Spring 2018

COMPRESSION OF WEARABLE BODY SENSOR NETWORK DATA USING IMPROVED TWO-THRESHOLD-TWO-DIVISOR DATA CHUNKING ALGORITHM

Robinson Raju
San Jose State University

COMPRESSION OF WEARABLE BODY SENSOR NETWORK DATA USING IMPROVED TWO-THRESHOLD-TWO-DIVISOR DATA CHUNKING ALGORITHM

A Thesis

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfilment

of the Requirements for the Degree

Master of Science

by

Robinson Raju

May 2018

The Designated Thesis Committee Approves the Thesis Titled

COMPRESSION OF WEARABLE BODY SENSOR NETWORK DATA USING IMPROVED TWO-THRESHOLD-TWO-DIVISOR DATA CHUNKING ALGORITHM

by

Robinson Raju

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

MAY 2018

Dr. Melody Moh	Department of Computer Science, SJSU
Dr. Teng Moh	Department of Computer Science, SJSU
Mr. Amit Meckoni	Software Engineering Architect, PayPal Inc.

ABSTRACT

COMPRESSION OF WEARABLE BODY SENSOR NETWORK DATA USING IMPROVED TWO-THRESHOLD-TWO-DIVISOR DATA CHUNKING ALGORITHM

by Robinson Raju

Compression plays a significant role in Body Sensor Networks (BSN) data since the sensors in BSNs have limited battery power and memory. Also, data needs to be transmitted fast and in a lossless manner to provide near real-time feedback. The paper evaluates lossless data compression algorithms like Run Length Encoding (RLE), Lempel Zev Welch (LZW) and Huffman on data from wearable devices and compares them in terms of Compression Ratio, Compression Factor, Savings Percentage and Compression Time. It also evaluates a data deduplication technique used for Low Bandwidth File Systems (LBFS) named Two Thresholds Two Divisors (TTTD) algorithm to determine if it could be used for BSN data. By changing the parameters and running the algorithm multiple times on the data, it arrives at a set of values that give >50 compression ratio on BSN data. This is the first value of the paper. Based on these performance evaluation results of TTTD and various classical compression algorithms, it proposes a technique to combine multiple algorithms in sequence. Upon comparison of the performance, it has been found that the new algorithm, TTTD-H, which does TTTD and Huffman in sequence, improves the Savings Percentage by 23 percent over TTTD, and 31 percent over Huffman when executed independently. Compression Factor improved by 142 percent over TTTD, 52 percent over LZW, 178 percent over Huffman for a file of 3.5 MB. These significant results are the second important value of the project.

Keywords: body sensor network, compression, TTTD, Huffman, data chunking

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor, Dr. Melody Moh, for her encouragement, patience, expertise and continuous guidance throughout my project and my earlier research studies. I would like to express my sincere gratitude to Dr. Teng-Sheng Moh, also my advisor for TTTD research, for his encouragement, patience, expertise and continuous guidance throughout my graduate studies. I would also like to thank Mr. Amit Meckoni for his valuable time, inputs and support throughout the project.

I am eternally thankful to my mother, Mrs. Rosamma Raju, without whose hard work I may never have completed schooling or had a chance at pursuing higher education. I am also grateful to my wife, Ms. Ashwati Kuruvilla, who extended her support during my late night and weekend study schedules. Great appreciation is due towards my current and former employers, eBay Inc. and PayPal Inc, for supporting my pursuit of a Master's Degree in Computer Science.

Lastly, a big thank you to the Department of Computer Science at San José State University for giving me this exceptional opportunity and to all the faculty who continuously help students succeed at the University.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ANN – Artificial Neural Network
ASCII – American Standard Code for Information Interchange
BSN – Body Sensor Network
CDC – Content Defined Chunking
CF – Compression Factor
CR – Compression Ratio
CS – Compressed File Size
CT – Compression Time
ECG – Electrocardiograph
EEG – Electroencephalograph
EMG – Electromyography
FS – File Size
HBC – Human Body Communication
HMM – Hidden Makhov Model
ICA – Independent Component Analysis
IoT – Internet of Things
KNN – K-Nearest Neighbor
LBFS – Low Bandwidth File System
LZW – Lempel Zev Welch
MEMS – Micro Electro Mechanical Systems
PCA – Principal Component Analysis
RFID – Radio-frequency identification
RLE – Run Length Encoding
RMS – Root Mean Square
SMA – Signal Magnitude Area
SP – Savings Percentage
SVM – Support Vector Machine
TTTD – Two Threshold Two Divisors

1. INTRODUCTION

As the population of the world rises and healthcare costs increase worldwide, human health monitoring has become a critical research area, since it helps tremendously in containing the expenses related to healthcare and enhancing the customer experience [1]. Though there have been devices to measure vital statistics from a person's body, they have mostly been wired, large and conspicuous. Recent trends towards improvements in micro-electro-mechanical systems (MEMS) technology [2], wireless communications, and digital electronics have allowed the development of miniature, low-cost, low power, multi-functional sensor that can sense and transmit data wirelessly. One family of these devices is wearable and implantable Body Sensor Networks (BSNs), which are IoT (Internet of Things) devices that can transmit health-related data from a person wirelessly to a node. The data can be used to monitor vital signs and provide real-time feedback. The data can also be used for machine learning and predictive analytics, to foresee medical infrastructure needs and lead the world towards a future of ubiquitous healthcare monitoring. Since real-time data analysis is critical in many cases, transmitting the data from sensors to sink nodes with speed and ease is vital. The sensors in the BSN have limited battery and memory available, and this makes data compression very crucial in a BSN. Also, since the data in question is related to health information, it should be accurate, and the compression algorithms have to be lossless. As the technology improves, the number of sensors and the amount of data that a sensor can capture and transmit also increases. This is another reason to focus on data compression at the sensor nodes.

The primary objective of the paper is to evaluate the performance of classical data compression algorithms and Two Thresholds Two Divisors (TTTD) data chunking algorithm on BSN data. The paper starts by giving a background on BSN, the general architecture, types of sensors, followed by data processing steps common in most BSNs. It then reviews existing research on compression of BSN data. After this, it gives a short overview of lossless compression algorithms like Huffman, LZW and RLE and data chunking technique called TTTD. It then reviews the results of experimental evaluation the aforementioned algorithms on BSN data. It establishes that TTTD can be used to compress BSN data with a different set of parameters, and has performance that is comparable to other algorithms. After the experimental evaluations, the paper proposes an approach to combine TTTD and Huffman algorithms to compress data more efficiently. This algorithm, TTTD-H is then evaluated on BSN data from multiple sources like Fitbit and smartphone datasets.

2. BACKGROUND

2.1 General Architecture of a BSN

A Body Sensor Network (BSN), is a wireless network of wearable computing devices. BSNs may be

- Embedded inside the body as implants
- Placed on the body in a fixed position
- In accompanied devices which people carry around, like in pockets, by hand, in a bag, and so forth.

Figure 1 describes the general architecture of a BSN. The sensor nodes at different parts of the body collect physical data and transmit to the sink node which then transmits it to the base station. Some sensors directly transmit to the base station or send data via Bluetooth to smartphones.

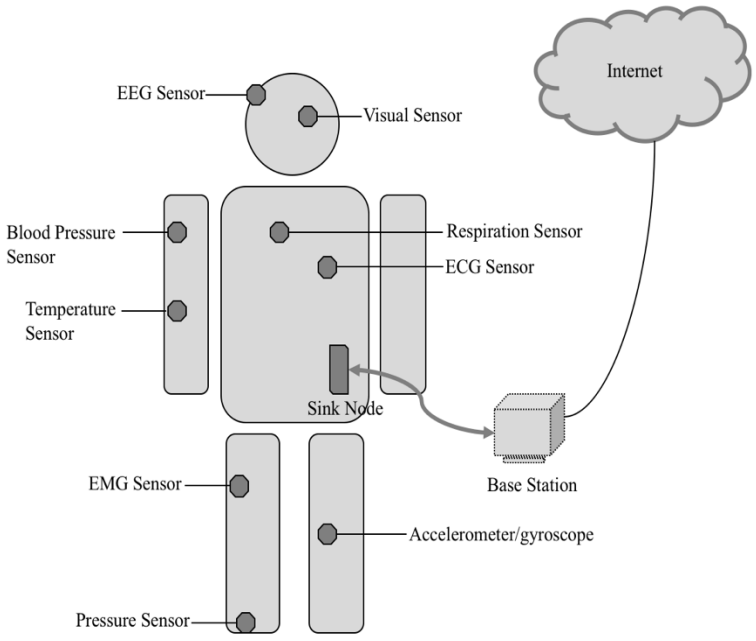


Figure 1. The Architecture of a Body Sensor Network

2.2 Sensors in a BSN

Sensors are IoT devices that connect the physical world with the measuring system and eventually, the internet. They collect information about the surrounding and are responsible for processing information and transmitting them. A sensor node ordinarily comprises the following modules [3]:

- Sensor module
- Processor module
- Wireless communication module
- Power supply module

While the sensor module collects and converts physical data into electrical signals, the wireless communication module transports the signal to various devices. The processor module controls the sensor nodes, and the power supply module provides energy to the nodes.

2.2.1 Classification of Sensors

Sensors could be classified by BSN attributes like types of signals, transmission media, deployment position, and so forth [3].

Classification by Types of Measured Signals

a) *Sensors that collect continuous time-varying signals:* This type of sensor collects data continuously, and the main requirement here would be real-time transfer of information. The continuous signal sensors can also generate a lot of data. Accelerometers or gyroscopes used in

Smartwatches, ECG, EEG, EMG sensors are examples of these type of sensors.

b) *Sensors that collect discrete time-varying physiology signals:* This type of sensor collects signals in discrete intervals. Temperature and humidity sensors capture measurements every 'x' minutes or hours, blood pressure monitors and measure BP every hour or day are examples of these types of sensors.

Classification by data transmission media

a) *Wireless sensors*: These sensors employ wireless communication technologies like Bluetooth, Zigbee, RFID, and so forth to communicate with other devices.

b) *Wired sensors*: These sensors are physically connected to other devices and transmit the data through wires.

c) *Human body communication (HBC) sensors*: These sensors use the human body as the transmission medium and they adopt sub-GHz frequencies for transmission.

Other Classifications

a) The sensors can also be classified based on deployment position as Wearable, Implantable or Surrounding. They can also be classified based on the automatic adjustment ability whether they are self-adapting or not.

2.2.3 Commonly Used Sensors in BSNs

TABLE 1. COMMONLY USED SENSORS IN BSNs

Sensors	Signal Type	Frequency	Position
Accelerometer	Continuous	High	Wearable
Artificial cochlea	Continuous	High	Implantable
Artificial retina	Continuous	High	Implantable
Blood-pressure sensor	Discrete	Low	Wearable
Camera pill	Continuous	High	Implantable
Carbon dioxide sensor	Discrete	Low/ Very low	Wearable
ECG/EEG/ EMG sensor	Continuous	High	Wearable
Gyroscope	Continuous	High	Wearable
Humidity sensor	Discrete	Very low	Wearable
Blood oxygen saturation sensor	Discrete	Low	Wearable
Pressure sensor	Continuous	High	Wearable/ Surrounding
Respiration sensor	Continuous	High	Wearable
Temperature sensor	Discrete	Very low	Wearable
Visual sensor	Continuous/ Discrete	High/ Low	Wearable/ Surrounding

2.3 Data Processing in BSNs

Data Processing, also referred to as Data fusion, is a process for handling the data from the sensors in an efficient manner. BSNs produce a considerable amount of data and data processing techniques are needed to filter noise efficiently, combine data from multiple sensors, extract necessary information and transmit to devices that need the

information for analysis. The following is a summarization of different steps in data fusion [3].

a) *Pre-processing*: Since the wireless and implantable sensors are constantly in a dynamic environment, the data that comes out of these sensors can many times have a lot more information than what is pertinent to being measured. 'pre-processing' is a step to remove the noise from the data without losing the vital information. Some of the techniques used for preprocessing are Fourier Transform, Wavelet Transform [4], Mathematical morphology filters [5], Kalman filter [6], Low-pass median value filter, Laplacian Transform, Gaussian filter and so forth.

b) *Feature Extraction*: The principal objective of this step is to extract features that represent the characteristics of the original data accurately. The classifiers use the features as inputs. Techniques regularly used in feature extraction include Support Vector Machine (SVM), K-Means clustering, Principal Component Analysis (PCA), Independent Component Analysis (ICA), and so forth. Commonly used features are time-Domain Features like Variance and Root Mean Square (RMS), frequency-Domain Features like Spectral Energy and Spectral Entropy, time-Frequency Domain Features like Wavelet Coefficients, heuristic Features like Signal Magnitude Area (SMA), Signal Vector Magnitude, and Inter-Axis Correlation and domain-specific Features like Time-Domain Gait Detection [7].

c) *Data Processing (Computing)*: The chief objective of the data computing step is to use Algorithms to analyze the data. Machine Learning Algorithms could be used to do Classification or Clustering. As per Lai et al. [3], the commonly used algorithms include thresh-old-based classification, hierarchical methods, decision trees, and K-Nearest Neighbor (KNN), Support Vector Machines (SVM), Artificial Neural Networks (ANN), Hidden Markov Models (HMM) and so forth. Peng et al. [8], recognized fourteen physical activities using a binary decision-tree with a Naïve Bayes classifier. Krishnan et al. [9], conducted research on how AdaBoost, HMM, and KNN, are used to analyze data from accelerometers to identify human hand activity.

d) *Data Compression*: After noise reduction, feature extraction, and optionally data fusion, the sensor nodes do compression of the data before sending it to the sink node or the base station. Data compression reduces the amount of data transmission and also lowers power consumption. This is important since power consumption is one of the main areas of concern in a WSN or a BSN [10]. Data Compression is the main topic of exploration in this paper.

3. RELATED STUDIES

3.1 The Need for Compression of BSN Data

As mentioned in the previous section, it is essential to compress the data from WIBSNs since the devices are small and data generated can be very frequent. The reasons for compression can be summarized as follows:

- *Battery power*: The sensors are small, and the battery has limited power.
- *Network bandwidth*: Compressed data needs less bandwidth. A lot of data from multiple sensors might be sharing the same channel to send information to the base station.
- *Data staleness*: Data loses effectiveness if not sent within a short period, especially in cases where it is life critical.
- *Data security*: Data sent in raw format could be snooped by other devices thereby compromising the privacy and security of the individual whose data is collected.

3.2 Review of research on BSN data compression

In their survey of BSNs, Lai et al. [3], mention that data compression in BSNs can be done using classical compression algorithms such as source encoding, differential encoding, and Huffman encoding. Sadler et al. [11] did a study on Data compression algorithms for energy-constrained devices and proposed a variant of Lempel Zev Welch (LZW) algorithm named s-LZW to reduce the amount of data sent across the network. Yoon et al. [12] used the s-LZW scheme as the compression method to improve energy utilization in solar-powered WSNs. It is also noted in the paper that s-LZW is a lossless compression algorithm widely used in WSNs. Wu et al. [13] did a case study on Pilates

motion recognition using BSNs, and they proposed a compression algorithm based on interception and differential encoding techniques. Hu et al. [14] analyzed bio-medical signals from low power BSNs and utilized an algorithm named Joint Orthogonal Matching Pursuit (JOMP) which could control interval times thereby reduce the amount of data processing and transmission. Charbiwala et al. [15], evaluated the effectiveness of a wireless Neural Recording System (NRS) and proposed using on-chip detection of action potentials, combined with compressive sensing techniques. Manikandan et al. [16] presented an ECG data compression algorithm based on Discrete Sinc Interpolation (DSI) technique which used an efficient Discrete Fourier Transform (DFT) to achieve compression and decompression. Tiwari et al. [17] did a survey and experimentation on classical lossless compression techniques like LZW, Huffman, and so forth and proposed a new algorithm named Aggregated Deflate-RLE (ADR) compression technique which combined Deflate and RLE compression techniques and achieved better performance.

4. EXISTING LOSS COMPRESSION TECHNIQUES: ALGORITHMS AND EXPERIMENTAL EVALUATIONS

4.1 Review of Existing Lossless Compression Techniques

A brief description about lossless compression algorithms like Run Length Encoding, Huffman Encoding, Lempel Zev Welch is given below.

4.1.1 Run Length Encoding Algorithm

Run Length Encoding is an algorithm where characters/symbols that are repeating in a sequence are coded just once. E.g., the input of WWWWWWBBBWWWB, gives the output as 6W3B3W1B when it is passed through the RLE algorithm. The algorithm is efficient if there are a lot of repeating symbols like an image with a line graph where pixels in the background color are the same.

Pseudocode [18]:

```
1 Loop: count = 0
2  REPEAT
3   get next symbol
4   count = count + 1
5  UNTIL (symbol unequal to next one)
      output symbol
6  IF count > 1
7   output count
8  GOTO Loop
```

4.1.2 Huffman Encoding Algorithm

Huffman encoding is an algorithm where symbols are encoded with bits in such a way that more frequently occurring symbols are assigned smaller bit strings. As an input Huffman would need an array or lookup table of frequencies for each symbol that may be in the dataset. The frequency can be pre-computed using a test dataset.

Pseudocode [19]:

```
1 Create a leaf node for each symbol and
     add it to the priority queue.
2 While there is more than one node in
     the queue:
3   Remove the node of highest priority
       (lowest probability) twice to get
       two nodes.
4   Create a new internal node with
       These two nodes as children and
       with probability equal to the sum
       of the two nodes' probabilities.
5   Add the new node to the queue.
6 The remaining node is the root node
     and the tree is complete.
```

4.1.3 Lempel Zev Welch (LZW) Algorithm

LZW compression is an algorithm where a sequence of symbols is mapped to a code from a lookup table. The lookup table initially has codes 0-255 to represent single bytes and the algorithm adds more symbol-sequences and codes as it reads the text.

Pseudocode [20]:

```
1 Initialize table with single character strings
2  P = first input character
3  WHILE not end of input stream
4    C = next input character
5    IF P + C is in the string table
6      P = P + C
7    ELSE
8      output the code for P
9    add P + C to the string table
10   P = C
11 END WHILE
12 output code for P
```

4.1.4 Data Chunking and TTTD Algorithm

Data Chunking is a technique primarily used in data deduplication systems for deduping data in files to reduce storage costs. During data chunking, the algorithm breaks data into smaller data elements called 'chunks'. Chunks are then fingerprinted and used later for duplicate detection. The simplest approach of data chunking, called 'Fixed size chunking', is to break the input into equal, fixed-size chunks. However, this approach has some key issues like 'Boundary Shift problem' and large Chunk size variances.

The concerns around the boundary-shift problem were addressed by content-defined chunking (CDC) algorithm which was proposed in Low Bandwidth Filesystem (LBFS) [21]. As shown in Figure 2, CDC uses a basic sliding window (BSW) technique where the sliding *window W* shifts one byte at a time from the beginning to the end of the file. During every shift, it computes a *hashvalue h* for the data in the window. The hashvalue is computed using Rabin Fingerprinting which makes it faster. The satisfying pre-condition in this instance is $(h \bmod D) = R$. The divisor D is a divisor that is chosen at the beginning depending on the average chunk size desired. R could be 0 or some number that is less than D. If the pre-condition is met, the algorithm sets that point P, as the breakpoint for the chunk boundary. Then a hash of the chunk is done and stored in memory with the key as the hash and value as either the data or compressed data. Before the hash is stored, a lookup is done to see if the hash already exists. If yes, just a pointer to the position is stored thereby reducing the space needed.

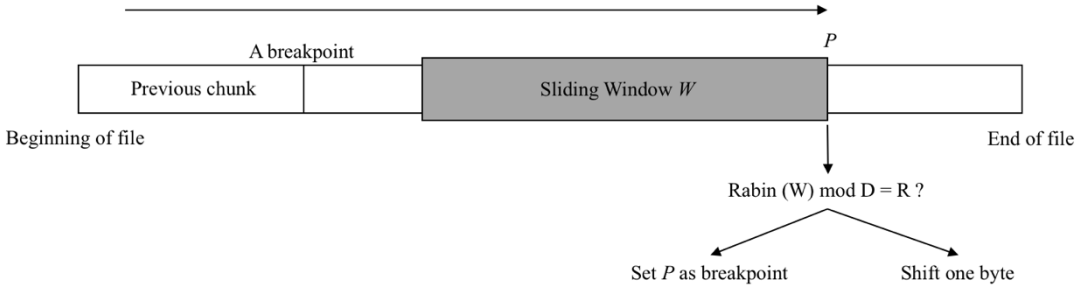


Figure 2. Sliding Window Algorithm

The TTTD algorithm, developed by HP laboratory [22] to solve the problem of large chunk sizes, uses the same concept as above with some modifications. As the name suggests, there are two thresholds, a maximum threshold (*maxT*) and a minimum threshold (*minT*), to limit the chunk sizes between two boundaries. In addition to this, there is a second divisor (second), which is used to determine backup breakpoint. The TTTD-S algorithm, developed at San Jose State University, is an improvement over TTTD where a parameter switch is used to improve the probability of using the main divisor thereby bringing the average chunk size closer to the middle of the two boundaries.

Though the TTTD algorithm is primarily used to optimize file storage in local or cloud storage systems, one of the objectives of this research is to evaluate if this algorithm could also be used for small data sizes with different parameters.

Pseudocode [26]:

```
1 int currP = 0, lastP = 0, backupBreak = 0 ;
2
3 for ( ; ! endOfFile( input ) ; currP++ ) {
4    unsigned char c = getNextByte( input ) ;
5    unsigned int hash = updateHash( c ) ;
6
7    if ( currP – lastP < minT ) {
```

```
8        continue ;
9     }
10    if (( hash % secondD ) = = secondD – 1 ) {
11       backupBreak = currP ;
12    }
13    if (( hash % mainD ) = = mainD – 1 ) {
14       addBreakpoint( currP ) ;
15       backupBreak = 0 ;
16       lastP = currP ;
17       continue ;
18    }
19    if ( currP – lastP < maxT ) {
20       continue ;
21    }
22    if ( backupBreak != 0 ) {
23       addBreakpoint( backupBreak ) ;
24       lastP = backupBreak ;
25       backupBreak = 0 ;
26    }
27    else {
28       addBreakpoint( currP ) ;
29       lastP = currP ;
30       backupBreak = 0
31    }
32 }
```

4.2 Experimental Evaluations

The metrics used for the analysis of the runtime results of the compression algorithms above are outlined below.

4.2.1 Metrics for Analysis

TABLE 2. METRICS FOR COMPRESSION ANALYSIS

Metric	Description	Formula
File Size (FS)	File size in *bytes*	-
Compressed File Size (CS)	File size of the compressed file in *bytes*	-

Metric	Description	Formula
Compression Time (CT)	Time taken to compress the file in *milliseconds*	-
Compression Ratio (CR)	*Ratio* of compressed file size to Original file size	*CS / FS*
Savings Percentage (SP)	*Percentage* of reduction in file size after compression	*100 * ((FS – CS) / FS)*
Compression Factor (CF)	*Ratio* of Original file size to Compressed file size	*FS / CS*

4.2.2 Experimental Objectives

The main objective of the experiments was to run the various algorithms to compute FS, CS, CT, CR, SP and CF for files with varying sizes. The data characteristics are similar among files. All files were data from Fitbit [23], a wearable device that has sensors like accelerometers and gyroscopes to measure steps and activity intensities during the course of a day. Files of smaller sizes were chosen since sensors typically do not have large RAMs available.

4.2.3 Experimental Configurations

The experiments were conducted on a machine with following hardware:

- 2.8 GHz Intel Core i7 2.8 GHz processor
- 16 GB 1600 MHz DDR3 Memory.
- 500GB Flash hard disk drive.

The configurations for the algorithms were as follows:

- *RLE*: Regex to find letters “[0-9]+|[a-zA-Z]”.
- *Huffman*: Size of frequency count table, R = 256.

- *LZW*: Number of input chars, R = 256; Codeword width, W = 12; Number of codewords, L = 2^W = 4096.
- *TTTD*: Prior to running this experiment, TTTD algorithm was run on the dataset with different sets of parameters. The initial trial was with the defaults from the original paper's optimal values by Eshghi, *et al.* [22], which were window size of 48 bytes, main divisor as 540, secondary divisor as 270, *maxT* and *minT* as 2800 and 460 respectively. These values resulted in the compressed file size being the same as the original file. After different trials, the following parameters were arrived at. These gave an optimal (>50%) reduction in the file sizes.

TABLE 3. TTTD PARAMETERS

Parameter Name	Value
Window Size (bytes)	4
Main Divisor (mainD)	540
Second Divisor (secondD)	270
Maximum Threshold (maxT)	15
Minimum Threshold (minT)	5

TABLE 4. DATASETS USED FOR THE EXPERIMENTS (FITBIT DATA)

Dataset No.	Data Name	Data Type	File Size (Bytes)
#1	Steps per day	*.csv	25,175
#2	Intensities per day	*.csv	70,581
#3	Intensities per hour	*.csv	482,671
#4	Steps per hour	*.csv	796,562
#5	Steps per minute	*.csv	3,481,174

4.3 Experimental Results

The dataset was compressed using the algorithms mentioned above, and results were recorded to have a side-by-side comparison.

TABLE 5. COMPARISON OF VARIOUS COMPRESSION ALGORITHMS

	FS	CS	CT	CR	SP
Huffman	25,175	11,971	130	0.476	52.45
	70,581	32,634	138	0.462	53.76
	482,671	242,107	155	0.502	49.84
	796,562	399,955	181	0.502	49.79
	3,481,174	1,137,343	311	0.327	67.33
LZW	25,175	9,200	203	0.365	63.46
	70,581	26,825	423	0.380	61.99
	482,671	161,429	7398	0.334	66.56
	796,562	268,850	18865	0.338	66.25
	3,481,174	621,182	199799	0.178	82.16
RLE	25,175	47,021	135	1.868	-86.78
	70,581	117,753	140	1.668	-66.83
	482,671	851,493	182	1.764	-76.41
	796,562	1,432,065	208	1.798	-79.78
	3,481,174	6,769,007	455	1.944	-94.45
TTTD	25,175	24,262	199	0.964	3.63
	70,581	65,543	249	0.929	7.14
	482,671	174,672	402	0.362	63.81
	796,562	363,847	459	0.457	54.32

	FS	CS	CT	CR	SP
	3,481,174	989,000	1090	0.284	71.59

4.4 Experimental Observations

- *Compressed File Size (CS):* The compressed file for RLE was larger than the original file, and hence RLE would not be a good fit for this type of data. TABLE 5. shows that compressed file size grows linearly with input size. LZW performs better than Huffman and TTTD algorithms in terms of compressed file size.
- *Compression Ratio (CR):* Compression Ratio is consistent for Huffman Algorithm for files of 25KB, 71KB, 483KB and 796KB files. For 3.5MB file, the ratio is better than smaller files. Of all the algorithms, LZW had the best CR for all sizes of files, followed by TTTD and then Huffman. LZW and TTTD algorithms perform much better for larger files than smaller ones in terms of CR.
- *Compression Factor (CF):* Since Compression Factor is reciprocal of Compression Ratio, the observations there apply to this also. Huffman is quite stable and LZW has the best compression factor.
- *Savings Percentage (SP):* Huffman is stable in terms of Savings percentage. LZW has higher SP in comparison to Huffman but not by a great margin. TTTD has lower SP in comparison to Huffman for smaller files but better SP in comparison to Huffman for larger files.
- *Compression Time (CT):* Compression time is very high for LZW in comparison to other algorithms. LZW took 199 seconds in comparison to 0.3 seconds for Huffman

and 1.1 seconds for TTTD. Huffman is best in terms of Compression time, followed by RLE and then TTTD.

5. PROPOSED METHOD

5.1 Experiment with Combination of Algorithms

Comparisons of algorithms in the last section gave the following insights:

- TTTD and LZW outranked other algorithms in terms of compressed file size, compression ratio, compression factor and savings percentage.
- TTTD and Huffman outranked other algorithms in terms of Compression Time.
- RLE was not suitable for this type of data since the Compression ratio was greater than 1.

The above insights brought forth the idea that perhaps multiple algorithms could be used to compress the compressed data and give an overall efficiency to the system. Hence in the next stage of experiments, the algorithms – Huffman, LZW and TTTD were run in a sequence in different orders with the output from the first algorithm being the input for the second. TABLE 6. has the results of performance of different combinations. The sequence *TTTD -> LZW -> Huffman (TLH)* performed better than others in terms of Compression Ratio. The sequence *TTTD -> Huffman -> LZW (THL)* performed better in terms of Compression Time. Since LZW takes time, when the experiment was run by removing LZW from the chain, the *TTDD->Huffman (TH)* algorithm performed much better in terms of time and was only fractionally different in terms of Compression Ratio and Savings percentage.

TABLE 6. PERFORMANCE OF ALGORITHMS IN DIFFERENT COMBINATIONS

Algorithm	FS	CS	CT	CR	SP
HLT (Huffman-LZW-TTTD)	3481174	760956	76087	0.219	78.14
HTL (Huffman-TTTD-LZW)	3481174	700017	52449	0.201	79.89
LHT (LZW-Huffman-TTTD)	3481174	613565	194295	0.176	82.37
LTH (LZW-TTTD-Huffman)	3481174	611280	194090	0.176	82.44
THL (TTTD-Huffman-LZW)	3481174	455984	18517	0.131	86.90
TLH (TTTD-LZW-Huffman)	3481174	353831	31703	0.102	89.84
TH (TTTD-Huffman)	**3481174**	**408642**	**1249**	**0.117**	**88.26**
TL (TTTD-LZW)	3481174	355863	31840	0.102	89.78

Based on the results, the proposal is to run TTTD and Huffman in sequence on the data, with the output of TTTD being the input of Huffman.

5.2 Outline of the Proposed Algorithm

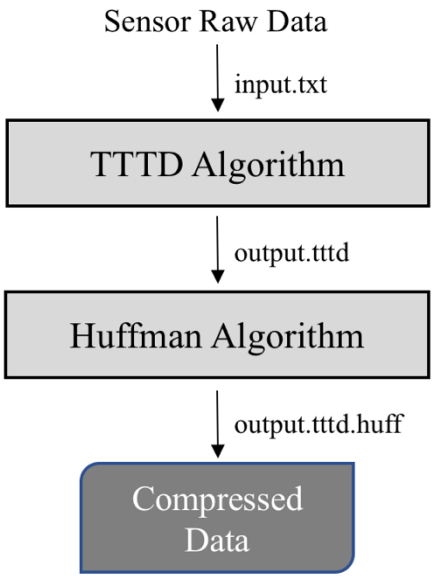


Figure 3. Outline of the Proposed Algorithm

6. PERFORMANCE EVALUATION

The proposed method was run on the data used in the baseline experiments (Fitbit data) and also on additional data to support the research. The second set of data was obtained from UCI (University of California, Irvine)'s 'Heterogeneity Activity Recognition Data Set'. Fitbit dataset had data about steps and intensities recorded by Fitbit wearable device. The Smartphone dataset has data from readings that were recorded while users executed activities carrying smartwatches and smartphones.

6.1 Fitbit Dataset

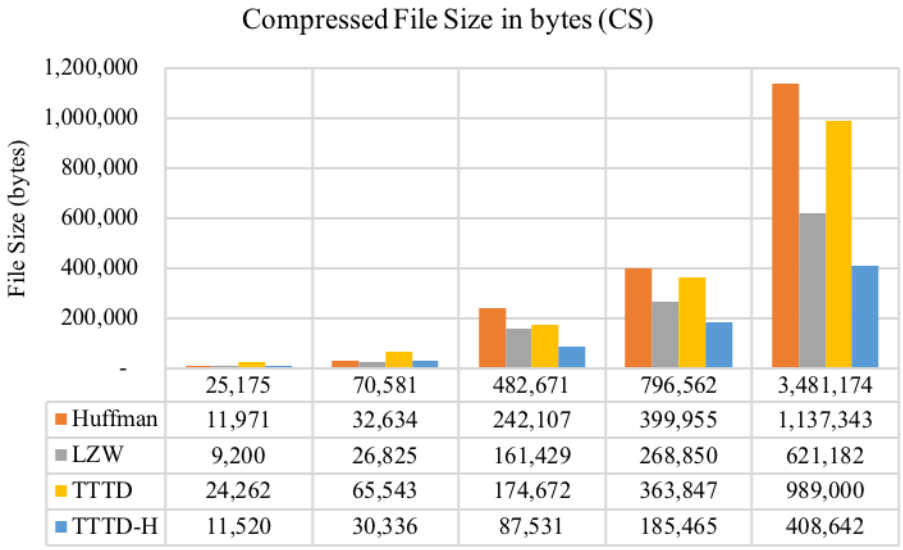
The metrics used for the analysis of the runtime results of the proposed compression algorithm are as outlined in Section 4. Below, the results are compared with the existing techniques.

TABLE 7. PERFORMANCE OF PROPOSED ALGORITHM IN COMPARISON TO OTHER COMPRESSION ALGORITHMS

Sl. No	Algorithm	FS (bytes)	CS (bytes)	CT (ms)	CR	SP	CF
1	Huffman	25,175	11,971	130	0.476	52.45	2.10
2	Huffman	70,581	32,634	138	0.462	53.76	2.16
3	Huffman	482,671	242,107	155	0.502	49.84	1.99
4	Huffman	796,562	399,955	181	0.502	49.79	1.99
5	Huffman	3,481,174	1,137,343	311	0.327	67.33	3.06
6	LZW	25,175	9,200	203	0.365	63.46	2.74
7	LZW	70,581	26,825	423	0.380	61.99	2.63
8	LZW	482,671	161,429	7398	0.334	66.56	2.99
9	LZW	796,562	268,850	18865	0.338	66.25	2.96

10	LZW	3,481,174	621,182	199799	0.178	82.16	5.60
11	TTTD	25,175	24,262	199	0.964	3.63	1.04
12	TTTD	70,581	65,543	249	0.929	7.14	1.08
13	TTTD	482,671	174,672	402	0.362	63.81	2.76
14	TTTD	796,562	363,847	459	0.457	54.32	2.19
15	TTTD	3,481,174	989,000	1090	0.284	71.59	3.52
16	TTTD-H	25,175	11,520	398	0.458	54.24	2.19
17	TTTD-H	70,581	30,336	404	0.430	57.02	2.33
18	TTTD-H	482,671	87,531	566	0.181	81.87	5.51
19	TTTD-H	796,562	185,465	616	0.233	76.72	4.29
20	TTTD-H	3,481,174	408,642	1263	0.117	88.26	8.52

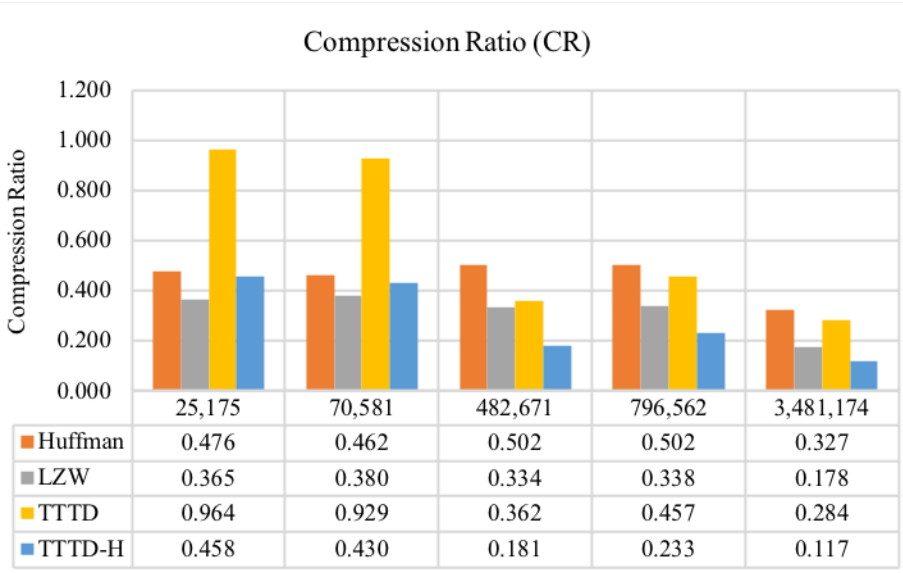
1) *Compressed File Size:* As shown in Figure 4, compressed size for TTTD-H is much lower than Huffman, LZW and TTTD. It is almost half of what Huffman or TTTD has individually. For small file sizes, LZW is better than other algorithms. For 25KB file, LZW compresses it to 9KB while TTTD is only 24KB. The proposed algorithm, TTTD-H is closer to LZW in terms of compressed file size, compressing the input to 11.5KB. As the file size becomes larger, especially after 100KB, TTTD and TTTD-H easily trumps other algorithms. For a 3.5 MB file, the output from LZW is 621KB while the output from TTTD-H is only 408KB, 50% better than LZW.



	25,175	70,581	482,671	796,562	3,481,174
Huffman	11,971	32,634	242,107	399,955	1,137,343
LZW	9,200	26,825	161,429	268,850	621,182
TTTD	24,262	65,543	174,672	363,847	989,000
TTTD-H	11,520	30,336	87,531	185,465	408,642

Figure 4. Comparison of Compressed file sizes with TTTD-H

2) *Compression Ratio*: As shown in Figure 5, the compression Ratio for TTTD-H is lower than Huffman, LZW and TTTD. For smaller files, the compression ratio of TTTD-H is comparable to that of Huffman and LZW, but for larger files, TTTD-H is much better. The ratio is 11.7% for a 3.5MB file in comparison to 17.8% for LZW and 32.7% for Huffman.

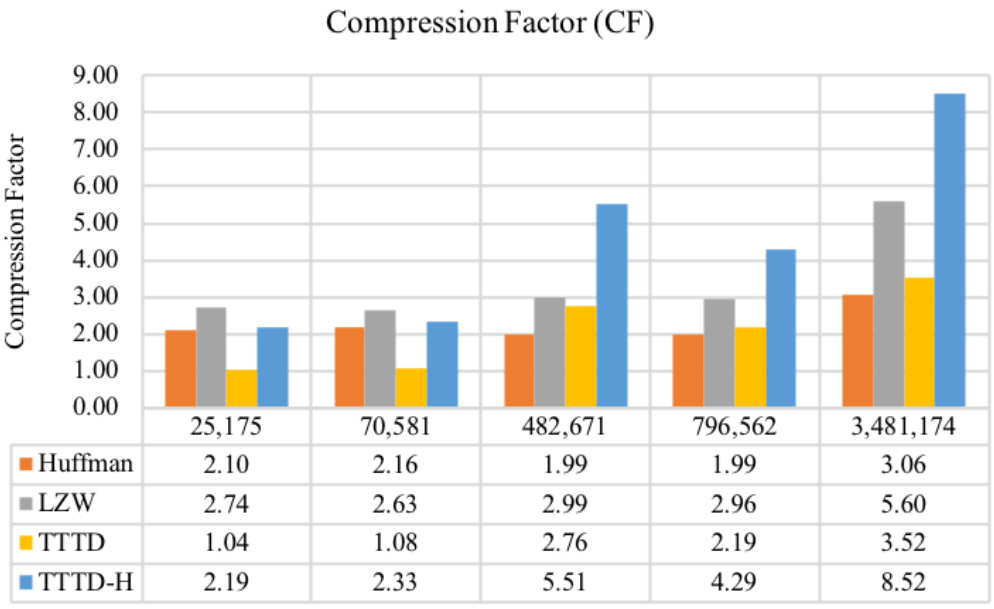


	25,175	70,581	482,671	796,562	3,481,174
Huffman	0.476	0.462	0.502	0.502	0.327
LZW	0.365	0.380	0.334	0.338	0.178
TTTD	0.964	0.929	0.362	0.457	0.284
TTTD-H	0.458	0.430	0.181	0.233	0.117

Figure 5. Comparison of Compression Ratio with TTTD-H

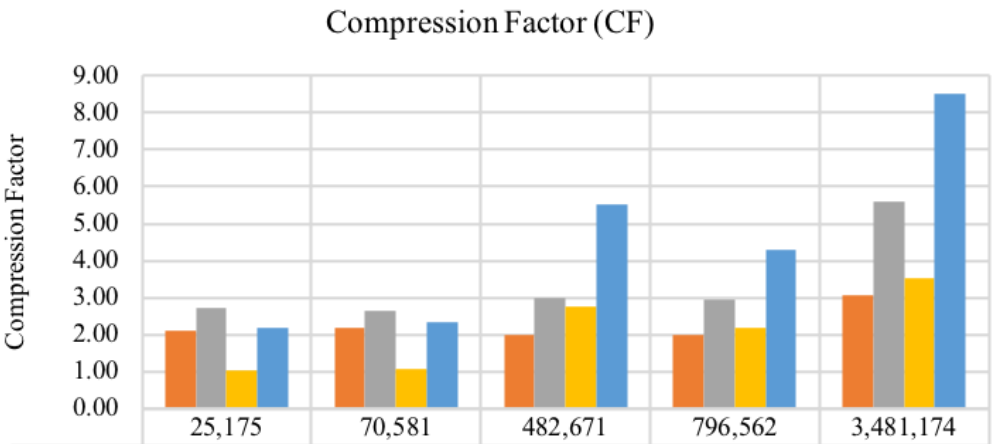
3) *Compression Factor*: Compression factor is the reciprocal of Compression Ratio and the values mirror the results in Compression Ratio chart in the other direction. As illustrated in

4)



	25,175	70,581	482,671	796,562	3,481,174
Huffman	2.10	2.16	1.99	1.99	3.06
LZW	2.74	2.63	2.99	2.96	5.60
TTTD	1.04	1.08	2.76	2.19	3.52
TTTD-H	2.19	2.33	5.51	4.29	8.52

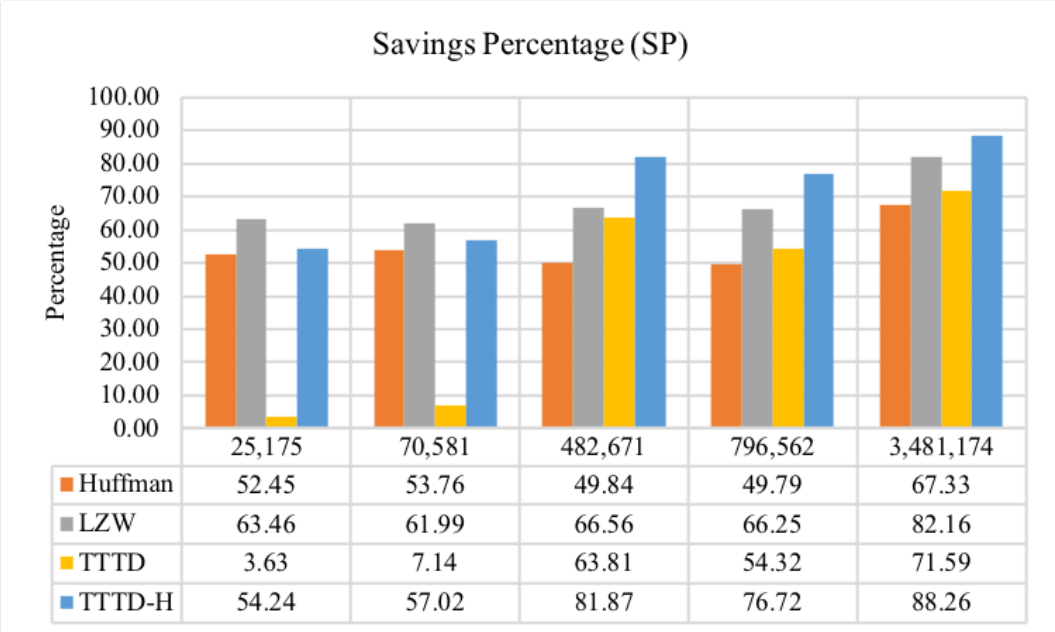
5) Figure 6, the Compression Factor for TTTD-H is higher than Huffman, LZW and TTTD. Compression factor for TTTD-H is 8.52 for a 3.5MB file in comparison to that of 5.6 for LZW and 3.06 for Huffman.



	25,175	70,581	482,671	796,562	3,481,174
Huffman	2.10	2.16	1.99	1.99	3.06
LZW	2.74	2.63	2.99	2.96	5.60
TTTD	1.04	1.08	2.76	2.19	3.52
TTTD-H	2.19	2.33	5.51	4.29	8.52

Figure 6. Comparison of Compression Factor with TTTD-H

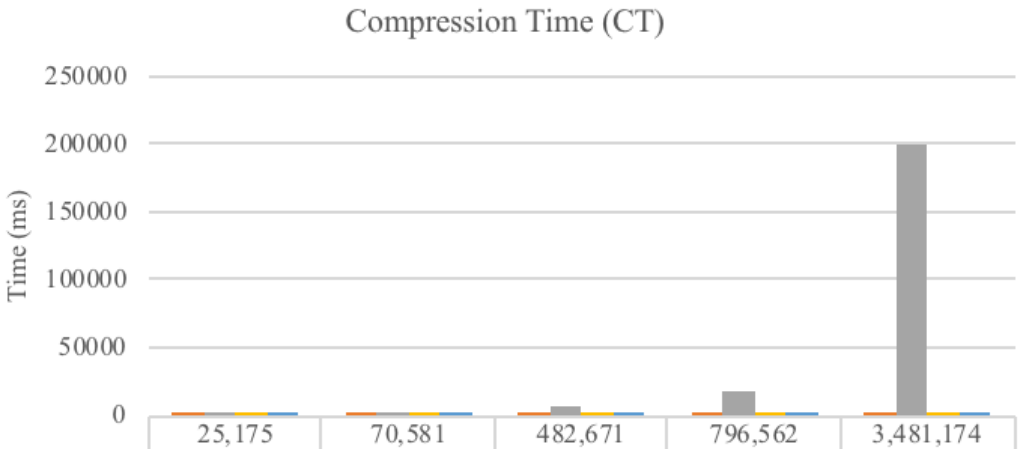
6) *Savings Percentage*: From Figure 7, it is clear that the Savings Percentage for TTTD-H is higher than Huffman, LZW and TTTD. The Savings Percentage is 54% for TTTD-H for 25KB file in comparison to 63% for LZW and 52% for Huffman. For larger files, however, TTTD-H has much better savings which are in line with Compression Ratio. For a 3.5MB file, the Savings percentage is 88% for TTTD-H much higher than other algorithms.



	25,175	70,581	482,671	796,562	3,481,174
Huffman	52.45	53.76	49.84	49.79	67.33
LZW	63.46	61.99	66.56	66.25	82.16
TTTD	3.63	7.14	63.81	54.32	71.59
TTTD-H	54.24	57.02	81.87	76.72	88.26

Figure 7. Comparison of Savings Percentage with TTTD-H

7) *Compression Time*: Illustrated in

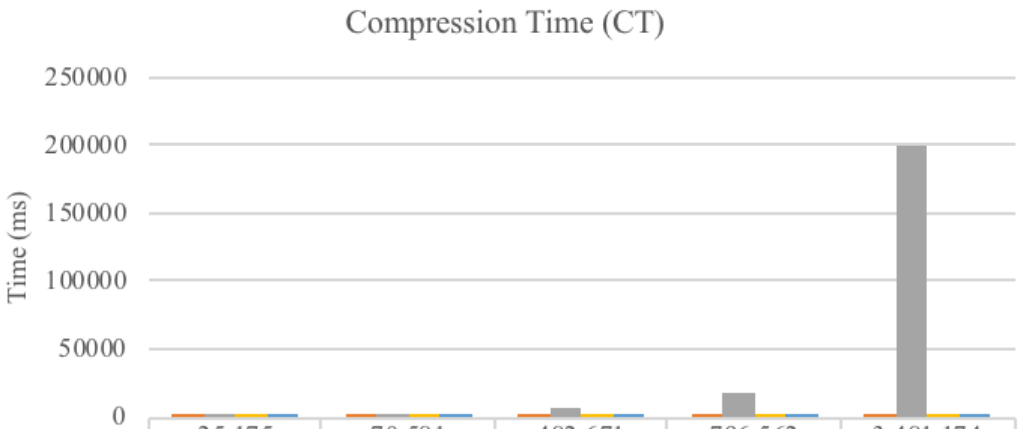


	25,175	70,581	482,671	796,562	3,481,174
Huffman	130	138	155	181	311
LZW	203	423	7398	18865	199,799
TTTD	199	249	402	459	1090
TTTD-H	398	404	566	616	1263

8)

9) Figure 8, the Compression Time for TTTD-H is higher than that of TTTD or Huffman but lower than LZW. LZW took close to 200,000 milliseconds to compress

a file of size 3.5MB. TTTD-H in comparison took only 1,200 milliseconds. As mentioned earlier, the value for TTTD-H is higher than other but the Compression Ratio and Savings Percentages are much higher. Since the value is only 0.2 seconds higher than TTTD for 3.5 MB file, this is the small increase to pay for the extra compression ratio. The benefit from making network bandwidth better by reducing the amount of data sent during I/O calls outweighs the Compression Time concerns.



	25,175	70,581	482,671	796,562	3,481,174
Huffman	130	138	155	181	311
LZW	203	423	7398	18865	199,799
TTTD	199	249	402	459	1090
TTTD-H	398	404	566	616	1263

Figure 8. Comparison of Compression Time with TTTD-H

6.2 Smartphone & Smartwatch Dataset

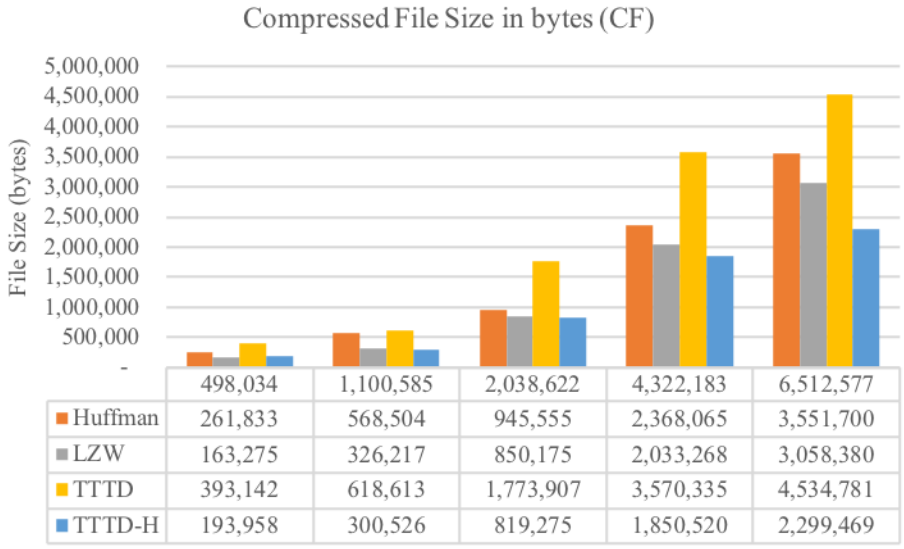
Further experiments were conducted with different sets of wearable sensor data to determine if the results were consistent. UCI dataset on *Heterogeneity Activity Recognition* [28] was used for this experiment. This dataset contains the readings of motion sensors from smartphones to track activities like ‘Biking’, ‘Walking’, ‘Walking

up the stairs', and so forth. The data was from devices like Samsung Galaxy S3, LG Nexus, Galaxy Gear, and so forth.

TABLE 8. DATASET USED FOR THE EXPERIMENTS (SMARTPHONE DATA)

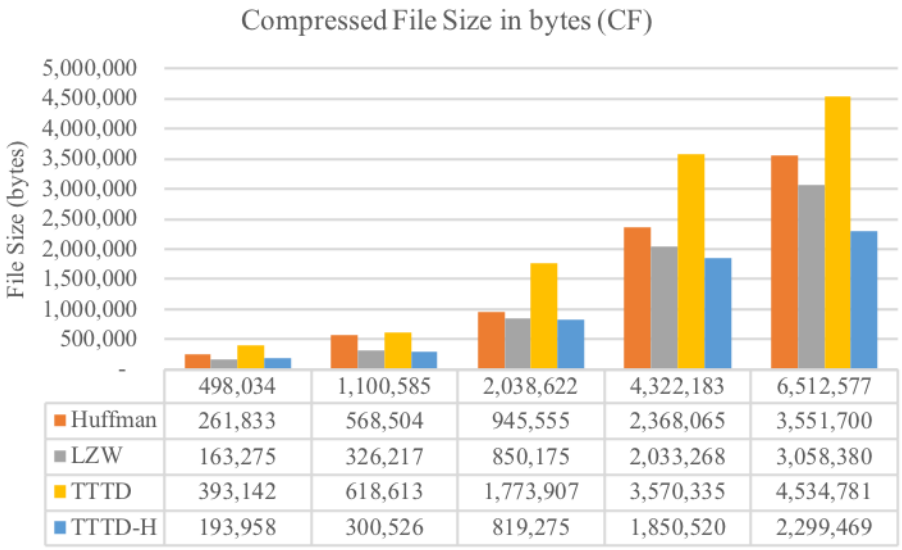
Dataset No.	Data Name	Data Type	File Size (Bytes)
#1	Phones accelerometer	*.csv	498,034
#2	Phones gyroscope	*.csv	1,100,585
#3	Samsung Galaxy Gear	*.csv	2,038,622
#4	Watch accelerometer	*.csv	4,322,183
#5	Watch gyroscope	*.csv	6,512,577

1) *Compressed File Size:* As shown in



	498,034	1,100,585	2,038,622	4,322,183	6,512,577
Huffman	261,833	568,504	945,555	2,368,065	3,551,700
LZW	163,275	326,217	850,175	2,033,268	3,058,380
TTTD	393,142	618,613	1,773,907	3,570,335	4,534,781
TTTD-H	193,958	300,526	819,275	1,850,520	2,299,469

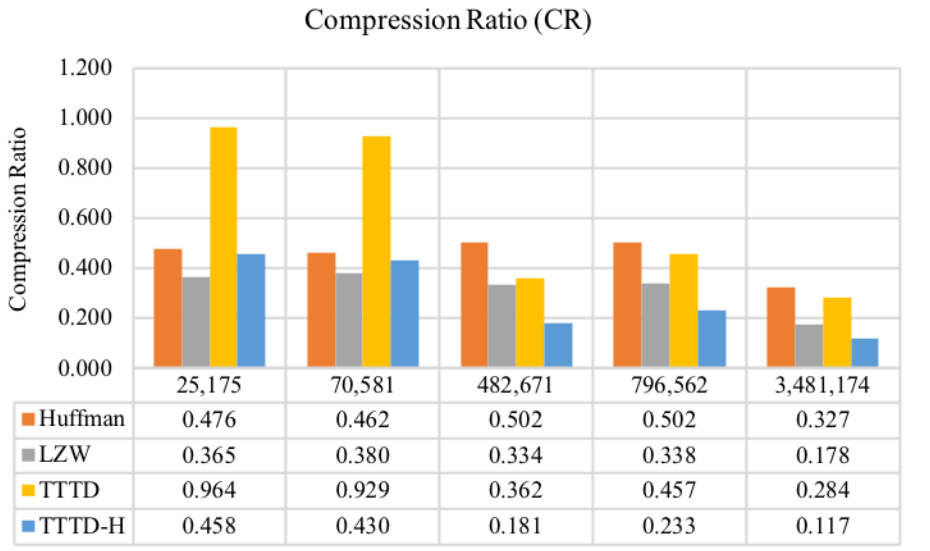
2) Figure 9, compressed size for TTTD-H is much lower than Huffman, LZW and TTTD. It is almost half of what Huffman or TTTD has individually.



	498,034	1,100,585	2,038,622	4,322,183	6,512,577
Huffman	261,833	568,504	945,555	2,368,065	3,551,700
LZW	163,275	326,217	850,175	2,033,268	3,058,380
TTTD	393,142	618,613	1,773,907	3,570,335	4,534,781
TTTD-H	193,958	300,526	819,275	1,850,520	2,299,469

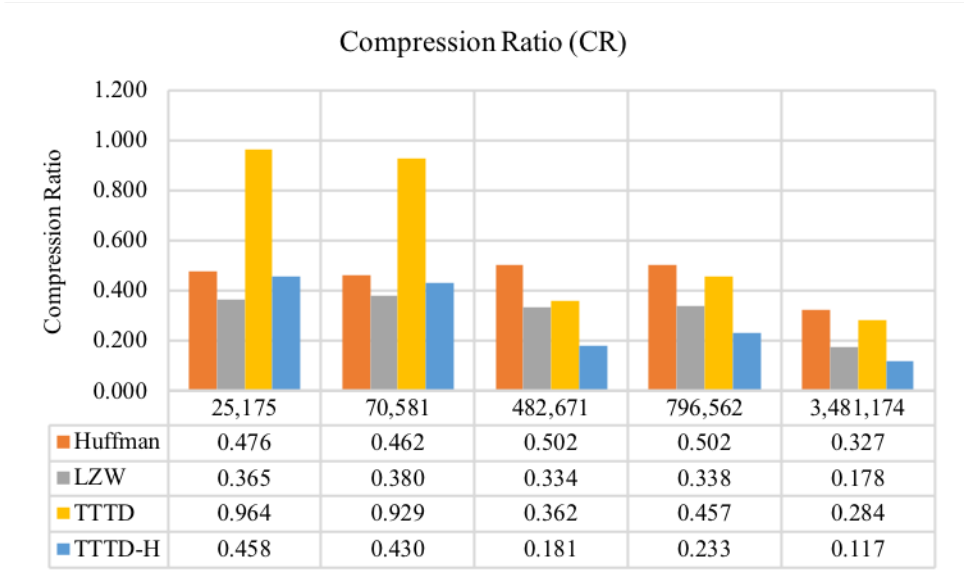
Figure 9. Comparison of Compressed file sizes with TTTD-H

3) *Compression Ratio*: As shown in



	25,175	70,581	482,671	796,562	3,481,174
Huffman	0.476	0.462	0.502	0.502	0.327
LZW	0.365	0.380	0.334	0.338	0.178
TTTD	0.964	0.929	0.362	0.457	0.284
TTTD-H	0.458	0.430	0.181	0.233	0.117

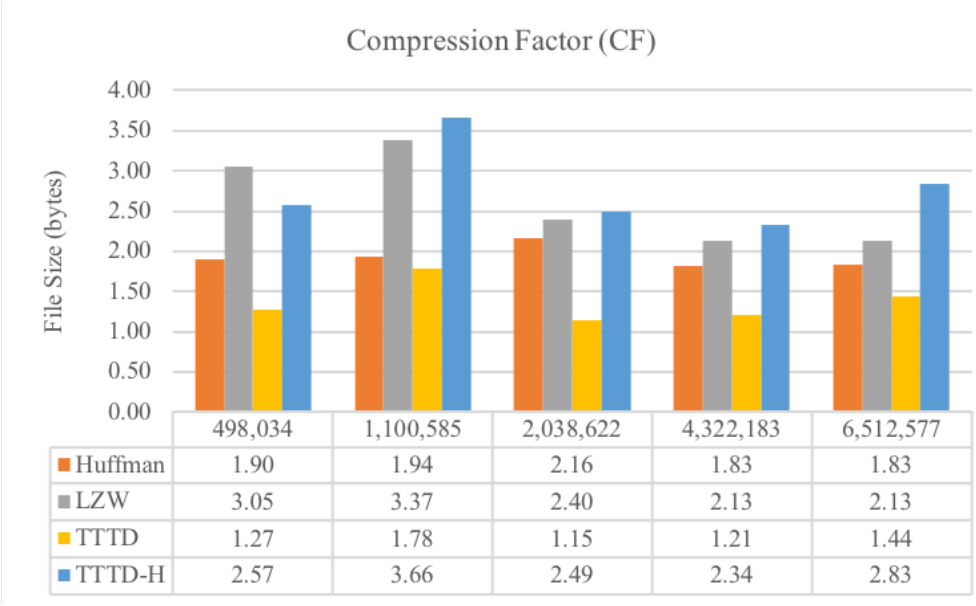
4) Figure 10, the compression Ratio for TTTD-H is lower than Huffman, LZW and TTTD.



	25,175	70,581	482,671	796,562	3,481,174
Huffman	0.476	0.462	0.502	0.502	0.327
LZW	0.365	0.380	0.334	0.338	0.178
TTTD	0.964	0.929	0.362	0.457	0.284
TTTD-H	0.458	0.430	0.181	0.233	0.117

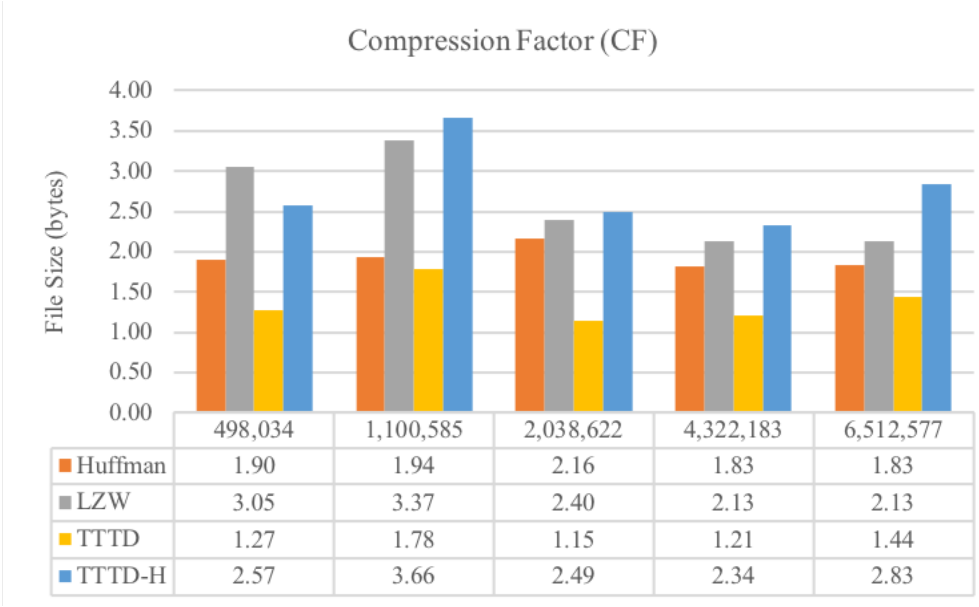
Figure 10. Comparison of Compression Ratio with TTTD-H

5) *Compression Factor*: As illustrated in



	498,034	1,100,585	2,038,622	4,322,183	6,512,577
Huffman	1.90	1.94	2.16	1.83	1.83
LZW	3.05	3.37	2.40	2.13	2.13
TTTD	1.27	1.78	1.15	1.21	1.44
TTTD-H	2.57	3.66	2.49	2.34	2.83

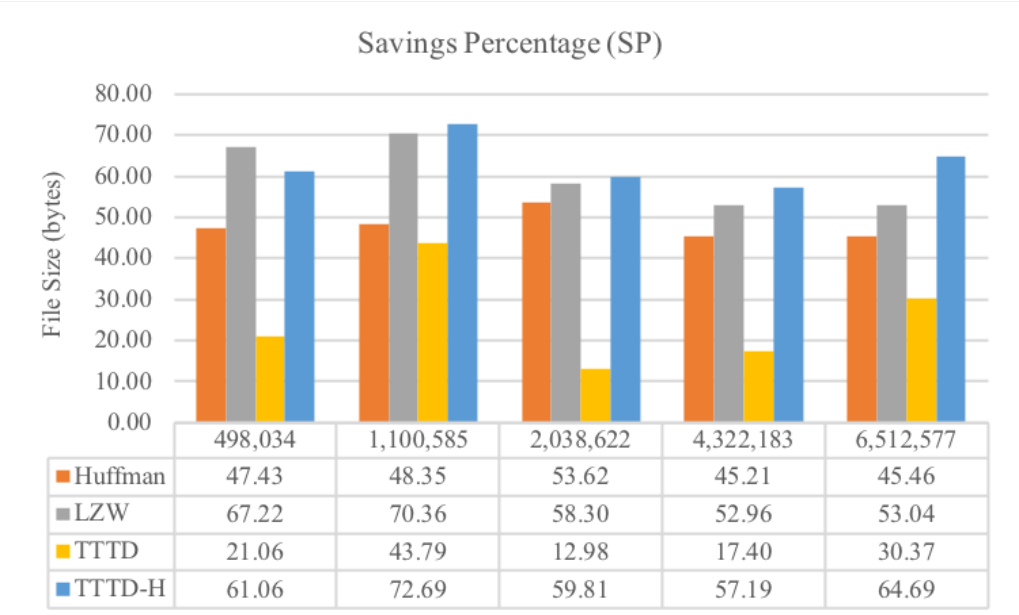
6) Figure 11, the Compression Factor for TTTD-H is higher than Huffman, LZW and TTTD.



	498,034	1,100,585	2,038,622	4,322,183	6,512,577
Huffman	1.90	1.94	2.16	1.83	1.83
LZW	3.05	3.37	2.40	2.13	2.13
TTTD	1.27	1.78	1.15	1.21	1.44
TTTD-H	2.57	3.66	2.49	2.34	2.83

Figure 11. Comparison of Compression Factor with TTTD-H

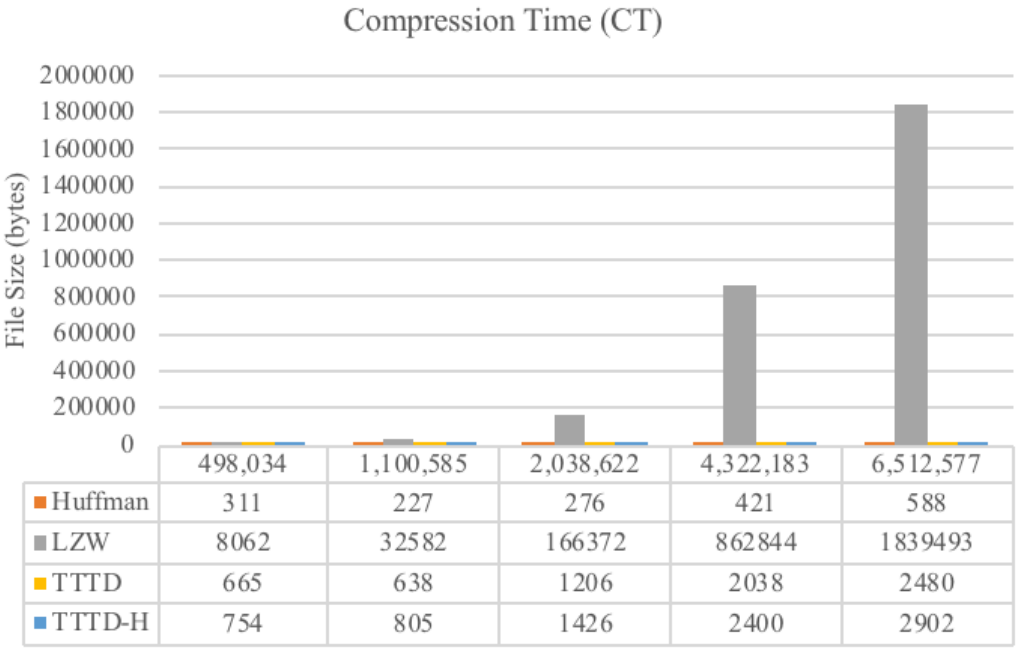
7) *Savings Percentage*: From Figure 12, it is clear that the Saving Percentage for TTTD-H is higher than Huffman, LZW and TTTD.



	498,034	1,100,585	2,038,622	4,322,183	6,512,577
Huffman	47.43	48.35	53.62	45.21	45.46
LZW	67.22	70.36	58.30	52.96	53.04
TTTD	21.06	43.79	12.98	17.40	30.37
TTTD-H	61.06	72.69	59.81	57.19	64.69

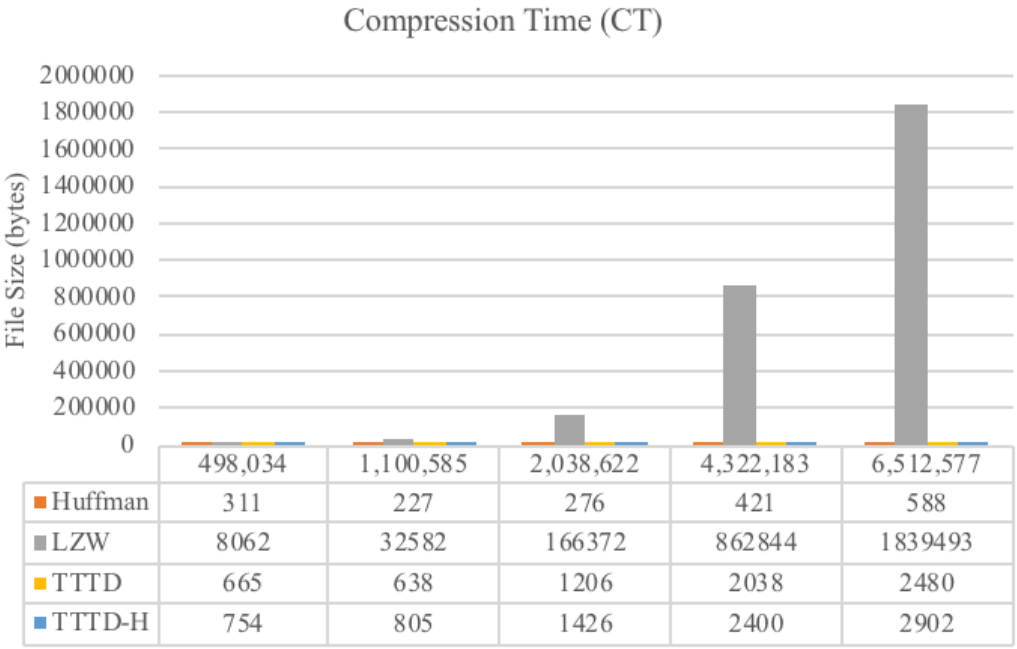
Figure 12. Comparison of Savings Percentage with TTTD-H

Compression Time: Illustrated in



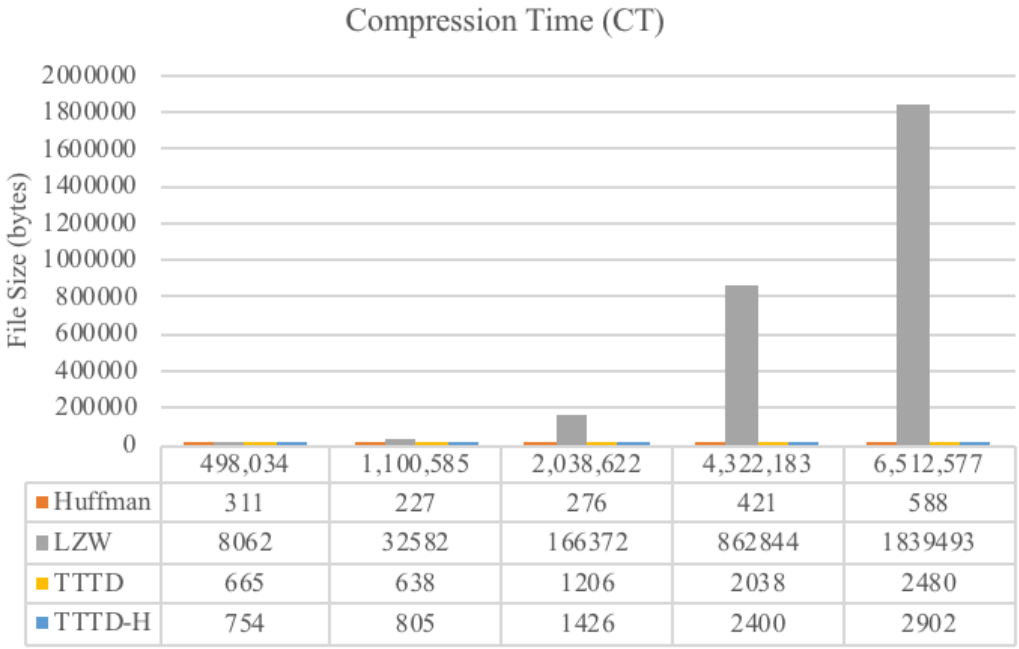
	498,034	1,100,585	2,038,622	4,322,183	6,512,577
Huffman	311	227	276	421	588
LZW	8062	32582	166372	862844	1839493
TTTD	665	638	1206	2038	2480
TTTD-H	754	805	1426	2400	2902

8) Figure 13, the Compression Time for TTTD-H is higher than that of TTTD or Huffman but much lower than LZW. The Savings Percentage and Compression Ratio is much higher for TTTD-H.



	498,034	1,100,585	2,038,622	4,322,183	6,512,577
Huffman	311	227	276	421	588
LZW	8062	32582	166372	862844	1839493
TTTD	665	638	1206	2038	2480
TTTD-H	754	805	1426	2400	2902

Figure 13



	498,034	1,100,585	2,038,622	4,322,183	6,512,577
Huffman	311	227	276	421	588
LZW	8062	32582	166372	862844	1839493
TTTD	665	638	1206	2038	2480
TTTD-H	754	805	1426	2400	2902

Figure 13. Comparison of Compression Time with TTTD-H

6.3 Analysis of Energy Consumption

6.3.1 Formula for Energy Consumption

Another important criteria to decide on the compression algorithm for BSN data is the rate of energy consumption at the sensor. As mentioned in Section *3.1 The Need for Compression of BSN Data*, reducing the battery power is one of the main goals of compression. Thus, the amount of extra energy needed for compression should be less than the amount of energy that is saved due to the reduced number of bits that are transmitted. Shin *et al.* [29], in their paper on '*Analysis of Low Power Sensor Node Using Data Compression'* formulated this as follows –

$$P_{no\_compression} = P_{memory} + P_{sensing} + P_{processing} + P_{transmission}$$

$$P_{with\_compression} = P_{no\_compression} + \Delta P_{memory} + \Delta P_{processing} - \Delta P_{transmission}$$

where

- $\Delta P_{memory} + \Delta P_{processing}$ are the power consumption increases for extra memory and processing that occurs due to compression.
- $\Delta P_{transmission}$ is the power saving due to lesser amount of data that is being sent.

For compression to be useful, the following condition should be satisfied

$$\Delta P_{memory} + \Delta P_{processing} < \Delta P_{transmission}$$

The amount of energy needed for computation is more than that needed for storing data in memory. Since $\Delta P_{processing}$ is the upper-bound, we could simplify the above formula as

$$2\Delta P_{processing} < \Delta P_{transmission}$$

In their paper on 'Energy-aware lossless data compression', Barr & Asanović [30], performed experiments and summarized the energy consumptions for computations and transmissions of sensor data. As per the analysis, the energy used by the processor for a single ADD is 0.86 nJ (0.86×10^{-9} J) and the energy used for sending a single bit is between 417 nJ (417×10^{-9} J) and 1090 nJ (1090×10^{-9} J). The paper mentions that sending a single bit is equivalent to performing 485 to 1267 ADD operations. For our computations, we take the average as 700 nJ for sending a single bit. Also, cache miss consumes 78.34 nJ of energy per bit for writing data into memory and cache hit needs 2.41×10^{-9} J of energy. To summarize,

- The energy used for computation = 0.86×10^{-9} J
- The energy used for writing data into memory = 78.34×10^{-9} J

- The energy used for reading data from memory = 2.41 x 10^{-9} J
- The energy used for transmission of 1 bit = 700 x 10^{-9} J

If we take the first experiment with Huffman as an example,

$\Delta P_{processing} = \Delta P_{cache\_hit} + \Delta P_{cache\_miss}$

$\Delta P_{processing} = (FS - CS) * 8 * 2.41 * 10^{-9} + CS * 8 * 78.34 * 10^{-9} =$ **7.76×10^{-3} J**

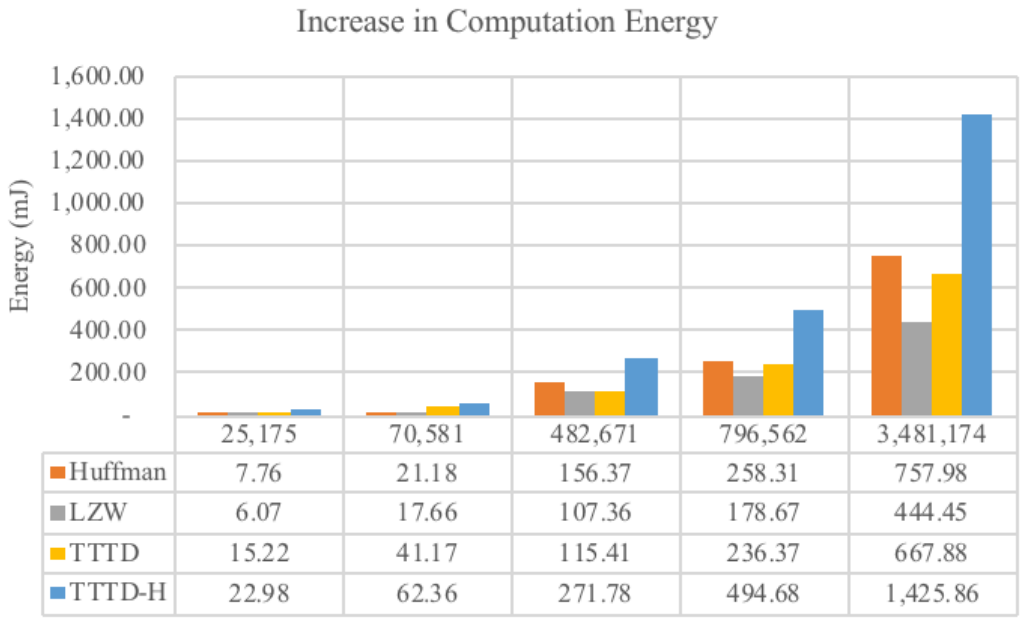
where FS = File Size and CS = Compressed File Size, both in bytes.

$\Delta P_{transmission} = (FS - CS) * 8 * 700 * 10^{-9} =$ **73.94×10^{-3} J**

We see that $2\Delta P_{processing} < \Delta P_{transmission}$ holds true and Energy saved can be calculated as $\Delta P_{transmission} - 2\Delta P_{processing} = 58.43$ mJ.

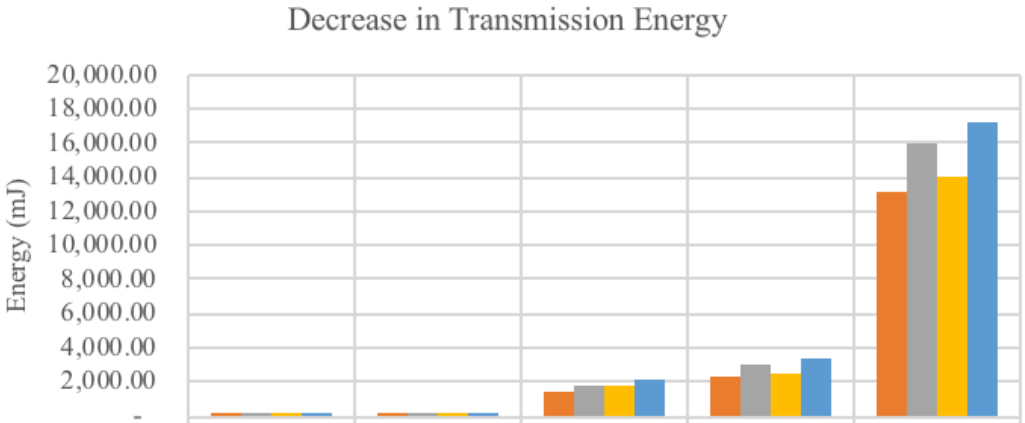
6.3.2 Computation of Energy Savings due to compression

Figures 14-19 show the Increase in processing time, decrease in transmission time and the energy saved due to compression for datasets #1 & #2. The actual energy saved due to compression might be larger than the values in the table since we made assumptions on upper bounds for energy consumption due to increased storage in memory.



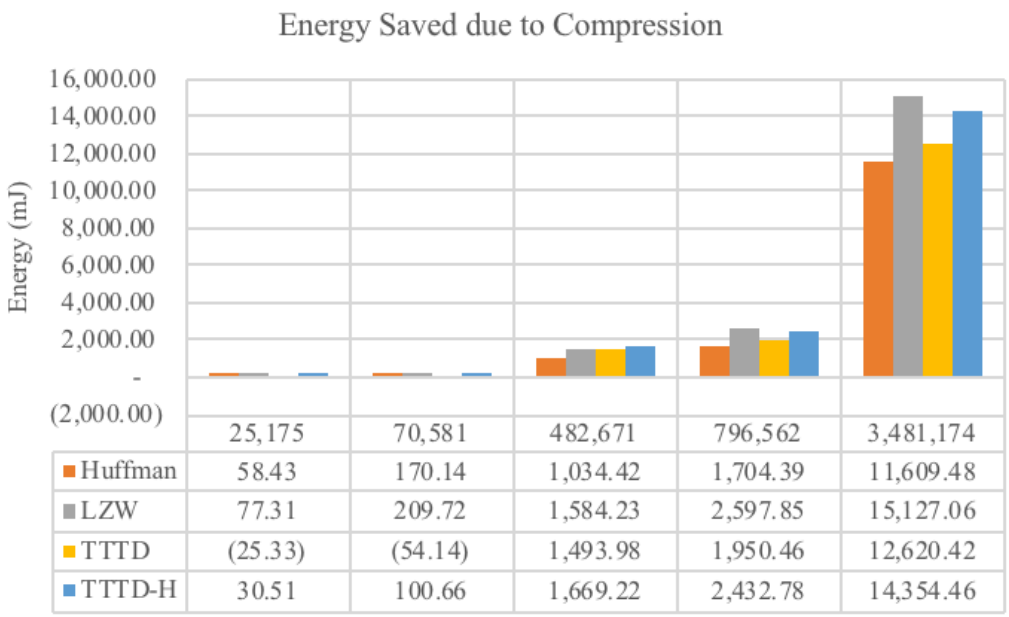
	25,175	70,581	482,671	796,562	3,481,174
Huffman	7.76	21.18	156.37	258.31	757.98
LZW	6.07	17.66	107.36	178.67	444.45
TTTD	15.22	41.17	115.41	236.37	667.88
TTTD-H	22.98	62.36	271.78	494.68	1,425.86

Figure 14. Dataset 1 - Comparison of Increase in Energy during Computation



	25,175	70,581	482,671	796,562	3,481,174
Huffman	73.94	212.50	1,347.16	2,221.00	13,125.45
LZW	89.46	245.03	1,798.96	2,955.19	16,015.96
TTTD	5.11	28.21	1,724.79	2,423.20	13,956.17
TTTD-H	76.47	225.37	2,212.78	3,422.14	17,206.18

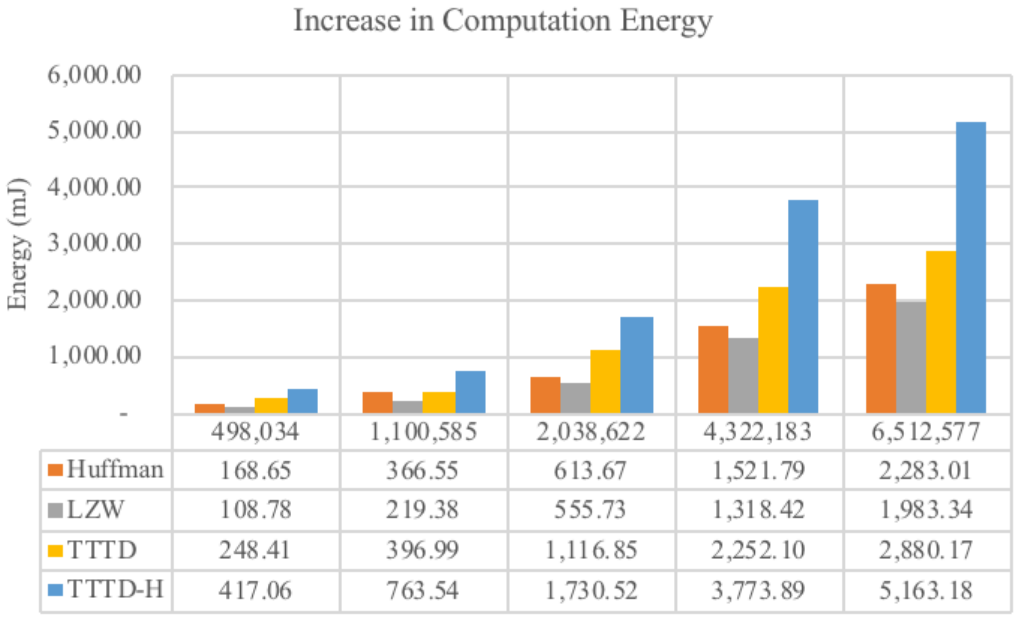
Figure 15. Dataset 1 - Comparison of Decrease in Energy during Transmission



	25,175	70,581	482,671	796,562	3,481,174
Huffman	58.43	170.14	1,034.42	1,704.39	11,609.48
LZW	77.31	209.72	1,584.23	2,597.85	15,127.06
TTTD	(25.33)	(54.14)	1,493.98	1,950.46	12,620.42
TTTD-H	30.51	100.66	1,669.22	2,432.78	14,354.46

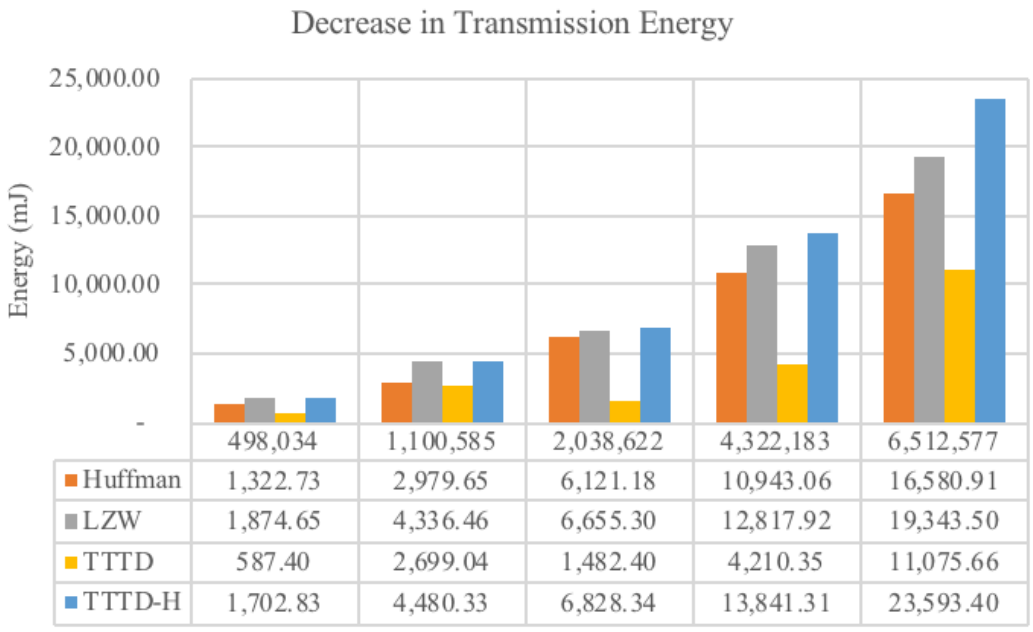
Figure 16. Dataset 1 - Comparison of Energy Saving due to Compression

TTTD-H consumes more energy during compression as expected and since compresses more data, the energy saving is higher because the transmission cost is much higher than the power used during computation.



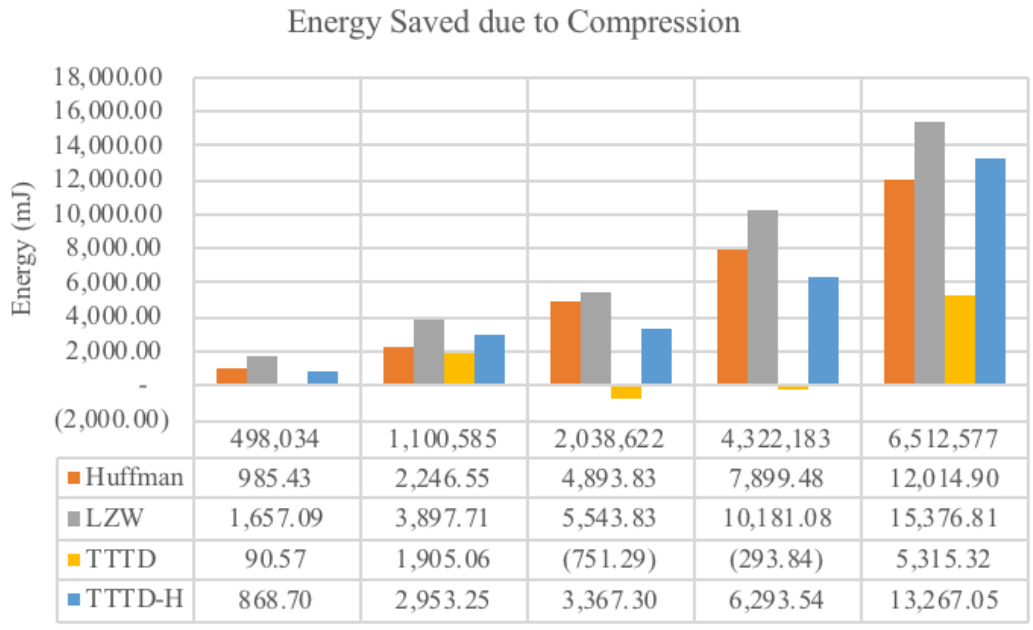
	498,034	1,100,585	2,038,622	4,322,183	6,512,577
Huffman	168.65	366.55	613.67	1,521.79	2,283.01
LZW	108.78	219.38	555.73	1,318.42	1,983.34
TTTD	248.41	396.99	1,116.85	2,252.10	2,880.17
TTTD-H	417.06	763.54	1,730.52	3,773.89	5,163.18

Figure 17. Dataset 2 - Comparison of Increase in Energy during Computation



	498,034	1,100,585	2,038,622	4,322,183	6,512,577
Huffman	1,322.73	2,979.65	6,121.18	10,943.06	16,580.91
LZW	1,874.65	4,336.46	6,655.30	12,817.92	19,343.50
TTTD	587.40	2,699.04	1,482.40	4,210.35	11,075.66
TTTD-H	1,702.83	4,480.33	6,828.34	13,841.31	23,593.40

Figure 18. Dataset 2 - Comparison of Decrease in Energy during Transmission



	498,034	1,100,585	2,038,622	4,322,183	6,512,577
Huffman	985.43	2,246.55	4,893.83	7,899.48	12,014.90
LZW	1,657.09	3,897.71	5,543.83	10,181.08	15,376.81
TTTD	90.57	1,905.06	(751.29)	(293.84)	5,315.32
TTTD-H	868.70	2,953.25	3,367.30	6,293.54	13,267.05

Figure 19. Dataset 2 - Comparison of Energy Saving due to Compression

7. CONCLUSION

7.1 Summary

In this paper, we reviewed the concept of wearable and implantable BSN and their lossless compression techniques, including RLE, LZW and Huffman. We also reviewed a data chunking algorithm used in many deduplication systems named TTTD. Experiments showed that Huffman is a stable algorithm, LZW had better compression ratio but takes more time. TTTD had lower compression ratio than LZW but was much faster. After this, we ran experiments by combining algorithms, and found that the sequence *TTTD -> LZW -> Huffman* performed better than others in terms for Compression Ratio and the sequence *TTTD -> Huffman -> LZW* performed better in terms of Compression Time. Finally, we proposed *TTTD -> Huffman (TTTD-H)* algorithm which performed significantly better than all the individual algorithms, with a slight increase in compression time. The result on improved compression factor implied a substantial reduction on sensor data to be transmitted; and therefore, a sizable saving on transmission energy.

7.2 Future Work

From LifeShirt (intelligent medical garment), STARPATCH (wireless cardiac monitoring) and IntelliDrug (implantable drug dispensing) reviewed by D Konstantas [24] in 2007, to implantable chips the size of a grain of rice in 2017, the last decade has seen an explosion of new sensors enter the market. Processes to analyze and store data analysis need to adapt to the growing demand and this area that needs more research. Since wireless data transmission uses most of the power in the sensor, there is a need to

have new methods to reduce the power consumption. That could be achieved by reducing the amount of data sent across by compression as discussed in the paper, reducing the amount of data sent across by sampling or by doing local processing. These efforts, especially 'local processing' would need sensors to have bigger memories. Sensors with more computing capability is another hot topic of research. The algorithm proposed in the paper can further be enhanced in multiple ways. Data from TTTD algorithm can be stored in a compressed form after a chunk is found. The chunk key lookup can be done using BloomFilter to make the lookup faster.

Another area of future research and development could be to store the data from sensors in the cloud after running through a data deduplication algorithm like TTTD. TTTD-S [26] which is an improvement over TTTD would be better suited to dedupe large amount of sensor data before storing them in the cloud. Storage in the cloud would help in getting more insights from historical data. Processing data using TTTD can also be parallelized. That means a large amount of data from multiple sensors can be deduped and stored by using Map Reduce frameworks like Hadoop. In summary, data compression and analyses of Wearable and Implantable BSN data is an important and growing field of research.

8. REFERENCES

[1] Dimitrov, D. V. (2016). Medical internet of things and big data in healthcare. Healthcare informatics research, 22(3), 156-163.

[2] Darwish, A., & Hassanien, A. E. (2011). Wearable and implantable wireless sensor network solutions for healthcare monitoring. Sensors, 11(6), 5561-5595.

[3] Lai, X., Liu, Q., Wei, X., Wang, W., Zhou, G., & Han, G. (2013). A survey of body sensor networks. Sensors, 13(5), 5406-5447.

[4] Song, K. T., & Wang, Y. Q. (2005, November). Remote activity monitoring of the elderly using a two-axis accelerometer. In Proceedings of the CACS Automatic Control Conference (pp. 18-19).

[5] Chen, Y., & Duan, H. (2006, January). A QRS complex detection algorithm based on mathematical morphology and envelope. In Engineering in Medicine and Biology Society, 2005. IEEE-EMBS 2005. 27th Annual International Conference of the (pp. 4654-4657). IEEE.

[6] Dong, L., Wu, J., Bao, X., & Xiao, W. (2006, June). Extraction of gait features using a wireless body sensor network (BSN). In ITS Telecommunications Proceedings, 2006 6th International Conference on (pp. 987-991). IEEE.

[7] Avci, A., Bosch, S., Marin-Perianu, M., Marin-Perianu, R., & Havinga, P. (2010, February). Activity recognition using inertial sensing for healthcare, wellbeing and sports applications: A survey. In Architecture of computing systems (ARCS), 2010 23rd international conference on (pp. 1-10). VDE.

[8] Peng, H., Long, F., & Ding, C. (2005). Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. IEEE Transactions on pattern analysis and machine intelligence, 27(8), 1226-1238.

[9] Krishnan, N. C., Juillard, C., Colbry, D., & Panchanathan, S. (2009). Recognition of hand movements using wearable accelerometers. Journal of Ambient Intelligence and Smart Environments, 1(2), 143-155.

[10] Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., & Cayirci, E. (2002). Wireless sensor networks: a survey. Computer networks, 38(4), 393-422.

[11] Sadler, C. M., & Martonosi, M. (2006, October). Data compression algorithms for energy-constrained devices in delay tolerant networks. In Proceedings of the 4th

international conference on Embedded networked sensor systems (pp. 265-278). ACM.

[12] Yoon, I., Kim, H., & Noh, D. K. (2017). Adaptive Data Aggregation and Compression to Improve Energy Utilization in Solar-Powered Wireless Sensor Networks. Sensors, 17(6), 1226.

[13] Wu, C. H., & Tseng, Y. C. (2011). Data compression by temporal and spatial correlations in a body-area sensor network: A case study in Pilates motion recognition. IEEE Transactions on Mobile Computing, 10(10), 1459-1472.

[14] Hu, F., Li, S., Xue, T., & Li, G. (2012). Design and analysis of low-power body area networks based on biomedical signals. International Journal of Electronics, 99(6), 811-822.

[15] Charbiwala, Z., Karkare, V., Gibson, S., Markovic, D., & Srivastava, M. B. (2011, May). Compressive sensing of neural action potentials using a learned union

of supports. In Body Sensor Networks (BSN), 2011 International Conference on (pp. 53-58). IEEE.

[16] Manikandan, M. S., & Dandapat, S. (2005, December). ECG signal compression using discrete sinc interpolation. In Intelligent Sensing and Information Processing, 2005. ICISIP 2005. Third International Conference on (pp. 14-19). IEEE.

[17] Tiwari, B., & Kumar, A. (2012, September). Aggregated Deflate-RLE compression technique for body sensor network. In Software Engineering (CONSEG), 2012 CSI Sixth International Conference on (pp. 1-6). IEEE.

[18] Kriegl, A. (2003, July). 3.3.1 Run-length encoding. Retrieved March 08, 2018, from http://www.mat.univie.ac.at/~kriegl/Skripten/CG/node44.html.

[19] Rosettacode.org. (2018, March). Huffman coding. Retrieved March 10, 2018, from https://rosettacode.org/wiki/Huffman_coding.

[20] Saikia, A. R. (n.d.). LZW (Lempel–Ziv–Welch) Compression technique. Retrieved March 10, 2018, from https://www.geeksforgeeks.org/lzw-lempel-ziv-welch-compression-technique.

[21] Muthitacharoen, A., Chen, B., & Mazieres, D. (2001, October). A low-bandwidth network file system. In ACM SIGOPS Operating Systems Review (Vol. 35, No. 5, pp. 174-187). ACM.

[22] Eshghi, K., & Tang, H. K. (2005). A framework for analyzing and improving content-based chunking algorithms. Hewlett-Packard Labs Technical Report TR, 30(2005).

[23] Ismail, S. (2018). Health datasets from Fitbit, Retrieved Feb 2018 from https://github.com/health-hacks/datasets/tree/master/fitbit.

[24] Konstantas, D. (2007). An overview of wearable and implantable medical sensors. Yearbook of medical informatics, 7(1), 66-69.

[25] Rabin, M. O. (1981). Fingerprinting by random polynomials. Cambridge, MA: Center for Research in Computing Techn., Aiken Computation Laboratory, Univ.

[26] Moh, T., & Chang, B. (2010). A running time improvement for the two thresholds two divisors algorithm. Proceedings of the 48th Annual Southeast Regional Conference on - ACM SE '10. doi:10.1145/1900008.1900101.

[27] Bo, C., Li, Z. F., & Can, W. (2013). Research on Chunking Algorithms of Data De-duplication. Proceedings of the 2012 International Conference on

Communication, Electronics and Automation Engineering Advances in Intelligent Systems and Computing, 1019-1025.doi:10.1007/978-3-642-31698-2_144.

[28] UCI Machine Learning Repository: Heterogeneity Activity Recognition Data Set. (2015, October). Retrieved Feb 2018 from https://archive.ics.uci.edu/ml/datasets/Heterogeneity Activity Recognition

[29] Shin, H. D., Ahn, S. W., Song, T. H., & Baeg, S. H. (2009). Analysis of Low Power Sensor Node Using Data Compression. IFAC Proceedings Volumes, 42(3), 34-39.

[30] Barr, K. C., & Asanović, K. (2006). Energy-aware lossless data compression. ACM Transactions on Computer Systems (TOCS), 24(3), 250-291.