

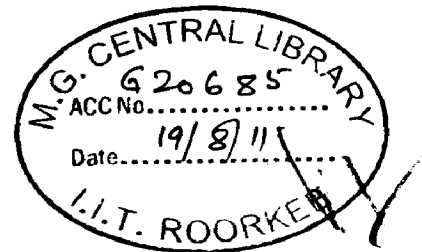
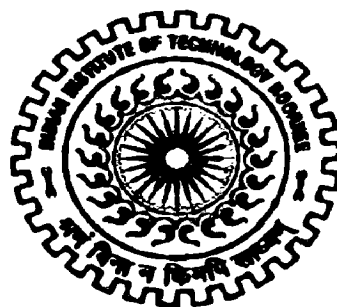
DATA COMPRESSION ALGORITHM FOR WIRELESS SENSOR NETWORKS

A DISSERTATION

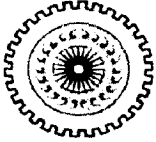
*Submitted in partial fulfillment of the
requirements for the award of the degree
of*
MASTER OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING

By

ASHISH.KUMAR MAURYA



**DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE -247 667 (INDIA)
JUNE, 2011**



INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE – 247 667, INDIA

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the dissertation entitled **“DATA COMPRESSION ALGORITHM FOR WIRELESS SENSOR NETWORKS”** in partial fulfillment of the requirement for the award of the degree of **Master in Computer Science and Engineering**, submitted to **Department of Electronics and Computer Engineering of Indian Institute of Technology Roorkee, Roorkee, India** is an authentic record of my own work carried out during the period of June 2010 to June 2011 under the supervision of **Dr. A. K. Sarje, Professor, Department of Electronics and Computer Engineering of Indian Institute of Technology Roorkee, Roorkee, India.**

The matter being presented in the Dissertation has not been submitted by me for the award of any other degree or diploma of this or any other Institute/University.

Date: 21-06-2011

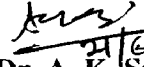
Place: Roorkee


(ASHISH KUMAR MAURYA)

CERTIFICATE

This is to certify that the above statement made by the candidate is true to the best of my knowledge and belief.

Date:


21/6/2011
(Dr. A. K. Sarje)
Supervisor

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to my learned mentor, Dr. A. K. Sarje, Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, for his encouragement, able guidance, painstaking supervision and innovative suggestions. His capability to judge things beyond the written text has helped this dissertation take its present form. He has been instrumental in shaping my approach towards accomplishing the task at hand and I feel privileged to have completed my dissertation under his supervision.

I wish to convey my sincere thanks to all my classmates and friends. The time spent with them is unforgettable and has touched all aspects of my life. The late night interesting discussions has helped me in think out of the box for all the problems that I faced. Each moment spent together is memorable.

Above all, I have no words to express my love for my family. I could not have reached this important milestone in my life without their constant support and encouragement. I take this opportunity to thank my parents for their unconditional love and motivation that shaped the confident individual in me.

I am thankful to the almighty for showering his grace that provided the courage and perseverance to overcome all obstacles that stood in my way.

ASHISH KUMAR MAURYA

ABSTRACT

Large scale wireless sensor networks (WSNs) have emerged as the latest trend in revolutionizing the paradigm of collecting and processing data in diverse environments. Its advancement is fueled by development of tiny low cost sensor nodes (SNs) which are capable of sensing, processing and transmitting data. Due to the small size of SNs there are various resource constraints. It is the severe energy constraints and the limited computing resources that present the major challenge in converting the vision of WSNs to reality.

Power saving is a critical issue in wireless sensor networks (WSNs) since sensor nodes are powered by batteries which cannot be generally changed or recharged. As radio communication is often the main cause of energy consumption, extension of sensor node lifetime is generally achieved by reducing transmissions/receptions of data. It is useful to apply data compression to reduce the volume of data, and the associated energy consumption of transmission. Data compression is the process of encoding information using fewer bits than an unencoded representation would use, through use of specific encoding schemes. Due to limited processing and storage resources of sensor nodes, data compression in sensor nodes requires the use of ad-hoc algorithms.

In this dissertation, we propose a simple and efficient data compression algorithm which is lossless and particularly suited to the reduced memory and computational resources of a WSN node. The proposed data compression algorithm gives good compression ratio for highly correlated data. Simulations for the proposed data compression algorithm are performed on TOSSIM. Some experimental results and comparisons with the Marcelloni et al. data compression algorithm is shown and discussed.

Table of Contents

Candidate's Declaration	i
Certificate.....	i
Acknowledgement	ii
Abstract	iii
Table of Contents.....	iv
List of Figures.....	vi
List of Tables.....	vii
Chapter 1 Introduction	1
1.1 Overview.....	1
1.2 Motivation.....	3
1.3 Problem Statement.....	5
1.4 Thesis Organization.....	6
Chapter 2 Background and Literature Survey	7
2.1 Introduction to Wireless Sensor Network.....	7
2.1.1 Wireless Sensor Network Model.....	7
2.1.2 Applications of Wireless Sensor Networks.....	9
2.1.3 WSNs versus MANETs.....	11
2.2 Data Compression.....	12
2.2.1 Compression Techniques.....	12
2.2.2 Modeling and Coding.....	14
2.3 Data Compression Schemes for WSNs.....	17
2.3.1 Distributed Compression.....	17
2.3.2 Coding by Ordering.....	19
2.3.3 Pipelined In-Network Compression.....	21
2.4 Shortcomings and Research Gaps.....	25

Chapter 3 Proposed Data Compression Algorithm	27
3.1 Introduction.....	27
3.2 Model of Proposed data compression algorithm.....	27
3.3 The Compression Algorithm.....	28
3.4 The Decompression Algorithm.....	33
Chapter 4 Experimental Results and Discussions	39
4.1 Performance Metrics.....	39
4.2 Simulation Environment.....	40
4.3 Results and Discussions.....	41
Chapter 5 Conclusions	45
5.1 Conclusions.....	45
5.2 Scope for future work.....	45
References.....	46
List of Publications.....	50
Appendix A Introduction to TinyOS.....	51

List of Figures

Figure 1.1	Possible deployment of a WSN for precision agriculture.....	2
Figure 1.2	Functional block diagram of a sensor node.....	2
Figure 2.1	Sensor network communication structure.....	7
Figure 2.2	Overview of sensor applications.....	10
Figure 2.3	Compression and reconstruction.....	13
Figure 2.4	Distributed Compression examples.....	17
Figure 2.5	Data path in coding by ordering data compression scheme.....	19
Figure 2.6	In-network Compression.....	22
Figure 3.1	Model of Proposed Compression Algorithm.....	28
Figure 3.2	Pseudo-code of the proposed encode algorithm.....	30
Figure 3.3	Flow chart of proposed data compression algorithm.....	32
Figure 3.4	Pseudo-code of the decode algorithm.....	35
Figure 3.5	Flow chart of decompression algorithm.....	37
Figure 4.1	Comparing the performance of algorithms on metric compression ratio.....	43
Figure 4.2	Comparing the performance of algorithms on metric saving percentage.....	43

List of Tables

Table 2.1	Permutation and its represented integer value.....	20
Table 4.1	Results of Mercelloni's et al. Algorithm.....	42
Table 4.2	Results of Proposed Data Compression Algorithm.....	42

Introduction

1.1 Overview

Wireless Sensor Network (WSN) consists of spatially distributed self-organizing, low-powered sensing devices with limited computational and communication resources to cooperatively monitor conditions, such as temperature, sound, vibration, pressure and humidity over a specific area for some specific purposes like target tracking, area monitoring, industrial monitoring, health monitoring, surveillance, environmental monitoring etc and report the collected data of all sensors to the user for analysis. In a typical application, a WSN is scattered in a region where it is meant to collect data through its sensor nodes. Figure 1.1 shows the possible deployment of a WSN for precision agriculture. In this figure, sensors detect temperature, light levels and soil moisture at hundreds of points across a field and communicate their data over a multi-hop network for analysis. Instead of the conventional methods, WSN deploys a large number of small nodes which gather data to be interpreted in a distributed manner. For ease of deployment, sensor devices should be inexpensive and have long lifetime. It is important to design protocols, software and hardware solutions to make the most efficient use of the limited resources of energy, computation and storage in a sensor node [1].

Each sensor node is a tiny device that includes three basic components (figure 1.2): a sensing subsystem for data acquisition from the physical surrounding environment, a processing subsystem for local data processing and storage, and a wireless communication subsystem for data transmission to a central collection point (sink node or base station). In addition, a power source supplies the energy needed by the device to perform the programmed task. This power source often consists of a

battery with a limited energy budget. In addition, it could be impossible or inconvenient to recharge the battery, because nodes may be deployed in a hostile or unpractical environment. On the other hand, the sensor network should have a lifetime long enough to fulfill the application requirements.



Figure 1.1 Possible deployment of a WSN for precision agriculture. [4]

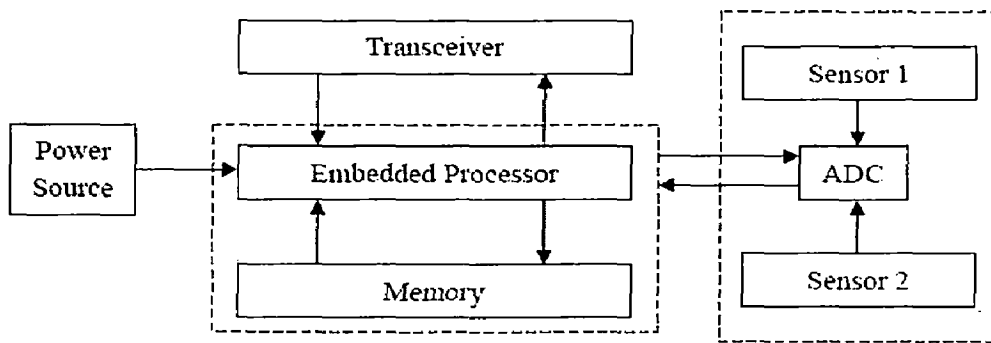


Figure 1.2 Functional block diagram of a sensor node [3]

Experimental measurements have shown that data transmission is very expensive in terms of energy consumption, while data processing consumes significantly less energy. The energy cost of transmitting a single bit of information is approximately

the same as that needed for processing a thousand operations in a typical sensor node [2]. The energy consumption of the sensing subsystem depends on the specific sensor type. In many cases it is negligible with respect to the energy consumed by the processing and, above all, the communication subsystems. In other cases, the energy expenditure for data sensing may be comparable to, or even greater than, the energy needed for data transmission.

Several energy conservation schemes have been proposed in the literature. They are mainly aimed at minimizing the energy consumption of the communication subsystem. With regard to this, there are two main approaches to energy conservation: duty cycling and in-network processing. Duty cycling schemes define coordinated sleep/wakeup schedules among nodes in the network. On the other hand, in-network processing consists in reducing the amount of data to be transmitted by means of compression and/or aggregation techniques [2]. Due to limited processing and storage resources of sensor nodes, data compression in sensor nodes requires the use of ad-hoc algorithms. Only a few researchers have discussed the possibility of embedding lossless compression algorithms into sensor nodes.

Obviously, compressing data can be a valuable help in power saving only if the execution of compression algorithms does not require an amount of energy greater than the one saved in reducing transmission: in [5] it is shown that compression prior to transmission in wireless battery-powered devices may actually cause an overall increase of power consumption, if no energy awareness is introduced, because compression algorithms are aimed at saving storage and not energy.

1.2 Motivation

WSNs receive a lot attention due to their unlimited potential. The motivation for the study comes from the unique challenges offered in the varied application domain. However, the following key issues need to be addressed while designing the sensor network [1]:

- **Real World Protocols:** Current WSN solutions are developed with simplifying assumptions about wireless communication and the environment, even though the realities are well known to be different. Many of these solutions work very well in simulation but in the real world they can be shown to work poorly in practice. Thus, there is a need to establish better models of communication realities to feedback into simulation tools.
- **Real-Time Data Delivery:** In many cases, sensor data must be delivered within time constraints so that appropriate observations can be made or actions taken. Few results exist to date that meet real-time requirements of WSN. Most protocols either ignore real time or attempt to process as fast as possible and hoping that the speed is sufficient to meet deadlines.
- **Limited Energy:** The energy constraint in sensor nodes is unlikely to be solved soon due to slow progress in developing battery capacity. Moreover, the untended nature of sensor nodes and hazardous sensing environments preclude battery replacement as a feasible solution. The surveillance nature of many sensor network applications requires a long lifetime. Current research focuses on providing full sensing coverage in the context of energy conservation.
- **Fault Tolerance:** Fault tolerance is the ability to sustain sensor network functionalities without any interruption due to sensor node failures. In most of the scenarios once deployed the nodes work on their own for re-configuration, routing setup, re-clustering etc. The fault tolerance level depends on the application of the WSN and is done by estimating initial density of nodes required, and designing fault tolerant protocols at different network layers.
- **Scalability:** The number of nodes can vary from a few sensor nodes to a few hundred or even more. New nodes may be added intermittently or the same type of nodes can be used for different application areas. Hence, any solution must scale up with varying number of nodes without affecting the quality of service.

- **Environment Interaction:** The nodes should be designed in a robust manner so as to withstand the harshest of environment like the bottom of ocean, battlefield, and fast moving vehicles and still deliver the unpredictable data as generated. WSNs are likely to exhibit low data rates for a large period of time and have very bursty traffic when an event occurs. It is unlikely that there will be generic solutions and application specific protocols are designed.
- **Handling large volume of bursty traffic:** WSN comprises of highly data intensive networks where hundreds and thousands of nodes generate data continuously. Since, nodes have limited storage and processing power it is imperative that the data is not lost due to limited memory and also to save energy by reducing redundant and spurious data on the network.

1.3 Problem Statement

The main objective of the present research work can be described by the statement of the problem expressed as follows:

“To develop a simple and efficient data compression algorithm for wireless sensor network that is lossless and particularly suited to the reduced memory and computational resources of a WSN node”.

To achieve the above objective of reducing memory requirement and WSN energy consumption following smaller objectives are set:

- To design preprocessor module.
- To reduce the total amount of data by building variable length codes.
- To reduce the total amount of data being transmitted on the network without losing the originality of data.
- To achieve better compression ratio for proposed data compression algorithm than previous data compression algorithms exists

As stated earlier, data communication is the most power consuming operation of the sensor nodes. Therefore the intention is to reduce memory and save energy by cutting down on the data being transmitted.

To achieve the above objectives the following design goals are set:

- Proposing a data compression algorithm to reduce the size of data by removing the redundancy in the data such that minimum numbers of bits have been transferred from originator node of data to the base station.
- Proposing a decompression algorithm to retain the originality of data at base station.

1.4 Thesis Organization

This dissertation report comprises of five chapters including this chapter that introduces the topic and states the problem. The rest of the report is organized as follows:

Chapter 2 details the fundamentals and provides a literature review of the various data compression algorithm and techniques used in wireless sensor networks. Research gaps and shortcomings are identified and described.

Chapter 3 describes the proposed data compression algorithm which reduces memory and saves energy by reducing the original size of data. This chapter also describes the decompression algorithm which decompresses the compressed data.

Chapter 4 discusses the implementation details and provides the experimental results of the proposed algorithms. In this chapter performance of the proposed algorithm is also compared with the existing one.

Chapter 5 concludes the dissertation work and gives suggestions for future work.

Background and Literature Survey

2.1 Introduction to Wireless Sensor Network

2.1.1 Wireless Sensor Network Model

The major components of a typical sensor network are shown in figure 2.1. They are: sensor nodes, the sensor field, the sink and the task manager.

A *sensor field* can be considered as the area in which the nodes are placed i.e. the area in which a particular phenomenon to occur.

Sensors nodes or notes are the heart of the network. They are in charge of collecting data and routing this information back to a sink.

A *sink* is a sensor node with the specific task of receiving, processing and storing data from the other sensor nodes. They serve to reduce the total number of messages that need to be sent, hence reducing the overall energy requirements of the network.

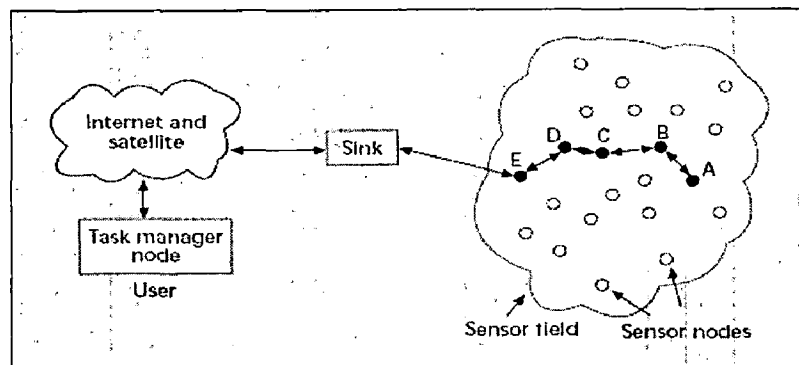


Figure 2.1 Sensor network communication structure

The *task manager or base station* is centralized point of control within the wireless sensor network, which extracts information from the network and

disseminates control information back into the network. It also serves as a gateway to other networks, a powerful data processing/storage centre and an access point for a human interface. Hardware wise the base station is either a laptop or a workstation. Data is streamed to these workstations either via the internet, wireless channels, satellite etc [1].

Basic features of sensor networks are [6]:

- Self-organizing capabilities
- Short-range broadcast communication and multi-hop routing
- Dense deployment and cooperative effort of sensor nodes
- Frequently changing topology due to fading and node failures
- Limitations in energy, transmit power, memory and computing power

Some of the advantages of wireless sensor networks over wired one are as follows [3]:

- **Ease of deployment:** These wireless sensor networks can be deployed at the site of interest without any prior organization, thus reducing the cost and time and also increasing the flexibility of organization.
- **Extended range:** One huge wired macro-sensor can be replaced by many smaller wireless sensors for the same cost. Such macro-sensor can sense only a limited region whereas network of smaller sensors can be distributed over a wider range.
- **Fault tolerant:** With macro-sensors failure of one node makes that area completely unmonitored. With wireless sensors, failure of one node does not affect the operation substantially. At most accuracy of data collected may be somewhat reduced.
- **Mobility:** If a region becomes unmonitored we can have the nodes rearrange the nodes themselves to distribute the node evenly (e.g. if placed at nodes),

i.e., these nodes can be made to move towards area of interest but mobility is lower compared to MANETs.

2.1.2 Application of Wireless Sensor Networks

Wireless Sensor Networks have changed the interface of information retrieval from the physical world. In a typical application, a WSN is scattered in a region where it is meant to collect data through its sensor nodes. Instead of the conventional methods, WSN deploys a large number of small nodes which gather data to be interpreted in a distributed manner [6, 7, 8, 9]. The application domains are:

- **Military:** Military could use sensor networks for battlefield surveillance; monitor vehicular traffic; track the position of the enemy; safeguard the equipment of the side deploying sensors; damage assessment and attack detection (chemical, biological).
- **Industrial Application:** Monitoring and control of industrial equipment; factory process control and industrial automation; wastewater monitoring; landfill ground well level monitoring.
- **Home Application:** Home automation can be brought about by creating a smart environment; theft detection; kids activity monitoring.
- **Environmental Monitoring:** Sensor networks can be used to monitor environmental changes. Rainfall and flood monitoring; forest fire detection; weather forecasting; geological studies; precision agriculture.
- **Habitat Monitoring:** Biocomplexity mapping by adapting to environmental dynamics, through coordinated actuation by programmed triggering of sensing to enable identification, recording and analysis of interesting events to understand the behavior of birds, animals etc.
- **Health Application:** Tele-monitoring of human physiological data; Tracking and monitoring patients and doctors inside a hospital; Drug administration in hospitals; monitoring people's health conditions; Sensors

for: blood flow, respiratory rate, ECG (Electrocardiogram), blood pressure, and oxygen measurement.

- **Structure monitoring:** Sensors could be used to monitor vibration that could damage the structure of a building. Such systems can detect, localize and estimate the extent of damage. Thermostats and temperature sensor nodes can be deployed all over the building's area.
- **Smart sensor networks:** These networks have a number of independent sensors. Each of the sensors make local decision and all the decisions are combined and weighed based on a specific algorithm and a global decision is taken.

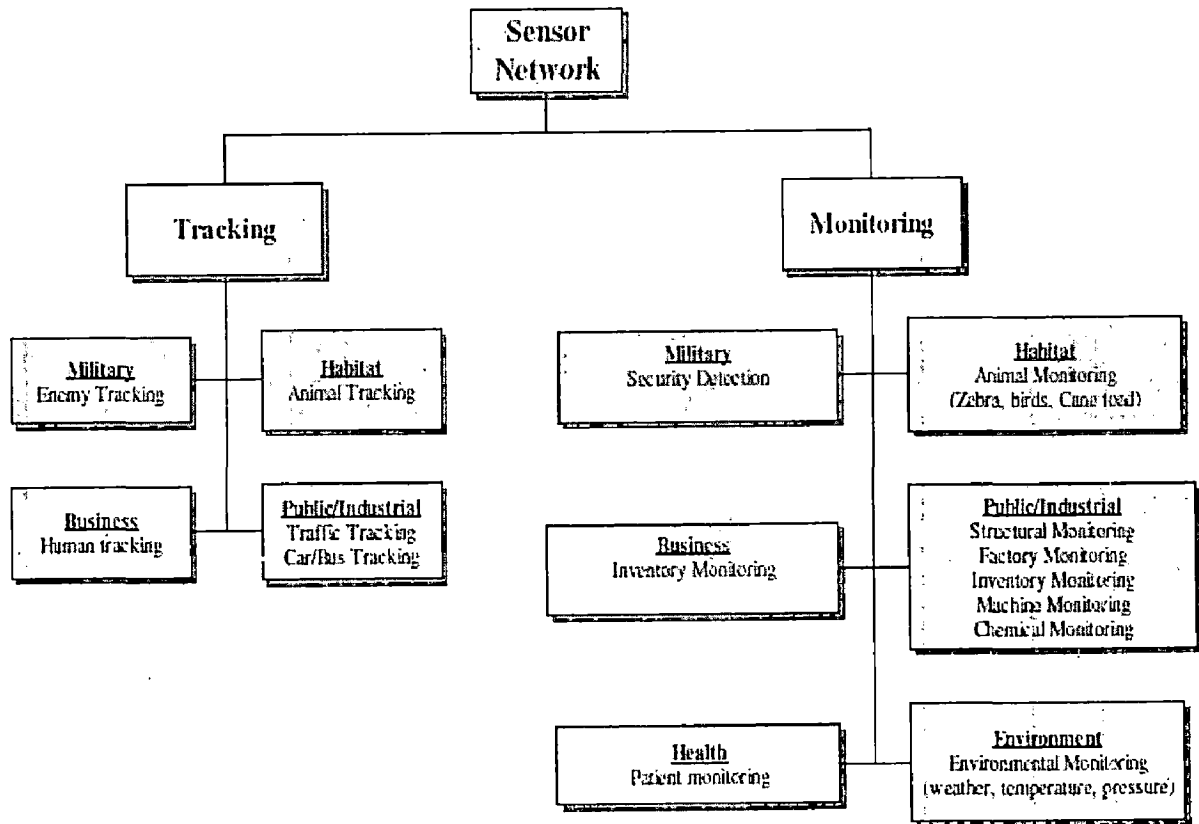


Figure 2.2 Overview of sensor applications [9]

2.1.3 WSNs versus MANETs

There is also considerable research in the area of mobile ad hoc networks (MANETs) [28, 29, 30]. WSNs are similar to MANETs in some ways; for example, both involve multihop communications. However, the applications and technical requirements for the two systems are significantly different in several respects.

- The typical mode of communication in WSN is from multiple data sources to a data recipient or sink (somewhat like a reverse multicast) rather than communication between a pair of nodes. In other words, sensor nodes use primarily multicast or broadcast communication, whereas most MANETs are based on point-to-point communications.
- In most scenarios (applications) the sensors themselves are not mobile (although the sensed phenomena may be); this implies that the dynamics in the two types of networks are different.
- Because the data being collected by multiple sensors are based on common phenomena, there is potentially a degree of redundancy in the data being communicated by the various sources in WSNs; this is not generally the case in MANETs.
- Because the data being collected by multiple sensors are based on common phenomena, there is potentially some dependency on traffic event generation in WSNs, such that some typical random-access protocol models may be inadequate at the queuing-analysis level; this is generally not the case in MANETs.
- A critical resource constraint in WSNs is energy; this is not always the case in MANETs, where the communicating devices handled by human users can be replaced or recharged relatively often. The scale of WSNs and the necessity for unattended operation for periods reaching weeks or months implies that energy resources have to be managed very judiciously. This, in turn, precludes high-data-rate transmission.

- The number of sensor nodes in a sensor network can be several orders of magnitude higher than the nodes in a MANET.

For these reasons the plethora of routing protocols and solutions that have been proposed for MANETs are not suitable for WSNs, and alternative approaches are required.

2.2 Data Compression

Data compression [5] is the process of encoding information using fewer bits than an unencoded representation would use, through use of specific encoding schemes. As with any communication, compressed data communication only works when both the sender and receiver of the information understand the encoding scheme. Compressed data can only be understood if the decoding method is known by the receiver. There are many known methods of data compression. They are based on the different ideas and are suitable for different types of data. They produce different results, but they are all based on the same basic principle that they compress data by removing the redundancy from the original data in the source file. The idea of compression by reducing redundancy suggests the general law of data compression, which is to "assign short codes to common events and long codes to rare events". Data compression is done by changing its representation from inefficient to efficient form [11].

2.2.1 Compression Techniques

When we speak of a compression technique or compression algorithm, we are actually referring to two algorithms. There is the compression algorithm that takes an input x and generates a representation x_c that requires fewer bits, and there is a reconstruction algorithm that operates on the compressed representation x_c to generate the reconstruction y . These operations are shown schematically in figure

2.3. We will follow convention and refer to both the compression and reconstruction algorithms together to mean the compression algorithm [10].

Based on the requirements of reconstruction, data compression schemes can be divided into two broad classes: *lossless* compression schemes, in which y is identical to x , and *lossy* compression schemes, which generally provide much higher compression than lossless compression but allow y to be different from x .

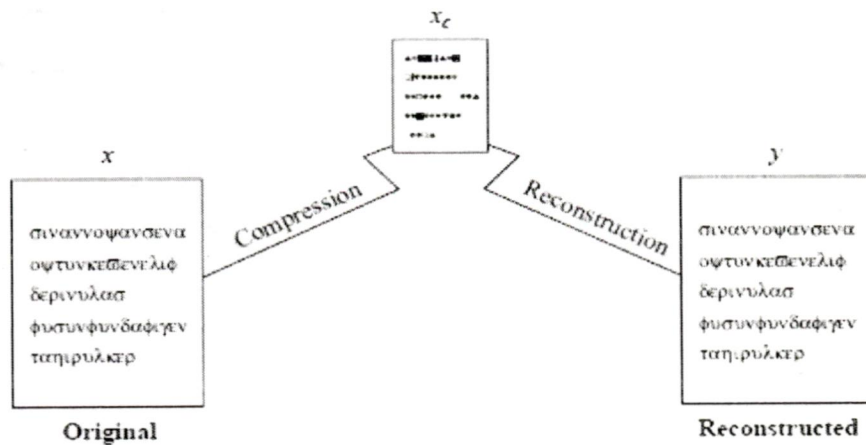


Figure 2.3 Compression and reconstruction [10]

Lossless Compression

Lossless compression techniques, as their name implies, involve no loss of information. If data have been losslessly compressed, the original data can be recovered exactly from the compressed data. Lossless compression is generally used for applications that cannot tolerate any difference between the original and reconstructed data. Text compression is an important area for lossless compression. It is very important that the reconstruction is identical to the text original, as very small differences can result in statements with very different meanings. Consider the sentences “Do *not* send money” and “Do *now* send money.” A similar argument holds for computer files and for certain types of data such as bank records [10].

There are many situations that require compression where we want the reconstruction to be identical to the original. There are also a number of situations in which it is possible to relax this requirement in order to get more compression. In these situations we look to lossy compression techniques.

Lossy Compression

Lossy compression techniques involve some loss of information, and data that have been compressed using lossy techniques generally cannot be recovered or reconstructed exactly. In return for accepting this distortion in the reconstruction, we can generally obtain much higher compression ratios than is possible with lossless compression.

In many applications, this lack of exact reconstruction is not a problem. For example, when storing or transmitting speech, the exact value of each sample of speech is not necessary. Depending on the quality required of the reconstructed speech, varying amounts of loss of information about the value of each sample can be tolerated. If the quality of the reconstructed speech is to be similar to that heard on the telephone, a significant loss of information can be tolerated. However, if the reconstructed speech needs to be of the quality heard on a compact disc, the amount of information loss that can be tolerated is much lower. Similarly, when viewing a reconstruction of a video sequence, the fact that the reconstruction is different from the original is generally not important as long as the differences do not result in annoying artifacts. Thus, video is generally compressed using lossy compression [10].

2.2.2 Modeling and Coding

The development of data compression algorithms for a variety of data can be divided into two phases [5]. The first phase is usually referred to as *modeling*. In this phase we try to extract information about any redundancy that exists in the data and describe the redundancy in the form of a model. With a perfect, concise

model that describes the generation of the input source which is to be compressed, one could reproduce the data without transmitting the source data. For example, if the sequence 1 1 2 3 5 ...6765 were to be transmitted, one could express it with a “model” of Fibonacci numbers. In practice, one must approximate and construct an approximate mathematical model for the data. In English text, for example, one can model the probability of a letter occurring as a probability conditioned on letters that have already been transmitted. This probabilistic model is transmitted with a description of how the data differs from the model. The second phase is called *coding* in which information is mapped to compact *codewords*. A codeword must decode to a unique value so there can be no doubt of the original message. In this phase, a description of the model and a “description” of how the data differ from the model are encoded, generally using a binary alphabet. Often the modeling and coding steps are tightly coupled [10]. Some popular coding schemes for data compression tools are:

Huffman Coding

If the probability of each source symbol is known a priori (perhaps by scanning through the source), a procedure known as static *Huffman coding* can be used to build an optimal code in which the most frequently occurring symbols are given the shortest codewords [5]. Huffman codes are established by storing the symbols of the alphabet in a binary tree according to their probability. As the tree is traversed from root to leaf, the code grows in length. When visiting the right child, a 0 is appended to the code. When visiting the left child, a 1 is appended. Thus, symbols which occur frequently are stored near the root of the tree and have the shortest codes. Since data compression tools rarely have the luxury of a priori knowledge and cannot afford two passes through the data source, variants of the Huffman algorithm have been developed that work dynamically and update the tree as source symbols are encountered [10, 11].

Arithmetic Coding

Optimal compression ratio for a data source is traditionally described with respect to Claude Shannon's definition of source entropy: a measure of the source's information and therefore the average number of bits required to represent it. Sometimes the most frequently occurring symbol can contain so little information that it would be ideal to represent it with less than 1 bit. Huffman codes are restricted to using an integral number of bits per symbol, increasing the coding overhead. Arithmetic codes have been designed to support a fractional number of bits per symbol to bring the average length of a codeword much closer to the optimal [10, 11]. Knowing the probability of occurrence for each symbol, a unique identifier can be established for a series of symbols. This identifier is a binary fraction in the interval $[0, 1)$. Unlikely symbols narrow this interval so that more bits are required to specify it, while highly likely symbols add little information to a message and require the addition of fewer bits as the interval refinement is coarser. As the fraction converges, the most significant bits become fixed, so the fraction can be transmitted most-significant-bit-first as soon as it is known. Arithmetic coding requires frequent division and multiplication, but this experiments show that an optimized implementation can run faster than the well-optimized UNIX *compact* program, an adaptive Huffman encoder [5].

Lempel-Ziv Codes

A Lempel-Ziv codebook [10] is made up of fixed-length codewords in which each entry has nearly the same probability of appearing, but in which longer groups of symbols are represented in the same length as single symbols. Thus, it may require extra bits to send the coded version of a single symbol, but a string of frequently occurring symbols can be represented with a fraction of the bits ordinarily required. Since only n codewords can be represented with $\lg(n)$ bits, systems for dynamically increasing the length of codewords exist.

There are two terms used in data compression one is compression rate and other is compression ratio. Compression rate is the rate of the compressed data. Typically, it is in units of bits/sample, bits/character, bits/pixels, or bits/second. Compression

ratio is the ratio of the size or rate of the original data to the size or rate of the compressed data. For example, if a gray-scale image is originally represented by 8 bits/pixel (bpp) and it is compressed to 2 bpp, such that the compression ratio is 4-to-1. Compression rate is an absolute term, while compression ratio is a relative term [5].

2.3 Data Compression Schemes for WSNs

Wireless sensor networks are resource constraint: limited power supply, bandwidth for communication, processing speed, and memory space. One possible way to achieve maximum utilization of those resources is by applying data compression on sensor data. Existing compression algorithms are not applicable for sensor nodes because of their limited resource. Therefore, it is necessary to design a low-complexity and small size data compression algorithm for sensor networks. Some of data compression schemes for WSNs are:

2.3.1 Distributed Compression

The basic idea behind the Distributed Compression scheme is using side information to encode source information [12]. For instance, there exist two sources (X and Y) as shown in figure 2.4.

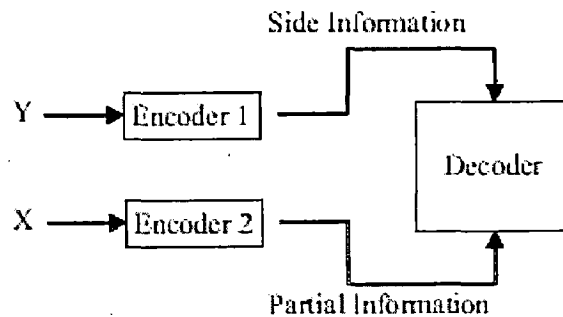


Figure 2.4 Distributed Compression examples

They are correlated and discrete-alphabet independent identically distributed. Since in a sensor network, sensor nodes will be densely populated in a sensor

field, this correlation condition can be satisfied easily. Then, X can be compressed at the theoretical rate of its conditional entropy, $H(X|Y)$, without the Encoder 1 accessing Y. The conditional entropy [12] can be expressed as

$$H(X|Y) = - \sum_y P_y(y) \sum_x P_x(x/y) \log_2 P_x(x/y) \quad (2.1)$$

The general scheme of Distributed Compression is first to compose cosets, whose codevectors of source X. The distance of any two codevectors in the same coset has to be large enough. An index value is assigned to each coset. When transmitting data to a decoder, the source X only sends an index value of coset, to which the codevector belongs. The source Y sends a codevector as a side-information. The decoder looks up the coset, which has the same index received from X. Then, the decoder selects one codevector, which has a closest value to the codevector sent by Y, in the coset.

A simple example of Distributed Compression [13] is as follow. There are two (X and Y) 3-bit data sets. The Hamming distance between X and Y is no more than one bit. If both Encoder 2 and Decoder know Y, X can be compressed to 2 bits. Then if Y is only known by the Decoder, what will happen to the compression rate of X? According to the Distributed Compression scheme, since the Decoder know Y and X is only one Hamming distant apart from Y, it is not efficient to distinguish X=111 from X=000. As a same reason, X=001 and 110, X=010 and 101, and X=011 and 110 do not need to be distinguished from each other. These sets of two X values are grouped as 4 cosets and assigned 4 different binary index numbers:

coset 1 = (000, 111) : 00

coset 2 = (001, 110) : 01

coset 3 = (010, 101) : 10

coset 4 = (011, 100) : 11

If X=010 and Y=110, the Decoder received Y=110 as a side information from Y and X=10 as a partial information form X. Then, at the Decoder, X=010 is

selected from coset 2 since 110 has a Hamming distance of 2 from 110. Whether X know Y or not, X can still compress 3 bits information into 2 bits.

Distributed Compression scheme can be applied to not only discrete sources as illustrated by the above example, but also continuous sources. Also, it can be used for both lossless and lossy compression scheme. For example, 4 cosets can be formed from an 8-level quantizer. It is noticed that two codevectors in each coset are grouped so that they can have the maximum possible distance from each other.

2.3.2 Coding by Ordering

The Coding by Ordering data compression scheme [14] is part of Data Funneling Routing. The compression scheme works as follows. First, a data path from sensor nodes in the interested region to a collector node is set up as shown in figure 2.5. In Data Funneling Routing, some of the sensor nodes work as data aggregation nodes. For example, nodes A, B, and D are data aggregation nodes. At an aggregation node, sensing data collected by other nodes is combined, and the aggregated data is sent to its parent node. At node D in figure 2.5, data collected by node E is combined with data collected by node D itself. Then, the aggregated data is transmitted to node B.

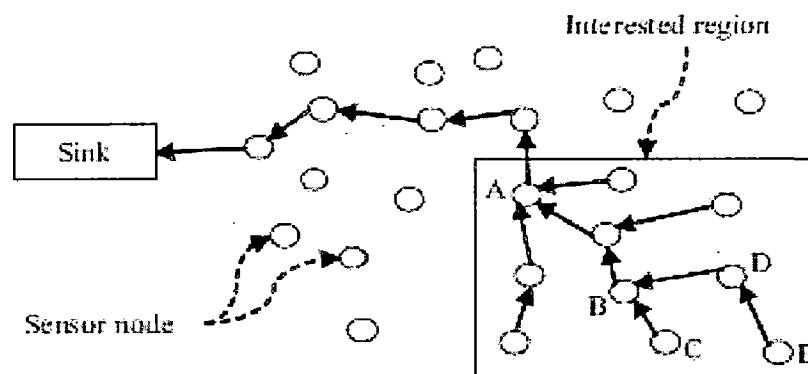


Figure 2.5 Data path in coding by ordering data compression scheme

In the algorithm, when data is combined at an aggregation node, some data is dropped. To include the information of dropped data in the aggregated data, the order of data packet is utilized. For example, four nodes (N1, N2, N3, and N4) send the data to an aggregation node (Na). The data value of each node can be any integer ranging from 0 to 5. If we decide to drop the data from N4 and express the data from N4 by ordering packets from other 3 nodes (N1, N2, and N3), there are $3! = 6$ possible ordering. Therefore, by using permutation of three packets, the data value of N4 can be included in an aggregated packed without actually including the packet of N4. The possible combination of permutation and data value is presented in Table 2.1.

Table 2.1 Permutation and its represented integer value

Packet Permutation	Integer Value
N1,N2,N3	0
N1,N3,N2	1
N2,N1,N3	2
N2,N3,N1	3
N3,N1,N2	4
N3,N2,N1	5

For a general case, let's assume that n is the total number of sensor nodes – each node has different a node ID, m is the number of nodes sending a packet to an aggregation node, k is the possible range of data value, and l is the number of sensor node dropped at the aggregation node. Then, the number of possible combination of IDs, which dropped nodes have, can be expressed as ${}^{n-m+l}C_l$. Since each of l nodes can take any value among possible k data values, there are k^l possible data value combinations. When combining possible IDs and data values, there is total of $({}^{n-m+l}C_l)k^l$ possible values. This combination of values needs to be expressed by $(m-l)!$ permutations. Therefore, the following inequality has to be satisfied.

$$(m-l) \geq ({}^{n-m+l}C_l)k^l$$

Theoretically, when $n = 27$, $k = 24$, and $m = 100$, approximately 44% of packets can be dropped at the aggregation node by applying Coding by Ordering. Since this method has good compression ratio and simple algorithm, it may be possible to use for WSNs. One difficulty of utilizing this scheme is that since there is no efficient algorithm mapping permutation to data value, it requires a mapping table. As the number of sensor nodes aggregated increases, the size of table increases exponentially.

2.3.3 Pipelined In-Network Compression

The basic idea of pipelined in-network compression scheme [15] is trading high data transmission latency for low transmission energy consumption. Collected sensor data is stored in an aggregation node's buffer for some duration of time. During that time, data packets are combined into one packet, and redundancies in data packets, will be removed to minimize data transmission.

For example, each data packet has the following form: <measured value, node ID, timestamp>. Then, the compressed data packet has the following form: <shared prefix, suffix list, node ID list, timestamp list>. The "shared prefix" is the most significant bits, which all measured values in combined data packets have in common. The length of shared prefix can be changed by a user based on the knowledge of data similarity. If the measured values are expected be close to each other, the length of prefix value can be set to relatively long. The "suffix list" is the list of measured values excluding the shared prefix part. The "node ID list" is the list of node identifiers and the "timestamp list" is the list of timestamp. The compression scheme is illustrated in Figure 2.6. In the figure, three nodes send the data packets to the compression node. At the compression node, three data packets are compressed into one packet. In this example, the length of shared prefix is set to 3. In this example, total number of bits is reduced from 33 to 27.

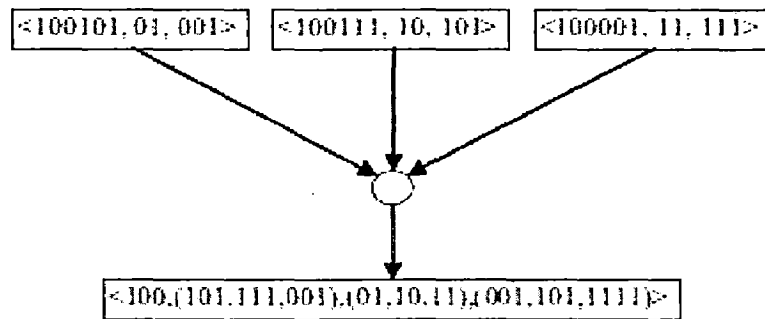


Figure 2.6 In-network Compression

One advantage of this simple compression scheme is that the shared prefix system can be used for node IDs and timestamps. By doing so, more data compression can be achieved. The efficiency of data compression depends on the length of shared prefix. If we could set a long shared prefix and measured values have commonality, the compression ratio increases. However, there is no similarity in measured sensor values. Even if we could set a long shared prefix, the efficiency of Pipelined In-Network Compression will decrease. In addition, if we are combining a large amount of data packets, than a large data buffer is required to temporary store those packets. Since a sensor node has only a limited size of memory space, enough buffer space will not be available.

Although focusing on wireless ad-hoc networks, Barr and Asanović show in [5] that the energy required for transmitting a bit can be equivalent to the energy consumption of a thousand microcontroller operations. However, their results were acquired from a Compaq Personal Server handheld, which features 32 megabytes of RAM and 16 kilobytes of cache. This significantly exceeds the resource constraints on many of today's mote platforms.

Pradhan *et al.* [13] suggested a framework for distributed compression, in which they have used joint source and channel coding that brings about the minimization in the amount of inter-node communication for compression using both a quantized source and correlated side information within each individual node. With the

introduction of many application domains like various kind of sensor networks that are severely energy and power constrained, the topic of energy efficiency is getting an important aspect to be taken into account. In some of the literatures, both of the aspects, data compression as well as energy efficiency are considered together.

Magli et al. [16] introduced a low-complexity video compression scheme which is based on JPEG data compression. This algorithm is specifically designed for the wireless video surveillance system.

Hans and Schafer [17] present an overview of lossless data compression in the context of audio data.

Zhuang and Li [18] have implemented the compression algorithms for seismic data. In this paper, the amount of energy reduction due to the reduction in data after compression has been estimated while considering only the energy costs of communication.

Sadler and Martonsi [19] have described a variation of lossless LZW (Lempel-Ziv-Welch) algorithm pertaining to the common sensor platforms with few kilobytes of memory. This version can carry out the compression of the data block with a length of 528 bytes at a time. The S-LZW (Simple LZW) algorithm causes the saving in energy by the factor of more than 1.5x locally, and over 2.5x as far as the overall network is concerned, for the tests carried out with the data retrieved from real sensor networks. However, the evaluation of the system was made out with delay-tolerant network setting while data was buffered before being transmitted.

Tsiftes et al. compare mechanisms to compress code updates to remotely reconfigure nodes in [20]. They present the algorithm, which combines multiple preprocessing and coding steps, while maintaining the characteristics that the decoder part can still be run on memory-constrained sensor platforms. Results show that network-wide

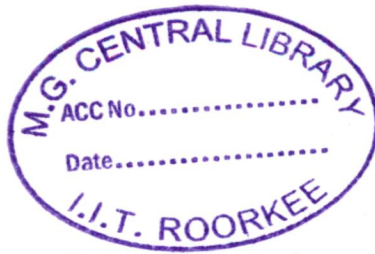
energy savings up to 67% can be achieved when compressing the code updates using GZIP (GNU zip).

Ju and Cui presented the EasiPC packet compression mechanism in [21]. Each packet field needs to be analyzed and classified according to its changing frequency in advance. Different compression methods are associated with each category; while randomly changing fields are always transferred uncompressed, sequence number fields are e.g. encoded by difference coding. In addition, the sensor readings are encoded by either difference coding or length-variable coding. Removing redundant information from the packet, the developed mechanism allows for compression gains of up to 50% as well as the corresponding reduced transmission delays. The results were however not verified by an energy analysis.

Reinhardt, M. Hollick, and R. Steinmetz [22] present a compression framework utilizing a stream-oriented compression scheme in which efficient data transfer between nodes is provided by shifting data compression into a dedicated layer for sensor networks.

Capo-Chichi et al. [23] proposed a data compression algorithm K-RLE inspired from Run Length Encoding which increases the ratio compression compared to RLE and SLZW where K is a precision parameter. The algorithm is lossy algorithm. The energy consumption study shows that while 2-RLE offers a better compression ratio than RLE and S-LZW(Simple LZW), it consumes half energy compared to S-LZW which uses the most energy.

Zhou Yan-li et al. [24] proposed an improved LZW (Lempel-Ziv-Welch) algorithm for wireless sensor network nodes. The compression algorithm is lossless data compression algorithm. The results of compression test on sample data shows that the algorithms can significantly improve the compression ratio and reduce the size of the dictionary as well as energy consumption in data transmission than LZW algorithm.



Kirsten Dolfus and Torsten Braun [25] explore the design space for data compression for wireless sensor and mesh networks by profiling common, publicly available algorithms. They focus on lossless, stand-alone (thus non-distributed) compression schemes to lower the amount of data stored within a network node or transmitted across the network. The performance of algorithms depends on the platforms used. Several goals such as a low overhead in terms of utilized memory and compression time as well as a decent compression ratio have to be well balanced in order to find a simple, yet effective compression scheme.

A different approach has been explored by Marcelloni et al. [26]: In order to keep the algorithm as simple as possible and to avoid complex computations on embedded nodes, their solution relies on a two-phase coding process based on a lookup table of the size of the analog-digital converter and compresses the raw bits of a sensor reading. Here, a codeword is a hybrid of unary and binary codes supplied by an adequate dictionary similar to the one used for DC coefficient coding in JPEG compression. Since the size of the dictionary is fixed and encoding is done via mapping, the algorithm is well suited for on-the-fly compression. However, the obtainable compression ratio is highly dependent on a good mapping strategy.

2.4 Shortcomings and Research Gaps

Many data compression schemes have been carefully designed for Sensor Networks so as to make the most efficient use of the limited resources in terms of energy, computation and storage. Most of these data compression schemes aim to exploit the technological improvements to make the devices more energy efficient rather than making them more powerful. To reduce energy consumption a lot of work has been done upon – finding optimal paths between source and sink for transfer, aggregation of smaller data units from different sources or generated at different time to a combined data unit (by data fusion), aggregation of nodes to act as a super node by clustering. Some techniques have even shown relation between routing and spatial &

temporal correlations to save energy by means of calculating joint entropy. However, following are the shortcomings and research gaps that have been observed:

1. Previous research for data compression in communication mainly focuses on how to decrease delay or save required transmission bandwidth. No any data compression algorithm discusses the energy savings with reducing memory when sensor data are compressed at the originating node.
2. It is well known that data transmission is the major contributor to energy consumption in WSN. There is a need to understand and study the effects of reducing the power consumption by shrinking raw data down to smaller volumes, which is desirable for data communication since the compressed data can require significantly less time and energy to transmit compared to the raw data.
3. There is little or no work done in the field of suppressing spurious or redundant data at the sensor nodes itself before dissemination. Such a technique would affect the amount of data in the network and reduce inter-node transmissions and result in energy and memory savings.

Proposed Data Compression Algorithm

3.1 Introduction

The studies in the previous chapter illustrate that sending data is more power consuming than computation, and thus minimizing data size before transmitting in wireless medium is effective to reduce total power consumption. Therefore, it is beneficial for WSNs to employ a data compression algorithm. In this chapter, a data compression algorithm for wireless sensor networks has been proposed. The compression algorithm is particularly suited to the reduced memory and computational resources of a wireless sensor network node. The compression scheme exploits the high correlation that typically exists between consecutive samples collected by the sensor onboard a node. Using this characteristic and following the principles of entropy compression [6], the algorithm is able to compute a compressed version of each value acquired from a sensor. Finally, the compression algorithm is lossless. A decompression algorithm is also proposed to retain the originality of data at base station

3.2 Model of Proposed Data Compression Algorithm

The proposed algorithm consists of three phases:

1. **Selector:** This phase selects three previous values with current value. Initially, assuming three readings are zero.
2. **Preprocessor:** It predicts the median value along with lowest and highest values among three previous values selected and finds the suitable interval in which current value belong. It calculates the deviation of current value in the suitable interval.

3. **Encoder:** Encoder gives the variable length code to the deviation obtained in the previous phase by removing the redundancy from data.

The basic idea of encoding is to map an alphabet to a representation for that alphabet, composed of sequences of bits of variable sizes, so that symbols that occur frequently have a smaller representation than those that occur rarely. Block diagram of proposed compression algorithm is shown in figure 3.1.

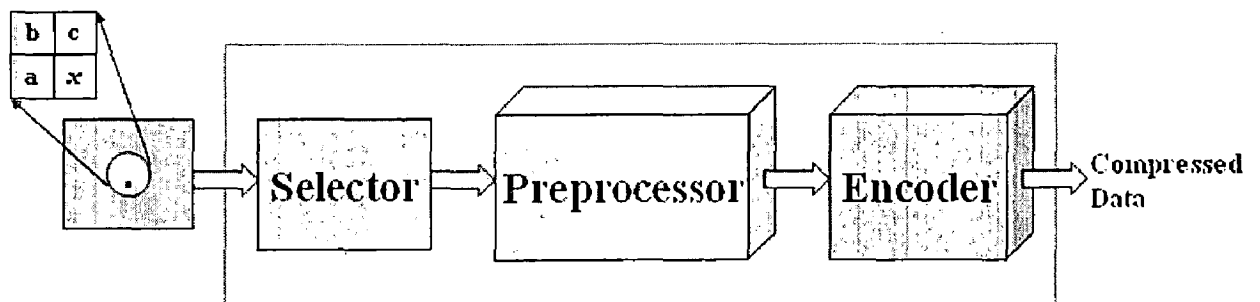


Figure 3.1 Model of Proposed Compression Algorithm

3.3 The Compression Algorithm

Suppose x_i is the current value of the reading taken by the sensor node where subscript i denote the sample number. Since the application of sensor nodes is mostly used to report the status of interested area to users, it is more useful to measure x_i acquired by sensor node. For each new acquisition x_i , the selector phase of compression algorithm selects three previous values (let three previous values are a_i , b_i and c_i) and pass these values to next phase. The preprocessor phase predicts the median value m_i along with lowest value l_i and highest value h_i among three previous values selected and finds the interval of the current value then computes the deviation d_i of current value in that interval. The encoder performs compression losslessly by encoding differences d_i more compactly based on their statistical characteristics. Each d_i is represented as a bit sequence bs_i composed of two parts $s_i|t_i$, where s_i codifies the interval in which current value x_i lies and t_i is the representation of d_i .

```

encode (xi, prevArray[])
// xi is current value and prevArray[] contains previous three
values
  ai = prevArray[0]
  bi = prevArray[1]
  ci = prevArray[2]
  li = minimum(ai, bi, ci)           // calculate lowest value
  mi = median(ai, bi, ci)           // calculate median value
  hi = maximum(ai, bi, ci)           // calculate highest value
  IF (li <= xi <= hi) THEN
    SET set_bit TO '0'
    SET si TO set_bit
    // si codifies the interval
    IF (xi < mi)
      SET set_bit TO '0'
      SET si TO << si, set_bit >>
      SET meani TO  $\left\lfloor \frac{l_i + m_i}{2} \right\rfloor$ 
      SET di TO (meani - xi)
    ELSE
      SET set_bit TO '1'
      SET si TO << si, set_bit >>
      SET meani TO  $\left\lfloor \frac{m_i + h_i}{2} \right\rfloor$ 
      SET di TO (meani - xi)
    ENDIF
  ELSE
    SET set_bit TO '1'
    SET si TO set_bit
    IF (x > h)
      SET set_bit TO '0'
      SET si TO << si, set_bit >>

```

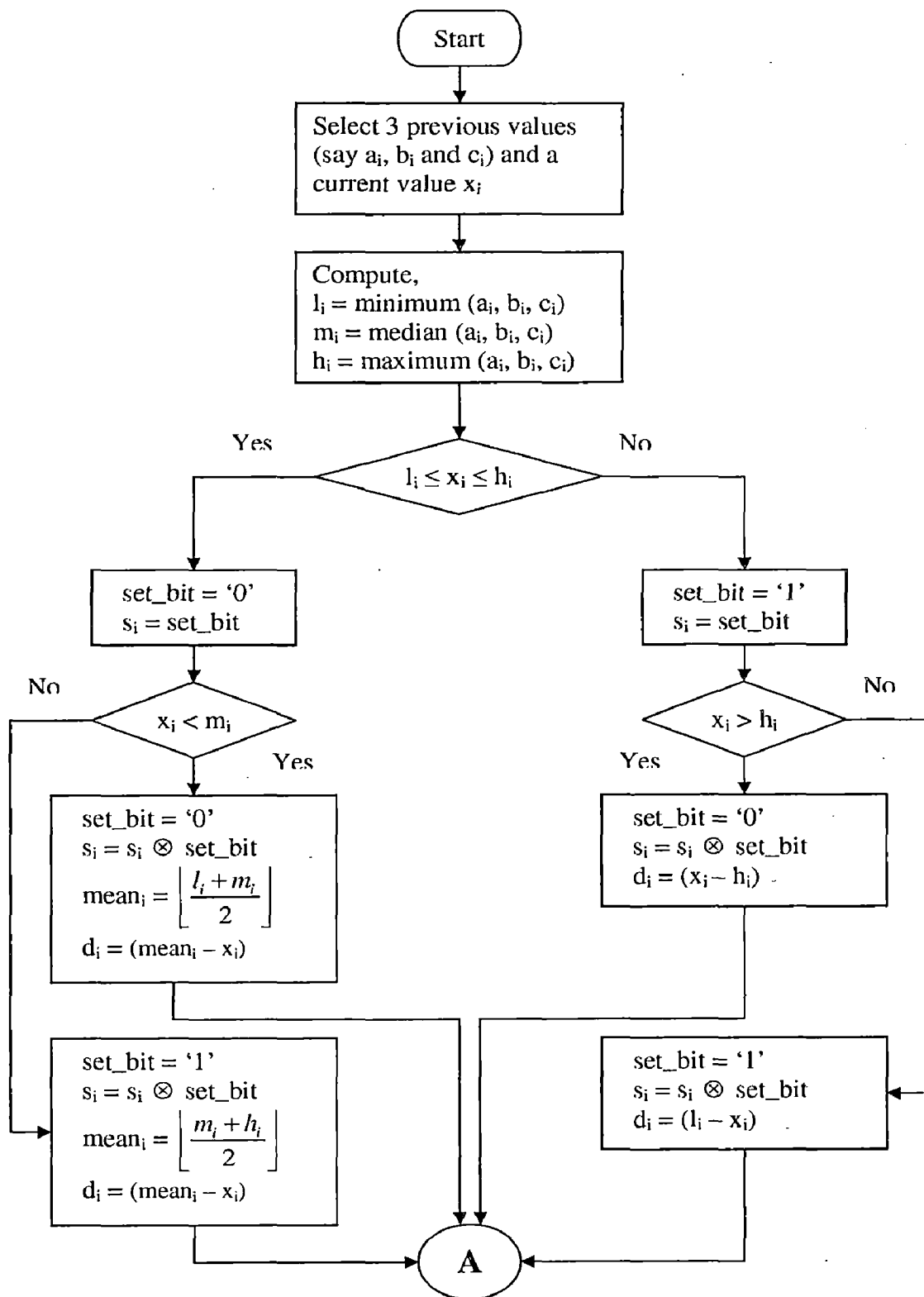
```

        SET  $d_i$  TO  $(x_i - h_i)$ 
    ELSE
        SET set_bit TO '1'
        SET  $s_i$  TO  $\ll s_i, \text{set\_bit} \gg$ 
        SET  $d_i$  TO  $(l_i - x_i)$ 
    ENDIF
ENDIF
ENDIF
IF  $d_i = 0$  THEN
    SET  $n_i$  TO 0 // build  $bs_i$ 
    SET  $bs_i$  TO  $s_i$  //  $t_i$  is not needed
ELSE
    IF  $d_i > 0$  THEN // build  $t_i$ 
        SET  $n_i$  TO  $\lfloor \log_2 |d_i| \rfloor + 1$ 
        SET  $t_i$  TO  $(d_i) \lfloor_{n_i}$ 
    ELSE
        SET  $n_i$  TO  $\lfloor \log_2 |d_i| \rfloor + 1$ 
        SET  $t_i$  TO  $(d_i - 1) \lfloor_{n_i}$ 
    ENDIF
    SET  $bs_i$  TO  $\ll s_i, t_i \gg$  // build  $bs_i$ 
ENDIF
RETURN  $bs_i$ 

```

Figure 3.2 Pseudo-code of the proposed encode algorithm

The s_i part of the bit sequence bs_i is computed in preprocessor phase of compression algorithm. The coder phase compute the t_i part of bit sequence bs_i and appends this t_i with s_i . In the preprocessor phase of compression algorithm, current value x_i may lie in any one of four intervals. First interval takes values greater than and equal to l_i and less than m_i whereas second interval takes values greater than and equal to m_i and less than h_i . The s_i code for first interval is '00' and for second interval is '01'. We calculate mean for these intervals to find deviation of current value x_i from mean value (say, $mean_i$) and compute the difference $d_i = (mean_i - x_i)$ for both intervals.



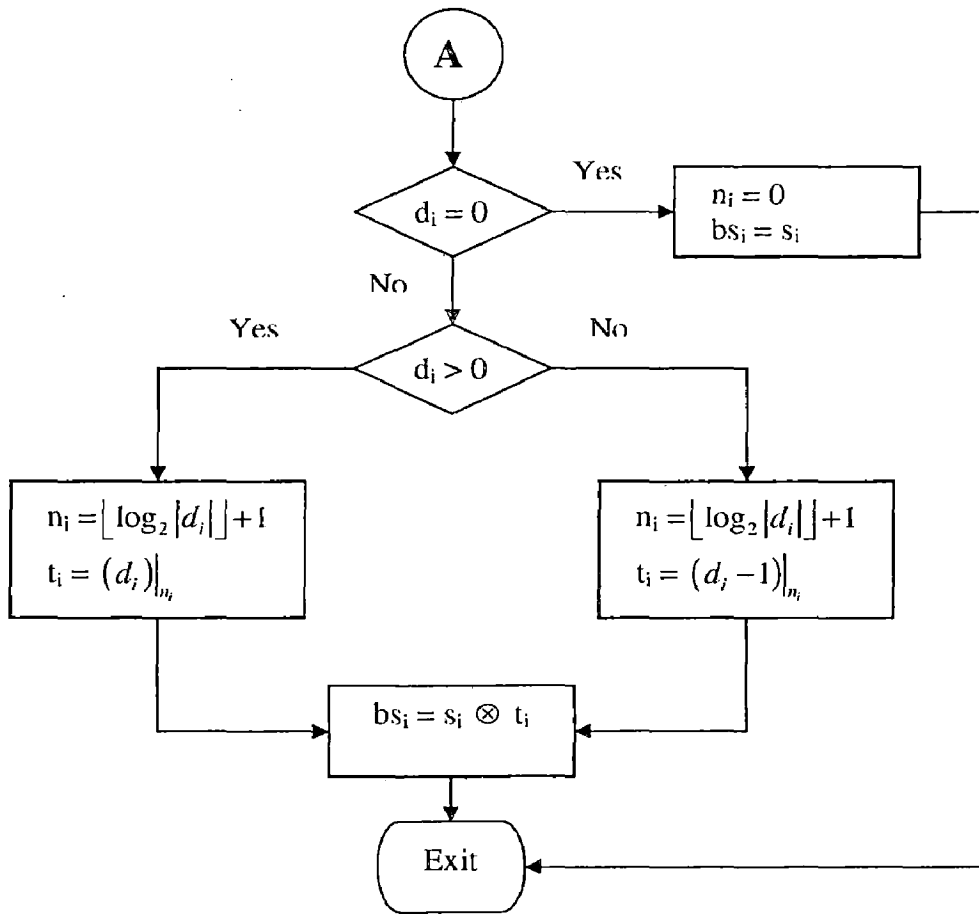


Figure 3.3 Flow chart of proposed data compression algorithm

Third interval takes value greater than h_i and the s_i code for this interval is '10'. Similarly fourth interval takes value less than l_i and '11' is the s_i code for this interval. Compute the difference $d_i = (x_i - h_i)$ for third interval and $d_i = (l_i - x_i)$ for fourth interval.

Suppose n_i is the number of bits needed to represent d_i . If $d_i = 0$, then $n_i = 0$ else $n_i = \lfloor \log_2 |d_i| \rfloor + 1$. If $d_i = 0$, then bit sequence bs_i is represented by s_i only. Otherwise, t_i is calculated and then concatenated with s_i to form bs_i .

The t_i part of the bit sequence bs_i is a variable-length integer code generated as follows:

1. If $d_i > 0$, t_i corresponds to the n_i low-order bits of the two's complement representation of d_i ;
2. If $d_i < 0$, t_i corresponds to the n_i low-order bits of the two's complement representation of $(d_i - 1)$;
3. If $d_i = 0$, t_i is not represented.

The procedure used to generate t_i guarantees that all possible values have different codes. Once bs_i is generated, it is appended to the bit stream which forms the compressed version of the sequence of measures x_i . Figure 3.2 summarizes the algorithm used to encode measure x_i . Here, $\langle\langle s_i, t_i \rangle\rangle$ denotes the concatenation of s_i and t_i while vl_{ni} denotes the n_i low-order bits of v . Flow chart of proposed data compression algorithm is shown in figure 3.3. In flow chart, symbol \otimes represents the concatenation operation.

3.4 The Decompression Algorithm

Decompression is the process of decoding information to get original data from compressed data, through use of specific decoding schemes. As with any communication, compressed data communication only works when both the sender and receiver of the information understand the encoding and decoding scheme. Compressed data can only be understood if the decoding method is known by the receiver. The decompression algorithm reconstructs the measure x_i from received bit sequence bs_i . Bit sequence bs_i is composed of two parts $s_i|t_i$, where s_i codifies the interval in which current value x_i lies and t_i is the representation of d_i (d_i is the deviation of current value in the suitable interval).

```
decode (bsi, prevArray[])
```

```
// bsi is received bit sequence and prevArray[] contains previous  
three values
```

```
  ai = prevArray[0]
```

```
  bi = prevArray[1]
```

```
  ci = prevArray[2]
```

```
  li = minimum(ai, bi, ci)           // calculate lowest value
```

```
  mi = median(ai, bi, ci)           // calculate median value
```

```
  hi = maximum(ai, bi, ci)         // calculate highest value
```

```
  SET count TO no. of bits in bsi
```

```
  SET ni TO (count - 2)
```

```
  IF ni = 0 THEN
```

```
    SET di TO 0
```

```
    SET si TO bsi
```

```
  ELSE
```

```
    // break bsi into si and ti
```

```
    SET si TO first two bits of bsi
```

```
    SET ti TO remaining bits of bsi
```

```
    // read first bit of ti
```

```
    IF ti[0] = '0'
```

```
      SET ri TO  $-(2^{n_i} - 1)$ 
```

```
      SET dti TO decimal value of ti
```

```
      SET dti TO (ri + dti)
```

```
    ELSE
```

```
      SET dti TO decimal value of ti
```

```
    ENDIF
```

```
    SET di TO dti
```

```
  ENDIF
```

```
  // read first bit of si
```

```
  IF si[0] = '0'
```

```
    // read second bit of si
```

```
    IF si[1] = '0'
```

```

    meani =  $\left\lfloor \frac{l_i + m_i}{2} \right\rfloor$ 

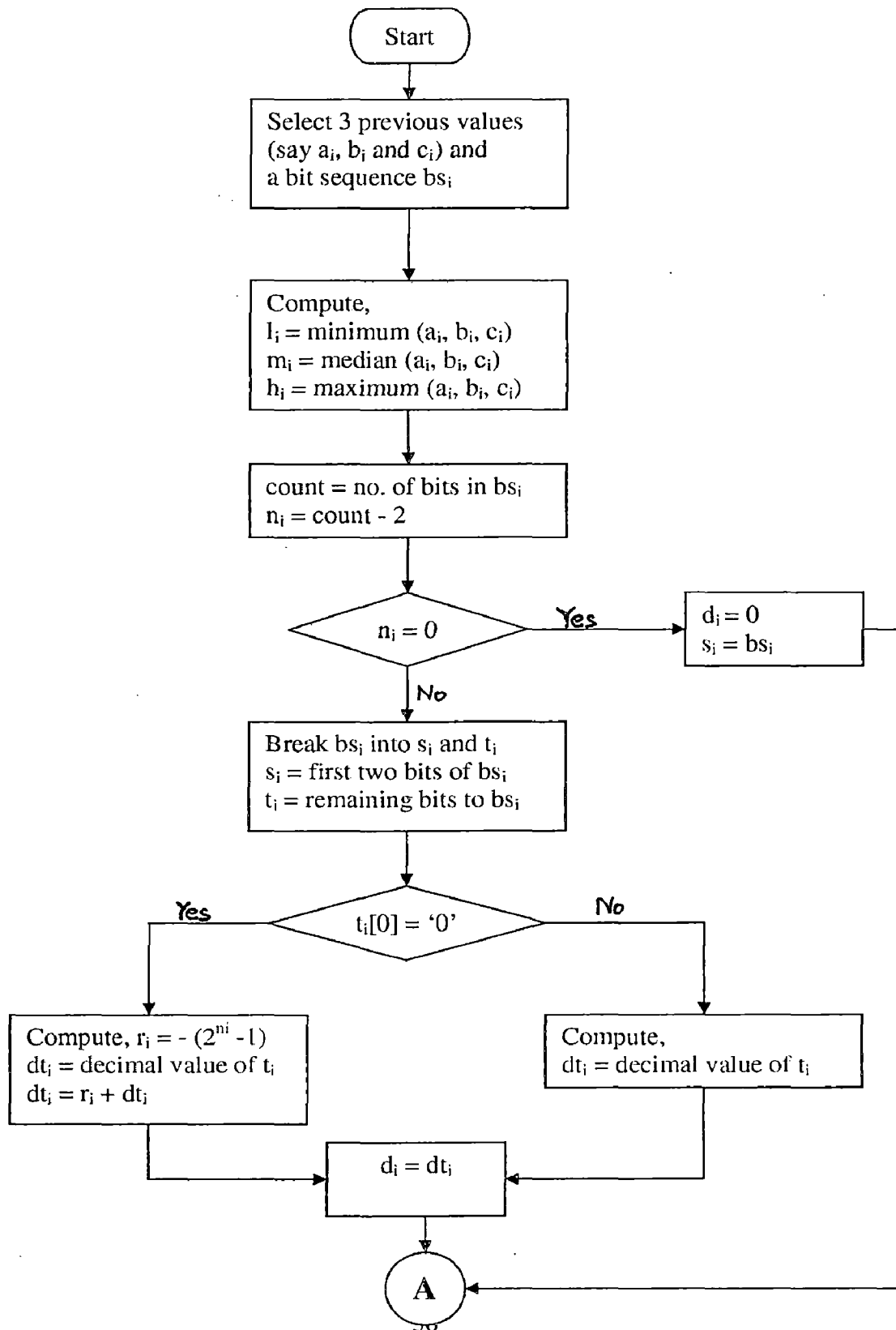
ELSE
    meani =  $\left\lfloor \frac{m_i + h_i}{2} \right\rfloor$ 

ENDIF
// compute measure xi
SET xi TO (meani - di)
ELSE
// read second bit of si
IF Si[1] = '0'
// compute measure xi
SET xi TO (di + hi)
ELSE
// compute measure xi
SET xi TO (li - di)
ENDIF
ENDIF
RETURN xi

```

Figure 3.4 Pseudo-code of the decode algorithm

The algorithm selects three previous values and compute lowest value l_i , median value m_i and highest value h_i among them. Count the number of bits presented in the bit sequence bs_i and set n_i to $(count-2)$. If $n_i = 0$, then $d_i = 0$ and $s_i = bs_i$ else break the bs_i into s_i and t_i . Assign first two bits of bs_i to s_i and remaining bits to t_i . Read first bit of t_i if it is '0', then compute $d_i = -(2^{n_i} - 1) + \text{decimal value of } t_i$) otherwise, assign decimal value of t_i to d_i . Read first and second bit of s_i such that s_i may take any one of four possible values from '00', '01', '10', '11'.



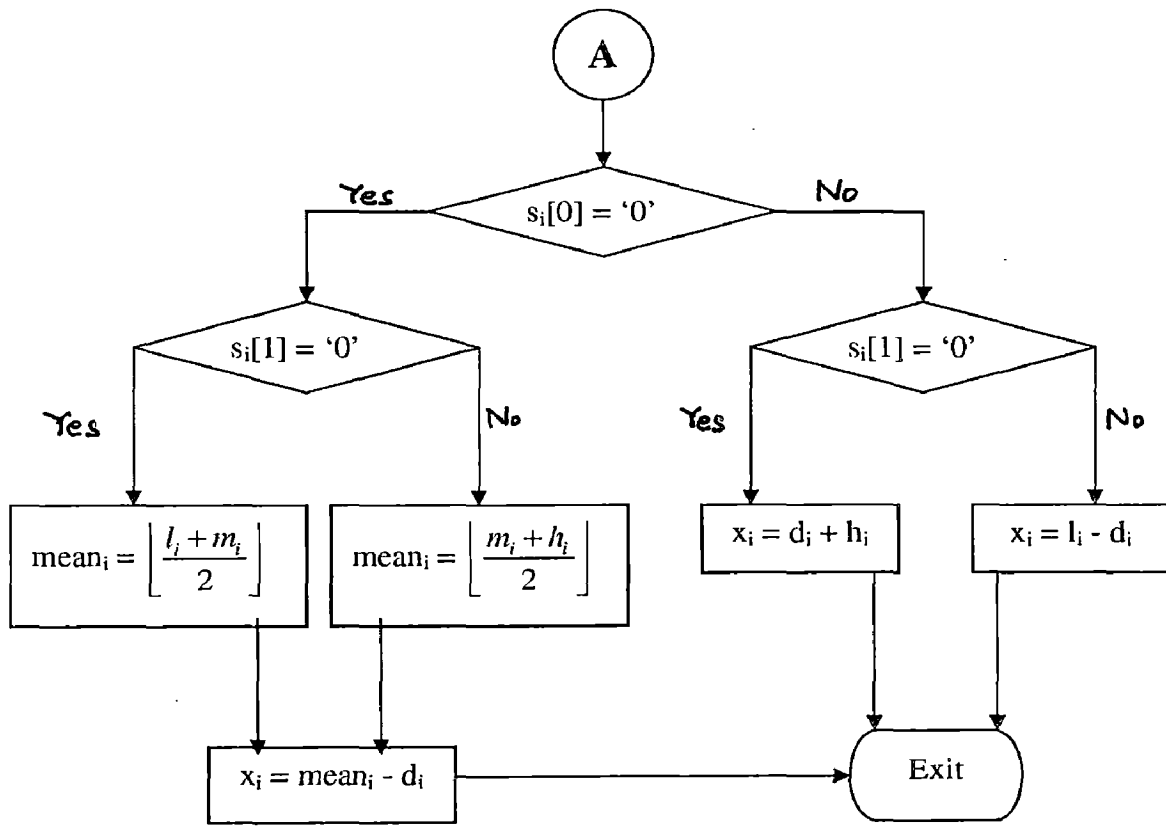


Figure 3.5 Flow chart of decomposition algorithm

There are four cases arise to compute measure x_i :

1. If $s_i = '00'$, then calculate $x_i = (\text{mean}_i - d_i)$ where, $\text{mean}_i = \left\lfloor \frac{l_i + m_i}{2} \right\rfloor$
2. If $s_i = '01'$, then calculate $x_i = (\text{mean}_i - d_i)$ where, $\text{mean}_i = \left\lfloor \frac{m_i + h_i}{2} \right\rfloor$
3. If $s_i = '10'$, then calculate $x_i = (d_i + h_i)$
4. If $s_i = '11'$, then calculate $x_i = (l_i - d_i)$

The procedure used to generate x_i from received bit sequence bs_i guarantees that all values are losslessly recovered. Figure 3.4 summarizes the algorithm used to decode bit sequence bs_i and flow chart of proposed decompression algorithm is shown in figure 3.5

Experimental Results and Discussions

4.1 Performance Metrics

Depending on the nature of the application there are various criteria to measure the performance of a compression algorithm.

Compression ratio

Compression ratio [27], also known as compression power, is used to quantify the reduction in data-representation size produced by data compression algorithm. It is the ratio between the size of the compressed data and the size of the uncompressed data and is defined as:

$$\text{comp_ratio} = \frac{\text{comp_size}}{\text{orig_size}}$$

where, `comp_size` and `orig_size` are the size of the compressed and the uncompressed bit stream respectively. The smaller value of compression ratio the better the data compression algorithm.

Compression factor

Compression factor [27] is the inverse of the compression ratio. That is the ratio between the size of uncompressed data and the size of compressed data and is defined as:

$$\text{comp_factor} = \frac{\text{orig_size}}{\text{comp_size}}$$

where, `comp_size` and `orig_size` are the size of the compressed and the uncompressed bit stream respectively. Compression factor should be more for better performance of the algorithm.

Saving percentage

Saving percentage [27] is the reduction in size of relative to the uncompressed size. It calculates the shrinkage of the source data as a percentage and is defined as:

$$\text{saving_percentage} = \left(1 - \frac{\text{comp_size}}{\text{orig_size}}\right) * 100$$

where, `comp_size` and `orig_size` are the size of the compressed and the uncompressed bit stream respectively. For better performance of the algorithm, it should be more.

4.2 Simulation Environment

To evaluate the performance of proposed data compression algorithm, we used TOSSIM [33], a discrete event simulator for TinyOS sensor networks.

Instead of compiling a TinyOS application for a mote, users can compile it into the TOSSIM framework, which runs on a PC. This allows users to debug, test, and analyze algorithms in a controlled and repeatable environment. As TOSSIM runs on a PC, users can examine their TinyOS code using debuggers and other development tools. TOSSIM builds directly from TinyOS code. To simulate a protocol or system, there is a need to must write a TinyOS implementation of it. TOSSIM's primary goal is to provide a high fidelity simulation of TinyOS applications. For this reason, it focuses on simulating TinyOS and its execution, rather than simulating the real world. While TOSSIM can be used to understand the causes of behavior observed in the real world, it does not capture all of them, and should not be used for absolute evaluations. TOSSIM does not model power draw or energy consumption. After a simulation is run, a user can apply an energy or power model to these transitions,

calculating overall energy consumption. Because TOSSIM does not model CPU execution time, it cannot easily provide accurate information for calculating CPU energy consumption.

In order to test the performance of the proposed data compression algorithm with the Mercelloni's et al. algorithm, the algorithms are simulated and tested with a set of data samples. Performances are evaluated by computing the above mentioned performance metrics.

Consider samples acquired by a sensor node are generated randomly every 2 minutes and the probabilities decrease with the increase of the values. Total numbers of samples collected by a sensor node in 24 hours or one day are 720 samples. Considering that uncompressed samples are normally represented by 16-bit unsigned integers, the original size of uncompressed data for 720 samples = 11520 bits (= 720×16). Different numbers of samples are taken such as 720 samples, 1440 samples, 2160 samples, 2880 samples, 3600 samples, 4320 samples, 5040 samples, 5760 samples, 6480 samples, 7200 samples to measure the performance of algorithms.

4.3 Results and Discussions

We have used the performance metrics given in section 4.1 for evaluating the performance of the proposed data compression algorithm and the Mercelloni's et al. algorithm [26]. We have selected the Mercelloni's et al. data compression algorithm to comparing the performance of proposed data compression algorithm because Mercelloni's et al. data compression algorithm have good compression ratio and saving percentage other than existing data compression algorithms. The results of the algorithms are shown in table 4.1 and table 4.2.

Table 4.1 Results of Mercelloni's et al. Algorithm

No. of Days	No. of Samples	Total Uncompressed Data in bits	Total Compressed Data in bits	Compression Ratio	Compression Factor	Saving Percentage
1	720	11520	3748	0.325347	3.073639	67.47
2	1440	23040	7621	0.330772	3.023225	66.92
3	2160	34560	11514	0.333159	3.001563	66.68
4	2880	46080	15688	0.340451	2.937276	65.95
5	3600	57600	19912	0.345694	2.892728	65.43
6	4320	69120	23833	0.344806	2.900180	65.52
7	5040	80640	27742	0.344022	2.906784	65.60
8	5760	92160	31327	0.339919	2.941871	65.96
9	6480	103680	35572	0.343094	2.914652	65.69
10	7200	115200	39330	0.341406	2.929062	65.86

Table 4.2 Results of Proposed Data Compression Algorithm

No. of Days	No. of Samples	Total Uncompressed Data in bits	Total Compressed Data in bits	Compression Ratio	Compression Factor	Saving Percentage
1	720	11520	2990	0.259459	3.852843	74.05
2	1440	23040	6027	0.261589	3.822797	73.84
3	2160	34560	9235	0.267216	3.742285	73.27
4	2880	46080	12245	0.265734	3.763169	73.42
5	3600	57600	15344	0.266389	3.753910	73.36
6	4320	69120	18491	0.267520	3.738035	73.25
7	5040	80640	21722	0.269370	3.712365	73.06
8	5760	92160	24622	0.267166	3.742994	73.28
9	6480	103680	27827	0.268393	3.725878	73.16
10	7200	115200	30639	0.265964	3.759914	73.40

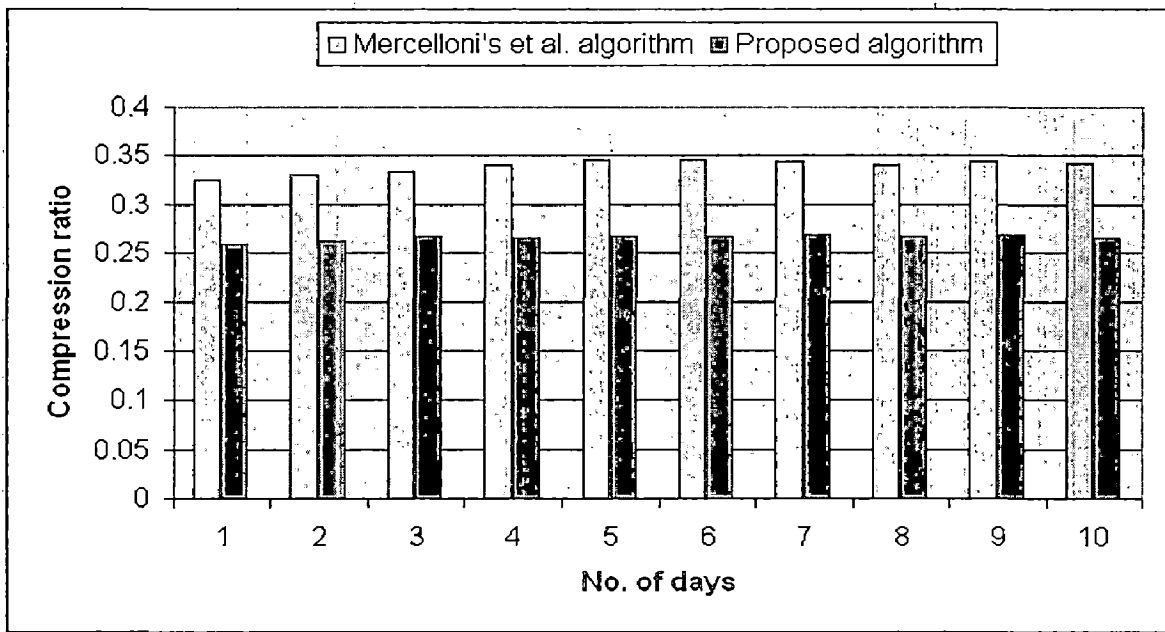


Figure 4.1 Comparing the performance of algorithms on metric compression ratio

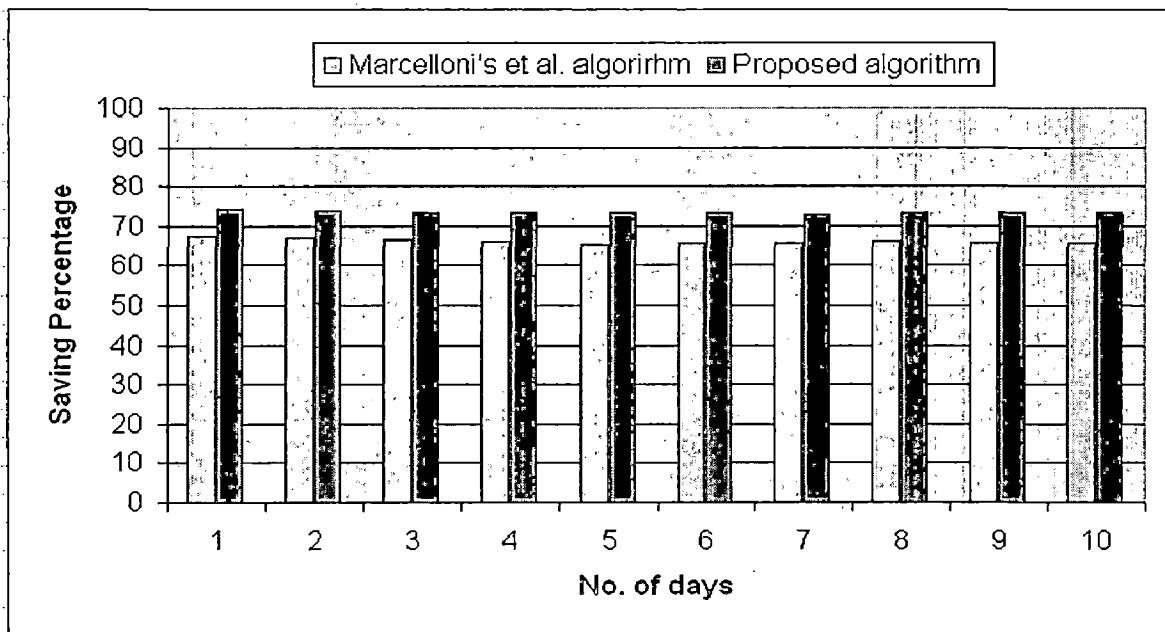


Figure 4.2 Comparing the performance of algorithms on metric saving percentage

Figure 4.1 and figure 4.2 shows the comparison of the proposed data compression algorithm and Merrelloni's et al. algorithm on metric compression ratio and metric saving percentage. The compression ratio of proposed data compression algorithm (0.26) is less than the Merrelloni's et al. algorithm (0.34) which means proposed algorithm compresses more data than Merrelloni's et al. algorithm. Proposed data compression algorithm also reduces memory requirement by about 73.3 percent in comparison to Merrelloni's et al. algorithm which only reduces about 65.8 percent.

Conclusions

5.1 Conclusions

Minimizing energy consumption and reducing memory is a key requirement in the design of sensor network protocols and algorithms. Since processing data consumes much less power than transmitting data in wireless medium, it is effective to apply data compression before transmitting data in order to reduce total power consumption and storage by a sensor node.

In this dissertation, we have proposed a data compression algorithm to achieve better compression ratio than other existing data compression algorithms. The proposed compression algorithm is lossless and particularly suited to the reduced storage and computational resources of a wireless sensor network node. The performance of proposed algorithm is evaluated and compared with Merrelloni's et al. algorithm on various metrics such as compression ratio, compression factor and saving percentage. The algorithms are simulated on TOSSIM.

5.2 Scope for Future Work

In the future, this work can be extended in following ways:

- This work can be implemented on actual mote like mica mote.
- The performance of algorithm may be evaluated on mica mote for different metrics.
- The compression algorithm may be extended to compress the image data as well.

References

- [1] Akyildiz, I.F.; Weilian Su; Sankarasubramaniam, Y.; Cayirci, E., "A survey on sensor networks," *Communications Magazine, IEEE* , vol.40, no.8, pp. 102- 114, Aug 2002.
- [2] G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella, "How to prolong the lifetime of wireless sensor networks," in *Mobile Ad Hoc and Pervasive Communications*, M. Denko and L. Yang, eds. Valencia, CA: American Scientific Publishers, to be published.
- [3] Carlos de Morais Cordeiro, Dharma Prakash Agarwal, "Ad-hoc and sensor networks: Theory and Applications," *World Scientific Publishing Company*, 2006, ISBN: 981-256-681-3
- [4] Jason Lester Hill, "System Architecture for wireless sensor networks," Ph.D. thesis, University of California, Berkley, pp. 4, 2003
- [5] K. C. Barr and K. Asanovi'c, "Energy-aware lossless data compression." *ACM Transactions on Computer Systems (TOCS)*, vol. 24, no. 3, pp. 250–291, August 2006.
- [6] Kazem Sohraby, Daniel Minoli, Taeab Znati, "Wireless Sensor Networks: Technology, Protocols, and Applications," *Wiley-Interscience, A John Wiley & Sons Inc. Publication*, 2007, ISBN: 978-0-471-74300-2
- [7] Arampatzis, Th.; Lygeros, J.; Manesis, S., "A Survey of Applications of Wireless Sensors and Wireless Sensor Networks," *Proceedings of the 2005 IEEE International Symposium on, Mediterrean Conference on Control and Automation*, pp.719-724, 27-29 June 2005
- [8] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks Elsevier* 38 (2002), 393–422. 2002
- [9] Jennifer Yick, Biswanath Mukherjee, Dipak Ghosal, "Wireless sensor network survey," *Computer Networks Elsevier* 52 (2008), 2292–2330, 2008.

- [10] Khalid Sayood, "Introduction to Data Compression," *Elsevier Inc. Publication*, 2006, ISBN-13: 978-0-12-620862-7.
- [11] David Salomon, "Data Compression: The Complete Reference," *Springer-Verlag New York, Inc. Publication*, 2004, ISBN 0-387-40697-2.
- [12] J. Kusuma, L. Doherty, and K. Ramchandran, "Distributed Compression for Sensor Networks," *In Proceedings of 2001 International Conference on Image Processing*, October 2001.
- [13] S. S. Pradhan, J. Kusuma, and K. Ramchandran, "Distributed Compression in a Dense Microsensor Network," *IEEE Signal Processing Magazine*, Volume: 19, Issue: 2, pp. 51-60, March 2002.
- [14] D. Petrovic, R. C. Shah, K. Ramchandran, and J. Rabaey, "Data Funneling: Routing with Aggregation and Compression for Wireless Sensor Networks," *In Proceedings of First IEEE International Workshop on Sensor Network Protocols and Applications*, May 2003.
- [15] T. Arici, B. Gedik, Y. Altunbasak, and L. Liu, "PINCO: a Pipelined In-Network Compression Scheme for Data Collection in Wireless Sensor Networks," *In Proceedings of 12th International Conference on Computer Communications and Networks*, October 2003.
- [16] E. Magli, M. Mancin, and L. Merello, "Low-Complexity Video Compression for Wireless Sensor Networks," *In Proceedings of 2003 International Conference on Multimedia and Expo*, July 2003.
- [17] M. Hans and R. W. Schafer, "Lossless compression of digital audio," *IEEE Signal Processing Magazine*, vol.18, no.4, pp. 21–32, July 2001.
- [18] Y. Zhang and J. Li, "Efficient seismic response data storage and transmission using ARX model-based sensor data compression algorithm," *Earthquake Engineering and Structural Dynamics*, vol. 35, pp. 781–788, 2006.
- [19] C. M. Sadler and M. Martonosi, "Data Compression Algorithms for Energy-Constrained Devices in Delay Tolerant Networks," in *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys)*, 2006.
- [20] N. Tsiftes, A. Dunkels, and T. Voigt, "Efficient Sensor Network Reprogramming through Compression of Executable Modules," in *Proceedings of the 5th Annual*

- IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, 2008.
- [21] H. Ju and L. Cui, "EasiPC: A Packet Compression Mechanism for Embedded WSN," in *Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2005.
- [22] A. Reinhardt, M. Hollick, and R. Steinmetz, "Stream-oriented Lossless Packet Compression in Wireless Sensor Networks," in *Proceedings of the Sixth Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, 2009.
- [23] Capo-Chichi, E.P.; Guyennet, H.; Friedt, J.-M.; , "K-RLE: A New Data Compression Algorithm for Wireless Sensor Network," *Third International Conference on Sensor Technologies and Applications (SENSORCOMM '09)*, pp.502-507, 18-23 June 2009
- [24] Zhou Yan-li; Fan Xiao-ping; Liu Shao-qiang; Xiong Zhe-yuan;, "Improved LZW algorithm of lossless data compression for WSN," *Third IEEE International Conference on Computer Science and Information Technology (ICCSIT-2010)*, vol.4, no., pp.523-527, 9-11 July 2010.
- [25] Dolfus, K.; Braun, T.;; "An evaluation of compression schemes for wireless networks," *International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT-2010)*, pp.1183-1188, 18-20 Oct. 2010.
- [26] Francesco Marcelloni and Massimo Vecchio, "A Simple Algorithm for Data Compression in Wireless Sensor Networks," *IEEE Communications Letters*, Vol. 12, No. 6, June 2008.
- [27] S. R. Kodituwakku, U. S. Amarasinghe, "Comparison of Lossless Data Compression Algorithms for Text Data," *Indian Journal of Computer Science and Engineering (IJCSE), Engg. Journals Publications*, Vol. 1, No. 4, pp. 416-425, 2007.
- [28] Ashish K. Maurya, Dinesh Singh; "Simulation based Performance Comparison of AODV, FSR and ZRP Routing Protocols in MANET," *International Journal of Computer Application*, Foundation of Computer Science, New York, vol. 12, no. 2, November 2010.

- [29] Ashish K. Maurya, Dinesh Singh, Anil K. Sarje, "Performance Comparison of DSR, OLSR and FSR Routing Protocols in MANET using Random Waypoint Mobility Model," *IEEE International Conference on Network communication and Computer (ICNCC-2011)*, New Delhi, India, 19-20 March, 2011.
- [30] Ashish K. Maurya, Dinesh Singh, Anil K. Sarje, "Comparative Performance Analysis of LANMAR, LAR1, DYMO and ZRP Routing Protocols in MANET using Random Waypoint Mobility Model," *IEEE International Conference on Network communication and Systems (ICNCS-2011)*, Kanyakumari, India, 8-10 April, 2011.
- [31] Phillip Levis, "TinyOS Programming," June 28, 2006
- [32] David Gay, Philip Levis, David Culler, Eric Brewer, "nesC 1.2 Language Reference Manual," August 2005.
- [33] Philip Levis and Nelson Lee, "TOSSIM: A Simulator for TinyOS Networks," September 17, 2003.

List of Publications

- [1] Ashish K. Maurya, Anil K. Sarje, "Median Predictor based Data Compression Algorithm for Wireless Sensor Network," *International Conference on Computer Science and Informatics (ICCSI-2011)*, Bhubaneswar, India, 19-20 June, 2011 (Accepted).

Introduction to TinyOS

A.1 Introduction

TinyOS [31] is a free and open source component-based operating system and platform targeting wireless sensor networks (WSNs). TinyOS is an embedded operating system written in the nesC programming language as a set of cooperating tasks and processes. It is intended to be incorporated into smartdust. TinyOS began as a project at UC Berkeley as part of the DARPA NEST program. It has since grown to involve thousands of academic and commercial developers and users worldwide.

TinyOS have following features:

- Conserving resources
- No file system
- No dynamic memory allocation
- No memory protection
- Very simple task model
- Minimal device and networking abstractions
- Application and OS are coupled—composed into one image

A.2 Programming Model

nesC (network embedded system C) [32] is a component-based C dialect. It is a static language which has no heap, no function pointers and no any dynamic

memory allocation. In some ways, nesC components are similar to objects. For example, they encapsulate state and couple state with functionality. The principal distinction lies in their naming scope. Unlike C++ and Java objects, which refer to functions and variables in a global namespace, nesC components use a purely local namespace. This means that in addition to declaring the functions that it implements, a component must also declare the functions that it calls. The name that a component uses to call these functions is completely local: the name it references does not have to be the same that implements the function. When a component A declares that it calls a function B, it is essentially introducing the name A.B into a global namespace. A different component, C, that calls a function B introduces C.B into the global namespace. Even though both A and C refer to the function B, they might be referring to completely different implementations. Every component has a specification, a code block that declares the functions it provides (implements) and the functions that it uses (calls).

TinyOS Components:

Programs are built out of components in which each component is specified by an interface which provides “hooks” for wiring components together. Components are statically wired together based on their interfaces to increase runtime efficiency. Components **use** and **provide** interfaces, commands, and events, specified by a component’s interface and the word “interface” has two meanings in TinyOS. Components implement the events they use and the commands they provide:

Components	Commands	Events
Use	Can Call	Must Implement
Provide	Must Implement	Can Signal

There are two types of components in nesC:

1. **Modules:** Implements the component specification (interfaces) with application code.

2. Configurations: Wires components together i.e. how components are wired together.

Configurations connect the declarations of different components, while modules define functions and allocate state. A component does not care if another component is a module or configuration and a component may be composed of other components. A configuration states must name which components it is wiring with the components keyword. Any number of component names can follow components, and their order does not matter. A configuration can have multiple components statements. A configuration must name a component before it wires it. A module contains C-like code while configuration doesn't use C-like code.

A.3 Concurrency Model

There are two types of execution contexts:

- 1. Tasks:** Tasks are longer running jobs having time flexibility and uses (currently) simple FIFO scheduling. Tasks are atomic with respect to other tasks, i.e., single-threaded but can be preempted by events. A task is always posted for later execution; control returns to poster immediately. Scheduler supports a bounded queue of pending tasks i.e. node sleeps when the queue is empty. For simplicity, tasks don't take arguments and don't return values.
- 2. Events:** Events (an overloaded term) are more precisely, hardware interrupt handlers. Time is a critical factor for events; events have shortened duration as much as possible by issuing tasks for later execution. Events follow LIFO semantics; can preempt tasks and earlier events.