DEVELOPMENT OF AN IMPROVED INTRUSION DETECTION BASED SECURED ROBUST HEADER COMPRESSION TECHNIQUE

By

Malachi Chidiebere EGBUGHA

Department of Electrical and Computer Engineering
Faculty of Engineering
Ahmadu Bello University Zaria, Nigeria

MAY, 2017

DEVELOPMENT OF AN IMPROVED INTRUSION DETECTION BASED SECURED ROBUST HEADER COMPRESSION TECHNIQUE

By

Malachi Chidiebere EGBUGHA, B.Sc. (UNICAL, 2012)

P14EGCP8009

A DISSERTATION SUBMITTED TO THE SCHOOL OF POSTGRADUATE STUDIES, AHMADU BELLO UNIVERSITY, ZARIA

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF A MASTER OF SCIENCE (M.Sc) DEGREE IN COMPUTER ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

FACULTY OF ENGINEERING

AHMADU BELLO UNIVERSITY, ZARIA

NIGERIA

MAY, 2017

DECLARATION

I,Malachi Chidiebere EGBUGHA, hereby declare that the work in this dissertation entitled "Development of an Improved Intrusion Detection Based Secured Robust Header Compression Technique"has been carried out by me in the Department of Electrical and Computer Engineering. The information derived from literature has been duly acknowledged in the text and a list of references provided. No part of this dissertation was previously presented for another degree or diploma at this or any other institution.

Malachi Chidiebere Egbugha_____ _____

Signature Date

CERTIFICATION

This Dissertation titled **"**DEVELOPMENT OF AN IMPROVED INTRUSION DETECTION BASED SECURED ROBUST HEADER COMPRESSION TECHNIQUE**"** by Malachi Chidiebere Egbugha meets the regulations governing the award of degree of Master of Science (MSc) in Computer Engineering of the Ahmadu Bello University, and is approved for its contribution to knowledge and literary presentation.

Dr. I. J. Umoh Chairman, Supervisory Committee	_____ Signature	_____ Date
Dr. A.M.S. Tekanyi Member, Supervisory Committee	_____ Signature	_____ Date
Dr. Y. Jibril Head of Department	_____ Signature	_____ Date
Professor S. Z Abubakar Dean Postgraduate School	_____ Signature	_____ Date

DEDICATION

This research work is dedicated to my Parents and Siblings.

ACKNOWLEDGEMENT

I wish to express my profound gratitude to Almighty God, The most gracious and merciful Lord. Despite the numerous efforts put into this research work, it would not have been possible without the help of Almighty God, the fountain of all wisdom, knowledge and understanding. I am highly indebted to you.

My deepest gratitude goes to my supervisor Dr. I. J. Umoh, for his efforts and constant supervision towards the success of this work. The completion of this work could not have been possible without your constant participation and assistance. Thank you very much sir. I am indebted and grateful to my co-supervisor Dr A.M.S. Tekanyi for his tireless efforts, thorough scrutiny, patience and high level of mentoring that helped me in the completion of this research. Thank you very much sir. I sincerely appreciate the effort, support and encouragement given to me by Prof. M. B. Mu'azu. Thank you very much Prof. My deep appreciation goes to the Computer & Control Research Group for their keen interest, suggestion and constructive criticisms during all those stages of my presentations.

I acknowledge with thanks all the lecturers of Electrical and Computer Engineering, Ahmadu Bello University, namely: Prof. B. G. Bajoga, Prof. B. Jimoh, Dr. A. M. S. Tekanyi, Dr. S. M. Sani, Dr. A. D. Usman and Dr. S. Garba, Dr. Y. Jibril, Dr. T.H Sikiru, Dr. K.A. Abu-Bilal, Engr.F. Sadiku, Engr. M. J. Mu'azu, Engr. A. I. Abdullahi, Engr. E. A. Gbenga, Engr. Bashir Sadiq, Engr. Salawudeen A. Tijjani, Engr. Olaniyan Abdulrahman and most especially, those whose names could not be mentioned. My deepest appreciation also goes to Abdullahi Tukur for all the administrative support towards me and the entire students of Department of Electrical and Computer Engineering. I also express my profound appreciation to Dr. E. A. Adedokun for his support and attention in the course of this research.

I am also thankful to my special friends Salawudeen A. Tijjani, Ndubisi .N,Mr Braimo, Nwanne Josphene, Mrs Husseinah Abubakar, Atuman G. Joel, Suleiman Hussein, Ocholi H., Ibrahim Abdulwahab, Magaji Suleiman, Daddy Jessi, Ore-ofe Ajayi, Kelechi Nwogu , Uju Asouzu, Oga Yemi Abalaba, Mentor,Oga Caleb,Five huns, Oga Abubakar, Oyinbo P., Oga Ovie, Muktar Abubakar and Valentine Ikpo. Their continuous support and contributions toward the success of this work would never be forgotten, may God reward them and strengthen our friendship. I am very much thankful to all my course mates, Monday, Mr Adewale, Mr Aliyu, Mr Alhassan, Balogun, and Mr Adamu, may God reward you all.

Finally, I am everly indebted to my parents Mr & Mrs Emmanuel Egbugha, your endless love ,support, kind and understanding personality will always be appreciated. Thank you very much. To my siblings Engr. Anselem Egbugha, Caleb Egbugha, Gerald Egbugha and Grace Egbugha. I owe you all a big thank you. To my Aunty Mrs Rose and her family, I am highly grateful for your constant support. A big thank to Pastor Kayode Olukayode and all those that have directly and indirectly contributed to the success of this work. May God bless.

Malachi Chidiebere EGBUGHA

May, 2017.

ABSTRACT

This research presents thedevelopment of an improved Intrusion Detection based Secured RObust Header Compression (IDSROHC)techniquefor handling brute force attack. The secured RObust Header Compression (secured ROHC) was developed to secure Internet Protocol version six (IPv6) packets against false initial refresh attack by encrypting the cyclic redundancy check field. However, the secured ROHC exposes the network to brute force attack due to the short field length of the cyclic redundancy check .An improved IDSROHC technique was developed using a modified selective watchdog intrusion detection algorithm. Weighted network graph and random way point model with pause time greater than simulation time was used to model the distribution of total of forty nodes. The nodes were randomly distributed within an area of 100 by 100 meters. Bellman-Ford algorithm was used to determine the shortest path of packets transmission. Poisson traffic model was used to model the payload size and processing time of packets. The interpretation interval, packet loss and average compression length of packets were calculated using the header compression model. Cyclic redundancy check field of packets was modelled and encrypted using symmetric block cipher. Brute force attack was modeled using a pseudorandom generator. A MATLAB graphical user interface was designed to aid presentation. An intrusion detection system based on selective watchdog technique was used to inspect trial packet with the aim of identifying malicious link. IDSROHC was validated via comparison with the Secured ROHC using throughput and packet delivery success. The results of this work showed that IDSROHC produced 4.97% improvement in throughput and 29% improvement in packet delivery success over secured ROHC.

TABLE OF CONTENTS

DECLARATIONI

CERTIFICATIONII

DEDICATIONIII

ACKNOWLEDGEMENTIIV

ABSTRACTVI

LIST OF FIGURESXII

LIST OF TABLESXIV

LIST OF ABBREVIATIONSXV

CHAPTER ONE:INTRODUCTION

1.1 BACKGROUND1

1.2 SIGNIFICANCE OF RESEARCH3

1.3 STATEMENT OF PROBLEM3

1.4 AIM AND OBJECTIVES4

1.5 METHODOLOGY4

1.6 DISSERTATION ORGANIZATION5

CHAPTER TWO:LITERATURE REVIEW

2.1 INTRODUCTION6

2.2 REVIEW OF FUNDAMENTAL CONCEPTS6

2.2.1 Internet Protocol (IP) header compression6

2.2.2 Shared-State internet protocol header compression8

2.2.3 Stateful internet protocol header compression8

2.2.4 Packet types13

2.2.5 Random waypoint mobility model14

2.2.6 ROHC model15

2.2.7 Average compression length model16

2.2.8 Packet loss and transmitted byte model17

2.2.9 Bellman-ford algorithm19

2.2.10 Packet delay model20

2.2.11 Poisson traffic model21

2.2.12 ROHC attack22

2.2.13 Intrusion Detection System (IDS)26

2.2.14 Categories of IntrusionDetection Systems27

2.2.15 Selective watchdog and watchdog intrusion detection system28

2.2.17 Internet protocol routing31

2.2.18 CRC33

2.2.19 Block cipher cryptographic system33

2.2.21 Performance metrics35

2.3 REVIEW OF SIMILAR WORKS36

CHAPTER THREE:MATERIALS AND METHOD

`3.1 INTRODUCTION42

3.2 ORIGINAL SECURE ROBUST HEADER COMPRESSION TECHNIQUE43

3.2.1 Node distribution using weighted network graph and random waypoint model45

3.2.2 Obtaining shortest-path for packet routing47

3.2.3 Calculating the required interpretation interval49

3.2.4 Obtaining the average compression length49

3.2.5 Generating the Cyclic Redundancy Check (CRC)50

3.3.6 Encrypting the Cyclic Redundancy Check (CRC)50

3.2.7 Key expansion model51

3.2.8 Calculating packet delay53

3.2.9 Calculating the payload size53

3.2.10 Brute force attack model53

3.2.11 Calculating burst loss and total transmitted byte54

3.3 DEVELOPMENT OF AN IMPROVED INTRUSION DETECTION BASED SECURED ROBUST HEADER COMPRESSION TECHNIQUE.54

3.3.1 Development of MATLAB GUI for improved intrusion detection based secured robust header compression.57

3.4 PERFORMANCE EVALUATION AND SIMULATION PARAMETERS58

CHAPTER FOUR: RESULTS AND DISCUSSIONS

4.1 INTRODUCTION60

4.2 SIMULATION60

4.3 RESULTS OF THE EFFECT OF OUT OF SYNCHRONIZATION PROBABILITY ON REQUIRED INTERPRETATION INTERVAL60

4.4 RESULTS OF THE EFFECT OF AVERAGE ERROR PROBABILITY ON INTERPRETATION INTERVAL62

4.5 RESULT OF RELATIONSHIP BETWEEN OUT OF SYNCHRONIZATION PROBABILITY AND THE INITIAL REFRESH TIMEOUT63

4.6 RESULTS OF THROUGHPUT FOR ROHC AND UNCOMPRESSED TRAFFIC64

4.7 RESULTS OF IMPROVED INTRUSION DETECTION BASED SECURED ROBUST HEADER COMPRESSION (IDSROHC)65

4.8 COMPARISON OF IDSROHC AND SECURED ROHC PERFORMANCES USING PACKET DELIVERY SUCCESS AND THROUGHPUT.68

CHAPTER FIVE:CONCLUSION AND RECOMMENDATION

5.1 SUMMARY72

5.2 CONCLUSION72

5.3 SIGNIFICANT CONTRIBUTIONS73

5.4 LIMITATIONS73

5.5 RECOMMENDATIONS FOR FURTHER WORK73

REFERENCES74

APPENDIX A

COMPLETE MATLAB M FILE FOR SECURED ROBUST HEADER COMPRESSION 78

APPENDIX B

MATLAB M FILE FOR NODES CREATION AND DISTRIBUTION USING WEIGHTED NETWORK GRAPH AND RANDOM WAYPOINT MODEL90

APPENDIX C

MATLAB M FILE FOR FINDING SHORTEST PATH USING BELLMAN-FORD ALGORITHM91

APPENDIX D

MATLAB M FILE FOR INTERPRETATION INTERVAL AND AVERAGE COMPRESSION LENGTH MODEL92

APPENDIX E

MATLAB M FILE FOR GENERATING OF CYCLIC REDUNDANCY CHECK93

APPENDIX F

MATLAB M FILE FOR IR PACKET PACKET94

APPENDIX

MATLAB M FILE FOR CRC ENCRYPTION95

APPENDIX H

MATLAB M FILE FOR KEY EXPANSION MODEL99

APPENDIX J

MATLAB M FILE FOR GUI101

LIST OF FIGURES

Figure 2.1: IR Packet Type 14

Figure 2.2: Compressed Packet Type 14

Figure 2.3: Network showing its nodes and flow 20

Figure 2.4: ROHC False IR Attack 23

Figure 2.5: ROHC False ACK Attack 24

Figure 2.6: ROHC False NACK Attack 24

Figure 2.7: Flowchart of brute force attack 25

Figure 2.8: Intrusion Detection Framework Architecture 27

Figure 2.9: Watchdog Operation (Kang et al., 2011)28

Figure 2.10 Selective Watchdog Technique 30

Figure 2.11: IPv6 packet 31

Figure 2.12: IP Routing Algorithm 32

Figure 3.1: Flowchart for implementation of Secured RObust Header Compression44

Figure 3.2: Snippet for Node Creation and Distribution45

Figure 3.3: Node Creation and Distribution46

Figure 3.4: Connection Matrix47

Figure 3.5: Snippet for finding the Shortest Path using Bellman-Ford algorithm48

Figure 3.6: MATLAB implementation of Bellman ford algorithm48

Figure 3.7: Snippet for calculating the interpretation interval49

Figure 3.8: Snippet for obtaining average compression length49

Figure 3.9: Snippet for Generating Cyclic Redundancy Check 50

Figure 3.10 CRC Encryption Flowchart51

Figure 3.11: Snippet of CRC Encryption51

Figure 3.12 Key Expansion Algorithm 52

Figure 3.13:Snippet for Key Expansion Model53

Figure 3.14:Snippet for Brute Force Attack54

Figure 3.15: Flowchart for implementation of the Improved intrusion detection based secured Robust header compression56

Figure 3.16 Developed Simulator for Improved Intrusion Detection based Secured Robust Header Compression.57

Figure 3.17: Snippet of GUI implementation58

Figure 4.1: Interpretation Interval against out of Synchronization Probability61

Figure 4.2: Interpretation Interval against Out of Synchronization62

Figure 4.3: Out of synchronization probability against initial refresh time out63

Figure 4.4: Throughput of ROHC and Uncompressed traffic64

Figure 4.5 Effect of brute-force attack on packet delivery success for IDSROHC model65

Figure 4.6 Effect of brute-force attack on the network throughput67

Figure 4.7: Comparing the result in terms of throughput between Secures ROHC and IDSROHC69

Figure 4.10: Comparing packet delivery success of Secured ROHC and IDSROHC 70

LIST OF TABLES

Table 2.1: Relationship between II, encoded bits and compressed header size 17

Table 3.1 Simulation parameters59

Table 4.1: Performance Evaluation of IDSROHC 71

LIST OF ABBREVIATIONS

Acronyms	Definition
ABIDS	Anomaly-Based Intrusion Detection Systems
BER	Bit Error Rate
CID	Context Identifier
CR	Context Replication
CRC	Cyclic Redundancy Check
False ACK	False ACKnowledgment
False IR	False Initialization/Refresh,
False NACK	False Negative ACKnowledgment
FO	First Order
GUI	Graphical User Interface
IDS	intrusion-detection system
IDSROHC	Improved Intrusion Detection based Secured robust header compression
II	Interpretation Interval
IPHC	Internet Protocol Header Compression
IPsec	Internet Protocol security
IPv4	Internet Protocol version four
IPv6	Internet Protocol version six

IR	Initialization and Refresh
MIPHC	Mobile adhoc network Internet Protocol Header Compression
OoS	Out of Synchronization Probability
ROHC	RObust Header Compression
RTP	Real-time Transport Protocol
RTT	Round Trip Time
SBSS	Signature-Based Intrusion Detection System
Secured ROHC	Secured RObust Header Compression
SO	Second Order
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VOIP	Voice Over Internet Protocol
VJHC	Van Jacobson Header Compression
W-LSB	Window-based Least Signicant Bits

CHAPTER ONE

INTRODUCTION

1.1 Background

The desire for industries to move towards an Internet Protocol Version six (IPv6) network architecture have pushed research in the direction of maximizing bandwidth. This is due to increased header size of IPv6 header as compared to the payload. Therefore, reducing the internet protocol header overload sent over the air becomes inevitable (Cheng & Moore, 2013)**.**The relative compression gain for specific flows (or applications) depends on the size of the payload used in each packet. For applications such as Voice Over Internet Protocol (VOIP), the size of the payload containing coded speech can be as small as 15-20 octets while the transport header will have 20 octets Hence,this gives atotal size of 60 octets for IPv6 header (Sandlund *et al.,* 2010).Therefore, header compression leads to quite significant compression gain. One method of providing increased bandwidth efficiency is the use of IP header compression techniques.Header compression provides more efficient use of bandwidth in a packet switched network by taking advantage of header field redundancies in packets belonging to the same packet flow(Majanen *et al.,* 2015).It involves a compressor and a decompressor operating according to a well-defined protocol. The compressor compresses the headers with respect to a reference state that it shares in common with the decompressor, while the decompressor uncompresses them to their original state on reception at the destination (Chishti & Mir, 2015).

Header compression techniques fall into two major categories: stateful header compression and stateless header compression technique.The stateful header compression technique builds hop-by-hop compression per flow. These include Van Jacobson Header Compression (VJHC), RObust Header Compression (ROHC) scheme and Internet Protocol Header Compression

(IPHC) .Stateless header compressions such as Mobile adhoc network Internet Protocol HeaderCompression (MIPHC)do not require state management. For a mobile adhoc network, a stateless header compression such as MIPHC is used to compress the layer three header while stateful header compression such as ROHC is use to compress the layer four header(Bow-Nan *et al.*, 2013).

The ROHC protocols provide an efficient and robust header compression. It is designed to operate efficiently and robustly over various link technologies with different characteristics(Sandlund *et al.*, 2010).While this exchange leads to efficient bandwidth utilization, there are several potential attack such as False Initialization/Refresh (False IR), False ACKnowledgment (False ACK) and False Negative ACKnowledgment (False NACK) attack that can lead to denial of service (the inability to decompress) (Cheng & Moore, 2013).

In other to solve this problem, research has focused on cryptographic method such as Internet Protocol security (IPsec) packet encryption after compression and secured Robust Header Compression (secured ROHC) by encrypting the Cyclic Redundancy Check (CRC) encryption(Cheng & Moore, 2013).Although the IPsec packet encryption after compression makes ROHC invisible to potential attackers, it is not a practical method to secure the ROHC due to the significant overhead in IPsec tunnelling. The CRC is only 3-8 bits long, which implies that there are no more than 256 possible combinations. Thus, a malicious node could still attempt a brute force approach, where it sends fake packets with all possible CRC combinations(Cheng & Moore, 2013).The traditional way of protecting wired/wireless networks encryption software is therefore no longer sufficient(Esfandi, 2010).Therefore there is a need for other non-cryptographic technique such as intrusion detection system.

There are two main approaches for security management, which are prevention-based and detection-based(Can & Sahingoz, 2015).Prevention-based security management (defense against

attack) aims to prevent any attack before it happens. Any proposed technique has to defend against the targeted attack(Butun *et al.*, 2014).Indetection-based (being aware of the attack that is present) security management, if an attacker manages to pass the measures taken by the prevention step, it means that there is a failure to defend against the attack. At this time, the security solution would immediately switch into the detection phase of the attack in progress and specifically identify the nodes that are being compromised(Butun *et al.*, 2014). In any security plan, if intrusion prevention (encryption, authorization, and authentication) named as the first line of security is defeated by attackers, then a second line of defence, intrusion detection comes into prominence(Can & Sahingoz, 2015).

Intrusion detection provides deterrence for an intruder and serves as an alarm mechanism for a computer system or a network to manage security plan successfully. An intrusion-detection system (IDS) can be defined as a software or hardware monitoring tool that detects internal or external cyber-attacks. An ID can observe and investigate system and user activities, recognize patterns of known attacks and identify abnormal network activity. An IDSROHC technique developed using a modified selective watchdog technique was therefore employed in this research to detect and mitigate brute force attack.

1.2 Significance of Research

The significance of this research is the development of a graphical user interface (GUI) based IDSROHC technique using a selective watchdog intrusion detection system, which has added capabilities to the standard secured ROHC through detecting and mitigating brute force attack. This has not been considered by the previous researchers.

1.3 Statement of Problem

ROHC is a standardized method to compress the IP, UDP, UDP-Lite, RTP, and TCP headers of Internet packets.While this compression technique leads to an improve utilization of network

bandwidth and reduction of end-to-end delay, there are several potential attack vectors such as False IR, False Ack, and False Nack that can lead to denial of service (inability to decompress) and man-in-the middle attack.In other to solve this problem previous researchers focused on encryption of CRC. However, the CRC is only 3-8 bits long, which implies that there are not more than 256 possible combinations. This means that a malicious node could still attempt a brute force approach where it sends fake packets with all possible CRC combinations. Hence, the developed improved intrusion detection secure robust header compression technique based on modifying selective watchdog technique is implemented to address this limitation.

1.4Aim and Objectives

The aim of this research work is to develop an improved intrusion detection based secured robust header compression technique.

The objectives of the research are as follows:

1. Replication and implementation of the secured robust header compression technique.
2. Develop animproved Intrusion Detection based SecureRObust Header Compressiontechnique (IDSROHC) using modified selective watchdog intrusion detection algorithm .
3. Compare the performance of the IDSROHC and secured ROHC using throughput and packet delivery success as performance metric.

1.5Methodology

The methodologies adopted for this research towards development of an improved intrusion detection based secured robust header compression technique are as follows:

1. Replication and implementation of the secured ROHCtechnique. To replicate and implement the existing secured ROHC the following steps were carried out:
    i. Node distribution using weighted network graph and Random Waypoint Model.

ii. Obtain shortest path from source to destination using the Bellman-ford Algorithm.

iii. Determine interpretation interval ROHC Model.

iv. Calculate average compression length using average compression length model

v. Encrypt packet using symmetric block cipher

vi. Calculate remainder polynomial using Cyclic Redundancy Check (CRC) Model

vii. Determine packet size and processing time using Poisson Traffic Model

viii. Apply Brute force attack Model

ix. Calculate Burst loss and total transmitted byte

2. Development of an improved graphical user interface IDSROHC technique.

   i. Step (i-vi) above will be repeated.

   ii. Apply Watchdog intrusiondetection system

   iii. Apply Brute force attack Model

   iv. Calculate Burst loss and total transmitted byte

   v. Develop MATLAB Graphical User Interface (GUI) .

3. Comparison of the performance of the IDSROHC and secured ROHC using throughput and packet delivery success.

1.6 Dissertation Organization

Chapter one presents the general introduction of this work. The rest of the chapters are structured as follows: Chapter two details reviews of related literature and relevant fundamental concepts about ROHC, Brute force attack, cryptography and intrusion detection system .Chapter three present mathematical model and algorithms relevant to the development of a secured ROHC and IDSROHC technique. Chapter four focuses on the analysis, performance and discussion of results. Finally, conclusion and recommendations for further work were discussed in chapter

five. The list of cited references and MATLAB codes in the appendices are provided at the end of this report.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

The chapter comprises of the overview of fundamental concept and the review of similar works. The review of fundamental concept presents theories and works that lay foundation for the success of this research while review of similar works present existing work related to the research.

2.2 Review of Fundamental Concepts

In this section, concepts fundamental to this research work such as review of existing Internet Protocol (IP) header compression andshared-state IP header compressionare reviewed, also works similar to this research are reviewed. The review also include techniques, methods and model equations considered in other research works

2.2.1 Internet Protocol (IP) header compression

Packet compression can be achieved by either header or payload compression, or the combination of both(Sun *et al.*, 2011). Packet compression allows much smaller amounts of packet drops, more simultaneous sessions, and a smooth and fast behavior of applications(Matias & Refua, 2005). The overhead portion, especially for a small packet couple with the desire for industry to move to an IPv6 (or dual stack IPv4/IPv6) have pushed research towards maximizing bandwidth efficiency (Jivorasetkul *et al.*, 2012).There are 20 bytes of IPv4 header, 40 bytes ofIPv6 header, 12 bytes of User Datagram Protocol(UDP) header and 8 bytes of Real-time Transport Protocol (RTP)header. These make a total of 40 bytes to 60 bytes. However, the effective payload of data packets in real-time multimedia services is usually 15-20 bytes.

Therefore, only 27% -33% of the bandwidth is used to transmit the actually useful data, and a lot of bandwidth is used to transmit the protocol headers(Chen *et al.*, 2012). Studies performed in 2008 by the Cooperative Association for Internet Data Analysis (CAIDA) have shown that the potential bandwidth savings per packet for IPv6packets on the core networks is 10–30 percent(Cheng *et al.*, 2013).Therefore, it is essential to find a way to improve the network bandwidth utilization. One method of increasing bandwidth efficiency is the use of IP header compression (Cheng *et al.*, 2013; Jivorasetkul *et al.*, 2013; Tomoskozi *et al.*, 2015).

IP header compression is the process of reducing protocol header overhead in order to improve bandwidth efficiency while maintaining the end-to-end transparency (Rawat & Bonnin, 2010).IP header compression concept relies on the characteristic that many header fields in consecutive packets belonging to the same packet flow remain a constant or change in predictable manner (Jivorasetkul *et al.*, 2012). A lot of header fields do not change their value or change according to certain rules during a communication session. Therefore,it is useful to separate the header fields according to their classification as static, dynamic, static known and static inferred(Majanen *et al.*, 2015). The header fields of an IP packet is classified in to the following:

(1)**Inferable fields**: These are fields that can be inferred by other header fields, such as the length field. The inferable field is never sent in header compression schemes.

(2**) Static fields:** The value of these fields does not change during the entire transmission of the data stream, such as the version field in IP headers. Static fields are just sent when a data stream is established.

(3)**Statically defined fields:** These fields are generally used to define the properties of data streams and to distinguish them, such as the address fields. Statically defined fields are also sent once when a data stream is established.

(4) **Changed fields:** These fields change between the packets in a data stream, in accordance with certain rules or randomly. It is necessary to compress the changed fields by using some of the compression algorithms(Chen *et al.*, 2012).

An IP header compression protocol comes in two broad categories. They are stateful IP headercompression and shared-state (often called stateless) IP header compression.

2.2.2 Shared-State internet protocol header compression

Shared-state IP header compression maintains preconfigured mappings on every node and requires little or no coordination(Cheng *et al.*, 2013). An example of a shared-state internet IP compression is Mobile adhoc network IP Header Compression (MIPHC). (Cheng *et al.*, 2013; Cheng *et al.*, 2014).

2.2.3 Stateful internet protocol header compression

Stateful IP header compression maintains a mapping of header information to a Context IDentifier (CID) for every IP hop and every flow (Cheng *et al.*, 2013; Koskela *et al.*, 2016). Examples of stateful IP header compression include VJHC,IPHC and ROHC(Chen *et al.*, 2012; Sun *et al.*, 2011). However this research focuses on ROHC.

2.2.3.1 ROHC scheme

VJHCand IPHC protocols were the first IP header compression created. The IPHC scheme was created to extend the work done in VJHC.However, it was not robust enough to support links with high bit error rates, high losses, and long round trip times. High Bit Error Rate (BER) and long Round Trip Time (RTT) are common characteristics of wireless links. Therefore an efficient and robust compression scheme was needed. The ROHC scheme was developed to fulfil these criteria(Madsen *et al.*, 2007). The ROHC scheme uses window based least significant bits encoding for the compression of dynamic fields in the protocol headers. Due to its feedback

mechanism,periodic context refreshes and Window-based Least Signicant Bits (W-LSB) encodingscheme, ROHC is robust on wireless links with high BER and long RTT(AB, 2004). The W-LSB is defined by an interpretation interval. Interpretation interval is the maximum number of packets that is lost in a row without losing context synchronization (Hermenier *et al.*, 2011). It is expressed mathematically as follows (Li *et al.*, 2013):

$$W = [-o_f, 2^b - 1 - o_f] \quad (2.1)$$

where:

W is the interpretation interval

o_f is the offset

b is the least significant bit

ROHC header compression frame work is a process of interaction between two state machines; a compressor state machine and a decompressor state machine, which are described by their context(Cheng *et al.*, 2013). Both the compressor and the decompressor maintain three interrelated states with slight differences.

1. States of the compressor

ROHC compressor maintains three states, which are the state of Initialization and Refresh (IR), the state of First Order (FO) and the state of Second Order (SO)(Chishti & Mir, 2015). At first, the compressor is at the IR state and then gradually transfers to higher states. As long as the compressor determines that the decompressor has enough information to extract the compressed packets successfully, the compressor tries to maintain the higher state(Venmani *et al.*, 2012).

The compressor will transfer its states mainly due to the following reasons:

i. The change of packet headers

ii. Receiving positive feedback information from the decompressor

iii. Receiving negative feedback information from the decompressor

iv. Rotation of the cycle time (that happens in the one-way mode, when there is no feedback information).

The pseudo code use for compression in robust header compression is as follows (ROHC)(Ang *et al.*, 2008):

```
WHILE there is an outgoing packet THEN
    IF suitable profile is found for the packet THEN
        IF matching context is not found for the packet THEN
            Create a new context
        END IF
        Code CID using the context
        IF ROHC feedback is found THEN
            Piggyback ROHC feed back
        END IF
        Compress packet header using the context and profile
        Concatenate compressed header and payload
    End IF
    Send outgoing packet
END
```

The compressor process is defined as (Cho *et al.*, 2006):

$$C(n_B) = \begin{cases} 1, & S(n_A) \text{ can be compressed} \\ 0, & S(n_A) \text{ cannot be compressed} \end{cases} \quad (2.2)$$

where:

$C(n_B)$ is the compressor process

$S(n_A)$ is the source of flow

n_A number of packets transmitted by the source S to the compressor C

n_B number of packets received by the compressor C from the source S

The interpretation interval $[-O_f, 2^b-1-O_f]$ is defined as V (O_f , b) for a WLSB code (b, O_f). In the absence of packet drops in channel A compression success occurs at the intersection of the following events (Cho *et al.*, 2006):

$$\left[S(\mathrm{n}_A)-\mathrm{S}(\mathrm{n}_A-\mathrm{h})\right]=\sum_{i=n_A-h+1}^{n_A}\in V(\mathrm{o}_{f,}\mathrm{b}),\forall \mathrm{h}=[1,\mathrm{w}] \quad (2.3)$$

The compression success probability is evaluated as:

$$PC(n_B)=P(\sum_{i=n_A\text{-}w+1}^{n_A}\Delta(i)\hat{I}V(o_f,b)) \quad (2.4)$$

where:

$\mathrm{S}(\mathrm{n}_A)$ is the source of flow

$V(\mathrm{o}_{f,}\mathrm{b})$ is the interpretation interval

$\mathrm{o}_{f,}$ is the offset parameter

W is the sliding window

b is the least significant bits of the field to be sent

$PC(n_B)$ is the compression success probability

2. States of the decompressor

The ROHC decompressor maintains three states, which are no context, static context and full context(Chishti & Mir, 2015). At the beginning, the decompressor is in the lowest state (no context state), and then gradually shift to higher states. Once the decompressor enters the highest state of full context, it generally will not leave this state.

Firstly, the decompressor works in the state of no context, when the decompressor does not have packets that have been extracted successfully. Once it successfully extracts the static and dynamic information from a packet, the decompressor enters the state of full context. The compressor remains in the state of full context. However it will to drop to a state of static context when it fails to extract the packets continuously. If the decompressor receives any First Order (FO) packets, it then transfer to the state of full context again. The decompresor will shift down to the state of no context only if it fails to extract the FO packets continuously.(Chen *et al.*, 2012; Venmani *et al.*, 2012).

The pseudocode use for decompression in ROHC is as follows (Ang et al., 2008):

```
WHILE there is incoming packet THEN
    IF padding octets are found THEN
        Strip padding octets
    END IF
    IF Add-CID octet is found THEN
        IF decompressor uses Large_CID THEN
            Discard packet
            CONTINUE
        END IF
    ELSE
        Decode CID
    END IF
    IF decompressor uses Large-CID THEN
        Decode CID from Large-CID octets
    ELSE
        CID is zero
    END IF
        Find context base on the CID
    IF packet type is IR THEN
        IF matching profile not found or CRC is bad THEN
          Discard packet
        CONTINUE
        END IF
        IF packet does not match existing context THEN
          Free old context
          Create a new context
        END IF
          Decode IR packet
    ELSE
        IF context is not found THEN
          Discard packet
          CONTINUE
        END IF
        IF packet type is IR-DYN THEN
            IF CRC is bad THEN
            Discard packet
            CONTINUE
            END IF
        Decode packet
        END IF
    END IF

        Return decompressed packet
END WHILE
```

The decompressor process is defined as follows(Cho *et al.*, 2006):

$$D(\mathrm{n}) = \begin{cases} 1, & S(n_A) \text{ is decopressed successfully} \\ 0, & S(n_A) \text{ is not decompressed successfully} \end{cases} \quad (2.5)$$

where:

$D(\mathrm{n})$ is the decompressor process

$S(n_A)$ is the source of flow

n_A number of packets transmitted by the source S to the compressor C

n_B number of packets received by the compressor C from the source S

2.2.4 Packet types

TheROHC uses three different packet types for communication: the Initialization and Refresh (IR), the Context Replication (IR-CR) and the Compressed (CO).

(a)Initialization and Refresh (IR) packets: The initialization and refresh packet type communicates the static part and the dynamic part of the context. Figure 2.1 shows the structure of an IR packet type:

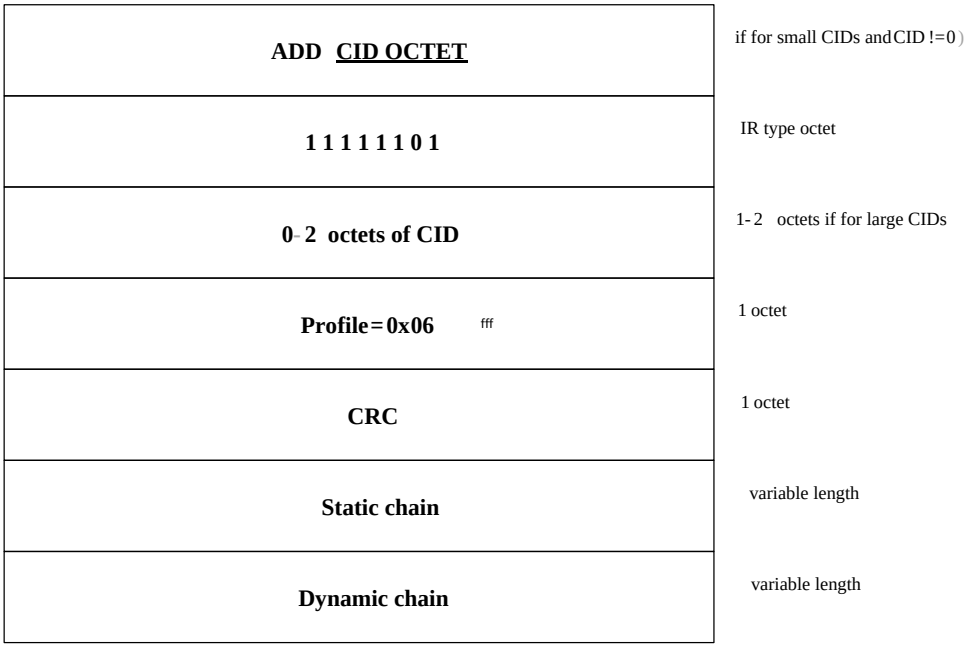


Figure 2.1: IR Packet Type(West, 2013)

(b) Compressed (CO) packets: All CO packets carry a CRC and can update the context. The general format for a compressed type header is shown in Figure 2.2.

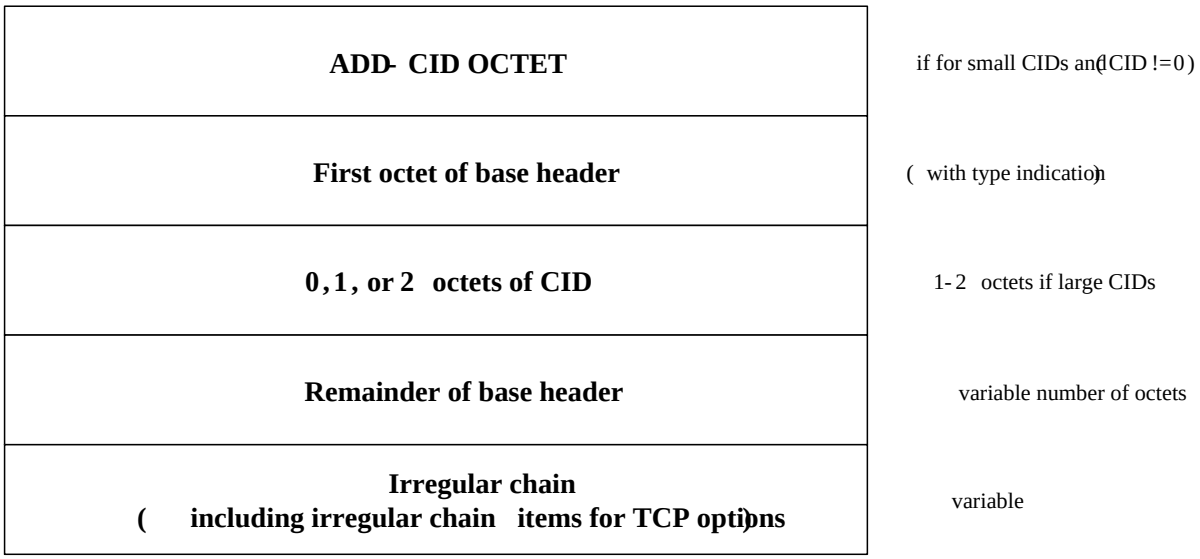


Figure 2.2: Compressed Packet Type (Jonsson & West, 2007)

2.2.5 Random waypoint mobility model

In mobility management, the random waypoint model is a random model for the movement of mobile users, and how their location, velocity and acceleration change over time. Mobility models are used for simulation of users movement pattern.The random waypoint mobility model

was first proposed by Johnson and Maltz. In random waypoint a node randomly chooses a new destination (x, y) (within a given playfield area) and a speed that is uniformly distributed between a minimum and maximum speed [minspeed; maxspeed]. Upon arrival at the destination, it pauses for a specified time (Tpause) period after which it again selects a random destination and speed and continues likewise. In the Random Waypoint model, maxspeed and Tpause are the two key parameters that determine the mobility behavior of nodes. If Tpause=0, this leads to continuous mobility. In addition, Long Tpause and small maxspeed leads to stable topology. In ROHC, the distance between two nodes (Compressor and Decompressor) using random way point model can thus be calculated as follows (Batabyal & Bhaumik, 2015):

$$R = \sqrt{(X_d - X_s)^2 + (Y_d - Y_s)^2} \qquad (2.6)$$

where:

R is the distance between the compressor (source node) and decompressor (destination node)

X_s and Y_s are the Cartesian coordinate of compressor (source node)

X_d and Y_d are the cartesian coordinate of decompressor (destination node).

2.2.6 ROHCModel

One of interesting properties of W-LSB is that decompression will not fail unless the number of consecutive packet losses exceeds the interpretation interval. Hence, interpretation interval plays an important role in the choice of W-LSB and has a direct impact on the performance of ROHC. A small interpretation interval generates fewer bits,has a faster decompression speed and shorter compressed header than a large interpretation interval. However, small interpretational interval

leads to higher compression efficiency and low robustness compared to large interpretational interval. Therefore, for small number of packet losses, a small interpretation interval is sufficient. When large packet loss is expected (i.e., poor bit error rate (BER) channel), large interpretational interval is required. Robust Header Compression model relates the interpretation interval with out of synchronization probability as express in equation 2.7(Hermenier *et al.*, 2013):

$$Po_{oS} = \frac{\in}{L_B}(1-\frac{1}{L_B})^{W} IRT \qquad (2.7)$$

where

PO_{os} is the Out-of-synchronization probability

IRT is the Initial Refresh Time out

W is the Interpretation interval

L_B is the burstlength

$\in$ is the average error probability

2.2.7 Average Compression Length Model

Header compression is possible because there is significant redundancy between header field values within packets, but in particular between consecutive packets belonging to the same flow. ROHC uses correlation of the fields in order to reduce the number of fields transmitted. In the best case, ROHC only needs to send Sequence Number (SN) encoded by W-LSB. It can compress IP headers to be as short as 1 byte. One of interestingproperties of W-LSB is that decompression will not fail unless the number of consecutive packet losses exceeds the interpretation interval(II). Hence, II plays an important role in W-LSB and has a direct impact on the performance of ROHC. A small II generates fewer bits and shorter compressed header than a large II, and has a faster decompression speed, while a small II might cause a higher decompression failure than a large II. Table 2.1 shows the relationship between II, encoded bits and compressed header size of internet protocol version six packets.

Table 2.1: Relationship between II, Encoded Bits and Compressed Header Size(Cha *et al.*, 2015)

II	Encoded Bits	Compressed header size
0	1	1-byte
1-2	2	1 byte
3-6	3	1-byte
7-14	4	1-byte
15-30	5	3 bytes
31-62	6	3 bytes
63-126	7	3 bytes

The average compressed header length (ACL) measures the average length of Internet protocol version six compressed headers. It is the actual header length of internet protocol packets and is dependent on the compressed header length, initial refresh time out, number of update packets and length of update packet. It is expressed mathematical Wang *et al.*, 2012) as:

$$ACL = \frac{(L_1 * (RFI - NUP) + L_2 * NUP)}{RFI} \quad (2.8)$$

where

L_1 is the compressed header length

RFI is the initialization/refresh time out

NUP is the confident variable

L_2 is the length of the update header

2.2.8 Packet Loss and Transmitted Byte Model

ROHC compressor uses a confidence variable *(NUP)* in order to ensure the correct transmission of header information. This means sending the same header packet of each compression level, at

least NUPtimes. The total number of transmitted byte (B_r) in the absence of packet loss is calculated as(Sohraby *et al.*, 2006):

$$B_r = N * \mathrm{NUP(U+P)} + \mathrm{N*(M-}NUP\mathrm{)*(ACL+P)} \quad (2.9)$$

where:

B_r is the transmitted byte

N is the number of flows

NUP is the confident

U is the uncompressed header size

P is the payload size

M is the total number of packets

ACL is the average compressed header length

When header compression is enabled, single packet losses in effect become burst losses of size (Casoni *et al.*, 2017):

$$Loss = \frac{C_{k,k+1}}{b} * RTT \quad (2.10)$$

where:

$Loss$ is the burst packet loss

$C_{k,k+1}$ is the link rate

RTT is the round trip time

b is the packet size in bit

When G number of packets are lost due to out of synchronization, the decompressor would sent a negative acknowledgement packet and return to its previous state. To regain synchronization, the compressor send uncompressed update packets to unable the decompressor repair its context table. Therefore, G packet losses in effect become burst losses of size(Casoni *et al.*, 2017):

$$Loss_L = G*(\frac{C_{k,k+1}}{b} * RTT) \quad (2.11)$$

Therefore the total number of transmitted byte with packet loss $Loss_L$ is calculated as (Sohraby *et al.*, 2006):

$$B_{w/o} = N * \mathrm{NUP}(\mathrm{U}+\mathrm{P}) + \mathrm{N}*(\mathrm{M}-NUP)*(\mathrm{ACL}+\mathrm{P}) + Loss_L*(\mathrm{ACL}+\mathrm{P}) \quad (2.12)$$

where:

N is the number of flows

NUP is the confident variable

U is the uncompressed header size

P is the payload size

ACL is the average compressed header Length

M is the total number of packets

b is the the packet size in bits.

RTT is the average round-trip time

$B_{w/o}$ is the transmitted byte with packet loss

2.2.9 Bellman-Ford Algorithm

The Bellman–Ford algorithm is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph. The algorithm was first proposed by Alfonso Shimbel in 1955, but was instead named after Richard Bellman and Lester Ford, Jr., who published it in 1958 and 1956, respectively. Edward F. Moorealso published the same algorithm in 1957, and for this reason it is sometimes called the Bellman–Ford–Moore algorithm. Bellman-Ford algorithm is a procedure used to find all shortest path in a graph from one source to all other nodes. The algorithm iterate on the number of arcs in a path. At step k of the algorithm, D(i) records the distance from node i to the destination node through the shortest path that consists of at most k arcs. The algorithm is presented as follows(Osunade, 2012):

Step 1, find the adjacent node w of the node i such that

$D(w) + \ell(i,w) = \min_j(D(j) + \ell(i,j))$ where the minimization is performed over all nodes j that are neighbours of node i.

Step 2, Update the distance, $D(j) = D(w) + \ell(j,w)$. Repeat this process until no changes are made. This steps continue to iterate until no changes are made.

2.2.10 Packet Delay Model

Delay is the elapsed time for a packet to travel from the sender through the network to the receiver.

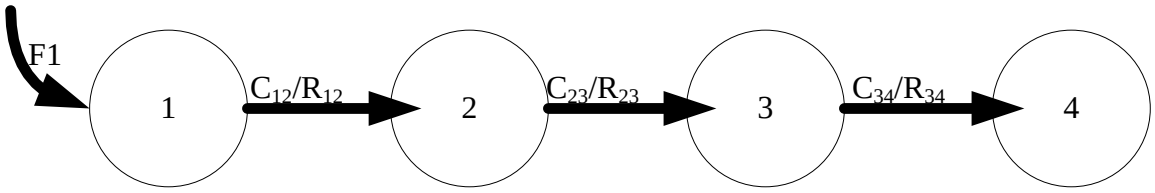


Figure 2.3: Network showing its nodes and flow

Fig. 2.3 illustrates a network with nodes $N = 4$ nodes and flow $f1$. The network is characterized by its transmitting capacity (in bps) Ci,j and its propagation delay Ri,j (in sec.). If there is no link between nodes i and j, then:C$i,j = 0$ and R$i,j = \infty$.

Along its course towards its destination, a packet is potentially delayed by three types of delays:

(i) Processing delays

(ii) Transmission delays

(iii) Propagation delays

The delay experienced by packets of flow $f1$ over the link between nodes k and $k + 1$, referred can be expressed as(Salehin *et al.*, 2013):

$$D_{k,k+1} = P_{k,k+1} + S_{k,k+1}R_{k,k+1} \quad (2.13)$$

The end-to-end delay experienced by packets of flow $f1$ over their $N - 1$ hops from the source to the destination is therefore express as(Salehin *et al.*, 2013):

$$D = \sum_{k=1}^{N-1} S_{k,k+1} + \sum_{k=1}^{N-1} R_{k,k+1} + \sum_{k=1}^{N-1} P_{k,k+1} \quad (2.14)$$

$$S_{k,k+1} = \frac{P_{f1}}{C_{k,k+1}} \quad (2.15)$$

$$R_{k,k+1} = \frac{d}{s} \quad (2.16)$$

where:

D is the end-to-end delay

$D_{k,k+1}$ is the delay experienced by packets of flow $f1$ over the link between nodes k and $k+1$,

$S_{k,k+1}$ is the transmission delay

$R_{k,k+1}$ is the propagation delay

$P_{k,k+1}$ is the processing delay

P_{f1} is the packet size

$C_{k,k+1}$ is the link rate

d is the length of transmitting medium

s =propagation speed of medium

2.2.11 Poisson Traffic Model

Teletraffic theory is the application of mathematics to the measurement, modeling, and control of traffic in communication networks. The aim of traffic modeling is to find stochastic processes to represent the behavior of traffic. Traffic source models can be defined from network measures to give an accurate statistical distribution of inter-arrival time and packet size at packet level. It is a mathematical approximation for real traffic behavior. Computer simulations have been growing

considerably, which poses different requirements for traffic source models. Ideally, a suitable traffic source model should represent real traffic or capture essential characteristics of traffic that have a significant impact on network performance. The Poisson arrival process has been a favorite traffic model for data and voice. The traditional assumption of Poisson arrivals has been often justified by arguing that the aggregation of many independent and identically distributed renewal processes tends to a poisson process. Poisson arrivals with mean rate λ are separated by interarrival times r is express as follows(Zhang *et al.*, 2015):

If Y is the random variable then Y has a poisson distribution with parameter $\lambda . e^{-\lambda}$

$$P_r(\mathrm{Y} = \mathrm{r}) = \mathrm{e}^{-\lambda} \times \frac{\lambda^r}{r!} \quad (2.17)$$

where :

λ is the mean rate

r is Numbers of generated packets and inter-arrival time

$e^{-\lambda}$ is the exponential function

2.2.12 ROHC Attack

ROHC requires information exchange between the compressor and decompressor to map Context IDentifies (CID) to header information. This process of information exchange can lead to various attacks. These attacks are as follows:

1 ROHC False Initialization and Refresh Attack (ROHC FIA): In normal ROHC operation, a compressor sends an uncompressed packet with an additional IR header to establish the context and may later alter portions of the context when packet fields change by sending additional IR packets. In the ROHC False IR Attack (FIA), the malicious node listens for IR headers establishing contexts. Each time it hears one, it spoofs the sender and send a new IR header to the decompressor to change the context in a few minor ways. Future packets received at the

intended receiver will fail CRC test at the decompressor and be dropped, while the malicious node, which knows the true context, is able to decompress them successfully.

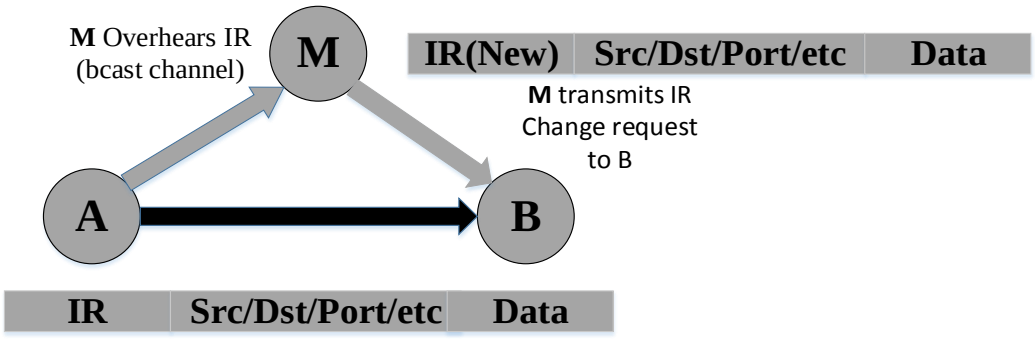


Figure 2.4: ROHC False IR Attack (Cheng & Moore, 2013)

Figure 2.4 illustrates this situation. Node A sends an IR packet intended for node B. Node M overhears the IR packet and initiates an IR change request to B, resulting in B unable to decompress compressed headers from A. The decompressor sends a negative acknowledgement (NACK) packet for the unsuccessful compressed packets. The compressor then sends a new IR header to repair the context. Following this new IR header, the malicious node changes the context. The cycle then repeats. The end result is that the decompressor successfully receives some fraction of the packets, depending on how many uncompressed frames the compressor sends before changing to the compressed state. Additionally, the relatively large number of uncompressed packets with IR headers increases the load on the network, negating much of the benefits of ROHC(Cheng & Moore, 2013).

2 ROHC False Acknowledgement Attack (FAA): Figure 2.5 illustrates the FAA.The ROHC FAA builds on the false initialization and refresh attack. It plays on the fact that compressors are often configured to accept acknowledgements (ACKs) for IR packets successfully received. Under this attack, after changing the context with a fake IR header, the malicious node listens for compressed packets. Upon receiving one, the malicious node sends a fake acknowledgement to

the compressor. As a result, the compressor will end up receiving both an acknowledgement and a negative acknowledgement for the same packet.

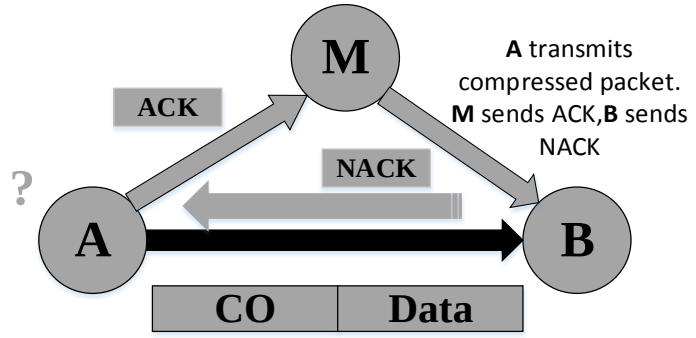


Figure 2.5: ROHC False ACK Attack (Cheng & Moore, 2013)

3ROHC False Negative Acknowledgement Attack (ROHC False NACK Attack): Figure 2.6 describes the false negative acknowledgement attack.

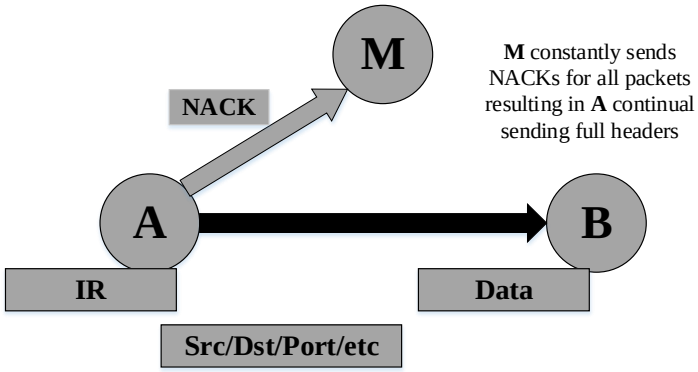


Figure 2.6:ROHC False NACK Attack (Cheng & Moore, 2013)

The ROHC False NACK attack leverages ROHC compressor built-in mechanisms to re-transmit IR headers when it receives a NACK packet, indicating that the decompressor believes there is damage to the static portion of the context. As shown in Figure 2.6, in this attack, the malicious node (M) simply sends false NACK to all ROHC packets received. The compressor (A) continues to send IR headers with each packet. The end result is that while the decompressor

decompresse all packets, there is no compression gains in the network. In fact, the additional IR headers make packets larger than the case without ROHC enabled(Cheng & Moore, 2013).

4 Brute force attack: A brute force attack is a trial-and-error method used to randomly guess a person's user name, password, credit card number, or cryptographic key. Brute force attacks may be used by criminals to crack encrypted data, or by security analysts to test an organization's network security. A brute-force attack is an attempt to discover a password by systematically trying every possible combination of letters, numbers, and symbols until the correct combination is discovered (Limaye et al., 2014). Brute force attacks are modelled using pseudorandom number generators such as linear congruential generators. The flowchart for bruteforce attack is shown in Figure 2.7

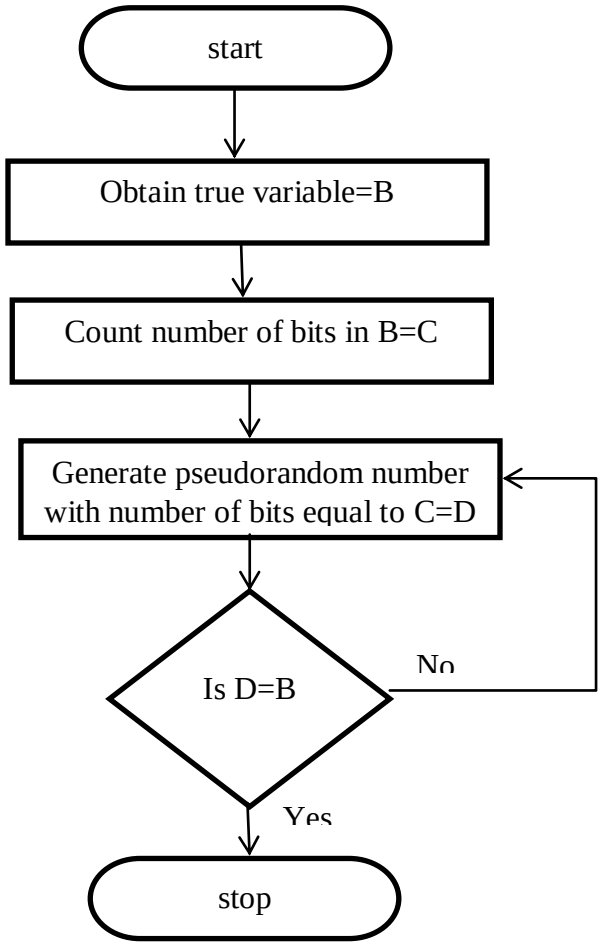


Figure 2.7: Flowchart of Brute Force Attack (Limaye et al., 2014)

5IP spoofing: This is a means of changing the information in the headers of a packet. Spoofing is used to impersonate a different machine from the one that actually sent the data. This can be done to avoid detection and/or to target the machine to which the spoofed address belongs. By spoofing an address that is a trusted port, the attacker can get packets through a firewall (Kaur *et al.*, 2015).

2.2.13 Intrusion Detection System (IDS)

Intrusion is an unwanted activity by an attempt to compromise the confidentiality, integrity, availability, or to bypass the security mechanisms of a computer or network. Intrusion detection provides deterrence for intruder and serves as an alarm mechanism for a computer system or a network to manage security plan successfully.Network Intrusion Detection systems (NIDS) are the most efficient way of defending against network-based attacks (Debar *et al.*, 2000; El-bakry & Mastorakis, 2008). An IDS is defined as software or hardware monitoring tool which detects internal or external cyber-attacks. It observes and investigates the system and user activities, recognize patterns of known attacks and identify abnormal network activity(Can & Sahingoz, 2015; Kaur *et al.*, 2015).The National Institute for Standards and Technology (NIST) describes intrusion detection as "the process of monitoring the events occurring in a computer system or network and analysing them for signs of intrusions (Liao *et al.*, 2013).Itis a software that automates the intrusion detection process and detects possible intrusions(Srivastava & Richhariya, 2013).Intrusiondetection systems serve three essential security functions: they monitor, detect, and respond to unauthorized activity(Srivastava & Richhariya, 2013). They monitor packets by analysis to detect the presence of a malicious node and respond to prevent against damage from the malicious attacker. Figure 2.8 shows the architecture of a common intrusion detection framework.

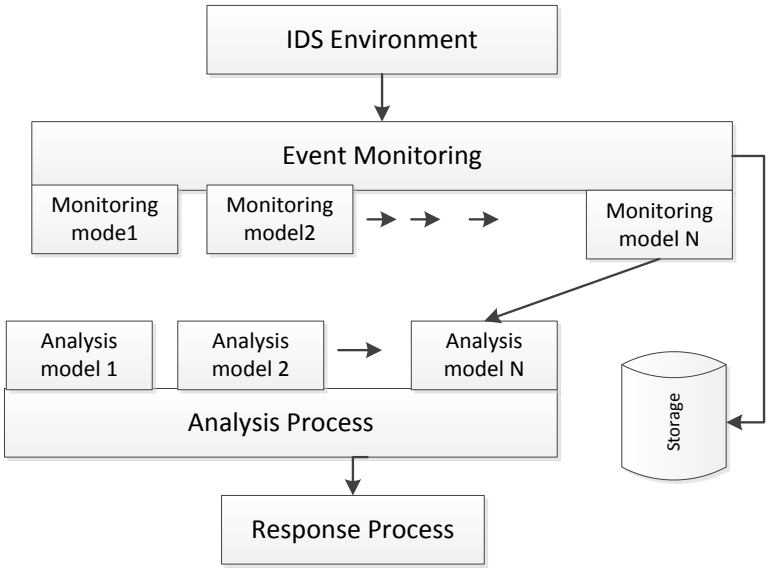


Figure 2.8: Intrusion Detection Framework Architecture (Dubey & Tripathi, 2015)

2.2.14 Categories of IntrusionDetection Systems

There exist two main categories of intrusion detection systems (Srivastava & Richhariya, 2013):

1. Signature-based (SBIDS)
2. Anomaly-based (ABIDS).

1 Signature-Based Intrusion Detection System (SBIDS): This type of intrusion detection system maintains a database of signatures of known attacks and tries to match these signatures against the analysed data. When a match is found, an alarm is raised. Signature detection involves searching network traffic for a series of malicious bytes or packet sequences. The main advantage of this technique is that signatures are very easy to develop and understand (Jyothsna *et al.*, 2011).SBSS is based on regular expressions and string matching and work well against fixed behavioural pattern(Dubey & Tripathi, 2015).

2Anomaly-Based Intrusion Detection Systems(ABIDS): The anomaly based detection is based on defining the network behaviour.If the network behaviour is in accordance with the predefined behaviour, then it is accepted else it triggers the event in the anomaly detection(Jyothsna *et al.*, 2011).It first builds a statistical model describing the normal network traffic, then flags any

behaviour that significantly deviates from the model as an attack**.** Anomaly detection IDSs look for behaviour that deviates from normal behaviour. Normal behaviour is defined by selected characteristics of a baseline data set on which the anomaly detection system has been trained. They are designed to detect general system misuse and attacks for which no signature exists, often called zero-day attacks(Azhagiri *et al.*, 2015; Dubey & Tripathi, 2015).

2.2.15 Selective Watchdog and Watchdog Intrusion Detection System

Figure 2.9 illustrate the watchdog technique.Watchdog is the basis for many of the proposed IDS technique that are found in literature(Al-Roubaiey *et al.*, 2010). It is an IDS developed by Sergio Marti, T.J. Giuli, Kevin Lai, and Mary Baker atStanford University in 2000 with an aim to improve the throughput of network in the presence of malicious nodes (Marti *et al.*, 2000).Watchdog node detects malicious misbehaviours by listening to its next hop's transmission. If a watchdog node overhears that its next node fails to forward the packet within a certain period of time, it increases its failure counter. Whenever a node's failure counter exceeds a predefined threshold, the watchdog node reports it as misbehaving. In this case, the pathrater cooperates with the routing protocols to avoid the reported nodes in future transmission(Shakshuki *et al.*, 2013).It was first used in a mobile adhoc network and proven to significantly increase the network throughput.

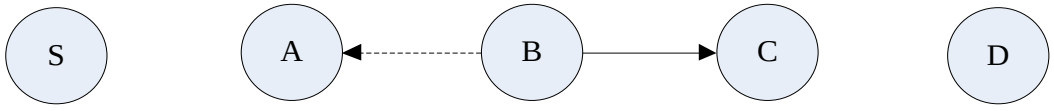


Figure 2.9: Watchdog Operation (Kang et al., 2011)

Figure 2.10 describes the flow chart for selective watchdog technique. The selective watchdog technique detects intrusion in the presence of black hole node in a network.it is an improvement over the watchdog technique. In selective watchdog technique, network nodes are divided into

clusters and only nodes in the cluster which have value greater than threshold monitor their neighbours.In black-hole attack there is great possibility that black-hole nodes have sent the highest sequence number to the source reply.Therefore, the IDS algorithm maintains the list of all the nodes which send the route reply to the source with sequence number greater than the threshold.

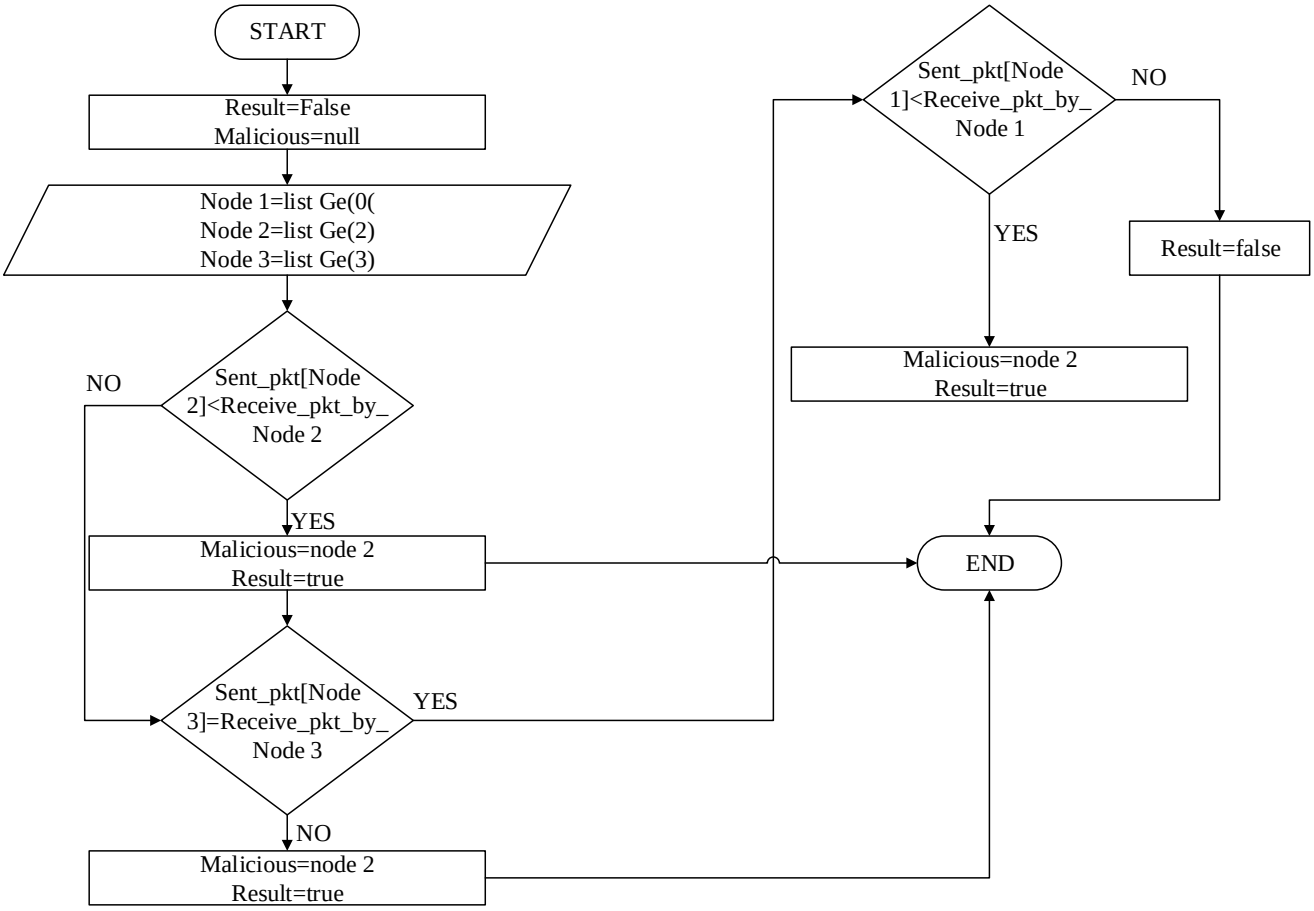


Figure 2.10 Selective Watchdog Technique (Dua & Mishra, 2014)

2.2.16 The Age of Internet Protocol Version Six (IPv6)

The current Internet Protocol (IP) version four (IPv4) was born to interconnect educational and government institutions in United States. Thus, the original IP is naturally light and simple. Since the IPv4 based Internet is widely used throughout the world, current status of IPv4 is beyond imagination of the initial motivation it was created for. Internet users have increased by the

geometric progression every year, which incurs shortage in address resources. IPv4 uses 32-bit identifier so that there are 4.3 billion identifiers numerically, but address allocation based on classes allows only 0.5 or 1 billion address space(Mun & Lee, 2005). In the early 1990s, the Internet Engineering Task Force (IETF) realized that the IPv4 address space was running out at a dangerous rate. Around 1990, about one-eighth of the 3.7 billion usable IPv4 addresses were given out, a number that doubled every five years. IPv4 is expected to run out of previously unallocated addresses by 2012. This apparently impending doom prompted the IETF to start work on "IP next generation" (IPng), which eventually led to the creation of the Internet Protocol Version six (IPv6) standards. The first IPv6 Request For Comments (RFC) was published in 1995(Beijnum, 2006; Perset, 2010).

IPv6 is a new version of IP which is expected to substitute IPv4. It is very difficult to predict exactly when IPv4 will eventually come to an end. However, most experts in IP area expect that the coexistence of two protocols will last for a long time. As a successor of IPv4, IPv6 is expected to bring lot of advantages over IPv4 such as more address space, stateless auto configuration, providing quality of service (Ali, 2012; Davies, 2012). It introduces concerns related to expanded packet headers as the size of packet header doubled from 20 bytes for IPv4 to at least 40 bytes for IPv6. Hence, the need for IP packet header compression.Figure 2.11 shows an IPv6 header.

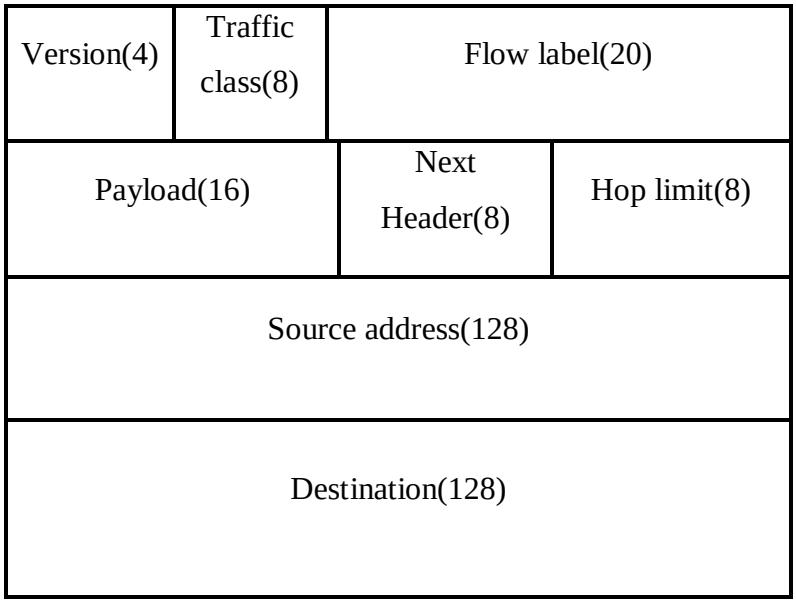


Figure 2.11: IPv6 Packet (Tong et al., 2011)

2.2.17 Internet Protocol Routing

Routing is the cumulative processes that discover paths through the network to specific destinations, compare redundant routes mathematically, and build tables that contain routing information.Routing protocols define a set of rules which govern the journey of message packets from source to destination in a network(Dhenakaran & Parvathavarthini, 2013). Figure 2.12 shows the routing algorithm for a router.

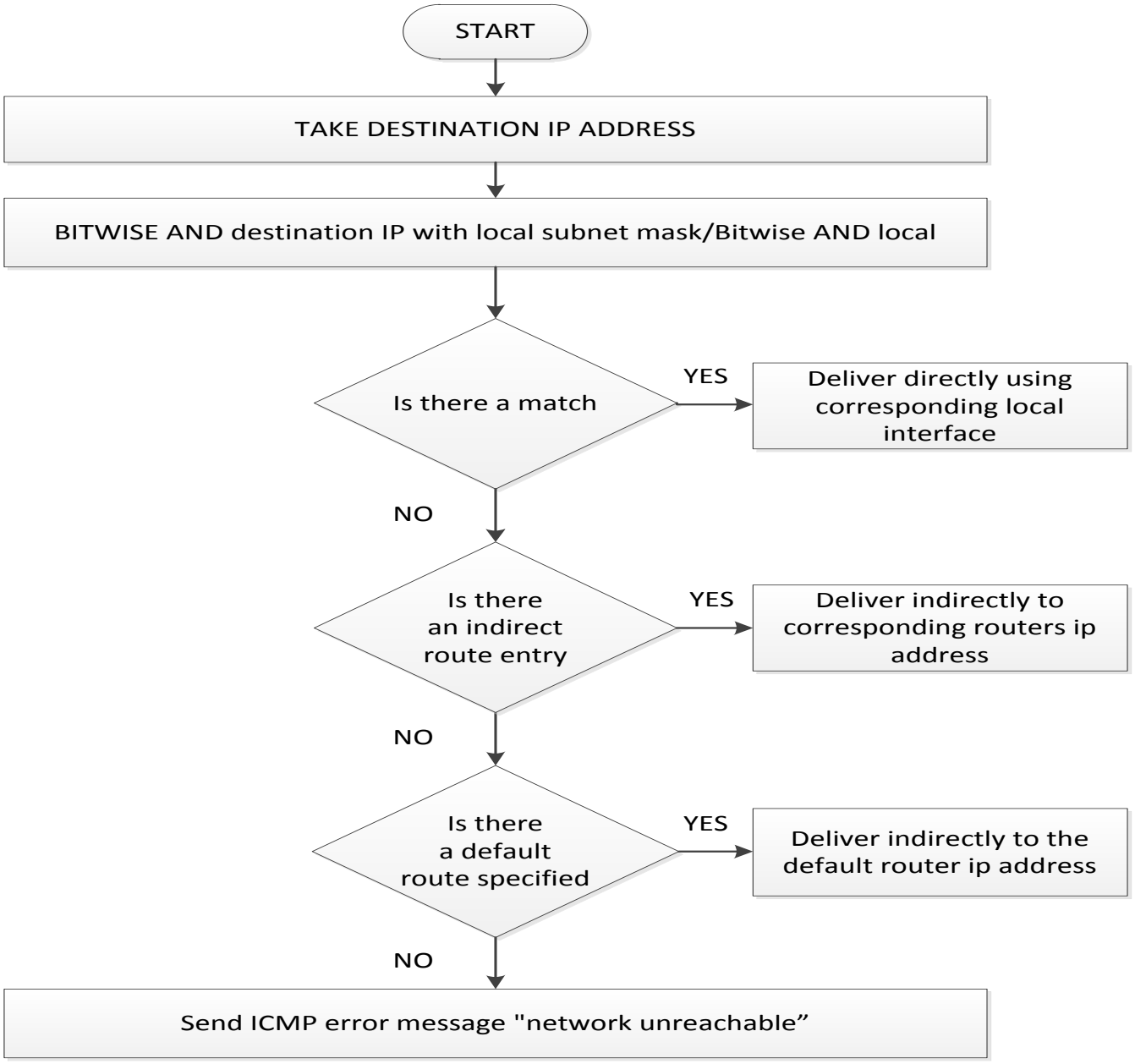


Figure 2.12: IP Routing Algorithm (Parziale et al., 2006)

The router must check that the received packet is properly formed for the protocol before it proceeds with protocol processing. This involves checking the version number, checking the header length field and calculating the header checksum.The router then performs a table lookup to determine the output port onto which to direct the packet and the next hop to which to send the packet along this route. This is based on the destination IP address in the received packet and the subnet mask(s) of the associated table entries.The result of this lookup could imply that the

destination address is one of the router's local addresses,to a next-hop router or a default route address.

2.2.18 CRC

CRC is one of the methods of detecting the errors in the information during transmission. CRC are used for detecting the corruption of digital content during its production, transmission, processing or storage. CRC algorithms treat each bit stream as a binary polynomial and calculate the remainder from the division of the stream and the corresponding remainder is transmitted together with the bit stream(Indu & Manu, 2012). At the receiver side, CRC algorithms verify that the correct remainder has been received. Long division is performed using modulo-2 arithmetic.

1. Let $N(\mathrm{X})$ be the data polynomial.
2. Let $\mathrm{G(X)}$ be a generator of degree n
3. Let $\mathrm{R(X)}$ be the residual after division of $N(\mathrm{X}).\mathrm{X}^n$ can be divided by $\mathrm{G(X)}$. Thus, the resulting code polynomial is:

   $$G(X).Q(\mathrm{X}) + \mathrm{R(X)} = \mathrm{N(X)}.\mathrm{X}^n \text{ Or } G(\mathrm{X}).\mathrm{Q(X)} = \mathrm{N(X)}.\mathrm{X}^n + R(\mathrm{X}) \quad (2.18)$$

   Thus,
   $N(\mathrm{X}).\mathrm{X}^n + R(\mathrm{X})$ is divided by $\mathrm{G(X)}$.
4. The sender sends $S(\mathrm{X}) = \mathrm{N(X)}.\mathrm{X}^n + R(\mathrm{X})$.
5. The receiver gets $H(\mathrm{X}) = \mathrm{S(X)} + \mathrm{E(X)}.\mathrm{E(X)}$ is called error polynomial. It has a term X^* if and only if there is an error (i.e. 0 became 1 or vice versa) at the bit k (k=0, 1, 2...).
6. The receiver divides $H(\mathrm{X}) / \mathrm{G(X)} = (\mathrm{S(X)} + \mathrm{E(X)} / \mathrm{G(X)})$ as follows(Kowalk, 2006):

$$\frac{H(\mathrm{X})}{G(\mathrm{X})} = \frac{S(\mathrm{X}) + \mathrm{E(X)}}{G(\mathrm{X})} = \frac{S(\mathrm{X})}{G(\mathrm{X})} + \frac{E(\mathrm{X})}{G(\mathrm{X})} = \frac{N(\mathrm{X}).\mathrm{X}^k + R(\mathrm{X})}{G(\mathrm{X})} + \frac{E(\mathrm{X})}{G(\mathrm{X})} = Q(\mathrm{X}) + \frac{E(\mathrm{X})}{G(\mathrm{X})} \quad (2.19)$$

2.2.19 Block Cipher Cryptographic System

Block cipher cryptographic system is cryptographic system which takes a number of bits and encrypts them as a single unit, padding the plaintext so that it is a multiple of the block size(Huang *et al.*, 2013).It is a function which maps n-bit plaintext blocks to n-bit cipher text

blocks,where n is called the block length.It is utilized within a data processing environment to ensure complete privacy of data and information that is stored or processed within a computing system. All communications transmitted from terminal input are encrypted into a block cipher by the use of cryptographic system operating under the control of an encryption key which is inputed to the terminal device. At the receiving station, the encryption key is used to decipher all received ciphered.

A block cipher is a triple of sets K, M, C and a mapping $F: K \times M \rightarrow C$ such that for all $k \in K$ the mapping $F_k : M \rightarrow C$ given by $F_k(m) = F(k, m)$ for all $m \in M$ is reversible. The key k also determines the reverse mapping $F^{-1}_k: C^1 \rightarrow M$, where $C^1 = F_k(M) = \{F_k(m) : m \in M\}$,such that $F^{-1}_k(c) = m$ if and only if $c = F_k(m)$ for all $m \in M$ and all $c \in C^1$.When each mapping F_k is a permutation these inverse mappings constitute a corresponding (inverse) block cipher $F^{-1}: K \times C^1 \rightarrow M$. The elements of K are the keys and the elements of M are the messages. The ciphertext $c = F_k(m) \in C$ is the encryption of $m \in M$ under key $k \in K$(Martin *et al.*, 2005).

2.2.20 Brute Force String Matching

A string is a sequence of characters from an alphabet.The simplest algorithm for string matching is a brute force algorithm.Brute-force string matching compares a given pattern with all substrings of a given text. Those comparisons between substring and pattern proceed character by character unless a mismatch is found. Whenever a mismatch is found, the remaining character comparisons for that substring are dropped and the next substring can be selected immediately.The brute force algorithm checks all positions in the text between 0 and n-m, whether an occurrence of the pattern starts there or not. Then, after each attempt, it shifts the pattern by exactly one position to the right (Christian & Lecroq, 2004).

for *i ←0 to n−m* ***do***

j ←0

while *j<m and P[j]=T[i +j]****do***

j ←j + 1

if *j =m return i*

return *−1*

(Pandey, 2008)

$$C(n,m)=\sum_{i=0}^{n-m}\sum_{j=0}^{m-1}1=\sum_{i=0}^{n-m}(\mathrm{m}-1-0+1)=\sum_{i=0}^{n-m}(\mathrm{m}) \quad (2.20)$$

(Levinti, 2012)

2.2.21 Performance Metrics

The performance metrics considered are:

1. Throughput
2. Packet Delivery Success(PDS)

Throughput: Throughput can be defined as the number of packets delivered per unit time. It is an all-round important performance measure for communication networks(Chauhan & Sharma, 2015a). A typical method of performing throughput measure is to transfer a big file in network and measure the time taken to complete the task(Tan *et al.*, 2012).

$$\text{Throughput}=\frac{\text{Filesize}}{\text{Transmission_Time}}(\text{bps}) \text{ (Meenakshi, 2014)} \quad (2.21)$$

$$\text{Throughput}=\frac{B_r - Loss_L * (\text{ACL}+\text{P})}{D} \quad (2.22)$$

where:

B_r is the totalnumber of transmitted byte

$Loss_L$ is the burst packet loss

ACL is the average compression length

P is the payload size

Packet Delivery Success (PDS) : It is the ratio of number of packets received at the destination to the number of packets generated at the source. A network should work to attain high PDS in order to have a better performance. PDS shows the amount of reliability offered by the network. The greater value of PDS means the better performance of the protocol(Meenakshi, 2014).

$$PDS=\left(\frac{Total_number_of_packets_received}{Total_number_of_packets_Send}\right)*100 \text{ (Chauhan \& Sharma, 2015a)} \quad (2.23)$$

$$PDS=\frac{B_r}{B_{w/o}}*100 \quad (2.24)$$

where:

$B_{w/o}$ is the transmitted with packet loss

B_r is the transmitted

2.3 Review of Similar Works

Some literatures relevant to the development of an improved intrusion detection based secured Robust Header Compression are described in this section

Marti *et al.*, (2000) proposed a technique to mitigate routing misbehaviour in the presence of nodes that agree to forward packets but fail to do so. The work developed a watchdog algorithm that identifies misbehaving nodes and a pathrater that helps routing protocols avoid these nodes. The technique maintained a buffer of recently sent packets and inspected each packet to identify misbehaving nodes. Packets without misbehaviour are removedfrom the buffer. If a packet remained in the buffer longer than a certain timeout, a failure tally is incremented for the node responsible for forwarding on the packet. If the tally exceeds a certain threshold, it identifies the node as a misbehaving node. Simulation results show increased throughput in the presence of misbehaving nodes with increased percentage of routing overhead. However, due to dependency on acknowledged packets, result shows thattheir technique did not detect a misbehaving node in the presence of false misbehaviour and partial dropping.

Al-Roubaiey *et al.*, (2010) introduced an Adaptive ACKnowledgment Intrusion Detection (AACK) scheme. Previous research work solved the false misbehaviour and partial dropping problem of watchdog technique. However,most of their work created more overhead to the routing protocol due to multiple acknowledgments for each packet along the path. The work presented a combination of an Enhanced-TWO ACKnowledgement (E-TWOACK) scheme and an end-to-end acknowledgment scheme with end-to-end acknowledgment as a default mode. To minimize the routing overhead problem of previous research, the work introduced a switching scheme for switching between the two techniques. Simulation results show an improvement in packet delivery ratio and routing overhead. However the work used anend-to-end acknowledgment packet scheme which resulted to high end-to-end delay on long paths due to significant time for the end to end acknowledgments.

Shanmugavadivu &Nagarajan. (2010) proposed a network intrusion detection system using fuzzy logic.The traditional intrusion detection relies on the extensive knowledge of security experts, in particular, on their familiarity with the computer system to be protected. To reduce this dependence, the work proposes the development ofan anomaly based intrusion detection system for detecting the intrusion behaviour within a network. A fuzzy decision-making module was designed to build the system more accurately for attack detection, using the fuzzy inference approach. An effective set of fuzzy rules for inference approach were identified automatically by making use of the fuzzy rule. At first, the definite rules were generated by mining the single length frequent items from attack data as well as normal data. Then, fuzzy rules were identified by fuzzifying the definite rules and these rules were given to fuzzy system, which classify the test data. Experimentation results showed that the proposed method is effective in detecting known intrusion in computer networks. However,result show that the method cannot find out unfamiliar intrusions.

Kim &Kim. (2012) Presented an agent-based honeynet framework for protecting servers from new internet worms in a campus network. In the work, agents remove malicious processes and executable files on servers infected by zero-day attacks as soon as the honeynet detects them.The work propose the installation of agent on the client and the server machine, which creates DecoyPorts designed to lure internet worms. Every interaction with a DecoyPort is then considered a suspect interaction. Result show that the agent-based honeynet detects known and unknown worms. However it requires high cost due to installation of both clients and servers.

Gavaskar *et al.*, (2012)introduced an effective method to prevent internet protocol spoofing in a network that compressed internet protocol version six(IPv6) header.The main objective of internet protocol (IP) compression is to avoid the overhead, which provides bandwidth utilization. However, due to context setup process, the internet protocol header compression exposes the network to Internet protocol spoofing vulnerability, which leads to denial of service. To solve this issue, the work proposed a method that encrypted the IPv6header using transformation function. Simulation results show an improvement in throughput and packet delivery success. However, in other to avoid end to end delay and avoid increase packet size, the work use a transformation function with short key length for encryption which makes the algorithm vulnerable to brute force attack.

Bow-Nan *et al.*, (2013)developed a Mobile adhoc network internet protocol (MIPHC) header compression. The work was tailored towards compressing internet protocol version four (IPv4) and internet protocol version six (IPv6) packets in Mobile adhoc network (Manet) environment. The MIPHC alleviated the need for context setup by pre-sharing a context table used by all nodes for compressing and decompressing packet header in the network. Simulation results revealed that the developed MIPHC scheme exhibited less end-to-end delay compared to hybridized Mobile internet protocol header compression (MIPHC/ROHC) and Robust Header Compression

(ROHC) technique. However the MIPHC scheme maintained less information in the context table which resulted to low compression gain and higher network load. In addition the work maintained a context setup which exposes the nodes to internet protocol spoofing and denial of service attack.

Cheng &Moore. (2013) examined three attacks and their effect on networks that use ROHC to compress and decompress internet protocol headers. These attacks include (1) false initialisation/refresh; (2) false acknowledgement attack and (3)false non-acknowledgement attack .The work showed that the attacks resulted in denial of service and packet interception.To mitigate the attacks, the work also proposed two methods which are internet protocol security packet encryption (IPsec) after compression and cyclic redundancy check encryption. Simulated results show an improvement in delivery success as well as average network load under the three attacks as compared to Robust header compression under the same attack. However, the internet protocol security packet encryption is not a practical method to secure the Robust header compression technique due to the significant overhead in internet protocol security tunnelling. In addition the CRC is only 3-8 bits long therefore, limited to 256 possible combinations. This means that a malicious node could still attempt a brute force approach where it sends fake packets with all possible CRC combinations therefore resulting in decryption of the CRC,denial of service and packet interception.

Shakshuki *et al.*, (2013) proposed a new intrusion-detection system named Enhanced Adaptive Acknowledgment (EAACK).In detecting malicious intent in a network, many of the intrusion detection technique adopts an acknowledgement-based scheme which does not guarantee security. To address this concern, the work adopted a Digital Signature Algorithm **(**DSA) and Rivest, Shamirand Leonard Adleman (RSA) digital signature to digitally sign the acknowledgement packets. The proposed Enhanced Adaptive Acknowledgment (EAACK)intrusion detection

system was used to solve false misbehaviour problem of watchdog technique. Simulation results demonstrated positive performances against Watchdog, Two Acknowledgement Scheme (TWOACK) and Adaptive Acknowledgment Intrusion Detection (AACK) in the cases of false misbehaviour report, therefore resulting to high packet delivery success. However, due to the size of the encryption key and complexity of the encryption algorithm, the proposed method generated more routing overheads and increased the end-to-end packet delay.

Cheng *et al.*, (2014)implemented a mobile adhoc network Internet protocol header compression (MIPHC) in a Linux kernel.The mobile internet protocol header compression implemented in the Linux kernel comprises of a module that handles the actual header compression and decompression and a userspace control process that communicates withthe kernel module through a generic netlink interface. The MIPHC contains a separate context identifier (CID) table for internet procol version four (IPv4) and internet protocol version six (IPv6) which are used to compress and decompress packets. These tables are maintained through the IPHC_CTL userspace client. The client sends generic netlink commands to the kernel module, injecting and removing context identifier (CID) to internet protocol (IP) subnet mappings, setting log levels and adding/removing MIPHC to various interfaces. Experimental result show less end-to-end delay compared to hybridized Mobile internet protocol header compression (MIPHC/ROHC) and Robust header compression (ROHC) technique. However Mobile adhoc network internet protocol header compression technique (MIPHC) maintained less information in the context table which resulted to low compression gain.In addition the work maintained a context setup which exposes the nodes to internet protocol spoofing and denial of service attack.

Dua &Mishra (2014)presented a selective watchdog intrusion detection scheme that detected intrusion in a network with black-hole node. In black-hole attack, the black-hole node will send the highest sequence number to the source in route reply.The work presented an intrusion

detection systemalgorithm that maintains a list of all the nodes sending route reply to the source with sequence number greater than the threshold value. Their technique dividednetwork of nodes into clusters with selected nodes as monitors. Suspected nodes were further monitored to detect the presence of black hole node. Simulation result show that the proposed scheme performs better than other intrusion detection system in terms of detection time. However, due to the dependency on acknowledged packets this scheme failed to detect forged acknowledgment packets.

Cha *et al.*, (2015)presented a technique that improved packet header compression using adaptive sliding window size. Existing robust header compression mechanism uses window based least significant bits encoding for compression of dynamic fields in the protocol headers with a fixed sliding window value.Hence,if the number of packets loss exceeds the value of sliding window, then compression rate is reduced. To address this problem, the work propose an adaptive sliding window that adjusted the values of sliding window according to packet loss. The method adjusts the value after a decompression failure. Theinitial value is set to 2 because the compressed header size in the value 2 is the smallest. If packet loss happens, then the value of sliding window is increased by one. However, if packet loss does not happen, then the value is set to the initial value 2 again.Experimental results show an increase in header compression ratio and decrease in the average header size. However, due to context setup between the compressor and decompressor, the proposed method exposes the network to brute force and denial of service attack.

Chauhan &Sharma (2015b)presentedan approach for improving the efficiency of internet protocol version 6 to 4 tunnelling mechanism using Tunnel header compression.The Tunnelling header compression technique compressed the internet protocol version six packets before encapsulating in an internet protocol version four packets. These encapsulated packets were sent

through a tunnel. At the Tunnel end point the receiving router striped off the internet protocol version four headers and decompress the inner internet protocol version six headers. Simulation results show that compressing the internet protocol version six headers resulted to better network deliverables in terms of throughput, average end-to-end delay, jitter and packet delivery ratio as compared to uncompressed network.However it resulted to increased end-to-end delay andmaintained a context setup that exposes the nodes to brute force and denial of service attack.

From the literatures reviewed, it is evident that most research work focused on securing robust header compression using cryptographic means such as encrypting the CRC field. The CRC encryption method solves the problem of spoofing and denial of service attack in robust header compression.However, the encryption method remains vulnerable to brute attack due to short available length of the cyclic redundancy check field. Literatures also suggested intrusion detection as a second line of defence when encryption is passed by security attack (Can & Sahingoz, 2015). From the work of Dua &Mishra. (2014), it is also evident that IDS based on selective watchdog technique has shown good performance when used to solve security problem. Therefore developing an algorithm by modifying the selective watchdog intrusion detection to mitigate brute force attack is the basis for the motivation of this research.

CHAPTER THREE

MATERIALS AND METHOD

`3.1Introduction

This chapter describes the methods, materials and procedures employed for the successful completion of this research. The secured ROHC and an IDSROHC technique based on selective

watchdog intrusion detection system were developed and simulated using MATLAB based GUI .The steps of the methodology adopted for this research, towards developing improved intrusion detection based secured robust header compression techniqueare highlightedin section 1.4.

3.2 Original Secure RObust Header Compression Technique

Figure 3.1 shows the flowchart for implementation of the existing Secured RObust Header Compression (ROHC). The processes involved in the replication of the existing Secured ROHC are discussed in details in the following sub-sections:

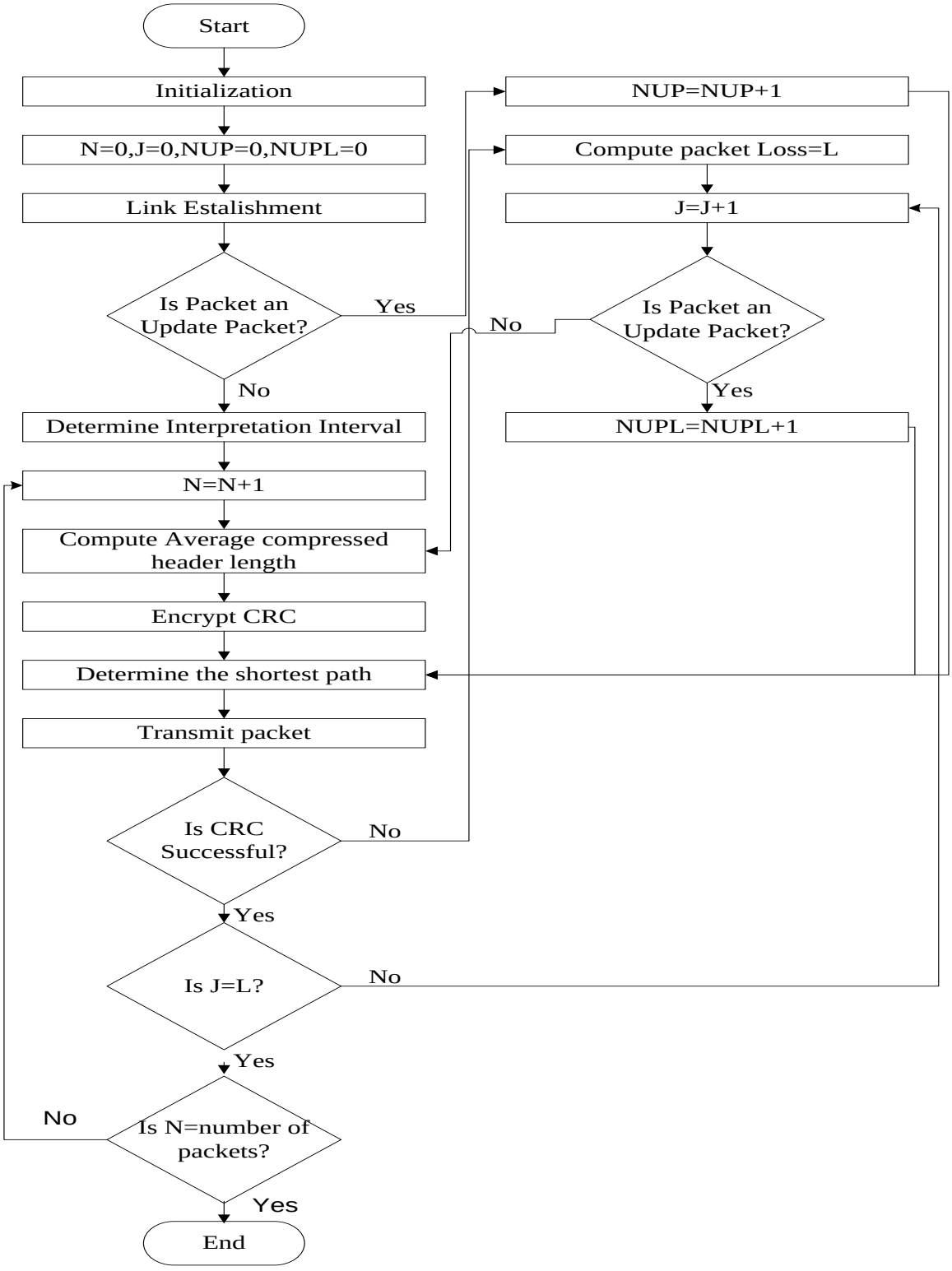


Figure 3.1: Flowchart for implementation of Secured ROHC

3.2.1 Node distribution UsingWeighted Network Graph and Random Waypoint Model

Figure 3.2 shows the snippet for creation and distribution of the nodes using weighted network graph and random waypoint model as discussed in section 2.2.5 while Figure 3.3 shows its MATLAB implementation. The complete program is shown in Appendix A and B.

```
s_input = struct('V_POSITION_X_INTERVAL',[10 100],...%(m)
                 'V_POSITION_Y_INTERVAL',[10 100],...%(m)
                 'V_SPEED_INTERVAL',[0.2 2.2],...%(m/s)
                 'V_PAUSE_INTERVAL',[0 1],...%pause time (s)
                 'V_WALK_INTERVAL',[0 3],...%walk time (s)
                 'V_DIRECTION_INTERVAL',[-180 180],...%(degrees)
                 'SIMULATION_TIME',500,...%(s)
                 'NB_NODES',40);
             nr=5;
             s=300000000;
             link=2000;%kilobyte per seconds
             L=5;% Burst length LB
             e=0.02;% average error probability
             R=300;
             L1=3;
             NUP=2;
             RFI=36;
             L2=60;
             pt=2;
              %The Random Waypoint mobility model
    s_mobility.NB_NODES = s_input.NB_NODES;%define number of nodes
    s_mobility.SIMULATION_TIME = s_input.SIMULATION_TIME;%define simulation tine
    nodex=[];
    nodey=[];
    %randomly select position of nodes
    for nodeIndex_tmp = 1:s_mobility.NB_NODES
        previousX = unifrnd(s_input.V_POSITION_X_INTERVAL(1),s_input.V_POSITION_X_INTERVAL(2));
        previousY = unifrnd(s_input.V_POSITION_Y_INTERVAL(1),s_input.V_POSITION_Y_INTERVAL(2));
          nodex(nodeIndex_tmp)=previousX;
           nodey(nodeIndex_tmp)=previousY;
```

Figure 3.2 :Snippet for Node Creation and Distribution

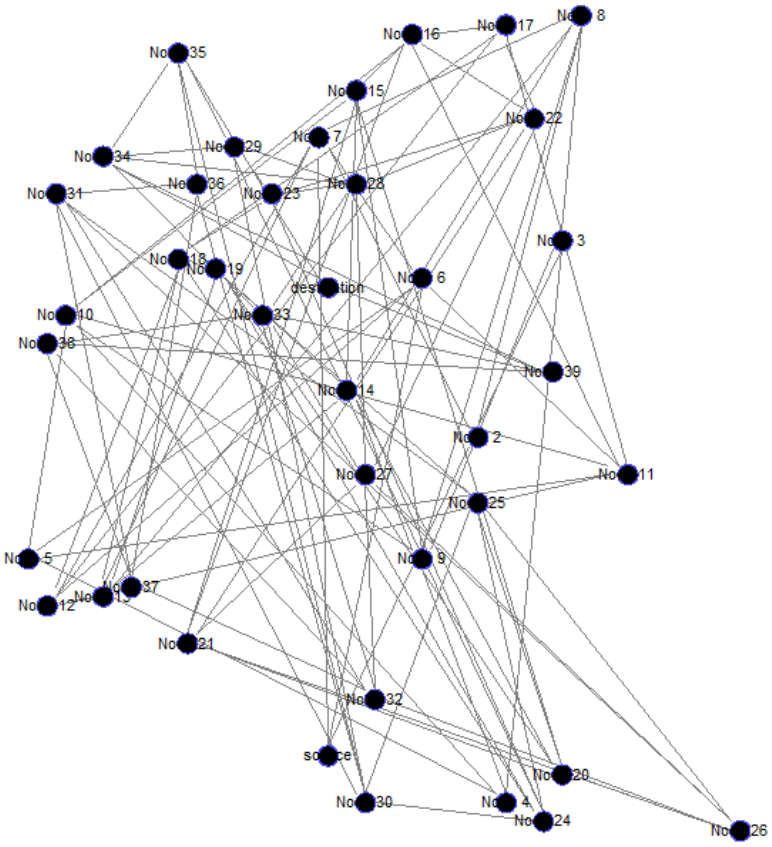


Figure 3.3: Node Creation and Distribution

In this research work, nodes were created using weighted network graph and spatially distributed within a simulation boundary area of 100 by 100m using the Random Waypoint Model (RWM).A graph G consists of a non-empty set V called a set of nodes or vertices or points, and a set E (which was a set of edges of the graph) . Each node was mapped to a set of pairs of edges (V) called the links as represented in equation 3.1.

$$G = (V, E) \quad (3.1)$$

Figure 3.4 shows the connection matrix used in this research. The work establishedconnection between the nodes using an undirected edge. The row of the connection matrix identifies the source nodes while the column identifies the destination nodes.To establish connection between the nodes, a weight of one was assigned to elements Cij of the matrix,where Cij is the link between source node i and destination node j.

$$C = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & - & - \\ - & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & - & - \\ - & - & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & - & - \\ - & - & - & 0 & 1 & 1 & 0 & 0 & 0 & 1 & - & - \\ - & - & - & - & 0 & 1 & 1 & 0 & 0 & 0 & 1 & - \\ - & - & - & - & - & - & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 3.4: Connection Matrix

3.2.2 Obtaining Shortest-Path for Packet Routing

In this work, the bellman-ford algorithm as discussed in section 2.2.9 was use to obtain the shortest path from source node to destination node. Weights were assigned to the edges of the network and stored in sparse matrix. Figure 3.4 shows the connection matrix used in this research work. The connection matrix is such that its element C_{ij} is one when there is an edge from vertex i to vertex j, and zero when there is no edge. The algorithm iterated on the number of arcs in a path. Figure 3.5 shows the snippet for finding the shortest path using Bellman-Ford algorithm while Figure 3.6 shows the Matlab implementation. The red coloured link represent the shortest path from the source node to the destination node. The complete program is shown in Appendix C.

```
[m,n,p,D,t,h,W] = Initialize(G);
p(r)=0; D(r)=0;                    % Set the root of the SP tree (p,D)
for iter = 1:n-1
    optimal = true;
    for arc = 1:m
        u = t(arc);
        v = h(arc);
        duv = W(arc);
        if D(v) > D(u) + duv;      %
           D(v) = D(u) + duv;      % Sp Tree not optimal: Update (p,D)
           p(v) = u;               %
           optimal = false;
        end
    end % for arc
    if optimal
        return                     % SP Tree p is optimal;
    end
end %
```

Figure 3.5: Snippet for Finding the Shortest Path using Bellman-Ford Algorithm

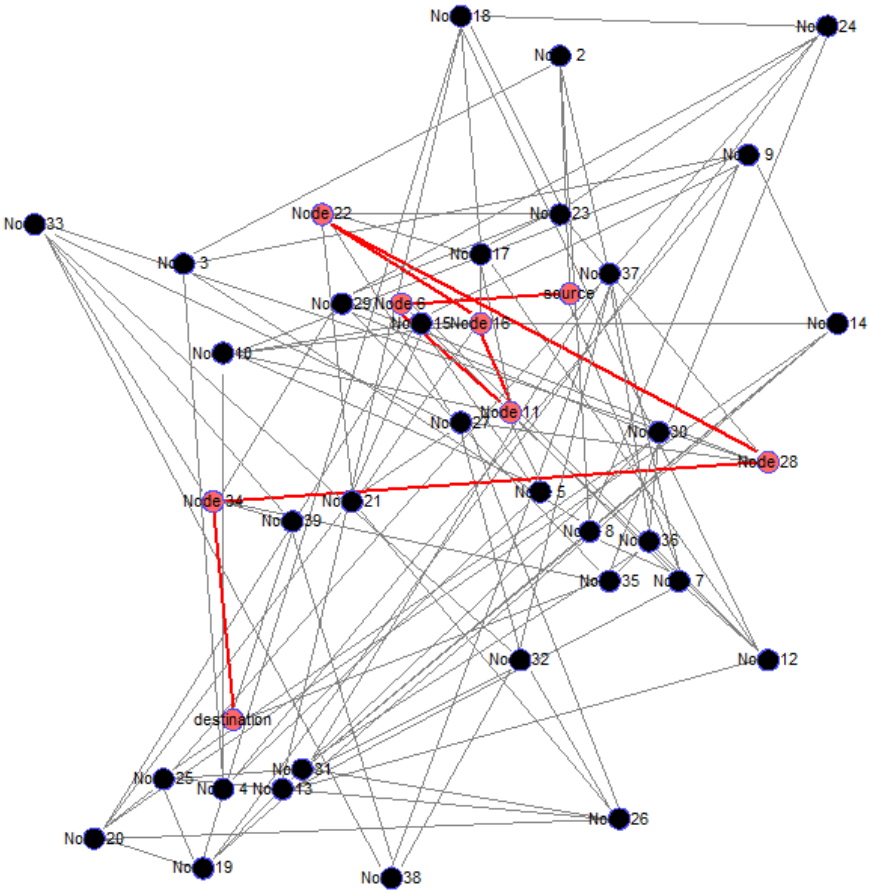


Figure 3.6: MATLAB Implementation of Bellman ford algorithm

3.2.3 Calculating the Required Interpretation Interval

In this work, the interpretation interval was obtained using the mathematical expression for robust header compression model discussed in section 2.2.6. The interpretational interval which performedat the best compression efficiency and high robustness was use in the mathematical expression of average compression length model discussed in section 2.2.7 to calculate the average compression length of Internet Protocol (IP) version six packets. Figure 3.7 shows the snippet for calculating the interpretation interval, the complete program is shown in Appendix D.

```
w=[1:1:40];
for z=1:size(w,2)
    p(z)=(R)*(e/L)*(1-(1/L))^w(z);
end
figure(1);
hbecon=plot(p,w);
set(hbecon,'color','red','LineWidth',3);
y=ylabel('interpretational interval W');
x=xlabel('QoS probability');
legend('e=2% R=300 ')
set(y,'color','blue','fontsize',14);
set(x,'color','blue','fontsize',14);
grid on;
set(gca,'XLim',[0 1], 'YLim', [0 40])
```

Figure 3.7: Snippet for Calculating the Interpretation Interval

3.2.4 Obtaining the Average Compression Length

To calculate the average compression length, the value of the Interpretation interval (W) gotten from the result of section 3.2.3 was used with Table 2.1 to obtain the compressed header length. The value of the compressed header length was use to obtain the average compressed length model from the mathematical equation 2.8. Figure 3.8 shows the snippet for obtaining average compression length. The complete program is shown in Appendix D.

```
%compute average compress header length
acl=(L1*(RFI-NUP)+L2*NUP)/RFI;
```

Figure 3.8: Snippet for Obtaining Average Compression Length

3.2.5Generating the Cyclic Redundancy Check (CRC)

The Cyclic Redundancy Check (CRC) is a technique for detecting errors in digital data. In this research the CRC was computed using the mathematical equation of 2.19. Figure 3.9 show the snippet for generating cyclic redundancy check. The complete program is show in appendix E.

```
function [CRC]=generatecrc(In)
M1=In;
G1 = [1 0 1 0 1 1 1 1 0 ];%generator polynomial
mL = length(M1);
gL = length(G1);
b=zeros(1,gL-1);
M1=horzcat(In,b);
```

Figure 3.9: Snippet for Generating Cyclic Redundancy Check

3.3.6 Encrypting the Cyclic Redundancy Check (CRC)

To produce an encrypted CRC, an uncompressed header was created with MATLAB*struct* command as shown in Appendix F. The uncompressed header was encrypted by applying a 16 bytes block symmetric key encryption. The encrypted uncompressed header was then fed into the CRC algorithm to produce an encrypted CRC.Figure 3.10 shows the flow chart for the encryption process.The plaintext vector was rearranged into a 16 byte 4 by 4 state matrix with each cell containing an 8 bit element. The encryption algorithm iteratively loops the state through a bitwise eXclusive OR (XOR) operation with the key by cylindrical permutation of the state matrix and substitution of the state matrix using a substitution box. It then computes a new state matrix by left-multiplyingthe current state matrix S by the polynomial matrix P as represented in equation 3.2(Cheng & Moore, 2013). Figure 3.11 shows the snippet of the CRC encryption process. The complete program is shown in Appendix G.

$$S_0 = P.S \quad (3.2)$$

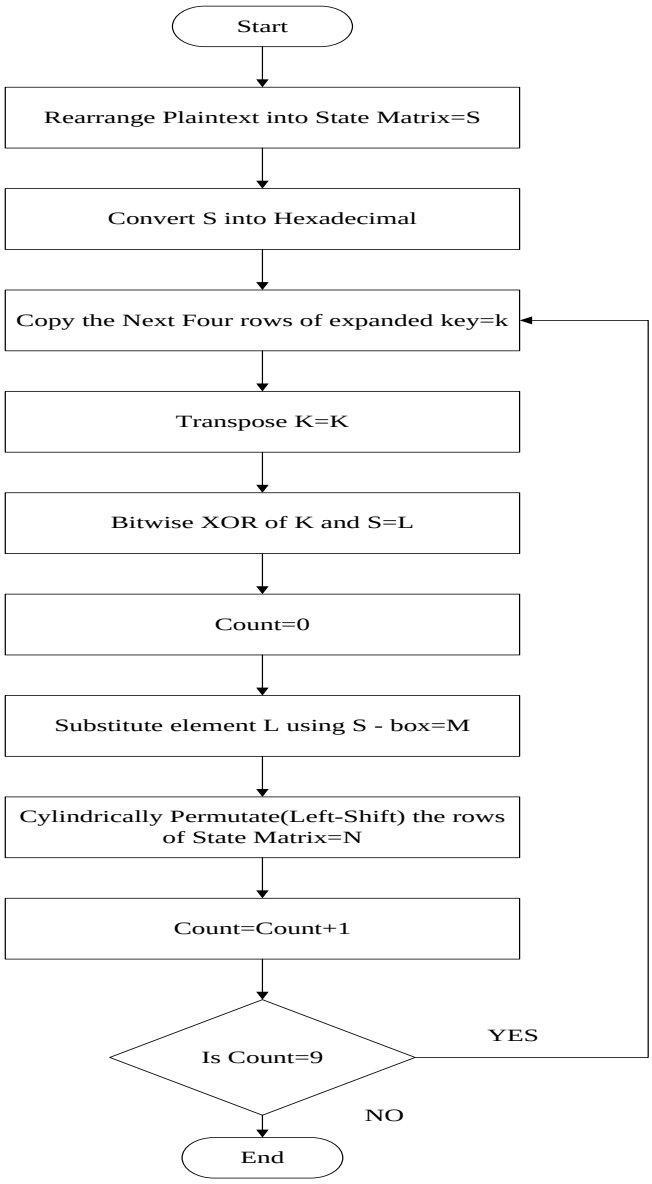


Figure 3.10 CRC Encryption Flowchart (Kazlauskas & Kazlauskas, 2009)

```
[s_box, inv_s_box, w, poly_mat, inv_poly_mat] = aes_init;
plaintext_hex = irpacket();
% Convert plaintext from hexadecimal (string) to decimal representation
plaintext = hex2dec (plaintext_hex)
ciphertext = cipher (plaintext, w, s_box, poly_mat, 1);
re_plaintext = inv_cipher (ciphertext, w, inv_s_box, inv_poly_mat, 1);
```

Figure 3.11: Snippet of CRC Encryption

3.2.7 Key Expansion Model

The encryption process ran through 40 iterative rounds. Each round required an encryption key.

To make encryption key available for every round, this work used a key expansion algorithm for

expanding the 16 bytes key. Figure 3.12 shows the flow chart for the key expansion process. At first the 16 bytes of the key vector were rearranged (row-wise) into a 4 × 4 initial key matrix. New rows were created by an element-wise-XOR operation of the direct predecessor row with the fourth row before it.

Additionally, for every fourth row, the predecessor row was rotated and substituted before applying the XOR operation. To expand the encryption key to 44 rows, the key expansion process ran iteratively for 40 rounds(Cheng & Moore, 2013). Figure 3.13 shows the snippet for the key expansion model. The complete program is shown in Appendix H

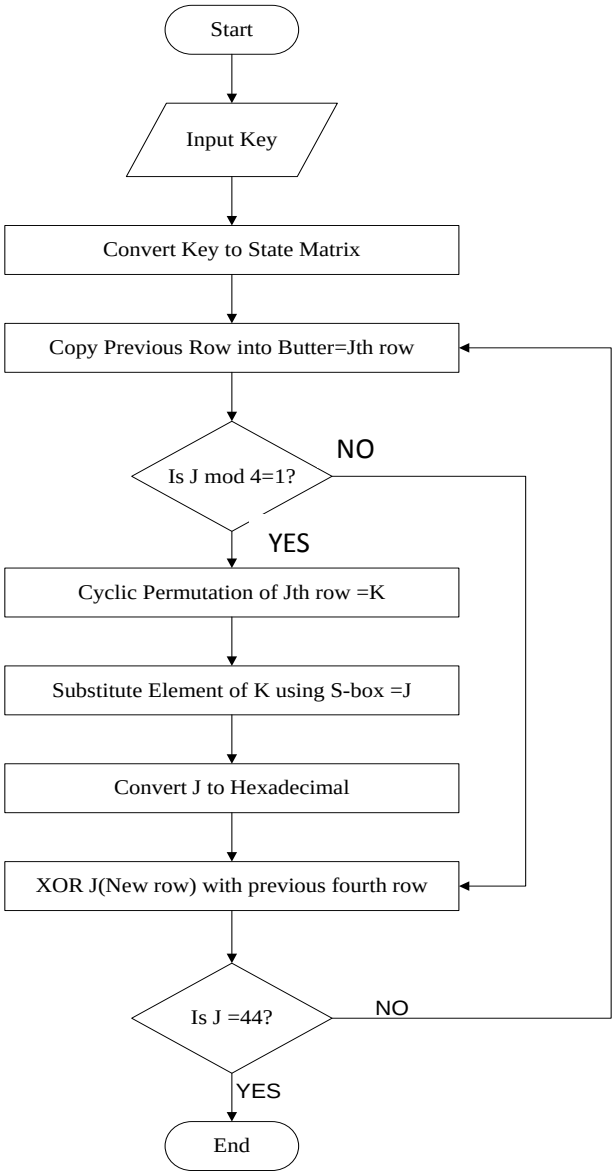


Figure 3.12 Key Expansion Algorithm (Kazlauskas & Kazlauskas, 2009)

```
function w = key_expansion (key, s_box, rcon, vargin)
if nargin > 3
else
    verbose_mode = 0;

end
if iscell (key) | prod (size (key)) ~= 16
    error ('Key has to be a vector (not a cell array) with 16 elements.')
```

Figure 3.13:Snippet for Key Expansion Model

3.2.8 Calculating Packet Delay

In other to properly evaluate performance, this work calculated the transmission and propagation time for packets to get to the receiving node from the sending node using packet delay model as described in section 2.2.10. The Poisson traffic model as explain in section 2.2.11 with mean rate equal to 2msec was used to obtain the processing time of nodes.

3.2.9 Calculating the Payload Size

In other to determine the payload size of packets, the Poisson traffic modelwith mean rate equal to 230byte, as describe in section 2.2.11, was used.

3.2.10 Brute Force Attack Model

In brute force attack, the attacker tries several CRC combinations by sending fake packet with the aim of changing the context table of the decompressor. In this research work the brute force attack was modeled using the model as describe in section 2.2.12. Figure 3.14 shows the snippet for brute-force attack. The complete program is shown in Appendix A.

```
for q= 2:pack
    pa(q)=pa(q-1)+ packet(q);
    pb(q)=pb(q-1)+ AC(q);
      t(q)=t(q-1)+ p(q);
      th(q)=pa(q)/t(q);
      tg(q)=pb(q)/t(q);
      %claculate packet loss
                if generate==round(rand(1,8))
                    Lo=Lo+1;
                    Lost=Lost+1;
                    if Lost>interval
                         Lost=Lost-1;
                    end

                end
end
```

Figure 3.14:Snippet for Brute force attack

3.2.11 Calculating Burst Loss and Total Transmitted Byte

In other to evaluate the performance of the model, the burst loss due to packet loss was obtained using the mathematical expression of equation (2.11). The total number of transmitted byte due burst loss was also obtained using the mathematical expression of equation 2.12.

3.3 Development of an Improved Intrusion Detection Based Secured Robust Header Compression Technique.

For the improved intrusion detection based secured robust header compression, all the steps of the first methodology were repeated. Figure 3.15shows the flow chart for the improved intrusion detection based secured robust header compression. To detect the malicious node the selective watchdog intrusion detection technique as described in section 2.2.15 was used. Trail packets were logged and partitioned by destination node. The watchdog intrusion detection system counted the number of distinct CRC of trial packets with the same source node address. It then dropped packets from the nodes with the count greater than interpretation interval.

$$Link = \begin{pmatrix} 0 & \text{if T>W} \\ 1 & \text{if T<W} \end{pmatrix} \qquad (3.3)$$

where:

T is the number of trial packets with distinct CRC

W is the interpretation interval

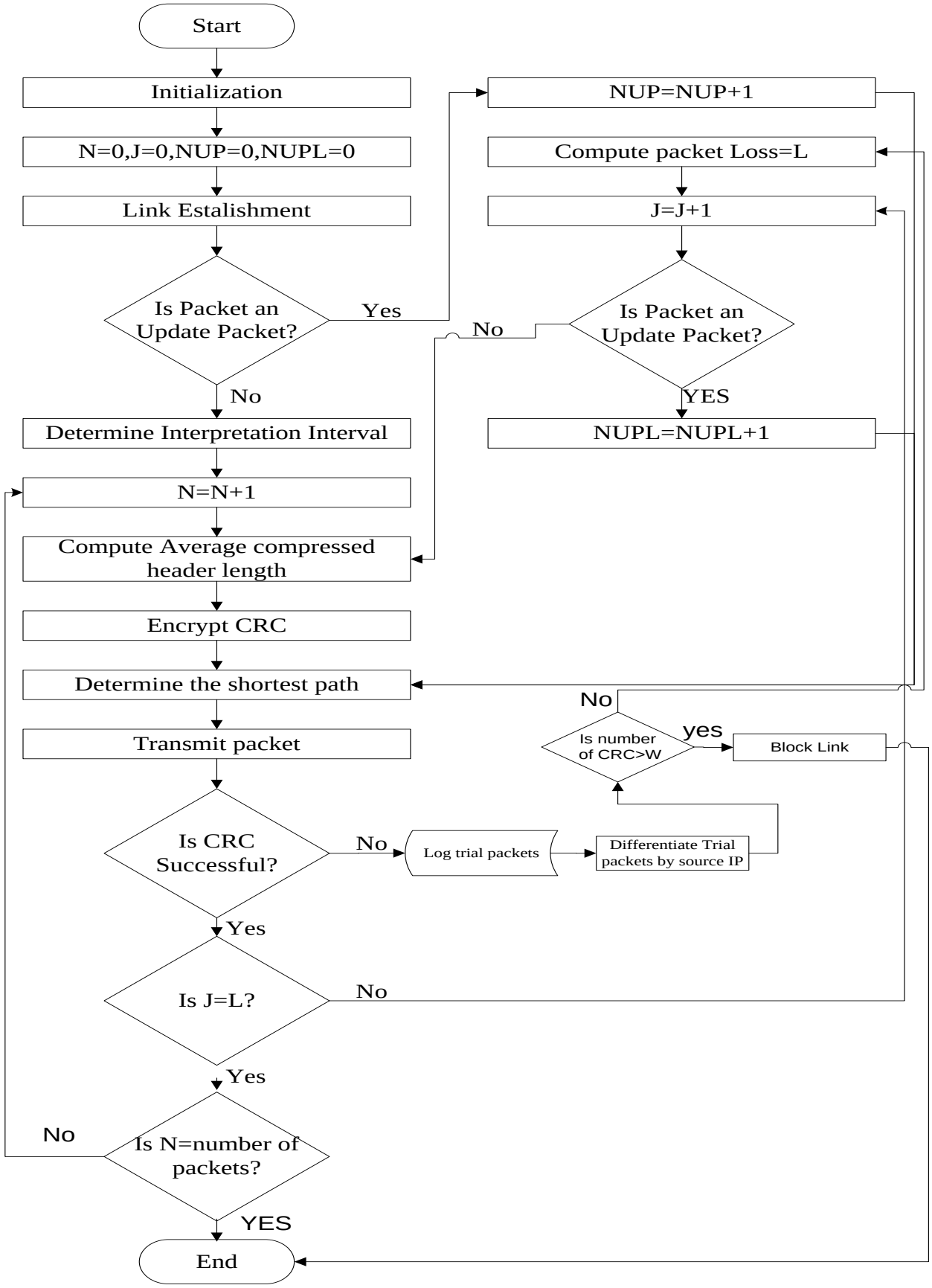


Figure 3.15: Flowchart for implementation of the Improved Intrusion Detection Based secured ROHC

3.3.1 Development of MATLAB GUI for improved intrusion detection based secured robust header compression.

To aid in easy development and interactivity of the improved intrusion detection based secured robust header compression technique, a MATLAB GUI was developed. Figure 3.16 shows the developed MATLAB GUI for the improved intrusion detection based secured robust header compression technique. Figure 3.17 shows a snippet of the GUI. The MATLAB functions (Script) that ran the developed simulator (GUI) is shown in Appendix J.

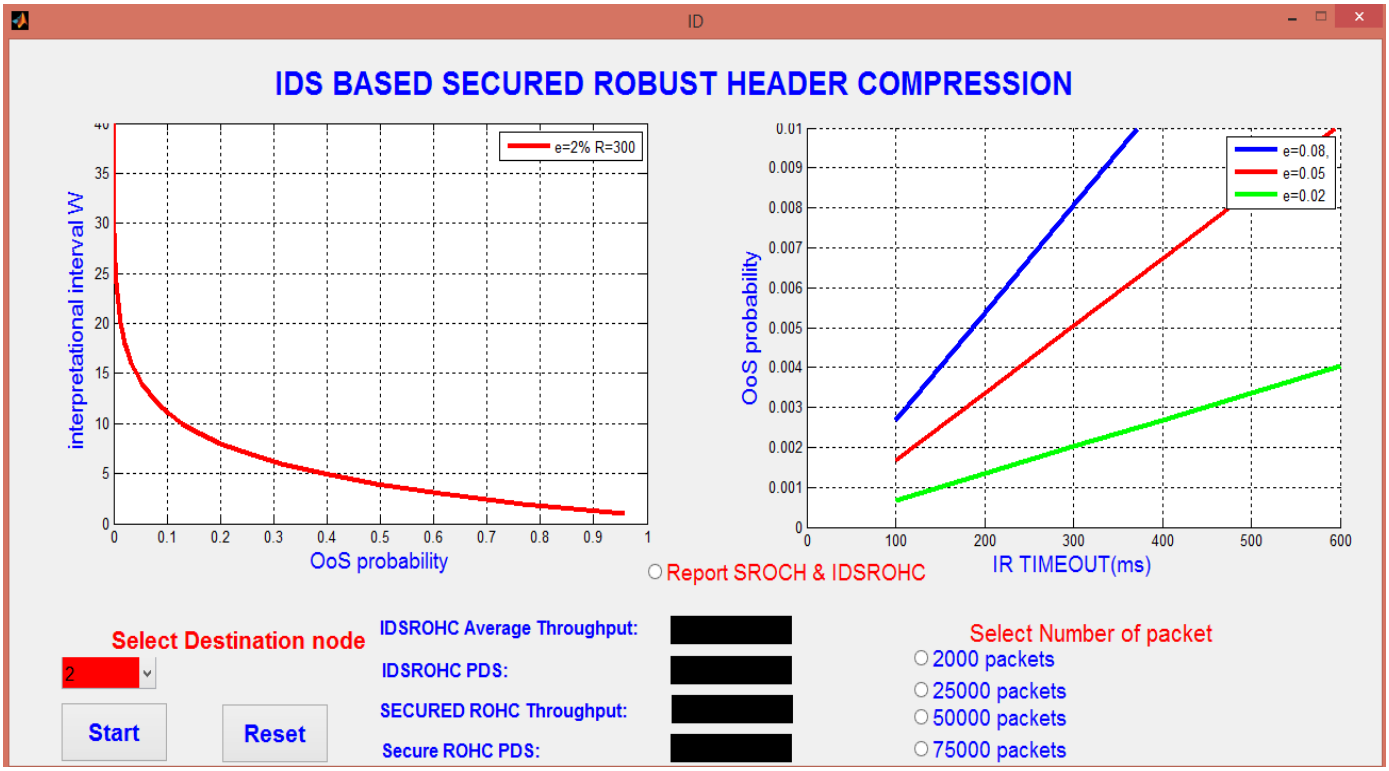


Figure 3.16 Developed Simulator for Improved Intrusion Detection based Secured ROHC.

```
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @ID_OpeningFcn, ...
                   'gui_OutputFcn',  @ID_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before ID is made visible.
function ID_OpeningFcn(hObject, eventdata, handles, varargin)
            nr=5;
            s=300000000;
            link=2000;%kilobyte per seconds
            L=5;% Burst length LB
            e=0.02;% average error probability
            R=300;
            L1=3;
```

Figure 3.17: Snippet of GUI Implementation

3.4 Performance Evaluation and Simulation Parameters

In other to evaluate performance, delivery success and throughput were used as the two key metrics. The mathematical expressions used were also described in equation (2.22) and (2.24).

The list of parameters required for the development of the simulator needed to simulate the secured and improved intrusion detection based secured robust header compression are listed in Table 3.1.

Table 3.1 Simulation Parameters

PARAMETER	VALUE
Simulator	MATLAB.
Studied protocol	IPv6.
Area	100 by 100 meters
Total number of nodes	40 nodes.
Routing Algorithm	Bellman-ford.
Link Bandwidth	200Kbps
Number of packets (M)	75000 packets
Uncompressed packet header size	60 byte
Mobility Model	Random way point.
Confidence variable	4
Keylength	16 byte
Block length	16byte
Burst length L_B	5
average error probability (Ɛ)	2%
Traffic Model	Poisson traffic Model
Length of update packet	60 byte
Initial Refresh Time out	300 seconds
Payload size Lamda	230 byte
Processing time Lamda	2ms

CHAPTER FOUR
RESULTS AND DISCUSSIONS

4.1 Introduction

In this section the performance of the secured ROHC and that of IDSROHCtechniques werecompared on the basis of throughput and packet delivery success. This is to determine by how much the improved model outperformed the existing model.

4.2 Simulation

The modeled Secured ROHC and parameters are selected based on the work of (Cheng & Moore, 2013).Figure 4.1 to Figure 4.3and Figure 4.5 to Figure 4.6 reports the results of IDSROHC while Figure 4.7 and Figure 4.8 compares the result of IDSRROHC and Secured ROHC.

4.3 Results of the Effect of Out of Synchronization Probability on Required Interpretation Interval

Figure 4.1 shows effect of Out of Synchronization Probability (OoS) on interpretation interval as discussed in section 3.2.3.

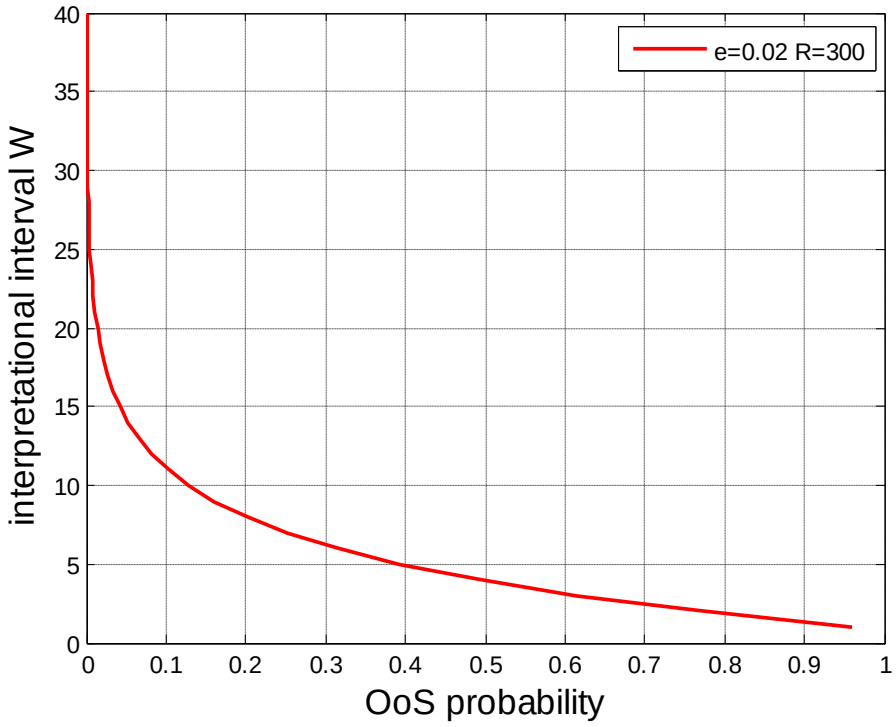


Figure 4.1: Interpretation Interval Against Out of Synchronization Probability

From Figure 4.1, it is observed that as the interpretation interval (window based least significant bits) decreases, the OoS increases. This implies that increase in the number of significant bit required for error correction decreases the OoS and decrease in the number of significant bit required for error correction also increases the OoS. The result of Figure 4.1 also shows that with burst length of 5 and average error probability of 2%, the OoS remains constantly at zero with increasing interpretation interval above 25. This impliesthat interpretation interval of 25 is needed to achieve the best compression efficiency and with high robustness. A burst length of 5 and average error probability of 2% are standard values for stable mobility(Hermenier *et al.*, 2013).

4.4 Results of the Effect of Average Error Probability on Interpretation Interval

Figure 4.2 shows a graph of interpretation interval against OoS as discussed in section 3.2.3.

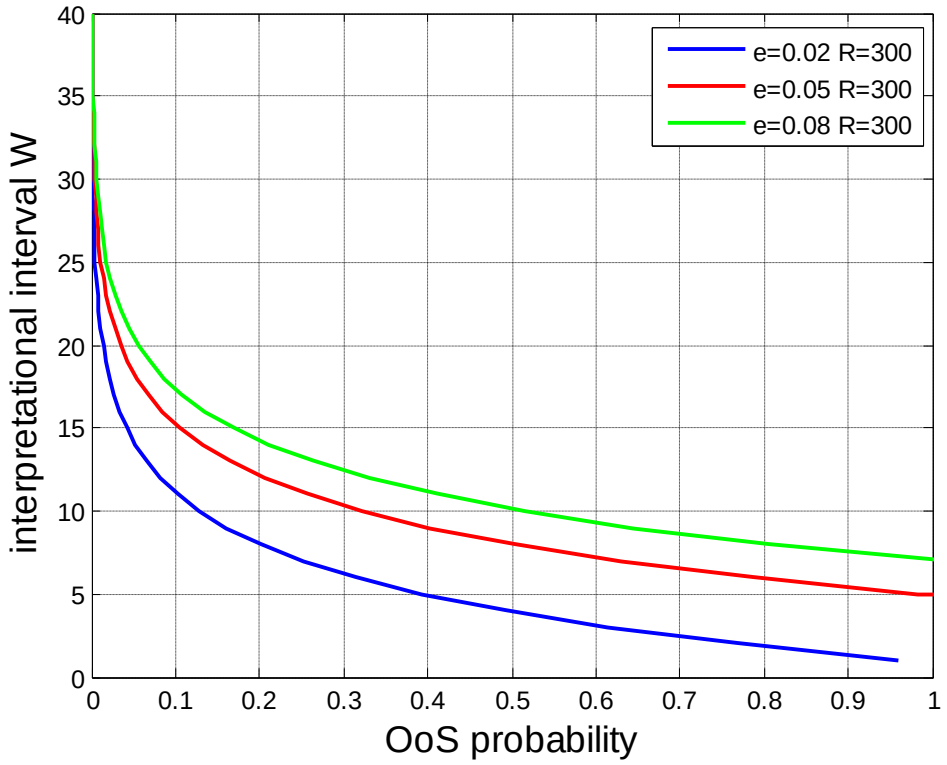


Figure 4.2: Interpretation Interval against Out of Synchronization Probability

From Figure 4.2, it is observed that for average error probability of 2%, a required interpretation interval of 25 is obtained. For average error probability of 5%, a required interpretation interval of 28 is obtained. For average error probability of 8%, a required interpretation interval of 30 is obtained. This implies that as the average error probability increases, the interpretation interval (window based least significant bits) needed to correct the error also increases. A burst length of 5 and average error probability of 2% are standard value for stable mobility(Hermenier *et al.*, 2013).

4.5Result of Relationship Between Out of Synchronization Probability and the Initial Refresh Timeout

Figure 4.3 shows the graph of OoS against initial refresh timeout for average error probabilityof 2%, 5% and 8% as discussed in section 3.2.3.

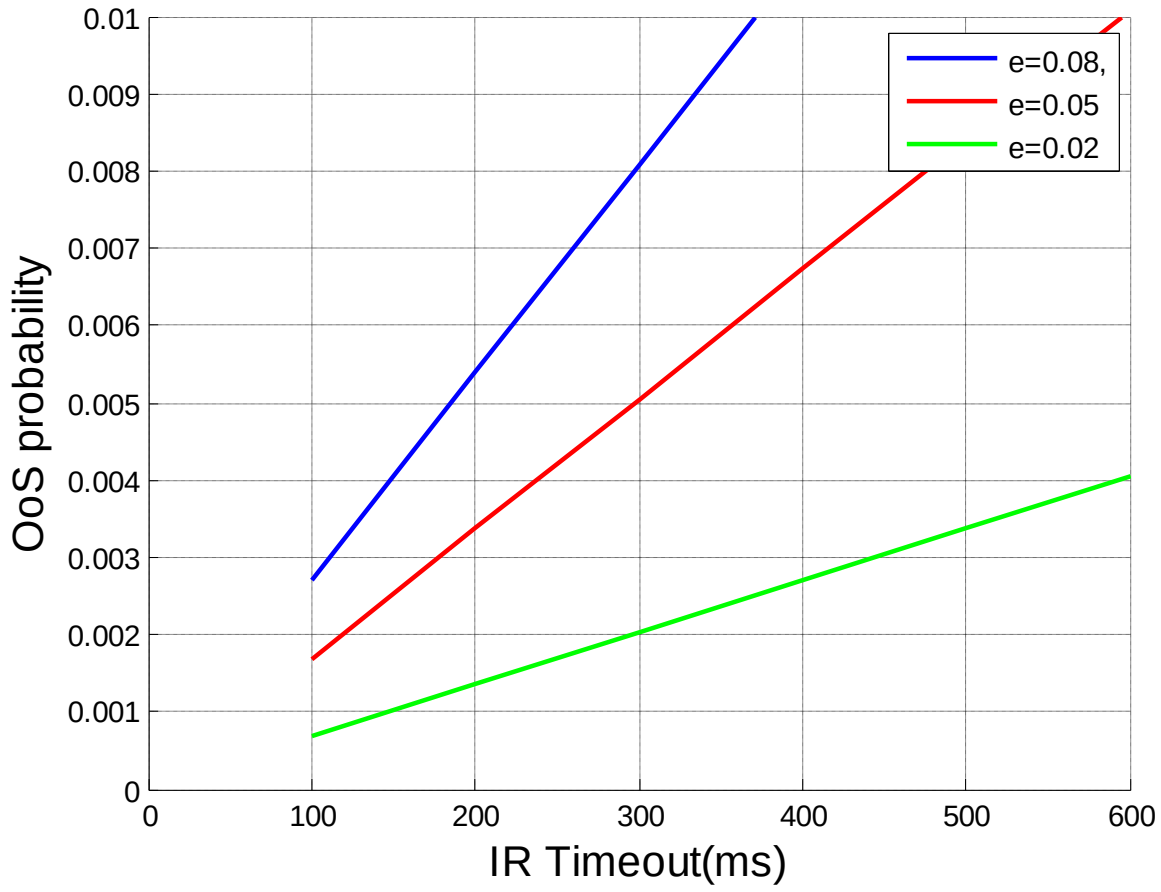


Figure 4.3: Out of Synchronization Probability Against Initial Refresh Time out

It is observed that with increase in the average error probability, the out of synchronization probability also increases. This implies that increase in average error probability increases the probability of losing synchronization between the compressor and decompressor. Initial refresh timeout ensure context synchronization between the compressor and decompressor. A low initial refresh timeout leads to increase in robustness due to frequent updating of context table. Figure 4.3 shows that decrease in initial refresh timeout increases the out of synchronization probability.

Initial refresh timeout which result to out of synchronization of 10% of average error probability is recommended to be used for design of Robust header compression system(Hermenier *et al.*, 2013). In this work an initial refresh timeout of 300ms and average error probability of 2% were used because of the standard (Hermenier *et al.*, 2013).

4.6 Results of Throughput for ROHC and Uncompressed Traffic

Figure 4.4 shows the throughput obtained for ROHC and Uncompressed traffic. The parameters used is discusses in Table 3.1.

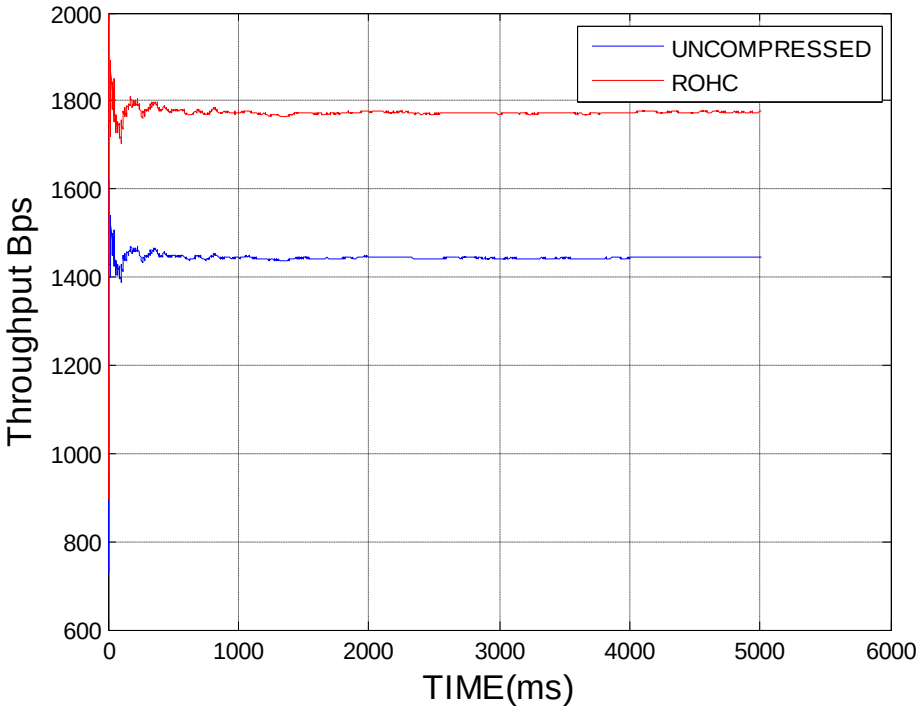


Figure 4.4: Throughput of ROHC and Uncompressed Traffic

Figure 4.4 shows a graph of throughput against Total packet delay for packet compressed using ROHC and that of uncompressed packet. The result generated show that the average throughput when packets are compressed using ROHC is 1773 bit per seconds while the result when packets are sent uncompressed is 1444 bit per seconds.This implies that applying ROHC for header

compression increases the network throughput by 22.8%.The 22.8% increase in network throughput is gotten using equation 4.1.

$$\text{Percentage improvement} = \frac{ROHC - UNCOMPRESSED}{UNCOMPRESSED} * 100 \quad (4.1)$$

4.7 Results of Improved Intrusion Detection Based Secured RObust Header Compression (IDSROHC)

Figures 4.5 and 4.6 show the result obtained when IDSROHC is used to secure packets against brute force attack as discussed in section 3.3. The parameters used are listed in Table 3.1.

Figure 4.5 shows a graph of packet delivery success against total packet delay when IDSROHC is use to protect the network against brute force attack as discussed in section 3.3.

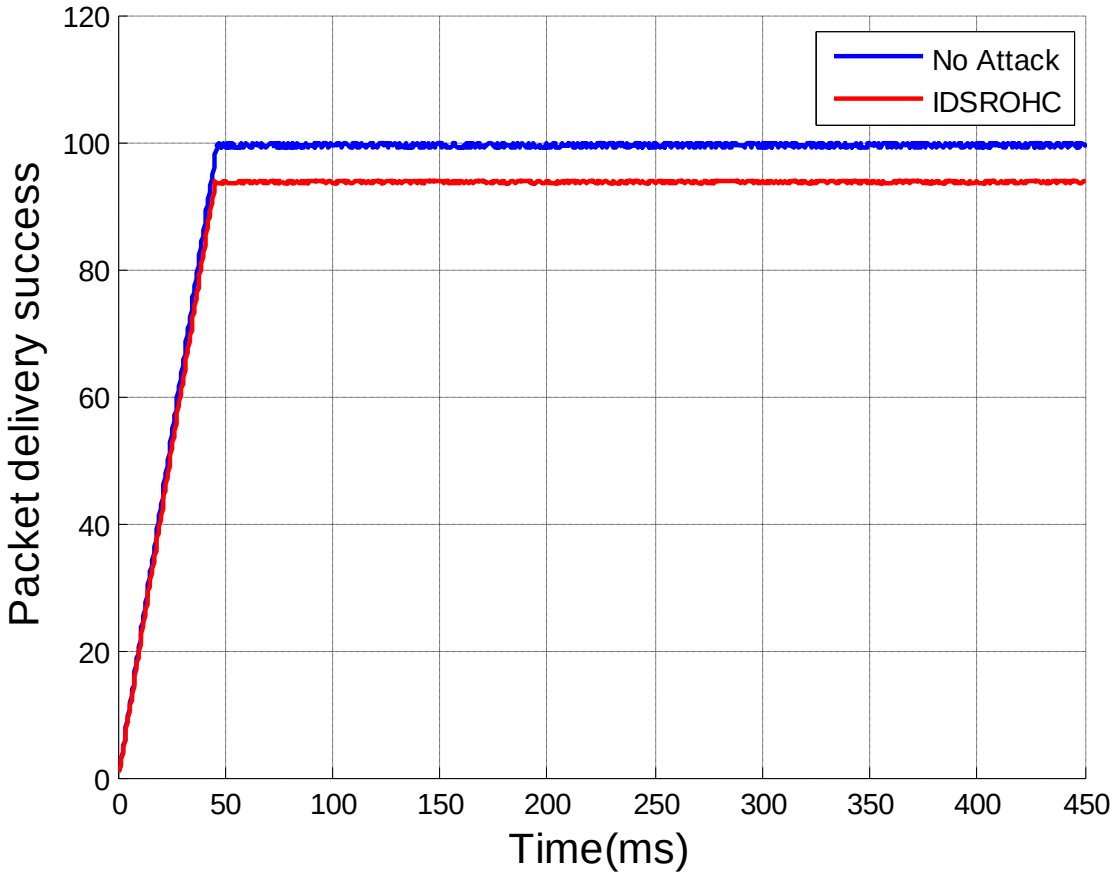


Figure 4.5 Effect of Brute-Force Attack on Packet Delivery Success for IDSROHC Model

Figure 4.5reveals the effect of brute-force attack on packet delivery success when IDSROHC is use to protect the network against brute force attack.From Figure 4.5 it is seen that the packet delivery success increaseswith increasing total packet delay. At 48ms total packet delay, the packet delivery success maintains a slightly stable graph. This indicate the time it takes for the compressor and decompressor to synchronize their context table. With the average packet delivery success calculated using equation2.24,result reveals the packet delivery success obtained when no attack is carried out against the network. From Figure 4.5, it is also observed that in the absence of brute-force attack, an average packet delivery success of 100% is obtained. With brute-force attack, the packet delivery success drop to 93%. This implies that 7% of packets are lost when IDSROHC is use to secure network against brute-force attack. The 7% percent value is gotten by subtracting 93% gotten for IDSROHC from 100% for IDSROHC with no attack.

Figure 4.6 shows a graph of throughput against total packet delay when IDSROHC is used to protect the network against brute force attack as discussed in section 3.3.

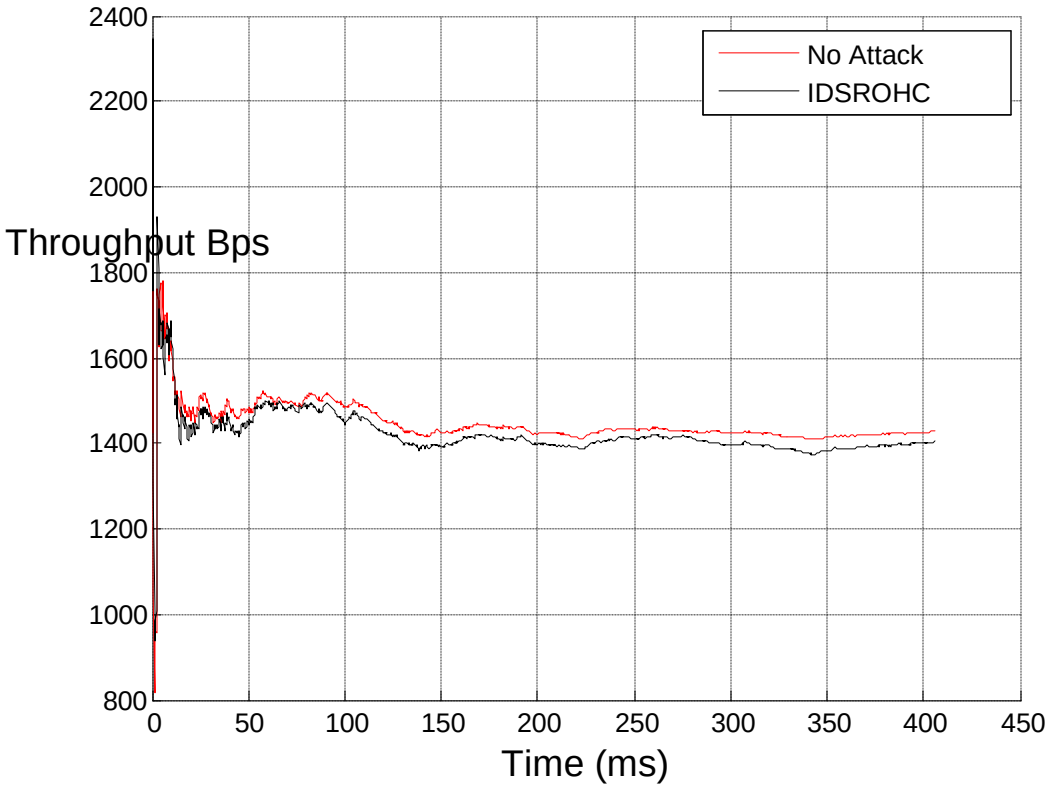
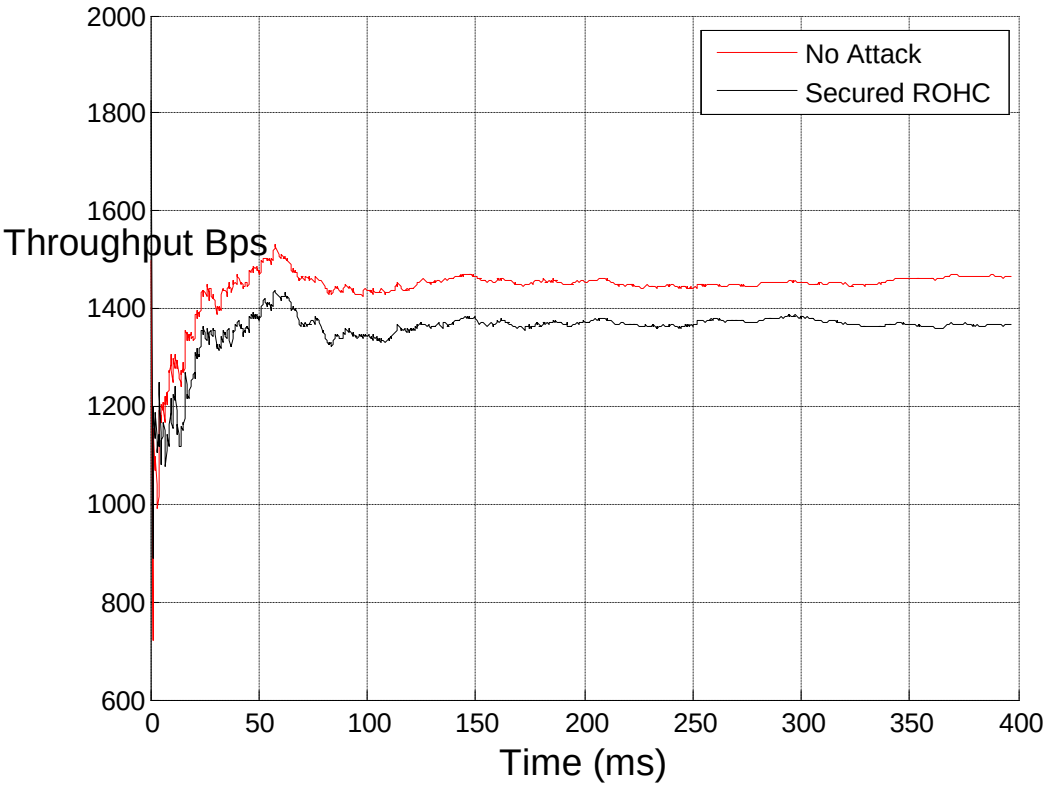


Figure 4.6 Effect of Brute-Force Attack on the Network Throughput

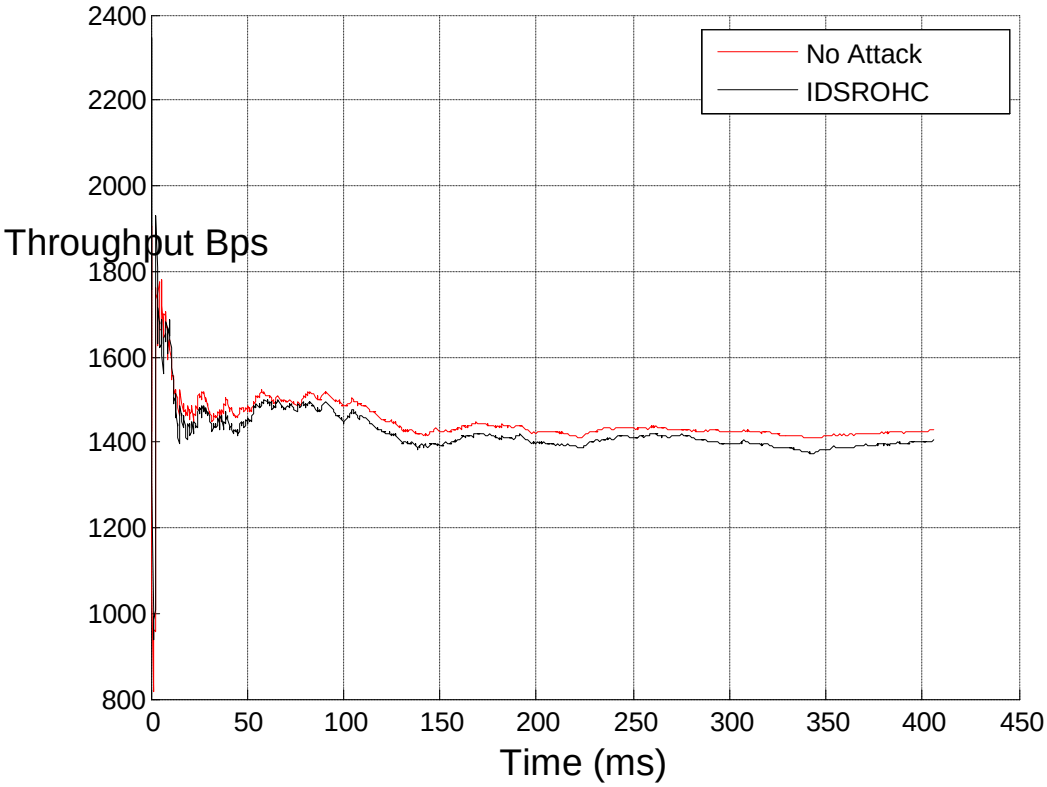
Figure 4.6reveals the effect of brute-force attack on throughput when IDSROHC is use to protect the network against brute force attack. From Figure 4.6 theresults for throughput obtained in the absence of brute force attack is evident. From Figure 4.6, it is also observed that in the absence of brute force attack, an average throughput of 1444bps was obtained. With the introduction of brute force attack, the average throughput drop to 1424bps. 1.40% reduction is obtained by using equation 4.1 .This implies 1.40% reduction in network throughput when IDSROHC is used to secure packets against brute force attack.

4.8Comparison of IDSROHC and Secured ROHC Performances using Packet Delivery Success and Throughput.

Figure 4.7 and 4.8 shows the comparison in terms of packet delivery success and throughput between Secured ROHC and IDSROHC as discussed in section 3.4. Figure 4.7a shows the throughput results for Secured ROHC while Figure 4.7b shows that of IDSROHC.



(a) Throughput of Secured ROHC results;



(b) Throughput of IDSROHC Results

Figure 4.7: Comparing the result in terms of throughput between Secures ROHC and IDSROHC

Figure 4.7 shows the comparison of throughput for Secured ROHC and IDSROHC as discussed in section 3.4. The Secured ROHC and IDSROHC were subjected to brute attack. From the result of Figure 4.7, average throughput for network without attack is obtained to be 1444 bit per seconds. The average throughput for Secured ROHC under brute force is obtained to be 1367 bit per seconds while that of IDSROHC is obtained to be 1424 bit per seconds. The percentage improvement in average throughput of IDSROHC over Secured ROHC is calculated to be 4.97%. This implies that with IDSROHC outperformed Secured ROHC in average throughput improved by 4.97%. The percentage improvement is calculated using equation(4.1).

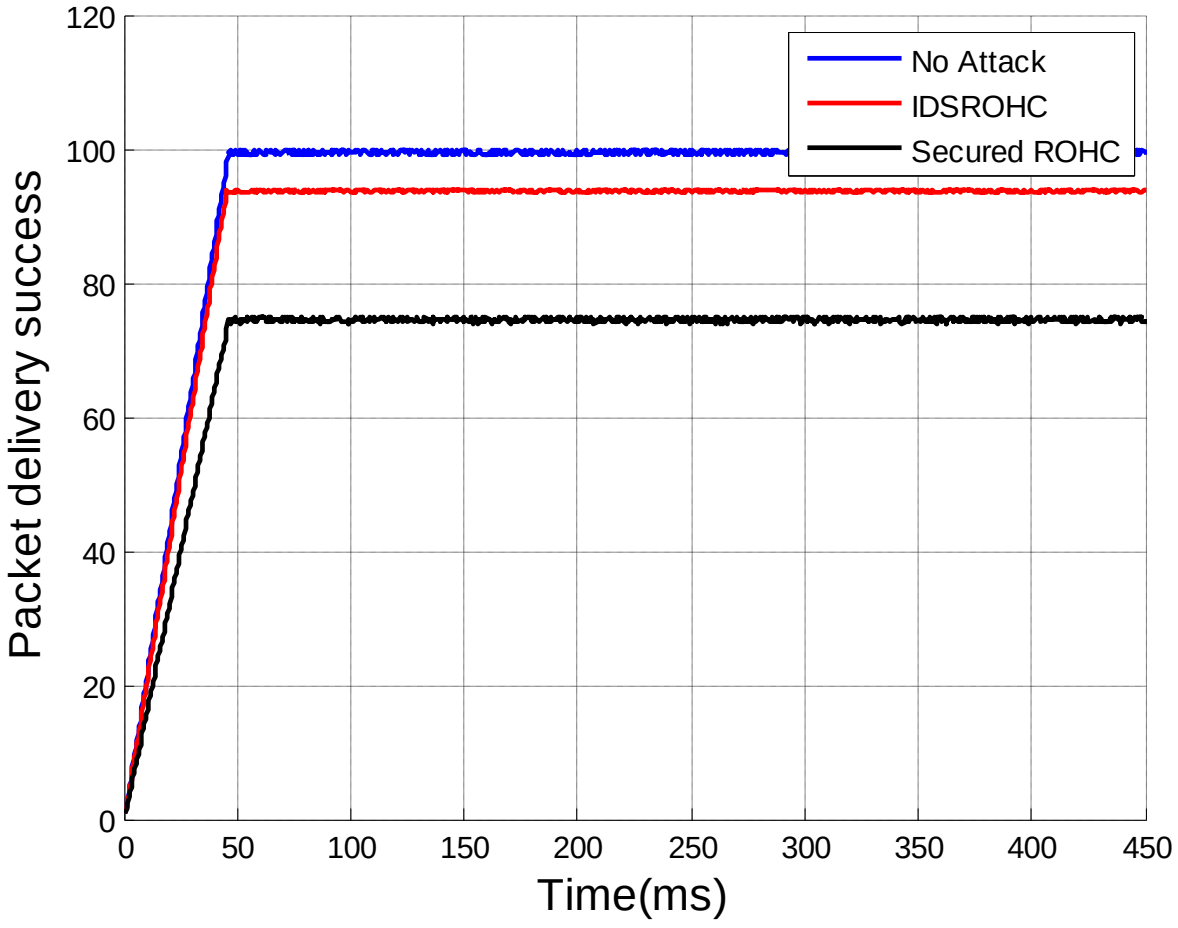


Figure 4.8: Comparing Packet Delivery Success of Secured ROHC and IDSROHC

Figure 4.8 shows the result for the comparison in terms of packet delivery success between Secured ROHC and IDSROHC. From Figure 4.8, It is observed that when Secured ROHC is use to protect the packet against brute force attack, an average packet delivery success of 72%is obtained. This means that 72% percent of packet sent by source was successfully delivered to the destination node. With IDSROHC use to protect packet against brute force attack, the work obtain 93% average packet delivery success. This implies that 93% of packet sent by source node was successfully delivered to the destination node. The percentage improvement between IDSROHC and ROHC is calculated to be 29% using equation(4.1). Therefore the developed IDSROHC showed 29% improvement in average delivery success over that of Secured ROHC. Table 4.1 shows the summary of result obtained and the percentage improvement when Secured ROHC is compared against IDSROHC.

Table 4.1: Performance Evaluation of IDSROHC

	Secured ROHC	**IDSROHC**	**Percentage Improvement**
Average throughput(Bps)	1367	1424	4.97%
Packet delivery success(%)	72	93	29%

CHAPTER FIVE

CONCLUSION AND RECOMMENDATION

5.1 Summary

This research hasdeveloped an improved Intrusion Detection based Secured RObust Header Compression (IDSROHC) using watchdog technique. The developed IDSROHC is aimed at mitigating brute force attack in a robust header compression network. The developed IDSROHC is GUI based and evaluated using packet delivery success and average network throughput. Conclusion, recommendation and limitation encountered during the course of this research are presented. Areas of future research have also been suggested.

5.2 Conclusion

Secured ROHC using processes such as random waypoint model, Bellman-ford calculation of shortest path, ROHC model, traffic model, packet encryption, calculation of cyclic redundancy check and brute force attack model. In other to mitigate the challenge (brute force attack) associated with Secured ROHC, an improved Intrusion Detection based Secured RObust Header Compression (IDSROHC) technique has been developed using watchdog based intrusion detection system. This was developed on a MATLAB graphical user interface platform. The result obtained show that when seventy five thousand packets was transmitted from the source node to destination node with IDSROHC use against brute force attack , throughput and packet delivery success improvement of 4.97% and 29% respectively was recorded over Secured ROHC .

5.3 Significant Contributions

A lot of Researches has been done on compressing IPv6 header using Robust Header Compression. Research has also been done on Securing RObust Header Compression. Significant contributions of this research work are as follows:

1. Development of a graphical user interface (GUI) improved Intrusion Detection based Secured RObust Header Compression technique (IDSROHC). This has added ability of secured RObust Header Compression (ROHC) to detect brute force attack link by using watchdog base intrusion detection system.
2. The IDSROHC was validated using two performance metrics (Packet delivery success and throughput). IDSROHC outperformed Secured ROHC by 4.97% throughput and 29% packet delivery success.

5.4 Limitations

The limitation of this research work is:

1. Queueing Delay and Network Congestion was not considered

5.5 Recommendations for further work

The following possible areas of further work are recommended for consideration for future research:

1. Further study can be done to Implement these security features into a kernel implementation of ROHC and evaluating in an emulation environment with real systems.
2. The algorithm can be modified to increase the encryption strength by applying randomize encryption to CRC

REFERENCES

AB, E. (2004). The concept of robust header compression, ROHC. Retrieved 8 march 2016, from http://www.effnet.com/pdf/uk/Whitepaper_Robust_Header_Compression.pdf

Al-Roubaiey, A., Sheltami, T., Mahmoud, A., Shakshuki, E., & Mouftah, H. (2010). AACK: adaptive acknowledgment intrusion detection for MANET with node detection enhancement. *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, 634-640.

Ali, A. N. A. (2012). Comparison study between IPV4 & IPV6. *International Journal of Computer Science Issues, 9*(3), 314-317.

Ang, W.-C., Wan, T.-C., Kataoka, K., & Teh, C.-H. (2008). Performance Evaluation of Robust Header Compression (ROHC) over Unidirectional Links using DVB- S Testbed. *Keio SFC Journal, 8*(2), 21-36.

Azhagiri, M., Rajesh, A., & Karthik, S. (2015). Intrusion Detection and Prevention System: Technologies and Challenges. 384-395.

Batabyal, S., & Bhaumik, P. (2015). Mobility models, traces and impact of mobility on opportunistic routing algorithms: A survey. *IEEE Communications Surveys & Tutorials, 17*(3), 1679-1707.

Beijnum, I. v. (2006). *Running IPv6* (J. Sumser Ed.). New york,United States: Springer publishers, 2.

Bow-Nan, C., John, Z., Jim, W., Scott, M., & Brian, H. (2013). MANET IP Header Compression. *MILCOM 2013-2013 IEEE Military Communicatinos Conference*, 494-503. doi: 10.1109/milcom.2013.91

Butun, I., Morgera, S. D., & Sankar, R. (2014). A survey of intrusion detection systems in wireless sensor networks. *IEEE Communications Surveys & Tutorials, 16*(1), 266-282.

Can, O., & Sahingoz, O. K. (2015). A survey of intrusion detection systems in wireless sensor networks. *6th International Conference on Modeling, Simulation, and Applied Optimization (ICMSAO), 2015*, 1-6.

Casoni, M., Grazia, C. A., Klapez, M., & Patriciello, N. (2017). How to avoid TCP Congestion without dropping Packets: an Effective AQM called PINK. *Computer Communications*.

Cha, H., Shon, T., Kim, K., & Hong, M. (2015). *Improving packet header compression with adaptive sliding window size.* Paper presented at the 2015 International Conference on Information Networking (ICOIN), 541-543.

Chauhan, D., & Sharma, S. (2015a). Addressing the bandwidth issue in end-to-to header compression over ipv6 tunneling mechanism. *I.J.Computer and information security, 9*, 39-45.

Chauhan, D., & Sharma, S. (2015b). Network Optimization of IPv6 Networks Using Tunnel Header Compression. *IJRET: International Journal of Research in Engineering and Technology, 4*(03), 113-116.

Chen, X., Guo, F., Dang, P., & Wu, L. (2012). *A survey of ROHC header compression schemes.* Paper presented at the 2012 2nd International Conference on Computer Science and Network Technology (ICCSNT), 331-335.

Cheng, B.-N., & Moore, S. (2013). Securing RObust Header Compression (ROHC). *IEEE Military Communications Conference, MILCOM 2013-2013*, 1383-1390.

Cheng, B.-N., Wheeler, J., & Hung, B. (2013). Internet protocol header compression technology and its applicability on the tactical edge. *Communications Magazine, IEEE, 51*(10), 58-65.

Cheng, B.-N., Wong, T., & Hung, B. (2014). A Linux Kernel Implementation of MANET IP Header Compression. *IEEE Military Communications Conference (MILCOM), 2014*, 1477-1483.

Chishti, M. A., & Mir, A. H. (2015). Survey of Header Compression Techniques over Multiprotocol Label Switching (MPLS). *Int. J. Com. Dig. Sys, 4*(2), 121-136.

Cho, C. Y., Seah, W. K. G., & Chew, Y. H. (2006). A framework and source model for design and evaluation of Robust Header Compression. *Computer Networks, 50*(15), 2676-2712.

Christian, C., & Lecroq, T. (2004). *Handbook of exact string matching algorithms*. King's College. : Citeseer, 1-17.

Davies, J. (2012). *UNDERSTANDING IPV6* (H. Bauer Ed. third edition ed.). United States of America: Octal Publishing, Inc. united states, 1-641.

Debar, H., Dacier, M., & Wespi, A. (2000). *A revised taxonomy for intrusion-detection systems.* Paper presented at the Annales des télécommunications, 361-378.

Dhenakaran, D. S., & Parvathavarthini, A. (2013). An overview of routing protocols in mobile ad-hoc network. *International Journal of Advanced Research in Computer Science and Software Engineering, 3*(2).

Dua, D., & Mishra, A. (2014). Selective Watchdog Technique for Intrusion Detection in Mobile Ad-Hoc Network. *International Journal on Applications of Graph Theory in Wireless Ad hoc Networks and Sensor Networks (GRAPH-HOC), 6*, 1-12.

Dubey, S., & Tripathi, N. (2015). A Survey on Intrusion Detection Systems. *International Journal of Scientific Research in Science, Engineering and Technology, 1*(6), 29-40.

El-bakry, H. M., & Mastorakis, N. (2008). A Real-Time Intrusion Detection Algorithm for Network Security *WSEAS TRANSACTIONS on COMMUNICATIONS, 7*(12), 1222-1234.

Esfandi, A. (2010). *Efficient anomaly intrusion detection system in adhoc networks by mobile agents.* Paper presented at the 2010 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT), , 73-77.

Gavaskar, S., Ramaraj, E., & Surendiran, R. (2012). A Compressed Anti IP Spoofing Mechanism using Cryptography. *International Journal of Computer Science and Network Security, 12*(11), 137-140.

Hermenier, R., Rossetto, F., & Berioli, M. (2011). *A simple analytical model for RObust Header Compression in correlated wireless links.* Paper presented at the 8th International Symposium on Wireless Communication Systems (ISWCS), 2011 634-638.

Hermenier, R., Rossetto, F., & Berioli, M. (2013). On the Behavior of RObust Header Compression U-mode in Channels with Memory. *IEEE Transactions on Wireless Communications, 12*(8), 3722-3732.

Huang, K.-T., Lin, Y.-N., & Chiu, J.-H. (2013). Real-time mode hopping of block cipher algorithms for mobile streaming. *International Journal of Wireless & Mobile Networks, 5*(2), 127-142.

Indu, I., & Manu, T. (2012). Cyclic redundancy check generation using multiple lookup table algorithms. *International Journal of Modern Engineering Research (IJMER), 2*(4), 2445-2451.

Jivorasetkul, S., Shimamura, M., & Iida, K. (2012). *End-to-end header compression over software-defined networks: A low latency network architecture.* Paper presented at the 4th International Conference on Intelligent Networking and Collaborative Systems (INCoS) 493-494.

Jivorasetkul, S., Shimamura, M., & Iida, K. (2013). *Better network latency with end-to-end header compression in SDN architecture.* Paper presented at the 2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM), 183-188.

Jonsson, L., & West, M. (2007). Network Working Group G. Pelletier Request for Comments: 4996 K. Sandlund Category: Standards Track Ericsson. 1-40.

Jyothsna, V., Prasad, V. R., & Prasad, K. M. (2011). A review of anomaly based intrusion detection systems. *International Journal of Computer Applications, 28*(7), 26-35.

Kang, N., Shakshuki, E. M., & Sheltami, T. R. (2011). *Detecting forged acknowledgements in MANETs.* Paper presented at the 2011 IEEE International Conference on Advanced Information Networking and Applications (AINA), 488-494.

Kaur, J., Singh, R., & Kaur, P. (2015). Prevention of DDoS and Brute Force Attacks on Web Log Files using Combination of Genetic Algorithm and Feed forward Back Propagation Neural Network. *International Journal of Computer Applications, 120*(23).

Kazlauskas, K., & Kazlauskas, J. (2009). Key-dependent S-box generation in AES block cipher system. *Informatica, 20*(1), 23-34.

Kim, I. S., & Kim, M. H. (2012). Agent-based honeynet framework for protecting servers in campus networks. *The Institution of Engineering and Technology, 6*(3), pp. 202–211.

Koskela, P., Majanen, M., & Valta, M. (2016). Packet Header Compression for the Internet of Things. *Sensors & Transducers, 196*(1), 43.

Kowalk, W. (2006). CRC Cyclic Redundancy Check Analysing and Correcting Errors. from http://lidecc.cs.uns.edu.ar/~ldm/data/oc/info/crc-bitfiltereng.pdf

Levinti, A. (2012). *Introduction to the Design and Analysis of Algorithms*. New Jersey Pearson Education,Inc.,publishing, 1-593.

Li, R., Cao, Y., Zhou, Z., Liao, J., & Mu, G. (2013). Method and device for decoding by using window-based least significant bits in robust header compression *U.S. Patent No. 8,418,037. Washington, DC: U.S. Patent and Trademark Office.*

Liao, H.-J., Lin, C.-H. R., Lin, Y.-C., & Tung, K.-Y. (2013). Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications, 36*(1), 16-24.

Limaye, A., Kakne, S., Tiple, P., & Bhamare, S. (2014). Scrutinizing the Art of Intrusion Detection. *International Journal of Engineering and Advanced Technology,* 356-361.

Madsen, T. K., Zhang, Q., Fitzek, F. H., & Katz, M. (2007). *Design and evaluation of ip header compression for cellular-controlled p2p networks.* Paper presented at the IEEE International Conference on Communications, 2007. ICC'07., 4046-4051.

Majanen, M., Koskela, P., & Valta, M. (2015). *ConstrainedApplicationProtocolProfileforRobustHeaderCompressionFramework*. Paper presented at the The Fifth International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies. 47-53

Marti, S., Giuli, T. J., Lai, K., & Baker, M. (2000). *Mitigating routing misbehavior in mobile ad hoc networks.* Paper presented at the Proceedings of the 6th annual international conference on Mobile computing and networking, 255-265.

Martin, K. M., Safavi-Naini, R., Wang, H., & Wild, P. R. (2005). Distributing the encryption and decryption of a block cipher. *Designs, Codes and Cryptography, 36*(3), 263-287.

Matias, Y., & Refua, R. (2005). *Delayed-dictionary compression for packet networks.* Paper presented at the Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies,INFOCOM 2005 . , 1443-1454.

Meenakshi. (2014). Impact of Network Size on Performance of Wireless Network Topology. *International Journal of Advance Research in Computer Science and Management Studies, 2*(9), 175-179.

Mun, Y., & Lee, H. K. (2005). *UNDERSTANDING IPv6*. United States of America: Springer Science+Business Media, In, 1-279.

Osunade, O. (2012). A Packet Routing Model for Computer Networks. *International Journal of Computer Network and Information Security, 4*(4), 13.

Pandey, S. (2008). Comparative Study on Text Pattern Matching for Heterogeneous System.

Parziale, L., Britt, D. T., Davis, C., Forrester, J., Liu, W., Matthews, C., & Rosselot, N. (2006). *TCP/IP Tutorial and Technical Overview* (Eigth Edition ed.). United State of America: International Business Machines Corporation

Perset, M. K. (2010). Internet addressing: measuring deployment of IPV6. Retrieved 20/08/2016, 2016, from https://www.oecd.org/internet/ieconomy/44953210.pdf

Pramanik, A., Choudhury, B., Choudhury, T. S., Arif, W., & Mehedi, J. (2015). *Simulative study of random waypoint mobility model for mobile ad hoc networks.* Paper presented at the Communication Technologies (GCCT), 2015 Global Conference on, 112-116.

Rawat, P., & Bonnin, J.-M. (2010). *Designing a header compression mechanism for efficient use of IP tunneling in wireless networks.* Paper presented at the 7th IEEE Consumer Communications and Networking Conference (CCNC), 1-5.

Salehin, K. M., Rojas-Cessa, R., Lin, C.-b., Dong, Z., & Kijkanjanarat, T. (2013). Scheme to Measure Packet Processing Time of a Remote Host through Estimation of End-Link Capacity. *IEEE TRANSACTIONS ON COMPUTERS, Vol. X*, pp 1-14.

Sandlund, K., Pelletier, G., & Jonsson, L. (2010). The RObust Header Compression (ROHC) Framework Network Working Group Request for Comments: 4996. 1-94

Shakshuki, E. M., Kang, N., & Sheltami, T. R. (2013). EAACK—a secure intrusion-detection system for MANETs. *IEEE Transactions on Industrial Electronics, 60*(3), 1089-1098.

Shanmugavadivu, R., & N.Nagarajan. (2010). NETWORK INTRUSION DETECTION SYSTEM USING FUZZY. *Indian Journal of Computer Science and Engineering (IJCSE), 2*(1), pp 101-111.

Sohraby, K., Wang, C., & Cansever, D. (2006). *Performance of Packet Header Compression And Context Replication in Wireless/Mobile Systems.* Paper presented at the Military Communications Conference, 2006. MILCOM 2006. IEEE, 1-7.

Srivastava, R., & Richhariya, V. (2013). Survey of current network intrusion detection techniques. *Journal of Information Engineering and Applications, 3*(6), 27-33.

Sun, T. L., Eng, T. C., & Ping, L. S. (2011). *Improving Quality-of-Service of Real-Time Applications over Bandwidth Limited Satellite Communication Networks via Compression*: INTECH Open Access Publisher, 55-80.

Tan, S., Yew, H. T., Arifianto, M., Saad, I., & Teo, K. T. K. (2012). *Queue management for network coding in ad hoc networks.* Paper presented at the IEE Third International Conference on Intelligent Systems, Modelling and Simulation (ISMS), 2012 657-662.

Tian1, A., & Fu1, C. (2013). The research of virtualization emulator based on open shortest path first *Applied Mechanics and Materials, 411-414*, 729-733.

Tomoskozi, M., Seeling, P., Ekler, P., & Fitzek, F. H. (2015). *Performance Evaluation and Implementation of IP and Robust Header Compression Schemes for TCP and UDP*

Traffic in the Wireless Context. Paper presented at the IEE 4th Eastern European Regional Conference on the Engineering of Computer Based Systems (ECBS-EERC), 2015 45-50.

Tong, Y., Li, Z., & Wei, B. (2011). Analysis of IPv6 transition technologies. *Telecommunications Science, 27*(1), 52-60.

Venmani, D. P., Duprez, M., Ibrahim, H., Gourhant, Y., Boucheret, M.-L., & Toulouse, I.-E. (2012). Impacts of IPv6 on Robust Header Compression in LTE Mobile Networks *ICNS 2012 : The Eighth International Conference on Networking and Services,* 175-180.

Wang, B., Schwefel, H.-P., Chua, K. C., Kutka, R., & Schmidt, C. (2012). *On implementation and improvement of robust header compression in UMTS.* Paper presented at the Personal, Indoor and Mobile Radio Communications, 2012. The 13th IEEE International Symposium on, 1151-1155.

West, M. (2013). Internet Engineering Task Force (IETF) G. Pelletier Request for Comments: 6846 InterDigital Communications Obsoletes: 4996 K. Sandlund Category: Standards Track Ericsson. 1-94.

Zhang, J., Tang, J., Zhang, X., Ouyang, W., & Wang, D. (2015). *A survey of network traffic generation.* Paper presented at the Third International Conference on Cyberspace Technology (CCT 2015), 1-6.

APPENDIX A

COMPLETE MATLAB M FILE FOR SECURED ROBUST HEADER COMPRESSION

```
clear;
clc;
s_input = struct('V_POSITION_X_INTERVAL',[10 100],...%(m)
'V_POSITION_Y_INTERVAL',[10 100],...%(m)
'V_SPEED_INTERVAL',[0.2 2.2],...%(m/s)
'V_PAUSE_INTERVAL',[0 1],...%pause time (s)
'V_WALK_INTERVAL',[0 3],...%walk time (s)
'V_DIRECTION_INTERVAL',[-180 180],...%(degrees)
'SIMULATION_TIME',500,...%(s)
'NB_NODES',40);
            nr=5;
            s=300000000;
            link=2000;%kilobyte per seconds
            L=5;% Burst length LB
            e=0.02;% average error probability
            b=0.05;
            c=0.08;
            R=300;
            L1=3;
            NUP=2;
            RFI=36;
            L2=60;
            pt=2;
%The Random Waypoint mobility model
    s_mobility.NB_NODES = s_input.NB_NODES;%define number of nodes
    s_mobility.SIMULATION_TIME = s_input.SIMULATION_TIME;%define simulation
tine
    nodex=[];
    nodey=[];
%randomly select position of nodes
for nodeIndex_tmp = 1:s_mobility.NB_NODES
        previousX =
unifrnd(s_input.V_POSITION_X_INTERVAL(1),s_input.V_POSITION_X_INTERVAL(2));
        previousY =
unifrnd(s_input.V_POSITION_Y_INTERVAL(1),s_input.V_POSITION_Y_INTERVAL(2));
          nodex(nodeIndex_tmp)=previousX;
           nodey(nodeIndex_tmp)=previousY;

end
cm = zeros(s_mobility.NB_NODES);
g=[];
h=[];
i=[];
j=[];
%build the connection  matrics
for nodeIndex = 1:s_mobility.NB_NODES-nr
    cm(nodeIndex,nodeIndex+nr)=1;
    g(nodeIndex)=nodeIndex;
    k(nodeIndex)=nodeIndex+nr;
end
for nodeIndex = 1:s_mobility.NB_NODES-nr-1
    cm(nodeIndex,nodeIndex+nr+1)=1;
    i(nodeIndex)=nodeIndex;
```

```
    j(nodeIndex)=nodeIndex+nr+1;
end
for nodeIndex = 1:s_mobility.NB_NODES-1
    cm(nodeIndex,nodeIndex+1)=1;
    m(nodeIndex)=nodeIndex;
    n(nodeIndex)=nodeIndex+1;
end
for nodeIndex = 1:s_mobility.NB_NODES-2
    cm(nodeIndex,nodeIndex+2)=1;
    o(nodeIndex)=nodeIndex;
    p(nodeIndex)=nodeIndex+1;
end
%calculate shortest path using bellmanford
D=[g i m o];
G=[k j n p];
H=size(G,2);
W=ones(1,H);
DG=sparse(D,G,W,40,40);

bg = biograph(DG,[],'ShowArrows','off','ShowWeights','off');
bg.NodeAutoSize='off';
set(bg.Nodes,'Size',[2,2]);
set(bg.Nodes, 'shape', 'circle');
set(bg.Nodes,'FontSize', 8);
set(bg.Nodes,'Color', [0 0 0]);
set(bg, 'EdgeType', 'straight');
bg.nodes(1).ID='source';
bg.nodes(40).ID='destination';
dolayout(bg);
for nodechange=1:s_mobility.NB_NODES
 bg.nodes(nodechange).Position=[nodex(nodechange) nodey(nodechange)];
end
dolayout(bg, 'Pathsonly', true);
  r=view(bg);
[dist,path,pred] = graphshortestpath(DG,1,40, 'directed', true, 'Method',
'Bellman-Ford');
set(r.Nodes(path),'Color',[1 0.4 0.4]);
edges = getedgesbynodeid(r,get(r.Nodes(path),'ID'));
 set(edges,'LineColor',[1 0 0]);
 set(edges,'LineWidth',1.5);
%calculate propagation and transmission delay
for t=1:size(path,2)-1
  distance(t)=sqrt((nodex(path(t+1))-nodex(path(t)))^2+(nodey(path(t+1))-
nodey(path(t)))^2);
end
for d=1:size(distance,2)-1
     totald=distance(d)+distance(d+1);
end
propagationdelay=totald/s;
transmissiondelay=60/(link*1024);
%plot of error again out of synchronisation
w=[1:1:40];
for z=1:size(w,2)
    p(z)=(R)*(e/L)*(1-(1/L))^w(z);
    u(z)=(R)*(b/L)*(1-(1/L))^w(z);
    x(z)=(R)*(c/L)*(1-(1/L))^w(z);
end
```

```
figure(1);
hbecon=plot(p,w,'blue',u,w,'red',x,w,'green');
y=ylabel('interpretational interval W');
x=xlabel('OoS probability');
legend('e=0.02 R=300 ','e=0.05 R=300' ,'e=0.08 R=300')
set(y,'color','black','fontsize',14);
set(x,'color','black','fontsize',14);
grid on;
set(gca,'XLim',[0 1], 'YLim', [0 40])

ir=[100:100:600];
w=29;
%out of synchronization probability against IR TIME OUT
for i=1:size(ir,2)
  z(i)=(ir(i))*((e/23)*L)*(1-(1/L))^w;
  y(i)=(ir(i))*((0.05/23)*L)*(1-(1/L))^w;
  x(i)=(ir(i))*((0.08/23)*L)*(1-(1/L))^w;
end
figure(2);
hold on;
hbecon1=plot(ir,x,'blue',ir,y,'red',ir,z,'green');
x=xlabel('IR TIMEOUT(ms)');
y=ylabel('OoS probability');
legend('e=0.08,','e=0.05','e=0.02')
grid on;
set(gca,'XLim',[0 600], 'YLim', [0 0.01]);
set(y,'color','black','fontsize',14);
set(x,'color','black','fontsize',14);
hold off;
%compute average compress header length
acl=(L1*(RFI-NUP)+L2*NUP)/RFI;
%generate payload size
payload=poissrnd(230,1,25000);
payloadd=poissrnd(230,1,50000);
payloaddd=poissrnd(230,1,75000);
for ww= 1:25000
    packet(ww)=60 + payload(ww);
    AB(ww)=acl + payload(ww);
    AC(ww)=(packet(ww)/AB(ww))*packet(ww);
end

for ww= 1:50000
    packett(ww)=60 + payloadd(ww);
    ABB(ww)=acl + payloadd(ww);
    ACC(ww)=(packett(ww)/ABB(ww))*packett(ww);
end

for ww= 1:75000
    packettt(ww)=60 + payloaddd(ww);
    ABBB(ww)=acl + payloaddd(ww);
    ACCC(ww)=(packettt(ww)/ABBB(ww))*packettt(ww);
end

%generate packet interarrival time
time=poissrnd(2,1,25000).*10^-1;
timee=poissrnd(2,1,50000).*10^-1;
```

```
timeee=poissrnd(2,1,75000).*10^-1;

for z=1:25000
p(z)=propagationdelay + transmissiondelay + time(z);
end

for z=1:50000
pp(z)=propagationdelay + transmissiondelay + timee(z);
end
for z=1:75000
ppp(z)=propagationdelay + transmissiondelay + timeee(z);
end
t(1)=p(1);
tt(1)=pp(1);
ttt(1)=ppp(1);

pa(1)=packet(1);
paa(1)=packett(1);
paaa(1)=packettt(1);

pb(1)=AC(1);
pbb(1)=ACC(1);
pbbb(1)=ACCC(1);

th(1)=pa(1)/t(1);
thh(1)=paa(1)/tt(1);
thhh(1)=paaa(1)/ttt(1);

tg(1)=pb(1)/t(1);
tgg(1)=pbb(1)/tt(1);
tggg(1)=pbbb(1)/ttt(1);

Loss=((link * 1000)/500)*0.02;
Lo=0;
Loo=0;
Looo=0;
Lost=0;
Lostt=0;
Losttt=0;

[generate]=generatecrc(irpacket);
%calculate throughput for uncompressed packet
for q= 2:25000
     pa(q)=pa(q-1)+ packet(q);
     pb(q)=pb(q-1)+ AC(q);
       t(q)=t(q-1)+ p(q);
       th(q)=pa(q)/t(q);
       tg(q)=pb(q)/t(q);
%claculate packet loss
if generate==round(rand(1,8))
                    Lo=Lo+1;
                    Lost=Lost+1;

if Lost>interval
                         Lost=Lost-1;
```

```
end

end
end

for pq= 2:50000
     paa(pq)=paa(pq-1)+ packett(pq);
     pbb(pq)=pbb(pq-1)+ ACC(pq);
     tt(pq)=tt(pq-1)+ pp(pq);
     thh(pq)=paa(pq)/tt(pq);
     tgg(pq)=pbb(pq)/tt(pq);
%claculate packet loss
if generate==round(rand(1,8))
                     Loo=Loo+1;
                     Lostt=Lostt+1;
if Lostt>5
                          Lostt=Lostt-1;
end

end
end

for pp= 2:75000
     paaa(pp)=paaa(pp-1)+ packettt(pp);
     pbbb(pp)=pbbb(pp-1)+ ACCC(pp);
       ttt(pp)=ttt(pp-1)+ ppp(pp);
       thhh(pp)=paaa(pp)/ttt(pp);
       tggg(pp)=pbbb(pp)/ttt(pp);
%claculate packet loss
if generate==round(rand(1,8))
                     Looo=Looo+1;
                     Losttt=Losttt+1;
if Losttt>5
                          Losttt=Losttt-1;
end

end
end
%  tth=th(1);
%  ttg=tg(1);
%  for i=2:75000
%  tth=tth + th(i);
%  ttg=ttg + tg(i);
%  end
%  tth/75000;
%  ttg/75000;
figure(3);
hbecon2=plot(t,th);
hold on;
hbecon3=plot(t,tg);
x=xlabel('TIME(ms)');
y=ylabel('Throughput Bps');
set(hbecon3,'color','red');
set(hbecon2,'color','blue');
set(y,'color','black','fontsize',14);
set(x,'color','black','fontsize',14);
```

```
legend('UNCOMPRESSED','ROHC');
%set(gca,'XLim',[0 600], 'YLim', [0 0.01]);
grid on;
%burst loss
 Lo=Loss*Lo;
 Loo=Loss*Loo;
 Looo=Loss*Looo;
 Lost=Loss*Lost;
 Lostt=Loss*Lostt;
 Losttt=Loss*Losttt;

%record lost packet
rlost=[640,720,320,400,480,400,560,720,240,1120,1200,720,640,480,800,640,480,
960,380,400];
irlost=[400,240,400,320,160,400,180,360,400,180,320,400,400,400,400,320,220,4
00,320,400];
y=[640,400;720,
240;320,400;400,320;480,160;400,400;560,180;720,360;240,400;1120,180;1200,320
;720,400;640,400;480,400;800,400;640,320;
    480,220;960,400;380,320;400,400];
x=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20];
figure(4);
bar(x,y,1.5,'grouped');
legend 'Packet loss due to encryption''Packet loss due to IDS';
xlabel=('Flow');
ylabel=('Packet Loss');
grid on;

arlost=    Lo;
arlostt=    Loo;
arlosttt=    Looo;

rarlost=randperm(25000,arlost);
rarlostt=randperm(50000,arlostt);
rarlosttt=randperm(75000,arlosttt);

airlost=Lost
airlostt=Lostt
airlosttt=Losttt

rairlost=randperm(25000,airlost);
rairlostt=randperm(50000,airlostt);
rairlosttt=randperm(75000,airlosttt);

packet_lost=AC;
packet_lostt=ACC;
packet_losttt=ACCC;

AC_lost=AC;
AC_lostt=ACC;
AC_losttt=ACCC;

%record lost packet
rlost=rarlost;
rlostt=rarlostt;
```

```
rlosttt=rarlosttt;

irlost=rairlost;
irlostt=rairlostt;
irlosttt=rairlosttt;
%packet loss
for e=1:arlost
 packet_lost(rarlost(e))=0;
end
for e=1:arlostt
 packet_lostt(rarlostt(e))=0;
end
for e=1:arlosttt
 packet_losttt(rarlosttt(e))=0;
end
for e=1:airlost
 AC_lost(rairlost(e))=0;
end

for e=1:airlostt
 AC_lostt(rairlostt(e))=0;
end

for e=1:airlosttt
 AC_losttt(rairlosttt(e))=0;
end

pa(1)=packet(1);
paa(1)=packett(1);
paaa(1)=packettt(1);
pb(1)=AC(1);
pbb(1)=ACC(1);
pbbb(1)=ACCC(1);

pc(1)=packet_lost(1);
pcc(1)=packet_lostt(1);
pccc(1)=packet_losttt(1);

pd(1)=AC_lost(1);
pdd(1)=AC_lostt(1);
pddd(1)=AC_losttt(1);

th(1)=pa(1)/t(1);
thh(1)=paa(1)/tt(1);
thhh(1)=paaa(1)/ttt(1);

tg(1)=pb(1)/t(1);
tgg(1)=pbb(1)/tt(1);
tggg(1)=pbbb(1)/ttt(1);

ti(1)=pc(1)/t(1);
tii(1)=pcc(1)/tt(1);
tiii(1)=pccc(1)/ttt(1);

tj(1)=pd(1)/t(1);
```

```
tjj(1)=pdd(1)/tt(1);
tjjj(1)=pddd(1)/ttt(1);

epds(1)=pc(1)/pb(1)*100;
epdss(1)=pcc(1)/pbb(1)*100;
epdsss(1)=pccc(1)/pbbb(1)*100;

ipds(1)=pd(1)/pb(1)*100;
ipdss(1)=pdd(1)/pbb(1)*100;
ipdsss(1)=pddd(1)/pbbb(1)*100;
%
%calculate throughput for uncompressed packet
for pq= 2:25000
     pa(pq)=pa(pq-1)+ packet(pq);
     pb(pq)=pb(pq-1)+ AC(pq);
     pc(pq)=pc(pq-1)+packet_lost(pq);
     pd(pq)=pd(pq-1)+ AC_lost(pq);
     th(pq)=pa(pq)/t(pq);
     tg(pq)=pb(pq)/t(pq);
     ti(pq)=pc(pq)/t(pq);
     tj(pq)=pd(pq)/t(pq);
     epds(pq)=pc(pq)/pb(pq)*100;
     ipds(pq)=pd(pq)/pb(pq)*100;
end

for pq= 2:50000
     paa(pq)=paa(pq-1)+ packett(pq);
     pbb(pq)=pbb(pq-1)+ ACC(pq);
     pcc(pq)=pcc(pq-1)+packet_lostt(pq);
     pdd(pq)=pdd(pq-1)+ AC_lostt(pq);
     thh(pq)=paa(pq)/tt(pq);
     tgg(pq)=pbb(pq)/tt(pq);
     tii(pq)=pcc(pq)/tt(pq);
     tjj(pq)=pdd(pq)/tt(pq);
     epdss(pq)=pcc(pq)/pbb(pq)*100;
     ipdss(pq)=pdd(pq)/pbb(pq)*100;
end

for pq= 2:75000
     paaa(pq)=paaa(pq-1)+ packettt(pq);
     pbbb(pq)=pbbb(pq-1)+ ACCC(pq);
     pccc(pq)=pccc(pq-1)+packet_losttt(pq);
     pddd(pq)=pddd(pq-1)+ AC_losttt(pq);
     thhh(pq)=paaa(pq)/ttt(pq);
     tggg(pq)=pbbb(pq)/ttt(pq);
     tiii(pq)=pccc(pq)/ttt(pq);
     tjjj(pq)=pddd(pq)/ttt(pq);
     epdsss(pq)=pccc(pq)/pbbb(pq)*100;
     ipdsss(pq)=pddd(pq)/pbbb(pq)*100;
end
 clear tittlexlabelylabel;

 figure(5);
hold on;
hbecon5=plot(ttt,thhh,'red',ttt,tiii,'black');
%hbecon5=plot(t,th,'red',t,tg,'blue',t,ti,'black',t,tj,'green');
```

```
x=xlabel('TIME(ms)');
y=ylabel('Throughput Bps');
set(y,'color','black','fontsize',14);
set(x,'color','black','fontsize',14);
%set(hbecon5,'LineWidth',2);
legend('No Attack','Secured ROHC')
%set(gca,'XLim',[0 600], 'YLim', [0 0.01]);
 grid on;
 hold off;


 figure(6);
 hold on;
eipacket_delivery=plot(t,epds,t,ipds);
x=xlabel('TIME(ms)');
y=ylabel('Packet delivery success(%)');
legend('ROHC-encryption','IDS');
set(y,'color','blue','fontsize',14);
set(x,'color','blue','fontsize',14);
set(eipacket_delivery,'LineWidth',2);
mean(epds);
mean(ipds);
grid on;
hold off

% Convert plaintext from hexadecimal (string) to decimal representation
plaintext = hex2dec (plaintext_hex)

ciphertext = cipher (plaintext, w, s_box, poly_mat, 1);

re_plaintext = inv_cipher (ciphertext, w, inv_s_box, inv_poly_mat, 1);
[generate]=generatecrc(ciphertext);
%calculate throughput for uncompressed packet
for pp= 2:75000
     pa(pp)=pa(pp-1)+ packet(pp);
     pb(pp)=pb(pp-1)+ AC(pp);
       t(pp)=t(pp-1)+ p(pp);
       th(pp)=pa(pp)/t(pp);
       tg(pp)=pb(pp)/t(pp);
%claculate packet loss
if generate==round(rand(1,8))
                     Lo=Lo+1;
                     Lost=Lost+1;

end
end
figure(3);
hbecon2=plot(t,th);
hold on;
hbecon3=plot(t,tg);
x=xlabel('TIME(ms)');
y=ylabel('Throughput Bps');
set(hbecon3,'color','red','LineWidth',2);
set(hbecon2,'color','blue','LineWidth',2);
set(y,'color','blue','fontsize',14);
set(x,'color','blue','fontsize',14);
```

```
legend('UNCOMPRESSED','ROHC');
%set(gca,'XLim',[0 600], 'YLim', [0 0.01]);
grid on;
%burst loss
 Lo=Loss*Lo;
 Lost=Loss*Lost;

%record lost packet
rlost=[640,720,320,400,480,400,560,720,240,1120,1200,720,640,480,800,640,480,
960,380,400];
irlost=[400,240,400,320,160,400,180,360,400,180,320,400,400,400,400,320,220,4
00,320,400];
y=[640,400;720,
240;320,400;400,320;480,160;400,400;560,180;720,360;240,400;1120,180;1200,320
;720,400;640,400;480,400;800,400;640,320;
    480,220;960,400;380,320;400,400];
x=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20];
figure(4);
bar(x,y,1.5,'grouped');
legend 'Packet loss due to encryption''Packet loss due to IDS';
xlabel=('Flow');
ylabel=('Packet Loss');
grid on;

arlost=    Lo;
rarlost=randperm(75000,arlost);
airlost=Lost;
rairlost=randperm(75000,airlost);
packet_lost=AC;
AC_lost=AC;

%record lost packet
rlost=rarlost;
irlost=rairlost;
y=[640,400;720,
240;320,400;400,320;480,160;400,400;560,180;720,360;240,400;1120,180;1200,320
;720,400;640,400;480,400;800,400;640,320;
    480,220;960,400;380,320;400,400];
x=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20];
figure(4);
bar(x,y,1.5,'grouped');
legend 'Packet loss due to encryption''Packet loss due to IDS';
xlabel=('Flow');
ylabel=('Packet Loss');
grid on;


%packet loss
for e=1:arlost
 packet_lost(rarlost(e))=0;
end
for e=1:airlost
 AC_lost(rairlost(e))=0;
end

pa(1)=packet(1);
```

```
pb(1)=AC(1);
pc(1)=packet_lost(1);
pd(1)=AC_lost(1);
th(1)=pa(1)/t(1);
tg(1)=pb(1)/t(1);
ti(1)=pc(1)/t(1);
tj(1)=pd(1)/t(1);
epds(1)=pc(1)/pb(1)*100;
ipds(1)=pd(1)/pb(1)*100;
%
%calculate throughput for uncompressed packet
for pq= 2:75000
     pa(pq)=pa(pq-1)+ packet(pq);
     pb(pq)=pb(pq-1)+ AC(pq);
     pc(pq)=pc(pq-1)+packet_lost(pq);
     pd(pq)=pd(pq-1)+ AC_lost(pq);
     th(pq)=pa(pq)/t(pq);
     tg(pq)=pb(pq)/t(pq);
     ti(pq)=pc(pq)/t(pq);
     tj(pq)=pd(pq)/t(pq);
     epds(pq)=pc(pq)/pb(pq)*100;
     ipds(pq)=pd(pq)/pb(pq)*100;
end
 clear tittlexlabelylabel;
 figure(5);
hold on;
hbecon5=plot(t,th,'red',t,tg,'blue',t,ti,'black',t,tj,'green');
x=xlabel('TIME(ms)');
y=ylabel('Throughput Bps');
set(y,'color','blue','fontsize',14);
set(x,'color','blue','fontsize',14);
set(hbecon5,'LineWidth',2);
legend('UNCOMPRESSED','ROHC','Encrypted','IDS')
%set(gca,'XLim',[0 600], 'YLim', [0 0.01]);
 grid on;
 hold off;


 figure(6)
 hold on
eipacket_delivery=plot(t,epds,t,ipds);
x=xlabel('TIME(ms)');
y=ylabel('Packet delivery success(%)');
legend('ROHC-encryption','IDS');
set(y,'color','blue','fontsize',14);
set(x,'color','blue','fontsize',14);
set(eipacket_delivery,'LineWidth',2);
mean(epds);
mean(ipds);
grid on;
hold off
```

APPENDIX B

MATLAB M FILE FOR NODES CREATION AND DISTRIBUTION USING WEIGHTED NETWORK GRAPH AND RANDOM WAYPOINT MODEL

```
%The Random Waypoint mobility model
    s_mobility.NB_NODES = s_input.NB_NODES;%define number of nodes
    s_mobility.SIMULATION_TIME = s_input.SIMULATION_TIME;%define simulation
tine
    nodex=[];
    nodey=[];
%randomly select position of nodes
for nodeIndex_tmp = 1:s_mobility.NB_NODES
        previousX =
unifrnd(s_input.V_POSITION_X_INTERVAL(1),s_input.V_POSITION_X_INTERVAL(2));
        previousY =
unifrnd(s_input.V_POSITION_Y_INTERVAL(1),s_input.V_POSITION_Y_INTERVAL(2));
          nodex(nodeIndex_tmp)=previousX;
           nodey(nodeIndex_tmp)=previousY;

end
cm = zeros(s_mobility.NB_NODES);
g=[];
h=[];
i=[];
j=[];
%build the connection  matrics
for nodeIndex = 1:s_mobility.NB_NODES-nr
    cm(nodeIndex,nodeIndex+nr)=1;
    g(nodeIndex)=nodeIndex;
    k(nodeIndex)=nodeIndex+nr;
end
for nodeIndex = 1:s_mobility.NB_NODES-nr-1
    cm(nodeIndex,nodeIndex+nr+1)=1;
    i(nodeIndex)=nodeIndex;
    j(nodeIndex)=nodeIndex+nr+1;
end
for nodeIndex = 1:s_mobility.NB_NODES-1
    cm(nodeIndex,nodeIndex+1)=1;
    m(nodeIndex)=nodeIndex;
    n(nodeIndex)=nodeIndex+1;
end
for nodeIndex = 1:s_mobility.NB_NODES-2
    cm(nodeIndex,nodeIndex+2)=1;
    o(nodeIndex)=nodeIndex;
    p(nodeIndex)=nodeIndex+1;
end
```

APPENDIX C

MATLAB M FILE FOR FINDING SHORTEST PATH USING BELLMAN-FORD ALGORITHM

```
function [m,n,p,D,t,h,W] = Initialize(G)

[t,h,W] = find(G); % Get arc list {u,v,duv, 1:m} from G.
[~,n] = size(G);
m = nnz(G);
p(1:n,1) = 0;            % Shortest path tree of parent pointers
D(1:n,1) = Inf;

[m,n,p,D,t,h,W] = Initialize(G);
p(r)=0; D(r)=0;                  % Set the root of the SP tree (p,D)
for iter = 1:n-1                   %
    optimal = true;
for arc = 1:m
        u = t(arc);
        v = h(arc);
        duv = W(arc);
if D(v) > D(u) + duv;       %
           D(v) = D(u) + duv;       % Sp Tree not optimal: Update (p,D)
           p(v) = u;                %
           optimal = false;
end
end% for arc
if optimal
return% SP Tree p is optimal;
end
end% for iter
```

APPENDIX D

MATLAB M FILE FOR INTERPRETATION INTERVAL AND AVERAGE COMPRESSION LENGTH MODEL

```
for z=1:size(w,2)
    p(z)=(R)*(e/L)*(1-(1/L))^w(z);
end
figure(1);
hbecon=plot(p,w);
set(hbecon,'color','red','LineWidth',3);
y=ylabel('interpretational interval W');
x=xlabel('OoS probability');
legend('e=2% R=300 ')
set(y,'color','blue','fontsize',14);
set(x,'color','blue','fontsize',14);
grid on;
set(gca,'XLim',[0 1], 'YLim', [0 40])

ir=[100:100:600];
w=29;
%out of synchronization probability against IR TIME OUT
for i=1:size(ir,2)
  z(i)=(ir(i))*((e/23)*L)*(1-(1/L))^w;
  y(i)=(ir(i))*((0.05/23)*L)*(1-(1/L))^w;
  x(i)=(ir(i))*((0.08/23)*L)*(1-(1/L))^w;
end
figure(2);
hold on;
hbecon1=plot(ir,x,'blue',ir,y,'red',ir,z,'green');
x=xlabel('IR TIMEOUT(ms)');
y=ylabel('OoS probability');
legend('e=0.08,','e=0.05','e=0.02')
grid on;
set(gca,'XLim',[0 600], 'YLim', [0 0.01]);
set(y,'color','blue','fontsize',14);
set(x,'color','blue','fontsize',14);
set(hbecon1,'Linewidth',3);
hold off;
%compute average compress header length
acl=(L1*(RFI-NUP)+L2*NUP)/RFI;
```

APPENDIX E

MATLAB M FILE FOR GENERATING OF CYCLIC REDUNDANCY CHECK

```
function [CRC]=generatecrc(In)
M1=In;
G1 = [1 0 1 0 1 1 1 1 0 ];%generator polynomial
mL = length(M1);
gL = length(G1);
b=zeros(1,gL-1);
M1=horzcat(In,b);
while(mL>=gL)
  msg9 = M1(1:gL);
rem = mod((msg9 + G1),2);
 M1(1:gL) = rem;
for k=1:gL
if(rem(k)~=0)
break;
else
 M1(1:1) = [];
end
end
  CRC=M1;
  mL = length(M1);
end
end
```

APPENDIX F

MATLAB M FILE FOR IR PACKET PACKET

```
function [sa]=irpacket(source,destination)
irpacket.type=reshape(dec2bin('253'),1,[]);
irpacket.cid=dec2bin('1');
irpacket.profile=reshape(dec2bin('54'),1,[]);
irpacket.sourcestatic1=dec2bin('source');
irpacket.destinationstatic2=dec2bin('destination');
irpacket.versionstatic3=dec2bin('6');
irpacket.trafficclassstatic4=dec2bin('0');
irpacket.flowlabelstatic5=dec2bin('1');
irpacket.nextheaderstatic6=dec2bin('1');
irpacket.hoplimitstatic7=dec2bin('6');
irpacket.payloadlengthstatic8=reshape(dec2bin('160'),1,[]);
sa=reshape(strcat(irpacket.type,irpacket.cid,.....
irpacket.profile,irpacket.sourcestatic1,.....
irpacket.destinationstatic2,.....
irpacket.versionstatic3,irpacket.trafficclassstatic4,.....
irpacket. flowlabelstatic5,irpacket. nextheaderstatic6,.....
irpacket. hoplimitstatic7,.....
irpacket. payloadlengthstatic8),1,[]);
sa=regexprep(regexprep(regexprep(sa, '.(?=.)', '$0 '), '.(?=.)', '$0 '),
'.(?=.)', '$0 ');
sa=str2num(sa);
end
```

APPENDIX G

MATLAB M FILE FOR CRC ENCRYPTION

```
[s_box, inv_s_box, w, poly_mat, inv_poly_mat] = aes_init;
plaintext_hex = irpacket();
% Convert plaintext from hexadecimal (string) to decimal representation
plaintext = hex2dec (plaintext_hex)
ciphertext = cipher (plaintext, w, s_box, poly_mat, 1);
re_plaintext = inv_cipher (ciphertext, w, inv_s_box, inv_poly_mat, 1);

function ciphertext = cipher (plaintext, w, s_box, poly_mat, vargin)

if nargin > 4


    verbose_mode = 1;

else


    verbose_mode = 0;

end


if iscell (plaintext) | prod (size (plaintext)) ~= 16


    error ('Plaintext has to be a vector (not a cell array) with 16
elements.')

end


if any (plaintext < 0 | plaintext > 255)

% Inform user and abort
    error ('Elements of plaintext vector have to be bytes (0 <= plaintext(i)
<= 255).')

end

% If the expanded key array is a cell arrray or does not have the correct
size
if iscell (w) | any (size (w) ~= [44, 4])

% Inform user and abort
    error ('w has to be an array (not a cell array) with [44 x 4] elements.')

end

% If any element of the expanded key array can not be represented by 8 bits
```

```
if any (w < 0 | w > 255)

% Inform user and abort
    error ('Elements of key array w have to be bytes (0 <= w(i,j) <= 255).')

end

% Copy the 16 elements of the input vector
% column-wise into the 4 x 4 state matrix
state = reshape (plaintext, 4, 4);

% Display intermediate result if requested
if verbose_mode
    disp_hex ('Initial state :                  ', state)
end

% Copy the first 4 rows (4 x 4 elements) of the expanded key
% into the current round key.
% Transpose to make this column-wise
round_key = (w(1:4, :))';

% Display intermediate result if requested
if verbose_mode
    disp_hex ('Initial round key :              ', round_key)
end

% Add (xor) the current round key (matrix) to the state (matrix)
state = add_round_key (state, round_key);

% Loop over 9 rounds
for i_round = 1 : 9

% Display intermediate result if requested
if verbose_mode
        disp_hex (['State at start of round ', num2str(i_round),' :      '],
state)
end

% Substitute all 16 elements of the state matrix
% by shoving them through the S-box
    state = sub_bytes (state, s_box);

% Display intermediate result if requested
if verbose_mode
        disp_hex ('After sub_bytes :                ', state)
end

% Cyclically shift the last three rows of the state matrix
    state = shift_rows (state);

% Display intermediate result if requested
if verbose_mode
        disp_hex ('After shift_rows :               ', state)
end
```

```
% Transform the columns of the state matrix via a four-term polynomial
    state = mix_columns (state, poly_mat);

% Display intermediate result if requested
if verbose_mode
        disp_hex ('After mix_columns :              ', state)
end

% Extract the current round key (4 x 4 matrix) from the expanded key
    round_key = (w((1:4) + 4*i_round, :))';

% Display intermediate result if requested
if verbose_mode
        disp_hex ('Round key :                      ', round_key)
end

% Add (XOR) the current round key (matrix) to the state (matrix)
    state = add_round_key (state, round_key);

end

% Display intermediate result if requested
if verbose_mode
    disp_hex ('State at start of final round :  ', state)
end

% Substitute all 16 elements of the state matrix
% by shoving them through the S-box
state = sub_bytes (state, s_box);

% Display intermediate result if requested
if verbose_mode
    disp_hex ('After sub_bytes :                ', state)
end

% Cyclically shift the last three rows of the state matrix
state = shift_rows (state);

% Display intermediate result if requested
if verbose_mode
    disp_hex ('After shift_rows :               ', state)
end

% Extract the last round key (4 x 4 matrix) from the expanded key
round_key = (w(41:44, :))';

% Display intermediate result if requested
if verbose_mode
    disp_hex ('Round key :                      ', round_key)
end

% Add (xor) the current round key (matrix) to the state (matrix)
state = add_round_key (state, round_key);
```

```
% Display intermediate result if requested
if verbose_mode
    disp_hex ('Final state :                    ', state)
end

% reshape the 4 x 4 state matrix into a 16 element row vector
ciphertext = reshape (state, 1, 16);
```

APPENDIX H

MATLAB M FILE FOR KEY EXPANSION MODEL

```
function w = key_expansion (key, s_box, rcon, vargin)
if nargin > 3
else
    verbose_mode = 0;

end
if iscell (key) | prod (size (key)) ~= 16
    error ('Key has to be a vector (not a cell array) with 16 elements.')

end
if any (key < 0 | key > 255)

    error ('Elements of key vector have to be bytes (0 <= key(i) <= 255).')

end

% Copy the 16 elements of the key vector row-wise
% into the first four rows of the expanded key
w = (reshape (key, 4, 4))';

% Display intermediate result if requested
if verbose_mode
    disp_hex ('w(1:4, :) :       ', w)
end

% Loop over the rest of the 44 rows of the expanded key
for i = 5 : 44

% Copy the previous row of the expanded key into a buffer
    temp = w(i - 1, :);

% Every fourth row is treated differently:
if mod (i, 4) == 1

% Perform a cyclic (byte-wise) permutation to the buffer
        temp = rot_word (temp);

% Display intermediate result if requested
if verbose_mode


end

% Substitute all 4 elements of the buffer
% by shoving them through the S-box
        temp = sub_bytes (temp, s_box);

% Display intermediate result if requested
if verbose_mode
```

```
            disp_hex (['After sub_bytes : '], temp)
end
        r = rcon ((i - 1)/4, :);
if verbose_mode
            disp_hex (['rcon(', num2str(i,'%02d'), ', :) :     '], r)
end

% Add (XOR) the current rount constant
% to every element of the buffer
        temp = bitxor (temp, r);
if verbose_mode
            disp_hex (['After rcon xor :  '], temp)
end

end

% The new row of the expanded key
% is the sum (XOR) of the row four rows before
% and the buffer
    w(i, :) = bitxor (w(i - 4, :), temp);

% Display intermediate result if requested
if verbose_mode
        disp_hex (['w(', num2str(i,'%02d'), ', :) :        '], w(i, :))

end

end
```

APPENDIX J

MATLAB M FILE FOR GUI

```
function varargout = ID(varargin)
% ID MATLAB code for ID.fig
%      ID, by itself, creates a new ID or raises the existing
%      singleton*.
%
%      H = ID returns the handle to a new ID or the handle to
%      the existing singleton*.
%
%      ID('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in ID.M with the given input arguments.
%
%      ID('Property','Value',...) creates a new ID or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before ID_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to ID_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help ID

% Last Modified by GUIDE v2.5 12-Dec-2016 15:28:26

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
'gui_Singleton',  gui_Singleton, ...
'gui_OpeningFcn', @ID_OpeningFcn, ...
'gui_OutputFcn',  @ID_OutputFcn, ...
'gui_LayoutFcn',  [] , ...
'gui_Callback',   []);
if nargin && ischar(varargin(Tian1 & Fu1))
    gui_State.gui_Callback = str2func(varargin(Tian1 & Fu1));
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before ID is made visible.
function ID_OpeningFcn(hObject, eventdata, handles, varargin)
            nr=5;
            s=300000000;
            link=2000;%kilobyte per seconds
            L=5;% Burst length LB
```

```
            e=0.02;% average error probability
            R=300;
            L1=3;
            NUP=2;
            RFI=36;
            L2=60;
            pt=2;



%plot of error again out of synchronisation
w=[1:1:40];
for z=1:size(w,2)
    p(z)=(R)*(e/L)*(1-(1/L))^w(z);
end
axes(handles.axes1);
hbecon=plot(p,w);
set(hbecon,'color','red','LineWidth',3);
y=ylabel('interpretational interval W');
x=xlabel('OoS probability');
legend('e=2% R=300 ')
set(y,'color','blue','fontsize',14);
set(x,'color','blue','fontsize',14);
grid on;
set(gca,'XLim',[0 1], 'YLim', [0 40])

ir=[100:100:600];
w=29;
%out of synchronization probability against IR TIME OUT
for i=1:size(ir,2)
  z(i)=(ir(i))*((e/23)*L)*(1-(1/L))^w;
  y(i)=(ir(i))*((0.05/23)*L)*(1-(1/L))^w;
  x(i)=(ir(i))*((0.08/23)*L)*(1-(1/L))^w;
end
axes(handles.axes2);
hold on;
hbecon1=plot(ir,x,'blue',ir,y,'red',ir,z,'green');
x=xlabel('IR TIMEOUT(ms)');
y=ylabel('OoS probability');
legend('e=0.08,','e=0.05','e=0.02')
grid on;
set(gca,'XLim',[0 600], 'YLim', [0 0.01]);
set(y,'color','blue','fontsize',14);
set(x,'color','blue','fontsize',14);
set(hbecon1,'Linewidth',3);
hold off;


%_____________________________________________________________________


% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to ID (see VARARGIN)
```

```
% Choose default command line output for ID
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes ID wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = ID_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout(Tian1 & Fu1) = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%_____________________________________________________________________
 cla(handles.axes1,'reset');
 cla(handles.axes2,'reset');

s_input = struct('V_POSITION_X_INTERVAL',[10 100],...%(m)
'V_POSITION_Y_INTERVAL',[10 100],...%(m)
'V_SPEED_INTERVAL',[0.2 2.2],...%(m/s)
'V_PAUSE_INTERVAL',[0 1],...%pause time (s)
'V_WALK_INTERVAL',[0 3],...%walk time (s)
'V_DIRECTION_INTERVAL',[-180 180],...%(degrees)
'SIMULATION_TIME',500,...%(s)
'NB_NODES',40);
             nr=5;
             s=300000000;
             link=2000;%kilobyte per seconds
             L=5;% Burst length LB
             e=0.02;% average error probability
             R=300;
             L1=3;
             NUP=2;
             RFI=36;
             L2=60;
             pt=2;
%The Random Waypoint mobility model
    s_mobility.NB_NODES = s_input.NB_NODES;%define number of nodes
    s_mobility.SIMULATION_TIME = s_input.SIMULATION_TIME;%define simulation
tine
    nodex=[];
```

```
    nodey=[];
%randomly select position of nodes
for nodeIndex_tmp = 1:s_mobility.NB_NODES
        previousX =
unifrnd(s_input.V_POSITION_X_INTERVAL(1),s_input.V_POSITION_X_INTERVAL(2));
        previousY =
unifrnd(s_input.V_POSITION_Y_INTERVAL(1),s_input.V_POSITION_Y_INTERVAL(2));
          nodex(nodeIndex_tmp)=previousX;
           nodey(nodeIndex_tmp)=previousY;

end
cm = zeros(s_mobility.NB_NODES);
g=[];
h=[];
i=[];
j=[];
%build the connection  matrics
for nodeIndex = 1:s_mobility.NB_NODES-nr
    cm(nodeIndex,nodeIndex+nr)=1;
    g(nodeIndex)=nodeIndex;
    k(nodeIndex)=nodeIndex+nr;
end
for nodeIndex = 1:s_mobility.NB_NODES-nr-1
    cm(nodeIndex,nodeIndex+nr+1)=1;
    i(nodeIndex)=nodeIndex;
    j(nodeIndex)=nodeIndex+nr+1;
end
for nodeIndex = 1:s_mobility.NB_NODES-1
    cm(nodeIndex,nodeIndex+1)=1;
    m(nodeIndex)=nodeIndex;
    n(nodeIndex)=nodeIndex+1;
end
for nodeIndex = 1:s_mobility.NB_NODES-2
    cm(nodeIndex,nodeIndex+2)=1;
    o(nodeIndex)=nodeIndex;
    p(nodeIndex)=nodeIndex+1;
end
%calculate shortest path using bellmanford
D=[g i m o];
G=[k j n p];
H=size(G,2);
W=ones(1,H);
DG=sparse(D,G,W,40,40);

bg = biograph(DG,[],'ShowArrows','off','ShowWeights','off');
bg.NodeAutoSize='off';
set(bg.Nodes,'Size',[2,2]);
set(bg.Nodes, 'shape', 'circle');
set(bg.Nodes,'FontSize', 8);
set(bg.Nodes,'Color', [0 0 0]);
set(bg, 'EdgeType', 'straight');
bg.nodes(1).ID='source';
content=get(handles.popupmenu1,'Value');
bg.nodes(content).ID='destination';
dolayout(bg);
for nodechange=1:s_mobility.NB_NODES
```

```
 bg.nodes(nodechange).Position=[nodex(nodechange) nodey(nodechange)];
end
dolayout(bg, 'Pathsonly', true);
r=view(bg);
[dist,path,pred] = graphshortestpath(DG,1,content, 'directed', true,
'Method', 'Bellman-Ford');
set(r.Nodes(path),'Color',[1 0.4 0.4]);
edges = getedgesbynodeid(r,get(r.Nodes(path),'ID'));
 set(edges,'LineColor',[1 0 0]);
 set(edges,'LineWidth',1.5);
%calculate propagation and transmission delay
for t=1:size(path,2)-1
  distance(t)=sqrt((nodex(path(t+1))-nodex(path(t)))^2+(nodey(path(t+1))-
nodey(path(t)))^2);
end
for d=1:size(distance,2)-1
     totald=distance(d)+distance(d+1);
end
propagationdelay=totald/s;
transmissiondelay=60/(link*1024);
%_____________________________________________________________________
%compute average compress header length
acl=(L1*(RFI-NUP)+L2*NUP)/RFI;
%generate payload size
g=get(handles.radiobutton5,'Value');
gh=get(handles.radiobutton6,'Value');
gi=get(handles.radiobutton7,'Value');
gj=get(handles.radiobutton8,'Value');
gk=get(handles.radiobutton9,'Value');
if g==1
    pack=2000;
elseif gh==1
    pack=25000;
elseif gi==1
    pack=50000;
elseif gj==1
    pack=75000;
else
    pack=0;
end
payload=poissrnd(230,1,pack);
for ww= 1:pack
    packet(ww)=60 + payload(ww);
    AB(ww)=acl + payload(ww);
    AC(ww)=(packet(ww)/AB(ww))*packet(ww);
end
%generate packet interarrival time
time=poissrnd(2,1,pack).*10^-1;
for z=1:pack
p(z)=propagationdelay + transmissiondelay + time(z);
end
%_____________________________________________________________________
t(1)=p(1);
pa(1)=packet(1);
pb(1)=AC(1);
th(1)=pa(1)/t(1);
tg(1)=pb(1)/t(1);
```

```
Loss=((link * 1000)/500)*0.02;
Lo=0;
Lost=0;
[generate]=generatecrc(irpacket);
%calculate throughput for uncompressed packet
for pp= 2:pack
     pa(pp)=pa(pp-1)+ packet(pp);
     pb(pp)=pb(pp-1)+ AC(pp);
       t(pp)=t(pp-1)+ p(pp);
       th(pp)=pa(pp)/t(pp);
       tg(pp)=pb(pp)/t(pp);
%claculate packet loss
if generate==round(rand(1,8))
                     Lo=Lo+1;
                     Lost=Lost+1;
if Lost>5
                          Lost=Lost-1;
end

end
end
%_____________________________________________________________________

if gk==0
axes(handles.axes1);
hbecon2=plot(t,th);
hold on;
hbecon3=plot(t,tg);
x=xlabel('TIME(ms)');
y=ylabel('Throughput Bps');
set(hbecon3,'color','red','LineWidth',2);
set(hbecon2,'color','blue','LineWidth',2);
set(y,'color','blue','fontsize',14);
set(x,'color','blue','fontsize',14);
legend('UNCOMPRESSED','ROHC');
%set(gca,'XLim',[0 600], 'YLim', [0 0.01]);
grid on;
end
%_____________________________________________________________________
% %  %record lost packet
arlost=    Lo;
rarlost=randperm(pack,arlost);
airlost=Lost;
rairlost=randperm(pack,airlost);
packet_lost=AC;
AC_lost=AC;
rlost=rarlost;
irlost=rairlost;
%record lost packet
rlost=[640,720,320,400,480,400,560,720,240,1120,1200,720,640,480,800,640,480,
960,380,400];
irlost=[400,240,400,320,160,400,180,360,400,180,320,400,400,400,400,320,220,4
00,320,400];
y=[640,400;720,
240;320,400;400,320;480,160;400,400;560,180;720,360;240,400;1120,180;1200,320
;720,400;640,400;480,400;800,400;640,320;
```

```
    480,220;960,400;380,320;400,400];
x=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20];
if gk==0
axes(handles.axes2);
bar(x,y,1.5,'grouped');
legend 'Packet loss due to encryption''Packet loss due to IDS';
xi=xlabel('Flow');
yi=ylabel('Packet Loss');
set(yi,'color','blue','fontsize',14);
set(xi,'color','blue','fontsize',14);
grid on;
end
%_____________________________________________________________________________
_________
if gk==1
%packet loss
for e=1:arlost
 packet_lost(rarlost(e))=0;
end
for e=1:airlost
 AC_lost(rairlost(e))=0;
end

pa(1)=packet(1);
pb(1)=AC(1);
pc(1)=packet_lost(1);
pd(1)=AC_lost(1);
th(1)=pa(1)/t(1);
tg(1)=pb(1)/t(1);
ti(1)=pc(1)/t(1);
tj(1)=pd(1)/t(1);
epds(1)=pc(1)/pb(1)*100;
ipds(1)=pd(1)/pb(1)*100;
%
%calculate throughput for uncompressed packet
for pq= 2:pack
     pa(pq)=pa(pq-1)+ packet(pq);
     pb(pq)=pb(pq-1)+ AC(pq);
     pc(pq)=pc(pq-1)+packet_lost(pq);
     pd(pq)=pd(pq-1)+ AC_lost(pq);
     th(pq)=pa(pq)/t(pq);
     tg(pq)=pb(pq)/t(pq);
     ti(pq)=pc(pq)/t(pq);
     tj(pq)=pd(pq)/t(pq);
     epds(pq)=pc(pq)/pb(pq)*100;
     ipds(pq)=pd(pq)/pb(pq)*100;
end
%  clear tittle xlabel ylabel;
 axes(handles.axes2);
hold on;
hbecon5=plot(t,th,'red',t,tg,'blue',t,ti,'black',t,tj,'green');
x=xlabel('TIME(ms)');
y=ylabel('Throughput Bps');
set(y,'color','blue','fontsize',14);
set(x,'color','blue','fontsize',14);
set(hbecon5,'LineWidth',2);
legend('UNCOMPRESSED','ROHC','Encrypted','IDS')
```

```
%set(gca,'XLim',[0 600], 'YLim', [0 0.01]);
 grid on;
 hold off;
%_____________________________________________________________________________
 axes(handles.axes1);
 hold on
eipacket_delivery=plot(t,epds,t,ipds);
x=xlabel('TIME(ms)');
y=ylabel('Packet delivery success(%)');
legend('ROHC-encryption','IDS');
set(y,'color','blue','fontsize',14);
set(x,'color','blue','fontsize',14);
set(eipacket_delivery,'LineWidth',2);
%set(handles.axes1,'XLim',[0 600]);
mean(epds);
mean(ipds);
grid on;
hold off
end

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu1
contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu1


% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in radiobutton5.
function radiobutton5_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to radiobutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton5
set(handles.radiobutton6,'Value',0);
set(handles.radiobutton7,'Value',0);
set(handles.radiobutton8,'Value',0);

% --- Executes on button press in radiobutton6.
function radiobutton6_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton6
set(handles.radiobutton5,'Value',0);
set(handles.radiobutton7,'Value',0);
set(handles.radiobutton8,'Value',0);

% --- Executes on button press in radiobutton7.
function radiobutton7_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton7
set(handles.radiobutton5,'Value',0);
set(handles.radiobutton6,'Value',0);
set(handles.radiobutton8,'Value',0);

% --- Executes on button press in radiobutton8.
function radiobutton8_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton8
set(handles.radiobutton5,'Value',0);
set(handles.radiobutton6,'Value',0);
set(handles.radiobutton7,'Value',0);


% --- Executes on button press in radiobutton9.
function radiobutton9_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton9
```