Hindawi
Journal of Sensors
Volume 2019, Article ID 9808321, 12 pages
https://doi.org/10.1155/2019/9808321

Research Article

Efficient Payload Compression in IP-based Wireless Sensor Network: Algorithmic Analysis and Implementation

Md. Motaharul Islam[1], **Ngnamsie N. Soualihou**[2] and **Arham A. Siddiquee**[3]

[1]*Department of Computer Science and Engineering, BRAC University, Bangladesh*
[2]*Sun Moon University, Daegu, Republic of Korea*
[3]*Technology Division, Robi Axiata Ltd., Bangladesh*

Correspondence should be addressed to Md. Motaharul Islam; motaharul.islam@bracu.ac.bd

Received 29 July 2018; Accepted 5 August 2019; Published 28 December 2019

Academic Editor: Syed K. Islam

Due to its efficiency in end-to-end communication, wireless sensor network based on the Internet protocol (IP-WSN) is used for monitoring purposes. Nowadays, the concerned agencies are giving their highest priority to monitor its critical infrastructure. Infrastructure health monitoring is the measure of estimating the state of infrastructure health or detecting the changes in structures that affect its performance. The traditional approach to monitor the infrastructure health is done by using centralized data acquisition hub. Installation and commissioning of these systems represent significant concerns, thus moving toward IP-WSN. As cost effectiveness and energy efficiency are major concerns, our proposed approach is to reduce the amount of overhead while keeping the infrastructure health monitoring system accurate. Our contribution in this paper is to reduce the amount of data to be transmitted by compressing the payload of the packets. Thus, we have proposed a double compression algorithm. In this way, the capacity of the sensor node will be increased since less time will be taken to transmit data between the intermediate node as well as the coordinator node. As a consequence, it will also extend the lifetime of the battery.

1. Introduction

Wireless sensor network (WSN) consists of a varying number of tiny sensor nodes. Normally, it is distributed in a wide geographical area based on the nature of application. It forms an ad hoc network to collect and transmit data for the area under supervision. The data collected by these sensors is aggregated at a sink node for further analytics [1]. WSN combines short-range wireless communication, minimum computing facilities, and sensing of the physical environment which leads to a new paradigm of networks. It can be deeply embedded in the physical environment fueled by the low cost and wireless communication facilities. Monitoring the environment is one of the most important application areas of WSN.

Thus, infrastructure health monitoring (IHM) is becoming a popular area of research as it offers opportunities in construction management and maintenance. We can develop postdisaster scenarios and rescue support. IHM is a multidisciplinary field where a number of areas are involved such as computer science, electronics, and civil engineering [1–4]. The traditional approach for infrastructure health monitoring involves conventional piezoelectric accelerometers and other sensor nodes connected to data collection boards. High installation, equipment, and maintenance costs are some major drawbacks of the traditional system [1].

The WSN approach to IHM offers the same functionalities as the wired approach. On the other hand, the WSN approach offers various advantages over the wired approach such as installation and maintenance cost reduction [5]. Despite all the advantages of the wireless system, it still requires end-to-end communication for monitoring the infrastructure and a better compression algorithm for IP payload. Hence, we propose an efficient payload compression for infrastructure health monitoring in WSN.

Wireless sensor network based on the Internet protocol (IP-WSN) is gaining importance for different areas of applications which include healthcare, infrastructure monitoring,

environmental monitoring, industrial control, and precision agriculture [6–8]. End-to-end communication of IP-WSN is a promising tool for IHM, especially in critical infrastructure where human intervention is almost impossible. In this regard, IPv6-based low-power wireless personal area network (6LoWPAN) provides a unique opportunity [9].

The IP-WSN consisting of low-power devices has to communicate with other regular devices. 6LoWPAN was introduced for this purpose. In 6LoWPAN, only the header is compressed. To the best of our knowledge, there is no significant research in the field of payload compression in 6LoWPAN-based IP-WSN until now. To make the system more efficient in terms of power consumption and the amount of data sent, we propose IP payload compression using arithmetic compression algorithm. The payload in our proposed approach is compressed at two levels, sensor and sink node level.

The major contributions of this article are mentioned below:

(a) We propose an architecture for IHM which is based on IP-WSN

(b) We provide an algorithmic analysis for the payload compression for IP-WSN-based IHM

(c) For payload compression, we propose a double compression algorithm in sensor node and sink node

(d) We have carried out simulation work for the payload compression. The result proves the effectiveness of our compression technique based on arithmetic coding algorithm

(e) Algorithmic analysis and simulation are conducted to show the efficiency of the proposed approach

The remaining of the paper is organized as follows. Section 2 reviews background and related works. The proposed architecture is presented in Section 3. Section 4 describes the algorithmic analysis. Section 5 describes the proposed compression algorithms. Section 6 depicts the simulation environment. Section 7 shows the performance evaluations, and finally, Section 8 concludes the paper.

2. Background and Related Works

Data compression for wireless sensor network is an ongoing area of research. Recently, several reliable protocols have been proposed [10, 11]. WISDEN is one of the wireless sensor network (WSN) which utilizes a combination of hop-by-hop and end-to-end recovery system for reliability. It has utilized run length encoding scheme. To overcome the bandwidth constraints of WSN, this paper mentions the applicability of compression techniques based on wavelet. WISDEN has a simple architecture where a central base station collects data. The challenging aspect of its design is more complicated than other sensor networks built until today. Infrastructure monitoring generates data at a much higher rate than most sensing environments. Moreover, the transmission of loss intolerant data and time-synchronized reading from different sensors are required by the application [12]. Resource constraints of the existing sensor platform along with high packet loss and relatively lower radio bandwidth posed significant challenges to this system [2]. An especially designed vibration card for infrastructure application has been used in WISDEN. Along with the description of the card, WISDEN focuses on its three novel software components. (i) Reliable data transport: here, a routing tree is constructed using existing topology management techniques. Packet loss is recovered by a combination of hop-to-hop and end-to-end recovery system. (ii) Compression: periods of inactivity in infrastructure response are suppressed by a simple rub length encoding scheme. The feasibility of wavelet compression techniques for the reduction of data rate requirement and latency improvement is evaluated by it. (iii) Data synchronization: a data synchronization scheme is implemented that requires little overhead and does not require global clock synchronization [2].

On the main span and the southern tower of the Golden Gate Bridge (GGB), a WSN has been deployed and tested for IHM [1]. Ambient infrastructure vibrations are measured reliably at a low cost and without hampering the operation of the bridge. Ambient vibrations are collected synchronously at 1 kHz rate, with less than 10 μs jitter, and with an accuracy of 30 μG by the 64 nodes deployed across the main span and tower of the GGB. The data sample has been collected reliably over the deployed nodes, with a bandwidth of 441 B/s. Existing theoretical models and previous studies of the bridge support the sampled data. This deployment is the largest WSN for IHM. In this work, small packet size is a bottleneck for network data transmission and bandwidth. Large packet size is not a good solution for the Mica motes due to the limited amount of RAM. In article [3], WSN for infrastructure health monitoring has utilized the Huffman coding technique. In this work, sensor node has collected data and also processed the data package. This algorithm is implemented into the sensor node to reduce the packet size to be transmitted. In this work, an on-site WSN infrastructure health monitoring of Chung-Sha Bridge in Taiwan has been implemented. The monitoring system successfully achieved the frequency analysis of the bridge infrastructure with 128 Hz sampling rate from end nodes. A local data processing node with an ARM Cortex M3 processor has been developed. Based on this local data processing node, the Huffman compression algorithm has been implemented and examined. The experimental results show that the wireless transmission payload is reduced by 60% and the node number of the implemented network could be increased by 3 times [3].

Both of the works are concerned with the performance. We have also found a wireless sensor network in infrastructure health monitoring based on ZigBee technology [13]. ZigBee has been used because it is very popular, low cost, low-power wireless mesh networking standard. It is suitable for complex networks with large spatial extension [14], multihop networks, proprietary metering, and automation solutions. But one of the drawback is the interoperation between ZigBee and IP-WSN nodes since one uses proprietary protocol and the other uses the standard IP protocol [4]. Our approach is to use IP-WSN in which 6LoWPAN nodes

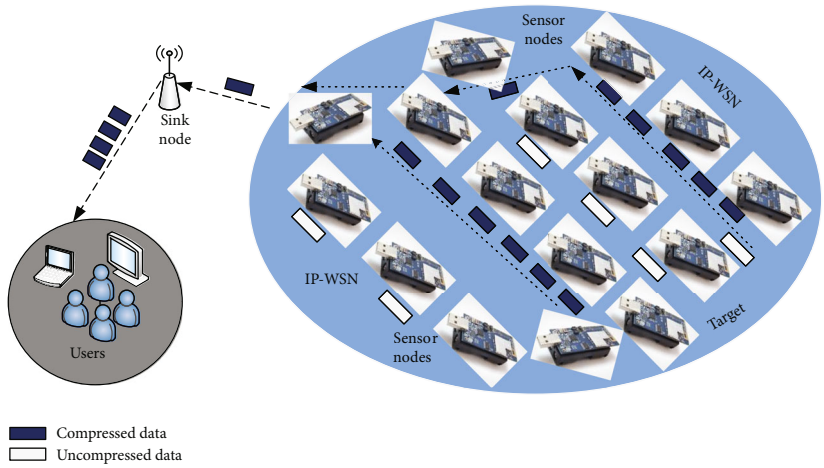

Figure 1: Proposed architecture of IHM.

are used. To accomplish this task, we have partially implemented it in our previous research work [15, 16]. For payload compression, we use arithmetic compression. 6LoWPAN defines an adaptation layer which allows the transportation of IPv6 packets over IEEE 802.15.4 links. 6LoWPAN reduces the IPv6 packets to fit within the MTU (127 bytes) of IEEE 802.15.4 frames. To do that, 6LoWPAN uses header compression and fragmentation and reassembly schemes only.

3. Proposed Architecture

IP-WSN consists of many sensor nodes where each of these nodes is very small in size. They have limited power, memory, and processing capacity. Each sensor consists of transceiver, power supply, processor, memory, and sensing circuitry. The overall architecture of our proposed IHM is shown in Figure 1. It depicts the compression mechanism at the node level as well as at the sink node level and storage of compressed data to the terminal; it clearly presents the system direction (specially the direction of data) that is from the sensor nodes through the sink node.

The sensor nodes are connected to one another in mesh topology. Each node compresses the data using the arithmetic compression algorithm and forwards it to the sink node. The compression of data in each node (i.e., local processing of data) compresses the data and thus saves power as well as increases the capacity of the nodes in the network such as the one used in [17] for power efficient management. The targeted nodes pass the compressed data to the sink node, and a further compression on that received data is performed again and then sent to the user's terminal. So, computation is done at the sink node more efficiently as the amount of data to be transferred is reduced by compression.

During the implementation, the sensor nodes are given IP addresses in order to uniquely identify them. This IP allocation allows us to specify the sink node, to which all the other nodes send the compressed data. This node is usually given a specific IP address because the specification represents the root node, and any IP addresses can be assigned to any other nodes by address autoconfiguration mechanism 6LoWPAN. Doing so will help the other nodes to locate the sink node and pass the data to it for further processing. The detail addressing scheme for the IP-WSN has been published in our previous research works [6, 7].

4. Algorithmic Analysis

4.1. Arithmetic Algorithm. In this algorithm, the symbols are not replaced by some code words. Instead, the symbols in the data are assigned values based on the mathematical models. The arithmetic algorithm can treat the whole symbols in a list or a data message to be transmitted as one unit. It does not use discrete number of bits or frequency distribution for the data symbols [18]. Each symbol is assigned an interval starting with the interval [0…1]. So, at the beginning the probabilities of occurrence of a set of symbols together with the cumulative probabilities are taken into consideration. These cumulative probabilities are used for encoding and decoding purposes. Figure 2 shows the probability assignment for the input symbols.

4.1.1. The Encoding Process. At the first step, we calculate the cumulative probabilities and then make ranges based on the obtained results. When we read a character, its range is considered as the new cumulative range on which our encoding will be done. The range is divided into the subparts, according to the probabilities of occurrence of the character being encoded. Then the next symbol is read, and this process of subpart formation is repeated as long as we have a character to encode in our source data. Once the end of the source data is found, we take a fraction of our subpart range formed. Therefore, using a fraction taken from our range, we can represent our entire source data into binary form. In [18], the encoded interval is [0.6607, 0.66303]. A sequence of bits is assigned to a number that is located in this range.

During this encoding process, two values are frequently calculated. The upper bound is obtained in equation (1)

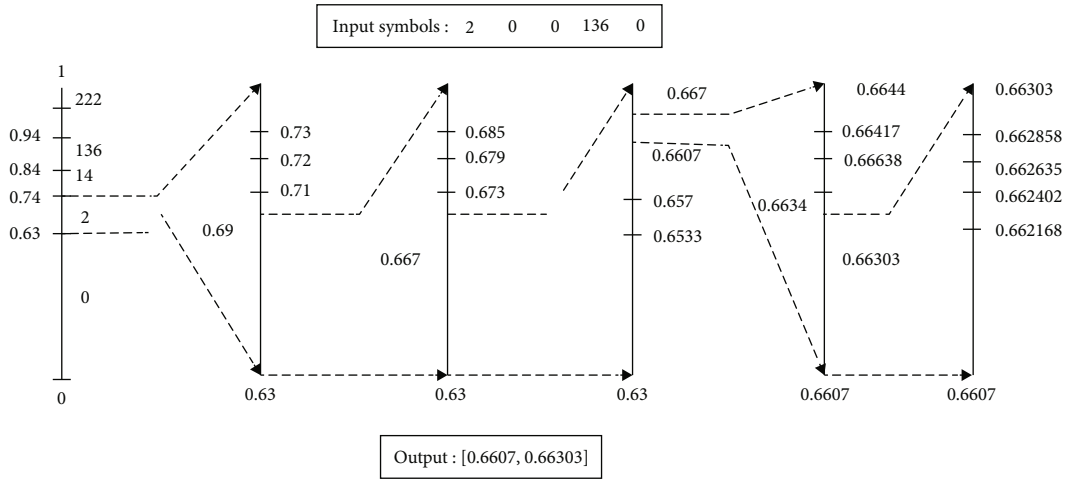

Figure 2: Encoding process.

Initialize l and u using the following.

$$l \leftarrow l + \left\lfloor \frac{(u - l + 1) * Cum\_count(x-1)}{TotalCount} \right\rfloor^{*}$$

$$u \leftarrow l + \left\lfloor \frac{(u - l + 1) * Cum\_count(x)}{TotalCount} \right\rfloor - 1^{**}$$

While (MSB of u and l are both equal to b or E_3 condition holds)
if (MSB of u and l are both equal to b)
{
Send b
Shift l to the left by 1 bit and shift 0 into LSB
Shift u to the left by 1 bit and shift 1 into LSB
While (Scale3>0)
{
Send complement of b
Decrement Scale3
}
}
if (E_3 condition holds)
{
Shift l to the left by 1 bit and shift 0 into LSB
Shift u to the left by 1 bit and shift 1 into LSB
Complement (new) MSB of l and u
Increment Scale3
}

Algorithm 1: Encoding in arithmetic algorithm.

and the lower bound obtained is shown in equation (2). The overall encoding process has been shown in Algorithm 1.

$$\text{Lower bound } L = \sum_{i=1}^{n} n^{n-i} C_i \prod_{k=1}^{i-1} f_k, \tag{1}$$

$$\text{Upper bound } U = L + \prod_{k=1}^{n} f_k. \tag{2}$$

We can summarize the encoding algorithm using the following pseudocode as mentioned in [18].

Figure 2 shows the compression of a set of data using the arithmetic algorithm and the corresponding process.

4.1.2. The Decoding Process. Once the compressed data reach the other end, it needs to be decompressed. This is done by following some mathematical model for applying the inverse process of the encoding. But to do that, the receiver needs to know the number of symbols that was sent as well as the probability l frequency distribution. We can summarize the

Initialize l and u.
Read the first m bits of the received bit stream into tag t.

$$k = 0$$

$$while\left(\left\lfloor \frac{(t-l+1)*TotalCount-1}{u-l+1} \right\rfloor \geq Cum\_Count(k)\right)$$

$$k \leftarrow k+1$$

decode symbol x

$$l \leftarrow l + \left\lfloor \frac{(u-l+1)*Cum\_count(x-1)}{TotalCount} \right\rfloor$$

$$u \leftarrow l + \left\lfloor \frac{(u-l+1)*Cum\_count(x)}{TotalCount} \right\rfloor - 1$$

while (MSB of u and l are both equal to b or E_3 condition holds)
  if (MSB of u and l are both equal to b)
    shift l to the left by 1 bit and shift 0 into LSB
    shift u to the left by 1 bit and shift 1 into LSB
    shift t to the left by 1 bit and read next bit from received bit stream into LSB
  }
  If (E_3 condition holds)
  {
    Shift l to the left by 1 bit and shift 0 into LSB
    Shift u to the left by 1 bit and shift 1 into LSB
    Shift t to the left by 1 bit and read next bit from received bit stream into LSB
    Complement (new) MSB of l, u, and t.
  }

Algorithm 2: Decoding in arithmetic algorithm.

```
Procedure Compressdata
        ThresholdValue ← maximum tolerated value during a natural disaster assigned to sensors
        → "if a sensed data exceeds this value then we must take that into consideration"
        waitValue ← amount of time to wait before sending the compressed value
        While TRUE do
                SensedData ← sensing the environment
                If sensedData is greater than ThresholdValue then
                        collectedData ← sensedData → "add the sensed data to the collection
                        of data that are greater than the threshold"
                end If
                If startSensingTime is greater or equal to waitValue
                        compressedData ← ArithmeticEncoding(collectedData)
                        CompressdataSink (compressedData)
                end If
        end while
end Procedure
```

Algorithm 3: Algorithm for compression at sensor node level.

decoding process with the following algorithm [18]. The overall decoding process has been shown in Algorithm 2.

4.2. Huffman Algorithm. The Huffman encoding algorithm works by replacing a symbol by a set of bits [19]. The set of bits are generated as follows: (i) at the beginning, there is a frequency operation that is performed to group identical symbols together and then order them from the symbol with the smaller frequency to the one with the largest frequency; (ii) then, there is a joint operation which is performed between the two smallest frequencies in order to generate a parent node; (iii) after that, the same process is again performed between the next two smallest frequencies including the parent generated. This process goes on until we have a root node and the tree is fully generated. Once the tree is generated, there is an assignment that is performed as follows: the left edges of the tree are assigned the bit 0 and the right edges will be assigned the bit 1. At the end, a table is

```
Procedure CompressdataSink(compressedDataFromSensors)
        sinkWaitingTimeValue ← amount of time to wait before sending the compressed value to
        the terminal
        collectSensorsCompressedData ← compressedDataFromSensors collect the data
        compressed by sensors nodes
        If startCollectingTime is greater or equal to sinkWaitingTimeValue
                compressedSinkData ← ArithmeticEncoding(collectSensorsCompressedData)
                sendToTerminal(compressedSinkData)
        end If
end Procedure
```

Algorithm 4: Algorithm description of compression at sink node level.

generated containing symbols and their corresponding code bits, where this table has to be known to both end device that are communicating. For the encoding process, to encode a symbol, we go down the tree and read the bits on the path from the root node to the desired symbol. And for the decoding process, we use the generated table to find the symbol corresponding to the code that we have received.

5. Proposed Compression Algorithms

Here, we have proposed two algorithms that support double compression for better efficiency. One compression is done at the sensor node level as depicted in Algorithm 3 and another at the sink node level as depicted in Algorithm 4. This will ensure better compression as well as better performance of the nodes in terms of power and memory constraints.

A threshold value is to be set which is to be considered while sensing the environment. Therefore, all the values below should be ignored by the sensors. Then, we set a *waitValue* too, which is the amount of time a sensor has to wait before transmitting data to the sink. At this time, we start sensing data to find if the value sensed (*SensedData*) is greater than the threshold value (*ThresholdValue*). We collect that value and store it in the variable containing the collection of values above the threshold. We also check if the time the sensors is intended to wait before transmitting the data has elapsed. If the response is positive, we pass that collection of values as argument to the *ArithmeticEncoding* method. Finally, it returns the compressed data (*compressedData*) which in turn is sent to the sink node by the use of the *CompressdataSink* method that takes the *compressedData* as argument.

Here, we start by setting a waiting time reference value (*sinkWaitingTimeValue*), which is the amount of time the sink node needs to wait before compressing the received value from the sink. Then, we store the data from sensors in a variable called *collectSensorsCompressedData*. Next, we check the time elapsed if the sink node is intended to wait before sending the data to the terminal. Then, we pass that collection of values received from the sensors to the arithmetic encoder to compress the data again. Finally, we send that double compressed data to the terminal by using *sendToTerminal* method with *compressedSinkData* as argument.

Table 1: Parameter values used in our simulation environment.

Parameters		Values
IP-WSN node number		25~120
Network coverage		120×120 M
Sensor node density		0.002~0.008
Transmission power		5.85×10^{-5} W
Sink node number		1-10
Transmission range		25 m
Packet size		2048 byte
Average compression ratio (bit/sample)	Arithmetic algorithm	7.3
	Huffman algorithm	5.3

6. Simulation Environment

Table 1 also shows the necessary parameters and corresponding values that are used in the simulation environment. From the table, we can see the number of sensor nodes used as well as the sink nodes needed. We can see the large difference between the average compression ratios of the two algorithms. During our experiment, we have decided to cover a moderate area.

We have used the following tools:

(i) Eclipse IDE: it is an IDE used to write and compile program

(ii) TinyOS: it is an embedded and component-based operating system. It provides platform for IP-WSN. It consists of the following:

(a) Scheduler (main)

(b) Application interface

(c) Application specific interface

(d) Component interface

(e) System component

(f) Hardware presentation/abstraction layer

Figure 3: Make file initialization.

```
#ifndef MOTETO_MOTE_H
#define MOTETO_MOTE_H


typedef nx_struct MotetoMoteMsg
{
    nx_uint16_t NodeId;  //this is the id of the node that is sending the data
    nx_uint8_t Data;

} MotetoMoteMsg_t;

enum
{
    AM_RADIO = 6   //am mean active message
};
#endif /* MOTETO_MOTE_H */
```

Figure 4: Definition of a structure type and an enum constant.

```
module MotetoMoteC
{

    uses   //general interfaces
    {
        interface Boot;
        interface Leds;
    }
    uses //UserButton related interfaces
    {
        interface Get<button_state_t>;
        interface Notify<button_state_t>;

    }
    uses   //Radio
    {
        interface Packet;  //interface that allows us to work with packets
        interface AMPacket;
        interface AMSend;   //allows you to send active message
        interface SplitControl as AMControl;   //allows you to do some basic extraction on data
        interface Receive;  // allows us to receive either in the serial port or in the radio

    }

}
implementation
```

Figure 5: Declaration of interfaces in our program.

(g) Mote device

(h) Sensor device

(iii) NesC: it is a component-based, event-driven programming language used to write programs that run on the TinyOS platform. NesC language is an extension of the C programming language

The implementation was done on the following files:

(i) The make file: the file consisted only of the rule that should be applied to the other file

(ii) A header file: the file where the main structure of our code was written

(iii) The module file: the file that contains all the interfaces as well as most of the source code

(iv) The configuration file: this file contains the code necessary to wire up all the components of our program together

The following are the program snippets of a few of our source code implementation. It is shown in the picture in Figures 3–7.

```
implementation
{
    //my global variables
    bool _radioBusy =FALSE;    //will store the status of the radio
    message_t _packet;
    event void Boot.booted()
    {
        call Notify.enable();
        call AMControl.start();   //start the radio chip when the sensor boot

    }

    event void Notify.notify(button_state_t val)
    {
        if(_radioBusy == FALSE)
        {
            //creating the packet
            MotetoMoteMsg_t* msg = call Packet.getPayload(& _packet, sizeof(MotetoMoteMsg_t));//creating a pack
            msg->NodeId = TOS_NODE_ID;
            msg->Data =(uint8_t) val;

            //sending the packet
            if(call AMSend.send(AM_BROADCAST_ADDR,  & _packet, sizeof(MotetoMoteMsg_t))==SUCCESS)
            {
                _radioBusy = TRUE;
            }
        }
    }


```

Figure 6: Partial implementation for collecting the data after being sensed.

```
configuration MotetoMoteAppC{
    //not doing anything for the time
}
implementation{
    components MotetoMoteC as App;  // Main module file
    components MainC; // boot interface
    components LedsC; // for led


    App.Boot -> MainC;
    App.Leds -> LedsC;

    components UserButtonC;
    App.Get -> UserButtonC;
    App.Notify -> UserButtonC;

    //Radio communication
    components ActiveMessageC;
    components new AMSenderC(AM_RADIO);
    components new AMReceiverC(AM_RADIO);

    App.Packet->AMSenderC;
    App.AMPacket->AMSenderC;
    App.AMSend->AMSenderC;
    App.AMControl->ActiveMessageC;
    App.Receive->AMReceiverC;
}
```

Figure 7: Declaration of the different components for our program.

7. Performance Evaluation

In performance evaluation, we have used the Huffman coding [3]. The performance is measured in terms of compression ratio, data size, memory consumption, and compressed data size.

Table 2 shows the sampled data used to draw Figure 8. The table contains the result of compression ratio for arithmetic and Huffman coding. We used the variable length packet size from 128 bytes to 2048 bytes.

In Figure 8, we have compared the compression ratio (CR) of the Huffman coding with that of the arithmetic

Table 2: Huffman and arithmetic compression ratio based on the size of data.

Packet size	Compression ratio (bit/sample)	
	Arithmetic	Huffman
128	4.65	4.38
256	5.4	4.78
512	6.55	5.27
1024	7.73	5.64
2048	12.02	6.37

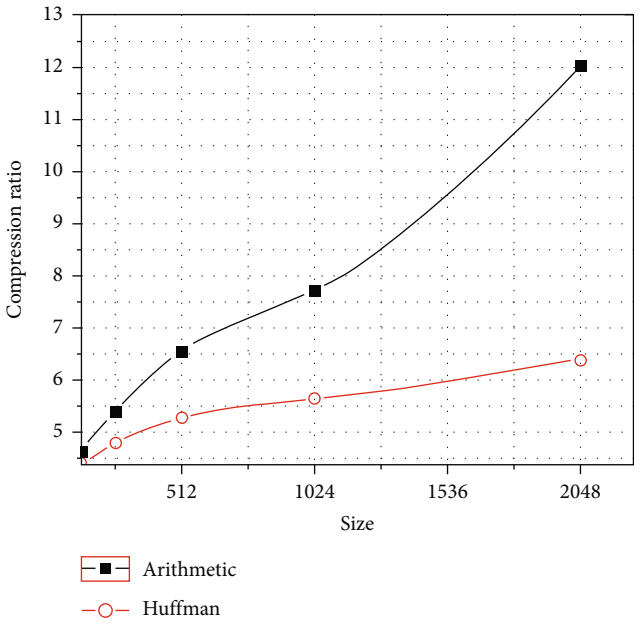

Figure 8: Arithmetic and Huffman comparison based on compression ratio.

coding. It is being observed that, with the increase of the size of the data, the compression ratio of the arithmetic coding is better than that of the Huffman coding. The compression ratio of the arithmetic coding is more than double of the Huffman coding as the data gets higher. On the other hand, the Huffman coding does not provide better compression with the increase of data size. We would like to mention that the data size in IHM is large and of variable length.

Figure 9 shows the amount of data that can be compressed when the same memory boundary is given. The figure depicts that the arithmetic coding can compress the whole data which is about 100%, provided to it at a specific time. The Huffman coding can only compress 60% of the data provided to it. And the run length encoding (RLE) can only compress 75% of the data provided to it. The node running these algorithms will run out of memory at a certain time. The compressibility of the arithmetic algorithm when provided with the same memory block is better than that of the Huffman coding and RLE.

Figure 10 gives us an idea about the memory utilization of Huffman, arithmetic, and RLE algorithms. From the figure,

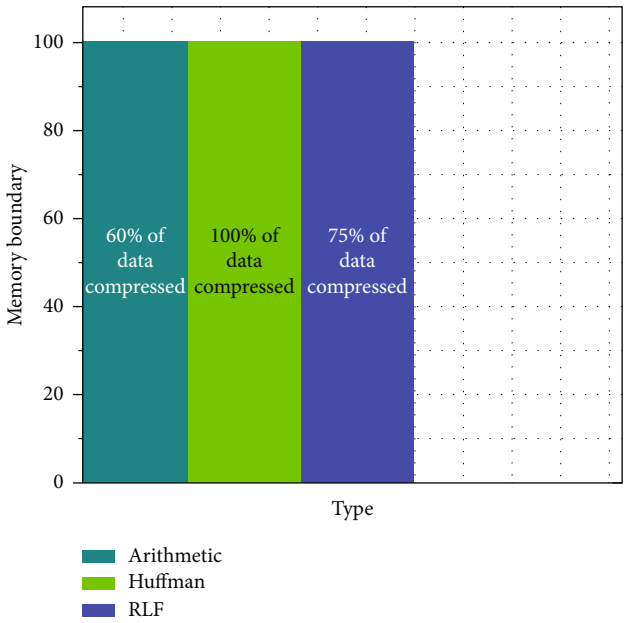

Figure 9: Data compression by three algorithms.

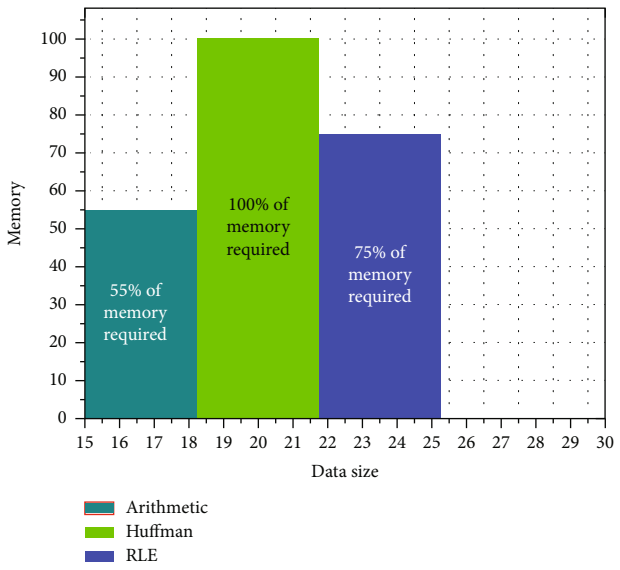

Figure 10: Comparison of the memory needed for computation.

we can see that the arithmetic coding algorithm uses less memory compared to the other two algorithms for their computations. So, in the case of memory utilization, the arithmetic coding is a much better choice for compression. As the sensor node's resources are very scarce, every bit of resource utilization is an important concern.

In Figure 11, in the case of the Huffman coding, the compression ratio gets better when there is more repetition of the values in the sensed data. But in the case of the arithmetic algorithm, there is not much change due to repetitive or non-repetitive data.

As in the case of IHM, the data we receive from the sensors are varied and there is very less repetition of data.

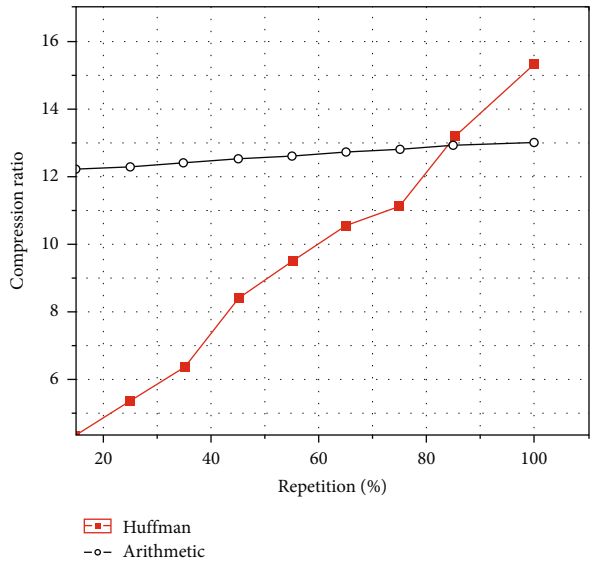

Figure 11: Comparison of the CR with respect to the repetition of data symbols.

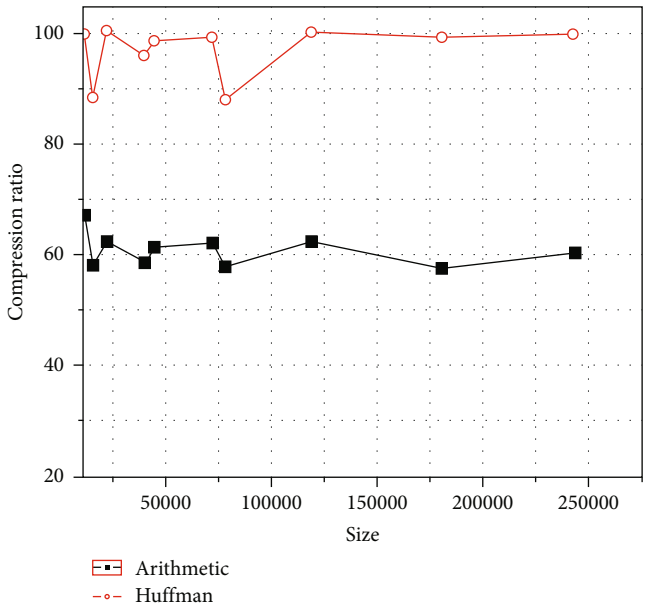

Figure 12: Comparison of the CR with respect to the packet size.

So, for this approach, the arithmetic algorithm gives a better compression ratio than the Huffman algorithm.

Figure 12 shows the comparison between the run length encoding and the Huffman encoding to represent compression ratio. As depicted in the figure, we see that the compression ratio of RLE is much higher than that of the Huffman encoding. On the other hand, the Huffman encoding is considered by many other researchers. This graph shows that the RLE is much better than the Huffman coding [20].

Table 3: Comparison between Huffman and arithmetic for repetitive data.

Repetition	Compression ratio	
	Huffman (bit/sample)	Arithmetic (bit/sample)
15	4.38	12.2
25	5.39	12.3
35	6.4	12.4
45	8.42	12.5
55	9.53	12.6
65	10.6	12.7
75	11.12	12.8
85	13.21	12.9
100	15.32	13

Table 3 represents data values for the compression ratio for both the Huffman coding and the arithmetic coding. These values are based on repetitive data. For repetitive data, the run length encoding shows better compression ratio than the Huffman coding.

Table 4 shows the comparison between the Huffman and arithmetic coding algorithms. It compares the two algorithms based on a few important factors. The most important one that we considered while conducting our research work was the compression ratio and the memory space. The compression ratio of the arithmetic algorithm is very good, while that of the Huffman is poor with respect to arithmetic coding. For the other factor such as memory space, we can see that the arithmetic algorithm requires a very low amount of memory space for its computation while the Huffman required more.

8. Conclusions

Among the application areas of IP-based wireless sensor network, infrastructure health monitoring (IHM) is a very promising field of research. IHM has many advantages, and it can be used to save both money and lives. By monitoring, renovating and maintaining these infrastructures when necessary can save a lot of money as well as keep the infrastructures fit. Besides, it can save lives which are lost due to natural disasters. Using IP-WSN technology in IHM can save lives and make the system less costly and affordable. The infrastructural health can be easily monitored, as well as structures in the remote areas can be monitored and can be easily maintained at a very low expense which will save lives in case of emergencies. One of the main disadvantages of the WSN node is the power constraint and limited memory. It is useful to mention that WSN node can be deployed in extreme environmental conditions and no human can go over to change a damaged node. Therefore, we need to make use of the node as efficiently and as long as possible. So, the data should be transferred efficiently between sensors. The compression algorithm and technique we proposed will increase the lifetime of the sensors as less data will be sent. Besides, the local processing will increase the lifetime as

Table 4: Comparative analysis of arithmetic and Huffman algorithm.

Comp. method	Comp. ratio	Comp. speed	Decomp. speed	Memory space	Comp. pattern matching	Permits random access
Arithmetic	Very good	Slow	Slow	Very low	No	No
Huffman	Poor	Fast	Fast	Low	Yes	Yes

well as the number of sensors deployed in the network. We have chosen the arithmetic algorithm over other types of algorithms because of the advantages that it offered on the compression ratio as well as the memory required for data processing.

Data Availability

The pseudocode and tools used to support the findings of this study are included within the supplementary information file.

Conflicts of Interest

The authors declare that there is no conflict of interest regarding the publication of this article.

Acknowledgments

This research has been carried out by individual research efforts and there is no financial gain through this article.

Supplementary Materials

An explanation of everything used or needed for this research work, from the pseudocode to the few snipped code of our implementation, has been organized as follows: Section 1 is all about our pseudocode; in Section 2, we have enumerated the tools used in our work; Section 3 shows the installation process of those different tools used in our research; Section 4 shows the different sensors used for the structural health monitoring we used; and finally, Section 5 displays few snipped codes of our implementation. *(Supplementary Materials)*

References

[1] S. Kim, S. Pakzad, D. Culler et al., "Health monitoring of civil infrastructures using wireless sensor networks," in *Information Processing in Sensor Networks, IPSN-2007. 6th International Symposium on. IEEE*, ACM, New York, NY, USA, 2007.

[2] N. Xu, S. Rangwala, K. K. Chintalapudi et al., "A wireless sensor network for structural monitoring," in *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, New York, NY, USA, November 03-05, 2004.

[3] C.-H. Hsu, C.-T. Lin, H.-P. Tserng, and J.-Y. Han, "An implementation of light-weight compression algorithm for wireless sensor network technology in structure health monitoring," *2014 IEEE World Forum on Internet of Things (WF-IoT)*, 2014, pp. 548–552, Seoul, South Korea, 6-8 March 2014.

[4] X. D. Jiang, Y. L. Tang, and Y. Lei, "Wireless sensor networks in structural health monitoring based on zigbee technology," *2009 3rd International Conference on Anti-counterfeiting, Security, and Identification in Communication*, 2009, pp. 449–452, Hong Kong, China, 20-22 Aug. 2009.

[5] D. Balsamo, G. Paci, L. Benini, and B. Davide, "Long term low cost passive environmental monitoring of heritage buildings for energy efficiency retrofitting," *2013 IEEE Workshop on Environmental Energy and Structural Monitoring Systems*, 2013, pp. 1–6, Trento, Italy, 11-12 Sept. 2013.

[6] M. M. Islam and E.-N. Huh, "Sensor Proxy Mobile IPv6 (SPMIPv6)-a novel scheme for mobility supported IP-WSNs," *Sensors*, vol. 11, no. 2, pp. 1865–1887, 2011.

[7] M. M. Islam and E.-N. Huh, "A novel addressing scheme for PMIPv6 based global IP-WSNs," *Sensors*, vol. 11, no. 9, pp. 8430–8455, 2011.

[8] M. M. Islam, M. A. Al Wadud, and E.-N. Huh, "Energy-efficient multilayer routing protocol for SPMIPv6-based IP-WSN," *International Journal of Sensor Networks*, vol. 18, no. 3/4, pp. 114–129, 2015.

[9] C. Bormann, "6LoWPAN-GHC: Generic Header Compression for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)," *Internet Engineering Task Force (IETF), RFC*, vol. 7400, pp. 1–24, 2014.

[10] F. Stann and J. Heidemann, "Rmst: reliable data transport in sensor networks," in *Proceedings of the First International Workshop on Sensor Net Protocols and Applications*, pp. 102–112, Anchorage, Alaska, USA, 11-11 May 2003.

[11] C.-Y. Wan, A. T. Campbell, and L. Krishnamurthy, "A reliable transport protocol for wireless sensor networks," in *In Proceeding of First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA 2002)*, pp. 1–11, Atlanta, September 2002.

[12] J. Elson and D. Estrin, "Time synchronization for wireless sensor networks," in *Proceedings of the 15th International Parallel & Distributed Processing Symposium*, p. 186, Washington, DC, USA, 2001.

[13] H. Zhi-gang and C. Cai-hui, "The application of Zigbee based wireless sensor network and GIS in the air pollution monitoring," *International Conference on Environmental Science and Information Application Technology (ESIAT 2009)*, vol. 2, 2009, pp. 546–549, Wuhan, China, 4-5 July 2009.

[14] C. Intanagonwiwat, R. Govindan, and D. And Estrin, "Directed diffusion: a scalable and robust communication paradigm for sensor networks," in *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking*, pp. 56–67, Boston, MA, August 2000.

[15] M. M. Islam, S. Ngnamsie, and S. Faizullah, "Structural health monitoring by payload compression in wireless sensors network: an algorithmic analysis," *International Journal of Engineering & Management Research*, vol. 8, no. 3, 2018.

[16] April 2019, http://103.82.172.44:8080/xmlui/bitstream/handle/123456789/100/EnhancedStructuralHealthMonitoring.pdf?sequence=1&isAllowed=y.

[17] J. P. Lynch, A. Sundararajan, K. H. Law, A. S. Kiremidjian, and E. Carryer, "Power-efficient data management for a wireless

structural monitoring system," in *In Proceedings of the 4th International Workshop on Structural Health Monitoring*, vol. 1, Stanford, CA, September 15-17 2003.

[18] G. G. Langdon, "An introduction to arithmetic coding," *IBM Journal of Research and Development*, vol. 28, no. 2, pp. 135–149, 1984.

[19] A. Shahbahrami, R. Bahrampour, M. S. Rostami, and Mostafa Ayoubi, "Evaluation of Huffman and arithmetic algorithms for multimedia compression standards," *International Journal of Computer Science, Engineering and Applications*, vol. 1, no. 4, pp. 34–47, 2011.

[20] S. R. Kodituwakku and U. S. Amarasinghe, "Comparison of lossless data compression algorithms for text data," *Indian Journal of Computer Science and Engineering*, vol. 1.4, pp. 416–425, 2010.