Turkish Journal of Electrical Engineering & Computer Sciences
http://journals.tubitak.gov.tr/elektrik/
Research Article
Turk J Elec Eng & Comp Sci
(2021) 29: 32 – 45
© TÜBİTAK
doi:10.3906/elk-2003-114

# Internet of things data compression based on successive data grouping

**Samer SAWALHA**[1,*], **Ghazi AL-NAYMAT**[1,2]
[1]Department of Computer Science, King Hussein School of Computing Sciences,
Princess Sumaya University for Technology, Amman, Jordan
[2]Information Technology Department, College of Engineering and Information Technology, Ajman University,
Ajman, United Arab Emirates

**Received:** 22.03.2020 • **Accepted/Published Online:** 09.06.2020 • **Final Version:** 27.01.2021

**Abstract:** Internet of things (IoT) is a useful technology in different aspects, and it is widely used in many applications; however, this technology faces some major challenges which need to be solved, such as data management and energy saving. Sensors generate a huge amount of data that need to be transferred to other IoT layers in an efficient way to save the energy of the sensor because most of the energy is consumed in the data transmission process. Sensors usually use batteries to operate; thus, saving energy is very important because of the difficulty of replacing batteries of widely distributed sensors. Reducing the total amount of transmitted data from the perception layer to the network layer in the IoT architecture will save the energy of the sensor. This paper proposes a new IoT data compression method; it is based on grouping similar successive data together and sending them as one row with a total number of occurrences. The decision of similarity is done by comparing the root-mean-square successive difference calculated on the training dataset. The evaluation of the proposed method was performed on the Intel Lab dataset and the compression performance of the proposed method was compared with other compression methods, where a great enhancement was achieved; the compression ratio was 10.953 with a reconstruction of temperature data error equal to 0.0313 °C.

**Key words:** Wireless sensor networks, Internet of things, data compression, data grouping, energy consumption optimization

## 1. Introduction

The Internet of things (IoT) is considered as the future of the digitization of the economy and society; it is used in a number of applications, such as smart cities, health, and the automotive industry [1]. The sensors are connected through communication networks; this generates a wide range of connected devices [2]. These sensors can measure, sense, and extract information based on the interaction between these sensors with the surrounding environment, such as humans, animals, mechanical devices, and electrical devices [3].

The IoT sensors generate a massive amount of heterogeneous data (sensing, multimedia, control); this data can be analyzed, processed, and used to extract some statistical information, or to take decisions and actions [4]. However, IoT technology faces some challenges which need to be taken into consideration while using it; the main challenge is how to deal with the big data generated from these sensors, and dealing with IoT data includes storing, processing, and transferring them over the networks [1,5].

Having a huge amount of sensed data that need to be moved between different layers in the IoT system consumes a lot of energy; 80% of sensor power is consumed in the data transmission process [6]. The energy in

*Correspondence: samerfs@hotmail.com

the wireless sensor network is considered a very critical issue because of the cost and the difficulty of replacing batteries of widely spread sensors.

All of the above reasons represent the motivation behind this proposed approach, as well as many other research ideas, which targeted this problem by implementing different data compression methods. These methods are used to decrease the overall data transmitted over the networks; this leads to a decrease in the sensors' energy consumption [7,8]. Compressing the data before the transmission process also leads to saving the bandwidth of the network; thus, the available resources can be used efficiently [9].

There are two main categories of data compression algorithms; the first involves lossless data compression algorithms where the original data can be regenerated from the compressed data without losing any information (exactly the same as the original data), but the compression ratio is much lower than the lossy compression algorithms. The lossy compression algorithms can be used to generate data that represent the original data with some accuracy (not 100%), but some of the data are lost in the compression and decompression processes [10].

In this research, we targeted the sensors that generate numerical values from the IoT and propose a new lossy compression algorithm to compress this data with a high level of compression ratio and decompressed data accuracy. The compression is done by grouping and counting almost similar successive data together. Then, the data are decompressed based on some grouping information to regenerate almost the same data as the original data. Then, we compare our work with other famous lossy compression algorithms in terms of compression ratio and the accuracy of decompressed data.

The rest of the paper is organized as follows: Section 2 discusses some previous studies done on IoT lossy compression techniques. Section 3 details our proposed data compression algorithm. Section 4 shows the implementation, evaluation, and comparison process with other compression algorithms. Finally, Section 5 concludes the paper.

## 2. Previous studies

Many studies addressed IoT data compression and storage, because of the importance of reducing the storage size needed to store the huge amount of IoT data, or to reduce the total amount of data needed to be transferred from the sensor nodes to the processing and storage nodes, leading to a reduction in the energy consumption, transfer time, etc.

Compressed sensing (CS) is one of the famous techniques to compress sensor data that is done by representing the sensor data with a smaller number of measurements. These measurements contain some important information that is needed in the data reconstruction process to have high accuracy in comparing with the original data. The measurements include linear values that are lower than the Shannon Nyquist limit [11]. However, the problem is that compressed sensing needs a large memory size to store the random sampling operator when having large signal amplitudes. CS also cannot be used directly in large-scale applications, which need a huge number of complex computations to encode the signals; thus, it cannot be used without having an embedded processor inside the sensors, leading to increased energy consumption, which is a major challenge of IoT.

The standard restricted Boltzmann machine (standard RBM) is a model that uses deep learning to compress and classify data by learning and extracting the main features from this data. These extracted main features are used to represent the overall data. RBM is a probabilistic method for density over observed variables; it uses a set of hidden variables to represent the presence of features. All of the observed variables are related to all hidden variables using different parameters. Convolutional RBM uses the basic idea of standard RBM to detect objects in images; it uses shared weights to preserve the spatial structure of the images [12]. RBM uses a small number of parameters with a simple network structure; it does not use complex computations to encode and compress the sensed data; thus, it does not consume a huge amount of energy.

The stacked RBM auto encoder model (stacked RBM-AE) is a model used to compress the sensor data by combining deep learning theory with the standard RBM model. The stacked RBM-AE is separated into two main parts. The first involves the encoder, which contains four standard RBMs with different sizes that are used to compress the sensor data. Each standard RBM uses an undirected graph that has two layers. The first layer is the visual layer that is used as input from the training dataset. The other layer is the hidden layer, which is used to map the visual layer data. The model was pretrained using the contrastive divergence algorithm on the four standard RBMs, and the activation probability of the visual layers and the activation probability of the hidden layers were calculated before updating the model parameters by minimizing the total model loss value. Then, they retrained the model using the pretrained model with the back propagation algorithm, which is an algorithm that uses an artificial neural network to refine the model parameters. The second part involves the decoder, which flips the encoder part to reconstruct the original data from the compressed data [13].

Wu et al. [14] proposed a novel framework for data prediction and a compression technique; the proposed compression technique was a lossless compression technique, where the least mean square (LMS) dual prediction algorithm was used with an optimal step size by decreasing the mean square deviation in the data prediction phase. Then, principal component analysis (PCA) was used in the compression and recovery processes, which were performed on the cluster heads (CHs) and the sink layers. Sensor nodes measure values, these values are grouped into multiple clusters based on the physical locations of the sensors. Dual prediction mechanisms using LMS with optimal step size which are implemented on the sensor nodes and their CHs will improve the prediction accuracy and increase the convergence speed. CHs extract the PCA of the collected value, so the redundant data will be eliminated. Finally, the data will be recovered at the base station. The proposed framework controls the errors. Abu Alsheikh et al. [15] proposed a new lossy compression method for reducing the total amount of data to be transferred by using machine learning algorithms; a neural network method was used to predict the human and environmental activities, before exploiting the nonlinear spatial-temporal correlation in the network to reduce the overall error. They also used an adaptive rate distortion to balance the data rate in the compression process. A low-cost data compression and decompression is achieved by implementing linear and sigmoidal operations, this data will be fed to machine learning algorithms to predict human activities and environmental conditions. Their framework introduced a level of security as an offline learned decompression dictionary which is needed to recover the data. Incebacak et al. [16] proposed a novel mathematical programming framework including mixed integer programming and linear programming (LP) models to compress the data as a lossy compression method, whereby the proposed programming framework was used to analyze the benefit of using various compression techniques in a way to increase the lifetime of the wireless sensor network. They provided contextual privacy against adversaries. They investigated the effect of optimal single-level compression and limited compression strategies through mixed integer programming models. Besides, they explored the impact of node density and limited transmission range due to contextual privacy scenarios. Liang et al. [17] proposed a lossless compression technique named sequential lossless entropy compression (S-LEC), on the LEC, modified by adding a sequential coding technique to enhance the compression performance. The proposed algorithm addressed the weakness of LEC in terms of the weakness of the lack of robustness. Also, it achieved highly robust compression performance for various sensor data streams. They devised the sequential LEC to exploit valuable sequential context information among adjacent residues for the data. It provided additional sequential code into its codewords. Another study by Ruxanayasmin et al. [18] used Lempel–Ziv–Welch (LZW) coding to decrease the overall energy consumption, which increased the network lifetime; the generated data were equal to one-third of the original file size, and the performance of their algorithm in terms of compression ratio was much better than other algorithms, such as Huffman coding and run length encoding. LZW is a fast and simple algorithm that got better results with repetitive data, and does not need to pass the string table to decompression code, the table can be recreated as it was, during compression, using input stream data. This avoids the insertion of a large string translation table with the compressed data. Antonopoulos et al. [19] proposed two lossless data compression methods. The first one was used for prediction, suitable for the Huffman code table in time series data, based on the lossless entropy compression and adaptive lossless entropy compression algorithms. The other compression method was for the enhancement of lossless compression of time series data based on increasing the average of neighborhood signals (variable length Golomb rice); it did not predict the signal, but rather encoded the differential signal with variable length using Golomb rice encoding. The proposed compression scheme provided an optimal compression rate and compression latency. Thomas et al. [20] proposed a new technique for lossy data compression based on distributed block truncation coding, whereby the sensor sent compressed binary data, and then the mean square error (MSE) was computed for the compressed and uncompressed data. If the MSE value was large, then the data array was divided into two halves and block truncation coding was applied for each half separately, thereby reducing the total MSE value. The proposed technique achieved a high compression ratio for clustered WSN, also it suited the slowly varying physical parameters with a high degree of temporal and spatial correlations. Yildirim et al. [21] proposed a new deep convolutional autoencoder model, which contained a neural network structure for compressing electrocardiogram (ECG) signals (lossy compression). This autoencoder is used to reduce the size of ECG signals and to improve the transmission process in many applications. They used deep convolutional auto encoder with a 27-layer network structure that was used to encode and decode the data. In the encoder part, the original ECG signal is encoded and reduced. It consists of 14 layers with consecutive 1D convolutions and pooling. At each layer different features of the input signal are learned. In the decoder part, it reconstructs the signal with low dimensional features. It contains 13 layers of up-sampling 1D convolutions to obtain the original signal. Deepika et al. [22] proposed a technique using a modified version of the ant colony algorithm with the diffusion wavelet algorithm. The diffusion is utilized as smoothing and scaling technique to enable multiscale applications. They modified the ant colony routing algorithm to speed up the convergence rate and to avoid the local optimal of the algorithm. Also, they proposed a data-gathering method by combining routing with compressive sensing methods to solve the sparsity problem of the gathered signals, it is based on the tree topology. The proposed system used the spatial correlation on a sensor node which led to inherent data sparsity, it also solved the problem of the sparsity of the signal.

All research stated above either have a fixed low compression ratio, or use complex computations which will lead to high consumption of energy at the sensor level, or have large reconstruction error in comparing with original data or all of them. This motivated us to propose a new method for compressing the data measured by the sensors that generate numerical values in the IoT with the following distinctions:

1. The method does not need to fix the compression ratio, which is calculated based on the data.

2. It is also performed using undemanding operations.

3. It does not consume memory or use the processor of the sensors. This reduces the total energy consumption of the sensors and saves the bandwidth, as demonstrated later.

## 3. IoT data compression based on successive data grouping

The data generated from the sensors are accumulated in a successive way. This means that each value measured by a sensor is added at the end of all previously sensed values, as preserving the order of the sensed data is sometimes very important in some applications. Measuring and sensing a value using a specific sensor in a specific environment can cause repetition of the data, such as measuring motion in front of a specific sensor, whereby the same sensed value is repeated many times until the motion is outside the sensor's range. Thus, the proposed technique in this paper takes advantage of the repetition of this data, and groups them to reduce the overall size of the original data.

Some sensors measure some events that change rapidly, which may only be a minor change, such as light sensors where each sensed value is not the same as the previous one. This leads to a very low compression ratio because the total number of grouped values is very low. Therefore, to have a high compression ratio, we trained the model on previously sensed values for each sensor, to find the root mean square of the successive deference (RMSSD). RMSSD is widely used in the medical field, e.g., to compare the heart rate variability in electrocardiogram graphs, the variability average is calculated for every two successive heart beats to identify any abnormalities [23]. In this paper, we derived our idea from the ECG heart rate variability by calculating the average variabilities for every two successive sensed values, then this value is used as a reference to compare in the compression process. If the difference between two successive values is greater than the RMSSD value, this difference is considered as new event (a major variation between the previous value and the new value) and need to be sent as a separate value. The small variations successive values will be compressed and sent once. The RMSSD used in limited fields for monitoring purposes, but it never used in compression.

The use of RMSSD in our proposed algorithm was done by calculating the average of differences between two consecutive sensed values of a specific sensor using the following equation:

$$RMSSD = \sqrt{\frac{1}{N-1} \times \sum_{i=1}^{N} (v_i - v_{i-1})^2} \tag{1}$$

where N is the total number of trained rows, $V_i$ is the current (new sensed value), and $V_{i-1}$ is the previous sensed value, as shown in Algorithm 1, which has O(N) complexity.

**Algorithm 1** Calculate RMSSD

```
1: Input TrainingSensedValues dataset;
2: Diff ← 0;
3: RMSSD ← 0;
4: for $i \leftarrow 1$ to $TrainingSensedValues.length$ do
5:     $Diff \leftarrow Diff + Absolute(TrainingSensedValues[i] - TrainingSensedValues[i-1])$;
6:     $RMSSD \leftarrow Diff/(TrainingSensedValues.length - 1)$;
7: end for
8: return RMSSD;
```

After training the model and finding out the mean average difference value between rows (from the training dataset), the RMSSD value is compared with the absolute difference between the new and previously sensed value. If the RMSSD value is larger than the calculated absolute difference, then the new value is considered the same as the previous value and grouped with it. If the RMSSD value is lower than the absolute difference, then the new sensed value is different from the previously sensed value and needs to be stored separately instead of being grouped with the previously sensed value.

$$\text{Decision} = \begin{cases} \text{group with previous} & \text{if } \mid v_i - v_{i-1} \mid \leq RMSSD \\ \text{store as new value} & \text{otherwise} \end{cases}$$

The grouping process is done by counting the total number of successive values that have the same or almost the same sensed values (using the RMSSD comparisons) and storing them as a single row with the sensed value, which then represents the entire group of sensed values. By doing the grouping process using the RMSSD value, the compression is considered to be lossy compression because some values are considered the same as the previously sensed value, which is less than or equal to the RMSSD value. This is shown in Algorithm 2, which has O(N) complexity, where N is the total number of sensed values.

**Algorithm 2** Compress Data

```
1: Input SensedValues dataset, RMSSD value;
2: Diff ← ∞ ;
3: Result ← Empty List with two columns (value, occurrence);
4: for $i \leftarrow 1$ to $SensedValues.length$ do
5:     $Diff \leftarrow Absolute(TrainingSensedValues[i] - TrainingSensedValues[i-1])$;
6:     if $Diff \leq RMSSD$ then
7:         Update Result.LastRow (occurrence=occurrence + 1);
8:     else
9:         Result.Add (SensedValues[i],1);
10:    end if
11: end for
12: return Result;
```

We reconstructed the original data or data almost the same as the original data to evaluate our work in terms of the total difference between the reconstructed and original data. Each row in the compressed dataset was duplicated sequentially as many times as the occurrence value for that row, resulting in the total number of rows being the same as the original dataset. The reconstructed data error would be less than or equal to the RMSSD value which we calculated in the training process. In the process of compression, if the difference between two successive values is less than or equal to the RMSSD value, it is ignored and considered to have the same value as the previously sensed value. This is shown in Algorithm 3, which has O(N×M) complexity, where N is the total number of values in the compressed dataset, and M is the summation of all occurrences of the values in the compressed datasets.

## 4. Implementation and evaluation

### 4.1. Data acquisition

In order to prove the compression performance of our methodology, we used the Intel Lab wireless sensor network dataset, which was collected by a research team from the University of California[1]. They used 54 sensors to

[1]Massachusetts Institute of Technology (2004). Intel lab data [online]. Website http://db.csail.mit.edu/labdata/labdata.html [accessed 1 September 2019].

**Algorithm 3** Reconstruct Data

```
1: Input CompressedValues dataset;
2: Result ← Empty List with one column (value);
3: for $i \leftarrow 0$ to $CompressedValues.length$ do
4:     for $j \leftarrow 0$ to $CompressedValues[i].occurrence$ do
5:         Result.Add (CompressedValues[i]);
6:     end for
7: end for
8: return Result;
```

collect some environmental data information such as the temperature, humidity, light, and voltage values of the surrounding lab for the period from 28 February 2004 to 5 April 2004.

### 4.2. Preprocessing dataset

Each sensor measured the surrounding environmental data every 31 s, and each value was attached with the timestamp and the sensor identifier it belonged to. We extracted the temperature values from the dataset, but there were some sensors that failed or exhibited abnormal events; thus, some temperature values exceeded the accepted range (with values of more than 100 °C or less than –30 °C). Therefore, a preprocessing step was done on the dataset to exclude any value less than –5 °C or any value greater than 45 °C based on prior knowledge.

After exploring the generated data, some sensors did not measure any temperature values such as node 5 and node 57. Other nodes measured a few number of values such as nodes 15, 56, 55, 58 with 2337, 2507, 2851, 4502 values, respectively. Other nodes such as node 47 has 159 missing values. In our experiments, we used sensor node number 7 to evaluate our work, it had 55,361 total sensed values, after removing the outliers in the previous step, it yielded 43,632 values that can be used in our experiments. We also choose node 7 because it allows comparing our results with some other previous methods that used the same node 7 in their experiments.

The generated preprocessed data were stored as one column in a text file with the extension.txt, ready to be compressed. Please note that, in our experiments, we did not do any normalization or rounding for the temperature values; the data were used as they were obtained, which would lead to better accuracy.

### 4.3. Compression and reconstruction

We compared our work in terms of performance with other compression algorithms, such as the CS, standard RBM, and stacked RBM-AE compression algorithms. In the CS algorithm, the total number of rows examined was 40,000 rows from sensor 7; the data were divided into eight segments because the CS algorithm cannot use long datasets, and the compression ratio was fixed at 10. Finally, the average of the results from all segments was calculated.

In order to evaluate the proposed methodology, we used different performance criteria as follows:

1. Compression ratio (CR): this was the ratio between the original data and the generated compressed data. The compression ratio is given in Equation (2) [24]

$$CR = \frac{D_o}{D_c} \tag{2}$$

where the $D_o$ is the total number of bytes of original data that need to be compressed, and $D_c$ is the total number of bytes of the resulting compressed data. The performance of the compression is much better when it has a high compression ratio.

2. Percentage of root-mean-square difference (PRD): this criterion was used to measure the percentage difference between the reconstructed data and the original data (the quality of reconstructed data), as given in Equation (3) [24].

$$PRD(\%) = 100 \times \sqrt{\frac{\sum_{i=0}^{D-1}(S_o(i) - S_c(i))^2}{\sum_{i=0}^{D-1}(S_o(i))^2}} \tag{3}$$

where $S_o$ is the original data value, $S_c$ is the reconstructed data value, and D is the total number of rows of data. The accuracy of the reconstructed data is considered much better when it has a low PRD value.

3. Quality score (QS): this criterion was used to evaluate the overall compression performance based on the ratio between the compression ratio and the accuracy of the reconstructed data. Having a larger quality score is much better for compression and reconstruction performance. The quality score is given in Equation (4) [24].

$$QS = \frac{CR}{PRD} \tag{4}$$

The first step in the proposed methodology was to calculate the RMSSD value based on Equation (1). The resultant value for RMSSD was equal to 0.08757. This RMSSD value was the first calculated parameter needed in our proposed methodology. After compressing the original data using the proposed methodology, the total number of rows in the generated compressed file was determined to be 3983 rows. By comparing the generated compressed file with the original file, the CR value was calculated based on Equation (2) and found to be equal to 10.954. We did not use any predefined CR value, as done for CS, standard RBM, and stacked RBM-AE, where the compression ratio was fixed to 10. However, in the proposed methodology, the CR value was calculated based on the resultant file; thus, our proposed method achieved a better compression ratio in comparison as shown in Figure 1.

Then, we reconstructed the data using the generated compressed file and calculated the PRD percentage by calculating the average value of the summation of absolute differences between the reconstructed data and the original data to determine the quality of reconstructed data based on the original data. The calculated PRD value was equal to $8.79 \times 10^{-4}\%$, while CS obtained a value of 38.4%, standard RBM obtained a value of 33.03%, and stacked RBM-AE obtained a value of 10.04%. Therefore, our methodology scored the lowest PRD value (indicating the best compression performance), as shown in Figure 2.

The minimum absolute error between the original data and the reconstructed data was 0 °C, suggesting no difference between the reconstructed data and the original data. The maximum absolute error was 0.0784 °C, which was obviously lower than the RMSSD value. Finally, the average absolute error was 0.0313 °C. Figure 3 shows a portion of the comparison between the original data and reconstructed data, presenting 10,000 points from the overall data.

As can be seen from Figure 3, the total distance between the two lines was very close; thus, our methodology was proven to have higher data reconstruction accuracy, with the reconstructed data being almost the same as the original data, in comparison to the CS algorithm (1.4143), standard RBM (1.0423), and stacked RBM-AE (0.2815), as shown in Figure 4.

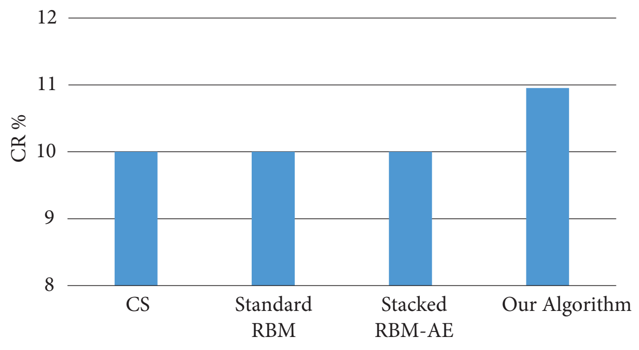

**Figure 1**. A comparison between the compression ratio that is used in the proposed algorithm (calculated based on the data) and other algorithms (fixed value = 10).

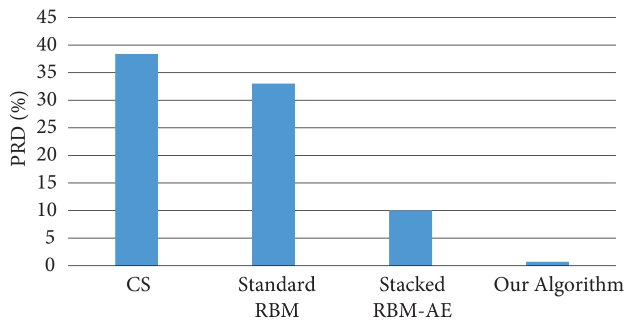

**Figure 2**. PRD comparison between different compression algorithms and the proposed one, which scores the lowest PRD value among the other algorithms.

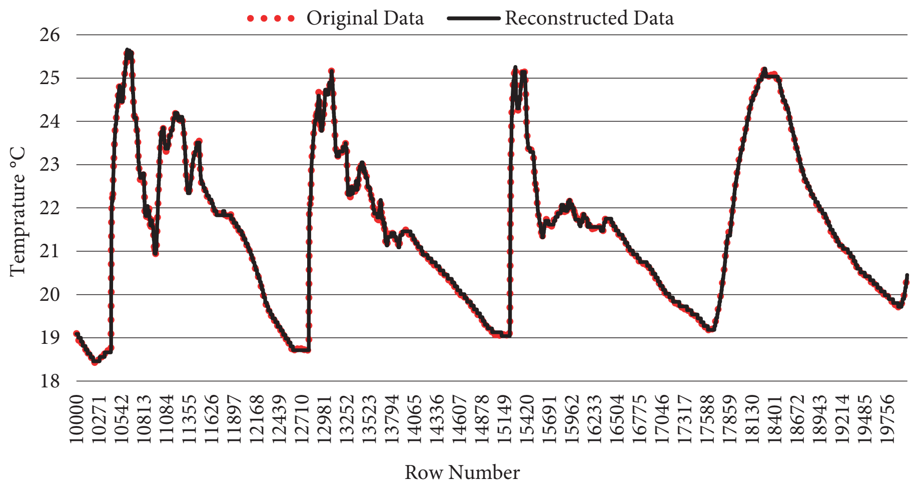

**Figure 3**. A comparison between the reconstructed and original data for sensor 7 (a portion of the data is depicted). This shows how the reconstructed data are very close to the original data.

Finally, we calculated the last criterion, i.e. the quality score (QS), of the compression performance using Equation (4). The resultant QS value was equal to $124.56 \times 10^4$, while CS obtained a value of 26.04, standard RBM obtained a value of 30.28, and stacked RBM-AE obtained a value of 99.6. Therefore, our algorithm scored the highest QS value, as shown in Figure 5. Therefore, our algorithm showed the best performance results in comparison to the CS algorithm and the standard RBM and stacked RBM-AE compression algorithms, as shown in Table 1. We also evaluated our algorithm using different datasets and calculated its performance criteria, as shown in Table 2.

[2]Massachusetts Institute of Technology (2004). Intel lab data [online]. Website http://db.csail.mit.edu/labdata/labdata.html [accessed 1 September 2019].

[3]UCI Machine Learning Repository (2012). Individual household electric power consumption data set [online]. Website https://archive.ics.uci.edu/ml/datasets/individual+household+electric+power+consumption [accessed 1 September 2019].

[4]Argo (2019). Argo Float Data and Metadata from Global Data Assembly Centre [online]. Website http://www.argo.ucsd.edu/ [accessed 1 September 2019].

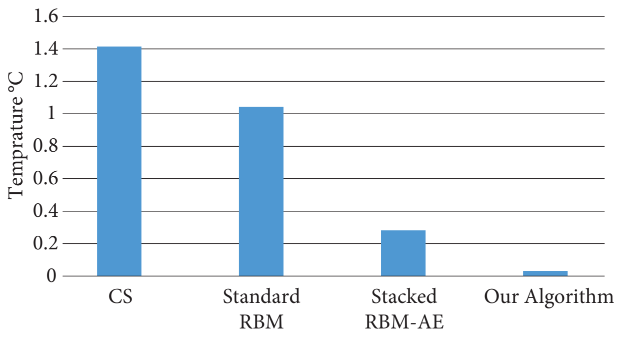

**Figure 4**. Reconstruction data error. A comparison between different compression algorithms and the proposed one, which scores the lowest reconstruction data difference.

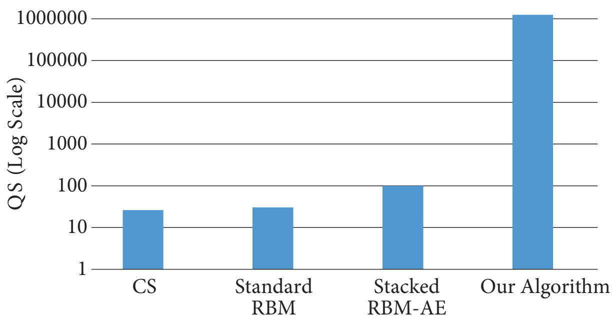

**Figure 5**. Quality score (logarithmic scale) comparison between different compression algorithms and the proposed algorithm, which outperforms other compression algorithms.

**Table 1**. Compression performance comparison. CS—compressed sensing; RBM—restricted Boltzmann machine; AE—auto encoder; CR—compression ratio; PRD—percentage of root-mean-square difference; QS—quality score.

| Algorithm | CR | PRD (%) | QS | Reconstruction error (°C) |
|---|---|---|---|---|
| CS | 10 | 38.40 | 26.04 | 1.4143 |
| Standard RBM | 10 | 33.03 | 30.28 | 1.0423 |
| Stacked RBM-AE | 10 | 10.04 | 99.6 | 0.2815 |
| Our algorithm | 10.954 | $8.79 \times 10^{-4}$ | $124.56 \times 10^{4}$ | 0.0313 |

**Table 2**. Compression performance comparison.

| Dataset | CR | PRD (%) | QS | Reconstruction error |
|---|---|---|---|---|
| Intel Lab (humidity)[2] | 10.894 | $1.28 \times 10^{-3}$ | $848.45 \times 10^{3}$ | 0.24 |
| IHEPC (active power)[3] | 10.869 | $8.66 \times 10^{-3}$ | $125.57 \times 10^{3}$ | 0.2501 |
| GSATM (humidity) [25] | 19.55 | $5.14 \times 10^{-6}$ | $3804.987 \times 10^{5}$ | 0.0062 |
| GSATM (temperature) [25] | 21.039 | $5.54 \times 10^{-7}$ | $3799.888 \times 10^{6}$ | 0.00045 |
| Argo (temperature)[4] | 10.43 | $1.633 \times 10^{-3}$ | $638.358 \times 10^{3}$ | 0.0991 |

## 4.4. Energy optimization

Sensors generate massive amounts of data, which are transferred to other IoT layers to be processed and stored. However, the problem is that transferring this data consumes a huge amount of energy since most of the energy is consumed in the data transmission process [26]. Therefore, in this paper, the main focus was on data compression of the transferred data, which would lead to a reduction in the transmission energy consumption.

Complex computations on the sensor level consume more energy; therefore, in the proposed methodology, all complex computations are done on the server layer (not on the sensor layer) to save energy. The RMSSD value is computed for each sensor, before being sent to the sensor layer for the compression process. The compression process in the proposed methodology is performed through comparisons and counting processes. If the difference between the currently sensed value and the previous value is less than the RMSSD, then the occurrence value is increased by one and it is not sent directly to the server. If the newly sensed value has a difference greater than the RMSSD value, then it is considered a new value; the sensor then sends the previous value with its occurrence counter to the server layer, as shown in Figure 6.

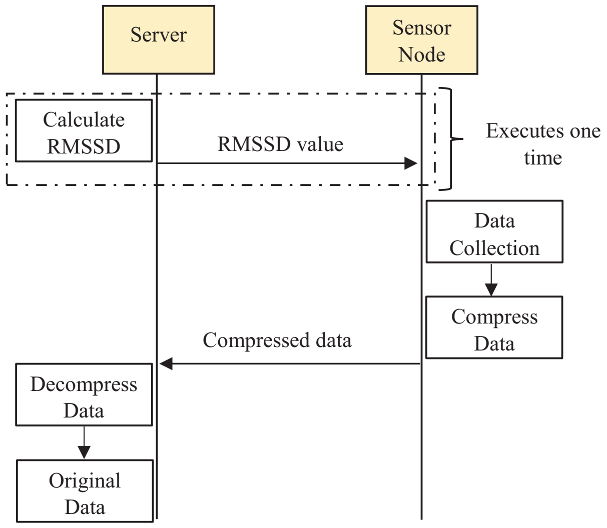

**Figure 6**. The workflow of the proposed methodology, which shows the processes that are performed on the server-side and the sensor node side.

The main purpose of using data compression is to reduce the total energy consumed in the transmission process as mentioned before. Therefore, in the proposed methodology, the worst-case scenario where all data have successive differences greater than the RMSSD value (which is infeasible), each value is sent to the server node separately, and the energy consumed is the same as when having no data compression. If we assume that the number of values is N and the energy consumed to transfer one value is E, then the total transmission energy consumed can be written as follows:

$$\text{Transmission Energy} = N \times E \tag{5}$$

The best-case scenario is where all successive data differences are less than the RMSSD value, whereby all data are grouped and sent as one value to the server. Then, the total transmission energy consumed can be written as follows:

$$\text{Transmission Energy} = 1 \times E \tag{6}$$

The average scenario is where half of the successive data differences are less than the RMSSD value, resulting in half of the data being sent to the server, written as follows:

$$\text{Transmission Energy} = \frac{N}{2} \times E \tag{7}$$

Data can be transferred using different communication types such as LTE, 3G, and Wi-Fi. The energy consumption for each communication type is shown in Table 3 [27], where $\alpha_{\mathrm{u}}$ is the energy consumed when sending the data in mW/Mbps, $\alpha_{\mathrm{d}}$ is the energy consumed when receiving the data in mW/Mbps, and $\beta$ is the instant power. Sending 1 Mbps consumes $\alpha_{\mathrm{u}} + \beta$ mW, and receiving 1 Mbps consumes $\alpha_{\mathrm{d}} + \beta$mW [28].

As per our experiments on sensor 7 in the Intel Lab Data, the total number of values in the original data file (before compression) was 43,632 float type values; thus, the overall size consumed was 43,632 $\times$ 4 B

**Table 3**. Data transfer energy consumption based on communication types [27].

| Communication Type | $\alpha_{\mathrm{u}}$ (mW/Mbps) | $\alpha_{\mathrm{d}}$ (mW/Mbps) | $\beta$ (mW) |
|---|---|---|---|
| Wi-Fi | 283.17 | 137.01 | 132.86 |
| 3G | 868.98 | 122.12 | 817.88 |
| LTE | 438.39 | 51.97 | 1288.04 |

= 170.44 kB. Sending this amount of data would consume 509.9 mW (377.06 + 132.86) when using the Wi-Fi communication type, 1974.98 mW (1157.1 + 817.88) when using the 3G communication type, and 1871.78 mW (583.74 + 1288.04) when using the LTE communication type.

After using the proposed methodology, the total number of values in the compressed file was 3983 float type values with an occurrence integer type value; thus, the overall size consumed was 3983 $\times$ (4 B + 2 B) = 23.34 kB. Sending this amount of data would only consume 184.48 mW (51.62 + 132.86) when using the Wi-Fi communication type, 976.29 mW (158.41 + 817.88) when using the 3G communication type, and 1367.95 mW (79.91 + 1288.04) when using the LTE communication type.

As can be seen, the proposed methodology saved the transmission energy needed to transfer the overall data from the sensor layer to the next layer, as shown in Table 4. If there are several layers before the last desired layer, then accumulated saving of transmission energy will happen.

**Table 4**. Sensor 7 data transfer energy consumption.

| Communication type | Original method (mW) | Proposed method (mW) |
|---|---|---|
| Wi-Fi | 509.9 | 184.48 |
| 3G | 1974.98 | 976.29 |
| LTE | 1871.78 | 1367.95 |

## 5. Conclusions

In this paper, we proposed a new compression method to compress IoT data based on grouping similar values together. First of all, the average of differences between successive rows was calculated; then, it was used in identifying the successive values that were grouped together. The proposed algorithm did not use any fixed compression ratio parameter like other compression algorithms; instead, the compression ratio was adaptively calculated based on the difference between successive data and the calculated RMSSD value. The reconstruction process was done by repeating the same value based on the counter value in front of each compressed value in the compressed file. We evaluated our work and compared it with various compression algorithms, such as the CS, standard RBM, and stacked RBM-AE compression algorithms. Our algorithm outperformed these algorithms in terms of compression ratio, the quality of reconstructed data, the effectiveness of the compression ratio, and the average reconstruction data error. The data were reduced by more than 90% with an average reconstruction data error of 0.0313. Hence, our algorithm can be used to reduce the total energy consumption, as it enables transferring less data from the sensor side to the processing and storage side, whereby the number of rows needed to be transferred is very low (less than 10% of the original data).

## References

[1] Patel KK, Patel SM. Internet of Things-IoT: definition, characteristics, architecture, enabling technologies, application & future challenges. International Journal of Engineering Science Computing 2016; 6: 6122-6131. doi: 10.4010/2016.1482

[2] Janani P, Siddhant V, Aditya K. Survey of data aggregation techniques in Internet of Things (IoT). International Journal of Recent Scientific Research 2018; 9: 23675-23679. doi: 10.24327/ijrsr.2018.0901.1514

[3] Zeinab KAM, Elmustafa SAA. Internet of Things applications, challenges and related future technologies. World Scientific News 2017; 2: 126-148.

[4] Al-Doghman F, Chaczko Z, Jiang J. A Review of aggregation algorithms for the Internet of Things. In: Proceedings of the 2017 25th International Conference on Systems Engineering (ICSEng 2017); Las Vegas, NV, USA; 2017. pp. 480-487. doi: 10.1109/ICSEng.2017.43

[5] Mattern F, Floerkemeier C. From the internet of computers to the Internet of Things. In: Sachs K, Petrov I, Guerrero P (editors). From Active Data Management to Event-Based Systems and More. Berlin/Heidelberg, Germany: Springer, 2010, pp. 242-259. doi: 10.1007/978- 3-642-17226-7_15

[6] Jia-Heng L, Xiao-Lin Z, Rong-Chao P, Feng L. An adaptive compression algorithm for Wireless Sensor Network based on piecewise Linear Regression. In: International Conference on Biomedical and Health Informatics, IFMBE Proceedings; Singapore; 2015. pp. 43-47. doi: 10.1007/978-981-10-4505-9_7

[7] Basheer A, Sha K. Cluster-based quality-aware adaptive data compression for streaming data. Journal of Data Information Quality 2017; 9 (2): 1-15. doi: 10.1145/3122863

[8] Azar J, Darazi R, Habib C, Makhoul A, Demerjian J. Using DWT Lifting Scheme for lossless data compression in wireless body sensor networks. In: Proceedings of the 2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC 2018); Limassol, Cyprus; 2018. pp. 1465-1470. doi: 10.1109/IWCMC.2018.8450459

[9] Feng L, Kortoçi P, Liu Y. A multi-tier data reduction mechanism for IoT sensors. In: Proceedings of the 7th International Conference on the Internet of Things (IoT 2017); Linz, Austria; 2017. pp. 1-8. doi: 10.1145/3131542.3131557

[10] Campobello G, Segreto A, Zanafi S, Serrano S. RAKE: A simple and efficient lossless compression algorithm for the Internet of Things. In: Proceedings of the 2017 25th European Signal Processing Conference (EUSIPCO); Kos Island, Greece; 2017. pp. 2581-2585. doi: 10.23919/EUSIPCO.2017.8081677

[11] Donoho D. Compressed sensing. IEEE Transactions on Information Theory 2006; 52: 1289-1306. doi: 10.1109/TIT.2006.871582

[12] Norouzi M, Ranjbar M, Mori G. Stacks of convolutional Restricted Boltzmann Machines for shift-invariant feature learning. In: Proceedings of the 2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops; Miami, FL, USA; 2009. pp. 2735-2742. doi: 10.1109/CVPR.2009.5206577

[13] Liu J, Chen F, Wang D. Data compression based on Stacked RBM-AE model for Wireless Sensor Networks. Sensors 2018; 18 (4273): 1-20. doi: 10.3390/s18124273

[14] Wu M, Tan L, Xiong N. Data prediction, compression, and recovery in clustered wireless sensor networks for environmental monitoring applications. Information Sciences 2016; 329: 800-818. doi: 10.1016/j.ins.2015.10.004

[15] Abu Alsheikh M, Lin S, Niyato D, Tan HP. Rate-distortion balanced data compression for Wireless Sensor Networks. IEEE Sensors Journal 2016; 16: 5072-5083. doi: 10.1109/JSEN.2016.2550599

[16] Incebacak D, Zilan R, Tavli B, Barceló-Ordinas JM, Garcia-Vidal J. Optimal data compression for lifetime maximization in wireless sensor networks operating in stealth mode. Ad Hoc Networks 2015; 24: 134-147. doi: 10.1016/j.adhoc.2014.07.019

[17] Liang Y, Li Y. An efficient and robust data compression algorithm in Wireless Sensor Networks. IEEE Communications Letters 2014; 18: 439-442. doi: 10.1109/LCOMM.2014.011214.132319

[18] Ruxanayasmin B, Krishna BA, Subhashini T. Implementation of data compression techniques in mobile ad hoc networks. International Journal of Computer Applications 2013; 80: 8-12. doi: 10.5120/13879-1764

[19] Antonopoulos CP, Voros NS. Resource efficient data compression algorithms for demanding, WSN based biomedical applications. Journal of Biomedical Informatics 2016; 59: 1-14. doi: 10.1016/j.jbi.2015.10.015

[20] Thomas S, Mathew T. In-network data compression in Wireless Sensor Networks using distributed block truncation coding. In: Proceedings of the 2018 Second International Conference on Advances in Electronics, Computers and Communications (ICAECC); Bangalore, India; 2018. pp. 1-7. doi: 10.1109/ICAECC.2018.8479478

[21] Yildirim O, Tan RS, Acharya UR. An efficient compression of ECG signals using deep convolutional autoencoders. Cognitive Systems Research 2018; 52: 198-211. doi: 10.1016/j.cogsys.2018.07.004

[22] Deepika M, Belwin AF, Sherin AL, Thanamani AS. An efficient compressive sensing data gathering using modified Ant Colony and Diffusion Wavelets in WSN. International Journal of Computer Science and Mobile Computing (IJCSMC) 2018; 7: 216-230.

[23] Desislava N, Petia M, Agata M, Petia G. ECG-based human emotion recognition across multiple subjects. In: Future Access Enablers for Ubiquitous and Intelligent Infrastructures: 4th EAI International Conference, FABULOUS 2019; Sofia, Bulgaria; 2019. pp. 25-36. doi: 10.1007/978-3-030-23976-3_3

[24] Jha CK, Kolekar MH. Electrocardiogram data compression using DCT based discrete orthogonal Stockwell transform. Biomedical Signal Processing and Control 2018; 46: 174-181. doi: 10.1016/j.bspc.2018.06.009

[25] Burgués J, Marco S. Multivariate estimation of the limit of detection by orthogonal partial least squares in temperature-modulated MOX sensors. Analytica Chimical Acta 2018; 1019: 49-64. doi: 10.1016/j.aca.2018.03.005

[26] Liu X, Wu J. A method for energy balance and data transmission optimal routing in Wireless Sensor Networks. Sensors 2019; 19 (3017): 1-20. doi: 10.3390/s19133017

[27] Huang J, Qian F, Gerber A, Mao ZM, Sen S et al. A close examination of performance and power characteristics of 4G LTE networks. In: Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services; Ambleside, UK; 2012. pp. 225-238. doi: 10.1145/2307636.2307658

[28] Jha SC, Koc AT, Vannithamby R. Device power saving mechanisms for low cost MTC over LTE networks. In: Proceedings of the 2014 IEEE International Conference on Communications Workshops (ICC); Sydney, Australia; 2014. pp. 412-417. doi: 10.1109/ICCW.2014.6881233