A RECONFIGURABLE SECURED WIRELESS SENSOR NETWORKS WITH XBEES AND ARDUINO

MOLLAH AHMED SHORIF
Bachelor of Science, Islamic University of Technology, 2004

A Thesis
Submitted to the School of Graduate Studies
of the University of Lethbridge
in Partial Fulfillment of the
Requirements for the Degree

MASTER OF SCIENCE

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

A RECONFIGURABLE SECURED WIRELESS SENSOR NETWORKS WITH XBEES AND ARDUINO

MOLLAH AHMED SHORIF

Date of Defence: December 19, 2014

Dr. Hua Li Supervisor	Associate Professor	Ph.D.
Dr. Gongbing Shan Thesis Examination Committee Member	Professor	Ph.D.
Dr. Wendy Osborn Thesis Examination Committee Member	Associate Professor	Ph.D.
Dr. Howard Cheng Chair, Thesis Examination Committee	Associate Professor	Ph.D.

Dedication

To my family.

Abstract

Wireless Sensor Networks (WSN) consists of many small sensor nodes that utilizes the sensors to collect sensor data and sends the data to the base station. Security in WSN has been a subject well researched for last few years and still many related researches are going on. Authentication with reconfiguration capability is a very important trait for the security of a network. The overall objective of this research is to develop a wireless sensor network which is reconfigurable and secure. I propose a reconfigurable authentication mechanism with on-line key changing capability using encryption. I discuss the types of attacks the proposed authentication can prevent and describe step by step implementation of this proposal in hardware. I compare AES with RC4 algorithm in my network setup and show how they react to RSSI, Latency and Throughput in variable parameters.

Acknowledgments

My sincere gratitude goes to my supervisor, Dr. Hua Li for his invaluable support, guidance and suggestion throughout my M.Sc. program. Without his supervision and continuous help, this thesis would not have been possible.

I would like to thank Dr. Gongbing Shan for his wisdom and insightful advices during our meetings that certainly helped me to shape up my research objective. I express my sincere gratitude to Dr. Wendy Osborn for taking the time to review my thesis and giving valuable feedback.

Lastly, I am very thankful to all my lab members for their support and help to make this journey a little easier.

Contents

Approval/Signature Page **ii**

Contents **vi**

List of Tables **viii**

List of Figures **ix**

1 Introduction **1**

1.1 Background 1
1.2 Motivation of the Thesis 2
1.3 Research Objectives 3
1.4 Main Contribution 4
1.5 Thesis Organization 5

2 Wireless Sensor Network and the Security **6**

2.1 Introduction to WSNs 6
2.2 Application of WSNs 7
2.3 WSNs topologies 8
2.3.1 Peer-to-Peer networks [8] 8
2.3.2 Star networks [8] 8
2.3.3 Tree networks [8] 9
2.3.4 Mesh networks [8] 10
2.4 Attacks on WSNs 10
2.4.1 Physical layer attacks 11
2.4.2 Data link layer attacks 12
2.4.3 Network layer attacks 12
2.4.4 Transport layer attacks 13
2.4.5 Application layer attacks 13
2.5 Authentication 13
2.5.1 User Authentication 13
2.5.2 Node Authentication 14
2.5.3 Data Authentication 14
2.6 Literature review 14

3 Proposed Secured Reconfigurable WSNs Architecture 19
3.1 Hardware components 20
3.1.1 Arduino UNO 20
3.1.2 XBee and its standards 22
3.1.3 XBee explorer 24
3.1.4 XBee shield 25
3.1.5 Temperature sensor 25
3.1.6 Humidity sensor 26
3.1.7 Different Wireless Sensor Networks modules 26
3.1.8 Reason for using XBee and Arduino UNO module 30
3.2 Software used 32
3.2.1 Matlab 32
3.2.2 Arduino UNO IDE 33
3.2.3 X-CTU 34
3.2.4 Docklight 35
3.3 Encryption techniques 36
3.3.1 RC4 36
3.3.2 AES 37

4 Key Exchange For The Authentication 38
4.1 System architecture 38
4.1.1 Circuits and wiring 38
4.1.2 Overview of the key exchange and authentication 40
4.2 Communication between nodes with encryption 41
4.2.1 Communication between two individual XBee and Arduino UNO . 41
4.2.2 Encryption and decryption in two XBee and Arduino UNO nodes . 42
4.3 Communication with authentication using encryption 44
4.3.1 Duplex communication between two nodes with sensor data 44
4.3.2 Input command in serial monitor of Arduino UNO 45
4.3.3 Authentication in XBee and Arduino UNO node 48
4.3.4 Authentication in Matlab 49
4.3.5 Receiving sensor data from all the nodes simultaneously 54

5 Performance Analysis and Result 56
5.1 Performance analysis with different feature 56
5.1.1 Received Signal Strength Indicator 56
5.1.2 Latency 60
5.1.3 Throughput 72

6 Conclusion and Future Works 82
6.1 Contribution 82
6.2 Conclusion 84
6.3 Future works 84

List of Tables

2.1 Examples of sensors [1] 7

3.1 Hardware capability of Arduino UNO [5] 21
3.2 Comparison between different WSNs nodes 31

5.1 RSSI values with transmit power for the RC4 57
5.2 RSSI values with transmit power for the AES 58

List of Figures

2.1 Peer to Peer network [8] 8
2.2 Star network [8] 9
2.3 Tree network [8] 9
2.4 Mesh network [8] 10
2.5 Sensor networks protocol stack [9] 11

3.1 Proposed Wireless Sensor Network architecture 19
3.2 Arduino UNO [5] 21
3.3 XBee 802.15.4 module [6] 23
3.4 XBee explorer module [29] 24
3.5 XBee shield module [5] 25
3.6 Temperature sensor [30] 26
3.7 Humidity sensor [31] 26
3.8 TelosB module [32] 27
3.9 Mica2 module [33] 28
3.10 IRIS module [34] 28
3.11 XBee and Arduino UNO module [5] [6] 29
3.12 Waspmote module [35] 29
3.13 Matlab GUI layout 32
3.14 Arduino UNO IDE [5] 34
3.15 X-CTU panel [36] 35
3.16 Docklight panel [37] 36

4.1 Circuit architecture with temperature sensor 39
4.2 Circuit architecture with humidity sensor 39
4.3 Overview of the authentication concept and procedure 40
4.4 Communication between two Nodes 42
4.5 Encryption and decryption in two XBee and Arduino UNO nodes 43
4.6 Duplex communication between two nodes with sensor data 45
4.7 Serial Monitor of Arduino UNO 46
4.8 Engineered Arduino UNO with wires 48
4.9 Implementing the authentication in XBee and Arduino UNO node 49
4.10 Key exchange and authentication in Matlab 50
4.11 A working Matlab GUI with different parts 51
4.12 Authentication concept with Mesh networking 53
4.13 GUI shows temperature sensor data shown in Node 1, 2 and 3 windows and humidity sensor data in Node 4 window simultaneously 55

5.1 RSSI values with transmit power for the RC4 57
5.2 RSSI values with transmit power for the AES 58
5.3 RSSI values with transmit power for RC4 and AES in 9600 baud rate . . . 59
5.4 Using mesh network for performance analysis 61
5.5 Latency values in Node 1 for baud rate 9600 62
5.6 Latency values in Node 2 for baud rate 9600 63
5.7 Latency values in Node 1 for baud rate 19200 64
5.8 Latency values for Node 2 for baud rate 19200 64
5.9 Latency values in Node 1 for baud rate 38400 65
5.10 Latency values in Node 2 for baud rate 38400 65
5.11 Latency values in Node 1 for baud rate 56700 66
5.12 Latency values in Node 2 for baud rate 56700 66
5.13 Latency values in Node 3 for baud rate 9600 67
5.14 Latency values in Node 4 for baud rate 9600 67
5.15 Latency values in Node 3 for baud rate 19200 68
5.16 Latency values in Node 4 for baud rate 19200 68
5.17 Latency values in Node 3 for baud rate 38400 69
5.18 Latency values in Node 4 for baud rate 38400 69
5.19 Latency values in Node 3 for baud rate 57600 70
5.20 Latency values in Node 4 for baud rate 57600 70
5.21 Throughput values in Node 1 for baud rate 9600 72
5.22 Throughput values in Node 2 for baud rate 9600 73
5.23 Throughput values in Node 1 for baud rate 19200 74
5.24 Throughput values in Node 2 for baud rate 19200 74
5.25 Throughput values in Node 1 for baud rate 38400 75
5.26 Throughput values in Node 2 for baud rate 38400 75
5.27 Throughput values in Node 1 for baud rate 56700 76
5.28 Throughput values in Node 2 for baud rate 56700 76
5.29 Throughput values in Node 3 for baud rate 9600 77
5.30 Throughput values in Node 4 for baud rate 9600 77
5.31 Throughput values in Node 3 for baud rate 19200 78
5.32 Throughput values in Node 4 for baud rate 19200 78
5.33 Throughput values in Node 3 for baud rate 38400 79
5.34 Throughput values in Node 4 for baud rate 38400 79
5.35 Throughput values in Node 3 for baud rate 56700 80
5.36 Throughput values in Node 4 for baud rate 56700 80

Chapter 1

Introduction

Information is important, especially when it deals with situations such as health condition of patients, tracking enemies for militaries and providing water level information for flood prevention. These are only a few of the major examples where we can take advantage of Wireless Sensor Networks (WSNs) to receive information. There are many other areas too where WSNs is used to make life easier. We are seeing more deployment of WSNs from the health sector to industry, from home automation to national security. Researchers are now investing their resources for their further use in critical situations, by proposing a standard platform that is more robust, resilient and smart. In addition, newer concepts are developed to make it more reliable, secure and user friendly.

1.1 Background

WSNs has come a long way from its initial days - the days from the Distributed Sensor Networks (DSN) program in the early 1980s by the Defense Advanced Research Projects Agency (DARPA) [1]. However, DSNs were not a focus of research until the 2000s. Previously, the University of California at Berkeley focused on the design of extremely small sensor nodes. It was a part of a project called Smart Dust [2], with a goal to incorporate complete sensor systems into tiny devices. The Berkeley Wireless Research Center (BWRC) introduced the PicoRadio project that focused on introducing sensor devices which need low-power [3]. The University of California collaborated with the Rockwell Science Center to propose the idea of Wireless Integrated Network Sensor (WINS) [4]. [1]

1.2 Motivation of the Thesis

In WSNs, the sensor nodes are distributed in a wide area. The sensor node is a combination of a number of devices which senses and processes the data, and transmits or receives it to/from another sensor node. This WSN can be deployed in any type of terrain and needs seldom to no intervention by the user. At the very beginning, these sensor nodes were used to collect trivial data such as agricultural soil condition, environment pollution or air quality, and industrial data logging. But over the course of time it started covering more areas that are more critical. Some examples include health monitoring, area monitoring for military, natural disaster prevention, and forest fire detection. When WSNs started to be used in these critical situations, the requirements of secure and authenticity increased. Researchers started focusing on data confidentiality and authenticity. Although many new security concepts have been proposed, very few of them are practically implemented. So there is a gap between conceptually proposed schemes and practical use. Some commercially produced devices are used in industry but they are not open to users. That means, the users are not allowed to change anything in that device, and the device must be used as it is in a package. There are some other producers of sensor nodes, where the devices are made solely for performing a particular job. If a user wants to add something additional, he/she has to buy another device. Therefore, I chose to go for something cheaper and open source. I wanted to use a device which would give the flexibility to add any type of sensor and also allow me to utilize its additional functionality. As an open source processor, Arduino UNO [5] has already put its mark on automation application and XBee [6] is potentially another very small, cheap, but powerful addition to it. I analyzed other factors such as processor capability and range of Radio Frequency (RF), and found very little difference from the conventional costly alternatives. A detail discussion is presented in Chapter 3. Another good reason for choosing these devices is that, they can be used with additional devices such as Global Positioning System (GPS) or WiFi, and can be programmed independently with any sensor available in the market. In spite of their powerful capability, XBee and

Arduino UNO have not been widely used in the WSNs field and to my knowledge no one has tried to propose security and authentication for it. Therefore, I propose and develop a novel authentication concept and implement it with a complete encryption scheme in XBee and Arduino UNO to minimize this limitation.

1.3 Research Objectives

My objective for this thesis is to propose a novel approach for security via authentication, reconfigurable key distribution, and implement this authentication and encryption in open source equipment. Then a Wireless Sensor Networks (WSNs) will be formed using the open source equipment. User will use Graphical User Interface (GUI) to control the network.

Encryption, authentication and key distribution are very important parts of network security. Encryption is a technique to convert data in a format, called a ciphertext, that can not be easily understood by unauthorized people. Authentication is a process to identify an individual as authentic in a network. Key distribution technique refers to the means of delivering a key to two parties who wish to exchange data without allowing others to see the key.

In WSNs, the sensor nodes are distributed in a wide area. This area can be very remote or impractical to reach for the user. The sensor nodes are used to connect and obtain data from each sensor. After reading the sensor data, these sensor nodes send back the sensor data through other sensor nodes to the base station. When transmitting crucial data in this manner, there are always major risks. For example, suppose, the sensor nodes are in enemy territory and carrying very sensitive data. Naturally, the adversary wants to intercept the communication and will try to read the data or information. So, to make sure that the data can not be compromised, these sensor nodes must be equipped with latest security measures. There are so many attacks in WSN that security needs to be addressed. The problem to work with these counter measures is that, they are very sophisticated and they demand many requirements, some of which are not possible to fulfill by the sensor nodes. One of the biggest reasons of this shortcoming is resource availability. The sensor nodes are very small in physical size, processor and memory.

Though many security measures are introduced and suggested for WSNs, not many practical implementations have been done. Most of the encryption or authentication strategies have been introduced hypothetically. Among these suggestions, a few were implemented in simulation or on the sensor nodes which are made by specialized manufacturers. So, there is still a big gap between the number of proposed security strategies and practical implementations in use. So, I decide to work with the cheapest possible equipment and try to see how it performs. So, my objective for this thesis is to propose a novel approach for security via authentication, and key distribution, and implement this authentication and encryption in open source equipment. Then, for the next step I want to check the viability of this concept by implementing it in a larger area with several sensor nodes. I want to provide the best possible control to the user with a user friendly graphical unit. Using this unit, the user can control or change the configuration of a sensor node that is far away from them. This WSN works simultaneously with the Internet, and multiple users can access the sensor data from different places.

1.4 Main Contribution

The major contribution of this work is the ability to reconfigure the key change and exchange in real time by the user. This approach uses key distribution technique using encryption. The encryption algorithm can be changed as the requirement of application as well. With this facility, some major WSNs attacks can be prevented. Another contribution is introducing an effective authentication concept and its implementation in hardware. By proposing and implementing the work in the chosen tools, future researchers may try to implement other capabilities in open source hardware. Therefore this hardware may obtain efficient remodeling and redesigning. Finally, I have performed a number of performance checks which will present a practical idea on where this research work stands.

1.5 Thesis Organization

The remainder of the thesis is organized as follows. In Chapter 2, an introduction of WSNs and their application are presented. This chapter also summarizes research on WSNs attacks and authentication. This chapter also presents the kind of attacks that my proposed protocol can prevent. In Chapter 3, a proposed architecture for secured reconfigurable WSNs is shown. The architecture is divided into three parts: Sensor, Receiver and Internet. The Receiver part controls the communication and sends requests with key to Sensor part for sensor data. The sensor node sends the sensor data encrypted by the new key. Later, this data can be saved in a file and be used in the Internet. In this chapter, I also talk about different parts of the hardware and software used in a small capacity, which includes transceiver, micro-controller and sensors. Chapter 4 describes thoroughly the procedure of the proposed key exchange scheme and the different steps of implementing the proposed architecture for reconfigurable WSNs. Outcome, findings and problems are discussed in depth for simple understanding of the work. Performance analysis and results are shown for different encryption types using the proposed authentication in Chapter 5. Chapter 6 concludes the thesis work by describing the major contribution of this thesis and presenting a future plan.

Chapter 2

Wireless Sensor Network and the Security

Fundamentals of Wireless Sensor Networks (WSNs) are introduced here in this chapter. I discuss different applications of WSNs as well as the challenges it faces. I then talk about attacks on WSNs. In addition, authentication schemes and their application with encryption are thoroughly discussed. Finally, a literature review on key distribution, and authentication in WSNs is presented.

2.1 Introduction to WSNs

Wireless Sensor Networks (WSNs) nodes contain a small memory, processor and transceiver unit. Since they are distributed in a wide area, they can be placed anywhere that might be inaccessible for users. So, they must have the capacity to collect the data by connected sensors, process the data and send it to the main part of the network called the base station. The base station can be another sensor device but usually with more memory, a better processor and a stronger transceiver. This base station connects wirelessly with other low capacity sensor nodes and reads data from them. In a complex situation, this base station can send a request for data first and then collect, store and process it into a desired format and display it to the user.

WSNs can work with any number of sensors based on their type of sensing. They can be classified for their different sensing ability. As an example, Table 2.1 shows some sensors.

Table 2.1: Examples of sensors [1]

Type of Sensors	
Temperature	Flow
Pressure	Position
Optical	Electromagnetic
Acoustic	Chemical
Motion,vibration	Humidity
Radiation	

2.2 Application of WSNs

WSNs can be used in many fields and applications. These applications are quite diverse and not limited to only industrial implementation. They have been used extensively in the health sector, environment quality assurance and preventing military intrusion. In this section we talk about some of the major contributions of WSNs in various applications.

Environmental applications include air quality monitoring, detecting forest fires, tracking and monitoring vegetation. Military applications provide many different services such as monitoring forces, equipment and ammunition, and detection of nuclear, biological and chemical attack. Security monitoring is fundamentally the same as environment monitoring but this does not collect any data. It looks for anomalies and if anything found, then it gives a signal to the user to take necessary action. In health care, WSNs is used extensively to monitor a patient's physiological data, control drug administration and send real time data to those concerned. For seniors, this can be used to monitor fall detection, unconsciousness detection, and exercise monitoring. WSNs in home applications includes monitoring temperature, intrusion control, event-based activity triggering, automatic light switching and more [1] [7].

2.3 WSNs topologies

Along with the advancement in WSNs technologies, major changes happened in network distribution as well. Many forms of networks have been introduced to accomplish different acts and they have different advantages and limitations. Here we summarize some renowned topologies that are used in WSNs [8].

2.3.1 Peer-to-Peer networks [8]

This topology is ad-hoc in nature and supports node to node communication. One node can communicate with any other node directly as long as they are in range [8].

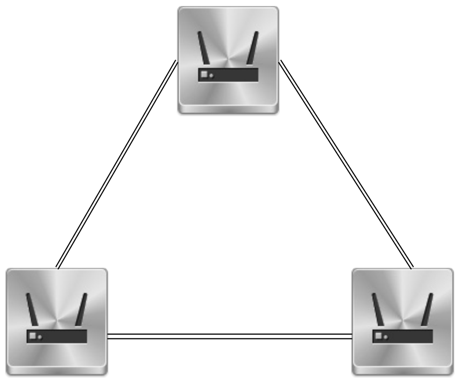


Figure 2.1: Peer to Peer network [8]

If they are not in range, one node can communicate through other nodes as well. The important aspect of this peer to peer network is their self organization and self healing (the capability of recovering failure). This topology can be implemented in industry monitoring or inventory tracking as data passes directly from one node to another in least possible time. Nodes act not only as a receiver or a sender, but also as a router in this kind of network topology. Figure 2.1 shows a peer to peer network [8].

2.3.2 Star networks [8]

Unlike a peer to peer network, in a star network, nodes can not communicate with each other. This network has a coordinator which works as a head node and all other nodes connect with it. So all data should at first go to the coordinator [8]. In a bigger

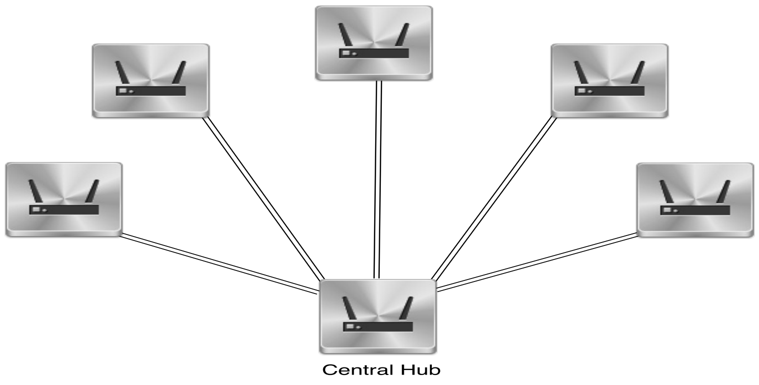


Figure 2.2: Star network [8]

network, the sensor nodes are usually powered by a battery but the coordinator is powered directly by electricity to sustain the demand of communication with other sensor nodes for uninterrupted data. Though sensor nodes connect themselves to the coordinator, they are independent operators [8].

2.3.3 Tree networks [8]

Tree topology is often called a hybrid star and peer-to-peer networking topology. A central hub connects with other nodes as well as a root node and thus making the situation like tree topology. Nodes connect with a central hub and then from the central hub, a signal goes to the root node. So when a new node wants to join this network it has to connect with central hub first [8].

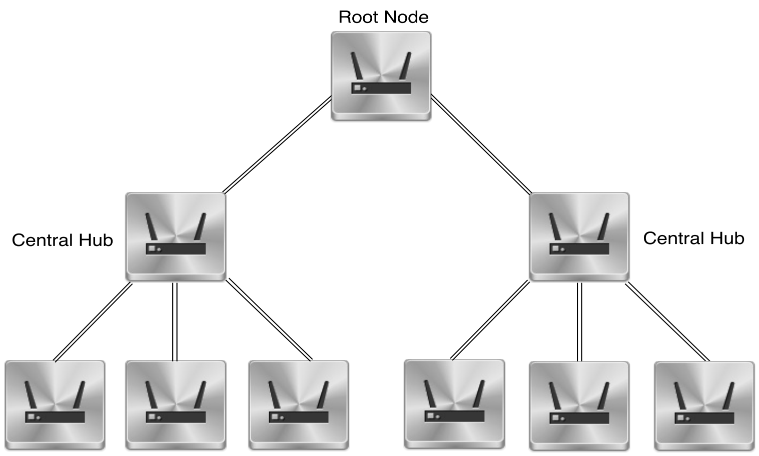


Figure 2.3: Tree network [8]

Figure 2.3 shows an example of a tree network topology for easier understanding of the concept.

2.3.4 Mesh networks [8]

A mesh network allows the nodes to communicate from one end to another by hopping through other nodes, which makes it self healing. Each node is connected with other available nodes in same area. A mesh topology can be very expensive due to its inclusive structure. It is the most complex structure among the network topologies [8].

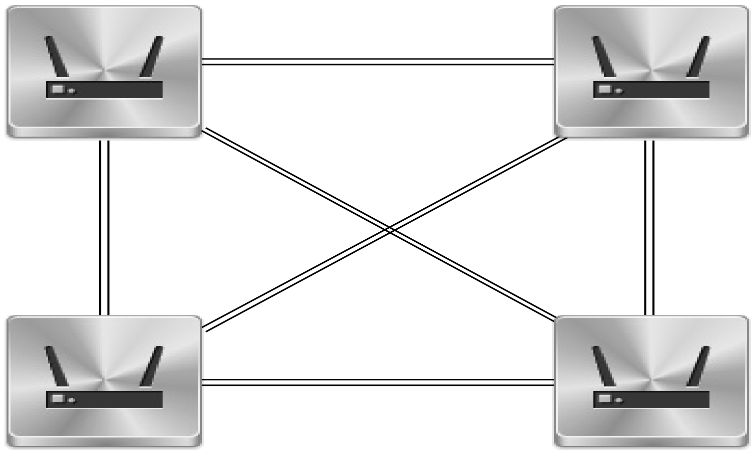


Figure 2.4: Mesh network [8]

2.4 Attacks on WSNs

Similar to the other wireless networks, WSNs is prone to attacks in different ways. The attacks in WSNs are mostly related to the network layers. The WSNs layers are similar to the OSI layers. We call the layers as WSNs OSI layers because they work similar to the OSI layers. A figure for the WSN OSI protocol stack is shown in Figure 2.5.

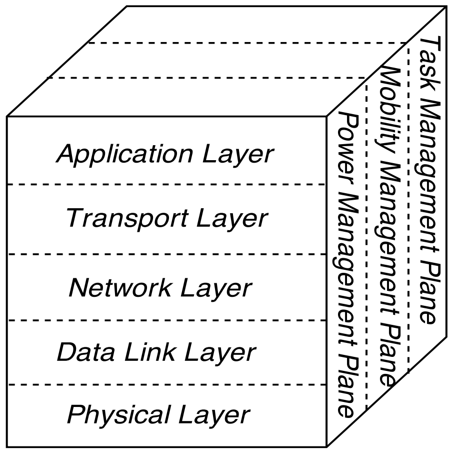


Figure 2.5: Sensor networks protocol stack [9]

We now discuss about the attacks that occur in different WSNs OSI layers [9].

2.4.1 Physical layer attacks

In the physical layer, mostly Jamming and Tampering are evident attacks.

(a) Jamming : Jamming is one of the Denial of Service (DoS) attacks. In this attack, the adversary sends a high energy signal and thus disturbs the operation of the network by frequency interference. An adversary can make one small network completely disable [10].

As a defense, the affected node can inform the other nodes as well as the base station about the attack. So, the base station and other neighbors would break their connections with this particular node. Techniques such as analyzing Received Signal Strength Indicator (RSSI) values, average time required to sense an idle channel or packet delivery ratio can be used to detect the jamming attack and can substantially protect the network from all jammers [10].

(b) Tampering or Destruction : Tampering or destruction is a physical attack on the node or network to extract sensitive information stored in a node. After acquiring the information, an attacker may attack the whole network with the information. To protect a node from this kind of attack, nodes can use tamper resistant packaging or self erasing capability. So, when an adversary attacks, the node either erases its memory or does not allow its information to be read [10].

2.4.2 Data link layer attacks

(a) Unfairness : Exhaustion or collision may cause unfairness in a network. Exhaustion happens when an adversary continuously tries to communicate with one major node making the other nodes halt. Collision happens when an adversary communicates in the same channel of a major node which results an unstable network. Using a small size of packet for communication is the best measure for dealing with this unfairness because one node will not occupy a channel for a longer time in this case [10].

(b) Interrogation : Medium Access Control (MAC) protocol uses Request To Send (RTS) or Clear To Send (CTS) message for minimizing a particular problem in nodes. An adversary uses this extra handshake measure to attack the same network by repetitively requesting these messages and exhausting a node's resource. A strong link layer authentication where a node accepts only from a known node may minimize this kind of attack [10].

2.4.3 Network layer attacks

(a) Hello Flood : A Hello message is sent by a node to its neighbors so that these can be aware of its presence and may find it to be a high quality route. So, in a hello flood attack, an adversary uses a very powerful antenna and sends hello messages to all the nodes. The nodes assume the adversary is another neighbor node and channel their traffic to this fake node. From this, the adversary starts working like a sinkhole attacker. This attack can easily be avoided by bi-directional communication and a key distribution mechanism [10].

(b) Sybil attack : Sybil attacks may affect network in many ways. Sybil nodes, which are malicious and illegitimate nodes, may communicate directly or indirectly with other legitimate nodes. Sybil nodes can fabricate or steal an identity by being assigned a random address or a stolen address respectively. Direct and indirect validation of the node is an effective countermeasure for this attack. In direct validation, a node directly tests other nodes and it accepts validation from another previously validated node in an indirect validation [10].

2.4.4 Transport layer attacks

De-synchronization Attack : An attacker tries to break the synchronization of two communicating nodes. It tries to forge packets with false sequence numbers to one end or the other. This forces the receiver to request for another transmission for full packet recovery. Therefore, these two nodes will be engaged for a long time and will die by energy drainage. Authenticating the entire message from one node to another can defend this attack. By authenticating all the packets, the adversary can not forge any malicious packets [10].

2.4.5 Application layer attacks

Deluge (reprogram) attack : Reprogramming is an advantage to remotely reprogram any node. If this reprogramming is not properly secured, an adversary can attack this portion and take control of the whole network. Authentication is again another way to prevent a network from this attack [10].

2.5 Authentication

Authentication is to determine the identity of an entity, as what it claims to be, in a network. Authentication, in real world, is usually accomplished by a user name and associated password but it can be done in other ways. If we consider the user as an entity, which is trying to access the network and is already known by name and password, the network accepts their credentials right away as authentic. In a WSN, authentication can be done in three distinct ways: 1. User authentication, 2. Node authentication and 3. Data authentication.

2.5.1 User Authentication

User authentication is a mean of identifying the user and verifying that the user is allowed to access some restricted services. It is about making a relationship between the user and the restricted service provider. The service provider asks for user identity to make sure that this is the user with valid credentials. Service provider assumes of the identity as non forgeable and by providing exact identity, user proves him or herself as authentic. This identity can be a password, a secret key or a biometric identification by which, provider connects credential with the authentic user.

2.5.2 Node Authentication

In WSN, nodes are distributed liberally in a wide area. When a new node is added to the already setup network it goes through some processes to be authentic. The nodes are easier to approve as authentic in the case of static WSN but it is more complex when the nodes are mobile. In the case of mobile nodes, it is very necessary to consider changes in nodes' position.

2.5.3 Data Authentication

Data authentication is very similar to node authentication but in this case it deals with the sent or received data. Adversary can insert message in a communication between nodes in WSN. If there is no process to identify the flaw for the data authentication, the nodes do not know about the error in data. To overcome this problem, it is necessary for the nodes to make sure that they know that the data they have received is from authorized source.

2.6 Literature review

In [11] and [12], experiments have been done to compare the performance of Zigbee/802.15.4 with other techniques. [11] shows the reason of preferring 802.15.4 based WSNs network over 802.11 networks. The study shows clear advantage of 802.15.4 over 802.11 in different parameters. [12] describes that Zigbee network is more flexible than the other technique analyzed with parameters such as throughput, delay, and connectivity.

An experiment on user authorization system using XBee module has been shown in [13]. It discussed authorization by user in remote nodes as well. The authorization included RFID

(Radio Frequency IDentification) card, microcontroller, LCD touch screen and XBee module. The node proves its authenticity by the user PIN number, which needs to be given by the user of the remote node. The biggest problem in this concept is presence of user. A WSN is a network where many nodes are spatially distributed in wide areas. Some of the nodes are very difficult to access, as these are used to collect data from the environment and send it to the coordinator frequently. So, the presence of the user putting their PIN into the system before sending the environmental data is impractical.

An authentication performance has been tested in [14]. This performance is done using two workstations and two XBees connected to them to communicate with each other. As two workstations are used for processing work, it is very different from the conventional WSNs. In WSNs nodes with small processor unit are used. So, implementing authentication in the WSN node is very different and more complex than implementing it in two workstations.

While discussing about data aggregation authentication in WSNs, Bhoopathy and Parvathi in [15] describes a secure way for authentication. At first the aggregator sends a message to all the nodes. The nodes sends back an acknowledgement message to the aggregator. After getting the feedback, it selects C number of nodes randomly, where C $\leq n$ (n=total nodes). The aggregator sends a set of values V to those nodes. V consists of IDs and keys. So, as an example, there are number of nodes from A to Z in a network, and A wants to send message to F node via the aggregator. At first it selects few nodes randomly and when A wants to send data, it slices and sends the message to the randomly selected number of nodes while keeping one slice within it. The nodes obtain the slices and decrypt them with stored key. Decrypted slices are then added together by the nodes and sent to the aggregator. The aggregator then obtains data from different sources, decrypts and add them all together. After adding and encrypting, it sends the whole package to the desired sink node. This is an effective procedure as the nodes change every time for carrying data to the aggregator, and the adversary would not be able to know which slice is going to which node. Even if adversary knows and captures one node, it can only read a small portion of the whole message.

Choi and Youn in [16] shows ways for pre-distribution in a multilevel WSN. So, this is not a Key distribution scheme. This is pre-key distribution only. It provides a mechanism to manage a key so that it will be distributed to two nodes. They have emphasized on key generation, not on key distribution. The reason presented is that if they can generate a key so secure, there is no need to worry about anything else. Then, after key generation there will be other steps like key distribution techniques and node authentication, but this paper‘s main idea is to generate a unique key.

For the limited capacity of the sensor nodes, Qiu, Zhou, Baek, Lopez in [17] discuss ways to establish authentication in dynamic wireless sensor networks. So, in this case when one dynamic node wants to communicate with a cluster head, it first sends an encrypted message to the base station with its identity, Message Authentication Code,and a random number. The base station verifies the revocation list when it receives one request message. If the sensor node is admissible then the base station checks the Medium Accesss Control (MAC) message. If the result is deemed acceptable, then the base station generates a session key and sends a message with the session key as well as approval message for the dynamic node.

Recently Piyare and Lee in [18] performed different performance analysis of a Zigbee network using the XBee module. Though no authentication or encryption has been considered in this experiment, RSSI, Throughput, and packet delay indicators have been discussed. In my work, which showed in chapter 5, I evaluated with many of the indicators to show their performance using encryption and authentication in different baud rates.

Though no authentication has been proposed or implemented, [19] has performed a simulation on precision-based agriculture using Zigbee. This application collects different data climatologically and sends it to the main node for analysis. In this paper, only simulation has been discussed but no practical implementation has taken place.

Shinghal and Raina in [20], presented about different parameters for evaluating two major encryption methods: AES and RC4. Though this experiment is done on workstations but the performance analysis opens up the possibility for an alternate to AES when the resources are limited. This is very crucial, as for all the parameters such as Encryption and Decryption time, CPU process time, Memory utilization RC4 outperformed AES. It is a major catalyst to use RC4 as the encryption in small memory and processor nodes to speed up the process as well as keeping more space in available memory.

Not many experiments have been done in a XBee and Arduino UNO node for simple communication, let alone for security implementation and authentication. One of the papers [21] discusses about how XBee and Arduino UNO can be used to transport sensor data from one end to another. It is a simple way to transporting sensor data, from end device to the coordinator which is connected to a PC.

Luo, Zheng, Pan, and Wu in [22] presented performance analysis between RC4, RC5 and other two major encryption methods. The analysis was done considering the memory requirement and execution time requirement, and in both cases, RC4 showed a better result than RC5.

Most of the proposals on key distribution and authentication in WSNs are hypothetical or have been implemented in a device which is industry based. So, a practical implementation of key distribution or authentication is very rare to find.

As in an identity based key management scheme [23], where the coordinator nodes use an ID to authenticate other sensor nodes. A coordinator makes a master key for the sensor nodes after calculating and storing the key on the sensor nodes. So, when the sensor node comes closer or needs communication with the coordinator, the coordinator checks with its database to verify the authenticity of the sensor node.

In [24], Jolly, Kuscu, Kokate, and Younis assumed the gateway nodes as not trustworthy, thus giving the sensor nodes two different keys to communicate. Each gateway stores the key it shares with the sensor nodes additionally with a previous key stored in it. Once it exchanges keys with the other gateway nodes, it erases the previous key and store this new key so that the adversary can not predict the always-changing keys.

In a sybil attack, the adversary tries to introduce itself as a valid node of the same network and then destabilize the network. In my proposed scheme, each node shares a unique symmetric key with a trusted base station or coordinator node, which can be used to verify the node's identity by sending encrypted command. Unfairness occurs during an exhaustion attack in a network [10]. In this case, the adversary continuously communicates with major nodes. In my proposed key exchange protocol, each node only communicates when the base station wants to communicate with it, so it does not respond to a false request for key negotiation generated by a fake node. This eliminates the possibility of exhaustion and unfairness attacks in this network. The same approach can also prevent the Hello flood attacks. In Hello flood attacks, the adversary uses same neighbor approach and lures other node to channel its data towards it [10]. But as in my work, since only the base station communicates with each sensor node, a sensor node does not listen to a malicious node's requests. My proposed work also deals with all forms of authentication. The base station communicates with the sensor node using a secret key that ensures user authentication. If a new senor node needs to be added to the network, the sensor node is pre-installed with a new key that is only known to the base station. So, in this case also, the node authentication is maintained. Data authentication can be ensured by RC4 encryption that has been used in this proposed scheme. The secured data communication by the sensor node ensures the base station that the senor node is also authentic.

Chapter 3

Proposed Secured Reconfigurable WSNs Architecture

A WSN is a combination of hardware and software. Both have a significant role to play in the network. In upcoming sections, I introduce the hardware used in my thesis work as well as the software. Though individually they are of less significance, together they can make a great impact on the performance of any network. To keep the network simple, I choose to work with a limited number of nodes. For my work, I used four nodes because they can be very handy and mobile during the long distance job. At the same time they give a good idea of a bigger network in practical scenario. My thesis concept can fairly be described as Figure 3.1

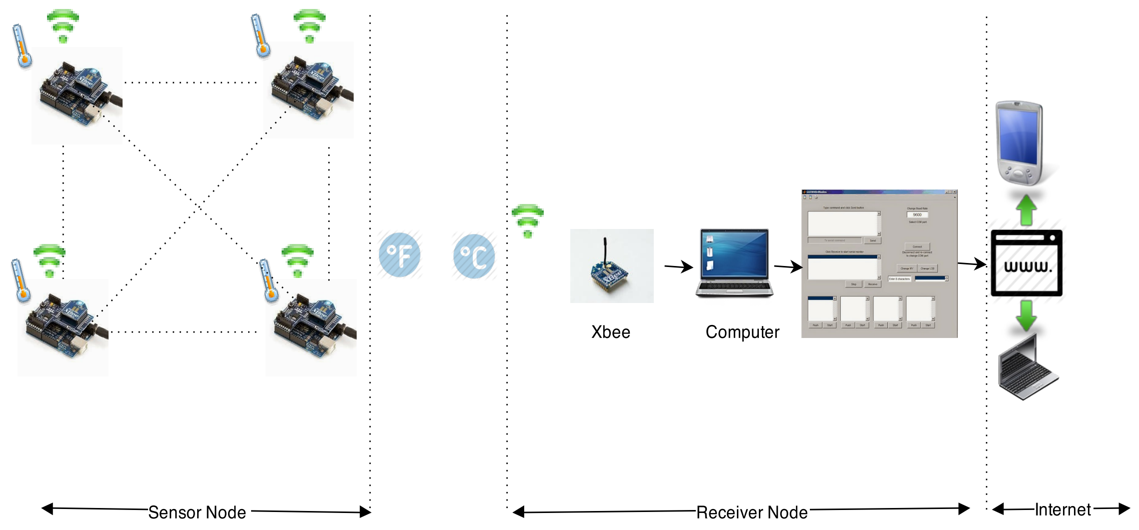


Figure 3.1: Proposed Wireless Sensor Network architecture

In Figure 3.1 the network has been divided in three major parts: sensor node, receiver node and Internet. The sensor node consists of four nodes which are connected with sensors and the receiver node is connected with a PC. The PC receives the data and uploads it to a specific website frequently. So, if even the user is not around the PC, he/she can check the status of the network and gets information about the network in almost real time. Next, I discuss about the major hardware, software and encryption methods have been utilized to implement the proposed concept.

3.1 Hardware components

The hardware is the crucial part for the thesis work. I choose components which are reliable, functional, and practical, yet cheaper. After exploring several options, I find XBee and Arduino UNO are a very good match for my work. Arduino UNO has been an integral part of the open source movement, and XBee has been available for a few years as a small but effective radio frequency component. I cover the hardware part in detail in the next sections.

3.1.1 Arduino UNO

Arduino UNO is a microcontroller board which has a small chip for computational work as well as a small memory storage. ATmega328 is the core component of this board, that works as the processor. Other components that make up this board include, 14 digital I/O pins, 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack and a reset button. The Arduino UNO is a model from the Arduino series [5].

Arduino UNO has one USB connector which can be used to upload code or for a power connection. Battery and main power options are available for power connection. The board can work on 6V to 20V. Less than 5V supply can make the board unstable, whereas more than 12V will overheat and damage the board. The recommended range is 7 volts to 12 volts [5].

The ATmega328 has 32 KB (with 0.5 KB used for the bootloader) of Flash memory. It

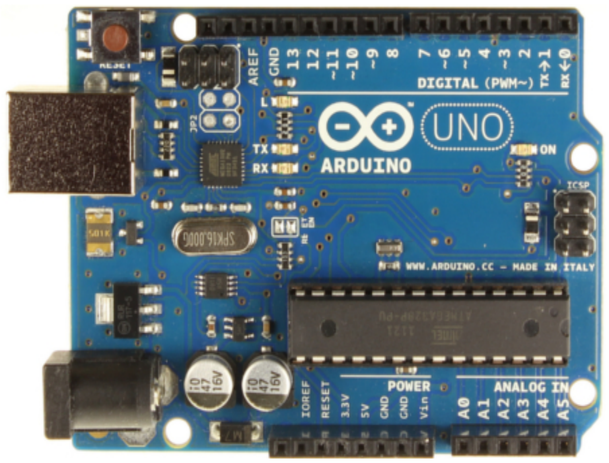


Figure 3.2: Arduino UNO [5]

also has 2 KB of SRAM and 1 KB of EEPROM [5].

The summary of hardware capability of Arduino UNO is as follows:

Table 3.1: Hardware capability of Arduino UNO [5]

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

The Arduino UNO has a number of facilities to communicate with a computer [25]. The ATmega328 provides serial communication, which is available on digital pins 0 (Receive) and 1 (Transmit) [5].

Universal Asynchronous Receiver/Transmitters (UARTs) are present in the Microcontroller to receive and transmit data serially. It transmits one bit at a time at a specified data rate (e.g. 9600bps, 19200bps, etc.). This method of serial communication is sometimes referred to as transistor-transistor logic (TTL) serial. Serial communication at a TTL level is between the limits of 0V and Vcc. Vcc is 5V or 3.3V [26].

While I am trying to make the communication secure, a general question that would come up is: Can Arduino UNO code be read from the chip once it is uploaded? When the code is uploaded in the Arduino UNO microprocessor, it is converted into Hexadecimal format. The pulled hexadecimal code is the converted machine code from the original code and is not as same as the original code. This protects the code primarily because the resources required to extract the original code from the machine code will be higher. Secondly, there is one ultimate way to protect the microprocessor, which is by locking boot loader bits. This allows the user to set the microprocessor bit in a way that will not allow anyone (including the authentic user) to read the data from Arduino UNO. The user can erase data from the chip and then upload data but cannot read what is already uploaded and bit locked. The boot loader lock mode can be enabled by software or serial or parallel programming but only can be erased by a chip erase command [27]. The facility provides much needed protection for WSNs.

3.1.2 XBee and its standards

XBee is the name of the Radio Frequency (RF) modem produced by Digi International that works as a wireless transceiver. Before discussing XBee in detail, the standard and protocol it is based on is presented first.

IEEE 802.15.4

IEEE [1] (Institute of Electrical and Electronic Engineers) introduced the 802.15.4 standard to fulfill the requirement of a certain segment of communication that needs a low data

[1]https://www.ieee.org/index.html

rate, low power and simple connectivity. The 802.15.4 specifies that the communication should occur in the 5MHz range and in 2.4 GHz band [28].

ZigBee

ZigBee is a protocol which was introduced by ZigBee Alliance, an organization which deals with the development of a protocol for low data rate applications. ZigBee was introduced as an enhancement of the 802.15.4 standard. Mesh topology is the most popular use in WSNs for ZigBee [28].

XBee

Based on these two standards and protocols, Digi International introduced two different series of XBee. XBee 802.15.4 (Series 1) is developed to meet the IEEE 802.15.4 requirement to support low cost point to point communication. XBee ZNet2.5 (Series 2) modules provide ZigBee compliant mesh. I use XBee 802.15.4 for Radio communication. Figure 3.3 shows a XBee 802.15.4.

Figure 3.3: XBee 802.15.4 module [6]

XBee 802.15.4 can work in unicast or broadcast mode. In unicast mode, the XBees communicate in one-to-one manner. In broadcast mode one XBee sends a common message to all the XBees in range, and all the XBees receive the common message broadcasted to them. In unicast, XBee address can be specified as the destination's 16 bit address, but in broadcast mode the address has to be set within a certain parameter.

XBee 802.15.4 works in different applications such as Transparent (AT) or Programming Interface (API). AT command mode is the default mode for the XBee module. In this mode, data that is received in the Data Input (DI) pin of XBee, queues up for RF transmission. When any RF data is received, it is sent to the Data Output (DO) pin of XBee. This command mode is pretty straightforward in functionality. Another command mode is API (Application Programming Interface) and it is an alternative to the AT mode. This mode is a little different than AT mode, as it converts the data into frames and then processes it to send. By API mode activated in XBee, data can be sent to multiple destinations without using the configuration. Packet delivery status can be received by the transmitter, and source address can be identified [6].

3.1.3 XBee explorer

The XBee explorer is an accessory for a XBee connection with a PC. It is an easy to use, USB to serial base unit for XBee to connect with PC. Explorer can be used in both XBee series 1 and 2. A mini A to B USB cable is needed to connect this adapter to a PC for its full functionality. It has a full XBee socket and 3 LEDs for TX (Transmission), RX (Receive) and Receive Signal Strength Indicator (RSSI). The user has direct control over the programming and serial pins of XBee after it is connected with the PC [29].

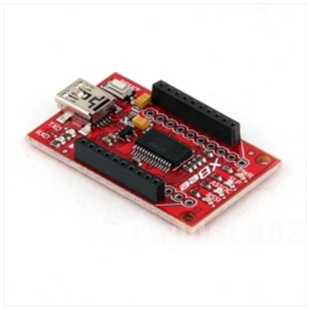


Figure 3.4: XBee explorer module [29]

3.1.4 XBee shield

XBee shield is another accessory that allows XBee to connect with Arduino UNO and provides XBee with the capability to communicate with an Arduino UNO program.

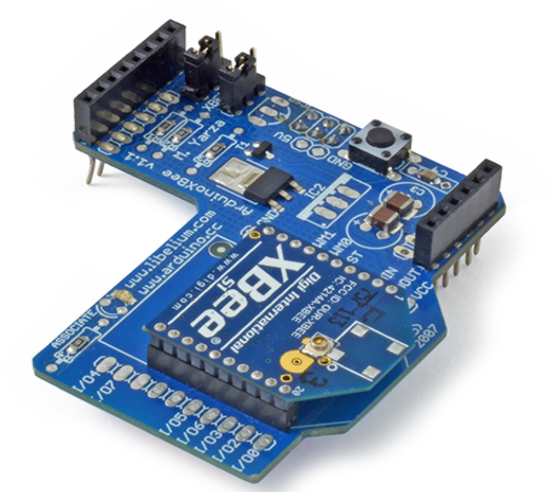


Figure 3.5: XBee shield module [5]

The XBee shield also has a socket for XBee. Two XBee/USB jumpers allow XBee to determine how to communicate with Arduino UNO. If the jumper is set to the XBee position, the microcontroller in Arduino UNO only receives a signal from XBee, but not from the USB port. When it is set to USB, then XBee module communicates directly with the computer via the USB port, but in that case, the microcontroller has to be removed first, otherwise the USB jumper does not work [5].

3.1.5 Temperature sensor

My work uses a TMP36 model temperature sensor as one of the components in my work. It gives an output in volts which can be converted to Celsius using 10mV/°C. Its voltage input is 2.7 to 5.5 VDC and its Operating Range is −40°C to +125°C [30].

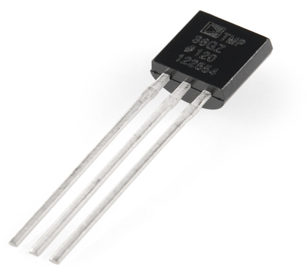


Figure 3.6: Temperature sensor [30]

3.1.6 Humidity sensor

This work also used a humidity sensor. Figure 3.7 shows the humidity sensor model used, which is the breakout HIH-4030. The HIH-4030 measures relative humidity and

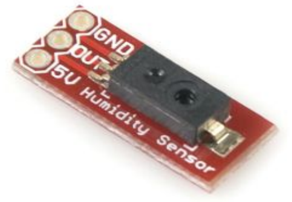


Figure 3.7: Humidity sensor [31]

delivers it as an analog output voltage. It has Ground, Output and supply volts of 5V. The sensor measures the humidity and delivers the output as analog volts. This output can be connected directly with an Analog to Digital Converter (ADC) port of Arduino UNO. Ground and 5V is connected with respective ports in Arduino UNO as well. The output shows as volts that needs to be converted to a humidity scale [31].

3.1.7 Different Wireless Sensor Networks modules

In WSNs, the sensor nodes collect environmental or physical condition by different sensors and transmit them to the base station. Most of the sensor nodes are compactly made by a manufacturer and sold as a unit. Each node accomplishes its assigned job by using a radio transceiver, a microcontroller, several circuits and a power supply. A Transceiver with an internal or external antenna communicates with other nodes, while a microcontroller works as a processor inside the node, the electronic circuits communicate with the connected sensors, and the power supply powers the sensor node. Each nodes can act as a data logger and processor, or a gateway for other nodes. As a data logger, the nodes access connected sensors, and read, process and transmit data. As a gateway, it facilitates the communication between two nodes.

Different sensor nodes are sold for different purposes. Manufacturers have their own specifications to use these sensor nodes. I collected data for different parameters of these well known sensor modules and compared them. My main goal is to find something that is open source, cheap and will fulfill my purpose. Before analyzing the sensor node I chose, I will summarize some renowned sensor modules.

1. TelosB:

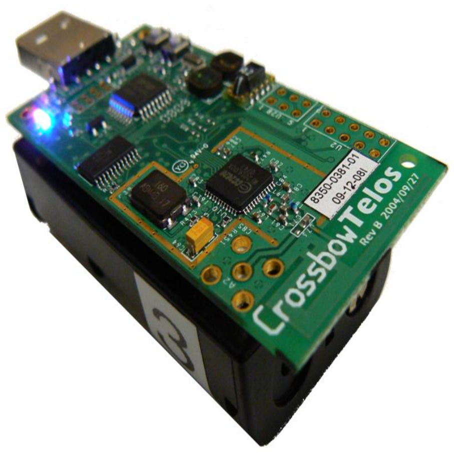


Figure 3.8: TelosB module [32]

The TelosB mote is 802.15.4 compliant, and developed and distributed by UC Berkeley. This is now available from Crossbow Technology and has an open source operating system. A separate TelosB sensor suite that integrates light, temperature and humidity sensors, is available to collect sensor data [32].

2. Mica2/MicaZ:

This is another Crossbow Technology manufactured mote. This is also 802.15.4 com-

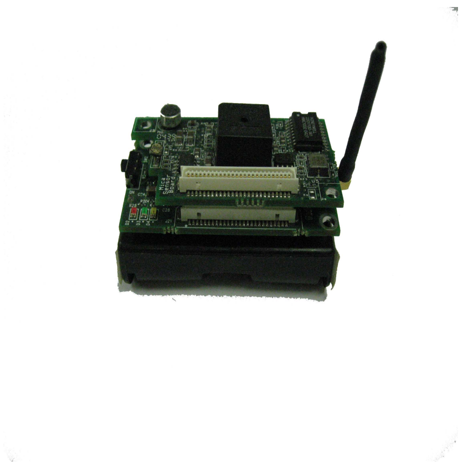


Figure 3.9: Mica2 module [33]

pliant and has a separate interface and sensor board to use as a sensor module [33].

3. IRIS:

Figure 3.10: IRIS module [34]

With some improvements from Mica2/MicaZ, Crossbow Technology introduced IRIS for data acquisition and processing. One of the major improvements as claimed is that IRIS has greater transmission range than the other two modules. It has individual sensor boards like TelosB/Mica/Micaz, meaning that different boards need to be bought for different sensoring purposes [34].

4. XBee and Arduino UNO:

This is an open source node that is not made by only one manufacturer, it is a com-

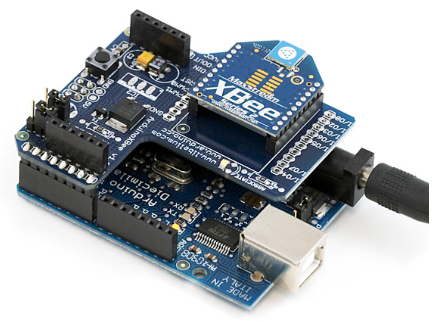


Figure 3.11: XBee and Arduino UNO module [5] [6]

bination of a transceiver from one manufacturer and a processor from another. In figure 3.11, XBee, which is manufactured by Digi International, has been used as transceiver unit to transmit or receive data with another XBee unit, while Arduino UNO has been used for processing. I chose this module for use in my research work [6].

5. Waspmote:

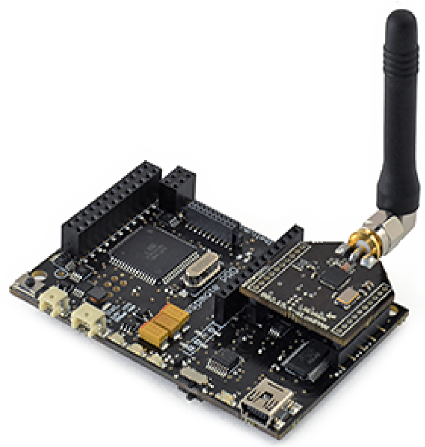


Figure 3.12: Waspmote module [35]

Waspmote is manufactured by Libillium. It operates in 2.4 GHz frequency and is very similar to the XBee and Arduino UNO node. Since the XBee and Arduino UNO both are open source, Libillium makes and distributes this module with its own name [35].

3.1.8 Reason for using XBee and Arduino UNO module

Before deciding to use XBee and Arduino UNO for my work, I went through some research on the different motes I just discussed. I compared the modules with different parameters that have more priority in my work. As an example, when I considered a module, size and weight were of less priority. I put more emphasis on a module that is cheap, open source, flexible and serves my purposes. After analyzing these parameters, which are shown in Table 3.2, it is evident that the XBee and Arduino UNO module is the most optimal choice for my research purposes. XBee and Arduino module is open source, cheap and has not been extensively used in WSNs for research purposes. I was looking for something with smaller memory or speed, so that the most efficient code can be used, rather than something with larger memory or faster speed. The biggest advantage of the XBee and Arduino module is that it is customizable. I obtained several parts separately and started making this module. In this way, different parts of the module can easily be replaced, added and changed, unlike in the manufacturers-specified modules such as TelosB, IRIS, Mica or MicaZ. As an example, for the other modules, a full package (with processor) has to be used even with a PC, even though a PC itself has a powerful processor, and the small processor in that module is a waste of a resource. In the XBee and Arduino UNO, XBee can be dismantled from Arduino UNO (the processor) and only XBee (which is the transceiver) needs to be used with a PC. In this way, XBee utilizes the bigger processor power of the PC. The same also applies for sensors, as in the XBee and Arduino UNO module. We can use several sensors simply by connecting the sensors with Arduino UNO. For other manufacturer specific modules, a user must buy a separate device with a separate sensor that again is a waste of a processor.

Table 3.2: Comparison between different WSNs nodes

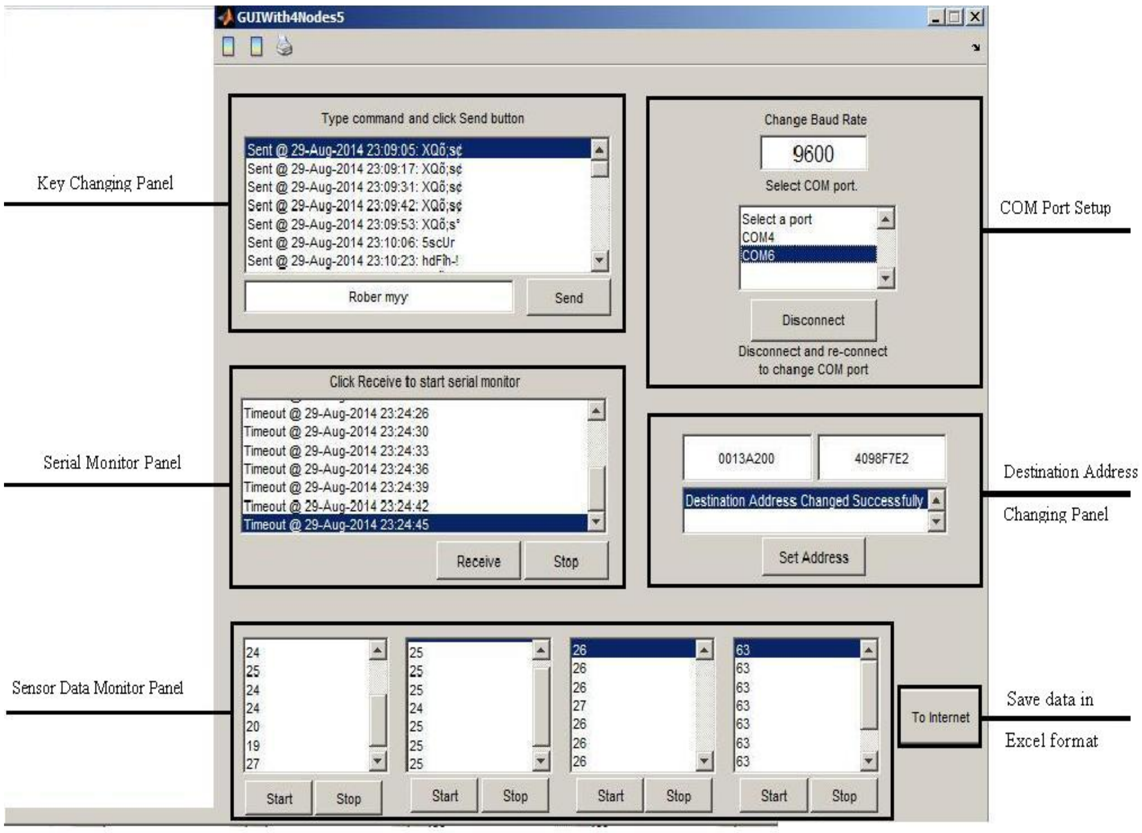
Factors	**TelosB TPR2400CA [32]**	**Micaz [33]**	**IRIS [34]**	**Waspmote [35]**	**XBee-Arduino [5] [6]**
Price (CAD)	$130	$400	$130	$255	$77
TX data rate	250 kbps	250 kbps	250 kbps	250 kbps	250 kbps
Indoor range	20 m to 30 m	20 m to 30 m	>50m	30 m	30m
Outdoor range	75 m to 100 m	75 m to 100 m	>300m	250 m	90m
RAM	10K bytes	4K bytes	8K bytes	8K bytes	2K bytes
EEPROM	16K bytes	4K bytes	4K bytes	4K bytes	1K byte
Program Flash Memory	48K bytes	128K bytes	128K bytes	128K bytes	32K bytes
Size (mm)	65 x 31 x 6	58 x 32 x 7	58 x 32 x 7	73.5 x 51 x 13	74 x 53 x 22
Weight (gm)	23	18	18	20	5+28=33

Table 3.2 shows different modules that are compared with each other according to some parameters. If we consider the price, there is a distinct difference between the motes manufactured by Crossbow Technology and the others, although the transmission (TX) data rate is exactly same for all of them. The data rate is an important aspect for a device because this is the speed by which data is transferred from one end to the other. Most of the devices' indoor and outdoor range is almost the same, If we consider IRIS for its better range, we can recall that its manufacturer improved its outdoor range. The programming code I use for this thesis work needs to be efficient and work in small memory because the smaller the memory, the lower the cost, which eventually lowers the whole network price. So, if for only one unit, the price difference is so high, it would be much higher bigger network.

3.2 Software used

3.2.1 Matlab

The PC works as the base station for the network as well as the receiver of the sensor data. A Matlab GUI works as an interface between the device and the user. Matlab is a programming language for the matrix manipulations, Graphical User Interfaces (GUI) and simulations. I used it to make the control panel for the user.



```
Sent @ 29-Aug-2014 23:09:05: XQõ;s¢
Sent @ 29-Aug-2014 23:09:17: XQõ;s¢
Sent @ 29-Aug-2014 23:09:31: XQõ;s¢
Sent @ 29-Aug-2014 23:09:42: XQõ;s¢
Sent @ 29-Aug-2014 23:09:53: XQõ;s'
Sent @ 29-Aug-2014 23:10:06: 5scUr
Sent @ 29-Aug-2014 23:10:23: hdFîh-!
```

Rober myy

9600

Select a port
COM4
COM6

```
Timeout @ 29-Aug-2014 23:24:26
Timeout @ 29-Aug-2014 23:24:30
Timeout @ 29-Aug-2014 23:24:33
Timeout @ 29-Aug-2014 23:24:36
Timeout @ 29-Aug-2014 23:24:39
Timeout @ 29-Aug-2014 23:24:42
Timeout @ 29-Aug-2014 23:24:45
```

0013A200 4098F7E2

Destination Address Changed Successfully

```
24    25    26    63
25    25    26    63
24    25    26    63
24    24    27    63
20    25    26    63
19    25    26    63
27    25    26    63
```

Figure 3.13: Matlab GUI layout

The layout of the Matlab GUI that I have developed is shown in Figure 3.2.1. The COM Port Setup (Communication port/ serial port interface on computers) which is located in the upper right corner, is for setting up and displaying available COM ports. After selecting the baud rate as well as the COM port, the user will click Connect button to connect to that COM port. After connecting with the COM port, user will put the destination address. The destination address is the serial address of the XBee and Arduino node, which the user wants to communicate. After putting the address, clicking on the Set Address button will write the address in XBee. Now the base station can communicate with the desired node. The base station and all of the sensor nodes are pre-installed with an initial key. The key changing panel in the upper left corner allows the user to change the key whenever he/she wants. Once the user changes the key in that panel, the key is also changed in the base station. The same key updated at the destination sensor node. It is a very important part of the whole process.

The Serial Monitor Panel shows all activities that take place during the communication. If the network is idle for a few seconds, it shows a time out at that very moment. The Sensor Data Monitor panel has four small windows to show incoming sensor data from different nodes. If the user clicks on the Send button on the Key Changing panel, the new key is sent after being encrypted to the destination node. The destination node gets the encrypted command, decrypts it with the stored key and sends the sensor data after encrypting it with the new key. When the base station side gets the new data from the destination, it decrypts the data and shows it in one of the windows of the Sensor Data Monitoring panel. Clicking on 'To Internet' button saves all of the sensor data in a file that can be used for the internet uploading. The whole procedure is discussed widely in the next chapter.

3.2.2 Arduino UNO IDE

The Arduino UNO IDE is split into three major areas. The area at the top of the window of Figure 3.14, features toolbars that controls the program. The middle white section is used to write the Arduino UNO programming code, and the bottom part of the window shows the status messages. Arduino code is similar to the C programming code, so compiler error or memory usage messages are shown in the bottom area [5].

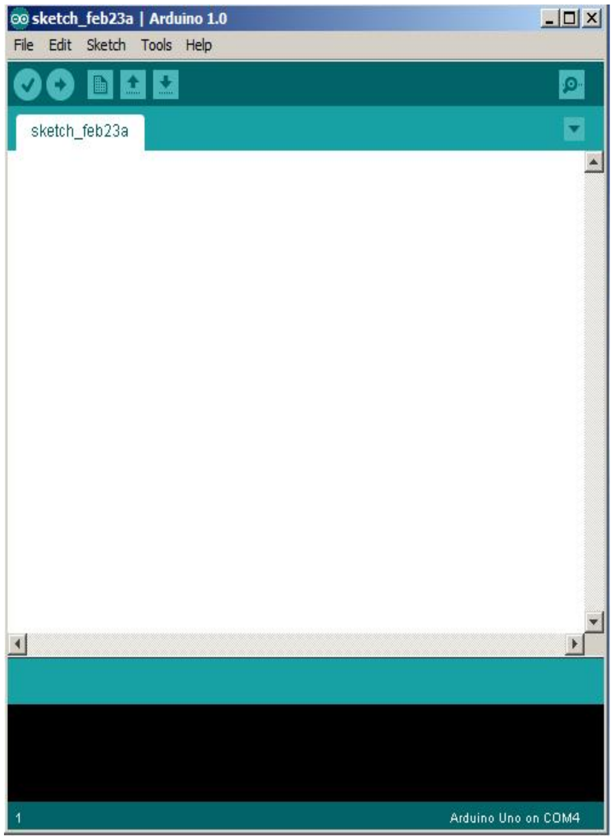
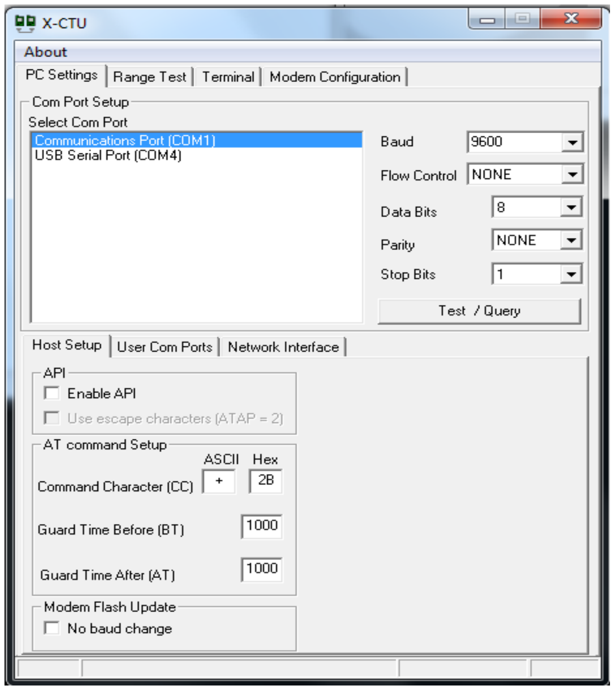


Figure 3.14: Arduino UNO IDE [5]

3.2.3 X-CTU

X-CTU is used in my work to configure the XBee with the XBee explorer. The X-CTU is an application which allows the user to connect with the XBee via a USB port and change the configuration of the XBee. X-CTU is used in this thesis work as it is one of the most used applications for the XBee configuration and is equally powerful in the Windows environment. It has several parts with different functionalities. Figure 3.15 shows a sample X-CTU panel with different tabs. The tab PC settings allows the user to select and change the COM port as well as the baud rate for a particular XBee. The Range test is used to measure the received level of signal between two XBees. Terminal is used for the user commands as well as to monitor the received message. So the most important part of the X-CTU is the Modem configuration, which allows the user to configure the XBee for desired firmware. The XBee can also be configured by using the command mode from the Terminal tab [36].



Com Port Setup

Select Com Port

Communications Port (COM1)
USB Serial Port (COM4)

Baud 9600
Flow Control NONE
Data Bits 8
Parity NONE
Stop Bits 1

API
☐ Enable API
☐ Use escape characters (ATAP = 2)

AT command Setup

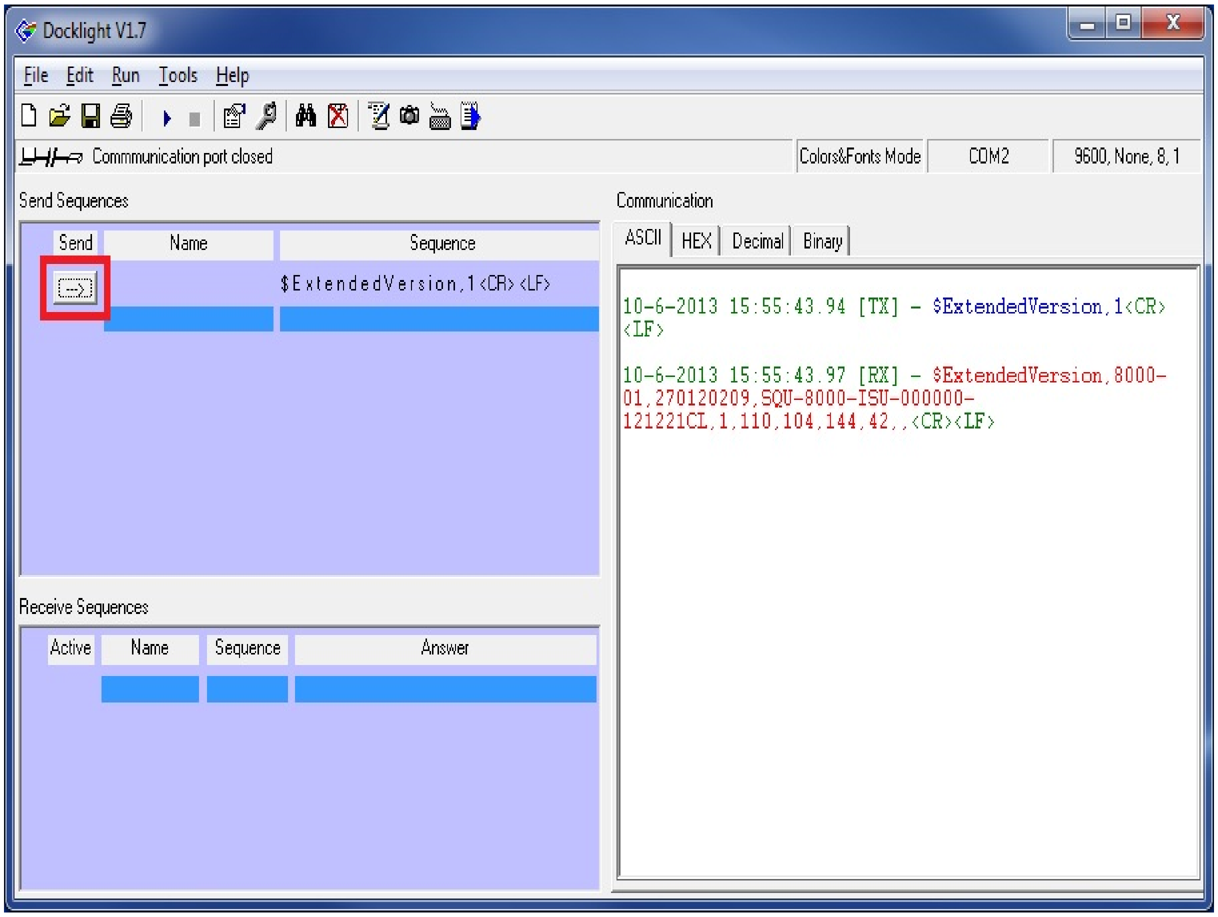
	ASCII	Hex
Command Character (CC)	+	2B
Guard Time Before (BT)		1000
Guard Time After (AT)		1000

Modem Flash Update
☐ No baud change

Figure 3.15: X-CTU panel [36]

3.2.4 Docklight

Docklight 2.0.5 is a development tool (i.e. testing, analysis and simulation) for the serial communication protocol. The user can monitor communications between two serial devices and can send or receive the data packets from one device to another using the Docklight. I use this software to calculate the performance of two serial devices in my implemented network that is discussed in Chapter 5 in more detail. Docklight can be used to respond to the incoming data as well as to detect the data sequence [37]. Figure 3.16 shows a Docklight panel.



$ExtendedVersion,1<CR><LF>

```
10-6-2013 15:55:43.94 [TX] - $ExtendedVersion,1<CR>
<LF>

10-6-2013 15:55:43.97 [RX] - $ExtendedVersion,8000-
01,270120209,SQU-8000-ISU-000000-
121221CL,1,110,104,144,42,,<CR><LF>
```

Figure 3.16: Docklight panel [37]

3.3 Encryption techniques

Encryption is a technique to convert a plaintext into a form, called a ciphertext, that cannot be easily understood by someone unauthorized. During the implementation and the performance analysis, two encryption techniques have been utilized. A short description of these techniques is as below:

3.3.1 RC4

RC4 has been used in the key distribution implementation as the default encryption technique. RC4, officially termed as ’Rivest Cipher 4’, is a stream cipher that operates in byte with variable key size. RC4 is used in Secure Socket Layer (SSL), Wired Equivalent Privacy (WEP) protocol, WiFi Protected Access (WPA) protocol. The operation of RC4 can be divided into two parts, Key-scheduling algorithm (KSA) and Pseudo-random generation algorithm (PRGA). KSA starts by initializing a 256-byte state vector with variable-length key of 1 to 256 bytes. After KSA operation, PRGA modifies the state vector and generates an output byte of almost random keys. Next, the key stream value of pseudo random keys are XORed with the plaintext to convert the plaintext into ciphertext [38].

3.3.2 AES

Advanced Encryption Standard (AES) is a block cipher with a state vector of 128 bits and a key length of variable size. AES works in 10 processing rounds for 128-bit keys where all the rounds are identical except the last one. Before any round-based processing for encryption begins, the input state array is XORed with the first four words of the key schedule. In decryption, the ciphertext state array is XORed with the last four words of the key schedule. Each encryption round operates in following four steps: 1) Substitute bytes, 2) Shift rows, 3) Mix columns, and 4) Add round key. The last step consists of XORing the output of the previous three steps with the four words from the key schedule. Each decryption round consists of the following four steps: 1) Inverse shift rows, 2) Inverse substitute bytes, 3) Add round key, and 4) Inverse mix columns. The third step consists of XORing the output of the previous two steps with four words from the key schedule [38].

Chapter 4

Key Exchange For The Authentication

For any secured network, the generation, exchange and distribution of the keys is very important. In the last chapter, I discussed about a proposed reconfigurable WSNs architecture, the software, and the hardware configuration I used. In this chapter, I talk about the system architecture of the proposed protocol, as well as the essential steps in its implementation in a hardware environment. The key exchange is one of the biggest and most complex element of the proposed work, and before giving the reader an exact idea of how I accomplished this, I want to discuss in a greater detail about the system architecture, the process and the implementation.

4.1 System architecture

4.1.1 Circuits and wiring

Four nodes connect with sensors to collect sensor data and send that data to the base station. The base station is composed of one XBee, which is connected to a PC via a mini A to B USB cable. Each sensor node consists of an XBee, an XBee shield and an Arduino UNO. This package is then wired with a sensor to a breadboard. I use temperature sensor in three sensor nodes and a humidity sensor in a fourth. The circuits and wiring for these two sensor nodes are shown in Figures 4.1 and 4.2.

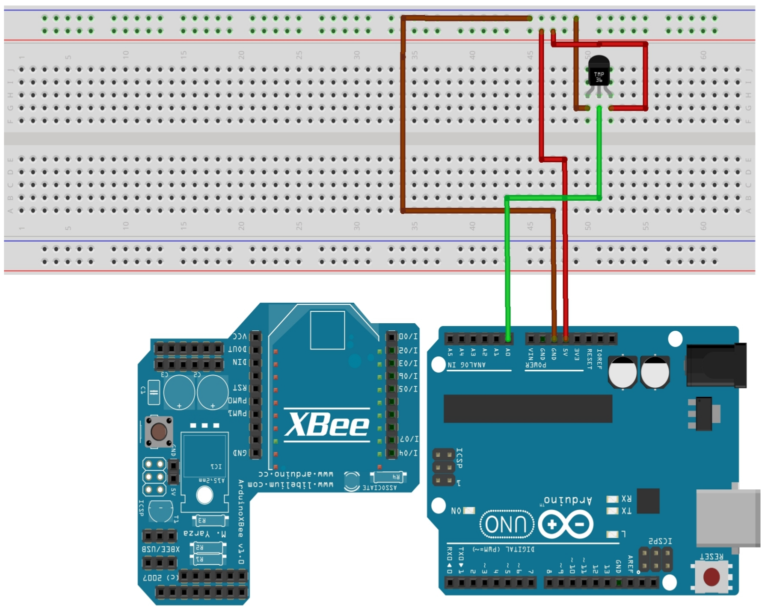


Figure 4.1: Circuit architecture with temperature sensor

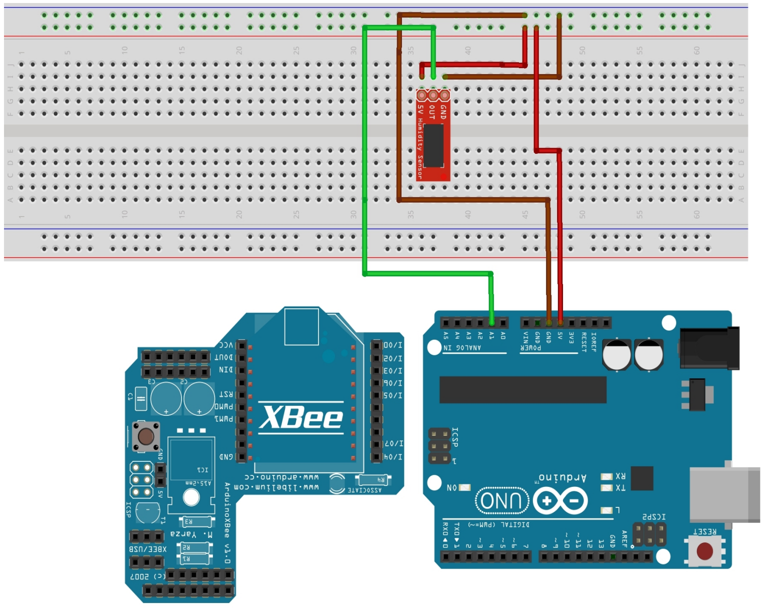


Figure 4.2: Circuit architecture with humidity sensor

4.1.2 Overview of the key exchange and authentication

Figure 4.3 shows the step by step process for key exchange and authentication:

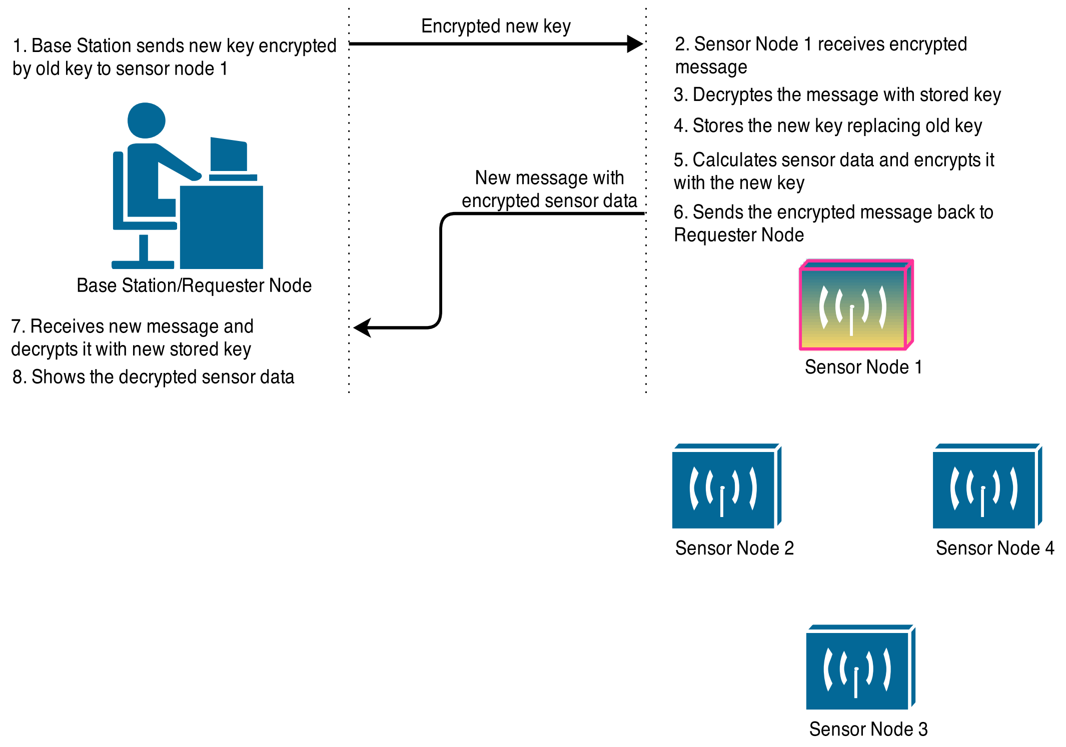

Encrypted new key
1. Base Station sends new key encrypted by old key to sensor node 1
2. Sensor Node 1 receives encrypted message
3. Decryptes the message with stored key
4. Stores the new key replacing old key
5. Calculates sensor data and encrypts it with the new key
6. Sends the encrypted message back to Requester Node
New message with encrypted sensor data
Base Station/Requester Node
7. Receives new message and decrypts it with new stored key
8. Shows the decrypted sensor data
Sensor Node 1
Sensor Node 2
Sensor Node 4
Sensor Node 3

Figure 4.3: Overview of the authentication concept and procedure

A key is pre-installed in each sensor node before the process starts. The sensor nodes are then distributed and the base station starts communicating with them. When the base station wants to read the sensor data, it communicates with a designated sensor node. For example, in Figure 4.3 the base station wants to obtain the sensor data from Node 1. First, it sends a sensor data request to the node. The request acts as an identifier, confirming the authenticity of the base station. The request is in fact a new key set by the base station which is then encrypted by the previous key before it is sent to the sensor. As the previous key has already been installed in the sensor node, after receiving the encrypted request from the base station, the sensor node decrypts the request with the previous stored key. The sensor node replaces the previous key with the new key, and uses this new key for further communication with the base station. After identifying the request, the sensor node calculates the sensor data (e.g. temperature, humidity, motion etc.), encrypts that data and sends it back to the base station. The base station receives the new encrypted message from that particular sensor node and decrypts it with the new key to reveal the sensor data. The process is similar if no new key is sent by the base station, with all data being encrypted with the most recent key. In same way, the base station sends and receives sensor data from other sensor nodes (in this example Node 2, Node 3 or Node 4) too.

In next section, I discuss the different stages of the physical implementation of the proposed key exchange and authentication. This work can is divided into two major parts: 1) Communication between nodes using encryption 2) Communication between nodes with authentication using encryption.

4.2 Communication between nodes with encryption

4.2.1 Communication between two individual XBee and Arduino UNO

This is the first phase of work that is done with the XBee and Arduino UNO package. Before initializing XBee, it should be configured in the X-CTU. Opening of the X-CTU shows the connected COM ports. So after selecting the desired COM port, the modem configuration tab must be selected. Then, clicking the Read tab shows the configuration of the particular XBee. From there, the user must change the configuration the way he/she wants.On the Read tab, the Modem, Function Set and Version are shown for the XBee. The user selects the desired parameters and addressing. For individual communication (one to one), the user can set the destination address in the address field. Then, the firmware must be downloaded from the Internet and uploaded to XBee. After setting up the two XBees, codes are uploaded for the two Arduino UNOs. One of them is uploaded with transmitting code; another one with receiving code.

Before establishing communication between these two nodes, I ensure that the XBees are communicating with each other. In order for the two XBees, they must be configured with same baud rate and same PAN (Personal Area Network) ID. After configuration in the X-CTU terminal, the user can send a command to the other XBee, and vice versa. Once they communicate with each other, the XBees are placed over the Arduino UNO. The receiver node should be able to receive the data sent by the sender node. Figure 4.4 shows one sender node sending a message to the receiver node.

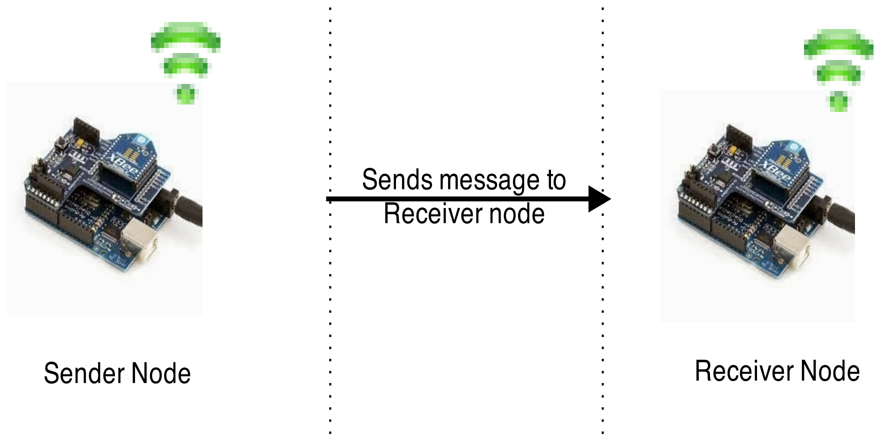


Figure 4.4: Communication between two Nodes

Findings : The two XBees need to be configured with the same baud rate and with the same PAN ID so they are able to communicate with each other. Setting the PAN ID to 0xFFFF helps the XBee to broadcast a message to all the XBees in same PAN.

Problems encountered and solution : The first problem encountered in this work is during configuration of the XBee. One of the XBees ceases to function entirely due to faulty hardware or software and it declined to let me make any changes in its configuration. The problem was later solved by resetting the XBee manually with a male to male wire by putting two ends of the wire to the reset and ground pin of the XBee respectively.

4.2.2 Encryption and decryption in two XBee and Arduino UNO nodes

This work is done after successfully implementing the first phase - communication between two nodes. Basically in this phase the job is divided into three separate stages.

Stage 1 : This stage deals with the encryption and the decryption in one sensor node only with a predefined data source and a key. In the first stage, the RC4 encryption and the decryption codes are uploaded to one Arduino UNO. The output shows the proper encryption of the data source, followed by decryption of the data source into plaintext.

Stage 2 : In this stage, the encryption and the decryption are done for two separate sensor nodes. The transmitter sends an encrypted plaintext. After receiving the data, the receiver decrypts it back to plaintext.

Stage 3 : One of the sensor nodes is uploaded with the encryption code and the key. The other node, which is the receiver node, is uploaded with the decryption key. Instead of using a pre-defined data source, the sensor data is encrypted in the transmission node. The sensor is connected with the node (XBee and Arduino UNO). The node reads the sensor, encrypts the data and sends it to the receiver node. The receiver node receives the data, decrypts with the stored common key and extracts the sensor data. The procedure is shown in Figure 4.5.

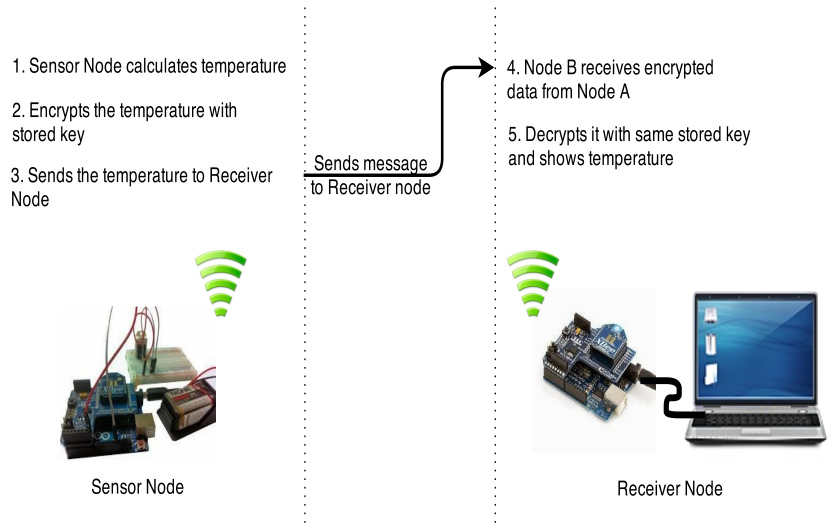

1. Sensor Node calculates temperature
2. Encrypts the temperature with stored key
3. Sends the temperature to Receiver Node
Sends message to Receiver node
4. Node B receives encrypted data from Node A
5. Decrypts it with same stored key and shows temperature
Sensor Node
Receiver Node

Figure 4.5: Encryption and decryption in two XBee and Arduino UNO nodes

Findings : Since the RC4 encryption is small yet powerful, it is very compatible with the smaller processor and memory size of Arduino UNO.

Problems encountered and solution : The first stage is straightforward. The program encrypts the plaintext with the RC4 encryption, and then decrypts the encrypted data with the help of key. But the same process, when split and put into different nodes, works incorrectly. The data is encrypted and sent, but the receiver does not receive it. This is solved by adding some delay before transmission. The cipher data is now received but then a new problem arises. The data is not received by the receiver in the desired way. The receiver receives the data one character at a time and without adding it with the other preceding character (s), it starts decrypting it with the key. The final result was unacceptable. Then I learnt about the new serial data communication problem, and solved it by introducing a buffer in receiver end. The buffer looks for data and as long as data is arriving, it adds it to the end of the buffer. Once no more data arrives, the contents of the buffer is decryptd in its entirety.

4.3 Communication with authentication using encryption

4.3.1 Duplex communication between two nodes with sensor data

Duplex communication is commonly known as communication between two nodes. The purpose of using duplex communication is to see whether the one node that sends data, can receive data from other node. To accomplish this job, at first two nodes need to be chosen. The first node sends a character and the second node receives it. After receiving, if it sees that the received data fulfills the condition (a condition is set so that the receiving node will react if it gets data from the particular node), it sends back a response to the first node. Then the first node receives and displays the message. Next, one node sends a request to a sensor node for a sensor data, receives it and displays it in the serial monitor.

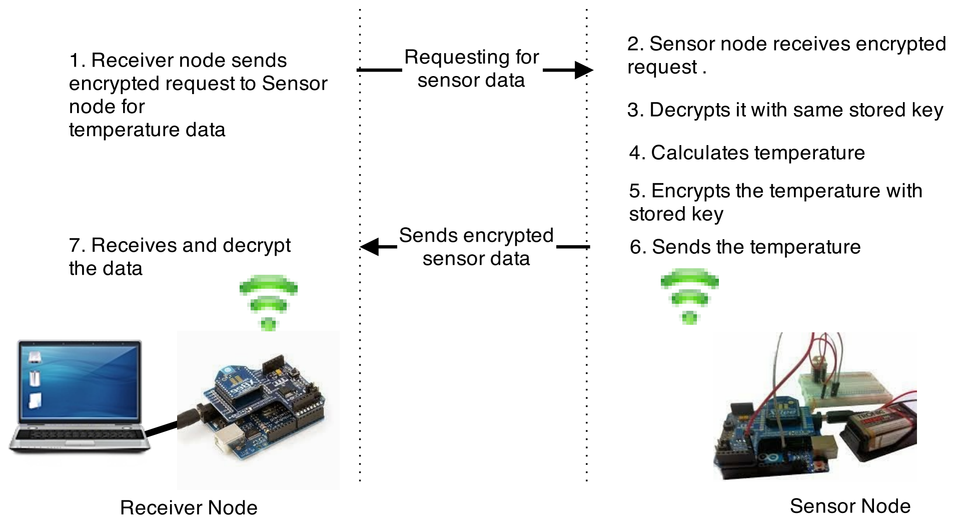

1. Receiver node sends encrypted request to Sensor node for temperature data
Requesting for sensor data
2. Sensor node receives encrypted request .
3. Decrypts it with same stored key
4. Calculates temperature
5. Encrypts the temperature with stored key
6. Sends the temperature
Sends encrypted sensor data
7. Receives and decrypt the data
Receiver Node
Sensor Node

Figure 4.6: Duplex communication between two nodes with sensor data

Findings : The synchronization is a major factor during the duplex communication. XBee Series 1 is a half duplex instead of full duplex, which means it cannot send and receive at the same time. It has to allocate a time for sending and then look for receiving data in the next allocated time.

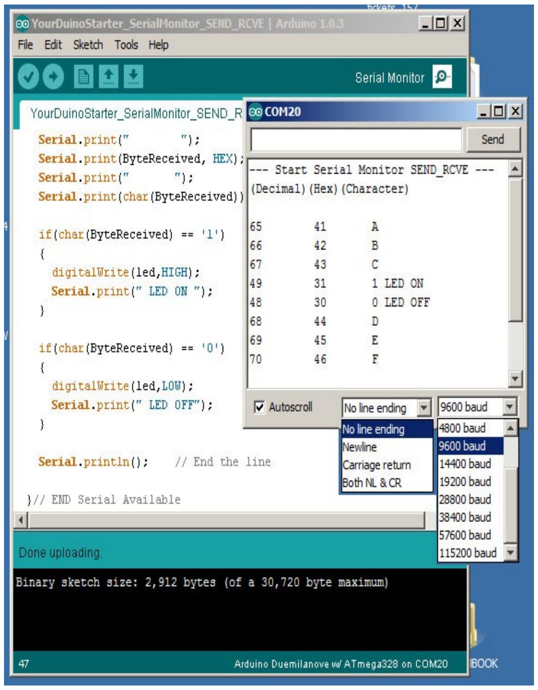
Problems encountered and solution : For the first phase of the work, no data was coming to the first node after sending the request for sensor data. Therefore, some modifications in code in terms of serial availability had to be made before the node started receiving some data from the other node. But strangely, the buffer which was used successfully before, suddenly stopped working. A new buffer had to be introduced for receiving data properly.

4.3.2 Input command in serial monitor of Arduino UNO

Previously discussed concepts are the foundation for the advanced level of communication for my work. My main goal is to put together all of the proposed concepts and implementation of those concepts for authentication. It is very important to give proper control to the base station, that is in charge of the whole network.

Again, two XBees and two Arduino UNO boards are used and initialized with all necessary codes and a pre-installed key. This key works as temporary key until a new key is introduced by the base station. When the base station creates a new key and sends it to a sensor node, the new key replaces the old key in both nodes. The new key is encrypted by the RC4 encryption with the previous key. After receiving the encrypted data, the sensor node decrypts it with its copy of the previously stored key. The authentication between these two party is ensured by using the same key. If the receiver can decrypt the encrypted data, this authenticates the user. Otherwise, the request is rejected. After storing the new key, the sensor node sends the sensor data to the base station.

The base station must put the key somewhere in the Arduino UNO IDE in order to to send or update the previous key. One way to do it is by putting the key in the Arduino UNO Serial monitor. A serial monitor for Arduino UNO is shown in Figure 4.7. The serial monitor has different panels for different purposes. In the upper panel, beside send button, there is one place to Enter a command and send it directly to the Arduino UNO. The code that is uploaded to the the Arduino UNO needs to have the capability to process the newly input data. Once the new key is received, the process works as discussed before.



```
  Serial.print("        ");
  Serial.print(ByteReceived, HEX);
  Serial.print("       ");
  Serial.print(char(ByteReceived));

  if(char(ByteReceived) == '1')
  {
    digitalWrite(led,HIGH);
    Serial.print(" LED ON ");
  }

  if(char(ByteReceived) == '0')
  {
    digitalWrite(led,LOW);
    Serial.print(" LED OFF");
  }

  Serial.println();    // End the line

}// END Serial Available
```

```
--- Start Serial Monitor SEND_RCVE ---
(Decimal)(Hex)(Character)

65      41      A
66      42      B
67      43      C
49      31      1 LED ON
48      30      0 LED OFF
68      44      D
69      45      E
70      46      F
```

```
Binary sketch size: 2,912 bytes (of a 30,720 byte maximum)
```

Figure 4.7: Serial Monitor of Arduino UNO

Findings : The XBee connects with Arduino UNO via the XBee shield [5]. But the shield we have been using has a major drawback. It has one Transmission (TX) and one Receive (RX) pin. Whenever XBee is connected to Arduino UNO, the TX and RX pins are already occupied with connections to XBee pins. The command mode in the serial monitor of Arduino IDE is used to send a command to Arduino UNO via the PC. This also uses the same TX and RX pins in Arduino UNO. Therefore, no command passes through the TX and RX pins since the pins are already used by the serial ports and thus nothing is executed.

Problems encountered and solution : The problem of the XBee shield was very crucial for the next step and initially, I could not find anyway to get forward. Then, I read about the jumper (the small removable plastic sleeves that each fit onto two of the three pins labelled Xbee/USB) XBee/USB on the Arduino UNO shield. I found that a little modification in the hardware and an additional software library might overcome the issue. In the hardware part, the jumper must be removed and connected with two wires in the middle of the pins as shown in Figure 4.8.Then, a special software library was used, since the standard Arduino library works for one serial connection only. The SoftwareSerial library [39] helps data from a PC to reach the XBee via Arduino UNO. As discussed in findings, while the XBee shield is employed on the Arduino UNO, no data passes from a PC as the pins that send data to the Arduino UNO from a PC is occupied by the XBee. But by using the SoftwareSerial library and the wiring connection in the Arduino UNO, other digital pins in Arduino UNO can be used to carry the data from PC to Arduino simultaneously. The library can be found on the Arduino UNO website. These software and hardware modifications allow the Arduino UNO to receive a command from the PC serial monitor as well as communicate with the other nodes at the same time.

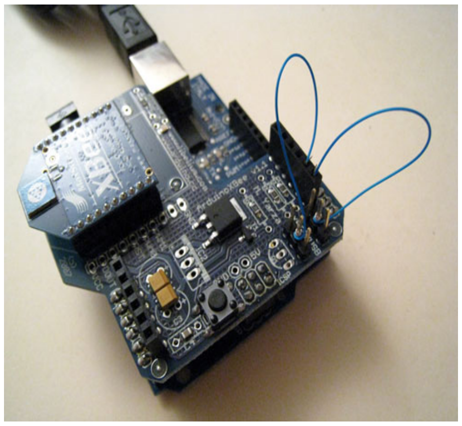


Figure 4.8: Engineered Arduino UNO with wires

Therefore, SoftwareSerial library takes care of the single TX/RX port problem and defines a new TX/RX port to use for the commands from the PC.

4.3.3 Authentication in XBee and Arduino UNO node

This phase deals with the implementation of authentication. An administrator sends a new key through the Arduino UNO IDE command panel. After clicking the Send button, the serial message is sent from the PC to the Arduino UNO. Then the Arduino UNO processses the serial message by encrypting it with RC4 encryption and sends it to the sensor node. Here, one thing to be noted is that the XBee configuration must be changed at the beginning. The destination address must be set so that the base station understand where to send the message. The address configuration in the destination mus be set as well. So, after receiving the encrypted data, the sensor node decrypts it with its pre-stored key. After decrypting the data, the Arduino UNO stores this data as a new key which is used to encrypt the sensor data. The sensor module is attached to the sensor node. After receiving the command from an authentic source, it reads the sensor data. After reading the sensor data, it encrypts the data with the new key. After encryption, it sends this new encrypted message to the requester. The requester then decrypts it and shows the data in the serial monitor.

Findings : The XBee module I use is called XBee 802.15.4 or Series 1 to distinguish from the other XBee module that is Series 2. Both the XBee modules have almost a similar

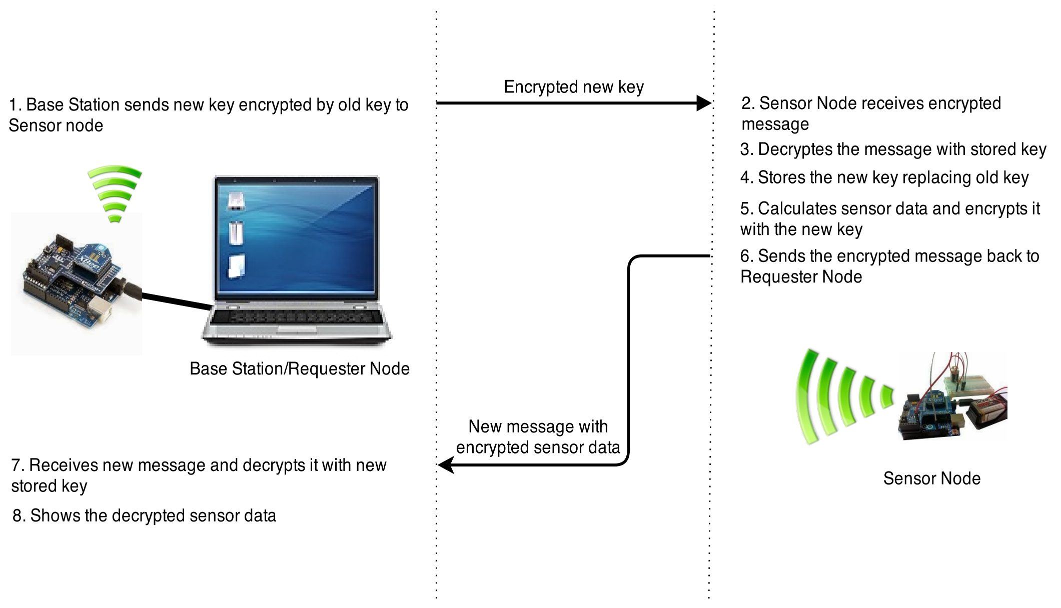

1. Base Station sends new key encrypted by old key to Sensor node
Encrypted new key
2. Sensor Node receives encrypted message
3. Decryptes the message with stored key
4. Stores the new key replacing old key
5. Calculates sensor data and encrypts it with the new key
6. Sends the encrypted message back to Requester Node
Base Station/Requester Node
Sensor Node
New message with encrypted sensor data
7. Receives new message and decrypts it with new stored key
8. Shows the decrypted sensor data

Figure 4.9: Implementing the authentication in XBee and Arduino UNO node

physical configuration, but their implementation functions and the topology they create are different. Each series can communicate only with other modules in the same series modules. The point to point function of Series 1 limits its capability to be used liberally in a network.

Problems encountered and solution : The base station needs to know the sensor value from the different sensor nodes. It is not practical every time to use the X-CTU to configure the XBee to send and receive messages from the different nodes. A solution must be implemented so that it is easier for the user who will manage the base station to just change the address without having to get involved too deeply in the functionality of XBee. However, the Arduino UNO Serial monitor this is not possible, because there is no function to change an address or determine a destination address. As a solution, a user interface needs to be implemented which caters to this requirement. My next work is related to this concept.

4.3.4 Authentication in Matlab

In an earlier phase, the requirement for a more user friendly interface has been discussed so that the user can input a command in the PC and send it to a sensor node, without having to use the X-CTU software to change the address each time. I feel that using the Matlab as an interface is a very good alternative due to its powerful functionality.

Stage 1 : Matlab has a strong graphical interface which is capable of many functions. So, I converted the serial monitor of Arduino UNO IDE to the Matlab with an address facility without having to change any hardware. However, I feel more flexibility and less expense can be achieved if I use only XBee on the PC side and exclude the Arduino UNO. But this leads to another bigger problem of needing to implement everything again in Matlab including last phase again. It includes converting encryption, decryption and the associated codes. Implementing the whole concept in Matlab works as shown in Figure 4.10:

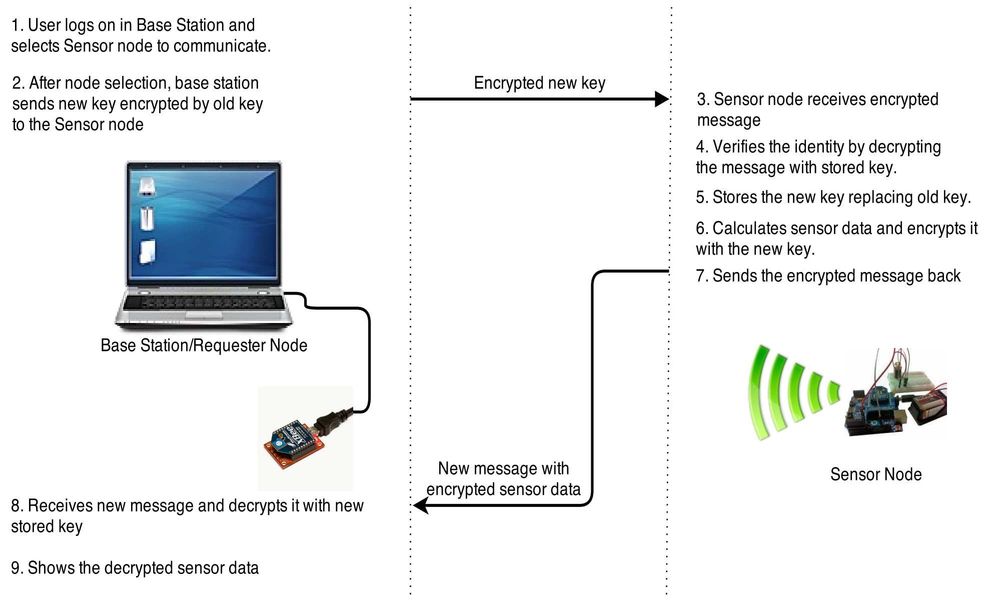
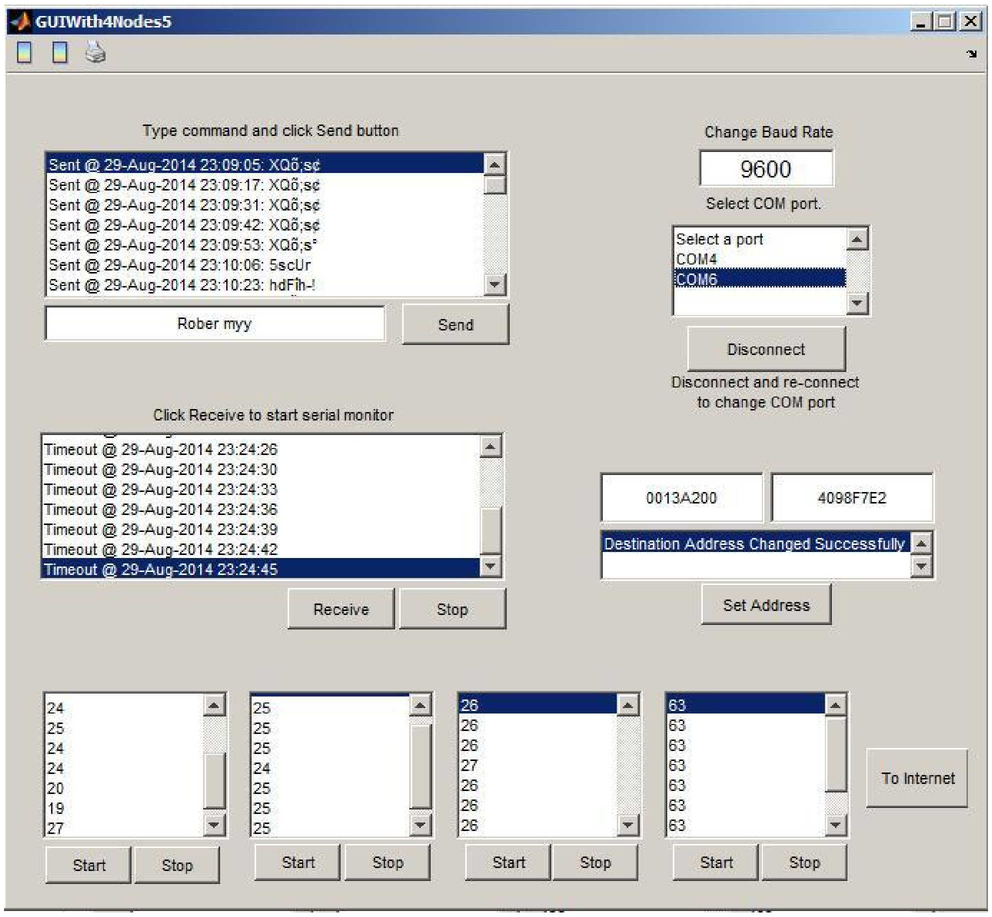

1. User logs on in Base Station and selects Sensor node to communicate.
2. After node selection, base station sends new key encrypted by old key to the Sensor node
Encrypted new key
3. Sensor node receives encrypted message
4. Verifies the identity by decrypting the message with stored key.
5. Stores the new key replacing old key.
6. Calculates sensor data and encrypts it with the new key.
7. Sends the encrypted message back
Base Station/Requester Node
Sensor Node
New message with encrypted sensor data
8. Receives new message and decrypts it with new stored key
9. Shows the decrypted sensor data

Figure 4.10: Key exchange and authentication in Matlab

At the left portion of Figure 4.10, one PC is connected with one XBee. The right portion shows an XBee and the Arduino UNO node connected with a sensor. First, the user sends request to the node by selecting that node's address. This part will be easier to understand if the interface is shown as well. The developed Matlab user interface is shown in Figure 4.11.

If we refer to Figure 4.10, we can see the sensor node receives a request from the base station and decrypts the data. It retrieves the key, calculates a sensor value and encrypts


GUIWith4Nodes5
Type command and click Send button
Send
Click Receive to start serial monitor
Receive
Stop
Change Baud Rate
Select COM port.
Disconnect
Disconnect and re-connect to change COM port
Set Address
Start
Stop
To Internet

```
Sent @ 29-Aug-2014 23:09:05: XQõ;s¢
Sent @ 29-Aug-2014 23:09:17: XQõ;s¢
Sent @ 29-Aug-2014 23:09:31: XQõ;s¢
Sent @ 29-Aug-2014 23:09:42: XQõ;s¢
Sent @ 29-Aug-2014 23:09:53: XQõ;s°
Sent @ 29-Aug-2014 23:10:06: 5scUr
Sent @ 29-Aug-2014 23:10:23: hdFîh-!
```

Rober myy

```
Timeout @ 29-Aug-2014 23:24:26
Timeout @ 29-Aug-2014 23:24:30
Timeout @ 29-Aug-2014 23:24:33
Timeout @ 29-Aug-2014 23:24:36
Timeout @ 29-Aug-2014 23:24:39
Timeout @ 29-Aug-2014 23:24:42
Timeout @ 29-Aug-2014 23:24:45
```

9600

Select a port
COM4
COM6

0013A200 | 4098F7E2

Destination Address Changed Successfully

24	25	26	63
25	25	26	63
24	25	26	63
24	24	27	63
20	25	26	63
19	25	26	63
27	25	26	63

Figure 4.11: A working Matlab GUI with different parts

with the new key. While implementing the concept in Matlab GUI, a different section has been built for keeping the XBee address format, which is shown in Figure 4.11. The XBee has a distinguished address format where Serial low (SL) and Serial High (SH) determines the address of the XBee. The address can be found either in the X-CTU configuration or on the back of the XBee. This address is 64-bit where SH gives a high 32 bits and SL gives low 32 bits. So, when the user sends a command to a particular XBee, he/she writes the SH and SL of the particular XBee in a panel. That is saved as DH and DL (Destination High and Low) for that particular XBee. After successfully configuring the address, the PC connects with the COM port. For this, a region has been made to connect with the available COM ports. The available COM ports are shown in the top right hand corner of the GUI and the user selects one and pushes the connect button. The user also can change the baud rate if required. In the upper left hand corner is the TX serial command, where the user can write a command as the key for the desired XBee. This works similar to Arduino UNO's serial monitor command section. There is also one area that shows the messages that are sent or received via XBee. There are four other serial monitors that show the sensor data received by the XBee after sending the key and receiving the encrypted data.

Findings : Matlab is a very powerful programming language that handles everything with matrices and produces a solid GUI with significant flexibility.

Problems encountered and solution : The first problem encountered was the need to implement of all the codes of the Arduino UNO once again in Matlab. The coding process in Matlab is different than in C or in the Arduino UNO programming. The new programming rules had to be studied. Since Matlab handles everything using matrices, the Arduino UNO codes had to be converted in a similar fashion.

Stage 2 : In stage 2, I add mesh network capability to the network. I already discussed the mesh topology in Chapter 2. The mesh topology is the most reliable topology considering its flexibility as well as its scalability. A Mesh network provides multiple points of failure recovery. As the nodes are connected with each other in mesh, one single point of failure does not make communication impossible. The chance of the data loss is low due to the alternate paths of the network.

The Series 1 XBee has a firmware installed for mesh networking. This firmware is called the Digimesh. A Digimesh network is defined with a unique network identifier. The modules must have a Digimesh network identifier in order to to communicate with each other. At first, the XBees were configured by the X-CTU with the Digimesh and a range test were performed to understand the periphery of XBee. This work is done in both open and closed places.

Two or more XBees are placed in such a way that they do not communicate with each other. Then, another XBee is placed in between them at a distance where that XBee can communicate with both of the other XBees. While doing so, the X-CTU needs continuous monitoring. After introducing the new XBee, the X-CTU shows a data signal from the remotest node. By this way, we can see that the Digimesh is working, because the remotest node sends the data to the middle (in-between) node, and the middle node routes it to the base station.

Then I try to change the key of the remotest node (now XBee is replaced with XBee+Arduino UNO+sensor) from another node (XBee+Arduino UNO+sensor) from the PC. I send the key by selecting the destination of the remotest node, while having the middle node turned off. There is no response and no key is changed or no acknowledgement is received. But when a new node is introduced in the middle as mentioned above, the key is changed and an acknowledgement is received on the PC side. Sensor data is also received from both of the nodes simultaneously without any interference or overlapping. The whole process is shown in Figure 4.12:

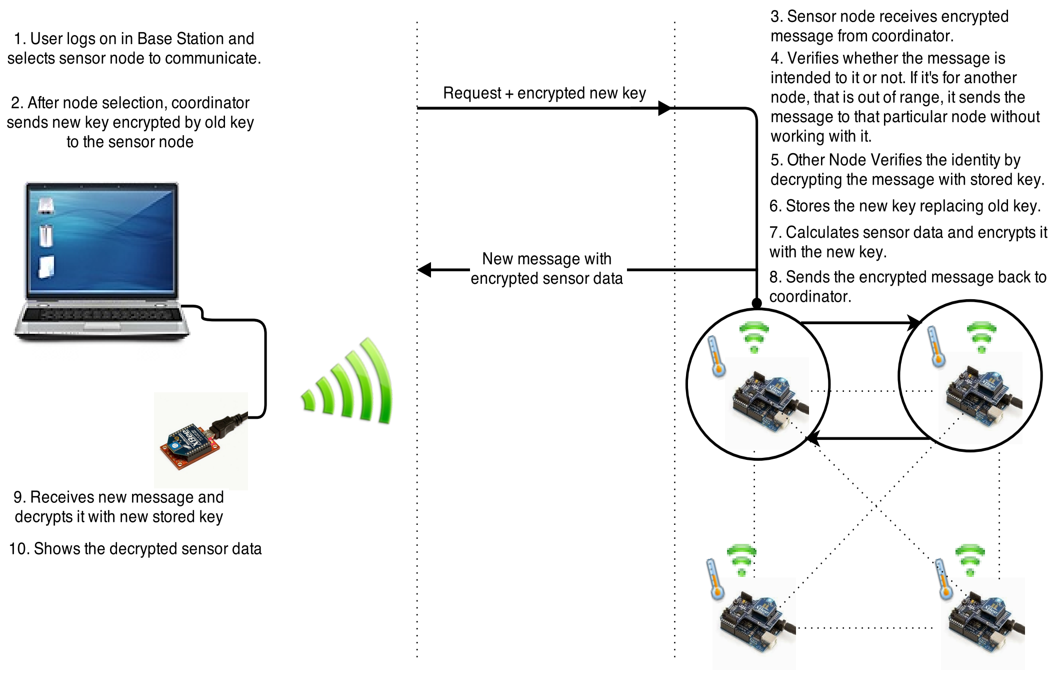

1. User logs on in Base Station and selects sensor node to communicate.
2. After node selection, coordinator sends new key encrypted by old key to the sensor node
Request + encrypted new key
3. Sensor node receives encrypted message from coordinator.
4. Verifies whether the message is intended to it or not. If it's for another node, that is out of range, it sends the message to that particular node without working with it.
5. Other Node Verifies the identity by decrypting the message with stored key.
6. Stores the new key replacing old key.
7. Calculates sensor data and encrypts it with the new key.
8. Sends the encrypted message back to coordinator.
New message with encrypted sensor data
9. Receives new message and decrypts it with new stored key
10. Shows the decrypted sensor data

Figure 4.12: Authentication concept with Mesh networking

Findings : The Digimesh 2.4 is a new firmware made by the Digi international, as XBee series 1 does not have any mesh capability in its default firmware.

Problems encountered and solution : The Digimesh requires proper addressing in the sensor nodes. While I worked with the normal point to point firmware, addressing was not very crucial. But for the Digimesh firmware, the sensor nodes have to know their destination. If the remotest node is out of the range from the base station and the base station wants to communicate with it, the base station has to communicate via another in-between node. The destination address of the in-between node should be set with the address of the base station. If no address is set, the in-between node does not understand which data is intended for it. It processes any data that passes through it even if that is intended for another node. Once the in-between node knows the destination address, it works as a router for other nodes and does not process the data which is intended for another node. That is why to ensure the stability of the network, all the nodes should be configured with a destination address.

4.3.5 Receiving sensor data from all the nodes simultaneously

In previous stage when the base station wants to know the status of a sensor node, it changes the destination address in the PC XBee and sends the key to the remote sensor for the data. This occurs as many times as necessary to read the sensor data. One major advantage of this key agreement procedure is key freshness. As each time a new key is generated to acquire the sensor data from the sensor node, the attacks due to old key can be avoided. As XBee series 1 only communicates in a one-to-one manner, the user must change the address each time to communicate with a different sensor node. Getting data simultaneously from all of the four nodes with different keys is not possible. We discovered that Matlab has a deficiency, which is that its Graphical Interface's push buttons do not work together. That means when one button is pressed the other buttons' activities stop working. Therefore, only one push button works at a time. This is not an ideal way to get the data from all of the sensor nodes. This is why I introduce another window for the sensor nodes, where one button will be pushed and all four results will be shown one by one. By this experiment, I want to see whether it is possible for the XBee to receive all of the signals one-by-one or not without the user intervention. In this method, all four nodes receive a known common key from the user. Then, after pressing the 'Enable All' button, XBee connected with PC starts receiving data from the nodes one-by-one in encrypted form. After receiving the data, the receiver decrypts and displays them in specific boxes. I set the cycle time of the sensor nodes to be different from each other, so that they do not collide. The GUI for the receiving sensor data from four of the sensor nodes without user intervention is as follows.

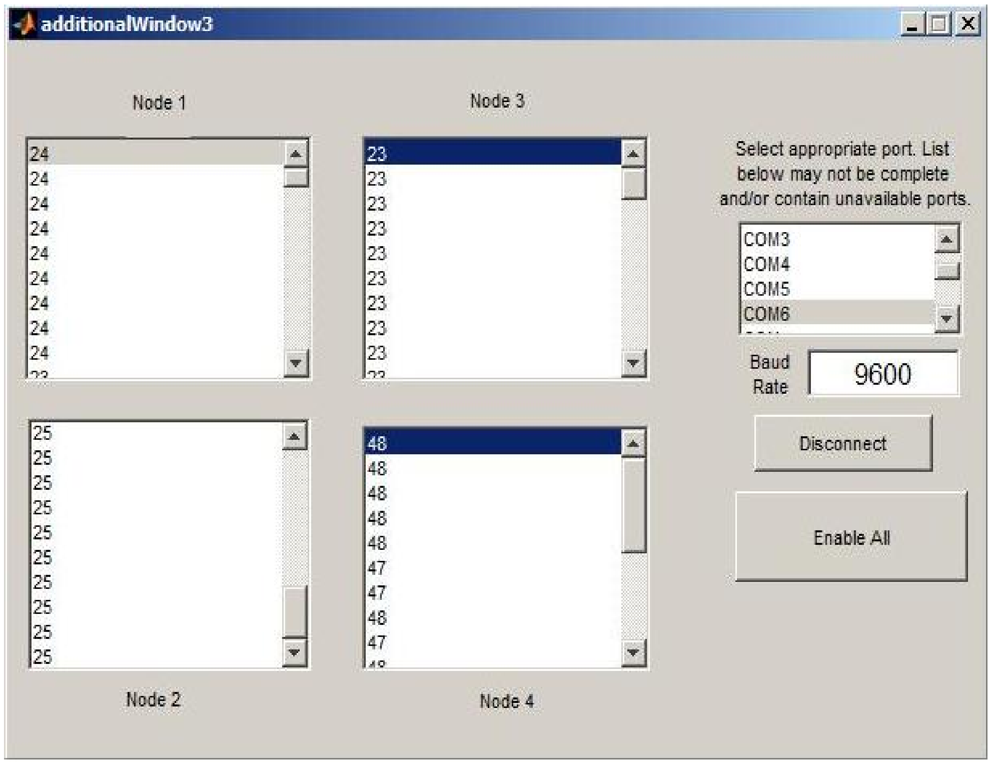


Figure 4.13: GUI shows temperature sensor data shown in Node 1, 2 and 3 windows and humidity sensor data in Node 4 window simultaneously

Findings : The Matlab buttons can not work together simultaneously. Only one button works at a single time, so if another process is dependent on another button, that button has to be pressed to start the process.

Problems encountered and solution : The timing for each cycle from the sensor nodes is critical. If two data arrive at the same time, they collide and both get lost. So, it is very important to set the time in such a way that the data do not ever collide with each other.

Chapter 5

Performance Analysis and Result

In this Chapter I discuss the methodology and results of my performance analysis of the network.

5.1 Performance analysis with different feature

The performance analysis considers for different parameters and the different attributes. Mostly it is a performance test between one block cipher (i.e. Advanced Encryption Standard (AES)) with another stream cipher (RC4 encryption) algorithm. The AES performance analysis has been carried out using the Arduino UNO code as well as by enabling the AES in the XBee radio configuration. This analysis covers the Received Signal Strength Indicator (RSSI), the Latency and the Throughput of the nodes.

5.1.1 Received Signal Strength Indicator

Received Signal Strength Indicator (RSSI) provides information on the strength of the signal that the receiver receives. It is a measurement of power of the received signal. Generally the higher the RSSI, the better the quality and speed of the signal. The RSSI depends mainly on the power of the transmitter, the sensitivity of the receiver, and the path loss of the signal. These factors are crucial as a signal passes through the air, and the distance it crosses can make a huge difference in the indicator [40]. The RSSI performance analysis has been done with different baud rates and different power levels to see the impact of power on the network. After changing the baud rate and power level of the two XBees, one XBee is placed with the Arduino UNO and another on the XBee explorer. The same data size (in bytes) has been used in all RSSI experiments. In Arduino UNO, a code for measuring RSSI has been uploaded. The user provides the data at the XBee side and it is sent to the XBee and Arduino UNO module side. The Arduino UNO processes it with its stored code and gives the RSSI value in a hexadecimal value in negative numbers. This value should be converted to decimal for easier calculation. Since the value is negative, the closer the value is to 0, the stronger the signal, the farther it is from 0, the weaker the signal [41].

The RC4 performance on the RSSI is shown in a chart as well as in a graph in Table 5.1 and Figure 5.1 respectively:

Table 5.1: RSSI values with transmit power for the RC4

Baud Rate	**Transmit Power**				
	4 (-0dbm)	**3 (-2dbm)**	**2 (-4dbm)**	**1 (-6dbm)**	**0 (-10dbm)**
9600	-55	-54	-56	-56	-60
19200	-54	-56	-56	-58	-63
38400	-69	-63	-63	-76	-64
57600	-60	-63	-62	-66	-63
115200	-74	-71	-79	-82	-88

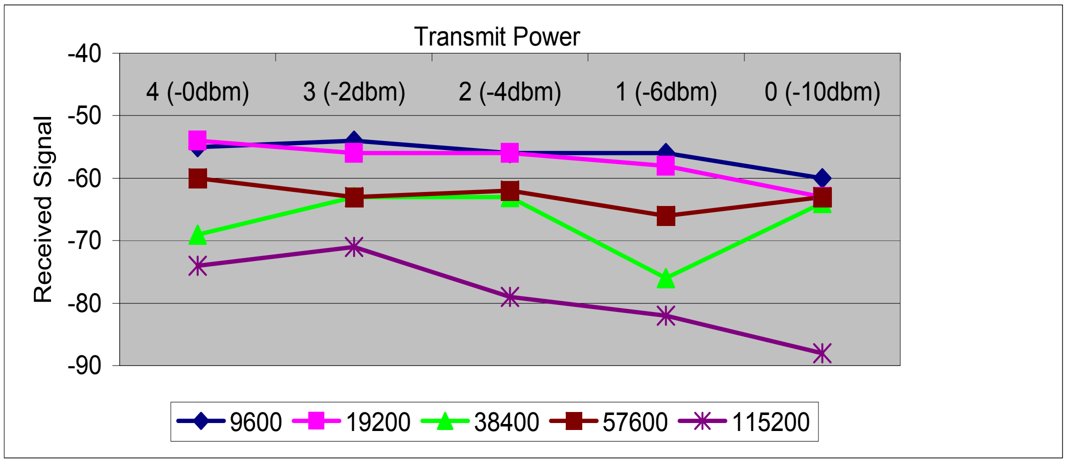


Figure 5.1: RSSI values with transmit power for the RC4

Table 5.1 shows a steady relationship between the receive levels and the transmit power with the different baud rates. The best overall performance is shown at the baud rate 9600. At the 19200 baud rate it is also showing a very impressive result. The lowest signal is received at the 115200 baud rate with -10dbm transmit power. A receive signal tends to decrease with a decreasing power level for each of the baud rates.

If we analyze Table 5.2, which indicates performance of the AES over different baud rates with the different power, we see a decline in performance of RSSI with a lower power level.

Table 5.2: RSSI values with transmit power for the AES

Baud Rate	**Transmit Power**				
	4 (-0dbm)	**3 (-2dbm)**	**2 (-4dbm)**	**1 (-6dbm)**	**0 (-10dbm)**
9600	-60	-67	-67	-76	-77
19200	-63	-63	-66	-66	-75
38400	-70	-72	-70	-71	-79
57600	-53	-55	-55	-56	-66
115200	-69	-68	-71	-71	-77

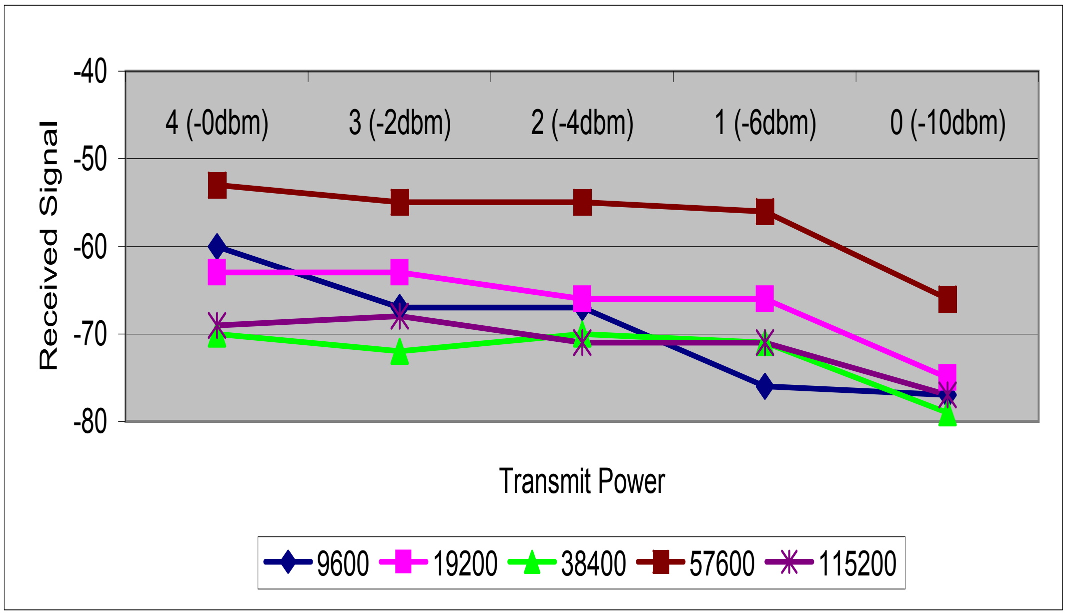
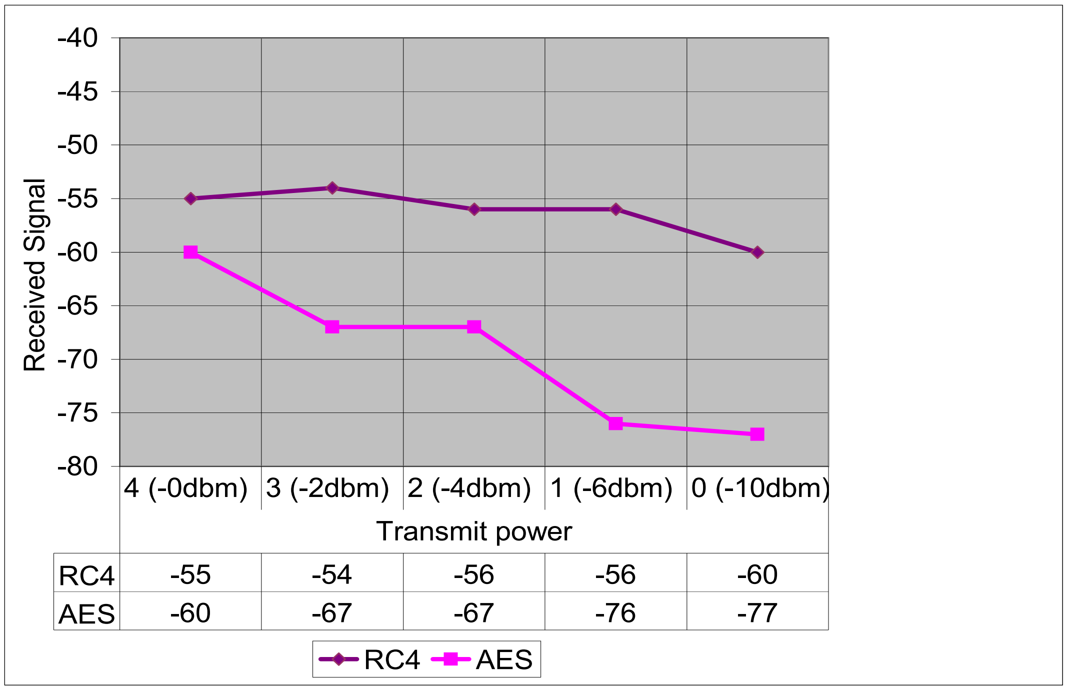


Figure 5.2: RSSI values with transmit power for the AES

In AES, a received signal shows similar behavior as for the RC4 values. Here a 57600 baud rate performs really well but a 19200 baud rate shows the same kind of result as the RC4. The results for the 115200 baud rates improved than RC4 but for 38400 and 9600 baud rate, it shows very poor performance.

For a clear indication of performance with RC4 and AES, only RSSI values in 9600 baud rate is shown as below:



	4 (-0dbm)	3 (-2dbm)	2 (-4dbm)	1 (-6dbm)	0 (-10dbm)
RC4	-55	-54	-56	-56	-60
AES	-60	-67	-67	-76	-77

Figure 5.3: RSSI values with transmit power for RC4 and AES in 9600 baud rate

In Figure 5.3, received signal has been compared with different transmit power. For 0dbm transmit power, the values are almost same for RC4 and AES encryption. But the gap between them start increasing only after changing the power to -2dbm. RC4 maintains a steady characteristics for all the five transmit power options but AES gradually decreasing with decreasing power.

One thing is evident here that the performance of the RC4 is better than the AES in terms of the RSSI. Although the best result for the AES is almost identical to RC4 but the baud rate is higher. Since, higher baud rates can destabilize the communication, the results for RC4 should be acceptable in all forms of baud rates.

5.1.2 Latency

Latency is an expression that provides a value of the time it takes for a packet of data to get from one point to another. It can be measured either one way or round trip. One way latency is the time difference from the time of sending by the sender to receiving the packet by the receiver. Round trip latency is calculated by the time it takes to go from the sender to the receiver and then again to the sender.

Latency in my work has been considered with the encryption and the decryption. This experiment uses four XBee and Arduino UNO nodes that communicate with another XBee and Arduino UNO node that is connected with the PC. The PC-side node acts as a sender, meaning it sends different sizes of encrypted data to the receiver nodes. The four nodes receive the encrypted data and decrypt it with its pre-stored key. The four nodes are distributed in such a way that the two of the nodes are in range of the PC-side node and the other two nodes are out of range. All the nodes are configured with the mesh capability. So that the out-of-range nodes can also send or receive data to/from the PC-side node. The one-way latency time is calculated from starting of the encryption in the sensor node to the finishing of the decryption in a receiver node. The purpose of this performance test is to analyze the effect of the baud rate and the data size on the communication latency considering the encryption and the decryption. Figure 5.4 shows the procedure of the performance test using a mesh network.

To finish this task, I use Docklight 2.0.5, which is a development tool for the serial communication protocol. First, at the sender side, Docklight is first configured with the desired baud rate and the COM ports. After opening the Docklight at the sender side, from Tools, the Project Settings tab is selected. Here, the user sets the COM and the COM port settings. At the receiver side, the COM port must be selected too. So, if a user clicks on the send button in the sender node side, the data should be shown in the receiver node of Docklight. Since, in both sides the originating time for sending and receiving are showing, it is easy to determine the time it takes for data to travel from the sender to the receiver.

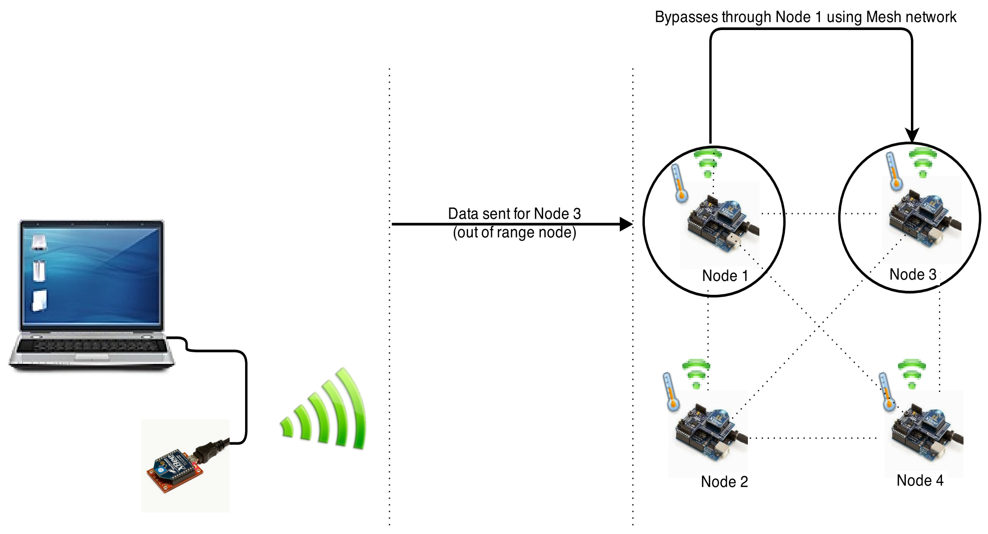
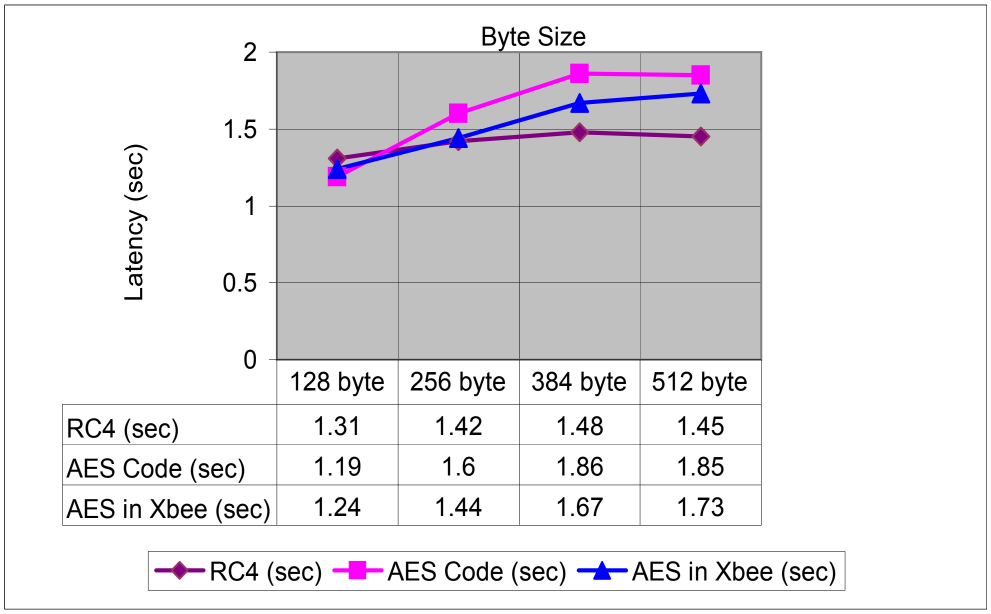


Figure 5.4: Using mesh network for performance analysis

Then, encryption and the decryption codes are uploaded to the nodes. Before uploading the code, the user has to confirm the baud rates for all the nodes and the data size for the code. The baud rate must be same for XBee and Arduino UNO for all the nodes. Any change in the baud rate has to be reflected both in the XBee and the Arduino UNO. After confirming the necessary data size in one node, the code is uploaded for the encryption and in the another node decryption code is uploaded. The sender sends encrypted data to the receiver, where it is decrypted. Although there are four nodes, the PC-side node communicates with only one node at a time because XBee series 1 only supports one to one communication. After calculating the latency for one node, it starts communicating with another one. For out-of-range nodes, the PC-side node communicates with them via other in-range nodes using the mesh network. In mesh networks, two out-of-range nodes can communicate with each other by using the radio frequency of another node in between. In Figure 5.4, Nodes 1 and 2 are in-range nodes but the Nodes 3 and 4 are out-of-range. So, when the user sends data for the Node 3, the data at first goes to Node 1. Node 1 checks the address and when it sees that data is intended for the Node 3, it passes the data onto the Node 3 without modifying it.

For all of the nodes, Latency is calculated considering the RC4 and the AES encryption. It is calculated with different baud rates and different data size. For the AES, two approaches have been utilized. For the first, AES is coded in Arduino UNO and then uploaded to Arduino UNO similarly to RC4 encryption. The code used for the AES encryption is from [42]. In the second approach, the default AES function in XBee is used, but in this case, no encryption code is uploaded to Arduino UNO. Only the XBee configuration for the AES has been kept enabled inside of the XBee module.

For the first tests, latency is calculated using the in-range nodes 1 and 2 with a 9600 baud rate and different data size. This allows me to analyze the effect of the same amount of data and the baud rate on latency for the Nodes. The reason for using Node 1 and Node 2 is because both are in-range nodes and within almost same distance from the PC.

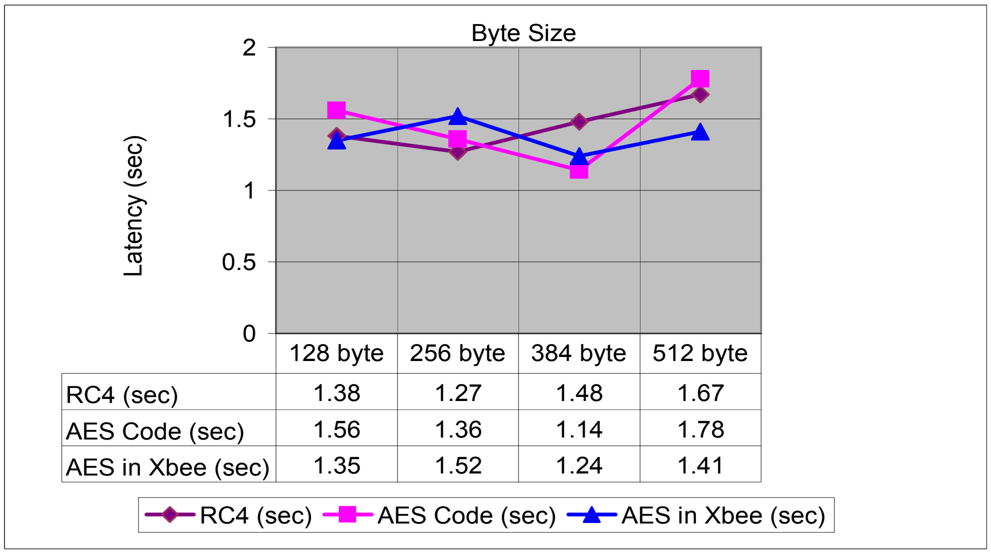


	128 byte	256 byte	384 byte	512 byte
RC4 (sec)	1.31	1.42	1.48	1.45
AES Code (sec)	1.19	1.6	1.86	1.85
AES in Xbee (sec)	1.24	1.44	1.67	1.73

Figure 5.5: Latency values in Node 1 for baud rate 9600

Figure 5.5 shows the elapsed time for different data sizes to travel from the PC to Node 1. RC4 takes an almost constant amount of time for all data sizes compared to other two AES encryptions. For the default AES fucntion in XBee and the AES coded encryption, the latency increases with bigger data sizes.

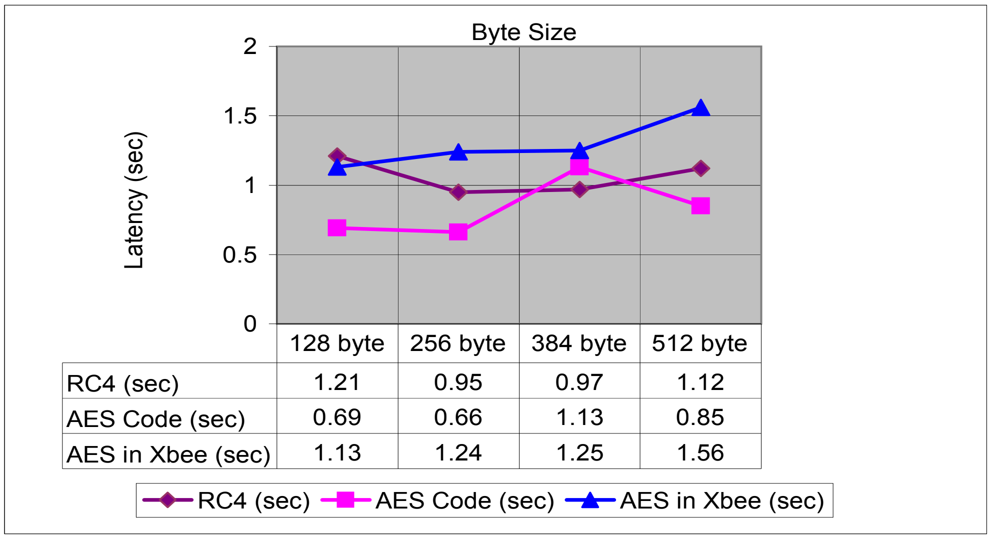
In Figure 5.6, for the 9600 baud rate and Node 2, we see the three lines bear dissimilarities in behavior. RC4 and AES code values starts with a decline for 256 byte. RC4 changes the trend quickly from the next data size and goes up but the other two either fluctuates or shows longer latency in subsequent data sizes.



	128 byte	256 byte	384 byte	512 byte
RC4 (sec)	1.38	1.27	1.48	1.67
AES Code (sec)	1.56	1.36	1.14	1.78
AES in Xbee (sec)	1.35	1.52	1.24	1.41

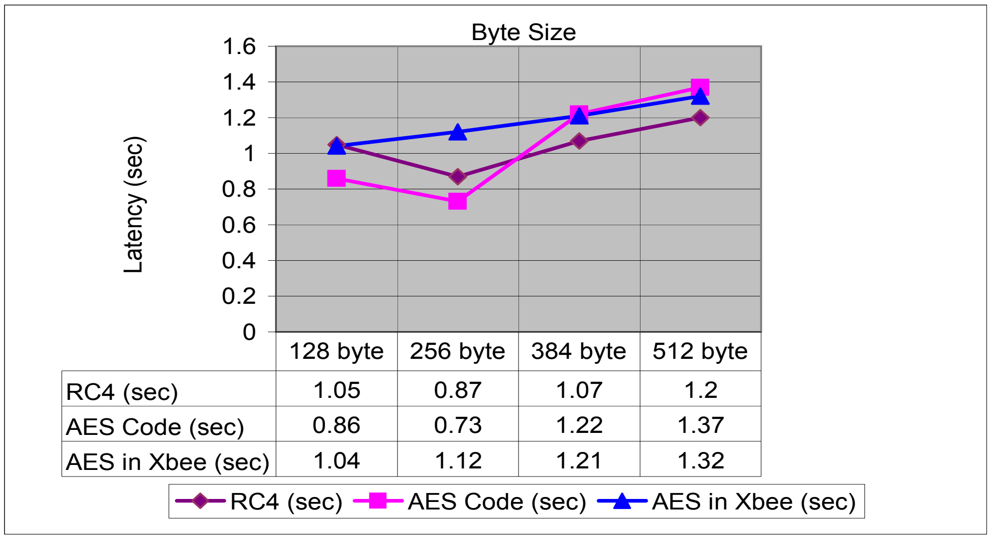
Figure 5.6: Latency values in Node 2 for baud rate 9600

For the next tests, the baud rate is increased to 19200. Figures 5.7 and 5.8 show the results for Node 1 and Node 2. In both cases, we see similar trend for RC4. RC4 starts with a higher latency but temporarily lowers before increasing consistently. Although AES in XBee has a longer latency than the other two strategies, it has a very consistent growing trend. Finally, the AES code shows some contradictory behavior where it takes les time for the biggest data size compared to the smaller data sizes. Figure 5.8, the latency for the AES in XBee increases with an increasing data size. RC4 and AES encryption decrease a little from the starting point but later these two also increase with the increased data size.



	128 byte	256 byte	384 byte	512 byte
RC4 (sec)	1.21	0.95	0.97	1.12
AES Code (sec)	0.69	0.66	1.13	0.85
AES in Xbee (sec)	1.13	1.24	1.25	1.56

Figure 5.7: Latency values in Node 1 for baud rate 19200

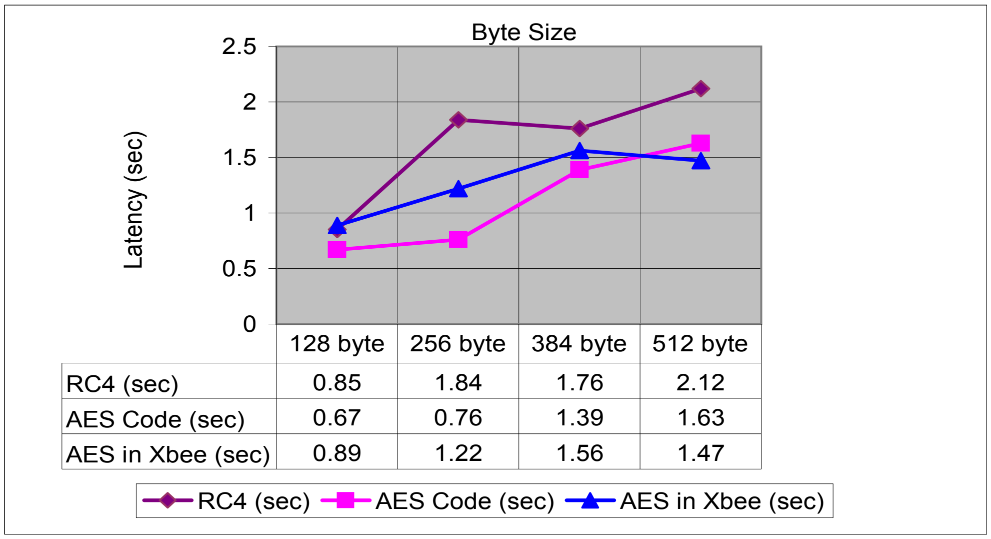


	128 byte	256 byte	384 byte	512 byte
RC4 (sec)	1.05	0.87	1.07	1.2
AES Code (sec)	0.86	0.73	1.22	1.37
AES in Xbee (sec)	1.04	1.12	1.21	1.32

Figure 5.8: Latency values for Node 2 for baud rate 19200

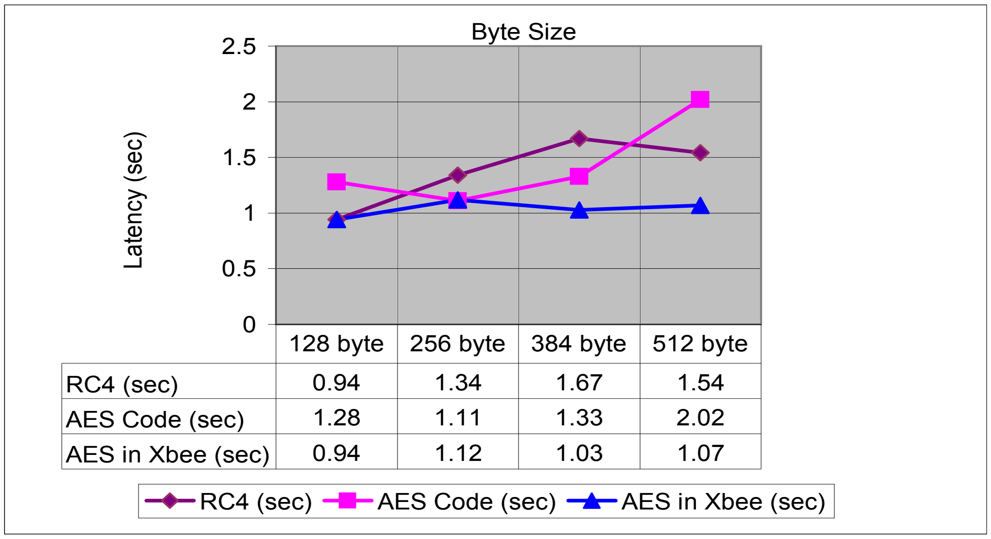
Next, Figure 5.9 and 5.10 show the results at a baud rate of 38400. For Node 1, the results start with a very low latency as shown in Figure 5.9. The communication is faster in smaller data. However, the latency increases with the increasing amount of data size over the course of time. Unlike Node 1, for Node 2 in Figure 5.10, the results are different for the RC4, the AES code and the AES in XBee. The latency range is almost similar but their behavior with same data size differs. RC4 and AES in XBee shows more constant values with the increasing data size whereas the AES code results are almost similar to those of

Node 1.



	128 byte	256 byte	384 byte	512 byte
RC4 (sec)	0.85	1.84	1.76	2.12
AES Code (sec)	0.67	0.76	1.39	1.63
AES in Xbee (sec)	0.89	1.22	1.56	1.47

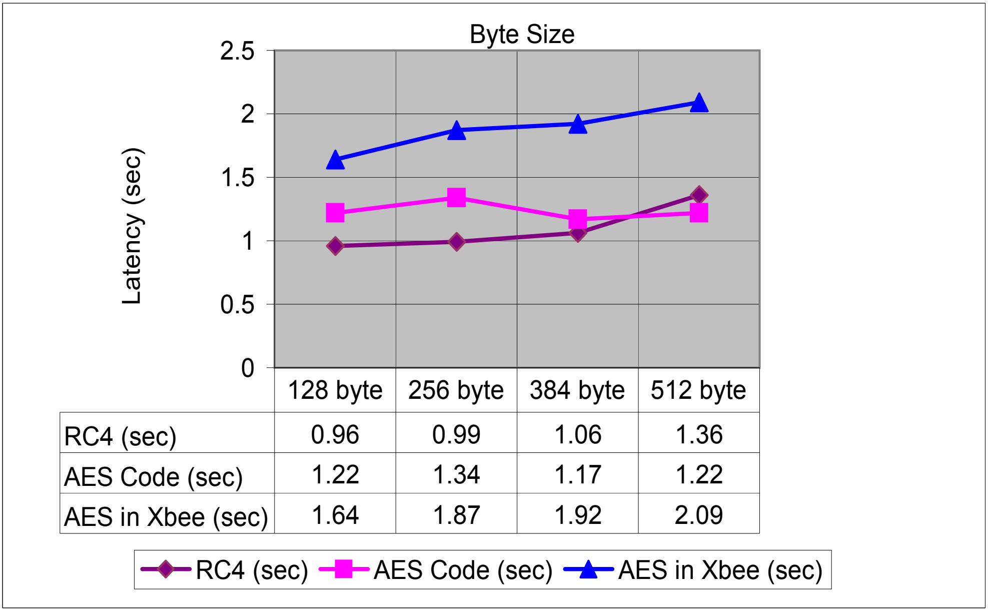
Figure 5.9: Latency values in Node 1 for baud rate 38400



	128 byte	256 byte	384 byte	512 byte
RC4 (sec)	0.94	1.34	1.67	1.54
AES Code (sec)	1.28	1.11	1.33	2.02
AES in Xbee (sec)	0.94	1.12	1.03	1.07

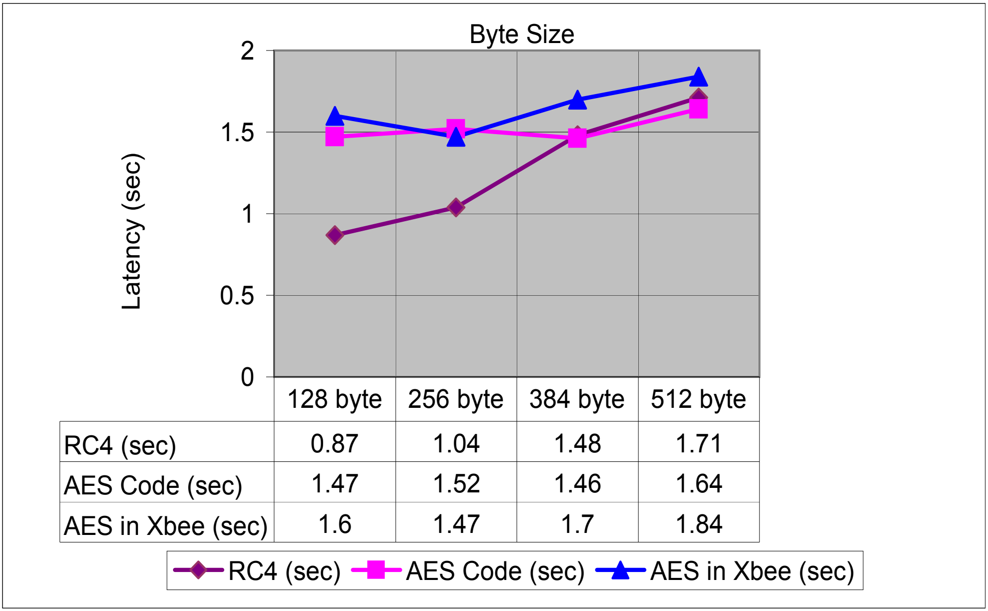
Figure 5.10: Latency values in Node 2 for baud rate 38400

In Figure 5.11, the latency results show a comparatively better result for baud rate 56700 possibly because more data is passing per second in higher baud rate . Latency for the RC4 and the AES in XBee increase slightly with the data size, while the AES Code result maintains a constant value throughout the different data size.



	128 byte	256 byte	384 byte	512 byte
RC4 (sec)	0.96	0.99	1.06	1.36
AES Code (sec)	1.22	1.34	1.17	1.22
AES in Xbee (sec)	1.64	1.87	1.92	2.09

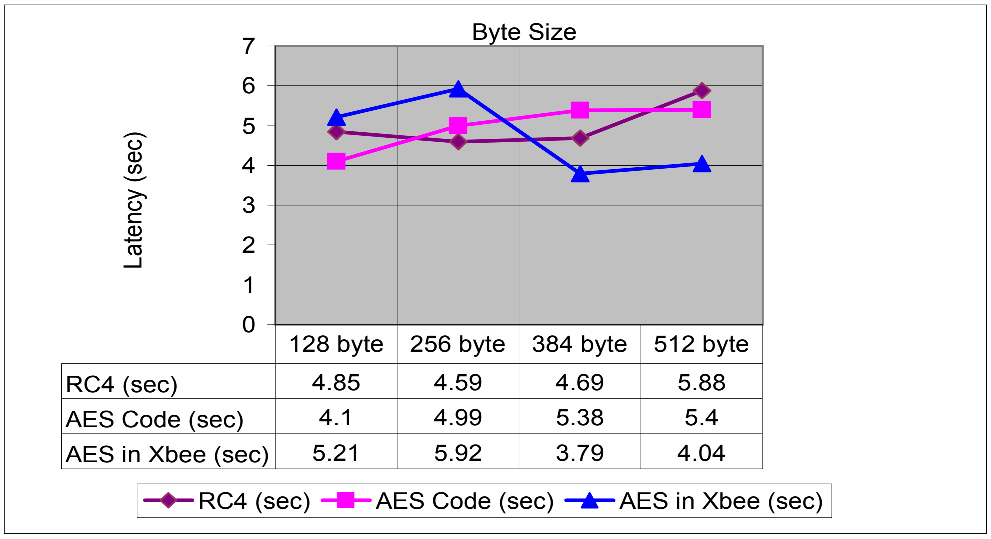
Figure 5.11: Latency values in Node 1 for baud rate 56700



	128 byte	256 byte	384 byte	512 byte
RC4 (sec)	0.87	1.04	1.48	1.71
AES Code (sec)	1.47	1.52	1.46	1.64
AES in Xbee (sec)	1.6	1.47	1.7	1.84

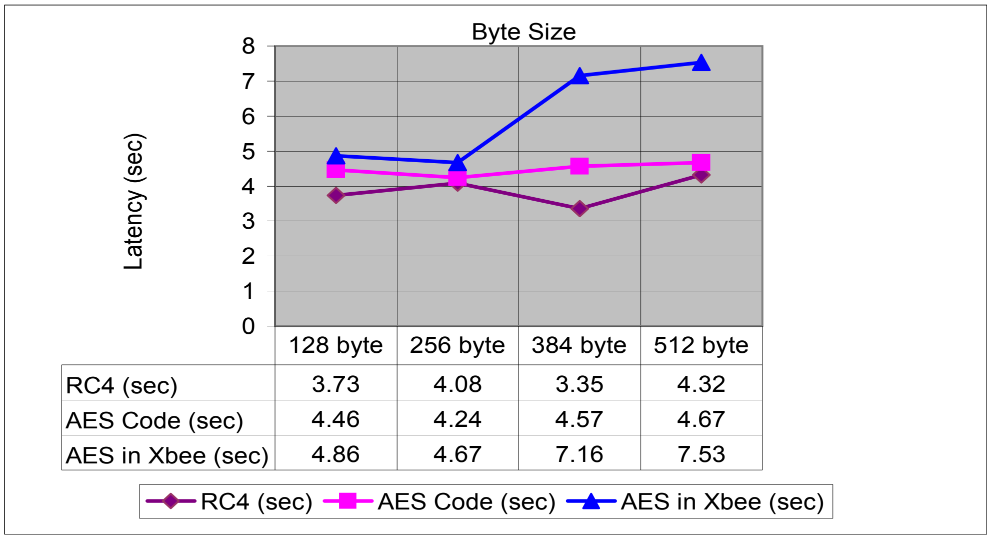
Figure 5.12: Latency values in Node 2 for baud rate 56700

Figure 5.12 shows the results for baud rate 56700 in Node 2. They all increase as the data size increases. Therefore, latency is higher in both figures. RC4 shows the most reliable and steady result and AES in XBee also increases steadily after 256 byte.



	128 byte	256 byte	384 byte	512 byte
RC4 (sec)	4.85	4.59	4.69	5.88
AES Code (sec)	4.1	4.99	5.38	5.4
AES in Xbee (sec)	5.21	5.92	3.79	4.04

Figure 5.13: Latency values in Node 3 for baud rate 9600



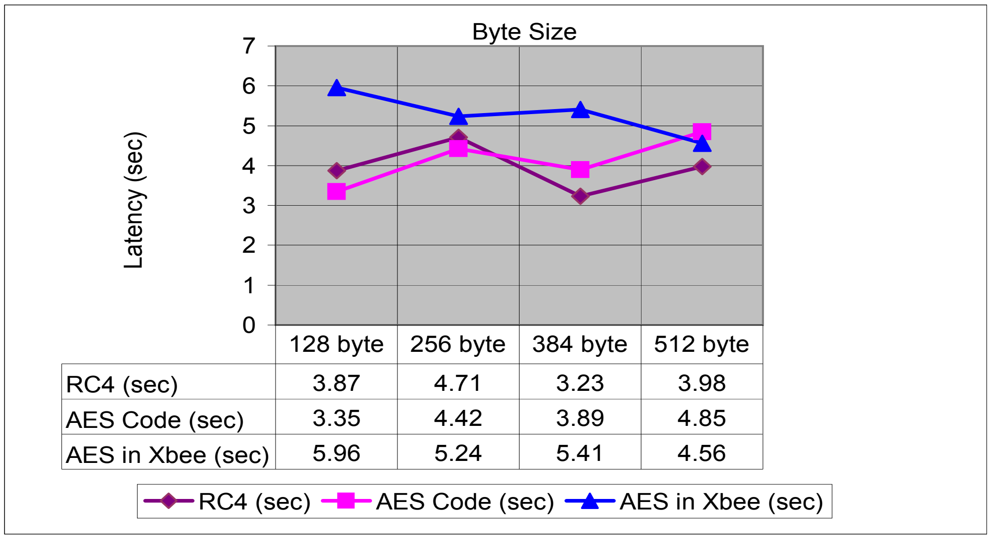
	128 byte	256 byte	384 byte	512 byte
RC4 (sec)	3.73	4.08	3.35	4.32
AES Code (sec)	4.46	4.24	4.57	4.67
AES in Xbee (sec)	4.86	4.67	7.16	7.53

Figure 5.14: Latency values in Node 4 for baud rate 9600

Node 3 and 4 are out-of-range nodes which is why these two nodes take more time to travel to receive the same amount of data than Nodes 1 and 2. Figure 5.13 and 5.14 indicate the increased amount of latency in Node 3 and Node 4 for the 9600 baud rate.

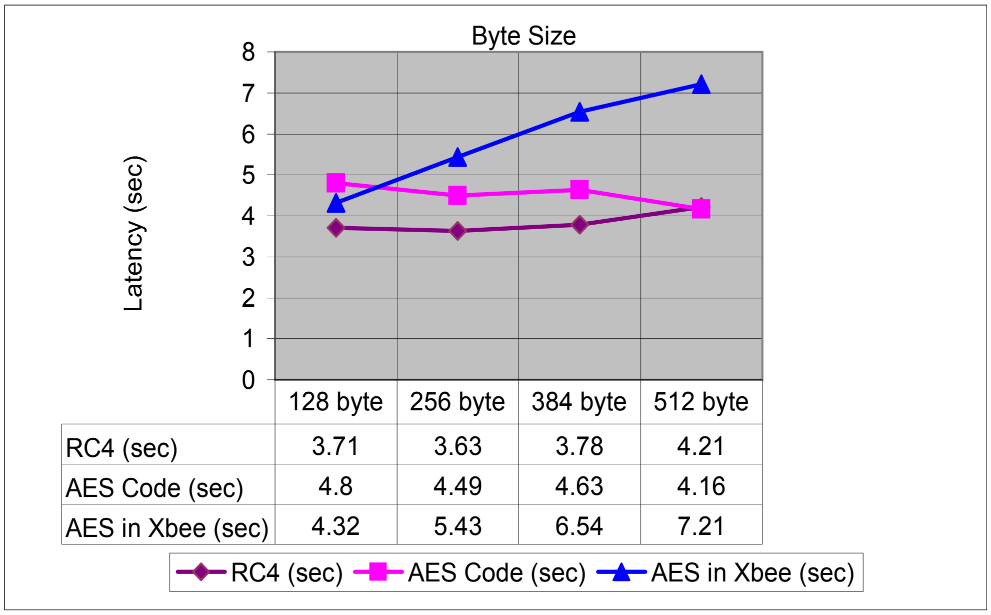
In both cases, the lowest latency is more than 3.5 sec and it increases to 8 sec. Therefore, Latency value differences are very evident than the value differences of Node 1 and Node 2. RC4 maintains a steady value with increasing data size. But, for the AES in XBee, the latency increase briefly before decreasing.

A similar comparison can be presented for 19200 baud rate, which is shown in Figure 5.15 and 5.16 .



	128 byte	256 byte	384 byte	512 byte
RC4 (sec)	3.87	4.71	3.23	3.98
AES Code (sec)	3.35	4.42	3.89	4.85
AES in Xbee (sec)	5.96	5.24	5.41	4.56

Figure 5.15: Latency values in Node 3 for baud rate 19200

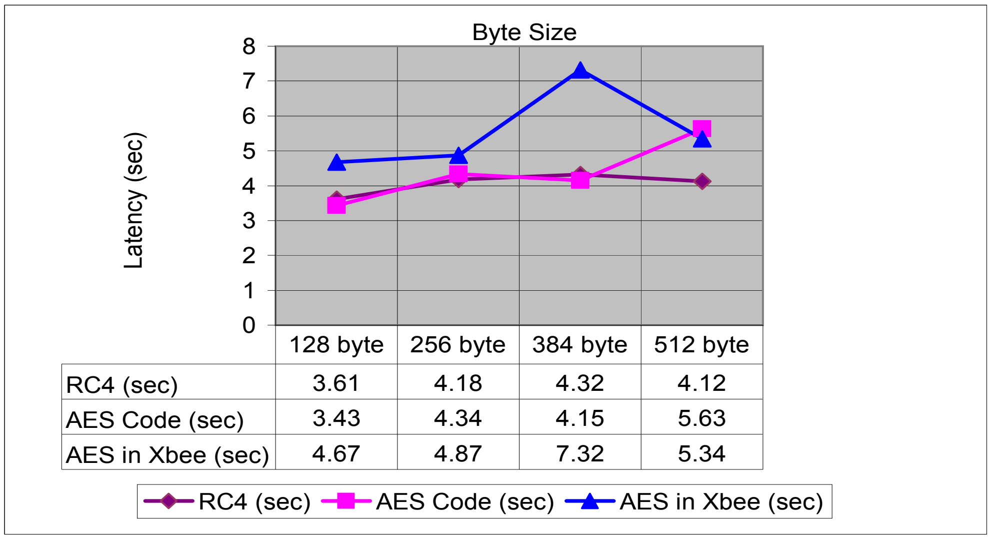


	128 byte	256 byte	384 byte	512 byte
RC4 (sec)	3.71	3.63	3.78	4.21
AES Code (sec)	4.8	4.49	4.63	4.16
AES in Xbee (sec)	4.32	5.43	6.54	7.21

Figure 5.16: Latency values in Node 4 for baud rate 19200

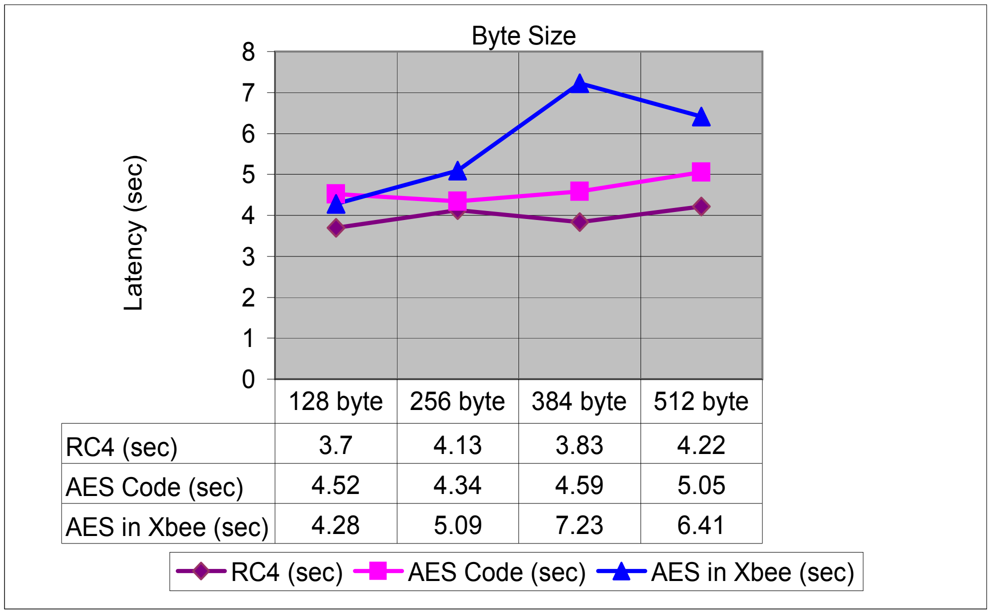
Node 3 does not show many changes, but Node 4 shows a lot of improvement in stability. Latency increases smoothly and distinctively when compared to Node 3. In Node 4, latency is increasing for RC4 and AES in XBee but results of AES code remains almost similar for all data sizes. But at the same time in Node 3, the latency fluctuates over time for different data sizes.

In Figures 5.17 and 5.18, RC4 shows more evidence of steady performance for the 38400 baud rate for Nodes 3 and 4.



	128 byte	256 byte	384 byte	512 byte
RC4 (sec)	3.61	4.18	4.32	4.12
AES Code (sec)	3.43	4.34	4.15	5.63
AES in Xbee (sec)	4.67	4.87	7.32	5.34

Figure 5.17: Latency values in Node 3 for baud rate 38400

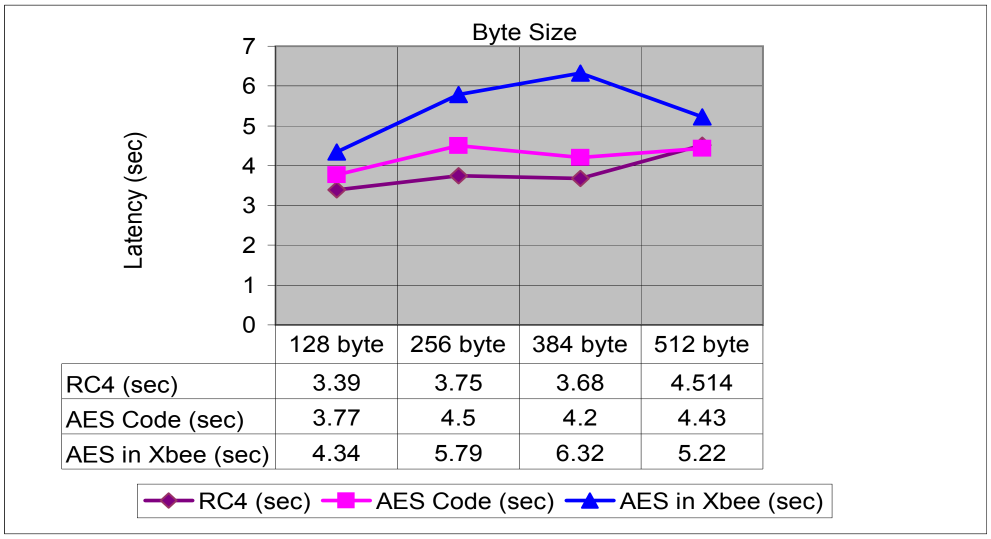


	128 byte	256 byte	384 byte	512 byte
RC4 (sec)	3.7	4.13	3.83	4.22
AES Code (sec)	4.52	4.34	4.59	5.05
AES in Xbee (sec)	4.28	5.09	7.23	6.41

Figure 5.18: Latency values in Node 4 for baud rate 38400

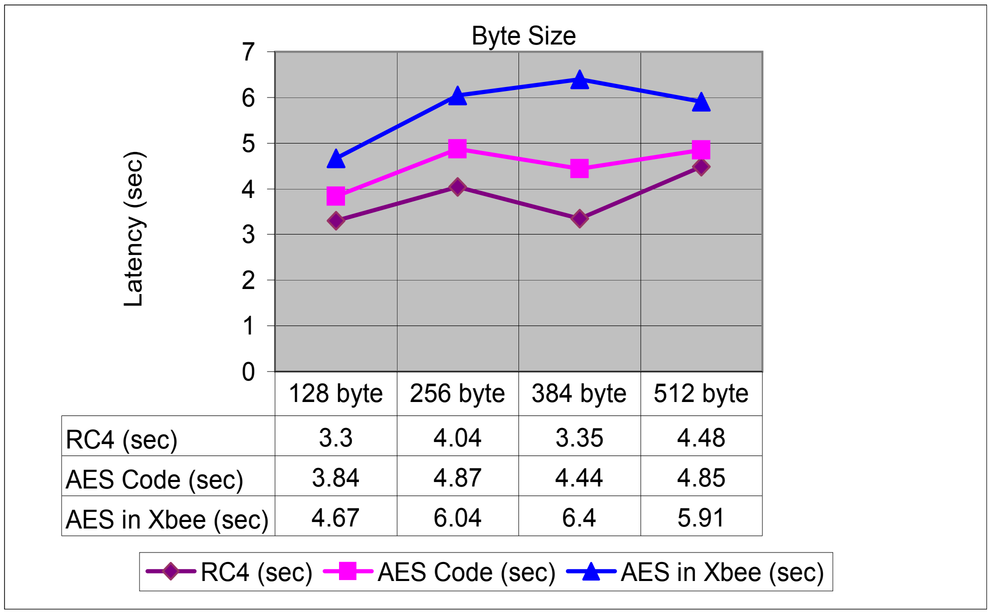
This node in the RC4 starts with very small amount of latency and increases with each data size but the increment amount is not very big. The AES code latency increases with proper interval, but the AES in XBee shows some unstable and unpredictable behavior.

In Figure 5.19 and 5.20, another stable or gradually increasing performance is shown by Nodes 3 and 4 for the baud rate 56700.



	128 byte	256 byte	384 byte	512 byte
RC4 (sec)	3.39	3.75	3.68	4.514
AES Code (sec)	3.77	4.5	4.2	4.43
AES in Xbee (sec)	4.34	5.79	6.32	5.22

Figure 5.19: Latency values in Node 3 for baud rate 57600



	128 byte	256 byte	384 byte	512 byte
RC4 (sec)	3.3	4.04	3.35	4.48
AES Code (sec)	3.84	4.87	4.44	4.85
AES in Xbee (sec)	4.67	6.04	6.4	5.91

Figure 5.20: Latency values in Node 4 for baud rate 57600

The RC4 and the AES code latencies behave similarly and show a similar growth pattern. The AES in XBee shows unstable latency values. One interesting fact here is that, for Node 1 and Node 2, AES in XBee show a stable trend in latency than AES code, but in Node 3 and 4, the AES code latencies show stability and gradual increase in almost all the cases.

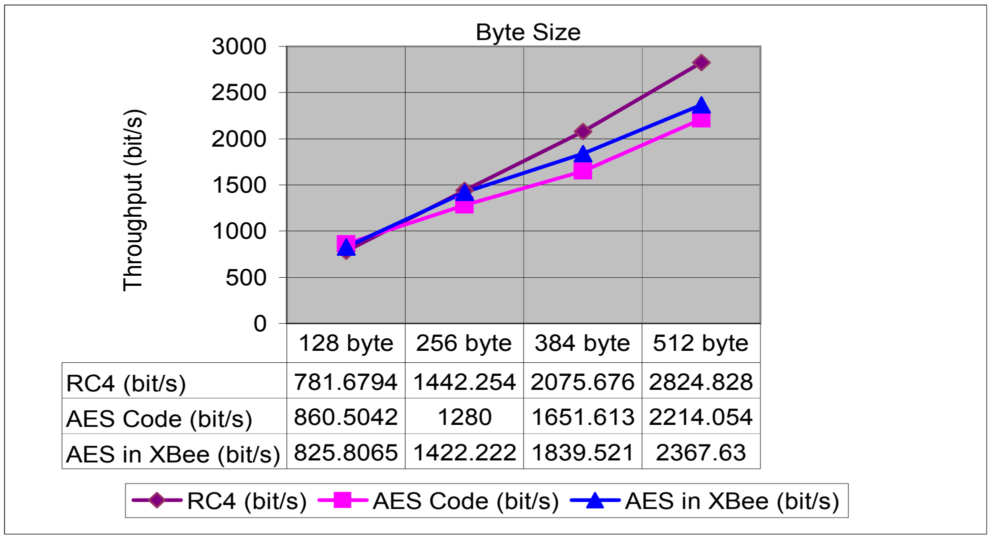
After analyzing Latency in four nodes, it is apparent that the RC4 shows more stable latency. Despite a fewer fluctuations in this trend, RC4 shows a gradually increasing trend or stable behavior for most of the baud rates. The time taken by the RC4 is also lower than the other two encryptions. RC4 starts with a lower latency and does not fluctuate widely. There are a few observations that need to be made about latency. The latency may have different times for even the same baud rate and the same data size. There are a few reasons for the different latency each time. One of the major reasons is the error correction mechanism. The error correction is also known as Reliable Delivery mode in XBee. This mode can be enabled by settingthe ATRR (Retries) value command to a nonzero value. The ATRR value determines how many times the sender would retry to send data if no acknowledgement from the receiver. If it is enabled, then the XBee sends an RF Packet header to the receiving module to request an ACK (acknowledgement) to be sent. If no ACK is received within 200ms, it resends the packet within a random period of a few milliseconds. The default retry by XBee is three times if the acknowledgement is not received [43]. Packetization Timeout (PT) has a default value for the character delays for different baud rates. The XBee sets the PT value to wait before sending the data it receives from the Arduino. If we set the PT value too small, XBee does not wait for more data from Arduino and sends data in different packets, though the two data are supposed to be in the same packet. This is why XBee waits the default 3ms before sending any data via its transmitter. This is a fixed amount, but to meet system requirements, XBee may change it to a wider margin. Clear Channel Assessment (CCA) checks whether a channel is free or not before sending data. If it is occupied, then XBee waits until it is free before sending the next packet of data. This can change the delay time significantly. These are a few reasons the delay time is different despite unchanged parameters [43].

5.1.3 Throughput

Packet size and the speed are two major factors when calculating throughput performance of a wireless device. Throughput is the amount of data sent or received during a defined period of time. It is calculated as the packet size divided by the total transmission time as indicated in the equation below [18]:

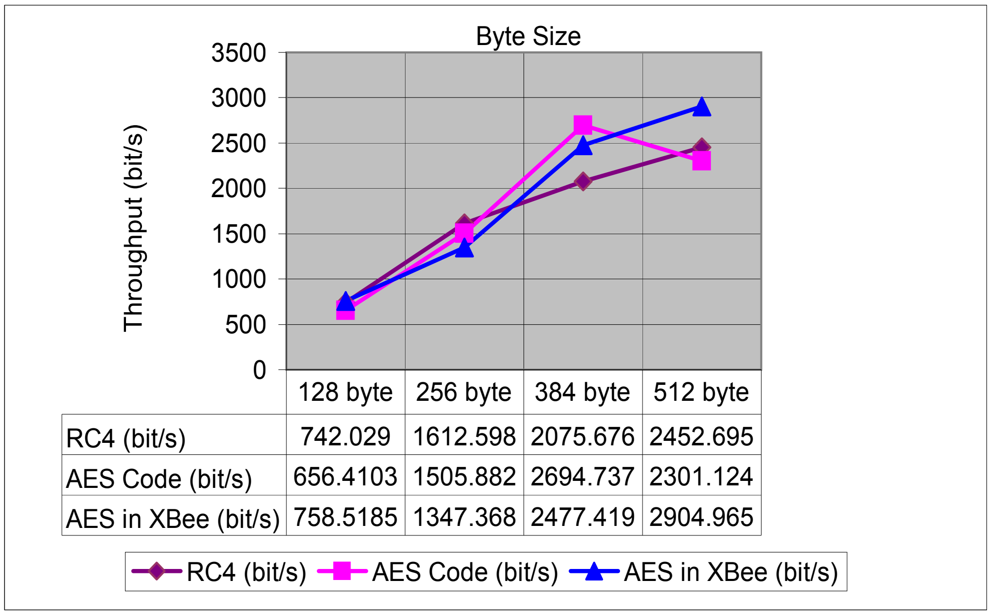
$$\text{TP} = \frac{8\times \text{ number of bytes}}{\text{total transmission time (sec)}} \text{ [18]}$$

I measure the values for throughput for different baud rates and packet sizes. This work is related to the previous Latency experiment. For the Latency experiments we obtain the total transmission time from one end to another. This time can also be used for Throughput calculations. If we consider the above equation to calculate throughput, we see that the amount of bytes is divided by the latency time. So, by using the equation, we can measure the throughput.



	128 byte	256 byte	384 byte	512 byte
RC4 (bit/s)	781.6794	1442.254	2075.676	2824.828
AES Code (bit/s)	860.5042	1280	1651.613	2214.054
AES in XBee (bit/s)	825.8065	1422.222	1839.521	2367.63

Figure 5.21: Throughput values in Node 1 for baud rate 9600

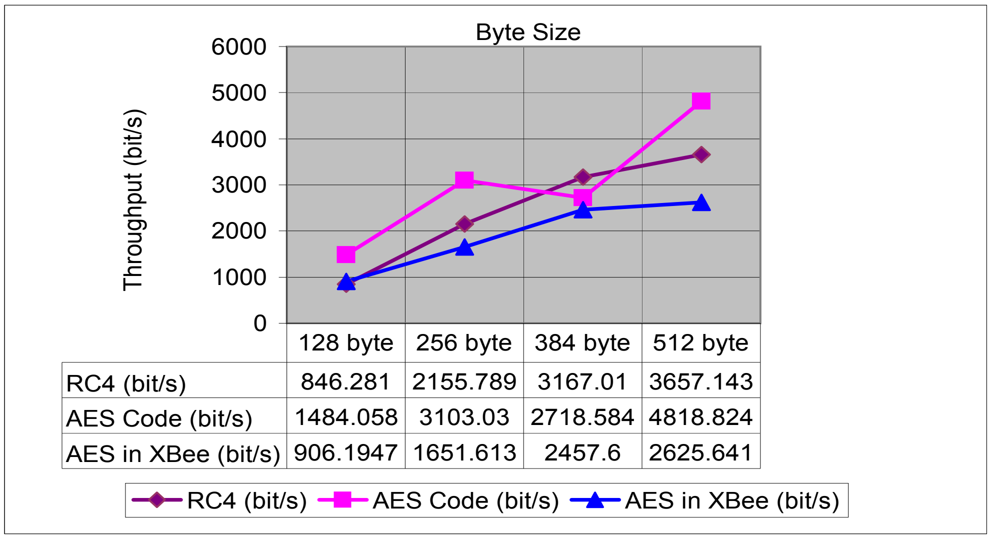


	128 byte	256 byte	384 byte	512 byte
RC4 (bit/s)	742.029	1612.598	2075.676	2452.695
AES Code (bit/s)	656.4103	1505.882	2694.737	2301.124
AES in XBee (bit/s)	758.5185	1347.368	2477.419	2904.965

Figure 5.22: Throughput values in Node 2 for baud rate 9600

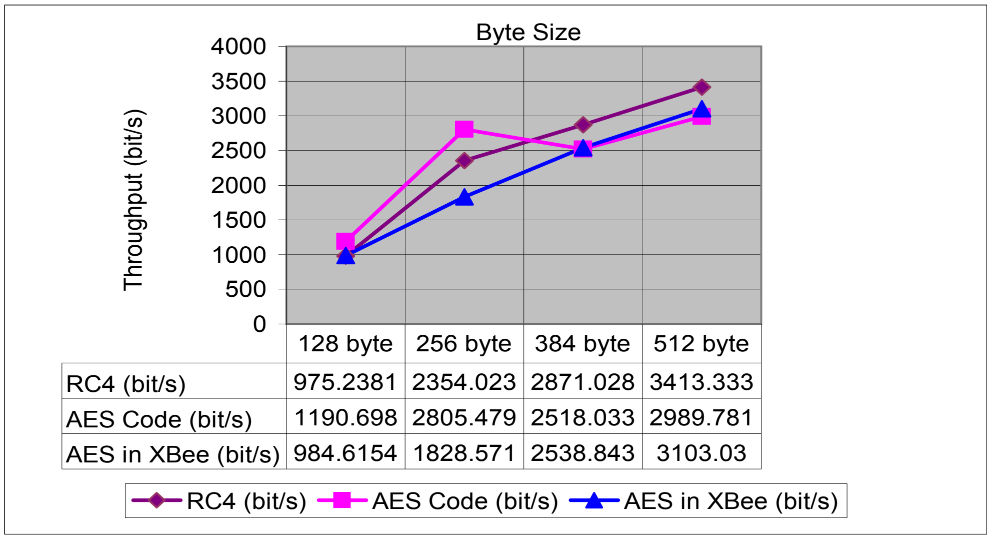
Figures 5.21 and 5.22 show the throughput for Node 1 and Node 2 at the 9600 baud rate. Most of the throughput values increase with the increased byte amount. In node 1, RC4 maintains a higher throughput in entire time except for a drop at 512 bytes. Though the AES code has a similar initial throughput as RC4 and AES in XBee, it can not keep the same trend till the end. It is always increasing but lags behind the other two encryption values. For Node 2, RC4 throughput can not maintain same high value after 256 bytes and starts lagging behind to other approaches. In both nodes, security measures show a significant increase in throughput with the increased data size.

Figure 5.23 and 5.24 show results for the 19200 baud rate. Node 1 and Node 2 show similar behavior for throughput. This time, the range of the throughput amount is increased with data size. RC4 shows comparatively constant throughput trend in two Nodes. AES code throughput fluctuates in both cases.



	128 byte	256 byte	384 byte	512 byte
RC4 (bit/s)	846.281	2155.789	3167.01	3657.143
AES Code (bit/s)	1484.058	3103.03	2718.584	4818.824
AES in XBee (bit/s)	906.1947	1651.613	2457.6	2625.641

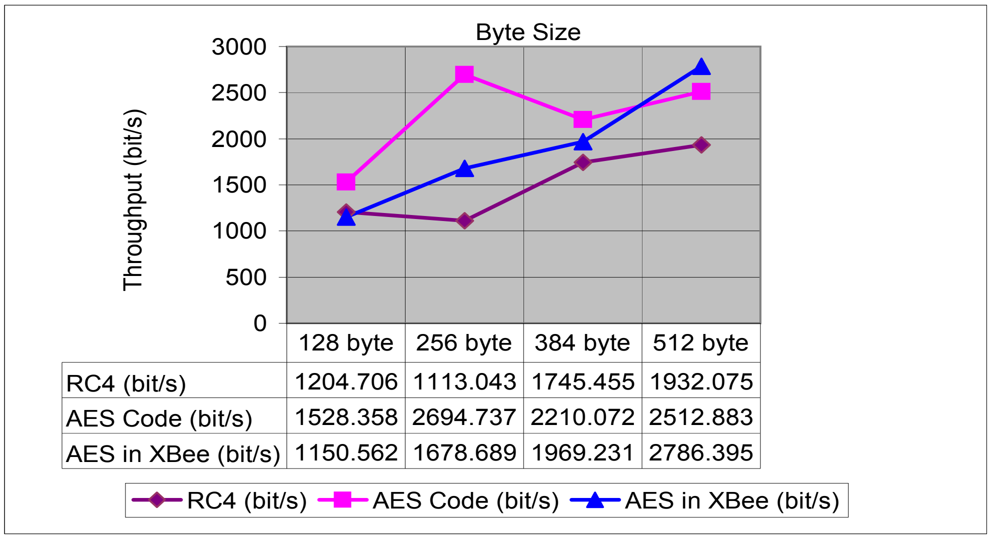
Figure 5.23: Throughput values in Node 1 for baud rate 19200



	128 byte	256 byte	384 byte	512 byte
RC4 (bit/s)	975.2381	2354.023	2871.028	3413.333
AES Code (bit/s)	1190.698	2805.479	2518.033	2989.781
AES in XBee (bit/s)	984.6154	1828.571	2538.843	3103.03

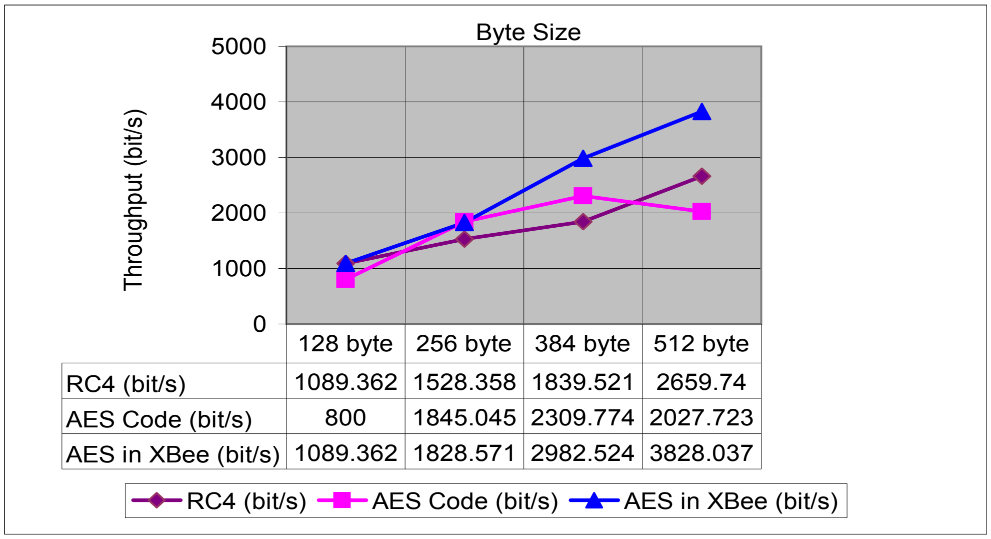
Figure 5.24: Throughput values in Node 2 for baud rate 19200

Figure 5.25 and 5.26 show the results for the the 38400 baud rate for Node 1 and Node 2. AES in XBee appears to be very robust, it throughput increases gradually, the amount it reaches is close to RC4. Also, RC4 lags behind the AES in XBee. Although it also gradually increases, it can not reach the amount to exceed the graph leaders in both nodes.



	128 byte	256 byte	384 byte	512 byte
RC4 (bit/s)	1204.706	1113.043	1745.455	1932.075
AES Code (bit/s)	1528.358	2694.737	2210.072	2512.883
AES in XBee (bit/s)	1150.562	1678.689	1969.231	2786.395

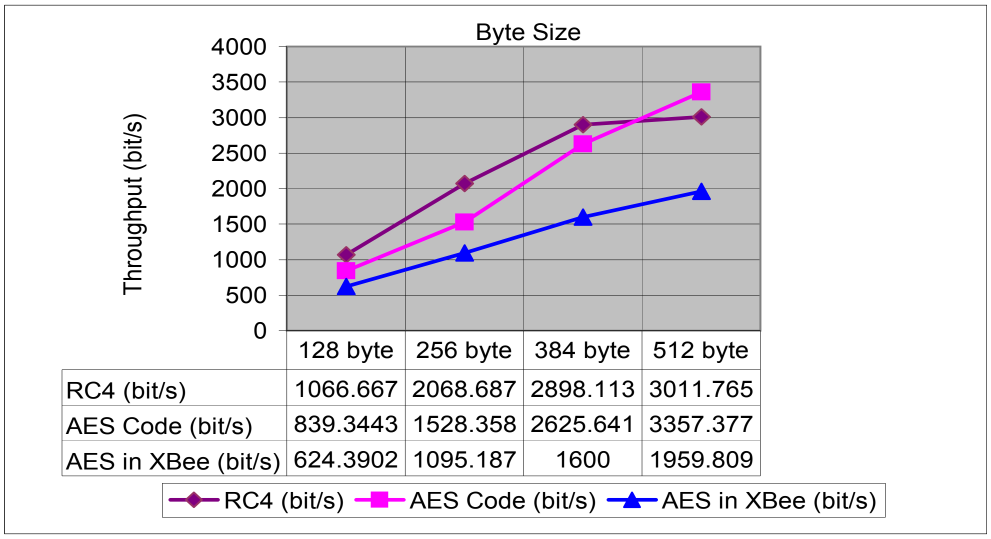
Figure 5.25: Throughput values in Node 1 for baud rate 38400



	128 byte	256 byte	384 byte	512 byte
RC4 (bit/s)	1089.362	1528.358	1839.521	2659.74
AES Code (bit/s)	800	1845.045	2309.774	2027.723
AES in XBee (bit/s)	1089.362	1828.571	2982.524	3828.037

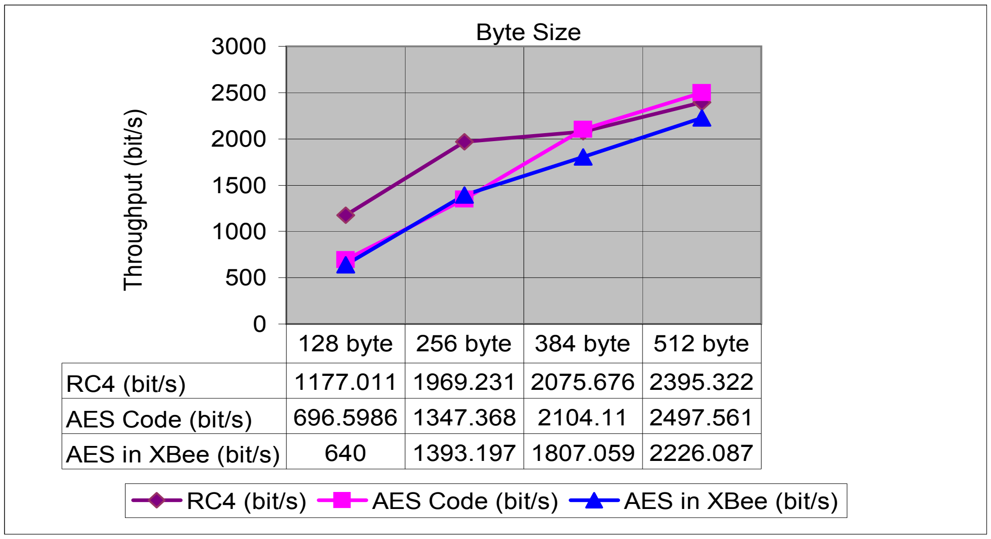
Figure 5.26: Throughput values in Node 2 for baud rate 38400

Figure 5.27 and 5.28 shows the results for the last baud rate 57600. RC4 initially has the highest throughput values and keeps increasing. and but AES code and AES in XBee shows steady increasing behavior with increased data sizes.



	128 byte	256 byte	384 byte	512 byte
RC4 (bit/s)	1066.667	2068.687	2898.113	3011.765
AES Code (bit/s)	839.3443	1528.358	2625.641	3357.377
AES in XBee (bit/s)	624.3902	1095.187	1600	1959.809

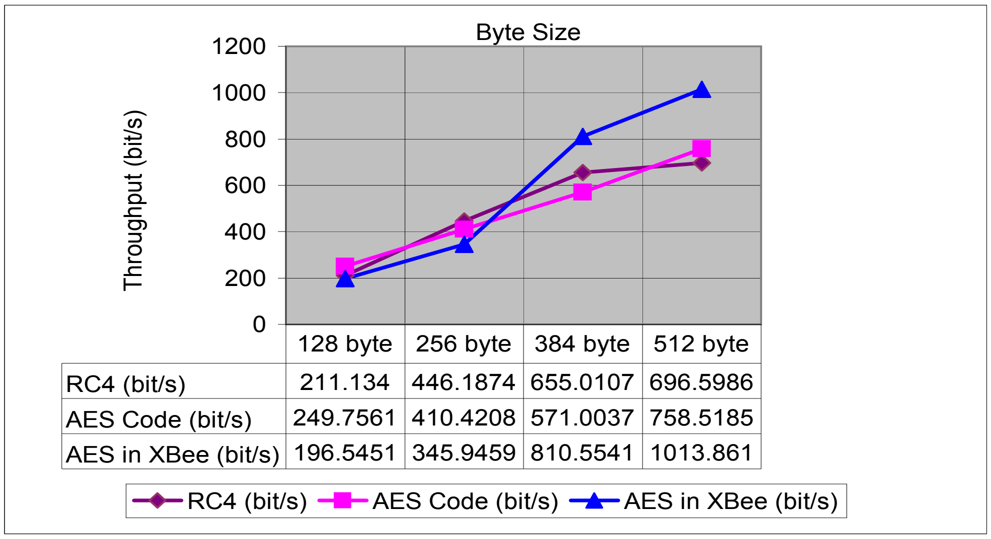
Figure 5.27: Throughput values in Node 1 for baud rate 56700



	128 byte	256 byte	384 byte	512 byte
RC4 (bit/s)	1177.011	1969.231	2075.676	2395.322
AES Code (bit/s)	696.5986	1347.368	2104.11	2497.561
AES in XBee (bit/s)	640	1393.197	1807.059	2226.087

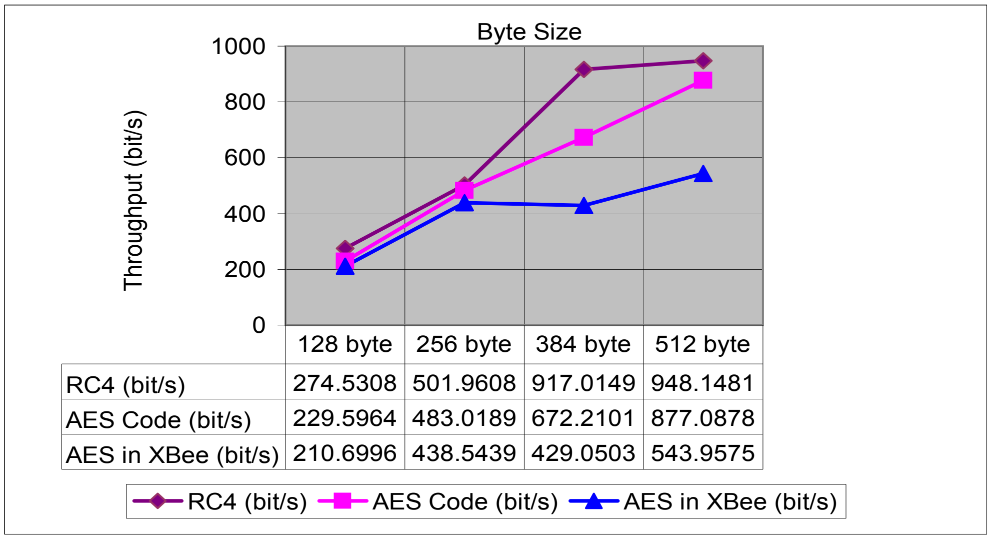
Figure 5.28: Throughput values in Node 2 for baud rate 56700

Next, we consider the throughput for Node 3 and Node 4 at the 9600 baud rate in Figures 5.29 and 5.30. We see that in node 3, the AES in XBee is leading the chart, while RC4 is lagging behind. The amount of overall throughput is small because the latency is higher. In Node 4, the RC4 grows sharply at the 256 byts and surpasses the AES encryption. This time, AES in XBee performs with lower throughput.



	128 byte	256 byte	384 byte	512 byte
RC4 (bit/s)	211.134	446.1874	655.0107	696.5986
AES Code (bit/s)	249.7561	410.4208	571.0037	758.5185
AES in XBee (bit/s)	196.5451	345.9459	810.5541	1013.861

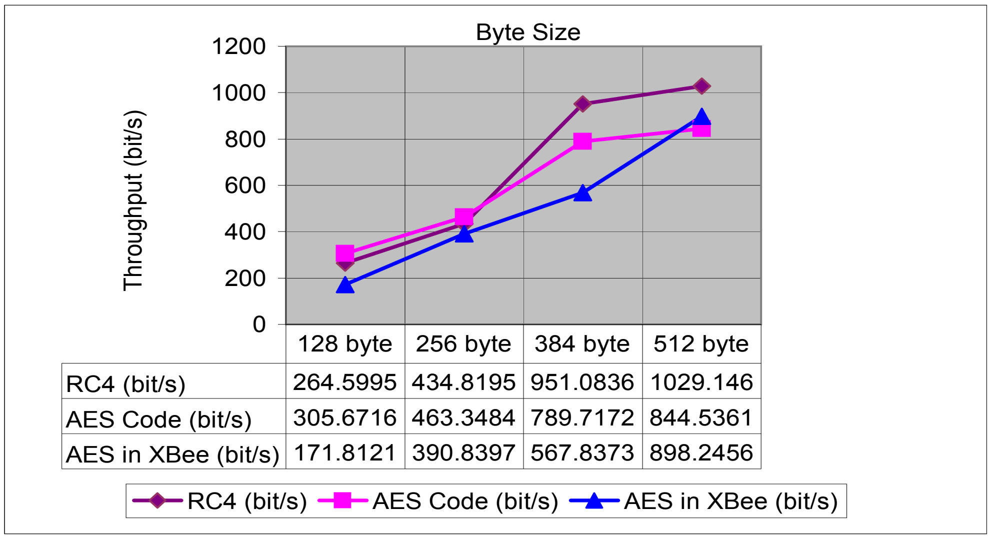
Figure 5.29: Throughput values in Node 3 for baud rate 9600



	128 byte	256 byte	384 byte	512 byte
RC4 (bit/s)	274.5308	501.9608	917.0149	948.1481
AES Code (bit/s)	229.5964	483.0189	672.2101	877.0878
AES in XBee (bit/s)	210.6996	438.5439	429.0503	543.9575

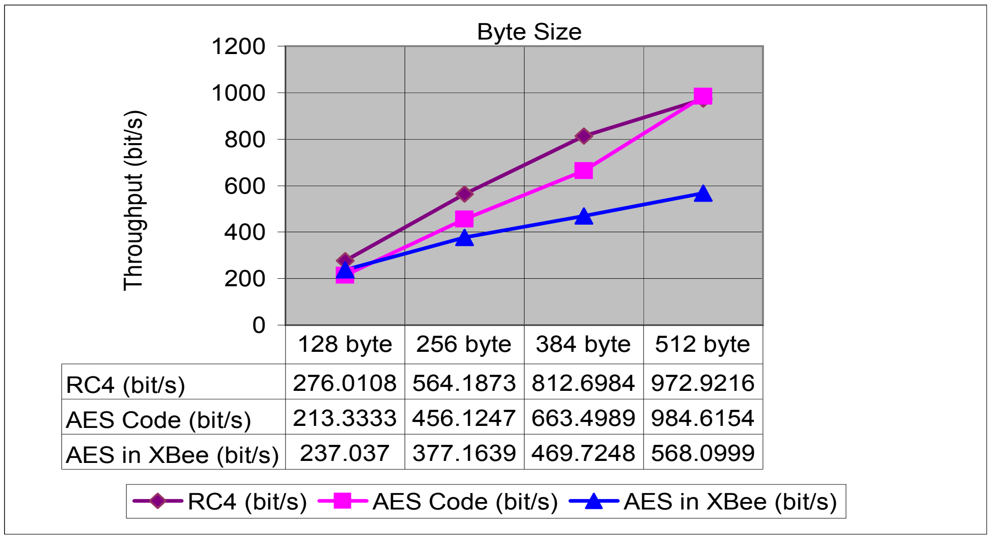
Figure 5.30: Throughput values in Node 4 for baud rate 9600

Node 3 and Node 4 act similarly for the baud rate 19200. In Figure 5.31 and 5.32, it is evident that the RC4 leads in both the graphs while the other two increase gradually in larger data size.



	128 byte	256 byte	384 byte	512 byte
RC4 (bit/s)	264.5995	434.8195	951.0836	1029.146
AES Code (bit/s)	305.6716	463.3484	789.7172	844.5361
AES in XBee (bit/s)	171.8121	390.8397	567.8373	898.2456

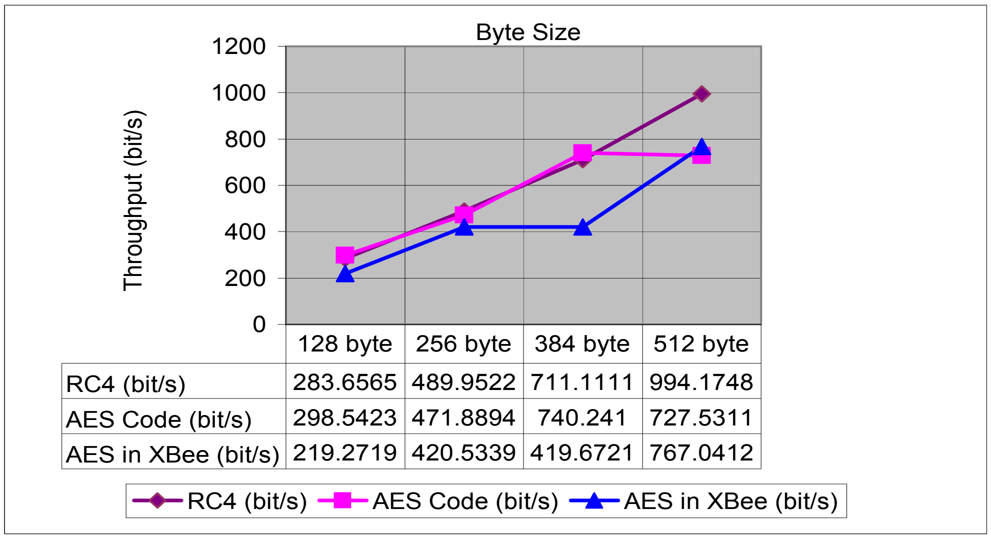
Figure 5.31: Throughput values in Node 3 for baud rate 19200



	128 byte	256 byte	384 byte	512 byte
RC4 (bit/s)	276.0108	564.1873	812.6984	972.9216
AES Code (bit/s)	213.3333	456.1247	663.4989	984.6154
AES in XBee (bit/s)	237.037	377.1639	469.7248	568.0999

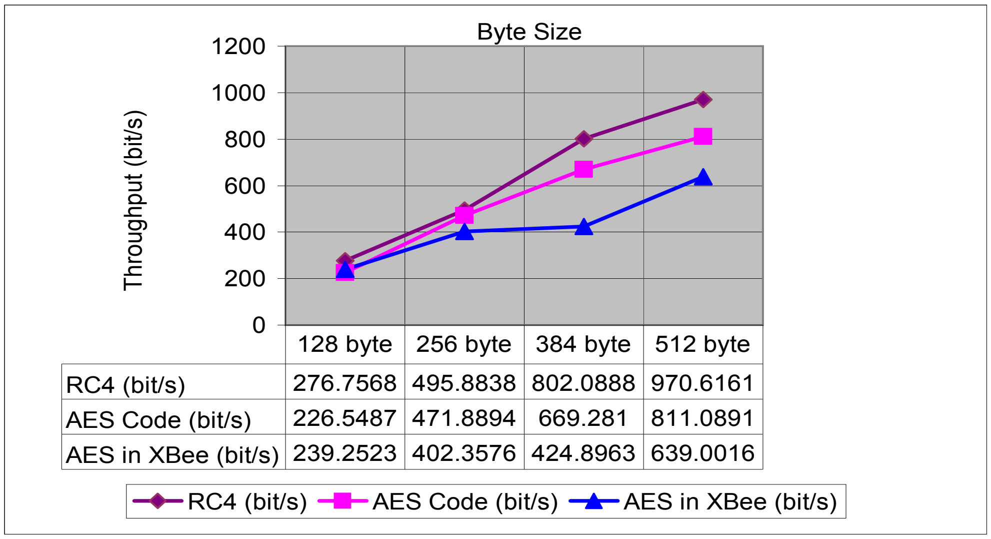
Figure 5.32: Throughput values in Node 4 for baud rate 19200

In Figure 5.31, the RC4 lags behind AES code. On the other hand AES in XBee increases gradually, though the throughput is lower than what the RC4 achieves. In Figure 5.32, AES code has a similar trend as RC4 encryption and later gives almost the same throughput.



	128 byte	256 byte	384 byte	512 byte
RC4 (bit/s)	283.6565	489.9522	711.1111	994.1748
AES Code (bit/s)	298.5423	471.8894	740.241	727.5311
AES in XBee (bit/s)	219.2719	420.5339	419.6721	767.0412

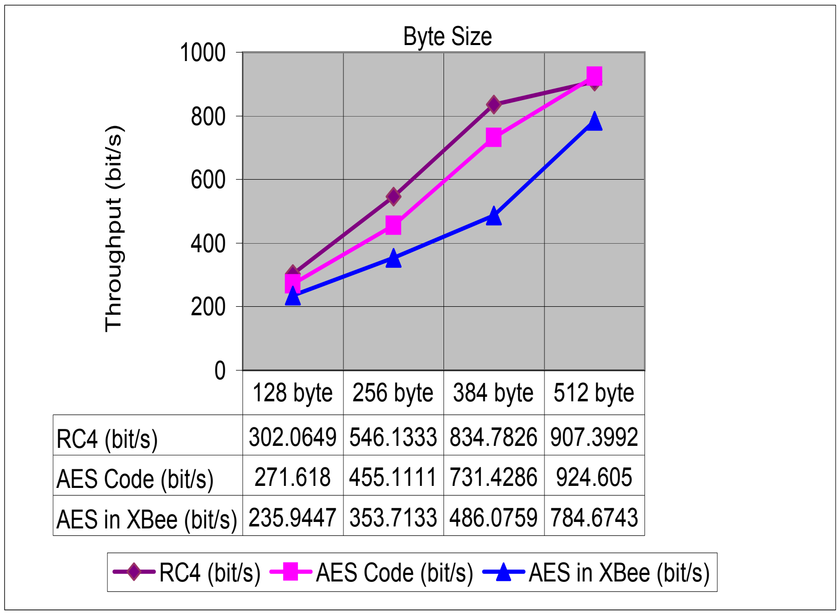
Figure 5.33: Throughput values in Node 3 for baud rate 38400



	128 byte	256 byte	384 byte	512 byte
RC4 (bit/s)	276.7568	495.8838	802.0888	970.6161
AES Code (bit/s)	226.5487	471.8894	669.281	811.0891
AES in XBee (bit/s)	239.2523	402.3576	424.8963	639.0016

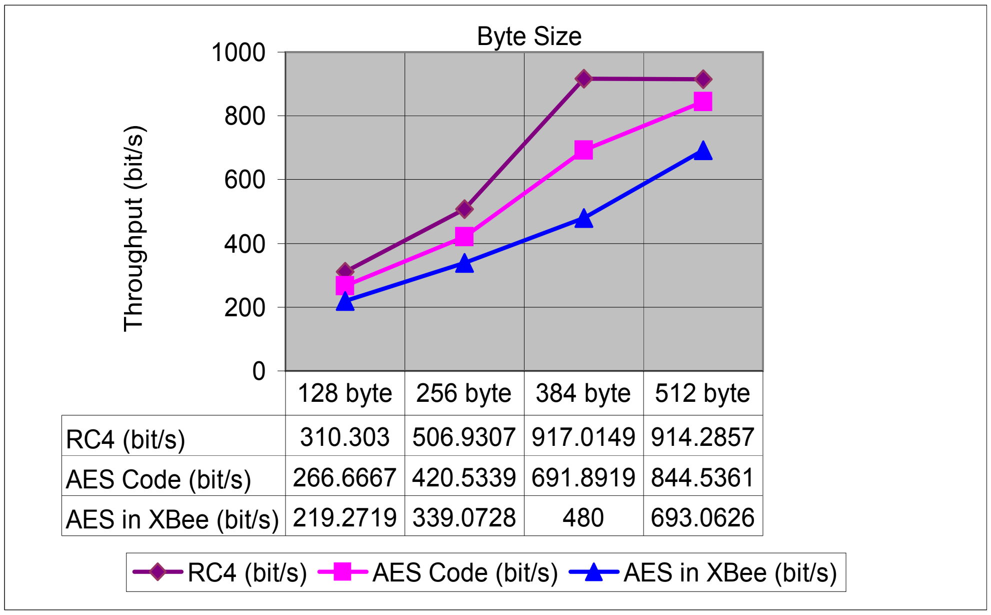
Figure 5.34: Throughput values in Node 4 for baud rate 38400

Figure 5.33 and 5.34 show the amount of throughput for Node 3 and Node 4 at the baud rate 38400. From Node 3, we see that RC4 and the AES code have a similar type of throughput, but by 512 bytes, RC4 performs better than the AES. For Node 4, the RC4 increases steadily with the increasing data size. Generally in all these cases the RC4 remains above in throughput than the other contenders except at 384 byte in Node 3.



	128 byte	256 byte	384 byte	512 byte
RC4 (bit/s)	302.0649	546.1333	834.7826	907.3992
AES Code (bit/s)	271.618	455.1111	731.4286	924.605
AES in XBee (bit/s)	235.9447	353.7133	486.0759	784.6743

Figure 5.35: Throughput values in Node 3 for baud rate 56700



	128 byte	256 byte	384 byte	512 byte
RC4 (bit/s)	310.303	506.9307	917.0149	914.2857
AES Code (bit/s)	266.6667	420.5339	691.8919	844.5361
AES in XBee (bit/s)	219.2719	339.0728	480	693.0626

Figure 5.36: Throughput values in Node 4 for baud rate 56700

For Figure 5.35 and 5.36 for the baud rate 56700 Node 3 and Node 4 show results that are almost similar to baud rate 38400. They both have the highest throughput at 900 bit/s. One thing is to be noted here is that unlike the other instances, here all the throughputs for every encryption are increasing till the end. For Node 3, the AES code encryption lags behind the RC4 almost always, but at the end, has a throughput that is similar to RC4. In the case of Node 4, RC4 maintains a gradual increase until 384 bytes. The other two values for the AES code and the AES in XBee increase but never surpass the RC4.

Throughput measures how fast data moves from one node to the another while latency only measure the time it takes to send the data between two nodes. Throughput can be affected by many factors. Packet loss, network congestion are some of the limitations which may affect throughput of a communication or a network. Latency and Throughput both are very crucial factors for analyzing the performance of a network. High throughput and low latency is desirable for the best network performance. For the same reason, as latency, the value of throughput in a particular approach is different in same baud rate and data size. Upon analyzing the above performances, it is apparent that the RC4 performs exceptionally well with respect to throughput, for both in-range and out-of-range nodes. The RC4 is ideal to use in small size hardware such as Arduino UNO as this is the fastest symmetric algorithm and needs a small number of operation to perform.

Chapter 6

Conclusion and Future Works

6.1 Contribution

The contributions of my thesis work includes the design of a key exchange protocol, its implementation, as well as its performance evaluation with different parameters, in a considerably newer wireless sensor node (i.e. XBee and Arduino UNO). Below I discuss some of the major contribution of the thesis work.

1. Protection against attacks : The major contribution of this thesis work is ensuring protection against different WSNs attacks. One of the major attacks we discuss in chapter 2 is sybil attack. In a sybil attack, the adversary tries to introduce itself as a valid node of the same network, whichever destabilize the network. In my proposed scheme, each node shares a unique symmetric key with a trusted base station, which can be used to verify the identity of the node by sending encrypted commands. Unfairness occurs during an exhaustion attack in a network [10]. In this case, the adversary continuously communicates with major nodes. In my proposed key exchange protocol, each node only communicates when the base station wants to communicate with it, so it does not respond to a false request for key negotiation generated by a fake node. This eliminates the possibility of exhaustion and unfairness attacks in this network. The same approach can prevent the Hello flood attacks too. In a Hello flood attack, the adversary uses the same neighbor approach and lures other nodes to channel their data towards it [10]. But in my work, because only the base station communicates with each sensor node, a sensor node does not listen to requests from any malicious node. My proposed scheme deals with all forms of authentication too.

The base station communicates with the sensor node using a secret key that ensures the user authentication. If a new senor node needs to be added to the network, the sensor node is pre-installed with a new key that is only known to the base station. So, in this case also the node authentication is maintained. Data authentication can be ensured by RC4 encryption that has been used in this proposed scheme. The secured data communication by the sensor node ensures the base station that the senor node is also authentic.

2. Reconfiguration : The key distribution and the encryption works together for the resiliency of the network. This encryption algorithm can be changed at the requirement of the application and users. Users can change the sensors too according to their requirement.

3. Real time key exchange : Another contribution, described in Chapter 4, shows the key changing capability by the user in XBee and Arduino UNO module. According to our knowledge, no works that associates authentication and encryption, have been implemented before in this hardware module. This shows a significant amount of minimized cost as well as usage of the different sensors in an open source equipment. This work can be extended more and additional measures can be added to it by improving the key exchange mechanism as well as utilizing different attributes of the Arduino UNO and the XBee.

4. Performance evaluation: Another contribution is discussed in Chapter 5, where performance analysis has been performed between the RC4 and AES encryptions in the XBee and Arduino UNO environment. Different parameters have been thoroughly discussed with the performance of these two encryptions.

I expect that this thesis will have a significant impact on future works for using XBee and Arduino UNO as an alternative to the other factory made motes in Wireless Sensor networks. Also, this work will give an incentive to other researchers to use open source and financially viable options.

6.2 Conclusion

With modern Wireless Sensor Network (WSNs), security is becoming an important issue. I have discussed about many of the attacks that can occur in WSNs and their impact also on a large scale. Now, human health is involved in attacks and one such attack can lead to devastating damage to a human and their safety. The biggest threats of WSNs is that an adversary can capture the nodes, destroy the network and force it to send wrong information. So, ensuring the security for a WSNs is very crucial for the well being of the network and eventually for the well being of the people who are connected with it. While encryption ensures the data is secured, authentication ensures that the two parties who are communicating with each other are valid. But for this very reason, only encryption in a network is not enough. As an example, If only encryption is introduced in a network, the problem it will encounter is that the encrypted data are no longer safe or data has been compromised during receiving because the sender did not verify the identity of the reciever. That is why I put equal attention to both authentication and encryption. Maximum result can only be achieved if we use both of them in a network at the same time. I propose an authentication with key exchange mechanism, implemented it in hardware and make the network more secure. Very few practical implementations exist in Wireless Sensor Networks, with most of the experiments being done by using simulation.

6.3 Future works

I show an authentication concept using the encryption in WSNs and also provide a key exchange and changing capability in real time between the nodes. I will try for more improvements in the logic of the authentication in future. Additional attributes of XBee and Arduino UNO in the network can be utilized for lower battery use during a node's idle state.

Much work has been done towards data availability in Internet too. The implemented GUI already has a button to save the sensor data for all the nodes in a separate file. This file can be uploaded to the Internet for monitoring. The Internet availability concept deals with the FTP and a remote server which are not readily available and thus it can be implemented fully in the future as well.

Bibliography

[1] Waltenegus Dargie and Christian Poellabauer. *Fundamentals of wireless sensor networks: theory and practice*. John Wiley & Sons, 2010.

[2] Joseph M Kahn, Randy H Katz, and Kristofer SJ Pister. “Next century challenges: mobile networking for ’Smart Dust’”. In: *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*. ACM. Seattle, Washington, USA, 1999, pp. 271–278.

[3] Gregory J Pottie. “Wireless integrated network sensors (WINS): the web gets physical”. In: *Frontiers of Engineering:: Reports on Leading-Edge Engineering from the 2001 NAE Symposium on Frontiers of Engineering*. National Academies Press. 500 Fifth St. N.W., Washington, D.C. 20001, 2002, p. 78.

[4] J Rabaey et al. “PicoRadio: Ad-hoc wireless networking of ubiquitous low-energy sensor/monitor nodes”. In: *VLSI, IEEE Computer Society Workshop on*. IEEE Computer Society. Orlando, FL, USA, 2000, pp. 9–12.

[5] *Arduino UNO Description*. `http://www.arduino.cc`. Accessed: 2014-07-30.

[6] *XBee Description*. `http://www.digi.com/xbee/`. Accessed: 2014-07-26.

[7] Kiran Maraiya, Kamal Kant, and Nitin Gupta. “Application based study on wireless sensor network”. In: *International Journal of Computer Applications (0975–8887) Volume* 21 (2011), pp. 9–15.

[8] Shamneesh Sharma, Dinesh Kumar, and Keshav Kishore. “Wireless Sensor Networks-A Review on Topologies and Node Architecture”. In: *International Journal of Com-*

puter Sciencessss and Engineeringand Engineeringand Engineeringand Engineering (Oct. 2013), pp. 19–25.

[9] Ahmad Abed Alhameed Alkhatib and Gurvinder Singh Baicher. “Wireless sensor network architecture”. In: *International conference on computer networks and communication systems (CNCS 2012) IPCSIT*. Vol. 35. IACSIT Press, Singapore, 2012, pp. 11–15.

[10] Saurabh Singh and Harsh Kumar Verma. “Security for Wireless Sensor Network”. In: *International Journal on Computer Science and Engineering* 3 (2011), pp. 2393–2399.

[11] Jianliang Zheng and Myung J Lee. “A comprehensive performance study of IEEE 802.15.4”. In: *Sensor Network Operations*. USA: IEEE Press Book Los Alamitos, 2004, pp. 218–237.

[12] Gianluigi Ferrari et al. “Wireless sensor networks: performance analysis in indoor scenarios”. In: *EURASIP Journal on Wireless Communications and Networking* 2007 (2007), 14 pages.

[13] Goran Horvat, D Sostaric, and D Zagar. “User authorization system using ZigBee WSN and AVR architecture”. In: *Telecommunications Forum (TELFOR), 2011 19th*. IEEE. Belgrade, 2011, pp. 381–384.

[14] Roszainiza Rosli, Yusnani Mohd Yusoff, and Habibah Hashim. “Performance analysis of ID-based authentication On Zigbee transceiver”. In: *Wireless Technology and Applications (ISWTA), 2012 IEEE Symposium on*. IEEE. Bandung, 2012, pp. 187–191.

[15] V Bhoopathy and RMS Parvathi. “Secure Authentication Technique for Data Aggregation in Wireless Sensor Networks.” In: *Journal of Computer Science* 8 (2012), pp. 232–238.

[16] Sung Jin Choi and Hee Yong Youn. “Mkps: A multi-level key pre-distribution scheme for secure wireless sensor networks”. In: *Human-Computer Interaction. Interaction Platforms and Techniques*. Beijing, China: Springer, 2007, pp. 808–817.

[17] Ying Qiu et al. “Authentication and key establishment in dynamic wireless sensor networks”. In: *Sensors* 10 (2010), pp. 3718–3731.

[18] Rajeev Piyare and Seong-ro Lee. “Performance Analysis of XBee ZB Module Based Wireless Sensor Networks”. In: *International Journal of Scientific & Engineering Research* 4 (2013), pp. 1615–1621.

[19] Keshtgari Manijeh and Deljoo Amene. “A wireless sensor network solution for precision agriculture based on zigbee technology”. In: *Wireless Sensor Network* 2012 (Jan. 2011), pp. 25–30.

[20] Nidhi Singhal and JPS Raina. “Comparative analysis of AES and RC4 algorithms for better utilization”. In: *International Journal of Computer Trends and Technology* 79 (July 2011), pp. 177–181.

[21] Vongsagon Boonsawat et al. “XBee wireless sensor networks for temperature monitoring”. In: *the Second Conference on Application Research and Development (ECTI-CARD 2010), Chon Buri, Thailand*. Pattaya, Chonburi, Thailand, Oct. 2010.

[22] Xiaohua Luo et al. “Encryption algorithms comparisons for wireless networked sensors”. In: *Systems, Man and Cybernetics, 2004 IEEE International Conference on*. Vol. 2. IEEE. Oct. 2004, pp. 1142–1146.

[23] Ashok Kumar Das and Debasis Giri. “An identity based key management scheme in wireless sensor networks”. In: *arXiv preprint arXiv:1103.4676* (24 March 2011).

[24] Gaurav Jolly et al. “A low-energy key management protocol for wireless sensor networks”. In: *Computers and Communication, 2003.(ISCC 2003). Proceedings. Eighth IEEE International Symposium on*. IEEE. 30 June-3 July 2003, pp. 335–340.

[25] *Arduino to Arduiono Communication using wire*. http://arduino.cc/en/Tutorial/MasterWriter. Accessed: 2014-08-23.

[26] *RS-232 vs. TTL Serial Communication*. https://www.sparkfun.com/tutorials/215. Accessed: 2014-07-14.

[27] *8 bit microcontroller with 4/8/16/32K Bytes of In-System Self-Programmable Flash Description*. https://www.sparkfun.com/datasheets/Components/SMD/ATMega328.pdf. Accessed: 2014-08-14.

[28] Digi International. *Demystifying 802.15.4 and ZigBee*. Tech. rep. 2008.

[29] *Xbee Explorer Dongle*. https://www.sparkfun.com/products/11697. Accessed: 2014-08-04.

[30] *Temperature sensor*. http://www.farnell.com/datasheets/85387.pdf. Accessed: 2014-07-23.

[31] *Humidity sensor*. https://www.sparkfun.com/datasheets/Sensors/Weather/SEN-09569-HIH-4030-datasheet.pdf. Accessed: 2014-07-23.

[32] *TelosB WSN Node*. http://www.willow.co.uk/TelosB_Datasheet.pdf. Accessed: 2014-07-28.

[33] *Micaz WSN Node*. http://www.openautomation.net/uploadsproductos/micaz_datasheet.pdf. Accessed: 2014-07-28.

[34] *Iris WSN Node*. http://www.memsic.com/userfiles/files/datasheets/wsn/iris_datasheet.pdf. Accessed: 2014-07-28.

[35] *Libilium Waspmote WSN Node*. http://www.libelium.com/v11-files/documentation/waspmote/waspmote-datasheet_eng.pdf. Accessed: 2014-07-28.

[36] *X-CTU Configuration and test*. http://ftp1.digi.com/support/documentation/90001003_A.pdf. Accessed: 2014-08-08.

[37] *Docklight Software*. http://www.docklight.de/pdf/docklight_manual.pdf. Accessed: 2014-08-04.

[38] William Stallings. "Cryptography and network security, principles and practices, 2003". In: *Practice Hall* ().

[39] *Arduino Softwareserial library*. http://arduino.cc/en/Reference/SoftwareSerial. Accessed: 2014-08-23.

[40] Veris. *Veris Aerospond Wireless Sensors: Received Signal Strength Indicator (RSSI)*. Tech. rep.

[41] *XBee - Multiple Node Discovery and RSSI*. http://tymkrs.tumblr.com/post/18193549622/xbee-multiple-node-discovery-and-rssi. Accessed: 2014-07-23.

[42] *AES code used for performance checking:* https://github.com/DavyLandman/AESLib. Accessed: 2014-08-12.

[43] *Xbee Product manual*. https://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Datasheet.pdf. Accessed: 2014-08-12.