15] A. S. Sidhu and M. Garg, "Research Paper on Text Data Compression Algorithm using Hybrid Approach," Int. J. Comput. Sci. Mob. Comput., vol. 3, no. 12, pp. 1–10, 2014.
[16] I. M. Pu, Fundamental data compression. Butterworth-Heinemann, 2005.
[17] B. Al-Himyari, "A Hybrid Compression Algorithm by Using Shannon-Fano Coding and Oring Bits," J. Kerbala Univ., vol. 6, no. 3, 2008.
[18] S. R. Kodituwakku and U. S. Amarasinghe, "Comparison of lossless data compression algorithms for text data," Indian J. Comput. Sci. Eng., vol. 1, no. 4, pp. 416–425, 2010.
[19] D. Salomon, Data compression: the complete reference. Springer Science & Business Media, 2004.
[20] M. S. Mahdi and N. F. Hassan, "A Proposed Lossy Image Compression based on Multiplication Table," Kurdistan J. Appl. Res., vol. 2, no. 3, pp. 98–102, 2017.
[21] A. J. Santoso, L. E. Nugroho, G. B. Suparta, and R. Hidayat, "Compression ratio and peak signal to noise ratio in grayscale image compression using wavelet," Int. J. Comput. Sci. Technol., vol. 2, no. 2, pp. 7–11, 2011.
[22] S. Rein, C. Gühmann, and F. Fitzek, "Compression of short text on embedded systems," J. Comput., vol. 1, no. 6, pp. 1–10, 2006.
[23] F. Marcelloni and M. Vecchio, "A Simple Algorithm for Data Compression in Wireless Sensor Networks," Commun. Lett. IEEE, vol. 12, no. 6, pp. 411–413, 2008.
[24] M. Vecchio, R. Giaffreda, and F. Marcelloni, "Adaptive Lossless Entropy Compressors for Tiny IoT Devices," vol. 13, no. 2, pp. 1088–1100, 2014.

## الخلاصة

محدودية مساحة التخزين، وارتفاع حركة مرور نقل بيانات المتحسس وكفاءة نقل الطاقة هي عينات من المشاكل الصعبة في تطوير تطبيقات إنترنت كل شيء. وقد تم في هذا البحث معالجه هذه المشاكل من خلال اقتراح طريقة ضغط بدون فقدان للبيانات، والتي تقوم على عمليات خفيفة بالتعقيد. الطريقة المقترحة تعمل بكفاءة حتى مع الاجهزة ذات الأداء المنخفض. وعلاوة على ذلك، تحسن أجهزة المتحسس بشكل فعال في إنترنت الكل شيء بواسطة التقليل من استهلاك الطاقة والموارد المستخدمة. وبالتالي، توفير الطاقة واطالة عمر أجهزة إنترنت الكل شيء. تم اختبار الطريقة المقترحة، على مجموعتين من البيانات كمعيار اساسي حسب نسبة الضغط المحسوبة على الرسائل بين نمط شخص الى شخص. بالإضافة إلى ذلك، نسبة ضغط على المتحسسات الطبية (دقات القلب ودرجة حرارة الجسم) بين نمط آلة والى شخص من إنترنت الكل شيء. في الاختبارين، الطريقة قد حصلت على مقياس ضغط كبير.


**الكلمات الدالة:** ضغط بدون فقدان للبيانات، جدول القاموس، إنترنت الاشياء، إنترنت الكل شيء.