



Performance analysis of IoT networks in terrestrial environment utilizing LZW data compression technique

 Ankur Sisodia¹⁺
 Ajay Kumar
Yadav²

^{1,2}Banasthali Vidyapith, Rajasthan, India.

¹Email: ankur22887@gmail.com

²Email: ajay.iitdhn@gmail.com



(+ Corresponding author)

ABSTRACT

Article History

Received: 24 July 2023

Revised: 3 October 2023

Accepted: 31 October 2023

Published: 13 December 2023

Keywords

Constrained application protocol

Data reduction

End to end delay

IOT network

LZW data compression

Routing overhead

Squeeze compressor

Throughput.

The proliferation of Internet of Things (IoT) devices in terrestrial environments necessitates efficient data transmission and management protocols to optimize network performance. This research paper presents a comprehensive study on the implementation of IoT networks in a terrestrial setting, focusing on the integration of LZW (Lempel-Ziv-Welch) compression with three prominent IoT protocols: CoAP (Constrained Application Protocol), M2M (machine-to-machine), and MQTT (Message Queuing Telemetry Transport). We did a thorough test of these protocols using the NetSim simulator. We looked at how well they worked in terms of throughput, end-to-end delay, and routing overhead, all of which are important for IoT network efficiency. The point of our study is to find out what the pros and cons of these protocols are when they are combined with LZW compression in a terrestrial IoT setting. The results of our analysis reveal significant variations in the performance of CoAP, M2M, and MQTT when subjected to data compression. Throughput efficiency, which directly impacts data transmission rates, is scrutinized alongside end-to-end delay, a critical factor in ensuring timely data delivery. Additionally, we explore the implications of protocol choice on routing overhead, a crucial metric for network resource utilization. This research contributes to the ongoing efforts to optimize IoT networks in terrestrial settings, offering valuable guidance to network architects and developers seeking to strike the right balance between protocol selection and data compression for improved IoT performance.

Contribution/Originality: By implementing LZW compression in an IoT network, we can enhance its performance. This IoT network model has been deployed in a terrestrial environment, incorporating data compression techniques. The model has been implemented and simulated using a range of protocols. Consequently, it is suitable for deployment in various IoT-related applications.

1. INTRODUCTION

The IoT (Internet of Things) is intelligently connecting the world. It is described as a wireless connection of sensing and actuation internet-connected devices prepared to gather and transfer data backed by embedded systems or sensor systems. Sensors, processors, memory, transceivers, and batteries are the five essential components of IoT devices [1]. All of these devices are linked via gateway functionality, which may communicate with the sensor as well as provide computation and storage capability. This gateway might be situated on the internet or at the network edge.

One of the most significant difficulties with the IoT is the massive amount of data production. To overcome this drawback and hence avoid the transmission and retention of vast volumes of data, on-board prompt analysis and reduction methods would be used. In general, radio communication consumes the most power. As a result, lossless compression can dramatically reduce transmission electricity expense by decreasing the quantity to be transmitted and, as a result, extend device lifetime [2].

As a result, the energy usage and battery life of the equipment, as well as the memory space of the sent data, are immediately affected, allowing for significantly less data to be maintained. LZW (Lempel-Ziv-Welch) is a prominent and commonly used reduction technique with easy implementation, an increased compression ratio, and efficiency. LZW is utilized in a wide range of applications (including GIF image formats and UNIX operating system files) [3].

Because it is a lossless data compression algorithm, LZW is frequently employed both in image and text reduction. As a result, the ability to do template-matching searches on LZW-compressed information seems to be something worth investigating. A code phrase is a citation to a dynamic vocabulary in the LZW compression method. Because of the adaptability of LZW compression, various code words are assigned to a similar set of characters based on their position inside the inlet sequence [4].

In 1977, the initial Lempel-Ziv data compression method was developed, supplemented by an alternative path in 1978. In 1984, Terry Welch presented his enhancements to the 1978 algorithm. The technique is very straightforward, and it is widely used. LZW compression, in a sense, substitutes sequences of characters with single codes [5]. Arrays, concatenated lists, and trees can all be used to achieve LZW. The method will search for codes utilizing consecutive traversing while being performed by chained lists while generating the vocabulary tables. As a result, the reduction in time increases as it consumes lots of time to only search the codes [6]. To tackle this difficulty, the authors utilise chained lists to create LZW, and they add the new variables to each node to maintain track of how many times the codes have been used. In addition to saving time, the frequency codes are shifted and put at the front end of a linked list while creating a table. Moreover, the optimisation will also have a better impact by optimising the front-insert site and using appropriate statistical numbers.

In this research, the authors propose and execute Multi-PLZW, a ground-breaking multiple correlation-based approach that will find every occurrence of a design in LZW-compressed data. Principally, the technique works in the same way as the Aho-Corasick algorithms (AC) [7], also with the added benefit of being able to use LZW compressed data immediately. Multi-PLZW uses a design from the previous step to construct two data structures. First is a limited automaton that has been on the set's generalized suffix tree [6].

In the current state, several proposed reduction algorithms for IoT focus on higher compression, distortions, or energy usage at the first level and disregard classifier effectiveness at the third level. The present study examines the effects of downsizing on the process of categorization within the context of the IoT. It concentrates on time-series data compression and categorization [8].

This work's stated problem can be stated as follows: What effect does the deployment of graph data lossy compressors on IoT have on deep instructional approach classification performance?

Due to the severe resource limits of IoT nodes, a reduction strategy addressing IoT application features must consider metrics: i) the total energy saved during transmission must be greater than the amount of energy consumed during processing. (ii) Embedded processors on IoT networks use limited on-chip memories. (iii) Any application, sensor, or activity can benefit from the compression algorithm. (iv) suitable for applications that require near-real-time processing. (v) simple to set up and adapt to different nodes. (vi) capable of high compression ratios while maintaining the quality of the data. (vii) The capability of a single device to handle many sensor readings [9].

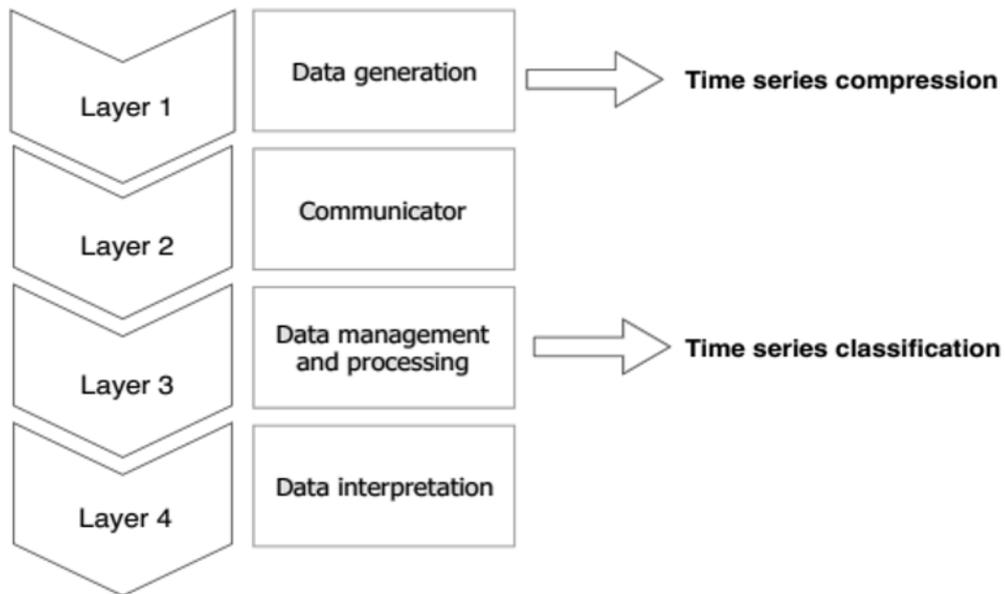


Figure 1. IoT architecture.

Following the pre-processing stage, there is a scanning stage that employs a series of functions to make use of the results that used in earlier phases. The scanning step employs two critical database systems: a linkage database and a graph of historical data [3]. Multi-PLZW could be used instantly on LZW-compressed information and is capable of reporting all incidents of features in a single cycle and in a constant period.

Time series move through many steps in an IoT Big Information infrastructure. Four levels can be found in an example of IoT Big Information infrastructure [10]. Time series data is generated through a multitude of IoT resources and thereafter collected and condensed prior to transmission at the initial stage. The second layer facilitates connectivity between devices and gateways by utilizing various communications technologies, as shown in Figure 1[5].

Many data compression methods in the IoT, with the newer technologies, concentrate on higher reduction, distortion level, and electricity usage at the first level while neglecting categorization findings inside the third step [11]. This article concentrates on series data reduction and categorization strategies in IoT networks, as well as the impact of reducing tasks on classifiers.

To address resource constraints, many dimension reduction strategies are offered in the research. An effective high-ratio LZW data reduction algorithm for univariate integrated time series data with much reduced delay and storage than provincial methods. In previous studies, the effect of dimension reduction on classification results was explored in relation to the topic presented in this article.

The duration was proportional to the combined size of the lossless compression and the specified patterns. The suggested technique outperforms all existing analogues in terms of time and computational power. It has an overall temporal complication of $(n + k + r)$, where n denotes the running time of a compact term, k denotes the frequency of a sequence, and r is the standardisation of a tendency in condensed text. It also has a computational cost of $\theta(k^2 + l + r)$, wherein l is indeed the form of the term dictionaries. One more advantage of our process is the ability to examine compressed data while unpacking it. Some rival algorithms decompress and then search to use the AC technique, which is particularly undesirable in internet contexts where file size is costly. We also present specifics and practical considerations for our method in this section. We also do theoretical and experimental evaluations of the proposed approach. The suggested algorithm surpasses previous strategies in terms of capacity and effectiveness, according to the results [12, 13].

The remainder of a task is structured as below: Section 1 also includes a list of related works. Section 2 depicts and analyses the suggested algorithm's efficiency. Section 3 conducts an experimental evaluation of the method's implementation. Finally, Section 4 brings the process to a conclusion.

2. PROPOSED SYSTEM

Compression sense is a digital signal processor approach that can successfully acquire and reconstruct a signal from fewer readings. Compression sense can collect and reflect fragmented signals at a frequency significantly lower than that used in the sampling theorem. Equation 1 outlines the fundamental measuring paradigm for spectrum sensing:

$$[B]_{M1} = [\rho]_{X,Y} [A]_{Y,1} \quad (1)$$

Wherein A is the initial patchy data with fixed length, B is indeed the compacted signal with duration X ($X \ll Y$), and $\rho \in RX \times Y$ seems to be a sensor matrix that allows signal restoration. Many signals have such low sparsity in time-domain representation, for instance. As a consequence, representation of these signals is scarce within the bandwidth response beneath a particular foundation. In such a case, A could be expressed as a singularly positioned integer, which is shown in Equation 2:

$$[A]_{X,1} = [\rho]_{Y,Y} [C]_{Y,1} \quad (2)$$

Wherein C is a bandwidth field description of an input signal A as well as the vectors, which translate directly from the spectral to the time domain. By combining Equations 1 and 2, the signal restitution enables C to be approximated utilizing, B as shown in Equation 3:

$$[B]_{X,1} = [\rho]_{X,Y} [\rho]_{Y,Y} [C]_{Y,1} \quad (3)$$

In this study, we compute ρ by performing the inverse Discrete Cosine Transform only on a column of the solution. For restoring the signal A, we have implemented an 11-norm minimization method, which is properly known as Basis Pursuit.

Because the data obtained in this work is assumed to be a time-series data of floating points, condensed B vector includes float point cloud data. In that situation, an additional efficient entropy coder is necessary to compress the resulting vector B.

2.1. Principles and Features of LZW

All singular values and associated values are included in the vocabulary at first. Then start compaction, take a format string, and compare that against the phrases in the dictionaries until it can't match anymore. Return the value for the correctly matched string. Then, as just a new string, combine that string with characters that led the matching to fail, and save this in the string databases with the appropriate code. The LZW phrase database is classified as a suffix database, as it encompasses all string prefixes inside its structure. Real-time dictionaries have been developed which means that it contains every string prefix. Real-time dictionaries are created. Decoders can rebuild a similar vocabulary based on the coding during the extraction phase and then return the stored information.

LZW's fundamental implementation of the original bridge vibration data is in denary decimal format, with a considerable length. Furthermore, the data lacks consistency and repeatability. As a result, the initial vibration data must first be processed. That is, before being compressed, the data must be converted to binary digits. The LZW compression algorithm and output are then integrated using the C language, based on unidirectional chained lists. The first dictionary is divided into three sections. '0','1', and '.' are the numbers. Reduction is a method that dynamically builds meaning while compressing raw data. When additional entries to the dictionary are needed, they should be added towards the end of the list.

Both the compression and recovery dictionaries have a fixed length of 256 characters. Sequential traversal is used to look for code in the vocabulary. The designed algorithm is capable of meeting the requirements for bridge

vibration data and has good reduction efficiency. However, because of the search's sequential traversal, each time spent on it is considerable, resulting in a long compression time.

By using the Constrained Application Protocol, we may link gadgets to the platform. CoAP can be deployed for both minimal and asset items, such as NB-IoT machinery. It also demonstrates how and where to authenticate a machine without the use of the Datagram Transport Layer Security (DTLS) or encryption algorithm.

Split, forecast, and modify are the three processes that make up the ambitious reforms for forward transformation. The beam is divided into even and unusual samples using the split method. The slow wavelet analysis is another name for this phase. Evenly balanced odd samples are substantially associated due to the signal's locally correlated nature. Odd samples are predicted from even ones in the forecast step [14]. Finally, the updating step ensures that smooth approximation signal and the starting signal have the same average. To obtain a more comprehensive understanding of the lifting procedure. There are major benefits to using the lifting method on IoT devices:

- (i) It leads to a faster wavelet transform solution and is ideal for low-energy practical uses.
- (ii) It saves memory and allows for full in-place calculations.
- (iii) Extremely simple to comprehend, build, deploy, and adjust to a variety of nodes.

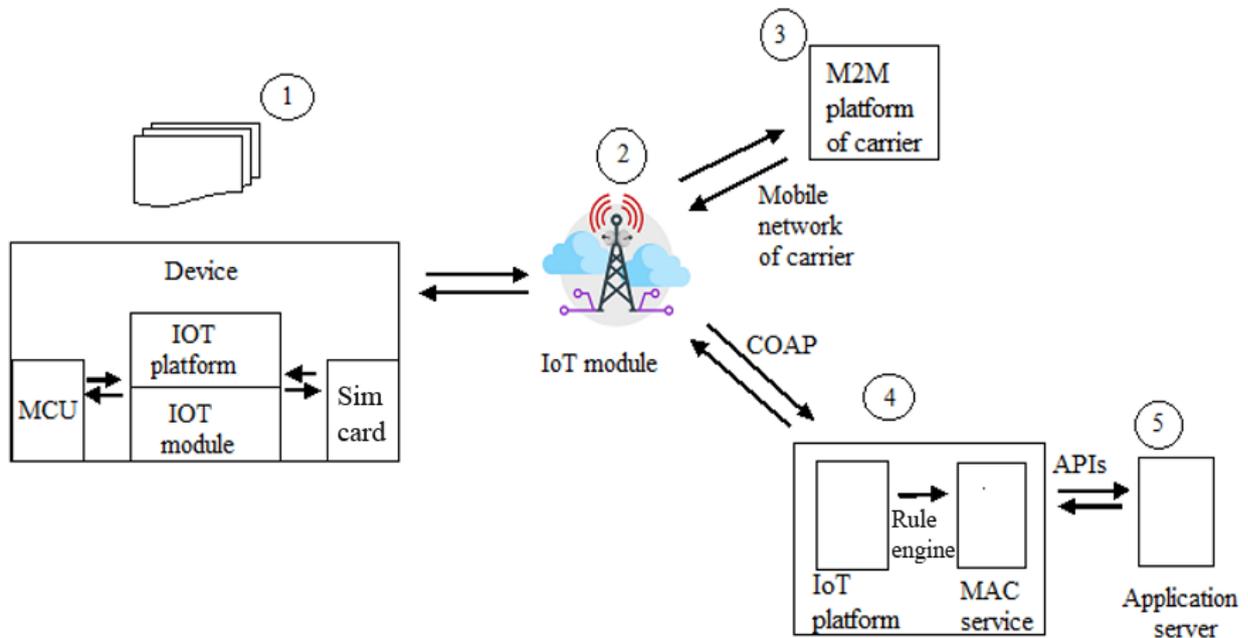


Figure 2. The IoT module is interfaced to an IoT network.

Figure 2 shows serial numbers 1-5, describing how and where to:

1. Consolidate an IoT application SDK (Software Development Kit) through into IoT modules of a product certification in the IoT Application dashboards then transmit the device license to the devices.
2. Attach an IoT system to the mobile phone network of a provider. Check with the local provider to verify that the IoT connection is accessible close to where the equipment is placed.
3. To manage internet activity and tariffs, the carrier's machine-to-machine (M2M) platform. The operator provides the M2M network functions.
4. Gather information in real time, and then distribute this to an IoT platform utilizing CoAP or UDP (User Datagram Protocol). IoT Platform allows you to protect the network from hundreds of consumer devices and handle large volumes of data. IoT technology also allows users to send data to a range of AliBaba cloud computing services for further processing. Massive data technologies, database servers, and data storage are instances of these facilities.

5. Be using the IoT Platform's information access-related API (Application Programming Interface) operations and messages pushing services to transfer data to company servers and connect devices.

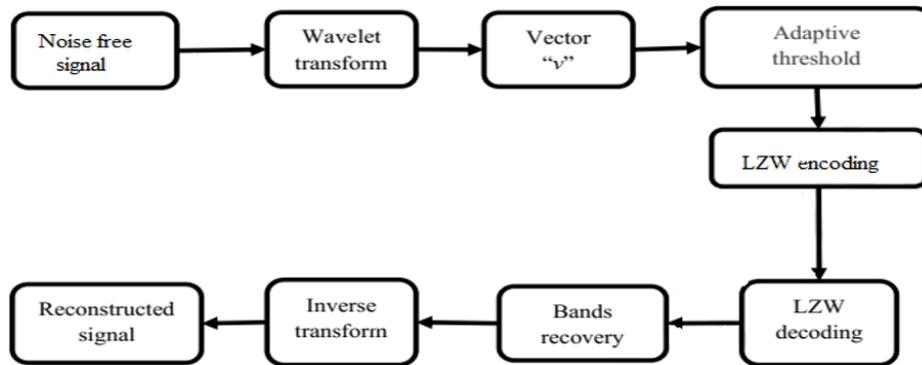


Figure 3. Wavelet and LZW encoding are being used to compress data.

As Figure 3 illustrates, data extraction is accomplished by the use of WT (Wavelet Transform), thresholding, and LZW. A noise-free signal was employed in this scenario, and the information is divided into six signals utilizing WT. The decomposed signal components are concatenated into a single feature vector (i.e., vector "v"), which is then subjected to thresholding. The vector components are then encoded using LZW. The initial stage in data reconstruction is to apply LZW decoding to the compacted signal. Following this, bands are generated using a coefficient distribution. Finally, inverse WT is used to recover the data. Decoding LZW includes extracting values from encoded data and then generating a coherent string from vocabulary that has been established. To reconstruct this same vocabulary in the same manner because it was constructed all through embedding, it also uncovers a next significance as from i/p and keep adding to the vocabulary, the sequence of a prevalent comment stream and the first main character of the comment stream procured by decrypting the next data input, or first main character of a thread, simply extracting if the next assessment can also be decoded.

2.1.1. Proposed Compression Scheme

Sensor information obtained in IoT applications is often a multimodal time-series. $X[X_1, X_2, \dots, X_M]$ is an M-dimensional multivariate time-series data made up of M separate time series data. The lifting approach is used to turn each univariate moving average into approximations and detail sets in our compression scheme. This process is known as column lifting. The ideas are then deleted, leaving just an M-dimensional collection of approximations $A = [A_1, A_2, \dots, A_M]$. Because of its simplicity, the hoisting method was employed using the Haar wavelet. By reapplying the wavelet operator to the likelihood function, various layers of modification can be achieved. The number of channels can be set by the user based on usage as well as data collection. The estimated array is then processed by the SZ method.

Based on the assumption that SZ is a specified concept and how irregular or ragged raw time-series data affects its outputs, it was coupled with the hoist strategy. The hoist technique conversion yields a tighter form of the initial start period, enabling the SZ's best-fit bending to help predict data coming within the user error. When data hits the edge or sink, the set of data adopts new tactics and uses the SZ method to decode its condensed approximation variables [1]. An inverted wavelet isn't necessary in order to extract all factual information because the wavelet transform data is loaded into a learning algorithm, as shown in Figure 4.

In this enhanced version of the technique, a novel mechanism called forward-moving on frequently-used elements is employed to improve the data architecture of chained lists. It may be inferred that the utilization of the afore-mentioned technique enhances search efficiency and decreases compression duration. In addition, achieving placement of often occurring forward entries through the selection of appropriate statistical frequencies leads to enhanced optimization outcome.

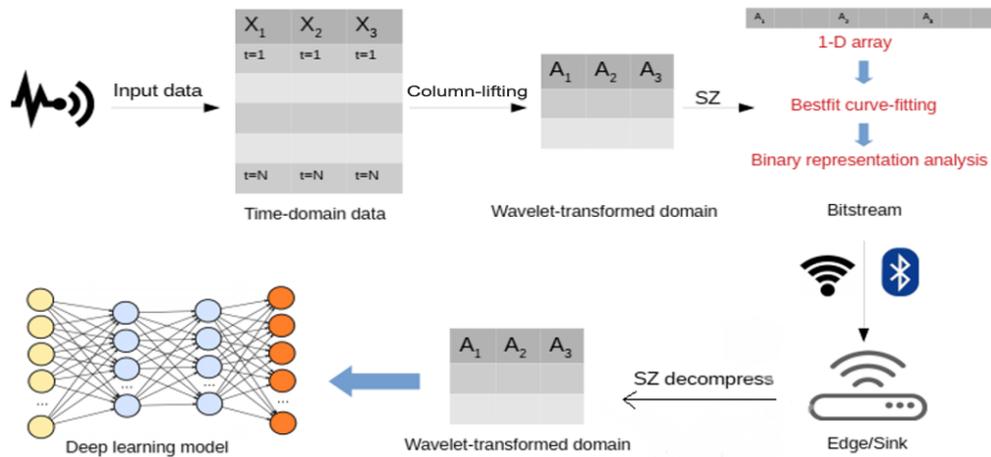


Figure 4. Compression technique for IoT time series classification.

2.1.2. LZW Encoding

LZW (Lempel-Ziv-Welch) is a lossless process. This technique is indeed a multipurpose method with various benefits over other programming methodologies, including simplicity and flexibility. In general, this technique compresses data via coding, resulting in a file that is half the size of the original. In various applications, such as tabular numbers, computer code editors, and signaling, the LZW approach delivers excellent compressor results [15]. The LZW-encrypted data in this case is entirely made up of 12-bit code, each of which relates to one of the code entries. Decompression is achieved by determining which characters and letters each symbol in the zip format represents using the coding table. Individual bytes from the input data are always assigned to the code table's codes 0-255. If only the first 256 codes had been used, every pixel in the source would have been converted to 12 bits within the LZW encoded file, which resulted in a file that was actually 2.5 times larger. During deflation, each 12-bit signal would be transformed back into single bytes using the code table.

The Figure 5 describes the Flow chart of LZW Algorithm:

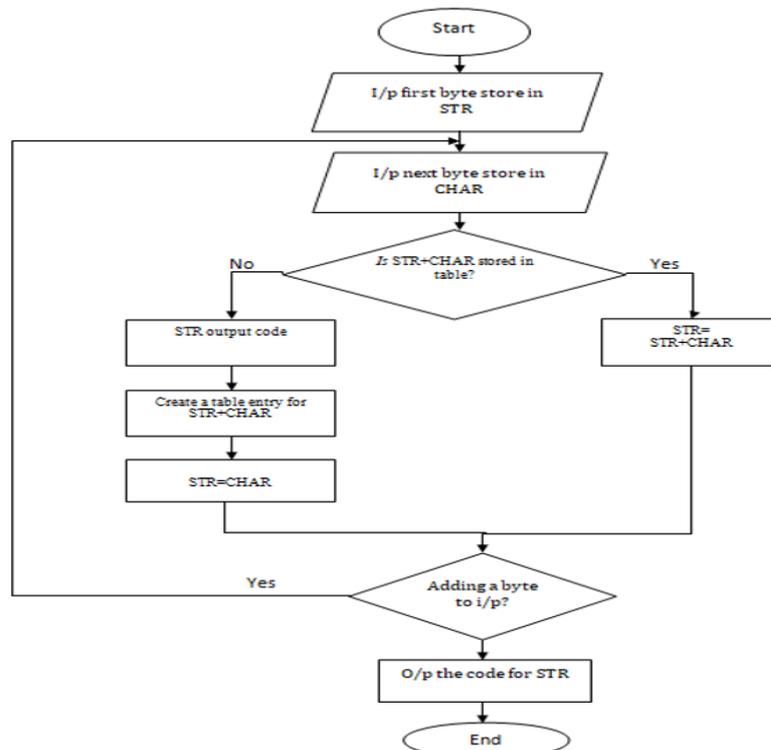


Figure 5. Flow-chart for the procedure followed in LZW.

The LZW encoding process consists of the following steps:

Step I: Initialization

- Begin with the initialization of the dictionary.

Step II: Identifying the Longest Phrase

- Search the current input for the longest matching phrase in the database.

Step III: Vocabulary Reduction

- Reduce the vocabulary by producing 'W' and removing 'W' from the input.

Step IV: Adding the Next Correlation

- Incorporate the next correlation into the input, following the addition of 'W'.

Step V: Completing the Cycle

- Return to Step II to complete the next iteration of the process.

The single-character sequences that correlate to all potential i/p letters are compiled into a vocabulary. A format string entrance is discovered by looking for larger substrings in the incoming packet until it is located. An appropriate phrase is found, and the database code for the phrase without preceding elements is discovered and led to output, and new entries are made to the dictionaries with the next character.

The last i/p letter is then used as the next starting point for searching for substrings. The same approach will be used; successively longer strings will be stored in the dictionary and made available, with the goal of encoding as single o/p numbers in the end, as shown in [Figure 5](#). This strategy is most effective on data with repeating configurations; as a result, the first bits of any data are compressed very little. However, as the size of the data grows, the CR tends to reach its maximum approximation.

LZW coding is used to reduce energy usage while increasing network lifetime. Files compressed using LZW are 1/3 the size of their original size. LZW achieves the maximum compression ratio in comparison to RLE for a variety of file types, such as text, speech, and images. In mobile ad-hoc networks, it's very popular. S-LEC is an LEC variant that employs a sequence coding strategy. To evaluate S-performance, LEC's actual statistics from sensor range or volcano tracking are contrasted with those of LEC and S-LZW. The CR and energy usage of sensor data were used as performance indicators to analyse the outcomes. When compared to LEC, S-LEC consumes less energy for dynamic volcano datasets.

2.1.3. Decoding

The LZW decoding procedure involves two steps: decoding the encoded data value and improving a cohesive phrase from the loaded vocabulary. In addition, it uncovers the next significance from i/p and continues to add the same sequence of the existing data and the first character in the sequence obtained by translating the next input data, which will be sufficiently simple to determine whether the next appraisal can be decrypted, to reconstruct the definitions in the same way that they were constructed all through encoding, and to reconstruct the same definitions through encryption.

The LZW decoding technique reverses the encoding process. The procedure is as follows:

Step I: Dictionary Creation

- Begin by creating a dictionary that initially holds all single characters from the data stream.

Step II: Finding the First Codeword (CoW)

- In the encoded data stream, locate the first codeword, denoted as CoW.

Step III: Providing the Decoded Data

- Decode the CoW and append its corresponding string to the decoded data stream.

Step IV: Reading the Following Codeword

- Read the next codeword, CoW, and assign it as PaW (Previous Word) = CoW.

Step V: Checking the Dictionary for CoW

- Check if the dictionary contains an entry for CoW.

Step VI: Continuing the Decoding Process

- If CoW is found in the dictionary, return to Step II to continue decoding.

Step VII: Ending Decoding

- If CoW is not in the dictionary, terminate the decoding process.

PaW= plain word, CoW = codeword, STR = string, CHAR = character

In response, the decoder continues to work until there is no more data and then interprets the final input vector without any additional vocabulary. As a consequence, decode builds a vocabulary that is a copy of the encoder's dictionary and uses it to decode the following i/p coefficients. This method does not require the entire vocabulary to be targeted with encrypted values. It is sufficient to use only the initial dictionary that contains solitary strings or Coefficients.

Algorithm 1: LZW compression algorithm

```
String j, char cc;
load character j
while(load character)
cc- move string to cc.
if(cc in dictionary)
Make code term;
else
cc should be updated, new characters added to cc.
Check the data in a dictionary
If (it isn't found in the dictionary), then
that string should be added to the dictionary;
end if;
```

The LZW algorithm uses a dictionary to generate a code for each character. In the dictionary, each character has a code and an index number. We read data from a file that we want to compress. Data is first entered into a buffer before being searched in a dictionary to generate a code. If the dictionary does not contain a matching character, then this character is entered into the dictionary, a new code is also assigned. If a character is found in a dictionary, a code will be generated for it. The number of bits in output codes is smaller than the number of bits in input data.

2.2. NetSim Simulator

A network simulation Tool called NetSim Simulator is used to examine the dynamics of communication networks using discrete event simulation. The NetSim Simulator is a powerful tool for simulating various protocols over IoT networks. In addition to supporting a variety of network elements, protocols, traffic types, and routing methods, it provides a highly modular platform for simulating IoT networks. NetSim is network simulation software that supports a variety of network protocols, such as machine-to-machine communication, CoAP, and MQTT.

2.2.1. Machine to Machine Communication Protocol

The protocol is open to all industries, and it's designed to support remote application management functionality for IoT devices. This protocol utilizes public networks and is cost-effective. A communication and data exchange environment is created between two machines. It allows machines to self-monitor and is able to adapt to changes in the environment.

2.2.2. Message Queue Telemetry Transmission

Message Queue Telemetry Transmission (MQTT) is a message protocol that is optimised for devices with limited processing capabilities. Based on the publish-subscribe paradigm, the protocol uses a flexible routing mechanism and asynchronous communication. MQTT works on the principle of TCP (Transmission Control Protocol); it contains three QoS levels:

- At level 0, messages are sent without any confirmation.
- At level 1, messages are sent to the sender, and it is guaranteed that they will be delivered, but there may be some duplicates.
- At level 2, there is also a guarantee of delivery, but only one message is delivered without duplicates.

Subscriber, publisher, and broker are the three primary components of MQTT.

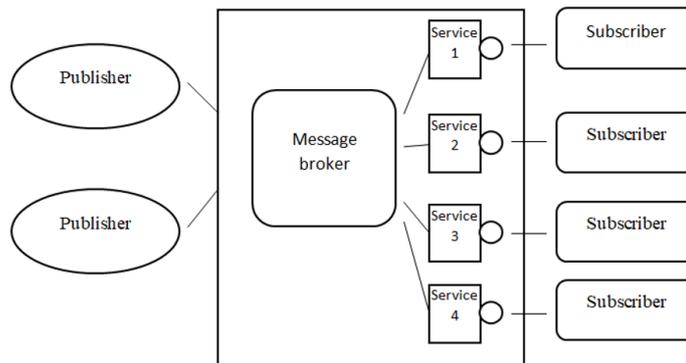


Figure 6. MQTT Architecture protocol.

A publisher can send and receive messages to subscribers. A publisher works with a broker to provide messages to subscribers about a given topic. Finally, the broker gets messages from publishers and sends them to interested subscribers, as shown in Figure 6.

2.2.3. Constrained Application Protocol (CoAP)

A resource-constrained application layer protocol is CoAP (Constrained Application Protocol), as shown in Figure 7. Because UDP is intrinsically unreliable, CoAP uses Provable (PRO) and Unprovable (UNP) messages to create its own dependability mechanism.

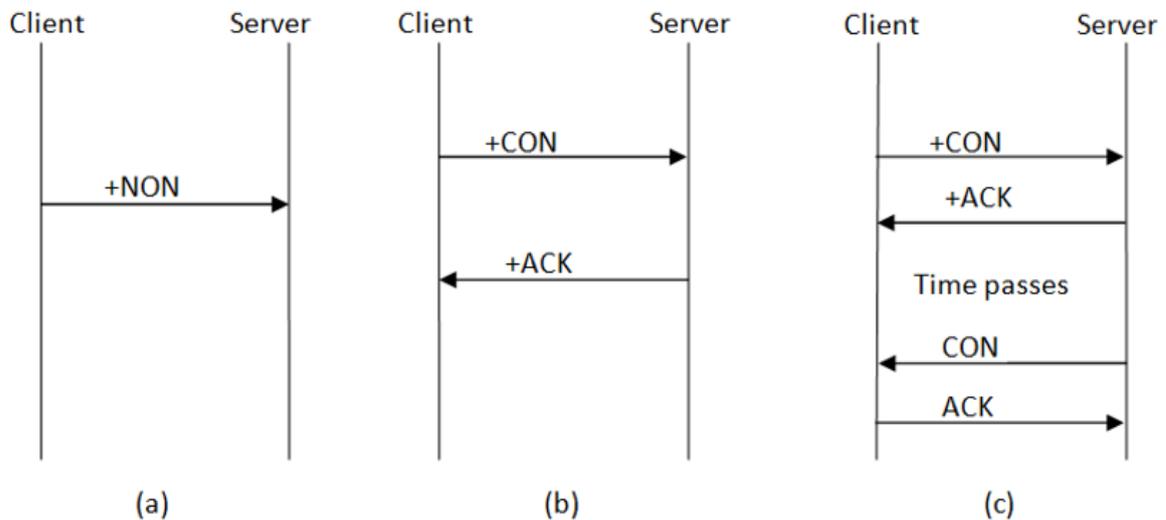


Figure 7. COAP a). Unprovable b). Hoist response c). Disparate response.

Note: NON= Non confirmation, CON= Confirmation.

A message that is Unprovable does not need to be confirmed. A server may respond with a reset message if it is unable to process an Unprovable message (RST). A confirmable message is used when an acknowledgement is requested. Furthermore, CoAP can deliver two different types of responses: hoisted and disparate. In hoisting, the response is returned in the request's acknowledgement (ACK) packet. Because the client would retransmit the request if the ACK packet with the hoisted data was lost, there is no need to acknowledge the hoisted response individually. The acknowledgement and response messages are sent in distinct packets in a disparate response. A response packet is sent by a server after it has processed the request. Following that, the client sends an acknowledgement message after getting the server answer. A separate response is useful when the server cannot answer a request in a confirmation message right away. The server will respond with a standard acknowledgment message, eliminating the need for the client to resend the request.

2.3. Performance Measures

Within this section, we explore and compare these three messaging protocols: M2M, CoAP, and MQTT. These messaging protocols are quite extensive and distinct from one another due to the variety of procedures and demands they've evolved through. The exact and relative comparisons of these systems depend on the many types of IoT devices, resources, apps, and the conditions and requirements assigned to each system. We compared routing overhead, average end-to-end delays and throughput in order to arrive at an appropriate routing protocol scheme for the IoT.

2.3.1. Routing Overhead

The routing overhead is the average number of route packets required for each data packet. This parameter reflects both network congestion and node power efficiency. Routing overhead is calculated with Equation 4:

$$\frac{N_1}{N_2} \quad (4)$$

It is N_1 packets that are sent and forwarded; it is N_2 packets that are received. The routing overhead is measured in bits per second (bits per second, or bps).

2.3.2. Average End to End Delay

The average end-to-end delay is affected by certain factors, like storage at the time of route discovery, lining at the interface queue storage, delays occurring during MAC (Media Access Control) retransmission, and transmission time. This figure is crucial for estimating the length of time it takes to find a path. For evaluating end-to-end delay, Equation 5 is used.

$$\sum \frac{(T_1 - T_2)}{N} \quad (5)$$

Where, T_1 is the arrival time of first data packet at destination time, T_2 is the time when the first packet is transmitted by source and N is the number of packets sent.

2.3.3. Throughput

As a percentage, throughput represents how many packets are received by the destination as compared to all packets sent by the source. The throughput is calculated with the help of Equation 6.

$$\frac{m_1}{m_2} \times 100\% \quad (6)$$

Where m_1 is the packets received by the destination and m_2 is the packets sent by the source. In bits per second (bits/s) the throughput is measured.

3. EXPERIMENTAL SETUP & FINDINGS

In this section, we define both the simulation setup and hardware implementation of our system in Figures 8 and 9, respectively. For the simulation setup, we have chosen the NetSim simulator and applied our IoT network with and without LZW compressions as shown in Figure 8. For hardware implementation, we have used sensors, an Arduino board, and programmed it with and without LZW compression, as shown in Figure 9.

First of all, we have designed our IoT network for simulation, which is described in the form of table given below in Table 1. In this setup, we have designed a network with a size of 1000*1000, 50 to 300 nodes, and a bandwidth of 2 mbps. For transferring the data, we have chosen the M2M, MQTT, and CoAP protocol and a packet size of 1024 bytes with rate of 5 packets per second. The simulation time is approximately 250-750 seconds, with a pause time of 15 seconds. The placement of nodes is random. Setup is shown in Figure 8.

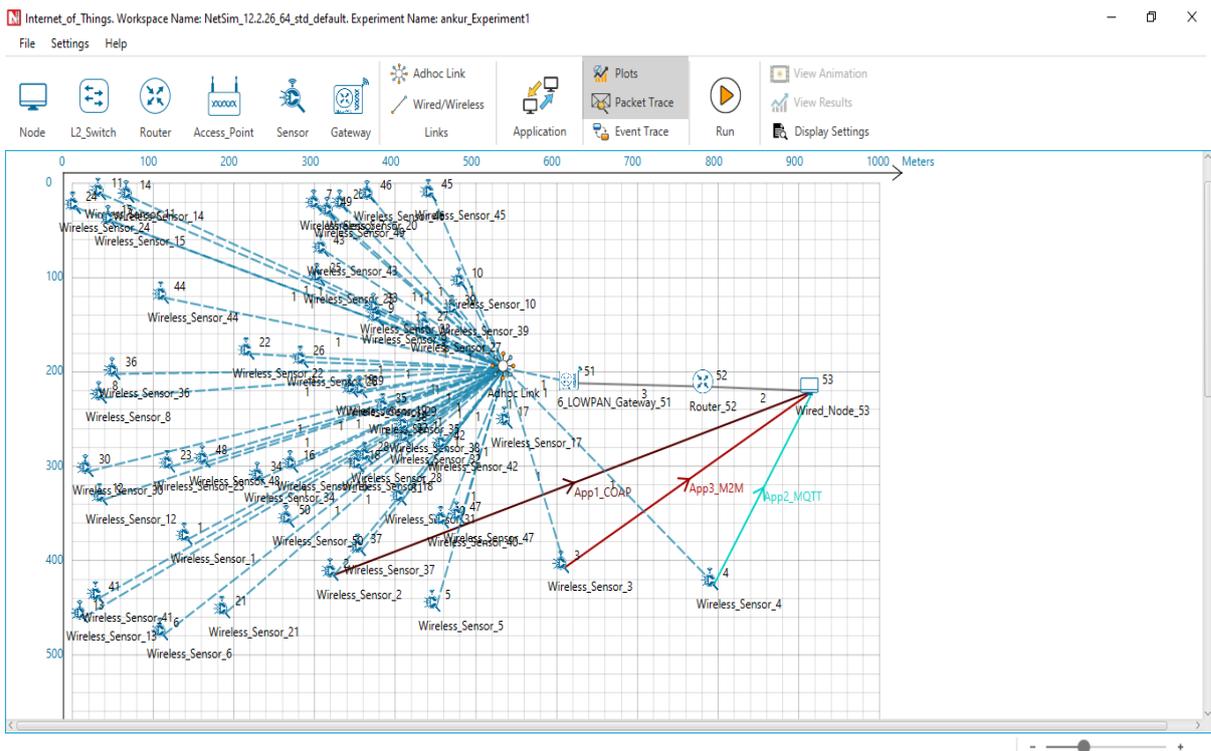


Figure 8. Simulation of IoT network with 50 nodes random placement.

Table 1. IoT network Simulation parameters.

Parameter	Value
Network size	1000*1000m
Number of nodes	50 to 300
Bandwidth	2 Mbps
Routing protocols	M2M, MQTT, CoAP
Packet size	1024 bytes
Packet rates	5 Packets/sec
Simulation time	250-750 sec
Pause time	15 sec
Node placement	Random

For hardware implementation, we have used sensors, an Arduino board, and programmed it with and without LZW compression, as shown in Figure 9.



Figure 9. IoT model.

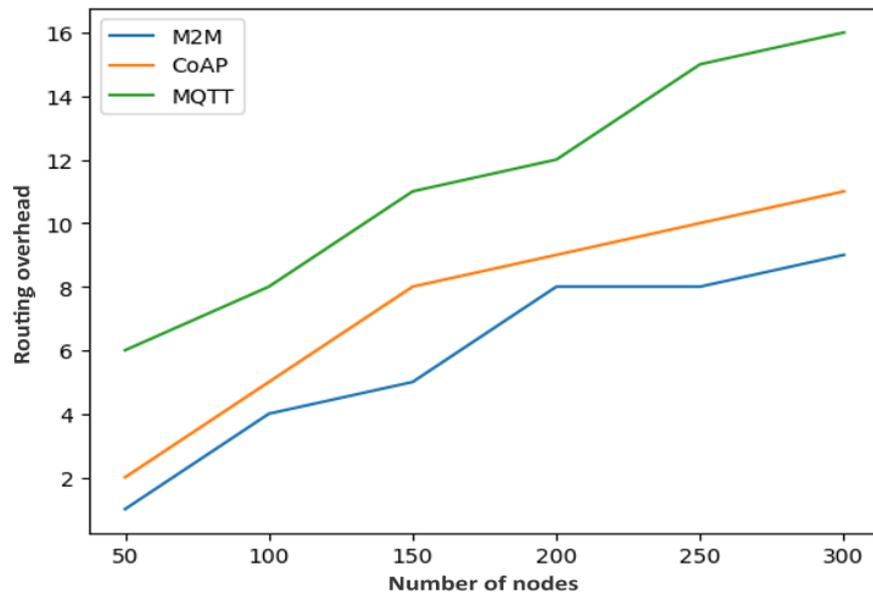


Figure 10. Routing overhead without compression with respect to variable number of nodes.

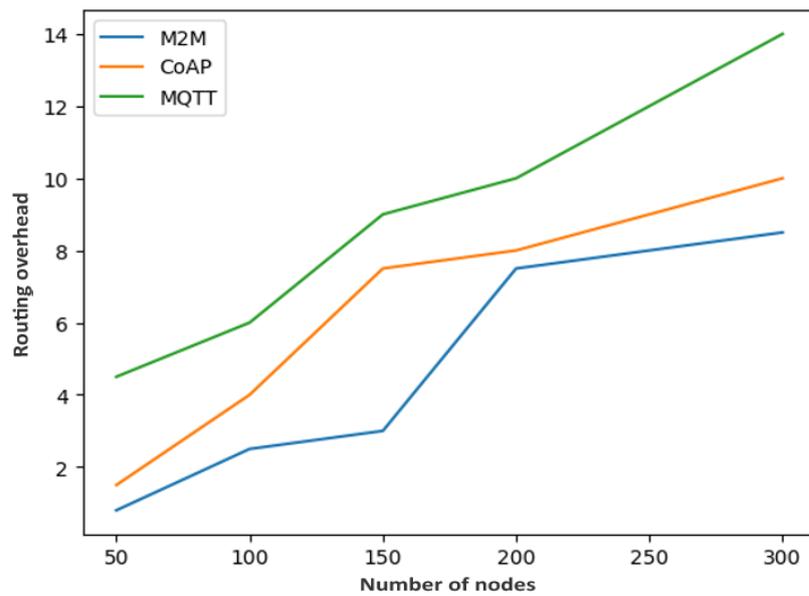


Figure 11. Routing overhead with compression with respect to variable number of nodes.

3.1. Results

3.1.1. IoT network with and without Compression

3.1.1.1. Routing overhead

In this, the performance of all three IoT routing protocols, M2M, CoAP, and MQTT, is shown in Figure 10 and 11, for the performance parameter, i.e., routing overhead. In my overall view, routing overhead increases with the number of nodes, in both with and without data compression. This is because we have to send more control packets to manage large amounts of nodes. If we compare the routing overhead individually for each protocol with and without data compression, it states that the routing overhead is greater without data compression for each protocol in comparison to with data compression. For example, with 50 nodes in M2M Routing Overhead without data compression, it is 1 and with data compression, it is 0.8; similarly, with 150 nodes in CoAP Routing Overhead without data compression, it is 8 and with data compression, it is 7.5, and also with 300 nodes in MQTT Routing Overhead without data compression, it is 16 and with data compression, it is 14. The main reason behind this is that, with the help of data compression, we are able to somehow reduce the size of data packets. From the above results, we can also see that MQTT has the biggest routing overhead compared to the other two protocols.

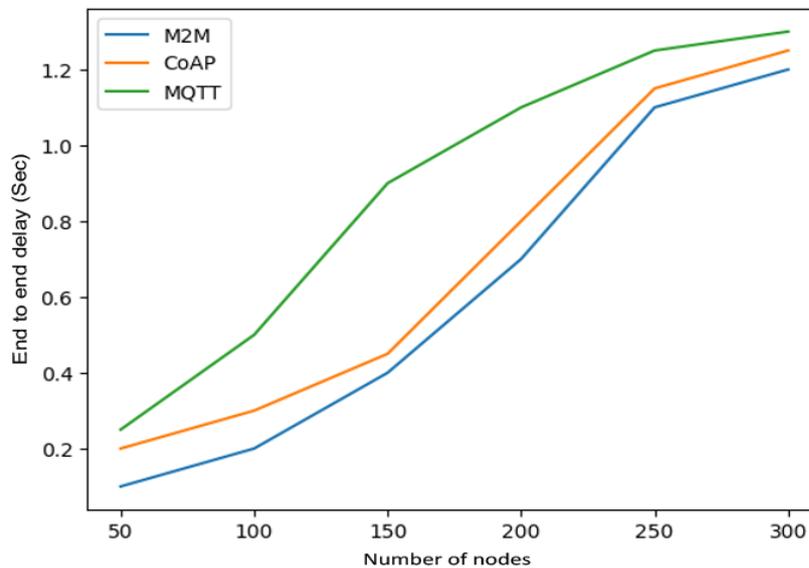


Figure 12. End to end delay without compression with respect to variable number of nodes.

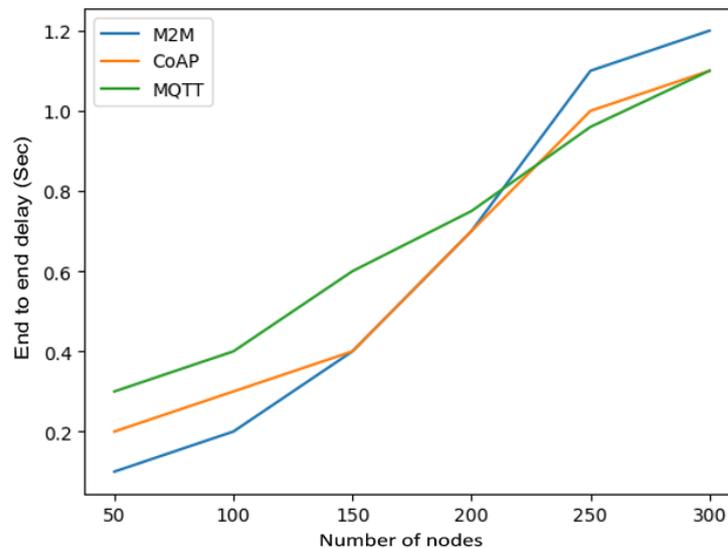


Figure 13. End to end delay with compression with respect to variable number of nodes.

3.1.1.2. End to End Delay

The average end-to-end delay is a very crucial performance parameter for any network, as it tells the overall time duration of any packet. As the number of packets increases from 50 nodes to 300, there is an increase in end-to-end delay. But with the implication of data compression end-to-end delay decreases in comparison to without data compression as the size of the data packet decreases because of the data compression. With the help of Figures 12 and 13, we can easily interpret that with 200 nodes in M2M, the end-to-end delay without data compression is 0.7 and with data compression is 0.4; similarly, with 250 nodes in the CoAP end-to-end delay without data compression is 1.15 and with data compression is 1.1; and also with 300 nodes in MQTT, the end-to-end delay without data compression is 1.3 and with data compression is 1.1. Based on the results of the comparison, we can conclude that if the primary goal is to develop speedy communication in future IoT applications, the M2M protocol mechanism is the best option.

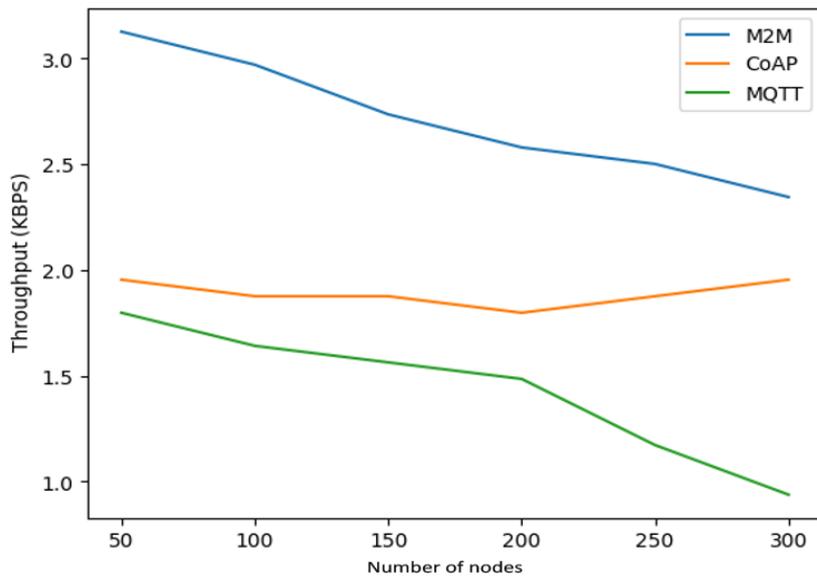


Figure 14. Throughput without compression with respect to variable number of nodes.

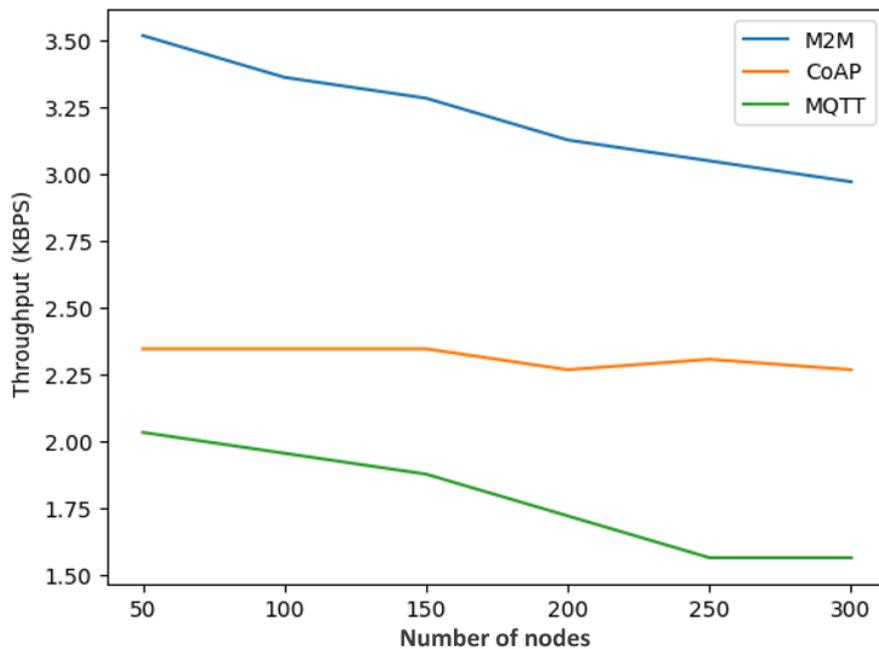


Figure 15. Throughput with compression with respect to variable number of nodes.

3.1.1.3. Throughput

Throughput is the specific number of data packets received by the receiver and sent by the sender at a particular time. The efficiency of the network can be evaluated with the help of throughput. In the overall analysis from Figures 14 and 15, we can see that as the number of nodes increases with and without data compression, the protocol throughput decreases. For 50 nodes in M2M throughput without data compression, it is 3.125 and with data compression, it is 3.515625; similarly, for 150 nodes in CoAP throughput without data compression, it is 1.875 and with data compression, it is 2.34375 and also for 300 nodes in MQTT throughput without data compression, it is 0.9375 and with data compression, it is 1.5625. In comparison with the other two routing systems, MQTT has the lowest throughput.

4. CONCLUSION

This paper presents transform-based compression algorithms for an LZW signal. The presented data compression techniques help in improving the efficiency of the network, as we measured with the help of different performance parameters as shown above. On the basis of the trials carried out in this study, many conclusions can be drawn. Remember that the main purpose is to provide a reduction method that will help optimize the lifetime of an IoT device. It is adaptable to every application, sensor, or activity and is appropriate for relatively close applications. Here, we overcome these limitations by incorporating LZW data compression into it. The data compression technique would be beneficial in reducing the size of data packets, and increasing it will increase the throughput, and decreasing the delay and routing overhead. Furthermore, this is an area that is rapidly growing and evolving, so it is possible that future developments may change the scenario described. A real-world IoT system could be an interesting platform for testing these protocols in the future.

Funding: This study received no specific financial support.

Institutional Review Board Statement: Not applicable.

Transparency: The authors state that the manuscript is honest, truthful, and transparent, that no key aspects of the investigation have been omitted, and that any differences from the study as planned have been clarified. This study followed all writing ethics.

Competing Interests: The authors declare that they have no competing interests.

Authors' Contributions: Conceived and designed the paper, wrote the paper, A.S.; analyzed and simulate the model, A.K.Y. Both authors have read and agreed to the published version of the manuscript.

REFERENCES

- [1] J. Azar, A. Makhoul, M. Barhamgi, and R. Couturier, "An energy efficient IoT data compression approach for edge machine learning," *Future Generation Computer Systems*, vol. 96, pp. 168-175, 2019. <https://doi.org/10.1016/j.future.2019.02.005>
- [2] A. Monther, M. Y. Abdul, Hamzah, and J. Moath, "Multi PLZW: A novel multiple pattern matching search in LZW-compressed data," *Computer Communications*, vol. 145, pp. 126-136, 2019. <https://doi.org/10.1016/j.comcom.2019.06.011>
- [3] U. Nandi and J. K. Mandal, "Modified compression techniques based on optimality of LZW code (MOLZW)," *Procedia Technology*, vol. 10, pp. 949-956, 2013. <https://doi.org/10.1016/j.protcy.2013.12.442>
- [4] M. Wilkinson, "A simple data compression scheme for binary images of bacteria compared with commonly used image data compression schemes," *Computer Methods and Programs in Biomedicine*, vol. 42, no. 4, pp. 255-262, 1994. [https://doi.org/10.1016/0169-2607\(94\)90097-3](https://doi.org/10.1016/0169-2607(94)90097-3)
- [5] M. Mandal and R. Dutta, "Identity-based outsider anonymous cloud data outsourcing with simultaneous individual transmission for IoT environment," *Journal of Information Security and Applications*, vol. 60, p. 102870, 2021. <https://doi.org/10.1016/j.jisa.2021.102870>
- [6] A. Fathalla, K. Li, A. Salah, and M. F. Mohamed, "An LSTM-based distributed scheme for data transmission reduction of IoT systems," *Neurocomputing*, vol. 485, pp. 166-180, 2022. <https://doi.org/10.1016/j.neucom.2021.02.105>

- [7] U. Jayasankar, V. Thirumal, and D. Ponnurangam, "A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications," *Journal of King Saud University-Computer and Information Sciences*, vol. 33, no. 2, pp. 119-140, 2021. <https://doi.org/10.1016/j.jksuci.2018.05.006>
- [8] U. Yaman and M. Dolen, "Direct command generation for CNC machinery based on data compression techniques," *Robotics and Computer-Integrated Manufacturing*, vol. 29, no. 2, pp. 344-356, 2013. <https://doi.org/10.1016/j.rcim.2012.09.005>
- [9] A. Izaddoost and M. Siewierski, "Energy efficient data transmission in IoT platforms," *Procedia Computer Science*, vol. 175, pp. 387-394, 2020. <https://doi.org/10.1016/j.procs.2020.07.055>
- [10] F. Zhang, Z. Li, M. C. Wen, X. Jia, and C. Chen, "Implementation and optimization of LZW compression algorithm based on bridge vibration data," *Procedia Engineering*, vol. 15, pp. 1570-1574, 2011. <https://doi.org/10.1016/j.proeng.2011.08.292>
- [11] F.-Y. Wu, K. Yang, and X. Sheng, "Optimized compression and recovery of electrocardiographic signal for IoT platform," *Applied Soft Computing*, vol. 96, p. 106659, 2020. <https://doi.org/10.1016/j.asoc.2020.106659>
- [12] Z.-H. Wang, H.-R. Yang, T.-F. Cheng, and C.-C. Chang, "A high-performance reversible data-hiding scheme for LZW codes," *Journal of Systems and Software*, vol. 86, no. 11, pp. 2771-2778, 2013. <https://doi.org/10.1016/j.jss.2013.06.024>
- [13] J. Uthayakumar, T. Vengattaraman, and P. Dhavachelvan, "A new lossless neighborhood indexing sequence (NIS) algorithm for data compression in wireless sensor networks," *Ad Hoc Networks*, vol. 83, pp. 149-157, 2019. <https://doi.org/10.1016/j.adhoc.2018.09.009>
- [14] S. Chandra, A. Sharma, and G. K. Singh, "A comparative analysis of performance of several wavelet based ECG data compression methodologies," *IRBM*, vol. 42, no. 4, pp. 227-244, 2021. <https://doi.org/10.1016/j.irbm.2020.05.004>
- [15] A. Moon, J. Kim, J. Zhang, and S. W. Son, "Evaluating fidelity of lossy compression on spatiotemporal data from an IoT enabled smart farm," *Computers and Electronics in Agriculture*, vol. 154, pp. 304-313, 2018. <https://doi.org/10.1016/j.compag.2018.08.045>

Views and opinions expressed in this article are the views and opinions of the author(s), Review of Computer Engineering Research shall not be responsible or answerable for any loss, damage or liability etc. caused in relation to/arising out of the use of the content.