*International Journal on Recent and Innovation Trends in Computing and Communication*
*ISSN: 2321-8169 Volume: 11 Issue: 9s*
*DOI: https://doi.org/10.17762/ijritcc.v11i9s.7425*
***Article Received**: 12 May 2023 **Revised:** 26 June 2023 **Accepted:** 25 July 2023*

# Analysis of the Performance of IoT Networks in Acoustic Environment by using LZW Data Compression Technique

**Ankur Sisodia[1], Dr. Ajay Kumar Yadav[2]**
[1]Research Scholar
Banasthali Vidyapith,
Rajasthan
ankur22887@gmail.com
[2]Assistant Professor
Banasthali Vidyapith,
Rajasthan
ajay.iitdhn@gmail.com

**Abstract**—The Internet of Things (IoT) has experienced phenomenal growth, opening up a wide range of applications in many settings. Due to the properties of sound propagation, IoT networks operating in acoustic environments in particular present special difficulties. Data compression techniques can be used to minimize overhead and maximize resource utilization in these networks to increase performance. The performance of IoT networks in acoustic environments is examined in this study, with a focus on routing overhead, throughput, and typical end-to-end delay. Lempel-Ziv-Welch (LZW) data compression is used to reduce data size and boost communication effectiveness. Three well-known protocols—MQTT, CoAP, and Machine-to-Machine (M2M)—are assessed in relation to acoustic Internet of Things networks. To mimic different acoustic conditions and collect performance metrics, a thorough experimental setup is used. Different network topologies, data speeds, and compression settings are used in the studies to determine how they affect the performance metrics. According to the analysis's findings, all three protocols' routing overhead is greatly decreased by the LZW data compression approach, which enhances network scalability and lowers energy usage. Additionally, the compressed data size has a positive impact on network throughput, allowing for effective data transmission in acoustic contexts with limited resources. Additionally, using LZW compression is seen to minimize the average end-to-end delay, improving real-time communication applications. This study advances knowledge of IoT networks operating in acoustic environments and the effects of data reduction methods on their functionality. The results offer useful information for network engineers and system designers to optimize the performance of IoT networks in similar situations. Additionally, a comparison of the MQTT, CoAP, and M2M protocols' suitability for acoustic IoT deployments is provided, assisting in the choice of protocol for particular application needs.

**Keywords**- Underwater IoT, M2M, MQTT, CoAP, Performance Parameters.

## I. INTRODUCTION

Recent advancements in the IoT have spawned a plethora of new applications in a variety of industries, including transportation, healthcare, and environmental monitoring [1]. These IoT networks frequently operate in difficult conditions, such underwater or in acoustically active areas, where traditional communication methods encounter substantial challenges. To sustain the overall functionality of these networks, it is essential to provide efficient and dependable data transfer [2].

The purpose of this research article is to evaluate the performance of IoT networks in acoustic situations using the LZW (Lempel-Ziv-Welch) data compression technique. The main emphasis is on measuring important performance indicators, such as throughput, routing overhead and average end-to-end delay. Additionally, the study explores the efficiency of three well-known protocols, namely MQTT, COAP, and machine-to-machine (M2M) communication, in promoting communication within the IoT network.

For IoT networks, the acoustic environment presents special difficulties since sound waves can cause interference and signal degradation, which increases packet loss, increases latency, and decreases network efficiency [3]. In such acoustic situations, the LZW data compression technology provides a potential remedy by lowering the size of data packets, improving transmission effectiveness, and maximizing network performance.

Several metrics are essential for evaluating the operation of IoT networks. In order to maintain network connectivity and guarantee dependable message delivery, routing systems transmit additional data known as routing overhead [4]. The pace at which data may be effectively sent between IoT devices is measured by throughput [5], and the time it takes for a packet to travel from its source to its destination is known as average end-to-end delay [6].

This study examines the efficacy of three protocols—MQTT, COAP, and machine-to-machine communication—in addition to measuring performance characteristics. A communications protocol called MQTT was created expressly to be small and suited for Internet of Things (IoT) applications. COAP, on the other hand, is a unique protocol designed for equipment with constrained resources and capabilities. Direct device-to-device connection is prioritized in machine-to-machine communication, doing away with the need for a centralized server. The feasibility of these protocols for IoT networks operating in acoustic environments can be determined by comparing them [7, 8, 9].

The results of this study will improve the functionality and dependability of IoT networks in acoustic settings by providing useful knowledge for the development and improvement of communication protocols. The findings could be used to build effective deployment techniques for IoT applications in situations like undersea monitoring, tracking wildlife, and industrial automation, where acoustic conditions are important.

## II. UNDERWATER IOT ARCHITECTURE

The Internet of Things (IoT) architecture for underwater environments typically consists of several layers to facilitate connectivity, data collecting, and control. Depending on the particular requirements and technology employed, the architecture may change [10]. A layered, typical underwater IoT architecture is shown here in a simplified form:

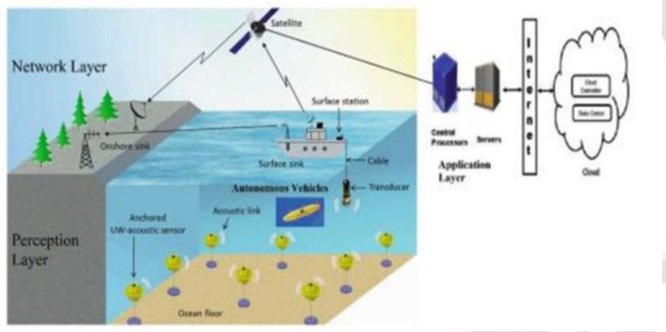

Figure 1. Underwater IoT Architecture [10]

### A. *Sensor Layer:*

- This layer is made up of numerous sensors created especially for data gathering and monitoring underwater. Examples include sensors that monitor acoustics, salinity, pH, pressure, temperature, and salinity.
- These sensors translate the ambient information they collect from the underwater environment into electrical impulses.

### B. *Communication Layer:*

- Data transmission from the sensor layer to the higher levels is facilitated by the communication layer.
- Technologies for underwater communication, such acoustic or optical communication (using LEDs or lasers, for example), are frequently utilised.
- Through these underwater communication channels, the sensor data is modulated and transferred.

### C. *Gateway Layer:*

- The gateway layer collects the data transmitted by the underwater devices and carries out the necessary protocol conversions and data preparation. It serves as a bridge between the underwater communication layer and the surface.
- Signal amplifiers, modulators, demodulators, error-correcting techniques, and encryption for safe data transfer may be included in the gateway.

### D. *Network Layer:*

- The network layer builds trustworthy communication channels, controls network addresses, and makes sure that data flows effectively between various undersea equipment and the surface.
- Protocols like IPv6 (Internet Protocol version 6) or other protocols tailored for underwater networks may be used by the network layer.

### E. *Cloud/Server Layer:*

- The cloud or server layer manages and processes the data that is received from the network layer. At this layer, data analyses, storage, and visualisation are carried out.
- To glean useful insights from the gathered data, sophisticated algorithms and machine learning approaches might be used.
- Through web-based or mobile applications, users can access the processed data and remotely operate the underwater equipment.

### F. *User Interface Layer:*

- The user interface layer offers an easy-to-use platform that makes it possible to engage with the underwater IoT system without any issues.
- Users can define thresholds, configure devices, monitor real-time data, and receive alerts or notifications.
- Depending on the particular needs, the user interface can be accessed through web portals, mobile apps, or custom applications.

*International Journal on Recent and Innovation Trends in Computing and Communication*
*ISSN: 2321-8169 Volume: 11 Issue: 9s*
*DOI: https://doi.org/10.17762/ijritcc.v11i9s.7425*
***Article Received**: 12 May 2023 **Revised:** 26 June 2023 **Accepted:** 25 July 2023*

## III. LZW COMPRESSION

A well-liked technique for compressing data without sacrificing any information is the Lempel-Ziv-Welch (LZW) compression algorithm. In file compression formats like GIF and TIFF, it is widely used. Using a dictionary to record frequently occurring patterns from the input data, LZW replaces them with shorter codes [11, 17].

The LZW algorithm's step-by-step instructions for coding (compressing) and decoding (decompressing):

**Coding (Compression):**

1. Fill a dictionary with all single-character entries that can be made.
2. Character by character, read the input data.
3. Include the most recent character in the existing pattern.
4. If the dictionary has the current pattern, update it to add the upcoming character and repeat step 4 if not.
5. If the dictionary does not already include the current pattern, print the code for the previous pattern, add the current pattern, and then reset the current pattern to the character that was most recently read.
6. Carry on with steps 3 through 5 until all of the input data has been read.
7. Generate the final current pattern's code.

**Decoding (Decompression):**

1. Fill a dictionary with all single-character entries that can be made.
2. Read each input code separately.
3. Get the dictionary's entry for the matching pattern for the present code.
4. Print the pattern's characters.
5. Add the first character of the recovered pattern to the existing pattern if there are more codes.
6. Add the first character of the obtained pattern to the existing pattern to create a new item in the dictionary.
7. Set the recovered pattern as the current pattern in step seven.
8. Continue performing steps 3 through 7 until all input codes have been handled.

It's important to note that both the coding and decoding processes should use the same initial dictionary setup.

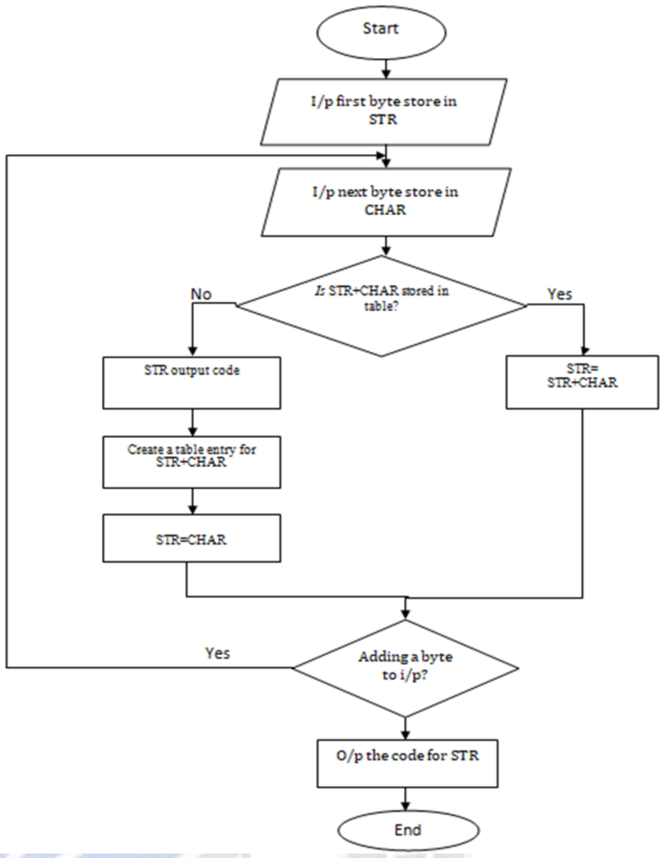
Start
I/p first byte store in STR
I/p next byte store in CHAR
Is STR+CHAR stored in table?
No
Yes
STR output code
Create a table entry for STR+CHAR
STR=CHAR
STR= STR+CHAR
Yes
Adding a byte to i/p?
O/p the code for STR
End

Figure 2. LZW Compression Flowchart

The basic steps of the LZW compression algorithm are shown in this flowchart. It begins by reading input data, initializing the current string, and initializing the dictionary. Then it moves into a loop and looks for the dictionary's longest matched string. If the following character is present, the current string and the new item are both added to the dictionary. Following the display of the code for the current string, the programmed sets the following character as the new current string. Up until all of the input data has been processed, this procedure continues. The last code for the current string is then outputted, followed by an end-of-file marker to indicate that the compressed data has ended [12].

## IV. PROTOCOLS

To facilitate communication between objects and systems, the Internet of Things (IoT) relies on a number of protocols. In IoT networks, these protocols make data and information sharing easier. Here are a few popular Internet of Things protocols:

**MQTT:** MQTT, or Message Queuing Telemetry Transport, is a messaging protocol developed especially for low-resource devices and unreliable, low-bandwidth networks. It is frequently used in Internet of Things (IoT) applications and operates on a publish-subscribe mechanism [13]. The MQTT protocol's salient features are as follows:

1 Publish-Subscribe Model: The publish-subscribe pattern, in which there are publishers and subscribers, is the foundation of MQTT. Messages are sent to a subject by publishers (also known as "publishing"), and messages from certain topics that interest subscribers are received by subscribers (also known as "subscribing").

2 Broker: A central message broker acts as a go-between for publishers and subscribers in MQTT communication. The way it works is by getting messages from publishers and sending them to the right subscribers.

3 Topic: Specific topics serve as communication channels or categories and are to which messages are published. Slashes (/) are used to delineate the levels of a hierarchical structure for topics that are represented as strings. For instance, "sensors/temperature" or "home/living-room/lights."

4 Quality of Service (QoS): Messaging Queue Telemetry Transport (MQTT) provides various options for ensuring different levels of quality of service (QoS) during message delivery. There are three levels:
   - QoS 0 (At most once): The communication is either sent once or not at all, and there is no confirmation or assurance of its successful delivery.
   - QoS 1 (At least once): The message will be successfully delivered at least once. The broker sends an acknowledgment (PUBACK) to the publisher after receiving the message and retransmits it until it receives an acknowledgment from the subscriber.
   - QoS 2 (Exactly once): This statement ensures that the message will be delivered with certainty and without duplication. It requires a sequential four-step interaction between the publisher, broker, and subscriber to ensure dependable transmission.

5 Connection: MQTT employs a simple TCP/IP communication protocol. The client uses a chosen port (the default is 1883) to create a TCP connection with the broker. The alternative method of using MQTT is via a safe SSL/TLS connection (default: 8883).

6 Client Types: MQTT clients can be classified into two types:
   - Publishers: Clients who share messages on particular subjects.
   - Subscribers: This refers to individuals or systems that opt to receive messages that are published to specific topics they have subscribed to.

7 Retained messages, which are messages that are saved by the broker and are sent to subscribers when they subscribe to a subject, are another feature of MQTT. Sending new subscribers the most recent status or configuration information can be done with the help of retained messages.

8 Last Will and Testament (LWT): If a client disconnects suddenly, clients can provide a LWT message, which the broker automatically publishes. With the help of this feature, situations in which a client disconnects itself accidentally can be handled.

9 Compact: MQTT is designed to have a small footprint, making it suitable for devices with constrained resources like sensors or microcontrollers. It uses a small binary payload format for messages, minimising processing cost and network bandwidth.

10 Wide Language and Platform Support: MQTT is accessible and simple to integrate into multiple IoT systems due to the availability of libraries and implementations in numerous programming languages.

**Machine-to-Machine (M2M):** The phrase "Machine-to-Machine (M2M) protocol" refers to a set of communication rules and conventions that make it possible for machines to interact and share data without the need for human intervention.M2M protocols make it possible for linked devices to communicate with one another and share information and carry out coordinated operations [14].

Several protocols are frequently utilised in M2M communication, including

1 Extensible Messaging and Presence Protocol (XMPP): Originally developed for instant messaging, XMPP has now been modified to support machine-to-machine (M2M) communication. It is an open communication protocol based on the extensible markup language (XML). The seamless and fast transmission of messages and presence information between multiple devices is made possible via XMPP.

2 AMQP (Advanced Message Queuing Protocol): AMQP is an open-standard protocol that enables message exchange between applications. It is appropriate for M2M communication scenarios because it offers dependable, secure, and interoperable communications across several platforms and programming languages.

3 RESTful APIs (Representational State Transfer): RESTful APIs are popular for M2M communication even if they are not a particular protocol. Machines can communicate with one another using common

*International Journal on Recent and Innovation Trends in Computing and Communication*
*ISSN: 2321-8169 Volume: 11 Issue: 9s*
*DOI: https://doi.org/10.17762/ijritcc.v11i9s.7425*
*Article Received: 12 May 2023 Revised: 26 June 2023 Accepted: 25 July 2023*

HTTP methods like GET, POST, PUT, and DELETE thanks to RESTful APIs. They are frequently utilised in IoT applications and make use of the current web infrastructure.

Depending on the needs of the devices, the network environment, and the type of data being shared, these protocols, among others, offer many alternatives for M2M communication. The capabilities of the device, bandwidth constraints, the necessity for security, and the specifications for system compatibility are only a few of the variables that affect the choice of a protocol.

**Constrained Application Protocol (CoAP):** Constrained Application Protocol, or CoAP, is a particular web protocol for transmitting data. Its main objective is to make it easier for devices with resource restrictions, like limited power or reduced capabilities, to communicate with one another. This protocol, which operates at the application layer, provides simple communication between networks with constrained capabilities, such as those used in the Internet of Things (IoT), and devices with limited resources, such as sensors and actuators.

CoAP has been developed expressly to have a minimum, effective, and deployable nature, making it appropriate for use on devices with limitations on their processing power, memory, and energy resources. It uses the User Datagram Protocol (UDP) and offers request/response architecture akin to HTTP (Hypertext Transfer Protocol). However, HTTP is better suited for conventional web applications while CoAP is created expressly for limited contexts [15].

Key features of CoAP include:

1 RESTful Design: CoAP adheres to the Representational State Transfer (REST) principles, enabling resources to be found and used using common HTTP-like operations like GET, POST, PUT, and DELETE.

2 Resource Discovery: CoAP includes techniques for resource discovery, allowing devices to declare the resources they have available and allowing other devices to find and use those resources.

3 Low Overhead: CoAP has less protocol overhead than HTTP since it offers efficient message serialisation and employs compact binary encoding.

4 Request/Response Model: CoAP uses a streamlined methodology for making and receiving information requests, allowing clients to submit queries to servers and receive pertinent responses back. Both dependable and unreliable messaging modalities are supported.

5 CoAP-to-CoAP Proxying: CoAP can be bridged to HTTP through proxy servers, enabling integration with existing web services and applications.

6 Resource Observation: Clients can use CoAP to watch resources and get alerts when they change. Applications requiring real-time monitoring can benefit from this capability.

When resource-constrained devices need to connect with one another or with backend servers, CoAP is frequently utilised in IoT applications. It is the perfect solution for environments with limited resources since it provides a quick and efficient replacement for established protocols like HTTP.

## V. PERFORMANCE PARAMETER

*A. Routing overhead: T*he quantity of extra data or control information required for the purpose of routing within a network is referred to as "routing overhead" in the parameter. Depending on the exact routing protocol or network architecture under consideration, the mathematical method for estimating routing overhead may change [16, 18].

$$\text{Routing Overhead} = (\text{Total Routing Information} / \text{Total Data Traffic}) \times 100 \quad (1)$$

Total routing information in this formula refers to the quantity of data or control information transferred during routing, such as routing tables, routing updates, or signalling packets. The actual user data or payload being transmitted over the network is represented by the total data traffic.

*B. Throughput:* Throughput is a metric that measures how quickly messages or data are effectively delivered and is used to assess how effective a communication route or system is. The context and the units used to measure data transfer affect the mathematical method for determining throughput [16, 19].

The formula for throughput would be as follows if we assume that the throughput is measured in bits per second (bps), and we represent the total quantity of data transferred during a given time period as "D" (in bits), and the length of that time period as "T" (in seconds).

$$\text{Throughput} = D / T \quad (2)$$

*C. Average end to end delay:* A performance indicator called average end-to-end latency is used to determine how long it typically takes a data packet or message to travel from its origin to its intended destination inside a communication network. The specific characteristics of the network and the protocols used are what determine the average end-to-end delay. [16].

The average total time it takes for a message to go from the source to the destination in a straightforward scenario without any waiting or processing delays can be calculated by adding together the transmission time and the signal propagation time.

$$\text{Average End-to-End Delay} = \text{Transmission Delay} + \text{Propagation Delay} \quad (3)$$

The following equation can be used to calculate transmission delay, which is the time it takes for a packet or message to go across the network:

Transmission Delay = Packet Size / Transmission Rate (4)

Simply said, "Packet Size" refers to the quantity of data, expressed in bits or bytes, that is contained in a packet. Similar to this, "Transmission Rate" refers to the rate in bits per second or bytes per second at which data is transmitted across a network link.

The time it takes for a packet or message to get from its source to its final destination is referred to as propagation delay. It can be calculated using the following formula:

Propagation Delay = Distance / Propagation Speed (5)

Where Distance is the actual distance travelled from source to destination and Propagation Speed is the rate at which signals move across a medium. These two quantities are typically expressed in meters per second or kilometers per second.

## VI. EXPERIMENTAL SETUP

We describe the hardware setup for implementing our system as well as the simulation configuration in this section, as shown in Figure and Figure, respectively. As illustrated in figure 8, we deployed our IoT network with and without LZW compression using the NetSim simulator as our simulation setup. We used sensors, an Arduino board, and programmed it with and without LZW compression for the hardware implementation.

**Simulation Setup:**

Several parameters and configurations need to be taken into account while setting up a simulation for an underwater IoT network using the NetSim simulator. A potential simulation configuration is as follows:

1 **Network Size:** Calculate the subaquatic network's dimensions in accordance with your requirements. You might specify the network's node count or the area in square meters, for instance.

2 **Number of Nodes**: Establish the number of nodes in the network, which may vary depending on the size of the simulation and the desired level of complexity. Take into account factors like node density and distribution across the network. 4. Routing Protocols:

3 **Bandwidth:** Identify the amount of bandwidth the undersea network has available for communication. The maximum data transfer rate possible between nodes is determined by this number.

4 Choose the ones that are appropriate for communication underwater. Opportunistic Routing and Store-Carry-Forward (SCF) are two popular protocols for underwater networks. Depending on the details of your simulation scenario, select the relevant routing protocols.

5 **Packet Size:** Identify the size of the network packets being transmitted. This value has an impact on the bandwidth that is accessible as well as the effectiveness of data transport. When choosing the packet size, take into account the communication restrictions underwater.

6 **Packet Rates:** Specify how quickly packets are created and sent over the network. This parameter has an impact on network congestion and traffic load. To reflect actual conditions, set a reasonable packet creation rate.

7 **Simulation Time**: Calculate the length of the simulation in units of simulated time. We may watch the network behaviour evolve over time by setting this parameter, which also determines how long the simulation will run.

8 **Pause Time**: To imitate real-world settings, include pauses or idle times in between simulation actions. For a true representation of the behaviour of underwater IoT networks, you may, for instance, incorporate times of low activity or sporadic communication.

9 **Node Placement:** Specify where nodes should be located in the underwater network. Think about the possible deployment possibilities, such random placement or a particular pattern. Make sure the node distribution accurately reflects the situation in real life you are attempting to imitate.

We defined these parameters in table 1 and set up the NetSim simulator to construct and run our simulation in accordance with those settings. The network is simulated with and without compression using the NetSim simulator, and the results are shown in fig. Figure with and without compression depicts the hardware implementation of this network using a sensor, LCD display, Arduino board, and other components.

Table 1: Underwater IoT Network Simulation Parameters

| **Parameter** | **Value** |
|---|---|
| Network Size | 900*900m |
| Number of nodes | 50 to 300 |
| Bandwidth | 3 Mbps |
| Routing Protocols | M2M, MQTT, CoAP |
| Packet Size | 1024 bytes |
| Packet Rates | 7 Packets/sec |
| Simulation Time | 350-850 sec |
| Pause Time | 20 sec |
| Node Placement | Random |

*International Journal on Recent and Innovation Trends in Computing and Communication*
***ISSN: 2321-8169 Volume: 11 Issue: 9s***
***DOI: https://doi.org/10.17762/ijritcc.v11i9s.7425***
***Article Received**: 12 May 2023 **Revised:** 26 June 2023 **Accepted:** 25 July 2023*

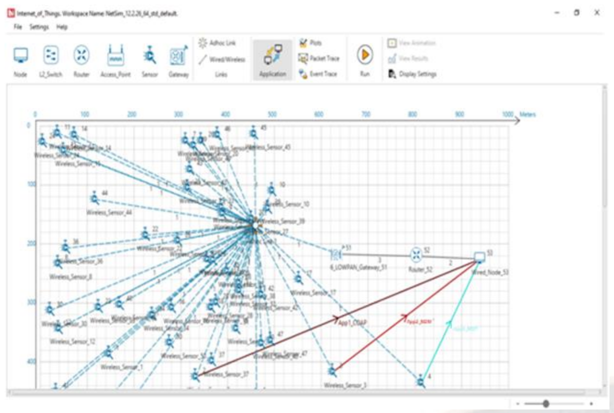
Figure 3. Underwater IoT Network Simulation

## Hardware Implementation

To build an IoT underwater hardware model consisting of a 5V power supply, DHT11 sensor, LDR sensor, IR sensor, and LCD display on both the sender and receiver ends, following components are used as shown in figure 4:

Components:

- Arduino board (e.g., Arduino Uno)
- 5V power supply or battery pack
- DHT11 temperature and humidity sensor
- LDR (Light Dependent Resistor) sensor
- IR (Infrared) sensor
- LCD display (compatible with Arduino)
- Jumper wires and breadboard for circuit connections

Steps:

1 Power Supply:
- Connect the 5V power supply or battery pack to the Arduino board.
- Make sure the power supply is suitable for underwater use and waterproofed.

2 DHT11 Sensor:
- Connect the DHT11 sensor to the Arduino board.
- Use jumper wires to connect the DHT11 sensor's data pin to a digital pin on the Arduino board.

3 LDR Sensor:
- Connect the LDR sensor to the Arduino board.
- Use jumper wires to connect the LDR sensor's output pin to another digital pin on the Arduino board.

4 IR Sensor:
- Connect the IR sensor to the Arduino board.
- Use jumper wires to connect the IR sensor's output pin to a digital pin on the Arduino board.

5 LCD Display:
- Connect the LCD display to the Arduino board.
- Follow the specific instructions for your LCD display to connect the necessary pins (typically, power, ground, data, and control pins).

6 Code:
- Write the Arduino code to read data from the DHT11, LDR, and IR sensors.
- Use appropriate libraries for each sensor and LCD display (e.g., DHT sensor library, LiquidCrystal library).
- The code should collect sensor data, display it on the LCD, and transmit it to the receiver.

7 Sender and Receiver Setup:
- Place the Arduino board, sensors, and LCD display for the sender.
- Repeat the same setup for the receiver, ensuring it has its placed properly.

8 Underwater Data Transmission:
- Establish a wireless communication method suitable for underwater use (e.g., acoustic communication, underwater radio, etc.).
- Implement the communication protocol and hardware necessary for data transmission between the sender and receiver.
- Modify the Arduino code to include the data transmission part, adhering to the specific communication protocol.

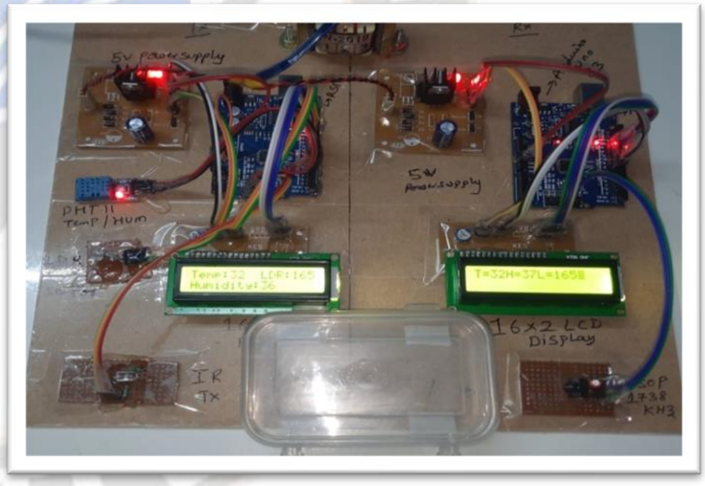

Figure 4. Hardware Model

## VII. RESULTS

### A. *IoT Network with and without Compression*

1 Routing Overhead:

The M2M, CoAP, and MQTT routing protocols were assessed for "routing overhead" in both the context of data compression and without data compression, as illustrated in Figures 5 and 6. The findings imply that regardless of data compression usage, routing overhead generally increases with node density.

When examining the routing overhead of each protocol independently, it was discovered that when data compression is

*International Journal on Recent and Innovation Trends in Computing and Communication*
***ISSN: 2321-8169 Volume: 11 Issue: 9s***
***DOI: https://doi.org/10.17762/ijritcc.v11i9s.7425***
***Article Received**: 12 May 2023 **Revised:** 26 June 2023 **Accepted:** 25 July 2023*

not used, the routing cost is larger than when it is. Here are some specific findings provided in the research paper:

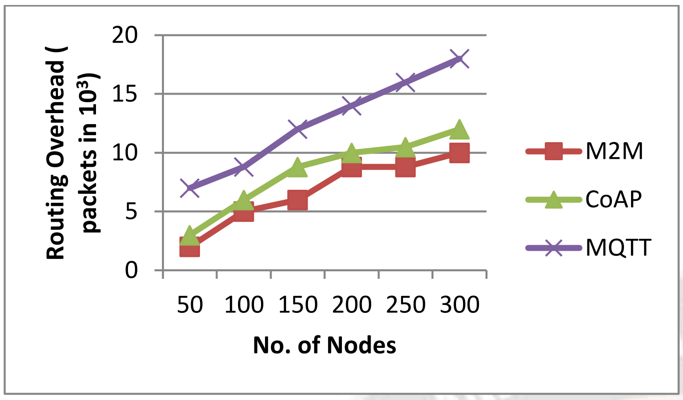

Figure 5. Routing Overhead without Data Compression

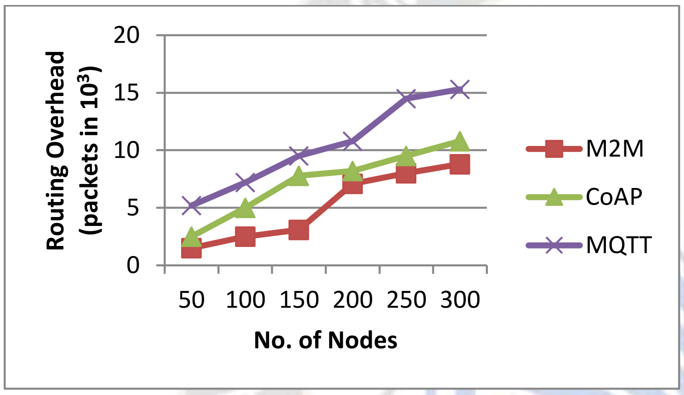

Figure 6. Routing Overhead with Data Compression

1 M2M Protocol:
- With 50 nodes:
  - Without data compression: 2 bps
  - With data compression: 1.5 bps

2 CoAP Protocol:
- With 150 nodes:
  - Without data compression: 8.8 bps
  - With data compression: 7.8 bps

3 MQTT Protocol:
- With 300 nodes:
  - Without data compression: 18 bps
  - With data compression: 15.3 bps

These findings suggest that data compression significantly lowers the routing overhead for each of the three protocols. In comparison to situations when data compression is not utilised, the routing overhead is smaller when data compression is implemented.

2 Throughput

Throughput is the rate at which messages or data are successfully transmitted across a communication channel or system. The throughput for three protocols—M2M, CoAP, and MQTT—is compared in Figures 7 and 8 for each protocol with and without data compression.

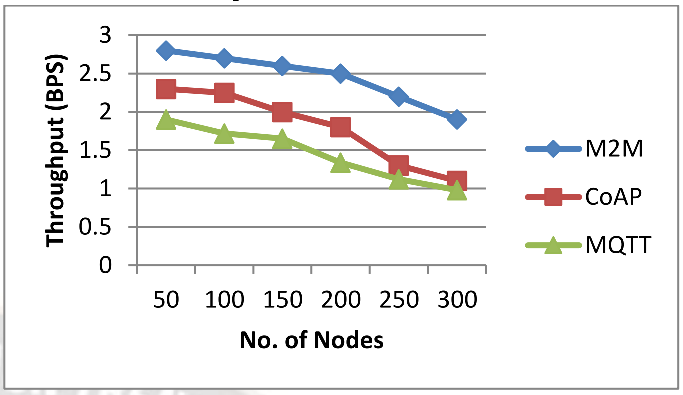

Figure 7. Throughput without Data Compression

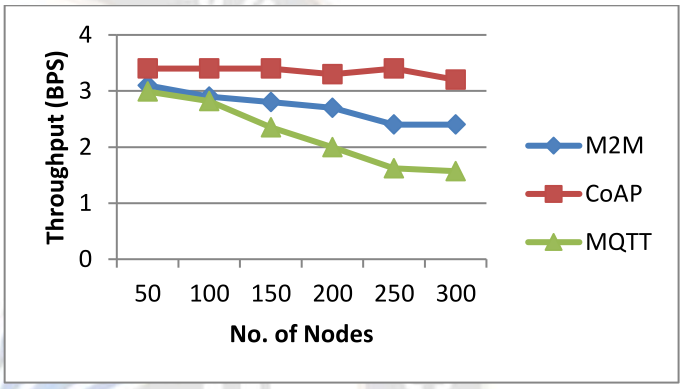

Figure 8. Throughput with Data Compression

1 M2M:
- For 50 nodes:
  - Without data compression: 2.98 bps
  - With data compression: 3.1235 bps

2 CoAP:
- For 150 nodes:
  - Without data compression: 2.155 bps
  - With data compression: 3.453 bps

3 MQTT:
- For 300 nodes:
  - Without data compression: 1.323 bps
  - With data compression: 0.195 bps

Based on the provided results, we can observe the following:

- For M2M, the throughput increases slightly from 2.98 bps to 3.1235 bps when data compression is enabled for 50 nodes.
- For CoAP, the throughput increases significantly from 2.155 bps to 3.453 bps when data compression is enabled for 150 nodes.

*International Journal on Recent and Innovation Trends in Computing and Communication*
***ISSN: 2321-8169 Volume: 11 Issue: 9s***
***DOI: https://doi.org/10.17762/ijritcc.v11i9s.7425***
***Article Received**: 12 May 2023 **Revised:** 26 June 2023 **Accepted:** 25 July 2023*

- For MQTT, the throughput decreases dramatically from 1.323 bps to 0.195 bps when data compression is enabled for 300 nodes.

Additionally, we can observe that MQTT has the lowest throughput among the three protocols.

3 Average End to End Delay

The performance of the M2M, CoAP, and MQTT routing protocols was evaluated in terms of the "end-to-end delay" parameter, taking into account the usage of data compression, based on the information shown in figures 9 and 10. The results show that there is an observed rise in the end-to-end delay as the number of packets increases from 50 nodes to 300 nodes. However, compared to situations when data compression is not used, the end-to-end delay decreases when data compression is used. This decrease in latency can be due to the smaller data packet sizes brought about by the use of data compression techniques.

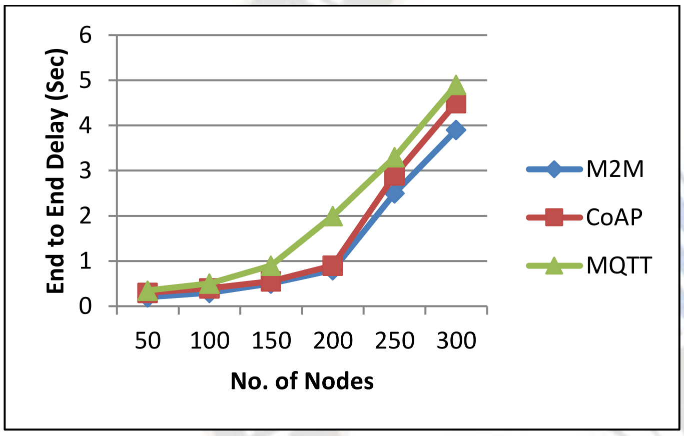

Figure 9. End to End delay without Data Compression

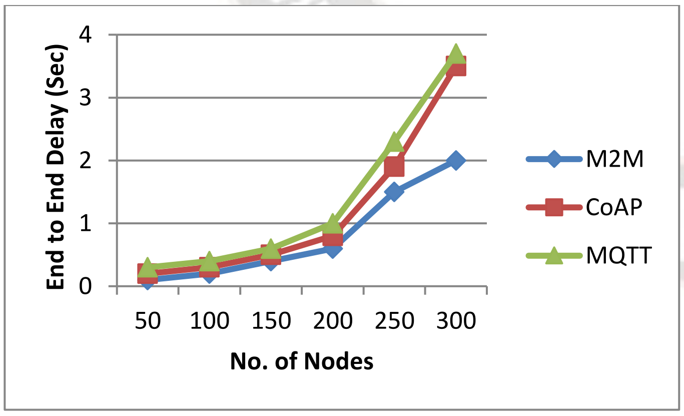

Figure 10. End to End delay with Data Compression

The research paper provides specific results to support this observation:

1. M2M Protocol:
   - With 200 nodes:
     - Without data compression: 0.8s
     - With data compression: 0.6s
2. CoAP Protocol:
   - With 250 nodes:
     - Without data compression: 2.95s
     - With data compression: 1.98s
3. MQTT Protocol:
   - With 300 nodes:
     - Without data compression: 4.9s
     - With data compression: 3.7s

These results lead to the conclusion that using data compression helps all three protocols' end-to-end latency be reduced. Compression reduces the size of data packets, which improves performance by cutting down on the time it takes to transmit and process the packets.

## VIII.CONCLUSION

In conclusion, this study used the Lempel-Ziv-Welch (LZW) data compression method to examine the performance of IoT networks operating in an acoustic environment. Throughput, routing overhead, and average end-to-end delay were taken into account as performance metrics. Additionally, three protocols—MQTT, CoAP, and machine-to-machine (M2M)—were assessed as part of the study.

The LZW data compression technique showed promising results in enhancing the functionality of IoT networks in the acoustic environment, according to the findings. By efficiently reducing the routing overhead through data compression, it improved the utilisation of resources and lessened network congestion. Throughput across the whole network was increased as a result of this decrease in routing overhead.

The average amount of time it takes to transmit data from the source to the destination was also significantly reduced as a result of the use of LZW compression. The compression technique decreased the time needed for data transmission by reducing the size of the communicated data, resulting in lower latency and faster connection between IoT devices.

In terms of protocols, this study assessed MQTT, CoAP, and M2M. Each protocol highlighted particular advantages and disadvantages. For IoT devices with limited resources, MQTT has shown to be a dependable and small-footprint protocol that offers effective message transport. On the other hand, CoAP showed outstanding support for low-power and lossy networks, allowing for effective resource discovery and communication. Direct device-to-device communication is the main goal of M2M, which increased connectivity and decreased reliance on centralised servers.

Overall, the study's findings highlight the significance of using efficient data compression methods, like LZW, in IoT networks that operate in acoustic settings. Compression techniques allow for significant improvements to be made in terms of data transfer speeds, routing burdens, and the average amount of time it takes for information to travel from its source to its destination. Additionally, choosing the right protocols while taking into account the unique needs and limitations of the network is essential for assuring optimal performance in IoT deployments.

## REFERENCES

[1] Darabkh, K. A., Amro, O. M., Al-Zubi, R. T., & Salameh, H. B. (2021). Yet efficient routing protocols for half-and full-duplex cognitive radio Ad-Hoc Networks over IoT environment. Journal of Network and Computer Applications, 173, 102836.

[2] Azar, J., Makhoul, A., Couturier, R., & Demerjian, J. (2020). Robust IoT time series classification with data compression and deep learning. Neurocomputing, 398, 222-234.

[3] Han, M., Duan, J., Khairy, S., & Cai, L. X. (2020). Enabling sustainable underwater IoT networks with energy harvesting: a decentralized reinforcement learning approach. IEEE Internet of Things Journal, 7(10), 9953-9964.

[4] Robinson, Y. H., Krishnan, R. S., Julie, E. G., Kumar, R., & Thong, P. H. (2019). Neighbor knowledge-based rebroadcast algorithm for minimizing the routing overhead in mobile ad-hoc networks. Ad Hoc Networks, 93, 101896.

[5] Meier-Kolthoff, J. P., & Göker, M. (2019). TYGS is an automated high-throughput platform for state-of-the-art genome-based taxonomy. Nature communications, 10(1), 2182.

[6] Venkataramanan, A. R. ., Kanimozhi, K. V. ., Valarmathia, K. ., Therasa, M. ., Hemalatha, S. ., Thangamani, M. ., & Gulati, K. . (2023). A Survey on Covid-19 & Its Impacts. International Journal of Intelligent Systems and Applications in Engineering, 11(3s), 129 –. Retrieved from https://ijisae.org/index.php/IJISAE/article/view/2550

[7] Gorbunova, A. V., Vishnevsky, V. M., & Larionov, A. A. (2020). Evaluation of the end-to-end delay of a multiphase queuing system using artificial neural networks. In Distributed Computer and Communication Networks: 23rd International Conference, DCCN 2020, Moscow, Russia, September 14–18, 2020, Revised Selected Papers 23 (pp. 631-642). Springer International Publishing.

[8] Dinculeană, D., & Cheng, X. (2019). Vulnerabilities and limitations of MQTT protocol used between IoT devices. Applied Sciences, 9(5), 848.

[9] Bansal, M., & Priya. (2021). Performance comparison of MQTT and CoAP protocols in different simulation environments. Inventive Communication and Computational Technologies: Proceedings of ICICCT 2020, 549-560.

[10] Railkar, P. N., Mahalle, P. N., & Shinde, G. R. (2021). Scalable Trust Management model for Machine To Machine communication in Internet of Things using Fuzzy approach. Turkish Journal of Computer and Mathematics Education (TURCOMAT), 12(6), 2483-2495.

[11] Qiu, T., Zhao, Z., Zhang, T., Chen, C., & Chen, C. P. (2019). Underwater Internet of Things in smart ocean: System architecture and open issues. IEEE transactions on industrial informatics, 16(7), 4297-4307.

[12] Safieh, M., & Freudenberger, J. (2019). Efficient VLSI architecture for the parallel dictionary LZW data compression algorithm. IET Circuits, Devices & Systems, 13(5), 576-583.

[13] Mohammadi, H., Ghaderzadeh, A., & Sheikh Ahmadi, A. (2022). A Novel Hybrid Medical Data Compression Using Huffman Coding and LZW in IoT. IETE Journal of Research, 1-15.

[14] Ms. Mohini Dadhe, Ms. Sneha Miskin. (2015). Optimized Wireless Stethoscope Using Butterworth Filter. International Journal of New Practices in Management and Engineering, 4(03), 01 - 05. Retrieved from http://ijnpme.org/index.php/IJNPME/article/view/37

[15] Khan, M. A., Khan, M. A., Jan, S. U., Ahmad, J., Jamal, S. S., Shah, A. A., ... & Buchanan, W. J. (2021). A deep learning-based intrusion detection system for mqtt enabled iot. Sensors, 21(21), 7016.

[16] Ahammad, D. S. K. H. (2022). Microarray Cancer Classification with Stacked Classifier in Machine Learning Integrated Grid L1-Regulated Feature Selection. Machine Learning Applications in Engineering Education and Management, 2(1), 01–10. Retrieved from http://yashikajournals.com/index.php/mlaeem/article/view/18

[17] Esposito, M., Kowalska, A., Hansen, A., Rodríguez, M., & Santos, M. Optimizing Resource Allocation in Engineering Management with Machine Learning. Kuwait Journal of Machine Learning, 1(2). Retrieved from http://kuwaitjournals.com/index.php/kjml/article/view/115

[18] Tyagi, D., Agrawal, R., & Singh, H. M. (2018, October). A survey on MAC layer protocols for machine to machine communication. In 2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN) (pp. 285-288). IEEE.

[19] Larmo, A., Ratilainen, A., & Saarinen, J. (2018). Impact of CoAP and MQTT on NB-IoT system performance. Sensors, 19(1), 7.

[20] Sisodia, A., & Kundu, S. (2019, November). Enrichment of performance of operation based routing protocols of WSN using data compression. In 2019 8th International Conference System Modeling and Advancement in Research Trends (SMART) (pp. 193-199). IEEE.

[21] Ahmed Ali, Anaïs Dupont,Deep Generative Models for Image Synthesis and Style Transfer , Machine Learning Applications Conference Proceedings, Vol 2 2022.

[22] Sisodia, A., & Yadav, A. K. (2022). Confabulation of Different IoT Approaches with and without Data Compression. Computer Integrated Manufacturing Systems, 28(11), 963-981.

[23] Sisodia, A., & Swati, H. H. (2020). Incorporation of non-fictional applications in wireless sensor networks. International Journal of Innovative Technology and Exploring Engineering (IJITEE), 9(11).

[24] Sisodia, A., Vishnoi, S., & Yadav, A. K. (2023). To Brace Society 5.0: Enhanced Reliability with a Cost-Effective Protocol for Underwater Wireless Sensor Network. In Sustainable

Computing: Transforming Industry 4.0 to Society 5.0 (pp. 171-185). Cham: Springer International Publishing.