A Project Report

On

An Analysis of Power Dissipation of Compression Techniques per Unit Computation and Unit Transmission in Cluster Heads of Wireless Sensor Networks

BY

ROHAN RUSSEL NEDUNGADI, SUDHEENDRA RAGHAV NEELA

17XJ1A544, 17XJ1A0555

Under the supervision of

Dr. RAMA MURTHY GARIMELLA

SUBMITTED IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS OF

SE 308: TEAM PROJECT

MAHINDRA ECOLE CENTRALE

COLLEGE OF ENGINEERING

HYDERABAD

(May 2020)

ACKNOWLEDGMENTS

We would like to sincerely thank Dr Rama Murthy Garimella, our faculty mentor, without which this project would not have been possible. He lent us a collection of his 30 paged personal notes, a very important thesis report and most important of all, his invaluable guidance and ideas. We would also like to express our gratitude to all the faculty, management and administration of Mahindra Ecole Centrale, Hyderabad for giving us this opportunity.

Mahindra Ecole Centrale

Hyderabad

Certificate

This is to certify that the project report entitled "An Analysis of Power Dissipation of Compression Techniques per Unit Computation and Unit Transmission in Cluster Heads of Wireless Sensor Networks" submitted by Mr. ROHAN RUSSEL NEDUNGADI (HT no. 17XJ1A0544) and Mr. SUDHEENDRA RAGHAV NEELA (HT No. 17XJ1A555) in partial fulfillment of the requirements of the course SE 308, Team Project, embodies the work done by them under my supervision and guidance.

(DR RAMA MURTHY GARIMELLA)

Supervisor

Mahindra Ecole Centrale, Hyderabad.

Date: 3rd May 2020

ABSTRACT

Wireless Sensor Networks have been quite popular for the constant monitoring of the physical conditions of the environment. Wireless Sensor Networks usually consist of multiple nodes which each have a sensor spread across a geographical area. Each node normally belongs to a cluster which it reports its data to when required. The cluster head keeps passing the data along from node to node until it reaches the central / head node. When passing data from cluster head to cluster head, the amount of data passed with each subsequent hop may increase with every hop due to the accumulation of multiple and thus, the power dissipation due to the transmission of the data increases with every hop thereby leading to uneven overall power dissipation amongst cluster heads.

Therefore, it is imperative to minimize the amount of data being passed with every hop instead of aggregating, accumulating and appending all the data together to pass it forward. A few ideas have been considered during the making of this report namely caching and the best spread of clusters using graph theory. However, this report primarily focuses on another method of minimizing data through compression. Two popular compressions techniques, Gunzip and Zip, have been analyzed primitively considering power dissipation due to computation and power dissipation due to transmission over a large number of files which consist of random and structured data.

CONTENTS

1. Introduction

A Wireless Sensor Network refers to a group of spatially dispersed and dedicated sensors for monitoring and recording the physical conditions of the environment and organizing the collected data at a central location. Wireless Sensor Networks are often used to measure real-time environmental conditions such as temperature, humidity, sound, light, concentration of a pollutant in different media and movement. Nodes and Cluster Heads of a Wireless Sensor Network are often powered by batteries and hence, it is of the utmost importance to save as much as power as possible. Two sources of power dissipation in a Wireless Sensor Network that we attempt to look at in this report are processing and transmission.

Power dissipated by processing refers to the power dissipated when any computation is done on a processor. Though the computation done on a processor may not be the same for every unit of computation because of the different types of computation and internal optimizations of the processor architecture, we assume an average value (α) of power dissipation per unit computation. The other source of power dissipation that we consider in this report is the power dissipated due to transmission which includes actively listening for messages and converting a message from one medium to another. As is the same with the power dissipation due to unit computation, we assume and average value (β) of power dissipation per unit transmission.

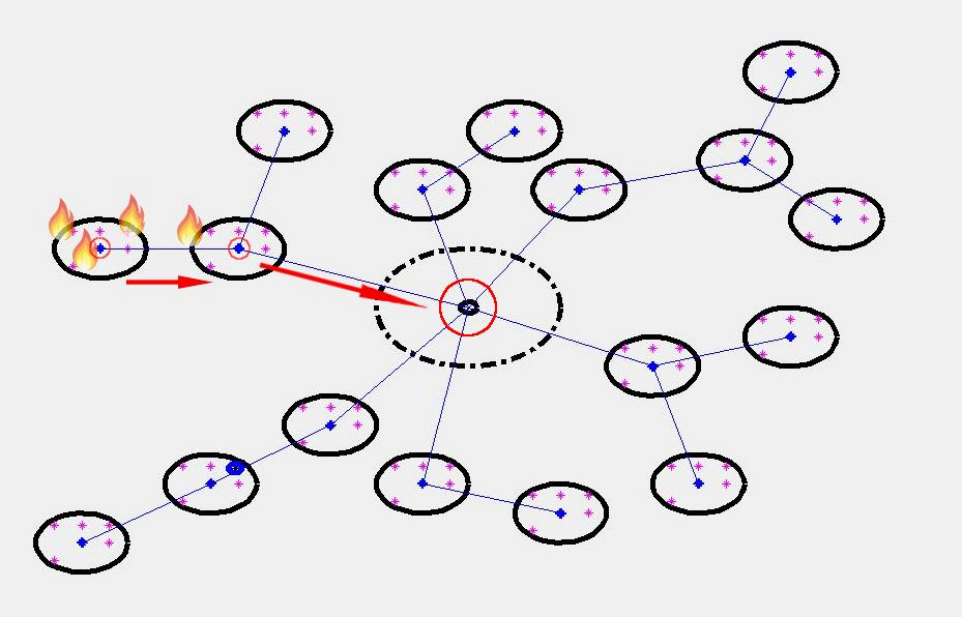


Figure 1. Wireless Sensor Network meant for detecting a Fire

Figure 1 is an illustration of a Wireless Sensor Network consisting of multiple nodes (indicated by the red dots) spread over a geographical area meant for detecting a

fire. Each of these nodes, part of one cluster (an encompassing black circle), reports the information it has collected to the nearest cluster head (indicated by blue dots). Each cluster head passes the information it receives forward to either the head node or another cluster head which then repeats the same process again. The head node (indicated by a solid blue circle and highlighted by a red circle in the center of Figure 1) aggregates and accumulates the received data to take the decision of notifying whether values have crossed their configured ranges.

To solidify this with an example, Figure 1 shows a fire happening in two of the clusters to the center-left of the image. Nodes in the two clusters use their temperature sensors to detect the outbreak of a fire and send a message to their cluster head. The cluster head then forwards this message until it reaches the head node which afterwards performs the necessary actions.

This example is simple and is said to be event-driven i.e., the nodes only alert the cluster head only when they detect a fire. As such, the length of the message is quite small and can be less than one megabyte. Therefore, the power consumed when transmitting the message forward from one cluster head to another cluster head is quite minimal. Moreover, very less power is dissipated due to computation as there is little to no computation being done in the cluster heads.

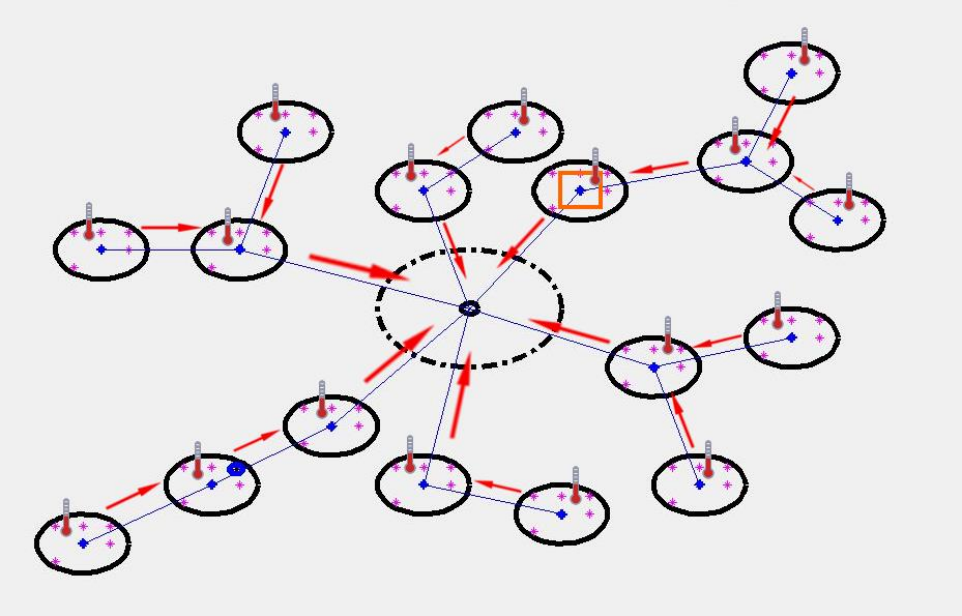


Figure 2. Wireless Sensor Network meant for accumulating temperatures

Figure 2 is an illustration of the same Wireless Sensor Network in Figure 1 with a fundamental difference that the head node collects all the temperature readings from the nodes every hour instead of listening for one or more messages. In this figure, each cluster comprises of five nodes which send their temperature readings to the cluster head. Each cluster head passes the readings onwards to another cluster head or to the

head node. The major issue with this type of Wireless Sensor Network is the size of the message being passed from cluster head to cluster head increases with every forward due to the receiving cluster head appending its own nodes' temperature reading.

In Figure 2, the cluster head encompassed by the orange box (center-top) receives a message which has fifteen readings from the previous cluster head. It then adds five readings from its own nodes before sending a message comprising of twenty readings to the head node. Although a message comprising of twenty readings may seem trivial at this instant, this scenario can be extrapolated to a real world Wireless Sensor Network comprising of hundreds or thousands of nodes distributed across multiple clusters in a very skewed spread; the power dissipated due to transmission naturally increases from one cluster head to another thereby leading to an uneven power dissipation across cluster heads even amongst the same level. On top of the power dissipated due to transmission along every new cluster head each message encounters, there will be a slight increase in power dissipation due to computation because of the appending of the current cluster's nodes' temperature readings.

This leads us to formulate the minute yet significant solution of compression. In this report, we attempt to analyse two popular compression techniques, gunzip[6] and zip[7], and compare the power dissipated due to computation by compression and transmission of the compressed file.

2. Problem Definition

With a basic introduction we now know that the major problem with Wireless Sensor Networks is power consumption. Wireless Sensor Networks are nothing but a sensor to record certain forms of data, for e.g. thermal-temperature, visual-camera which are wirelessly connected to each other to form a network over a physical location. Being wireless helps us to position these sensors in various areas that may be void of any power source, e.g. thermal sensors in the forests. Hence these sensors come with a small power source, usually a regular battery, within themselves.

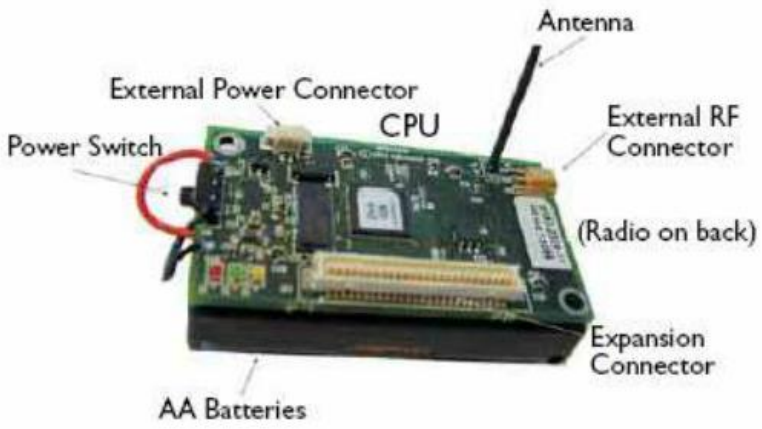


Figure 3: Modern day sensor node with various parts shown. As you can see, they are powered by regular double AA batteries.

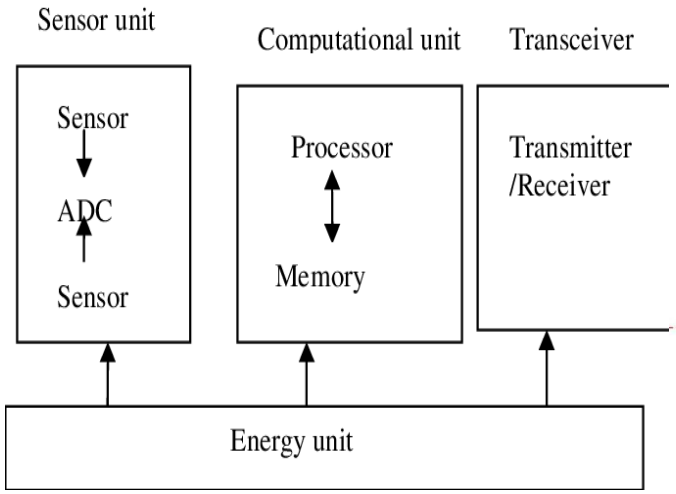
Now in most cases the sensors which are placed are of the perishable kind i.e. once run out of battery the device isn't retrieved. But there are many cases when the device is to be reused i.e. non-perishable, where once the device runs out of battery the operator must physically go and change the battery to the node which may be a very cumbersome task when dealing with frequent changes in a large number of nodes. In either case we can identify the problem to be the power dissipation. Hence many scholars have driven research in the area to find the most effective means of power consumption in wireless sensor networks.

While this area consists of many fields where research is possible, such as software optimisation, hardware optimisation, topology and distribution, or protocols used the author will focus on analysis in the software section.

Coming to analyse the software section, from past observations we know that transmission generally takes the highest amount of power amongst all functionality within the node. Many papers have tried to reduce the transmission to save power, but the drawback is it need not always reduce the power consumption. As shown in figure 4a we can see that the architecture of a node includes possibly to put a processing unit

along with local memory onto the node, which extracts power from the common energy source. Figure 4b shows a raspberry pi implementation of the architecture.

In an example where the transferable data belongs to a finite set and the size of the data is large, we can pre-process the data into a smaller value such as an index. This works perfectly fine in most cases, where table lookup and pre-processing can save on transmission cost. But what in the case the data does not belong to a finite set, e.g. a visual sensing node needs to send images. In this case the only way to reduce the transmission costs is by compressing the data. The other issue that arises here is that the power consumption for complete compression is almost equivalent to that of transmission in many cases. A correct choice of compression algorithm could make all the difference where the total cost could be reduced depending on the nature of the data



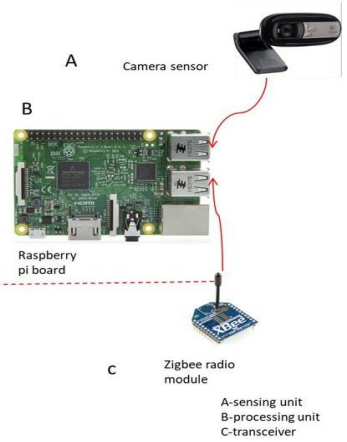


Figure 4a *Figure 4b*

Figure 4: The sensors as shown in the figures 2a and 2b usually include a microcontroller board which has the computation unit along with a network unit and the sensor. Figure 4a shows the architecture of a general Wireless Sensor Network node while figure 4b shows a modern-day implementation of the architecture.

This paper will focus primarily on an analysis on this trade-off given by the equation below and present an answer to what is the optimal way.

Energy Consumption per node = Computation energy + Transmission energy

New Energy Consumption per node = α x Computation energy + β x Transmision energy

α, β are the multiplicity factors to which the respective energy is increased/decreased.

Total energy reduced to in % = new energy x 100/ old energy

3. Background Related Work

3.1. Balancing computational and transmission power consumption in wireless image sensor networks [1]

This is the paper that paved the way for our research analysis on the subject. It has done extensive work in the area with great tested results. The paper works on understanding the consumption of power in Wireless Sensor Networks and trying to save it by using various compression algorithms and testing them.

The paper clearly displays the actual values faced when conducting the experiment showing the amount of power consumed by the different working parts of the node. It gives a brief insight in what kind of values we are dealing with and brings out the significance of the matter.

The paper has considered a visual sensor node for its reference on all experiments and its resulting values. The data being transferred are images which ask for large power consumption. The paper has analysed the different factors and features of an image that must be considered important and required as a sink node preference.

It goes on to explain which features are required when and how to choose the certain values that are required to achieve a defined result. It has extensively worked on actual data and algorithms to show various influential variable values. It provides them with a great analysis to compare and choose the appropriate factors we may be looking for.

Based on the required factor it presents options of various existing compression algorithms such as lossy and lossless algorithms for images. For the given experiment the authors have considered JPEG images and gone ahead with the respective algorithms such as JPEG2000, Sub Sampling (SS), Discrete Cosine Transform (DCT), Set Partitioning in Hierarchical Trees (SPIHT) and JPEG;

It then shows a well put comparison of the various algorithms along with varied values of different images features and factors such as quality index value of the decompressed image.

The issue with the experimenting results of this paper are that the results are specific to a predefined narrow domain and not proven to work in general. The paper also only works to discuss the factors involved in visual sensor nodes and not all Wireless Sensor Networks in general. This affects the experiment as the algorithms picked for compression also are specific only to the JPEG format. We are planning to look at a more generalised result.

3.2. A Survey on Data Compression in Wireless Sensor Networks [2]

This paper is a very advanced research into the area and follows the same motives as ours. It works on analysing and explaining transmission power consumption and the need to reduce it.

The paper works on very novel ideas and algorithms cited by other papers that are efficiently designed for wireless sensor networks based on the various factors that apply to Wireless Sensor Networks. One of the many factors being that Wireless Sensor Networks have a limitation on resources i.e. limited bandwidth, limited energy, limited processing capabilities and limited memory.

The author used the following algorithms for testing purposes: bzip2 (BWT algorithm), compress (LZE algorithm), LZO (LZ77), PPMd (PPM), and zlib (LZ77). These algorithms were combined with the novel cited compression techniques as listed below:

Coding by Ordering, Pipelined In-Network Compression, JPEG200, Low-complexity video compression, Distributed Compression.

Although the resulting analysis of this experiment was very promising, it did not fit our study. The paper focused on providing an analysis with a combination of techniques and displaying the respective effective reduction in consumption. In our paper we want to focus more on the analysis of the transmission versus compression energy consumption and significance of the role played by the data.

3.3. Design of Modified Adaptive Huffman Data Compression Algorithm for Wireless Sensor Network [3]

The paper's aim focuses on dealing with energy losses in transmission and sets out to find an optimal solution to it. It focuses on innovative techniques of using encoding to compress the required data to be sent. The proposed algorithms under study are the variants of Huffman encoding - Simple Huffman Encoding, Adaptive Huffman Encoding.

The author analyses both the algorithms and presents the advantages and forth comings of them both. While simple Huffman has the speed in look up , it is limited by the fixed domain of values that it can process and adaptive Huffman encounters this issues by being able to flexibly add new data to the encoding as when it encounters one , but the issue being that it processes a larger deal , i.e. it has to re shape the tree based on frequency and other factors.

The modified adaptive Huffman Encoding algorithm is proposed by the authors of this paper for the solution to the shortcomings of both these algorithms. It shows the

effects of the different algorithms and how the efficiency of power consumption increases with each algorithm.

While the algorithm shows us the effective use of Huffman encoding, it also demonstrates the drawbacks faced while using encoding hence why we shall not consider encoding as general a technique as compression for our study.

3.4. A Simple Algorithm for Data Compression in Wireless Sensor Networks [4]

This is a simple to the point paper that does a research on finding an effective compression algorithm for wireless sensor networks. Though it does not focus on the aim of our paper it gives us insights on the working of some of the existing techniques and provides us with a solution to overcome some of them.

It has done research and used techniques such as Huffman encoding and displayed its results to show it isn't the best result that could be offered. The findings show that the encoding algorithms have brought about 67-68 % compression ratio which is showing a decent reduction in size but could be done much better.

It later works on some algorithms such as S-LZW, gzip and bzip2 which are existing algorithms in the domain and shows them to provide a much better result of 30-40% compression ratio for various data values.

It shows a great comparison as to how compression techniques can better the results. But the shortcomings are that it hasn't considered the power consumption and weighed it against the transmission power saved. The authors haven't provided a detailed analysis on how when choosing different techniques, the overall energy is saved.

3.5. Low Complex and Power Efficient Text Compressor for Cellular and Sensor Networks [5]

The approach and idea behind the need for this paper was very interesting. It works on finding a low complex power efficient text compressor which is used to compress text messages in wireless mobile phones while can also be used for Wireless Sensor Networks. The author has displayed a very strong point in the end showing that his compression results led to much shorted messages being sent and hence making the charges made by cellular broadband companies unnecessarily high.

The paper works to show the trade-offs between compression and transmission energy consumption and in this setup has proven to show that compressing and decompressing followed by reduced transmission requires much less energy than simply transmitting.

The paper has very well visualised most of its results and comparisons of trade-offs for which they were aiming for. After extensive work and testing on actual nokia phones they came up with a working java app that can compress text messages and hence allow the user to be able to send more words and length for the same cost provided the app is installed on the receiver side as well.

While the paper clearly works on proving the analysis and savings of the compression techniques over the simple transmission, it's not a general study of all the cases which this paper is trying to do. It does provide some beautiful insights on the topic which fuels some of our arguments in this paper but does not work on giving a complete analysis on the subject.

4. Implementation

Our implementation chapter remains rather small as only two programs were necessary for the generation of our results. We attempted to analyse two popular compression techniques, gunzip[6] and zip[7], along with a bundling technique that does not compress, tarball[8], over two sets of data, structured and unstructured. We started out by gathering two files of which one was unstructured, random and newly created (sourced from /dev/urandom[9]) while the other was structured, ordered and pre-existing (a 2 Megabyte JSON file). This was to simulate two possible scenarios where data could be similar or the data could not. After sourcing these two files, each file was iteratively broken into larger pieces where they were compressed while being timed. The custom shell script that was used for the program can be found at [10]. For the purpose of this report, a high-level overview of the implementation is provided below in Figure 5.

```
function capture(integer filesize, string filesource):

    create `newfile` from filesource of filesize bytes

    begin `time_tarball`
    generate `tarball` from `newfile`
    end `time_tarball`

    store `time_tarball`

    begin `time_gunzip`
    generate `gunzip` from `newfile`
    end `time_gunzip`

    store `time_gunzip`

    begin `time_zip`
    generate `zip` from `newfile`
    end `time_zip`

    store `time_zip`

    store compressed files sizes

function main():
    for filesize in (0, 2 Megabytes, 2500 bytes):
        capture(filesize, `/dev/urandom`)
        capture(filesize, `file.json`)
```

Figure 5. A high-level implementation of the program used to capture raw data

This simple shell script creates a new file of increasing size using the dd[11] command and consequently records the time taken to bundle using tar and compress using gunzip and zip. The time measured is recorded using the time[12] command and the total time taken is the summation of the user and sys parameters of the output. The

user and sys parameters summed represent the total amount of CPU execution time the command took. The script then records the sizes of the compressed files.

For creating the graphs in Chapter 5. Results, we used python and matplotlib to render our captured data. We had four dimensions to plot namely, File Size, α, β and New Energy Consumption per Node, henceforth referred to as output. Our method of graphing these onwards was to map output to the colour domain defined by matplotlib's cmap copper. However, on plotting these we realized that the output numbers were very close and hence a thorough analysis could not be done just by using graphs. Hence, for the sake of thorough understanding, we have written the actual values for varying File Size, α, β and output.

Output is calculated absolutely (not relatively) according to the formula as mentioned above in Chapter 2. Problem Definition:

$$\text{Output} = \alpha \times (T \times (|FS_{original} - FS_{new}| + FS_{original})) + \beta \times (FS_{new})$$

[Formula 1]

Where α is the power dissipated per unit computation, β is the power dissipated per unit transmission, $FS_{original}$ is the original File Size, FS_{new} is the new File Size and T is the time taken to compress $FS_{original}$ to FS_{new}.

The Absolute value is required sometimes when the tarball (bundling) technique adds more information when bundling files together and FS_{new} is larger than $FS_{original}$ so as to make the power dissipated by computation not negative.

5. Results

We first show the graphs that our program generated. Figure 6, 7, 8 are the graphs of File Size vs. α vs. β vs. output (colour) for Gunzip (Compression), Tarball (Bundling) and Zip (Compression) on an unstructured file respectively.

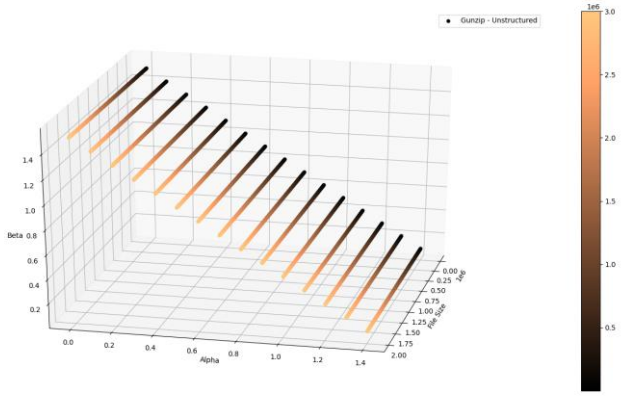


Figure 6. File Size vs. α vs. β vs. output (colour) using Gunzip (Compression) on an unstructured file

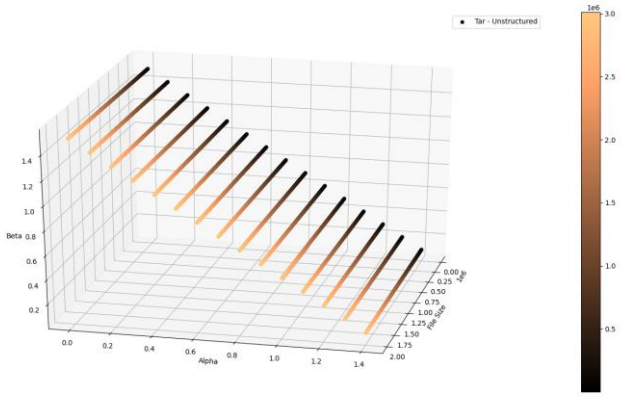


Figure 7. File Size vs. α vs. β vs. output (colour) using Tarball(Bundling) on an unstructured file

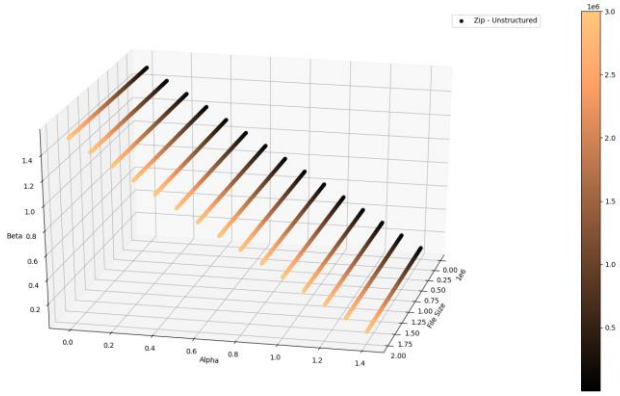


Figure 8. File Size vs. α vs. β vs. output (colour) using Zip (Compression) on an unstructured file

Figure 9, 10, 11 are the graphs of File Size vs. α vs. β vs. output (colour) for Gunzip (Compression), Tarball (Bundling) and Zip (Compression) on a structured file respectively.

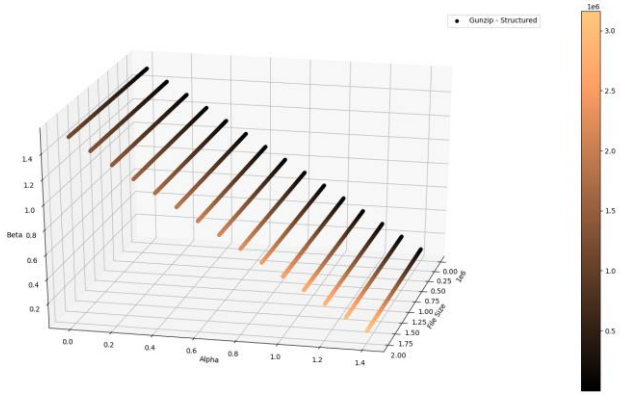


Figure 9. File Size vs. α vs. β vs. output (colour) using Gunzip (Compression) on a structured file

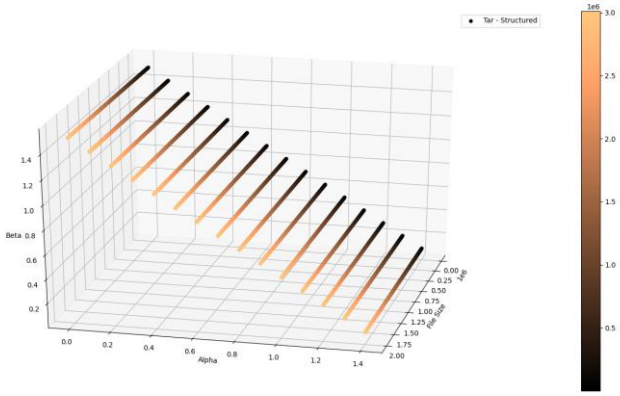


Figure 10. File Size vs. α vs. β vs. output (colour) using Tarball (Bundling) on a structured file

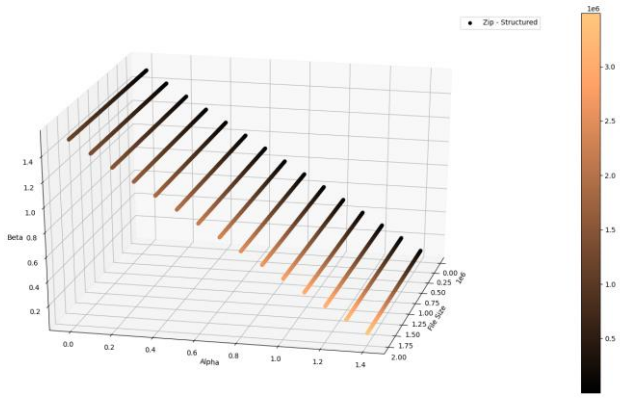


Figure 11. File Size vs. α vs. β vs. output (colour) using Zip (Compression) on a structured file

All the six graphs (Figure 6 – 10) may look the same as the images are small; however, they are not. On closer inspection, we noticed that the output ranges for similar file sizes, similar α and similar β result in similar outputs for Gunzip and Zip as they compress to very similar sizes. Hence, we have provided the exact values for two file sizes, 500 Kilobytes and 1.75 Megabytes, with 3 pairs of α and β, (0.3, 1.2), (0.7, 0.8) and (1.2, 0.3), so as to provide a definite visible difference in the outputs. The reasoning behind the choosing of the pairs of α and β are as follows:

- (0.3, 1.2) - α = 0.3 represents a very low power processor and β = 1.2 represents a very power demanding (or high range) transmitter.
- (0.7, 0.8) - α = 0.7 and β = 0.8 represent a well-rounded power demanding processor and transmitter.

- (1.2, 0.3) - α = 1.2 represents a very powerful processor and β = 0.3 represents a very low power demanding (or low range) transmitter.

Table 12. Gunzip (Compression) Structured

$FS_{original}$	α	β	FS_{new}	T	Output
500Kb	0.3	1.2	197770	0.045	248154.105
500Kb	0.7	0.8	197770	0.045	183766.245
500Kb	1.2	0.3	197770	0.045	103131.42
1.75Mb	0.3	1.2	701034	0.15	967194.27
1.75Mb	0.7	0.8	701034	0.15	854718.63
1.75Mb	1.2	0.3	701034	0.15	714124.08

Table 13. Tarball (Bundling) Structured

$FS_{original}$	α	β	FS_{new}	T	Output
500Kb	0.3	1.2	501760	0.002	602413.056
500Kb	0.7	0.8	501760	0.002	402110.464
500Kb	1.2	0.3	501760	0.002	151732.224
1.75Mb	0.3	1.2	1761280	0.003	2115121.962
1.75Mb	0.7	0.8	1761280	0.003	1412722.688
1.75Mb	1.2	0.3	1761280	0.003	534724.608

Table 14. Zip (Compression) Structured

$FS_{original}$	α	β	FS_{new}	T	Output
500Kb	0.3	1.2	197834	0.096	260524.1808
500Kb	0.7	0.8	197834	0.096	212221.7552
500Kb	1.2	0.3	197834	0.096	151843.7232
1.75Mb	0.3	1.2	701016	0.322	1111601.0544
1.75Mb	0.7	0.8	701016	0.322	1191703.7936
1.75Mb	1.2	0.3	701016	0.322	1291832.2176

Table 15. Gunzip (Compression) Unstructured

$FS_{original}$	α	β	FS_{new}	T	Output
500Kb	0.3	1.2	500286	0.032	605145.9456
500Kb	0.7	0.8	500286	0.032	411435.2064
500Kb	1.2	0.3	500286	0.032	169296.7824
1.75Mb	0.3	1.2	1750521	0.068	2136335.8284
1.75Mb	0.7	0.8	1750521	0.068	1483741.5996
1.75Mb	1.2	0.3	1750521	0.068	667998.8136

Table 16. Tarball (Bundling) Unstructured

$FS_{original}$	α	β	FS_{new}	T	Output
500Kb	0.3	1.2	501760	0.002	602413.056
500Kb	0.7	0.8	501760	0.002	402110.464
500Kb	1.2	0.3	501760	0.002	151732.224
1.75Mb	0.3	1.2	1761280	0.003	2115121.962
1.75Mb	0.7	0.8	1761280	0.003	1412722.688
1.75Mb	1.2	0.3	1761280	0.003	534724.608

Table 17. Zip (Compression) Unstructured

$FS_{original}$	α	β	FS_{new}	T	Output
500Kb	0.3	1.2	500256	0.042	606610.4256
500Kb	0.7	0.8	500256	0.042	414912.3264
500Kb	1.2	0.3	500256	0.042	175289.7024
1.75Mb	0.3	1.2	1750446	0.096	2150948.0448
1.75Mb	0.7	0.8	1750446	0.096	1517986.7712
1.75Mb	1.2	0.3	1750446	0.096	726785.1792

6. Conclusion

It can be calculated from the primitive formula that the total power dissipated is:

$$\text{Output} = \alpha \times (T \times (|FS_{original} - FS_{new}| + FS_{original})) + \beta \times (FS_{new})$$

[Formula 1]

Where α is the power dissipated per unit computation, β is the power dissipated per unit transmission, $FS_{original}$ is the original File Size, FS_{new} is the new File Size and T is the time taken to compress $FS_{original}$ to FS_{new}.

Using Formula 1, we calculated the absolute power dissipated when the compression techniques Gunzip and Zip were applied and the absolute power dissipated when the bundling technique Tarball was applied. In Tables 18 and 19 below, a final comparison of Gunzip and Zip is made with respect to Tarball.

Table 18. $Output_{Tarball}$ vs $Output_{Gunzip}$, $Output_{Zip}$ for structured data

Structured Data	Average $Output_{Tarball} \div Output_{Gunzip}$	Average `$Output_{Tarball} \div Output_{Zip}$
500Kb	1.923	1.733
1.75Mb	1.528	1.166

Table 19. $Output_{Tarball}$ vs $Output_{Gunzip}$, $Output_{Zip}$ for unstructured data

Unstructured Data	Average $Output_{Tarball} \div Output_{Gunzip}$	Average `$Output_{Tarball} \div Output_{Zip}$
500Kb	0.956	0.942
1.75Mb	0.914	0.882

Initial results show that the Gunzip compression method on average outperforms the Zip compression method for structured and repetitive data. The Gunzip compression method also outperforms the Zip method for unstructured and random data, however both perform worse than the original Tarball bundling itself. Future work can lead towards analysing more compression techniques, estimating the optimal values of α and β and analysing the current existing compression techniques over finer ranges of file sizes, α and β.

7. References

[1] Balancing computational and transmission power consumption in wireless image sensor networks -
L. Ferrigno1, S. Marano2, V. Paciello2, A. Pietrosanto2
VECIMS 2005 – IEEE International Conference on Virtual Environments, Human-Computer Interfaces, and Measurement Systems Giardini Naxos, Italy, 18-20 July 2005

[2] A Survey on Data Compression in Wireless Sensor Networks
Naoto Kimura and Shahram Latifi
Electrical and Computer Engineering, University of Nevada, Las Vegas
Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05) 0-7695-2315-3/05 $ 20.00 IEEE

[3] Design of Modified Adaptive Huffman Data Compression Algorithm for Wireless Sensor Network
C. Tharini and P. Vanaja Ranjan Department of Electrical and Electronics Engineering, College of Engineering, Guindy, Anna University, Chennai, India
Journal of Computer Science 5 (6): 466-470, 2009 ISSN 1549-3636 © 2009 Science Publications

[4] A Simple Algorithm for Data Compression in Wireless Sensor Networks
Francesco Marcelloni, Member, IEEE, and Massimo Vecchio, Member, IEEE
IEEE COMMUNICATIONS LETTERS, VOL. 12, NO. 6, JUNE 2008

[5] Low Complex and Power Efficient Text Compressor for Cellular and Sensor Networks
Frank H.P. Fitzek∗ Stephan Rein∗∗ Morten V. Pedersen∗ Gian Paolo Perrucci∗ Thorsten Schneider∗ Clemens Guhmann ¨ ∗∗ ∗Aalborg University, Denmark ∗∗Technical University of Berlin, Germany

[6] https://www.freebsd.org/cgi/man.cgi?query=gzip

[7] https://www.freebsd.org/cgi/man.cgi?query=zip

[8] https://www.freebsd.org/cgi/man.cgi?query=tar

[9] https://linux.die.net/man/4/urandom

[10] https://gist.github.com/IceCereal/b2813cbd842c102828b9a4396e3af866

[11] https://linux.die.net/man/1/dd

[12] https://linux.die.net/man/1/time