DEVELOPMENT OF AN INTERNET PROTOCOL TRACEBACK SCHEME FOR DENIAL OF SERVICE ATTACK SOURCE DETECTION

BY

Omoniyi Wale SALAMI

Department of Computer Engineering,

Faculty of Engineering,

Ahmadu Bello University,

Zaria, Nigeria

February, 2020

DEVELOPMENT OF AN INTERNET PROTOCOL TRACEBACK SCHEME FOR DENIAL OF SERVICE ATTACK SOURCE DETECTION

BY

Omoniyi Wale SALAMI, ***B.Sc. (UNILAG) 1996***

P15EGCP8010

salami.omoniyi@outlook.com

A DISSERTATION SUBMITTED TO THE SCHOOL OF POSTGRADUATE STUDIES, AHMADU BELLO UNIVERSITY, ZARIA IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF MASTER OF SCIENCE (MSc) DEGREE IN COMPUTER ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING,
FACULTY OF ENGINEERING,
AHMADU BELLO UNIVERSITY,

ZARIA, NIGERIA

February, 2020

DECLARATION

I declare that the work in this dissertation entitled “Development of an Internet Protocol Traceback Scheme for Denial of Service Attack Source Detection” has been carried out by me in the Department of Computer Engineering. The information derived from the 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.

Omoniyi Wale SALAMI _____ _____

(Student)

CERTIFICATION

This dissertation entitled "DEVELOPMENT OF AN INTERNET PROTOCOL TRACEBACK SCHEME FOR DENIAL OF SERVICE ATTACK SOURCE DETECTION" by Omoniyi Wale SALAMI meets the regulations governing the award of the 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. E. A. Adedokun (Member, Supervisory Committee)	_____ Signature	_____ Date
Prof. M. B. Mu'azu (Head of Department)	_____ Signature	_____ Date
Prof. S. Z. Abubakar (Dean School of Postgraduate Studies)	_____ Signature	_____ Date

DEDICATION

This research work is dedicated to all the kind-hearted people who are readily available to assist without discrimination and leave positive impact in others life.

ACKNOWLEDGEMENT

All praises are due to Allah (SWT), the creator of heaven and earth, the Lord of universe. May the peace and blessings of Allah be upon His messenger, Muhammad.

I express my sincere gratitude to my supervisory team and mentors, Dr. I. J. Umoh, and Dr. E. A. Adedokun, for their motivation and support from the beginning to the end of my research work. I highly appreciate their patience and expertise correcting my mistakes and explaining how to do things right to me without which this work would neither have been rich in content nor be literarily presentable. I pray to God Almighty to reward you beautifully for the right guidance you provided me. It is a rare opportunity and blessing for one to have a boss who's helping hands are always opened to whoever needs assistance, without discriminating. I am one of such lucky ones that have such a boss in person of Professor. M. B. Mu'azu. I gained from him insights that helped shaped the problem and directed me to the solution. Sir, an attempt to describe or quantify the benefits of your openheartedness will result to undervaluation. My prayer to God Almighty is to continue to be with you in your inestimable service to humanity and keep the devils far away from you in your affairs. I can never forget the candid straight forwardness of Professor Muhammad, Y. Fatihu, Faculty of Veterinary Medicine, ABU. Meeting him was a blessing for me. He was a key player in my career. May God Almighty reward him for always being charitable in his opinions of others. I will always be grateful to Dr. A. D. Usman for rendering me his assistance when I needed it. May Almighty God always ease your affairs for you. I hope it will not be an understatement to call my cicerone in person of Dr. Ahmed Tijani Salawudeen (aka TJ) the Foundation Builder. He placed me on the initial point on the path with the least cost that led to existing proficient guiding agents in the large search space of research areas. The feed backs were perfect and the local updates were precise. Dr. TJ, I am very proud of you that is why I

described how it all happened eloquently in a Professor Wole Soyinka style with the terminologies you taught me. I am very grateful to you for showing me the direction. May Allah always guide you on the straight path. I am very grateful to all my lecturers. There is no part of what they taught me that did not become useful for the success of this work. May God Almighty bless you all.

My profound gratitude goes to all members of Computer and Control Research group. I appreciate your efforts critiquing my work and the helpful suggestions you provided me. May God bless you all.

My heartfelt appreciation goes to my M. Sc. Computer/Control colleagues, 2015/2016 set. You were all good to me and assisted in one way or the other. Thank you all and God bless you.

I am very grateful to my late Dad and Mum. May Allah bless them and provide best care for them more than they did to me when I was a baby. They gave me the best they could and laid my solid foundation in life. I thank all my siblings from Alhaji Abdulazeez Ajibade Abdssalam (aka Araba) to Nafiu Abdssalam and everybody in between. I also thank all my nephews and nieces from Mr Abdulwaheed Abimbola Abdssalam to Zeenat Kayode Abdssalam, and everybody in between, my children inclusive. Barakah Allah fiikum.

Lastly and most importantly, to my wife, my true sweetheart I am very grateful for your support that gave me focus on this work.

Omoniyi Wale SALAMI

February, 2020.

ABSTRACT

This dissertation presents the development of an Internet Protocol (IP) traceback scheme for the detection of a denial of service (DoS) attack source base on shark smell optimization algorithm (SSOA). Detection of the source of DoS attack is very important due to the serious damages the attack do cause and the need to bring the perpetrators to justice to stop the menace. DoS attack is a major threat to the security of network systems and consists of attacks that exploit the vulnerability in a network to overload it with tasks and prevent it from attending to other legitimate users. Flash event (FE) can cause traffic surge in a part of the network crossed by the attack path that is being traced. Flash event traffic surge can be very similar to a DoS attack and may mislead the present IP tracebacks schemes that are based on swarm optimization algorithms when tracing the source of an attack using flow-based search method. The challenge is more pronounced with flow-based search for detecting attack source because the flash event flow surge share very similar characteristics with DoS flooding attack. In order to mitigate the challenge of flash event traffic surge causing error in IP traceback schemes, DoS attack source traceback scheme based on shark smell optimization algorithm called the SSOA-DoSTBK was developed. It is incorporated with discernment policy for implementing hop-by-hop search to avoid flash event traffic surge and ascertain the nodes that are actually involved in routing the attack packets. This scheme was simulated in Network Simulator version 2 (NS2). The performance of SSOA-DoSTBK was evaluated using False Error Rate (FER), convergence time, and ability to detect spoofed IP attack source based on the correctness of the returned path as performance metrics. It was compared with results obtained from a scheme reported in literature called the modified ant colony system algorithm for IP traceback (ACS-IPTBK). The SSOA-DoSTBK performed better in FER and spoofed IP attack tests by as much as 32.06%. However, ACS-IPTBK converged faster than the SSOA-DoSTBK in the tests by as much as 1.2%.

Table of Contents

DECLARATION II
CERTIFICATION III
DEDICATION IV
ACKNOWLEDGEMENT V
ABSTRACT VII
List of Figures XI
List of Tables XII
List of Appendices XIII
ABBREVIATIONS XIV

CHAPTER ONE

INTRODUCTION

1.1 Background on Network Attacks 1
1.1.1 Background on Internet Protocol Packets Source Detection 3
1.2 Significance of Research 4
1.3 Statement of Problem 4
1.4 Aim and Objectives 5

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction 7
2.2 Review of Fundamental Concepts on DoS attack IP traceback 7
2.2.1 The DoS Attack and its Variants 7
2.2.2 Flash Event 11
2.2.3 DoS Attack IP Traceback Methodologies 14
2.2.4 DoS Attack Source Detection Process 17
2.3 Shark Smell Optimization Algorithm 18
2.3.1 Initialization 20
2.3.2 Scouring 20
2.3.3 Advancing 21
2.3.4 SSOA Exploitation 22
2.3.5 SSOA Exploration 23
2.3.6 Flowchart of SSOA Algorithm Search Process 23
2.4 Reconstructing the Network Topology 25
2.4.1 Implementation of Waxman Topology 27

2.4.2 Determining edges on attack path 28
2.5 Network Simulator Version 2 (NS2) 32
2.6 Review of Similar Works 33

CHAPTER THREE

MATERIALS AND METHODS

3.1 Introduction 38
3.2 Materials 38
3.3 Methodology 38
3.3.1 Development of the SSOA-DoSTBK 38
3.3.2 Discrimination Policy 39
3.3.3 Solving DoS IP Traceback Problem Using SSOA-DoSTBK 44
3.3.4 Comparison of SSOA-DoSTBK with ACS-IPTBK 47
3.3.5 Performance Evaluation 48

CHAPTER FOUR

RESULTS AND DISCUSSIONS

4.1 Introduction 49
4.2 Simulation Results 49
4.2.1 Evaluation of False Error Rate 49
4.2.1.1 FER of the Schemes under DoS attack 49
4.2.1.2 FER of the Schemes under Combined FE and DoS attack 50
4.2.1.3 FER of the Schemes under Combined FE and Spoofed DoS attack 51
4.2.2 Performance Evaluation 52
4.2.2.1 Performance under DoS attack 53
4.2.2.2 Performance under concurrent FE traffic and DoS attack 53
4.2.2.3 Performance under concurrent FE traffic and Spoofed DoS attack 54
4.2.3 Evaluation of Convergence time 56
4.2.3.1 Convergence under DoS attack 56
4.2.3.2 Convergence under Concurrent FE and DoS attack 57
4.2.3.3 Convergence under Concurrent FE and spoofed DoS attack 57
4.3 Attack path detection results 59
4.3.1 FER Tests Results 59
4.3.2 Returned Attack Path Correctness Tests Results 60
4.3.3 Convergence Time Tests Results 61
4.4 Quantified Comparison of Results 62

4.5 Discussions 62
4.6 Packets Required for Attack Path Reconstruction 63

CHAPTER FIVE

CONCLUSION AND RECOMMENDATIONS

5.1 Summary 64
5.2 Conclusion 64
5.3 Significant Contributions 65
5.4 Recommendations for Further Work 66
References 68

List of Figures

Figure 2.1**:** Different DoS Attack Mechanisms 8
Figure 2.2: Screenshot of Wireshark during UDP flood attack 10
Figure 2.3: Probability distribution of IP addresses in DDoS and FE traffics 12
Figure 2.4: The algorithm of DoS attack source IP traceback 18
Figure 2.5: Shark Smell Optimization Algorithm (SSOA) Flowchart 25
Figure 2.6: A randomly connected network 26
Figure 3.1: Comparing parameters values in x_0 with flow records, $x_{e,k}$, detected on a link. 43
Figure 3.2: Attack path reconstruction by SSOA-DoSTBK. 45
Figure 3.3: Flowchart of the Proposed SSOA-DoSTBK 46
Figure 4.1:Results of FER for SSOA-DoSTBK and ACS-IPTBK under DoS 50
Figure 4.2: Results of FER for SSOA-DoSTBK and ACS-IPTBK under DoS and FE 51
Figure 4.3: Results of FER tests for SSOA-DoSTBK and ACS-IPTBK when FE traffic surges occurred during spoofed DoS attack source traceback 52
Figure 4.4: Performance of SSOA-DoSTBK and ACS-IPTBK for DoS attack tracing without FE 53
Figure 4.5: Performance of SSOA-DoSTBK and ACS-IPTBK when tracing DoS attack and there were traffic surges due to flash event in the network 54
Figure 4.6: Performance of SSOA-DoSTBK and ACS-IPTBK for spoofed DoS source tracing when FE was present in the network 55
Figure 4.7: SSOA-DoSTBK and ACS-IPTBK convergence time for tracing DoS attack when there is no FE 56
Figure 4.8: SSOA-DoSTBK and ACS-IPTBK convergence time for FE and DoS attack..... 57
Figure 4.9: SSOA-DoSTBK and ACS-IPTBK convergence time for FE and Spoofed DoS attack. 58

List of Tables

Table 2.1:Parameters for generating random topologies based on Waxman's scheme..... 28
Table 2.2: The discernment policy parameters for attack flow identification 29
Table 3.1: Weight values assigned to discrimination parameters..... 40
Table 3.2: Representation of the Discrimination Dataset 40
Table 3.3: Parameters values in x_0 as obtained from the detected attack packets..... 41
Table 3.4: Sample flow records for two different traffics on a link connecting node j 42
Table 4.1: SSOA-DoSTBK results compared to ACS-IPTBK..... 62

List of Appendices

Appendix A-I: C++ Code listing for SSOADOSTBK Header file 73
Appendix A-II: C++ Code listing for SSOADOSTBK C-plusplus file 74
Appendix A-III: OTCL Code listing for calling SSOADOSTBK TCL file 78
Appendix A-IV: C++ Code listing for ACS-IPTBK Header file 79
Appendix A-V: C++ Code listing for ACS-IPTBK C-plusplus file 80
Appendix A-VI: C++ Code listing for ACS-IPTBK C-plusplus file (Random number generator) 89
Appendix A-VII: OTCL Code listing for calling ACS-IPTBK TCL file 92
Appendix A-VIII: The Main OTCL Code listing for IP Traceback process: TCL file 94
Appendix A-IX: Data for SSOA-DoSTBK and ACS-IPTBK False Error Rate Measure.... 102
Appendix A-X: Data for SSOA-DoSTBK and ACS-IPTBK Convergence Measure 105
Appendix A-XI: Data for SSOA-DoSTBK and ACS-IPTBK Path Correctness Measure... 108

ABBREVIATIONS

ACO	Ant Colony Optimization
ACS-IPTBK	ant colony system algorithm for IP traceback, by Wang *et al*. (2016)
AODV	Ad-Hoc On-Demand Distance Vector
AWK	A UNIX based programming language for text processing that can be used for data extraction and reporting
BGP	Border Gateway Protocol
BOTNET	Network of compromised computers controlled by an attacker
CAIDA	Centre for Applied Internet Data Analysis
CBR	Constant Bit Rate
CCN	Completion Condition Number
CNN	Cable News Network
CPU	Central Processing Unit
DARPA	Defence Advanced Research Projects Agency
DE	Differential Evolution
DPM	Deterministic Packet Marking
DPPM	Dynamic Probability Packet Marking
EA	Evolutionary Algorithm
EEM	Extended Entropy Metric
E-LDAT	A lightweight DDoS flooding attack detection and IP traceback scheme by Bhuyan *et al*. (2016)
EPDA	Explicit Packet Dropping Attack
ETT	Efficient Traceback Technique
FAR	False Acceptance Rate
FER	False Error Rate
FIFA	Fédération Internationale de Football Association
FRR	False Rejection Rate
FTP	File Transfer Protocol
GA	Genetic Algorithm
GT-ITM	Georgia Tech Internetwork Topology Models
HCF	Hop-Count Filtering
HSA	Harmony Search Algorithm

ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IGA	Improved Genetic Algorithm
IP	Internet Protocol
IPDA	Implicit Packet Dropping Attack
IPSO	Iteration Particle Swarm Optimization
ISI	Information Sciences Institute
ISL	Inter-Switch Link
ISO/IEC	International Organization for Standardization and the International Electrotechnical Commission
MAC	Media Access Control
MANET	Mobile Ad-hoc Network
MOD	Mark on Demand
MPDU	MAC Protocol Data Unit
MPLS	Multiprotocol Label Switching
NAM	Network Animation Module
NSF	National Science Foundation
NV	No Value
OTCL	The Object-Oriented version of TCL
PC	Personal Computer
PDA	Packet Dropping Attacks
SOPA/PIPA	Stop Online Piracy Act / Protect IP Act
PPM	Probabilistic Packet Marking
PSO	Particle Swarm Optimization
PSOTVAC	PSO with Time-Varying Acceleration Coefficients
RAM	Random Access Memory
RFC	Request for Comments
SGB	Stanford Graph Base for creating topologies
SRD	Simulated Rain Drop
SSOA	Shark Smell Optimization Algorithm
SSOA-DoSTBK	SSOA based DoS attack source detection scheme
SWT	Subuanahu WaTa’ala, Islamic words for praising Allah

SYN	TCP session initialization request packet
TCP	Transmission Control Protocol
TTL	Time-To-Live
UDP	User Datagram Protocol
UK	United Kingdom
USC	University of Southern California
VINT	Virtual Inter-Network Testbed
VLAN-ID	Virtual Local Area Network Identification code
WBAN	Wireless Body Area Network

CHAPTER ONE

INTRODUCTION

1.1 Background on Network Attacks

Network attacks are cybercrimes. It include unauthorized practices such as use of restricted online assets without permission, stealing or gaining unauthorized access into a system, exposing private resources, or malicious disabling or altering or destroying services of a system on the network (ISO/IEC, 2009). Computer network is now involved in most of human day-to-day activities because it makes the way things are done easier. The need for adequate security in computer networks is a rapidly growing area of interest because of the increasing reliance on the networks and the new networks attacks that are springing up at an alarming rate. Attacks on computer networks have serious effects on business and economy because the networks carry large volume of data that are the main focus of business executives for making business decisions. Also, government and security establishments, including military, rely on the data on the networks for making vital decisions and strategical planning. Because of the relative importance of computer networks in vital areas of human endeavours attack on it have direct or indirect impacts on many people. Denial of Service (DoS) attack is a prominent network attack. DoS is not used to steal, eavesdrop, bridge privacy, or compromise data integrity on a system rather it is used to deny victim access to their own network and clients lose transactions. DoS attack and its variants are the largest ravaging network problems. It is identified in literature as the most powerful damaging attacks used to harm a business or organization (Mary & Begum, 2017). In recognition of the serious setbacks that cybercrimes are causing to humanity different countries of the world, including Nigeria, have enacted laws and policies to fight the scourge of cyber-attacks. Examples are the United States Stop Online Piracy Act and Protect IP Act (SOPA/PIPA)

(Schmitz, 2013), The UK Data Protection Act (Data Protection Act, 1998), and the Nigerian cybercrime act 2015 (Cybercrime Act, 2015). Resolving DoS attacks requires identifying its perpetrators and engaging legal battle against the perpetrator to serve as deterrent and to be able to compensate the victim. Successful legal battle can only be achieved based on proven infallible facts used to establish criminal offence against a perpetrator. Network forensic professionals use Internet Protocol (IP) traceback tools to acquire network data that can be used as fact about an attack and also detect the source of the attack.

Denial of service (DoS) attack is a type of cybercrime that require IP traceback scheme specifically designed to take into consideration its intricate attributes and discriminate it from normal transactions on the network that are transmitting large data which may be symptomatically comparable to DoS (Bhandari *et al.*, 2016). A normal network traffic scenario known as flash event (FE) is very similar to Distributed DoS (DDoS) attack, which is a variant of DoS. Flash event (or flash crowd (Bhandari *et al.*, 2016)), refers to a situation whereby a circumstance arouses interest of a majority of network users toward accessing a particular network resource on a server. A practical example of flash event when legitimate traffic overwhelmed the server is the case of 1998 FIFA hosting website that experienced more visitor than its capacity (Chawla *et al.*, 2016). Both flash event and DDoS attack generate heavy traffic from different sources to a particular server. File download manager software that is capable of breaking a download into different segments and use multiple threads to download the segments concurrently is similar to a simple DoS attack. Depending on the size of the traffic generated by the download process, it may affect the services of the server like a DoS attack. High packets flow traffic that is caused by flash event can be distinguished from a DoS attack by studying some characteristics of the traffic. Flash event characteristics like rate of request from the same source IP address to see the timing between

request packets arrival, the sizes of request packets and their contents, and relation between packets will be different from that of DoS attack that is automated (assuming no spoofing). Also, examining the characteristics of the source nodes collectively in terms of the physical distribution of source nodes based on the IP, and the randomness of packet generations between nodes may show a correlation between packets from different nodes to be more apparent. Other characteristics that may be examined to differentiate FE and DoS are packets traffic features including delays, throughput, packets sequences and entropy, and their randomness to deduce if they are from the same node or different nodes (Mohamed *et al.*, 2018). Most of the characteristics mentioned are applicable to DoS attack detectors. IP traceback scheme needs a more dynamically adaptable approach than the detector and thus require other features of the traffic flows. This is to ensure that it is robust against different possible traffic flow surges along the edges of the attack path.

1.1.1 Background on Internet Protocol Packets Source Detection

The process by which data packets are traced back to their source using available information on the packets, e.g. source IP address, is called IP traceback. IP traceback technique employs a mechanism for storing routing path information of the packets like marking schemes (Bhavani *et al.*, 2015; Suresh & Ram, 2018), so that it can later be used to trace the packet back to its source from the receiving end. IP traceback can use a single packet (Malik & Dutta, 2017), or many packets (Saurabh & Sairam, 2016), to acquire adequate information that can be extracted from them for the traceback. If the source information on a packet is correct the IP address in the packet header can simply be used with applications like Telnet, tracert, traceroute, or ping together with IP geolocation tool, e.g. IP locator, to implement IP traceback to detect the node where the packet originated. IP traceback become challenging when the IP spoofing and concealment of source address are employed by attackers to hide their identity (Bhuyan *et al.*, 2016), as often done in DoS attack. IP spoofing is the process by

which other IP address than the true IP address of the source of the packet is used as the packet's source address in the packet header. The IP address used may belong to a trusted computer (Daya, 2013), or just taken arbitrarily. This practice is usually carried out by attackers to conceal their identity or impersonate the owner of the address used (Tiwari *et al.*, 2014).

1.2 Significance of Research

IP traceback is a tool that is used by network forensic professionals to acquire data about an attack and detect the source of a network attack. Accurate detection of source of attack is essential to prevent further malicious transmissions from the same source or expose the perpetrator for the purpose of taking appropriate actions as may be necessary. Many IP traceback schemes for detecting the source of DoS attacks have been reported in literature but they do not have the facility for differentiating FE flow from DoS attack flow. This may cause them to mistakenly identify FE traffic paths as attack path during traceback process. This research work proposes an IP traceback scheme for acquiring accurate data about an attack and detect genuine source of the attack by avoiding other network traffics, including flash event traffics, which may cause false error in its results. This will ensure acquisition of infallible data about the attack for detecting the source and enhance taking appropriate actions against the real perpetrator of the attack.

1.3 Statement of Problem

The problem that SSOA-DoSTBK solved is the difficulty to distinguish the sudden surge in traffic flow of normal network usually caused by a flash event from the DoS attack traffic which can be misleading to IP traceback mechanisms during traceback process. There are DoS attack detectors that are capable of differentiating DoS traffic and normal traffic with large flow but existing IP traceback schemes with this capability is not known to the

researcher as at the time of this research. The existing IP traceback solutions that are based on nature inspired algorithms used flow-based approach to detect the edges that attack packets followed to reach the victim. In some cases, normal network transaction can involve large flow of packets on a segment on the attack path, e.g. File Transfer Protocol (FTP) uploading or downloading large files, or large number of users accessing a resource on a server at the same time. It will be difficult for a flow-based mechanism to differentiate the two flows on a shared part of a multipath transmission and can cause false alarm. This can cause a network forensic investigator to acquire incorrect facts that cannot be used by a cybercrime legal practitioner to convince the jury and establish a criminal case against the attacker. SSOA-DoSTBK employs hop-by-hop based search and used discernment rules to mitigate the challenge.

1.4 Aim and Objectives

The aim of this research work is the development of an internet protocol traceback scheme for denial of service attack source detection.

The followings are the objectives of this research work;

1. To develop the SSOA based DoS attack source detection scheme called the SSOA-DoSTBK
2. To simulate the SSOA-DoSTBK used for implementing the discernment policy using Network Simulator version 2 (NS2)
3. To compare the SSOA-DoSTBK with another IP Traceback scheme that is based on a nature inspired algorithm. The modified ant colony system algorithm for IP traceback (ACS-IPTBK) scheme developed by Wang *et al* (2016) was selected for the comparison because it is based on a nature inspired algorithm that is commonly used for IP traceback, the ant colony optimization (ACO) algorithm. There have been trend

of improvements on using ACO for IP Traceback from ordinary ACO, Ant System, Ant Colony System, up to the ACS-IPTBK as reported in literature. The comparisons were based on False Error Rate (FER), convergence time, and ability to differentiate sudden surge in normal traffic flow from attack flow and detection of source of spoofed IP packets as performance metrics. The FER includes False Acceptance Rate (FAR), and False Rejection Rate (FRR).

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

This literature review comprises of the review of fundamental concepts and the trend of the previous works that are related to this research with a view to gain the knowledge of the fundamental concepts of the research area.

2.2 Review of Fundamental Concepts on DoS attack IP traceback

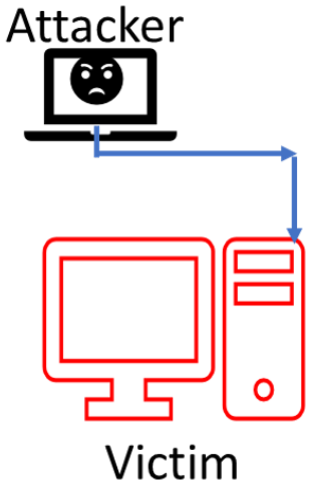
The fundamental concepts of the DoS attack IP traceback and the SSOA are discussed here with related previous research works that are relevant to this work.

2.2.1 The DoS Attack and its Variants

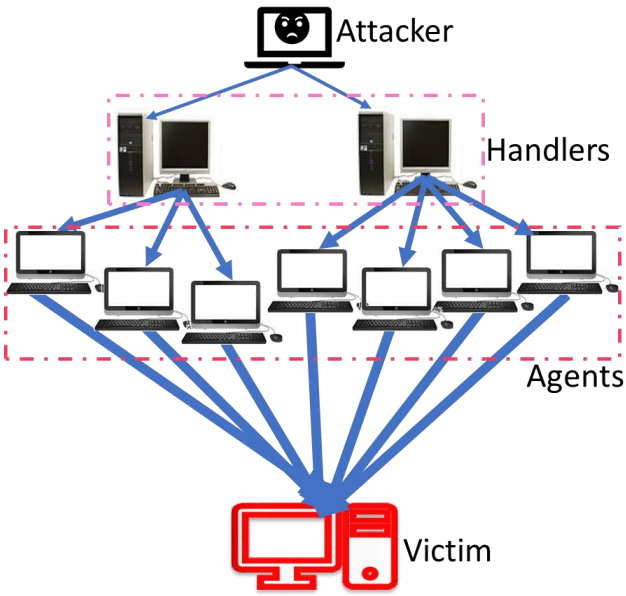
Denial of service attack is one of the dreadful network attacks that basically exhaust the computing resources of the victim. It denies the legitimate clients of the victim access to the desired services as provided by the victim's system at the desired time. This may subject the victim to great loss. There are several mechanisms to carry out DoS attack, prominent among them are; the simple DoS attack, the reflective DoS attack, the Coordinated DoS attack, and the distributed DoS attack, as illustrated in Figure 2.1.

The simple DoS attack in Figure 2.1a is so called because it usually involves one attacker and a victim but it is fundamental to all other DoS attacks. Identification of the original malicious node where the attack was initiated is often not easy because the attackers usually spoof or conceal their identity in the attack packets header.

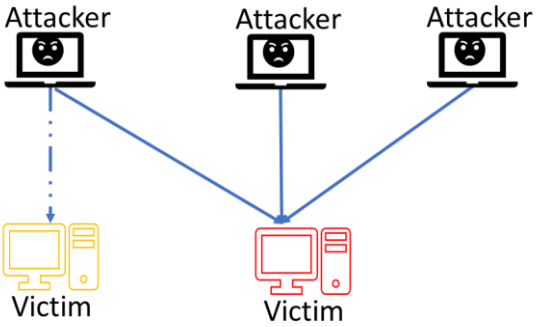
Distributed Denial of Service (DDoS) shown in Figure 2.1b is a DoS attack carried out by conspiring multiple compromised hosts in the network to attack a victim and bring down its operations for a reasonable length of time (Maheshwari & Krishna, 2013).



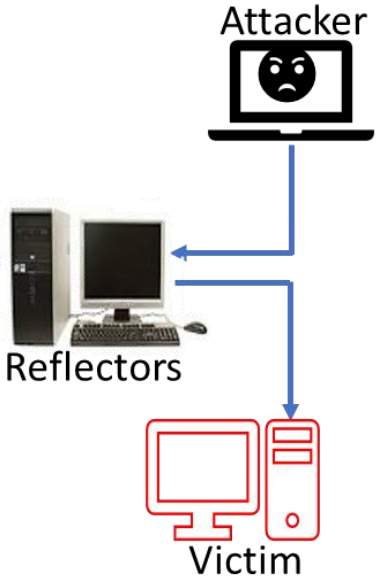
(a) Simple Denial of Service



(b) Distributed Denial of Service



(c) Coordinated Denial of Service



(d) Reflective Denial of Service

Figure 2.1**:** Different DoS Attack Mechanisms

Figure 2.1c shows coordinated denial of service which is a DoS attack in which malicious nodes sequentially or concurrently target different network vulnerabilities in a victim's system and use it to disrupt its network communication (Ahmed & Fapojuwo, 2016; Tan *et*

al., 2010). Reflective denial of service, Figure 2.1d, is a DoS attack whereby the attacker sends crafted data packets spoofed with the victim's IP address as source address to compromised nodes, called reflectors, making reflectors to send replies to the victim (Rasti *et al.*, 2015). Other mechanisms except the simple DoS usually involves attacker conspiring with intermediary systems compromised unknowingly and unwillingly by the attacker to magnify the effects of the attack on the victim's system. Each of such compromised systems is called bot and all bots in an attack are referred to altogether as BOTNET. The higher the number of the bots used the more complex the attack because the attacker may shield itself with the compromised systems and make it difficult to trace the attack path back to the original attacker. Many big and well-established organisations in IT industry have reported been attacked with a form of DoS attack at one time or the other. Among the reported cases is a computer that was taken out of service for 2 days in the University of Minnesota in 1999, in year 2000 Yahoo! Inc., Amazon, Buy.com, Cable News Network (CNN), and eBay all experienced DoS attacks that either caused them to stop functioning completely or significantly slowed down their operations for a period of time (Bhuyan *et al.*, 2014).

There are two broad categories of DoS attacks which are high-rate and low-rate DoS (Mohamed *et al.*, 2018). The high rate DoS attacks are flooding attacks including SYN flooding, Internet Control Message Protocol (ICMP) message flooding, Smurf attack, Transmission Control Protocol (TCP) attack, etc. (Zargar *et al.*, 2013). They generate large volume of traffic packets at a high rate and can consume as much as 600Gbps of the network bandwidth (Mohamed *et al.*, 2018). Figure 2.2 shows screenshot of traffic data captured with Wireshark during User Datagram Protocol (UDP) flood attack (Stone, 2000).

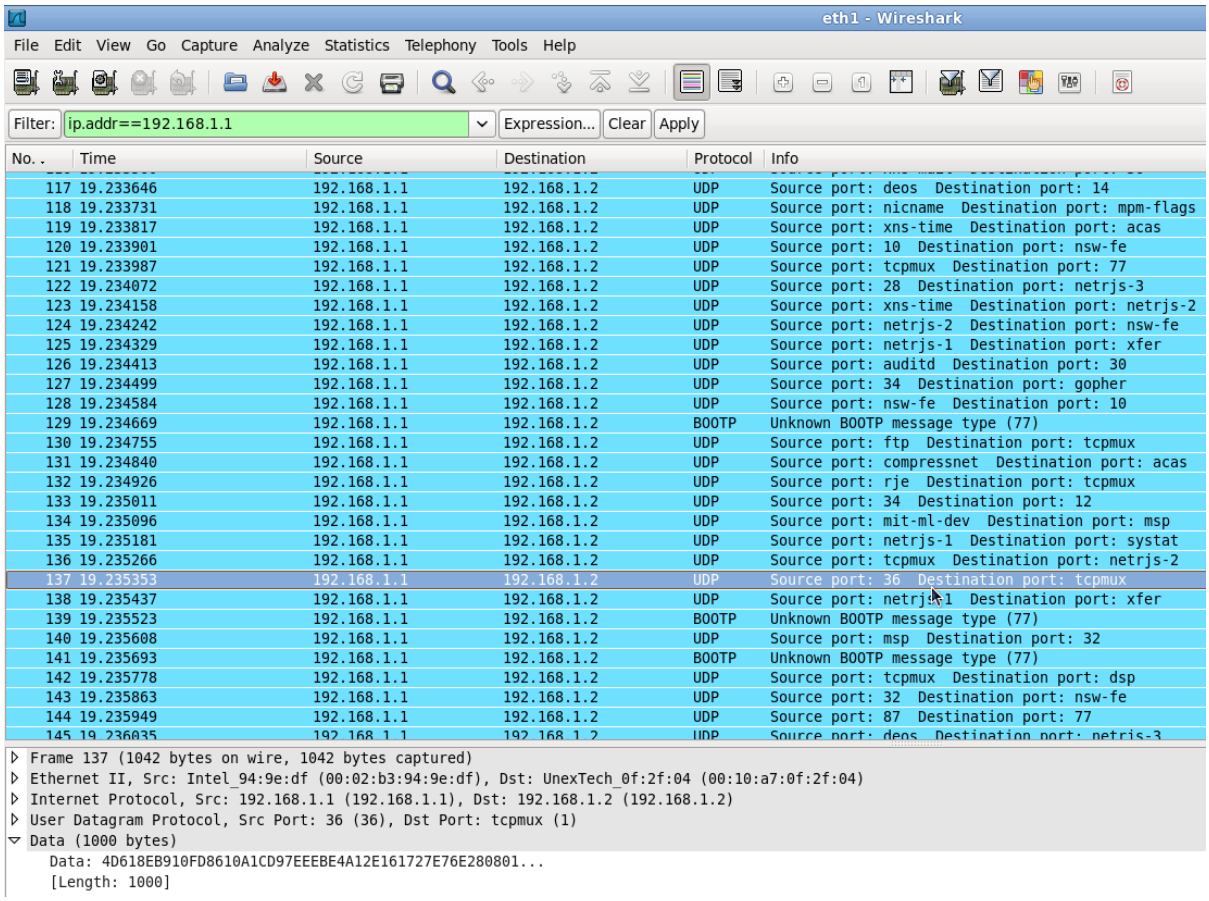


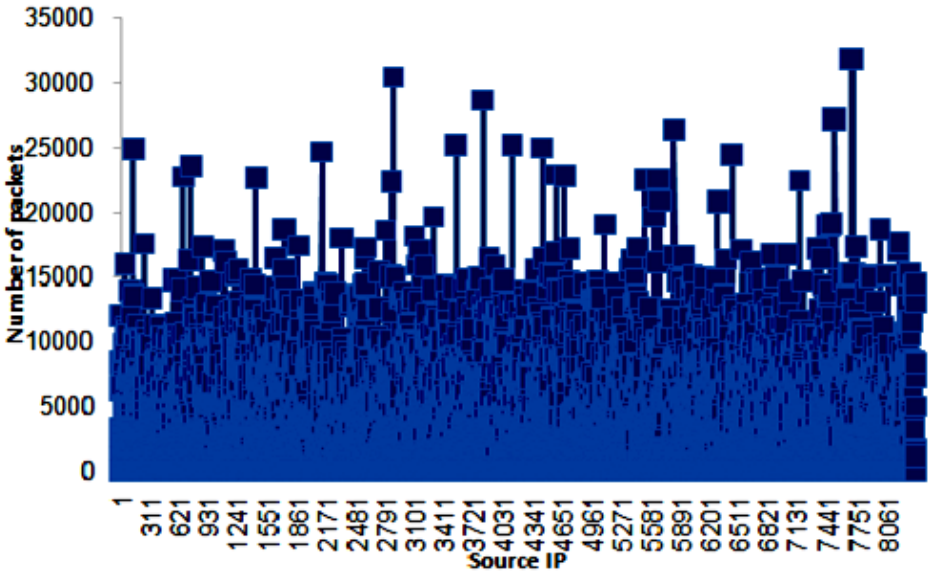
Figure 2.2: Screenshot of Wireshark during UDP flood attack

Some characteristics of DoS attacks that can be deduced from Figure 2.2 include the time of arrival of the attack packets as can be seen under Time column in the screenshot. The arrival times are very close. For example the time interval between the highlighted entry and the entry immediately below it is 8.4×10^{-5}sec or 84 µsec. The packets stream is coming from the same source to the same destination as can be seen under columns for Source and Destination hosts in the screenshot. This interval is too small for host receiving floods of continuous packets arrival from the same source on a network. It will not allow packets from other source to be accepted by the destination host. The protocol used as indicated under Protocol column in the screenshot is UDP. The attack targeted various ports on the victim to deny several transactions destined for those ports connection.

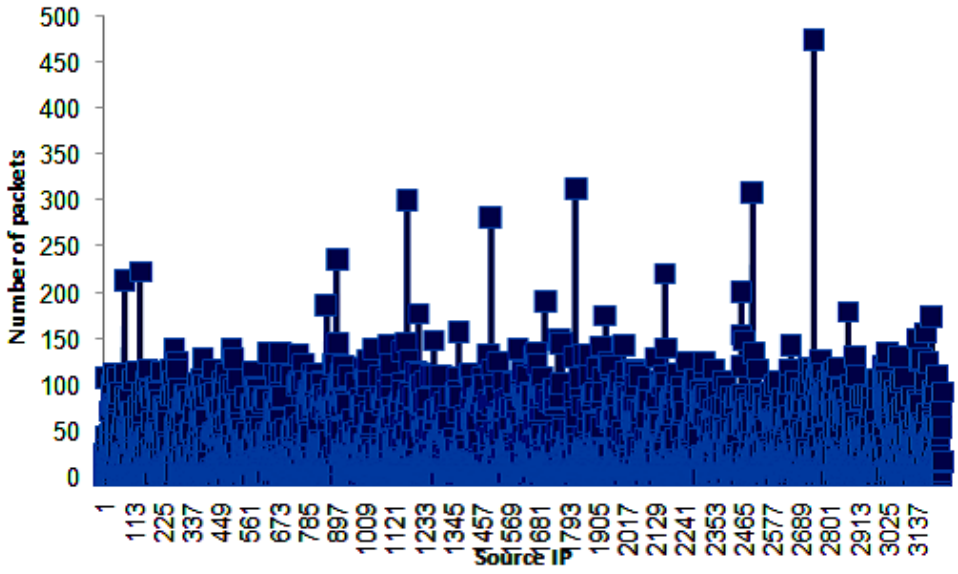
While most of the DoS attacks overwhelm the victim's system with packets of data there are others that do the damage with a very few crafted packets, almost undetectable by traditional DoS detection systems, e.g. slow-rate TCP DoS attack (Qian *et al.*, 2017). Packet Dropping Attacks (PDA), Explicit Packet Dropping Attack (EPDA) and Implicit Packet Dropping Attack (IPDA), are another type of DoS attacks where the attackers do not overwhelm the victim's computing resources but exploit the Mobile Ad-hoc Network (MANET) routing protocol called Ad-Hoc On-Demand Distance Vector (AODV) to maliciously influence the victim to use the attacker's machine as a packet routing node. The attacker will just be dropping all packets from the victim without routing them and deny the intended receivers of the packets of the data (Hemant Pareek, 2014). Another low-rate DoS is the selective packet dropping attack whereby the attacker identifies high priority packets that are important for the network communication, e.g. frequency-hopping synchronisation packets in a frequency-hopping network or packets carrying white spaces lists in cognitive-radio network. The attacker will be dropping the high priority packets selectively and prevent optimum performance of the network (Balachander *et al.*, 2017).

2.2.2 Flash Event

Flash event (FE) is a term used to describe a sudden increase of number of users trying to access resources on a host causing heavy traffic on the network (Dhingra & Sachdeva, 2014). Heavy traffic on computer networks is caused by various reasons and manifests in different ways, e.g. bottleneck and increased latency due to network congestion. Among other things that can cause high traffic on computer networks is deliberate injection of crafted large stream of data into a network which is known as attack, e.g. DoS attack. Flash event can be mistaken for DoS attack because of some similar features of the two. Figure 2.3 (Chawla *et al.*, 2016), shows the similarities in the flash event traffic and the DoS attack traffic. The



(a) Probability distribution of IP addresses in FE of FIFA 1998 Dataset



(a) Probability distribution of IP addresses in DDoS traffic of CAIDA Dataset

Figure 2.3: Probability distribution of IP addresses in DDoS and FE traffics

dataset used for the graphs were Centre for Applied Internet Data Analysis (CAIDA), and Fédération Internationale de Football Association (FIFA) 1998 website datasets. Comparing Figure 2.3a and Figure 2.3b shows the striking semblances between the DoS and flash event. It can also be seen in the figures that there are far more packets in the DoS traffic than the

flash event traffic but their patterns are very similar. The number of flash event packets ranged from 0 to 500 generated by less than 3500 distinctive IP addresses in Figure 2.3a. However, those of DoS attack ranged between 0 and 35000 packets generated by less than 8200 distinctive IP addresses in Figure 2.3b. Based on the observations from Figure 2.3, it can be inferred that the entropy is wider in Figure 2.3a of flash event than in Figure 2.3b of DoS attack. This is because averagely, more packets are generated per IP address in Figure 2.3b than in Figure 2.3a. In the case of the DoS attack the attacker has good knowledge of the system it is attacking and deliberately overloads it but flash event is just incidental caused by users' attraction to particular online items.

Some of the major differences between flash event and DoS attack are;

1. In contrast to the purpose of DoS attack, flash event is not perpetrated to inflict bad consequences on the affected host but happens as a result of large visitors being attracted to the item on the host.
2. The attacker knows and target the major metrics that can impact on the performance of the network it is attacking. He understands the impact of its action on the network. Users causing flash event are unaware of the effect of their activities on the network.
3. Correlation between DoS traffic packets is stronger than those of flash event traffic packets (Chawla *et al.*, 2016). This is evident because flash event traffic packets generation is purely random and highly unpredictable but DoS attack packets are generated automatically with or without deliberate randomization, depending on the attack.

Flash event can cause significant latency on the network that may disrupt the services of the affected host (Sachdeva & Kumar, 2014), just like a DoS. If the flash event traffic is overwhelming the host may be unable to respond effectively to other transactions.

Since flash event by itself is not an unwanted occurrence but just that it may have undesirable effects on the system identifying the individuals at its sources is unwarranted. The solution to it is better planning and upgrading of the capacity of the system to be able to cope with such

situations. Thus, efficient IP traceback scheme should be able to differentiate flash event traffic from DoS attack traffic during the traceback process.

2.2.3 DoS Attack IP Traceback Methodologies

The identity of the sender of a data packet is vital for its traceback using IP traceback method but IP spoofing and concealment which is used by attackers to delude the victim poses challenge to using IP traceback for identifying attackers. Because of the malicious intention of attackers, they often falsify the true identity of the originating source of attack packets by spoofing it. Therefore, sophisticated algorithms are developed to implement IP traceback for detecting the source of packets with falsified header information. There are various methods for implementing IP Traceback scheme (Murugesan *et al.*, 2014). Some of the common methods available in the literature are highlighted in the following.

Packet marking is one of such techniques used for storing information that could be used to reconstruct the path to the source of the packet (Saurabh & Sairam, 2016). Packet marking may be Deterministic Packet Marking (DPM) or Probabilistic Packet Marking (PPM). In DPM, the first ingress edge router marks every packet passing through it with the IP address of the router. But the PPM mark the packet irregularly based on probability value for the marking set for it. PPM requires large number of packets and DPM has large convergence time.

Link testing is also a method used for IP Traceback which could be either input debugging or control flooding. Input debugging is whereby the victim creates signature of the detected attack. Based on the known topology of the network the victim search for the signature in the upstream routers up to the attack source in real-time while the attack is still active. The challenge of this method is that it requires the coordination from the network administrators

and thus the network administrators must understand its implementation. Because it requires human intervention it will be slow and may not even complete traceback if the coordination failed or the topology changes (Murugesan *et al.*, 2014). In the case of control flooding each incoming links of selected node is flooded iteratively with large bursts of traffic. The link through which the attack came is inferred from the response rate pattern from the flooded links and is used to select the next node and continues up to the attack source. This method causes network overhead problem.

ICMP Traceback is method whereby a special ICMP message is sent together with a packet that is sampled. The message contains the routing router information including its neighbouring forward and backward links routers on the path to the destination and source. It is sent at random based on a low probability for sampling routed packets. It creates additional overhead on the network.

IP Logging traceback method stores the information about the routed packets, like packets digest, signature, and fields of IP header on all routers or some of the routers within the domain that routed the packets. It required large memory space for storing most packets of flows that passed through it.

Some IP traceback schemes for DoS attacks are developed based on the same or similar working principles of those mentioned or other IP traceback methods. Among them are Efficient Traceback Technique (ETT) for detecting DDoS attack in cloud-assisted healthcare environment (Latif *et al.*, 2016), and the scheme using completion condition to determine the minimum required packet for IP traceback (Saurabh & Sairam, 2013), they both used packet marking method. The improved Ant Colony Optimization (ACO) algorithm-based system for solving IP traceback proposed by Wang *et al.* (2016) used a nature inspired algorithm for

flow-based search. Nature inspired algorithms are developed for complex computing where exact results may be hard to achieve or not determinable due to availability of just a few input parameters. Ant colony system is a swarm algorithm that involves multiple agent ants engaged in a search process at the same time. Its inherent parallelism and interdependent sequences of random decisions were exploited by ACS-IPTBK to find the most probable attack path based on the path with the highest flow between the victim and the attacker. Each ant does not necessarily traverse the full length of every possible paths but the most probable attack path is constructed from the combination of the most probable links returned by ants based on the high traffic between their end nodes (Wang *et al.*, 2016). This approach does not address the challenges of finding the source of spoofed attack packets when there is flash event that may cause other genuine transactions to generate heavy traffic on the network along shared edges that connect multiple paths. This is because when the source of the spoofed attack packets is hidden and detection is based only on heavy flow of packets other legitimate transactions that generate heavy flows of packets may be mistaken for attacks and edges involved will be included in attack path. Shark Smell Optimization Algorithm (SSOA**)** is another nature inspired optimisation algorithm that was developed to be used for obtaining optimal results when only a few input parameters are available. Among assumptions on which SSOA was developed are that there is an injured fish in the sea, and the fish is ejecting blood which shark is using to trace it (Abedinia *et al.*, 2014). By analogy, the shark is the victim using SSOA-DoSTBK to trace source of an attack. The injured fish is the DoS attacker and the sea is the network. The blood is the attack packets and the attacker is assumed to be sending attack packets when the IP traceback is going on, in the case of real-time traceback. Shark execute local search at each position along the path by moving round in circle to exploit all the points around the position and detect the point with highest smell at the position to determine the direction to follow to the next position (Abedinia *et al.*, 2014), in a

similar way to hop-by-hop search of IP traceback. The closeness of the SSOA search process to the tracing back of the DoS attack packets to the source make it considered suitable for improving IP traceback for detecting source of DoS attack.

IP traceback tools use information deduced from the data obtained and other relevant information available on the network to trace the source of the data. Network data are very volatile because most of the important data that may be vital for traceback or use as legal evidence often get overwritten within short time. This necessitated the need for a sophisticated IP traceback tool that can collect reliable network data and detect the attack path accurately, within available time constraint.

2.2.4 DoS Attack Source Detection Process

DoS attack IP traceback is an investigative process that is usually carried out to identify an attacker carrying out malicious operation to disrupt normal flow of the victim's services in a network. DoS attack source detection begins when a DoS attack is detected. Thus, it begins from the point of detection tracing back to the attacker. The victim tries to find the host where the malicious packets were originating from by exploring the available flow paths that leads to it. It compares the packets flow on the paths with that of the attack packets to be able to discover the path that the attack packets follow to reach it. Thus, it reconstructs the same path back to the attacker. One of the metrics often used for the path reconstruction include the flow traffic whereby edges with higher packets flow are selected as probable attack path (Hamedi-Hamzehkolaie, *et al*, 2014). There are mechanisms that keep records of known compromised hosts and query the hosts if any of them sent packets to the victim during the time that the attack occurred (Chen *et al.*, 2006). Some mechanisms use NetFlow tools on the router to traceback the DoS attack source (Bhuyan *et al.*, 2016). Figure 2.4 shows an algorithm of DoS attack source detection.

GET attack packets attributes from an attack detector

SET traceback rules according to the traceback scheme design

WHILE Not HALT instruction encountered REPEAT the following steps:

FIND edges that carry ingress traffic to this node

APPLY traceback rules to the identified ingress edges

SELECT the most probable ingress edge as part of the attack path

ADD the edge to the attack path edges in Tabu list

MOVE to the node on the other end of the just selected edge

IF this node is the host or the closest router to the host of the attack packets HALT

END IF clause

END WHILE loop

RETURN attack path and the attack packets host.

Figure 2.4: The algorithm of DoS attack source IP traceback

2.3 Shark Smell Optimization Algorithm

SSOA was proposed by Abedinia *et al.* (2014). The fundamental concepts of this algorithm were based on the superior ability of shark to find its prey in a large search space, the sea, within a short period of time using its strong smell sense. The algorithm was evaluated by comparing its performance against thirty-two (32) other popular optimization techniques often employ to solve engineering problems, it produced better results. The optimization algorithms it was tested against are: Evolutionary Algorithm (EA), Genetic Algorithm (GA), Improved Genetic Algorithm (IGA), Differential Evolution (DE), Particle Swarm Optimization (PSO), Iteration Particle Swarm Optimization (IPSO), Harmony Search Algorithm (HSA), PSO with Time-Varying Acceleration Coefficients (PSOTVAC), etc. (Abedinia *et al.*, 2014). This metaheuristic algorithm is one of the highly efficient and very fast optimization techniques proposed in recent time that surpasses performances of those thirty-two optimization algorithms it was compared with in many applications. SSOA has

been used, either directly (Abedinia *et al.*, 2014; Ehteram *et al.*, 2017; Gnanasekaran *et al.*, 2016; Noruzi, 2016; Shayanfar *et al.*, 2016) or modified (Abedinia & Amjady, 2015; Ahmadigorji & Amjady, 2016; Ghaffari *et al.*, 2016; Setayeshi *et al.*, 2017), or hybridized (Abedinia & Amjady, 2016), or improved hybridized version (Ahmadigorji & Amjady, 2016), to solve many engineering problems and its applications have confirmed its effective performances.

Sharks belong to the subclass of Elasmobranch fishes known as Chondrichthyes, in Greek, it means *cartilage fish*, i.e. fish with skeleton made of cartilage instead of fibre bones. Sharks brains are large and process substantial sensory information. Almost all the sensory information is used for hunting prey which makes sharks great hunters. Among the sensory information most commonly used to detect and locate preys are smell, sound (Abedinia *et al.*, 2016), and electroreception (Kalmijn, 1982; Kempster & Collin, 2011; Montgomery & Bodznick, 1999). Sound and smell are used for remote sensing of presence of a prey; the electrosensory system detects its proximity when shark is very close to it. Shark can detect field as low as 5nV/cm (5 nano-volts per centimetre) at a distance of about 38cm (Kalmijn, 1982). This indicates that shark uses electro-reception to detect close-bye prey. This research is concerned with the shark's smell sense upon which SSOA was developed based on how it is used by shark for hunting.

To model the shark's process of hunting prey using its blood smell sensing noesis the stages in the process are defined with mathematical expressions presented in the following subsections. The assumptions considered for the modelling of the artificial shark by Abedinia *et al.* (2014) are:

1. The injured fish (prey) ejecting the blood into the sea is relatively static compared to shark's movement.

2. The blood is being continuously ejected during the search; the concentration of blood smell is highest at point of its ejection but decreases as it moves away from the source. Thus, shark follows the increasing intensity of the blood smell from the point where it detected it towards the injured fish to find the prey.

3. Only one injured fish ejecting blood is available in the search space at the time of search.

2.3.1 Initialization

The search for a prey begins when smell of the fish ejected blood is detected by the shark. The search behaviour of shark is modelled by the following equations, Abedinia *et al.* (2014).

A set of position vectors consisting of solutions is generated for this initial position of the shark as

$$\left[X_1^k, X_2^k, \ldots, X_N^k \right], \qquad \textit{where } N = \textit{Population Size} \tag{2.1}$$

Each position, X_i^k consists of candidate solutions at stage *k* of the shark movement as

$$X_i^k = \left[x_{i,j}^k, \ldots, x_{N,D}^k \right] \tag{2.2}$$

i = 1 to N, j = 1 to D, where N = population size, and D = number of decision variables. Each X_i^k represents possible position of the shark, and the $x_{i,j}^k$ represent the smells at each point of the position. The initial position of the shark at the beginning of the search is randomly generated.

2.3.2 Scouring

Shark advances from its present position to another position with a stronger blood smell. It

moves from one position to another at a velocity. The different velocities of the shark are generated as

$$V^k = \left[V_1^k, V_2^k, \ldots, V_N^k\right] \quad N \text{ is the population size} \tag{2.3}$$

Each V_i^k corresponds to each X_j^k position. Like the $x_{i,j}^k$ as smells at different positions the corresponding $v_{i,j}^k$ components of the velocity V_j^k that shark moves to different smell concentration points are generated as

$$V_j^k = \left[v_{i,j}^k, \ldots, v_{N,D}^k\right] \tag{2.4}$$

i = 1 to N, J = 1 to D, where N = population size, and D = number of decision variables.

2.3.3 Advancing

The shark initial velocity moving from initial stage to the next is calculated by getting gradient (∇) of the objective function (OF), which is the fitness function of the problem, as $\nabla(OF) = \frac{\partial(OF)}{\partial X}$ and multiply the gradient by weighting factor $\eta_k \in [0, 1]$ for the stage k and a random number with uniform distribution $R1 \in [0, 1]$ to give

$$V_{i,j}^k = \eta_k . R1 . \frac{\partial(OF)}{\partial x_j}\Big|_{x_{i,j}^k} \tag{2.5}$$

In equation (2.5), η_k is used to determine actual velocity of the shark relative to the gradient function. Consequent velocities after the initial velocity are dependent on the previous velocity before it. The dependence is incorporated in the following equation

$$v_{i,j}^{k} = \eta_k .R1. \frac{\partial (OF)}{\partial x_j}\Big|_{x_{i,j}^{k}} + \alpha_k .R2. v_{i,j}^{k-1} \tag{2.6}$$

R2 is another random number with uniform distribution in [0, 1]. The parameter α_k is the inertia coefficient for stage *k* and it belongs to [0, 1].

Shark naturally has velocity limiter that define the ratio of maximum to the minimum velocity it can travel which is 4 using its maximum velocity of 80*km*/*hr* and its minimum velocity of 20*km*/*hr*. The velocity limiter is indicated as β. The shark's velocity at stage *k* is selected as the minimum between calculated value with the gradient function and the weighted value weighted with, β_k, shark's natural velocity limiter at stage k, as, Abedinia *et al.* (2014);

$$| v_{i,j}^{k} | = \min \left[\left| \eta_k .R1. \frac{\partial (OF)}{\partial x_j}\Big|_{x_{i,j}^{k}} + \alpha_k .R2. v_{i,j}^{k-1} \right|, \left| \beta_k .v_{i,j}^{k-1} \right| \right] \tag{2.7}$$

2.3.4 SSOA Exploitation

The new position of shark, Y_i^{k+1}, at every stage is dependent on its previous position, X_i^{k}, and the velocity (V^k) within the time interval (Δ_{tk}) moving from X_i^{k} to Y_i^{k+1}. Thus, the new position is given as

$$Y_i^{k+1} = X_i^{k} + V_i^{k} \cdot \Delta_{tk} \tag{2.8}$$

For convenience Δ_{tk} is set as $\Delta_{tk} = 1$ in equation (2.8)

Shark execute local search at each stage by making circular movement round the position to exploit all the points around the position and detect the point with highest smell at the

position. Shark uses the point with the highest smell to proceed to the next position. The shark exploitation is modelled as:

$$Z_i^{k+1,m} = Y_i^{k+1} + R3.Y_i^{k+1} \tag{2.9}$$

Where $i = 1 \ldots N$, N = population size, and $k = 1 \ldots k_{max.}$ k_{max} is the total number of stages. $R3$ is another random number with uniform distribution in the range [−1, +1], $1 \leq m \leq$ M, M is the number of points in the local search. Points $Z^{k+1,m}$ are in the vicinity of position Y^{k+1}. The point with strongest smell, X_i^{k+1}, is selected as, Abedinia *et al.* (2014):

$$X_i^{k+1} = argmax\left\{OF\left(Y_i^{k+1}\right), OF\left(Z_i^{k+1,1}\right), \ldots, OF\left(Z_i^{k+1,M}\right)\right\} \tag{2.10}$$

M is the total number of points in the local search path, $1 \leq i \leq$ M.

2.3.5 SSOA Exploration

The selected point with the best smell X_i^{k+1} in equation (2.10) is used as the shark's new position.

The shark foraging process continues till stopping criteria is satisfied. Iteration count k is incremented and the steps highlighted in the flow chart in Figure 2.5 are repeated if k is less than k_{max}. Otherwise, X_i^{k+1} is returned as the optimum solution.

2.3.6 Flowchart of SSOA Algorithm Search Process

The key steps in the search process of SSOA are highlighted in Figure 2.5. The algorithm comes in hand for a quick scrutiny of large data with close similarity for selecting the ones with the best fit within a limited time constrain. Like other nature inspired algorithms, the SSOA user-define parameters can be used to control its speed and results accuracy. It is

employed in this work for adroit examination of the traffic flow records on the upstream routers to identify the nodes and edges involved in transmitting the attack flow to the victim. Using the criteria in the discrimination policy, the algorithm is used to ensure that the proposed scheme focuses mainly on the relevant section of the network for the search in order to optimise computation time and resources. Hop-by-hop tracing of IP address is highly efficient but known to involve difficult task, and time consuming to traverse all possible paths. SSOA was used to leverage the hop-by-hop search to return optimum results within shortest possible time. The flowchart of SSOA itemising the steps involved in SSOA search process is shown in Figure 2.5 (Abedinia *et al.*, 2014).

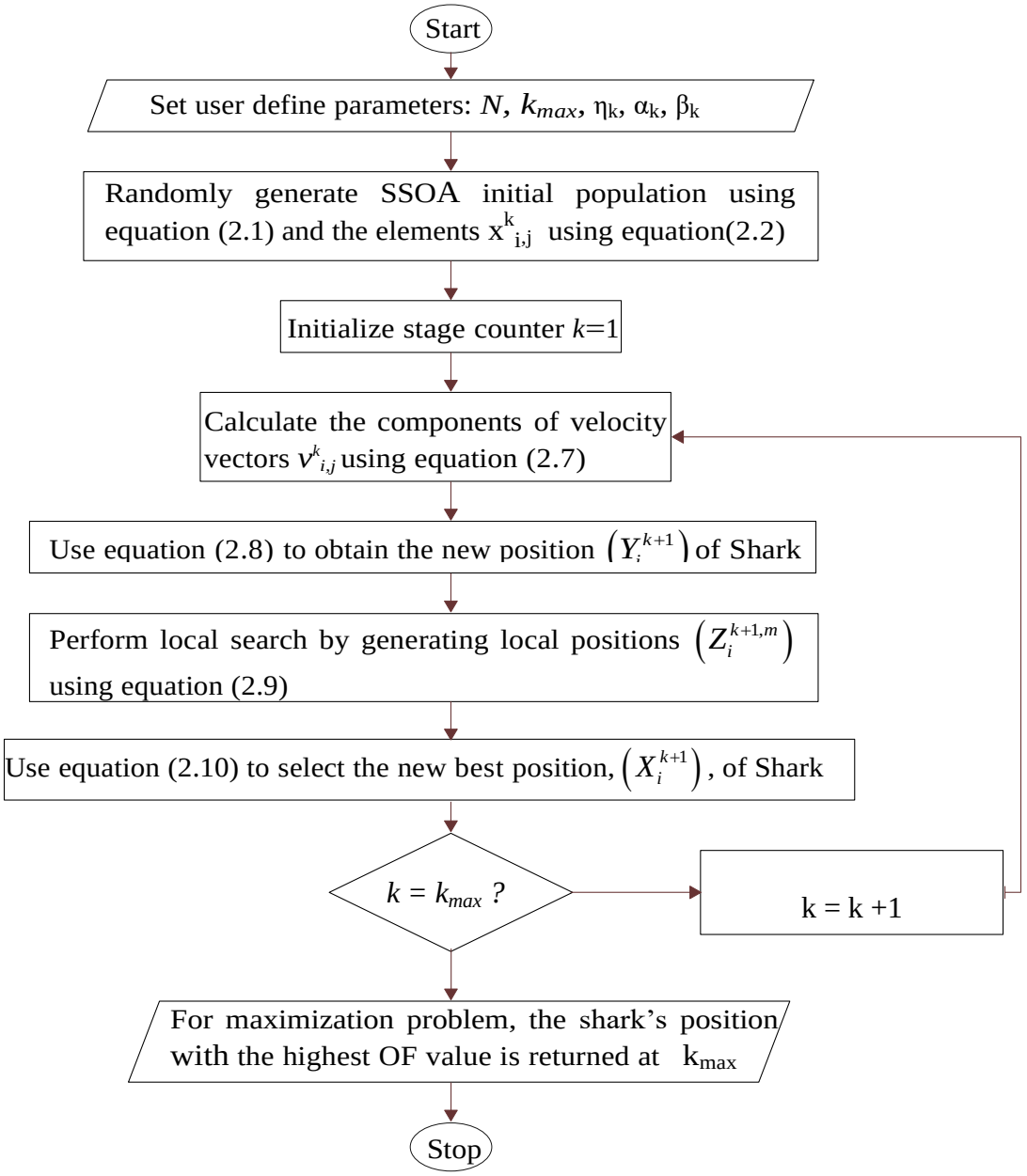


Figure 2.5: Shark Smell Optimization Algorithm (SSOA) Flowchart

2.4 Reconstructing the Network Topology

Directed graph, $G = (V, E)$, was used to represent the network topology where E represents the edges and set of nodes on a path i are represented as $V = [v_i, \ldots, v_{ND}]$. $i = 1$ *to* ND. ND is number of nodes in the network. V_s and V_d are set of source nodes and the sink nodes respectively. Among the nodes one was randomly set to be the attacker (V_{atk}) and another one to be the victim (V_{vtm}). The SSOA-DoSTBK was employed to determine the most probable

path between the V_{atk} and the V_{vtm} which was the attack path. Using experimental network model generator according to Waxman model, N (200 was used) nodes was randomly placed at integer coordinates distributed over a $n \times m$ (20 × 20 was used) meters rectangular area that was the search domain. Adjacent nodes v_i and v_j was connected to form edges with assigned probability as calculated with (Wang *et al.*, 2016):

$$p(i, j) = \eta . e^{\left(\frac{d(i,j)}{L\gamma}\right)} \tag{2.11}$$

$d(i, j)$ is the Euclidean distance between nodes v_i and v_j, L is the maximum possible distance between any two nodes in the network. η and γ are parameters with values in the range [0,1], they are used to vary the graph characteristics. η increases average degree of the nodes and γ increases ratio of the number of long edges to the number of short edges when the parameters are increased. Figure 2.6 shows a simple network with randomly connected nodes. It contains an attacker and a victim, and the attack path through which the attack packets were transmitted is indicated with red colour.

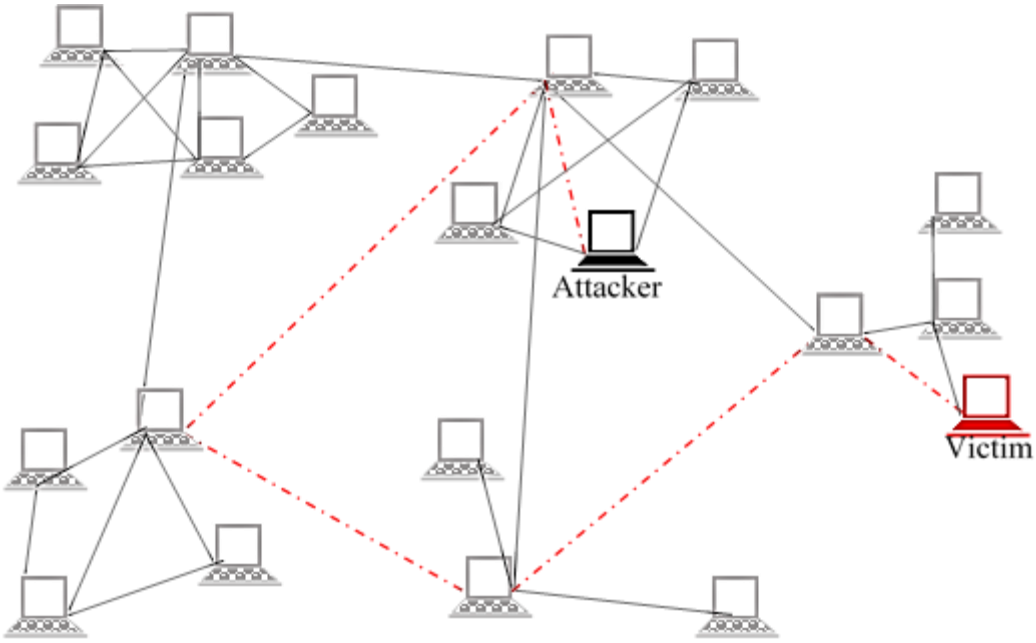


Figure 2.6: A randomly connected network

2.4.1 Implementation of Waxman Topology

Waxman model, also called Waxman graph, is a random network topology model for simulating experimental networks. Waxman graphs introduced by Waxman is a particular class of random graphs for modelling the network topology that can be used for evaluating routing algorithms as an alternative to the Gilbert-Erdös-Rényi (GER) random graphs. It is a more realistic setting for testing networking algorithms (Roughan *et al.*, 2015). The nodes in a Waxman graph are uniformly distributed over a rectangular area. The edges that link the nodes are added through a random mechanism (Equation (2.11)) that determines the probability that two nodes are directly connected. The probability of connection existing between a set of two nodes decreases exponentially as their Euclidean distance increases. As a characteristic of the general class of spatial models, which Waxman graphs belongs to, the connectivity properties of its graph are directly related to the spatial relationships among the nodes. The Waxman random topology was simulated in network simulator 2 (NS2) using random network generator that connects neighbouring edges based on the probability of their connectedness as determined by equation (2.11). The values for the parameters of equation (2.11) are given in Table 2.1 (Wang *et al.*, 2016).

Table 2.1:Parameters for generating random topologies based on Waxman's scheme ($N = 200$, $\gamma = 0.10$)

η	**Max degree**	**Min degree**	**Max Avg degree**	**Max length**	**Avg length**
0.3	9.30	0.00	5.97	36.40	4.28
0.6	13.50	0.10	11.47	39.83	4.32
0.9	17.70	0.90	16.74	37.42	4.31
1.2	22.90	1.60	22.68	40.83	4,32
1.5	26.40	3.20	28.03	41.05	4.32

2.4.2 Determining edges on attack path

Routers log vital information that is useful for attack path reconstruction (Router Data Flow Overview, 2017). Based on the Cisco Systems NetFlow Services Export V9 the Internet Engineering Task Force (IETF) in October 2004 published Request for Comments (RFC) 3954 (Claise *et al.*, 2004), to establish a standard for NetFlow by different vendors and remove interoperability problems. Cisco called their flow monitoring tool NetFlow while Juniper called it J-Flow and Huawei called theirs NetStream. Others branded theirs with different names. The difference is in the name but the capabilities is essentially the same. Logging of some of the information may require to be enabled if they are not enabled by default. This is why cooperation of Autonomous Systems (AS) owners is vital for a successful IP traceback for source detection. Appropriate authority can make policy that will make AS owners cooperate to assist crime investigations. Cisco routers can keep the data packet traffic record shown in Table 2.2 if IP flow-capture is enabled (NetFlow Commands, 2009).

The expression for determining the fitness of an edge been on attack path was developed based on set criteria as explained in the following.

A set of unique parameters of the attack packets flow as shown in Table 2.2 was used to set the discernment policy.

Table 2.2: The discernment policy parameters for attack flow identification

Parameter	Description
ICMP	The value of the ICMP type and code fields from the first ICMP datagram in a flow. This is used to indicate the attack packet type.
IP-ID	Captures the value of the IP header Identification field from the first IP datagram in a flow. All packets in a flow carry the same value in their IP-ID header fields. It is used together with fragment offset number to reconstruct packet fragments
MAC-addresses	Captures the values of the source Media Access Control (MAC) addresses from ingress packets and the destination MAC addresses from egress packets from the first packet in a flow. This parameter only applies to traffic that is received or transmitted over Ethernet interfaces
Packet-Length	Captures the value of the packet length field from IP datagram in a flow.
TTL	Captures the value of the Time-to-Live (TTL) field from IP datagram in a flow.
VLAN-ID	Captures the value of the 802.1q or Inter-Switch Link (ISL)Virtual Local Area Network Identification (VLAN-ID) field from VLAN-encapsulated frames in a flow when the frames are received or transmitted on trunk ports.
Dst IP address	IP address of the destination device. Must be victim address.
Dst Port Msk AS	Destination Border Gateway Protocol (BGP) autonomous system. This is always set to 0 in Multiprotocol Label Switching (MPLS) flows. It must correspond to that of the hop querying its neighbouring hops.
Pkts	Number of packets through this flow switched by this router.
Active	The time in seconds that this flow has been active as at the time this query was issued.
NextHop	Specifies the BGP next-hop address. This is always set to 0 in MPLS flows. It must correspond to the address of the hop querying its neighbouring hops.

The discrimination policy parameters were stored in array, X_0. When reconstructing the attack path, each neighbouring node is examined against the flow parameters stored in array, X_0, and matching parameters were assigned values according to it corresponding weights.

The value obtained for the parameters of packets flow records on each node examined was stored in $x_{e,k}$, for k = 1 to K, K been the total traffic flows examined on the node connected by edge e. The sum of matching parameters of all packets on the edge e with the attack packet parameters in x_0, denoted as χ_e, is determined by

$$\chi e = \sum_{k=1}^{K} \left(x_{e,k} \cap x_0 \right) \tag{2.12}$$

Each χ_e for all edges on node j on the attack path was stored as

$$\chi_j = \left[\chi e, \ldots, \chi_E \right], \; 1 \leq \text{e} \leq \text{E}, \tag{2.13}$$

E is the total edges connected to the node. It is considered that there may be two or more χ_e values in equation (2.13) that may be accidentally the same. This is because different sets of numbers can sum up to the same value, e.g. sums of 5 and 5, 3 and 7, 6 and 4 all give 10. Further steps are considered to avoid such situation that may confuse the IP traceback in determining a probable edge that is part of attack path. The average value of the matching features for edge e is calculated.

$$\beta_e = \frac{\chi e}{E} \tag{2.14}$$

Equation (2.14) determines the most common set of values. It will be small if smaller values are more than bigger values, or big if otherwise. But it will still be the same for χ_e values that are equal if those χ_e values have the same number of terms.

Since the occurrence of traffic flows in a packet switch network is random, it is assumed that the path that different attack packets will follow may not be the same. Thus those χ_e with same values may not have set of numbers that have exactly the same terms. Variance can show the difference in the terms of different χ_e. It will also magnify differences that are very close. Thus, the variance of the distribution of the sums of matching parameters on edges e was derived from

$$\alpha_e = \frac{\sum_1^K \left(x_{e,k} - \mu\right)^2}{K} \tag{2.15}$$

In equation (2.15) $1 \leq k \leq K$, K is the total number of matching parameters, μ is the mean of the different matching parameters obtained for the edges.

The probability of edge e been on the attack path, p_e was calculated using

$$p_e = \frac{\chi_e}{\sum_{e=1}^{E}(\chi_e)} \tag{2.16}$$

Using values obtained from equations (2.12), and (2.14) to (2.16) the fitness of the edge been on attack path is determined as follows:

$$f\left(\chi_j\right) = max\left[\left(\frac{\beta_e}{\alpha_e}\right).\left(\frac{\chi_{j,e}}{\alpha_e}\right).e^{\left(\left(\frac{\chi_{j,e}}{\alpha_e}\right)^{p_e}\right)}\right]_{e=1}^{E} \qquad \chi_{j,e} \geq 0 \tag{2.17}$$

$1 \leq e \leq$ E, E is the total edges on the node j

In equation (2.17), $\left(\frac{\beta_e}{\alpha_e}\right)$ further differentiate edges with same values of χ_e based on the distributions of their matching parameters. The probability p_e is used to reduce the range of values to a more manageable size without altering the relative differences between the values. As the shark advances from the victim towards the attacker, at every node (*j*) on the attack path each edge ℓ carrying packets to (*j)* was examined using equations (2.12) - (2.17).

2.5 Network Simulator Version 2 (NS2)

NS2 is the version 2 of the commonly used simulator for testing new network algorithms. It is a discrete event simulator developed for networking research. NS2 provides support for simulation of TCP, routing, and multicast protocols over wired and wireless networks. Network simulator 2 has been developed under the Virtual Inter-Network Testbed (VINT) project in 1995 comprising researchers from the University of California at Berkeley, University of Southern California (USC)'s Information Sciences Institute (ISI), Lawrence Berkeley National Laboratory and Xerox Palo Alto Research Centre. The VINT project was sponsored by the Defence Advanced Research Projects Agency (DARPA) and the National Science Foundation (NSF) (Gupta *et al.*, 2013). NS2 is currently maintained by the ISI.

Network simulator 2 is written in C++ and uses OTcl, the Object-Oriented version of Tcl, as a command and configuration interface (Kevin & Kannan, 2015). Other modules like Georgia Tech Internetwork Topology Models (GT-ITM), Stanford Graph Base (SGB), AWK, etc, are available for use with NS2 to accomplish different tasks. The NS2 scripts for generating network topology is written in Tool command language (Tcl) and the protocols are written in C++. Tool command language/C-plus-plus language (Tcl/Cl) interface is available in NS2 for linking the OTcl and the C++ scripts. The ACS-IPTBK and SSOA-DoSTBK algorithms was

written in C++ language and linked through the Tcl/Cl to the OTcl. The simulation animation was generated with the NS2 Network Animation Module (NAM).

2.6 Review of Similar Works

The works from previous researchers on different methods of implementing IP traceback and its applications are reviewed in this section.

Hamedi-Hamzehkolaie *et al.*(2014) proposed DoS traceback scheme based on bee algorithm. Bee algorithm applies traffic flow information as attraction which is used as a feature to find the attack route. The intensive traffic flows of the attack packets are considered as the food resource the bees are exploring. The most probable attack path is considered to be the path explored by most bees based on the amount of nectar and closeness to the hive. The bees follow the traffic flow to reach a router which makes the router with more DoS attack traffic flows to be selected by more bees in a kind of positive feedback loop that results in the majority of bees to converge on one route. This method will return false positive in the presence of a sudden surge in legitimate traffic flow because occurrence of flash event was not considered by the authors.

Bhagat & Pasupuleti (2015) used Simulated Rain Drop (SRD) for developing a relay mechanism as a solution to mitigate distributed DoS in cloud computing. SRD is a swarm algorithm and it is used to cover the large area of the cloud computing. When DoS attack caused bottleneck in cloud computing the SRD relay mechanism will search for the available maximum throughput where data can be rerouted to avoid congested edges affected by DoS attack. This solution only finds alternative route for data rerouting in the presence of DoS attack.

Wang *et al.* (2016) proposed a modified ant colony system (ACS-IPTBK) algorithm for IP traceback that can detect attack path with minimum number of packets and without necessarily having complete network routing information. Each ant traverses an attack path. In addition to the global rule of the original ant colony optimisation used for ant evolution, ACS-IPTBK add local rule for a deeper search. The ants drop pheromone on the path as they traverse it. The path traversed by the highest number of ants, which is the path with highest amount of pheromone, is returned as the most probable attack path. This method did not consider sudden surge in legitimate traffic flow and it may return false positive in the presence of such surges.

Saurabh and Sairam (2016) proposed improved probability packet marking (PPM) for IP traceback for DDoS. In order to reduce the number of packets needed to reconstruct an attack path they introduced an improvement to an existing method of calculating the number of packets needed for complete path information, called Completion Condition Number (CCN), by incorporating the calculation of upper bound on the variance of the packet distribution. They applied their proposed method to multipath attack with shared paths by first calculating the number of packets passing through each shared path and then calculate the CCN for each path. They were able to improve the correctness of attack graph reconstruction with their proposed method. PPM packets are overwritten randomly by the routers down the path away from the router that marked it. When attack packets pass through different paths it may be difficult for the victim to distinguish the path.

Latif *et al.* (2016) presented efficient traceback technique (ETT) based on Dynamic Probability Packet Marking (DPPM) approach. Unlike in other PPM schemes they use variable marking probability based on the number of hops travelled by a packet to reach the target node instead of using fixed marking probability. Their scheme used Media Access

Control (MAC) header instead of IP header because Wireless Body Area Network (WBAN) which it was designed for uses MAC Protocol Data Unit (MPDU). The DPPM is written in a portion of data payload of the MPDU. The proposed scheme cannot work with TCP/IP protocol and it reduces amount of data that can be contained in its data payload by using part of the data payload for marking.

Bhuyan *et al.* (2016) proposed a lightweight system based on Extended Entropy Metric (EEM) for DDoS flooding attack detection and IP traceback, called E-LDAT. They calculate EEM for each distribution of source IP addresses and compare the calculated EEM values against a threshold value set for distributions. Any value of EEM greater or equal to threshold is considered an attack packet and its source address is added to the list to be submitted to router for traceback using NetFlow. Been dependent solely on NetFlow for its traceback, this scheme may not be able to go beyond a router on which NetFlow is not enabled. Moreover, legitimate traffic flow may surge up beyond the threshold value and may thus be detected as attack leading to wrong source detection.

Yu *et al.* (2016) present a feasible IP traceback framework based on Mark on Demand (MOD) implemented on deterministic packet marking (DPM) mechanism. A router detects surge in packet flow and request a unique marking from the MOD server which it used to mark the packets with sudden high flow. When an attack is detected, the victim requests extracted information from the attack packet from the MOD server and use the information for IP traceback. This is a novel proactive mechanism for detection and traceback of DDoS attacks. The detection which may not necessarily confirm a DoS attack can occur on any router along the transmission path. MOD addressed the scalability problem of DPM. They also considered storage problem peculiar to DPM mechanisms and mitigate it by minimising

the marking to one unique mark on a packet. After attack has been confirmed detected by a router on the path, the traceback is carried out cooperatively by the monitoring system and the MOD. The router that ascertained the attack will check if the attack packet is marked and leave the mark as it is, or if not request a unique mark from MOD and mark the packet. It then notifies the monitoring system with the unique mark on the packet about the attack. The monitor request details of the packet stored on the MOD including the first router that marked it. The information is used to trace the source of the attack to the source router or as close as possible to it. Practical implementation of this mechanism is highly dependent on cooperation from most of the operators on the Internet which may be difficult to achieve. It also increases overhead on Internet by requiring every flow to be monitored by examining the number of packets carry by the flow and see if the number is above a threshold or not. Also, request to MOD server increases transactions per flow.

Venkataramanan & Ravi (2017) developed an IP traceback mechanism using particle swarm system. Particle swarm system is an enhanced Particle Swarm Optimization (PSO) algorithm with local updating rule incorporated in addition to the global updating rule of the original PSO. The incorporated local updating rule is used to enhance PSO efficiency by avoiding trapping in local suboptimal solution and be able to return better results. The mechanism was reported to be able to reconstruct attack path using fewer packets than needed by similar previous schemes and was able to trace the source of spoofed attack packets. This scheme is based on flow-level information for reconstructing the attack path but possible surge in data traffic due to occurrence of flash event is not considered.

Saini *et al.* (2017) developed hybrid IP traceback mechanism using two nature inspired optimization algorithms, namely, Ant Colony Optimization and Particle Swarm Optimization. The two optimization algorithms are swarm algorithms and known to be effective in solving

combinatorial optimization problems. The PSO was used to improve the convergence rate and reduce the computational complexity of ACO algorithm. Because ACO normally performs search operation on the basis of distance it is combined with velocity based PSO algorithm so that premature convergence can be avoided and faster convergence with lesser number of ants can be achieved. Thus, the scheme combined the distance metric from ACO with the direction and velocity metric from PSO to compute probability metric for the IP traceback process. The objective was to improve the traceback time and reduce number of packets required for the IP traceback problem. It was reported that the scheme was able to find a more globally optimum solution for IP traceback problem with improved convergence rate using fewer numbers of ant and particle agents. Like other available IP traceback solutions reviewed that are based on nature inspired algorithms, this solution used flow-based search method to trace the source of an attack. Possible effects of flash events were not considered in the paper. Thus, the solution is also fallible to the effects of flash events traffic surge during traceback process.

CHAPTER THREE

MATERIALS AND METHODS

3.1 Introduction

The details of the materials and the procedures used for the successful completion of this research are discussed in this chapter and the SSOA-DoSTBK is developed.

3.2 Materials

The resources used for developing a simulation of the prototype of the developed scheme are listed here:

1 Operating System: Ubuntu 16.04 LTS
2 Simulation software: Network Simulator version 2 (NS2), ns-allinone-2.35
3 Programming software: C++, OTCL
4 Graph plotting software: GNUplot.
5 Laptop PC: x64-based processor, Intel Core i3-3110M CPU 2.4 GHz, 4.0 GB RAM. The knowledge of computer network data transmission and vulnerabilities.

3.3 Methodology

The methodology adopted for achieving the aims and objectives set for this work is explained in this section. Details of the steps for creating the components of the developed scheme and the implementation of the tools used for creating them are given. Explanations on how the scheme was executed and how results were obtained were made at the end of this section.

3.3.1 Development of the SSOA-DoSTBK

SSOA-DoSTBK simulation was developed in NS2. Wired Network was used for the path tracing to enable physical visibility of the path tracing process.

Ping and messaging agents were set up using OTCL classes. Messaging was used for generating normal network transaction between other nodes. Ping agent was used to generate

the ping of death attack by the randomly selected attacker on the randomly selected victim. Other communication agents including TCP, FTP, and CBR were establish for different network transactions on the network. The Tcl code listing used for setting up the topology and establishing communications among the nodes can be found in Appendix A-VIII.

The network consisting of 25 autonomous systems with 8 nodes each was defined using the Tcl code. The topology used for the simulation was generated in the simulator using a random topology simulator code written in Tcl language of the simulator. The random topology simulator code randomly placed the 200 nodes at the coordinate points in the network area. The WaxMan's connectedness probability used to establish connection links attributes which include transmission rates and bandwidth between nodes based on their distance apart was generated using equation (2.11). The nodes distances apart, d(i,j),in equation (2.11)was set to 1 to connect each node to its one hop distance neighbouring node. L is the longest possible distance between any two nodes. Different values of η were selected from [0.3, 0.6, 0.9, 1.2, 1.5] for different generations of the simulation in that order. The value of $\gamma = 0.1$ was used for different values of η selected. The values were used for different simulations to vary the links attributes for the simulations.

3.3.2 Discrimination Policy

A set of the parameters from the attack packets that uniquely identifies the attack were extracted from the attack packets obtained from DoS detector. Unique flow parameters that are often recorded by routers were also identified. These parameters were assigned weight values according to their uniqueness to the flow as shown in the Table 3.1 and stored in array, x_0 (Equation 3.1). First column of Table 3.1 are the parameters that can be found on real life routers, e.g. Cisco routers' flow records. Second column are their equivalent in NS2 that were used for the simulation.

Table 3.**1**: Weight values assigned to discrimination parameters

Parameter	NS2 Parameter	Value
ICMP	Pkt Type	4
IP-ID	Pkt Id	5
MAC-addresses	Not used	
Packet-Length	Pkt_sz	1
TTL	Not used	
VLAN-ID	Not used	
Dst IP address	Dst Node Id	4
Dst Port Msk AS	Not used	
Pkts	Packets Count	NV
Active	Host log Time	1
NextHop	Next hop node	2

NV = No Value

Table 3.2 shows the structure of the discernment policy when the weighing values are assigned to the selected parameters for identifying the attacks.

Table 3.2: Representation of the Discrimination Dataset

Pkt Type 4	Pkt Id 5	MAC-addresses	Pkt_sz 1
TTL	VLAN-ID	Dst Node Id 4	DstPortMskAS
Packets Count 0	Host log Time 1	Next hop node 2	

The comparison string in x_0 is

$$x_0 = [4,5,*,1,*,*,4,*,0,1,2] \tag{3.1}$$

The values in the string indicates ICMP, IP-ID header field value, MAC-addresses, Packet-Length, TTL, VLAN-ID, Destination IP address, Destination Port Mask AS, Packet, Active, and NextHop, from left to right as explained in Table 3.1. The fields with wildcard value "*"

are ignored during the comparison. An illustration of the use of the discrimination policy during the traceback process is given in the following steps.

I) Set up discernment policy, x_0, dataset based on the details extracted from the detected attack packets: Table 3.3 shows how dataset in discernment policy is set up. Each parameter together with its value as obtained from the details of the attack packets obtained from the attack detectors are stored in the discernment policy.

Table 3.3: Parameters values in x_0 as obtained from the detected attack packets

ICMP	8
IP-ID	256
MAC-addresses	
Packet-Length	100 bytes
TTL	129
VLAN-ID	
Destination IP address	10.5.12.34
Destination Port Mask AS	
Packets	9137
Active	1463103010
Next Hop	198.168.1.1

II) As an example, the records of the data of two traffics with flows k=1 (Flow Set ID 256) and k=2 (Flow Set ID 312) found on a hop is shown in Table 3.4. The first column to the left in Table 3.4 shows the parameter names. The values of each parameter as found on the host are indicated in the second column to the right in the table. The data are obtained from the NetFlow records on the router. These are the parameters that are compared with the dataset in the discernment policy by matching their values. The value of a parameter should be exactly the same both in the discernment policy and in the flow record if the parameter normally carries a fixed value, e.g. source and destination addresses do not change after they are set for a session. If the value normally changes, e.g. TTL value that decreases from hop to hop, the value in the discernment policy must fall within appropriate range based on expected difference between when it was recorded on the hop and when it was received by the victim.

Table 3.4: Sample flow records for two different traffics on a link connecting node j

Flow Set ID	256
Packets	9137
ICMP	8
MIN_TTL	127
MAX_TTL	131
MIN_PKT_LNGTH	98 bytes
MAX_PKT_LNGTH	102 bytes
IPv4 SRCADDR	105.1.22.54
IPv4 DSTADDR	10.5.12.34
IPv4 NEXT HOP	198.168.1.1
FIRST_SWITCHED	1463103000
LAST_SWITCHED	1463103008
Bytes	5344385
Flow Set ID	312
Packets	6125
ICMP	9
MIN_TTL	221
MAX_TTL	227
MIN_PKT_LNGTH	98 bytes
MAX_PKT_LNGTH	102 bytes
IPv4 SRCADDR	192.168.1.27
IPv4 DSTADDR	10.5.12.34
IPv4 NEXT HOP	192.168. 1.1
FIRST_SWITCHED	1463102050
LAST_SWITCHED	1463102056
Bytes	388934

III) The process of determining the most probable nodes and edges involved in routing the attack packets is shown in Figure 3.1. The packets parameter and the value they have in the attack packets header fields are stored in x_{ij}, the discernment policy, and the parameters of two flows with ID 256 and ID 312 found on a hop are indicated under $x_{g,k}$, as shown in Figure 3.1. The values of the two flows parameters are compared by cross checking their values against the same parameters in the discernment policy. If a parameter in the discernment policy exist and its value in the discernment policy is the same with its value in the flow the corresponding weight value for it in the discernment policy as assigned in Equation 3.1 will be recorded for it. All value of matching parameters in each flow are added up based on Equation 2.12. Sum of the values for each flow is then tested with Equation 2.17 for fitness. The best fit obtained from equation 2.17 is selected as the attack traffic.

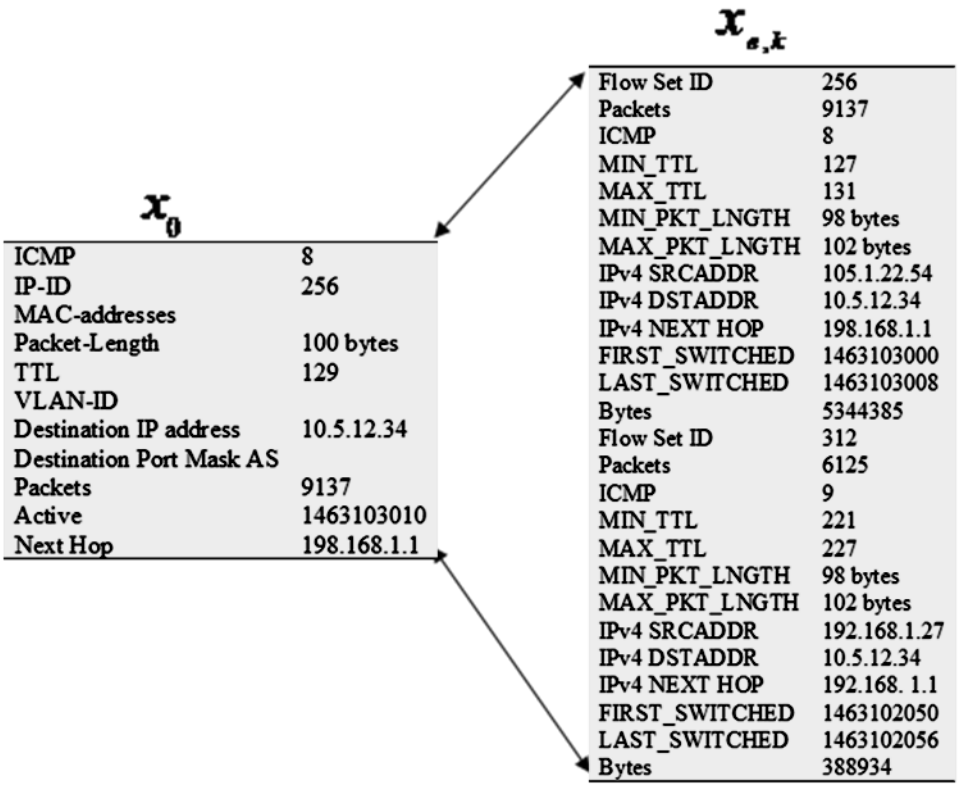


Figure 3.1: Comparing parameters values in x_0 with flow records, $x_{e,k}$, detected on a link

Based on explanation given in step (III), using assigned weight values in Equation (3.1) for corresponding matching parameter, χ_e for traffic flow k=1 is 17, while for traffic flow k=2 is 7 using equation (2.12). The edge through which traffic flow k=1 was transmitted and the node the edge connected are included in the attack path list.

The parameter that has highest value of 5 is the one that uniquely identify the flow, i.e. the flow id. The ones that must be the same for all packets of the flow, though they are not unique to the flow, are assigned 4, i.e. packet type, and the destination address. Value weighing 2 is assigned to Dst Port Msk AS and NextHop address because many other packets routed by the same hop will have the same value. NextHop address can only change for packet routed after the routing table of the hop changed. Others are assigned weigh value 1 because either they cannot be relied upon because the attacker can easily change them, e.g.

Mac address or IP address, or they are not highly important for identifying the flow but they are needed, e.g. server active time or Packet length. Number of packets routed by the hop within the time frame is not assigned value because it includes other packets that are not attack packets. It is use only to estimate traffic on the hop within the time frame.

3.3.3 Solving DoS IP Traceback Problem Using SSOA-DoSTBK

Prevention of DoS attack involve detection of the attack (e.g. by Intrusion Detection System (IDS)) and blocking the traffic from the source of the attack (e.g. by Firewall service or Intrusion Prevention System (IPS)). Finding the source of the attack to bring the attacker to book and/or penalize the attacker to compensate the victim for loss incurred requires network forensic to establish facts confirming the attacker as the perpetrator. IP traceback is a process that can be used in network forensic to gather the required facts. In this research it was assumed that there are effective DoS detection and prevention systems on the attacked machine. The detection system identifies the malicious packets and passes it to SSOA-DoSTBK to expose the attacker. SSOA-DoSTBK employs the novel features of the SSOA together with its innovative discernment policy to efficiently solve the DoS IP traceback problem and identify most probable path to the source of the attack under the constraint of available minimum required number of packets and minimum convergence time. The paths were assumed to be non-cyclic to avoid local optima and enhanced faster convergence. The method that was employed in this research for the IP traceback leveraged on the fact that even if attackers may be able to manipulate information in the header of the malicious packets they cannot alter the records of the path taken by the packets from the source to the receiver as logged by the routers that route it along the route. Such records include the flow identification information which can be inferred from the packets' IP header Identification field, Time-to-Live (TTL) value can also be used for Hop-Count Filtering (HCF) (Qian *et al.*,

2017), and hop address of the ingress and egress edge routers on the routers with IP flow recording enabled.

The developed IP traceback scheme examined the hop traffics focusing on those traffics the attacked system IP address is their destination address. The parameters with values 5 are given higher priority in the order of search, then other ones are considered in descending order of their assigned weight values. The header parameters of each packet k on a neighbouring node j connected by edge e are denoted $x_{e,k}$, $k = 1$ to total number of examined packets, K. This is repeated for each neighbouring node k till all the K neighbouring nodes of the present host, H, have been examined. The amount of attack packets routed by the neighbouring node j is estimated with equation (2.12). χ_e is the total weights of matching parameters estimated on node j connected by edge e to H. Attack path reconstruction by SSOA-DoSTBK is shown in Figure 3.2. The returned attack path consists of the yellow edges from the victim that is dropping packets to the attacker. The nodes involved in routing the attack packets along the attack path are indicated with red colour.

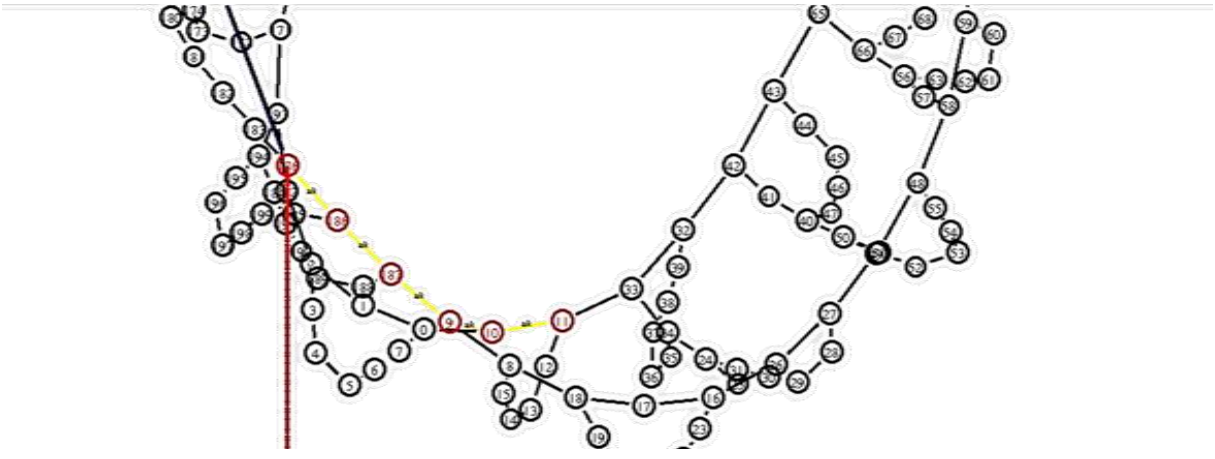


Figure 3.2: Attack path reconstruction by SSOA-DoSTBK.

The sequence of the steps in the SSOA-DoSTBK IP traceback process as explained here in sub-sections 3.3.1 to 3.3.3 is illustrated in the flow chart shown in Figure 3.3.

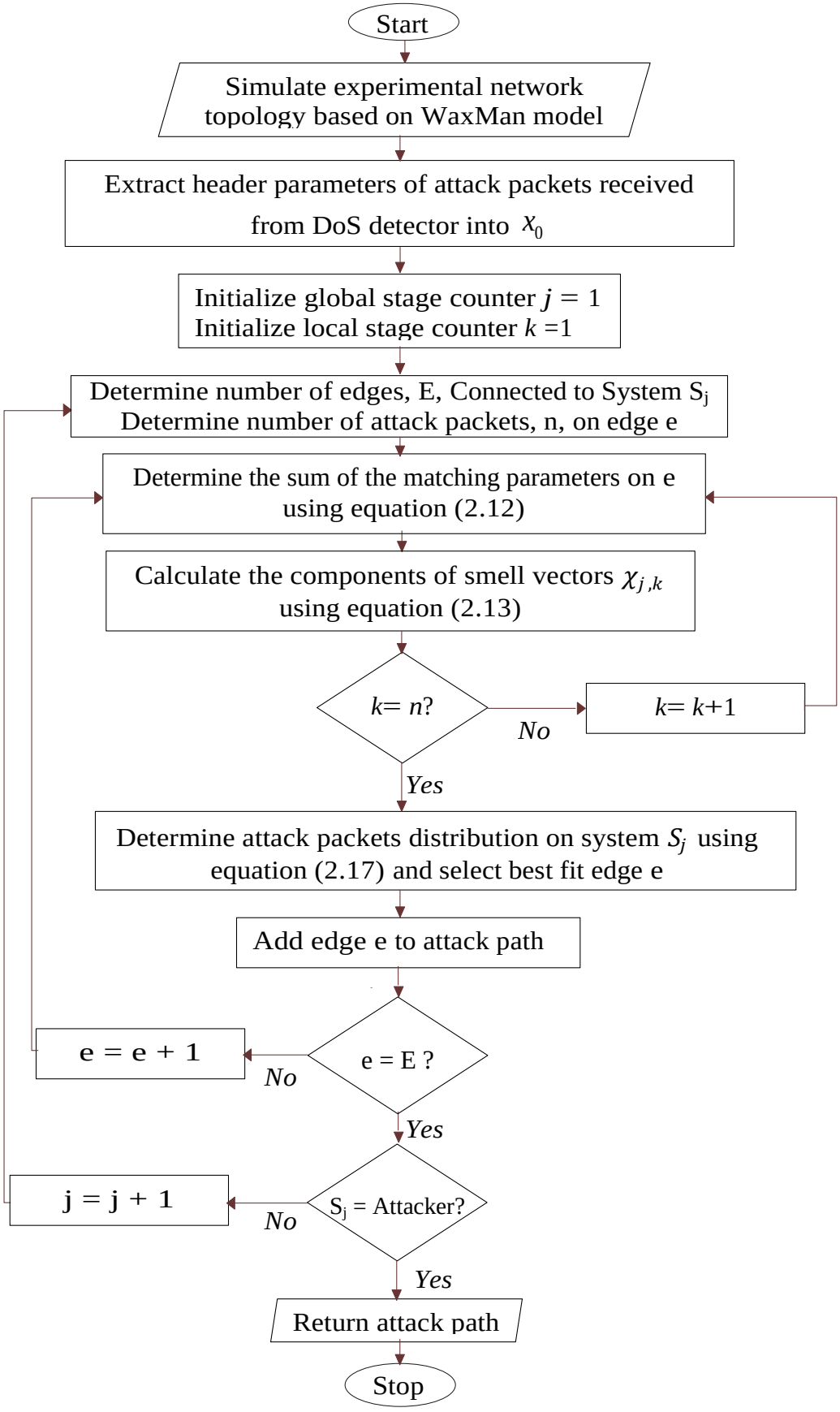


Figure 3.3: Flowchart of the Proposed SSOA-DoSTBK

Figure 3.3 summarily illustrates the steps explained in subsections 3.3.1 to 3.3.3 for the development of the developed solution called SSOA-DoSTBK.

3.3.4 Comparison of SSOA-DoSTBK with ACS-IPTBK

The codes for the SSOA-DoSTBK algorithm (SSOADOSTBK.h, Appendix A-I & SSOADOSTBK.cc, Appendix A-II) and the codes for ACS-IPTBK algorithm (ACS-IPTBK.h, Appendix A-IV & ACS-IPTBK.cc, Appendix A-V) were written in C++ and incorporated into NS2 by recompiling NS2. The C++ codes for the algorithms of developed solution and the benchmark solution were run on g++ 5.0.4 compiler on Ubuntu 16.04 LTS. The C++ objects of the algorithms were access through OTcl/CL using Tcl in ns-allinone-2.35. The OTcl codes for calling SSOA-DoSTBK (Appendix A-III) calls the C++ codes for SSOA-DoSTBK (Appendix A-II) and pass the necessary traffics parameters to it for computation. SSOADOSTBK.cc carryout the necessary computations and returns values to Main_IPtraceback.tcl (Appendix A-VIII) for simulation. Similarly, OTcl codes for calling CAS-IPTBK (Appendix A-VII) calls ACPIPTBK.cc (Appendix A-V) pass necessary parameters for initial settings and for each of the class objects when it calls. Randoms.cpp (Appendix A-VI), is used to set initial population of ants and random placements of the ant agents on initial nodes. ACPIPTBK.cc computes necessary parameters and returns the values to Main_IPtraceback.tcl for simulations. The simulations generated values are listed in tables in Appendix A-IX to Appendix A-XI for FER, Performance in attack packets detection, and convergence time that were used as the metrics for comparing the results of the developed scheme with the benchmark. These values are the data that were used for calculating $pecentage\,improvement$ and *Correctness* in Section 3.3.5 and were also used for plotting the results in *Results and Discussions* (Chapter Four).

Results were obtained from SSOA-DoSTBK for False Error Rate, Performance, and convergence time when there was no flash event when DoS attack was traced, when there was flash event when DoS attack was traced, and when there was flash event when spoofed DoS was traced. Performance was measured in terms of correctness based on the number of attack packets available on the return path. The results obtained for SSOA-DoSTBK were compared with similar results obtained from ACS-IPTBK for the same tests under the same conditions.

3.3.5 Performance Evaluation

The performance of the developed scheme was calculated and compared to the ACS-IPTK that it was benchmarked against as follow;

To illustrate the computation of the comparison metrics, if the average of the results obtained from SSOA-DoSTBK in a test is S_{AVE} and A_{AVE} was obtained from ACS-IPTBK in the same test, e.g. average of data obtained from False Error Rate tests for SSOA-DoSTBK and ACS-IPTBK in (Appendix A-IX), comparison of the average performance of the two schemes is calculated as

$$pecentage\,improvement = \frac{S_{AVE} - A_{AVE}}{A_{AVE}}\% \tag{3.2}$$

The performance of the schemes in the traceback of the simulated attacks, including spoofed packets, under different conditions were measure in terms of correctness of the attack path returned by examining the number of attack path on the returned path as,

$$Correctness = \frac{Average\ attack\ packets\ on\ returned\ path}{Total\ packets\ routed\ on the\ path} \times 100\% \tag{3.3}$$

CHAPTER FOUR

RESULTS AND DISCUSSIONS

4.1 Introduction

The evaluation of SSOA-DoSTBK with ACS-IPTBK is discussed here in this chapter. The evaluation tests conducted were three types. Each of the three tests was carried out under three different conditions. These tests types were False Error Rates, Correctness of returned path, and convergence time. Each of those tests were conducted with DoS attack only, combined DoS attack with flash event, and spoofed DoS attack with flash events. Two dimensional (2D) graphic plots of the results are presented for clearer view of the performance of the developed scheme.

4.2 Simulation Results

To evaluate the efficiency of the developed SSOA-DoSTBK IP traceback scheme, the attack path tracing results are recorded (Appendix A-IX, Appendix A-X, Appendix A-XI) and compared with results obtained in similar tests from the benchmark scheme, ACS-IPTBK, in the following sub sections.

4.2.1 Evaluation of False Error Rate

Here, the false error rates from the results of each of the two IP traceback schemes were evaluated. The evaluation was carried out under the three conditions stated in section (4.1). The results were presented in this section.

4.2.1.1 FER of the Schemes under DoS attack

For this evaluation DoS was traced in the network when there was no surge in the network due to flash event. SSOA-DoSTBK and ACS-IPTBK were used to detect the source of the attack. 100 generations of an average of 100 iterations each were carried out for each IP

traceback schemes. The data obtained from the results of these tests are recorded in (Appendix A-IX, Column: DoS) for SSOA-DoSTBK and ACS-IPTBK. The graphical representation of the results is presented in Figure 4.1. As could be seen on the plot SSOA-DoSTBK presented a lower false error rate than the ACS-IPTBK by 31.8% on average based on equation(3.2).

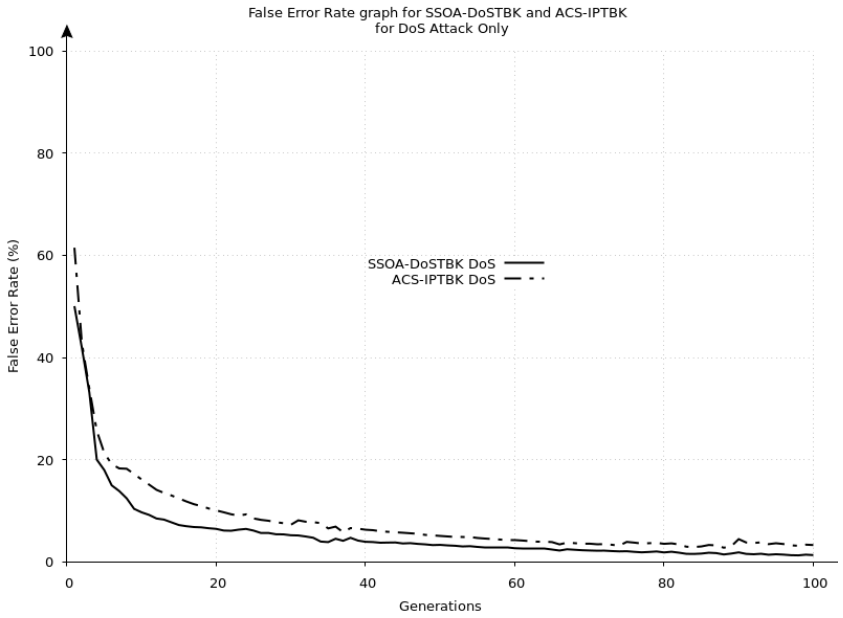


Figure 4.1:Results of FER for SSOA-DoSTBK and ACS-IPTBK under DoS

4.2.1.2 FER of the Schemes under Combined FE and DoS attack

The false error rate of the two IP traceback schemes were also compare when Flash Event generated traffic surge on some segments of the network when the attack path was traced. The data obtained from the results of these tests are recorded in (Appendix A-IX, Column: DoS & FE) for SSOA-DoSTBK and ACS-IPTBK. The results for the two schemes are plotted in Figure 4.2. The ACS-IPTBK generated more false error rate in this test than the SSOA-DoSTBK. The traffic produced by flash event can be mistaken for an attack by an IP trace back scheme that operate base on flooding characteristics of a high rate DoS attack.

ACS-IPTBK examines the edges based on amount of routing packets on them at the time of the DoS attack. But high routing packets may not ordinarily mean that all packets routed within the particular time frame belong to the same flow since a router in packet-switch network receives different packets randomly from neighbouring nodes. This is also evident in Figure 4.2. Based on equation (3.2) SSOA-DoSTBK results were better than ACS-IPTBK by 32.06% on average.

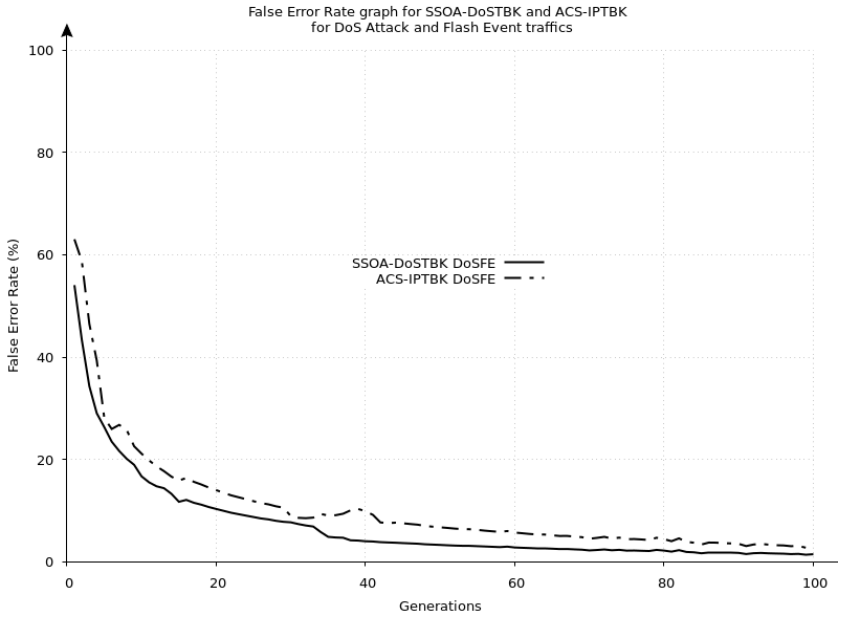


Figure 4.2: Results of FER for SSOA-DoSTBK and ACS-IPTBK under DoS and FE

4.2.1.3 FER of the Schemes under Combined FE and Spoofed DoS attack

The two schemes, SSOA-DoSTBK and ACS-IPTBK false error rate were observe when flash event generated traffic surge on some nodes in the network at the time spoofed DoS attack was being traced to its source with the schemes. The data obtained from the results of these tests are recorded in (Appendix A-IX, Column: Spoofed DoS & FE) for SSOA-DoSTBK and ACS-IPTBK. The percentages of errors in the returned results by each of the two schemes for 100 simulations were plotted together in Figure 4.3. In this test, SSOA-DoSTBK returned

lower error rate percentages than the benchmark scheme, ACS-IPTBK, as shown by the curves in Figure 4.3. The challenges of the IP traceback schemes under this condition are very similar to those discussed in section (4.2.2) but with the additional challenge of unknown source of the attack. SSOA-DoSTBK presented better results than ACS-IPTBK in this test as well. The results obtained from SSOA-DoSTBK were 28.45% better on average than those obtained from ACS-IPTBK based on equation (3.2) as it can be seen in Figure 4.3.

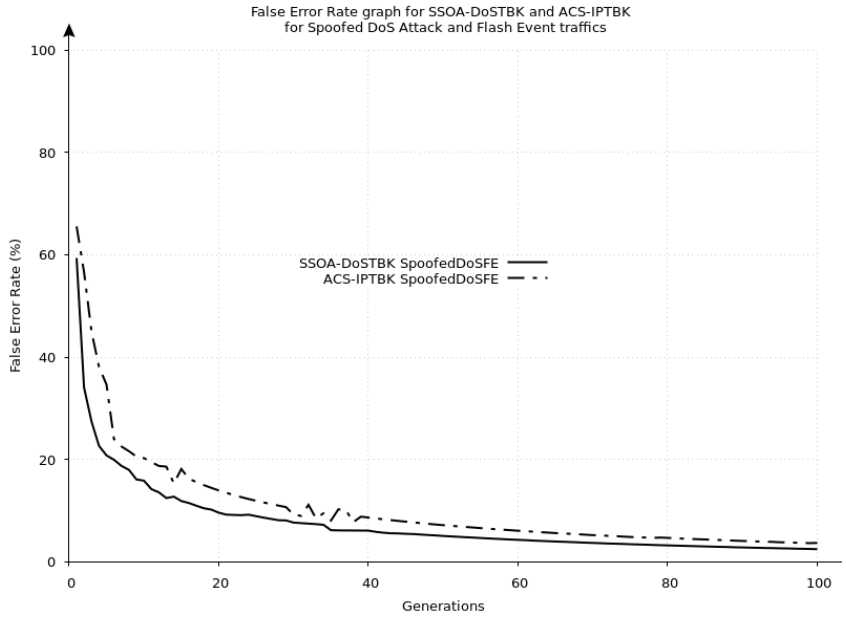


Figure 4.3: Results of FER tests for SSOA-DoSTBK and ACS-IPTBK when FE traffic surges occurred during spoofed DoS attack source traceback

4.2.2 Performance Evaluation

The detection correctness of the scheme is measured by comparing the average number of attack packets that were on the attack paths that the scheme returned as a ratio of the total packets routed along the path. Average number of attack packets on the attack path is calculated as the total number of attack packets on the path divide by the number of the hops on the path using equation (3.2). The simulations were carried out as explained in section (4.2.1).

4.2.2.1 Performance under DoS attack

When DoS attack was traced but no flash event surge in flow in the network the performance of the two schemes were measured. The data obtained from the results of these tests were recorded in (Appendix A-XI, Column: DoS) for SSOA-DoSTBK and ACS-IPTBK. The results were plotted in Figure 4.4.

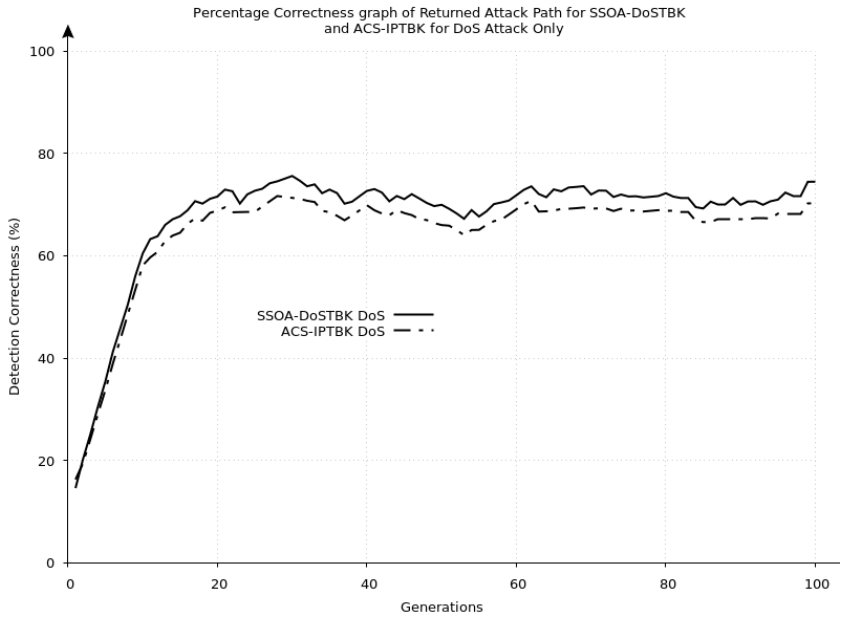


Figure 4.4: Performance of SSOA-DoSTBK and ACS-IPTBK for DoS attack tracing without FE

As can be seeing in the Figure 4.4, SSOA-DoSTBK presented higher performance than the ACS-IPTBK. SSOA-DoSTBK recorded by 4.76% average performance improvement over the ACS-IPTBK.

4.2.2.2 Performance under concurrent FE traffic and DoS attack

The performance of the two IP traceback schemes, SSOA-DoSTBK and ACS-IPTBK were tested in the presence of flash event surge during DoS attack traceback in the same network. The data obtained from the results of these tests were recorded in (Appendix A-XI, Column: DoS & FE) for SSOA-DoSTBK and ACS-IPTBK. The graphical illustration of the result of

the test is plotted in Figure 4.5. The SSOA-DoSTBK recorded better performance than the ACS-IPTBK by 11.6% on average result.

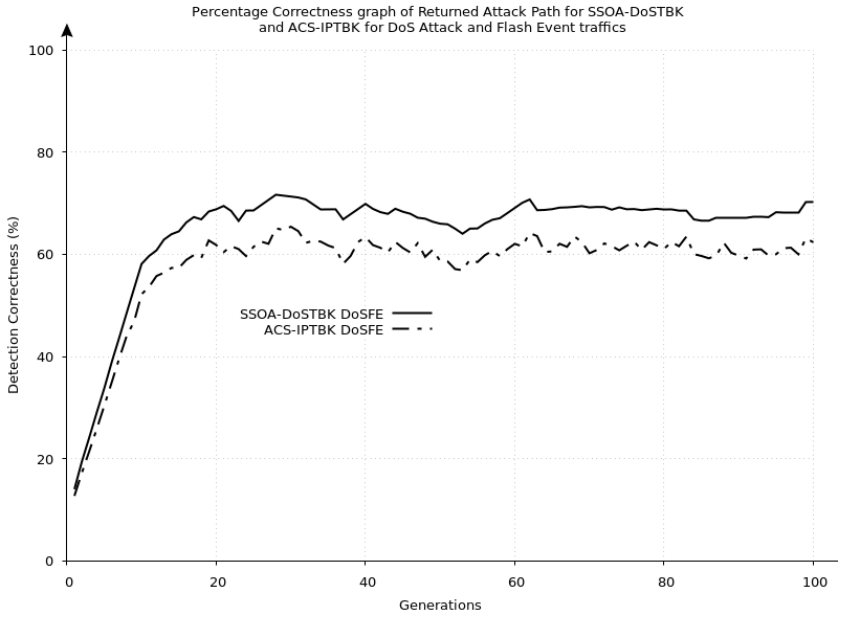


Figure 4.5: Performance of SSOA-DoSTBK and ACS-IPTBK when tracing DoS attack and there were traffic surges due to flash event in the network

4.2.2.3 Performance under concurrent FE traffic and Spoofed DoS attack

This performance test was carried out with spoofed packets used for the DoS attack and generated flash event causing heavy normal traffic in the network during the attack source traceback. Both SSOA-DoSTBK and ACS-IPTBK were used to trace the source of the attack and their performance measures were estimated using equation (3.2). The data obtained from the results of these tests are recorded in (Appendix A-XI, Column: Spoofed DoS & FE) for SSOA-DoSTBK and ACS-IPTBK. The results obtained are plotted in Figure 4.6. The developed SSOA-DoSTBK showed better performance over the ACS-IPTBK by 5.2%.

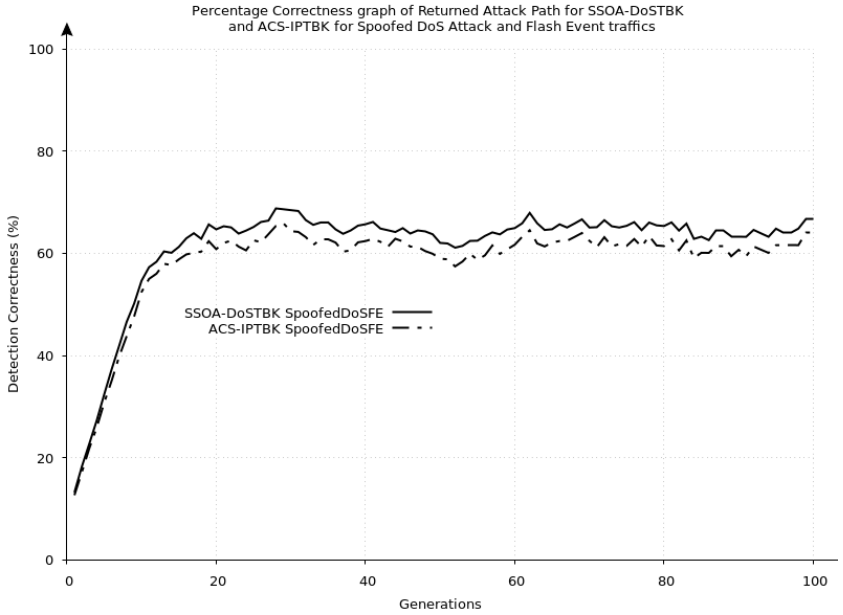


Figure 4.6: Performance of SSOA-DoSTBK and ACS-IPTBK for spoofed DoS source tracing when FE was present in the network

The developed scheme, SSOA-DoSTBK, returned better results than the benchmark the scheme, ACS-IPTBK, in all the tests conducted for False Error Rate and Performance in terms of the correctness of the path returned by each of the schemes based on the number of attack packets present on the attack paths returned. This is due to the fact that SSOA-DoSTBK actually scrutinise the traffics found on the hops examined using the discernment policy as the template of the attack features to ensure that the actual attack traffic was identified. But ACS-IPTBK estimates the fitness of a traffic based on its size in terms of the amount of its packets. Also, the improvement margin of SSOA-DoSTBK performance over the benchmark, ACS-IPTBK, was highest when DoS was traced in the presence of flash event surges for the False Error Rate tests and Performance tests. This confirmed the fact earlier stated that flow-based method cannot effectively mitigate the effects of flash event surge in normal traffic during IP traceback process unless a feature is specifically incorporated to address it.

4.2.3 Evaluation of Convergence time

The efficiency of the developed scheme, SSOA-DoSTBK, was also tested on how fast it could accomplish the task compared to the scheme used for benchmarking it, ACS-IPTBK. The results obtained under the three conditions in which the tests were carried out was recorded in Appendix A-X and presented in the following.

4.2.3.1 Convergence under DoS attack

The time taken by the developed scheme and the test scheme to return attack paths were recorded when the schemes were tested for source detection of DoS attack without surge in normal traffic caused by flash event in the network. The data obtained from the results of these tests are recorded in (Appendix A-X, Column: DoS) for SSOA-DoSTBK and ACS-IPTBK. The plot of the results is shown in Figure 4.7. ACS-IPTBK recorded a faster convergence than SSOA-DoSTBK by 0.4% on average values.

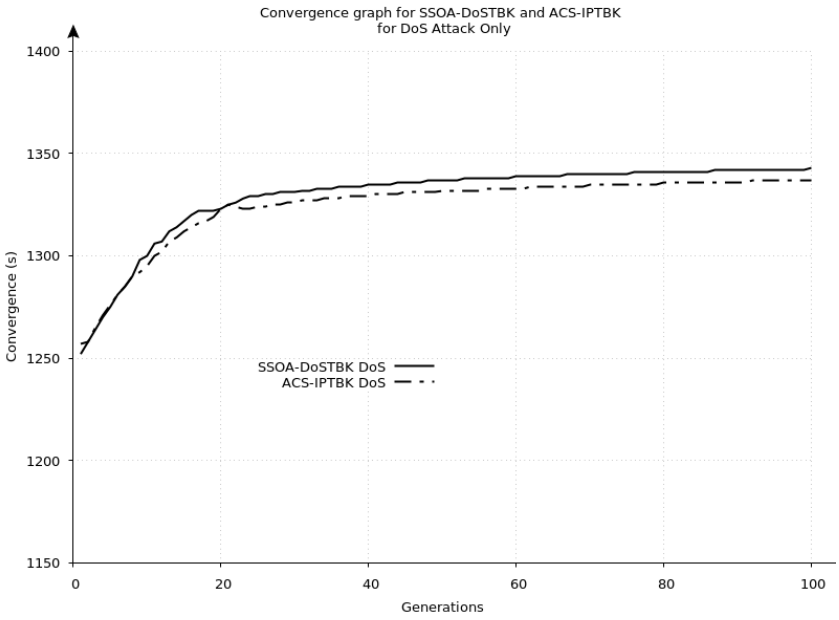


Figure 4.7: SSOA-DoSTBK and ACS-IPTBK convergence time for tracing DoS attack when there is no FE

4.2.3.2 Convergence under Concurrent FE and DoS attack

The convergence time was recorded for developed scheme and the benchmark scheme when they were used to trace back attack paths in the presence of surge in traffic in the network during the DoS attack traceback. The data obtained from the results of these tests are recorded in (Appendix A-X, Column: DoS & FE) for SSOA-DoSTBK and ACS-IPTBK. The results of the tests under this condition are plotted in Figure 4.8. Also, the benchmark scheme, ACS-IPTBK, recorded faster convergence than the developed scheme, SSOA-DoSTBK, in this test under this condition. The test scheme was faster than the developed scheme by 0.78% on average.

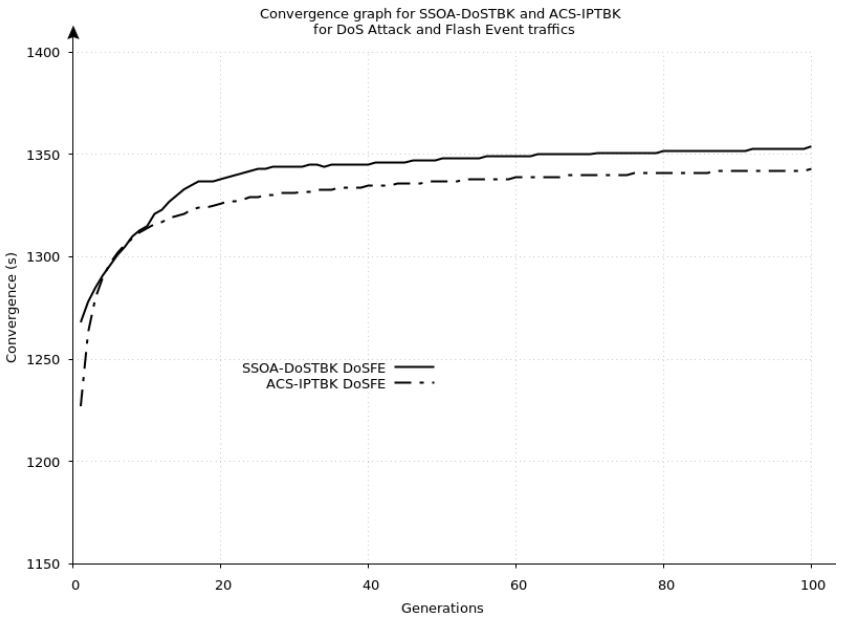


Figure 4.8: SSOA-DoSTBK and ACS-IPTBK convergence time for FE and DoS attack.

4.2.3.3 Convergence under Concurrent FE and spoofed DoS attack

The convergence time was recorded when the packets of the DoS attack were spoofed to conceal the attacker and, at the same time, traffic surge was caused by Flash Event in the network when the attack path is being traced. The IP traceback schemes are expected to be

more engaged under this condition because it has to deduce the source of the attack and also detect the attack path. The data obtained from the results of these tests are recorded in (Appendix A-X, Column: Spoofed DoS & FE) for SSOA-DoSTBK and ACS-IPTBK. The recorded data for these tests are plotted in Figure 4.9. It can be seen in the Figure 4.9 that highest convergence time were recorded by both schemes under this condition. The ACS-IPTBK still recorded faster convergence than the developed scheme under this condition. It was faster than the SSOA-DoSTBK by 1.2% on average which is the highest margin recorded for all the convergence tests.

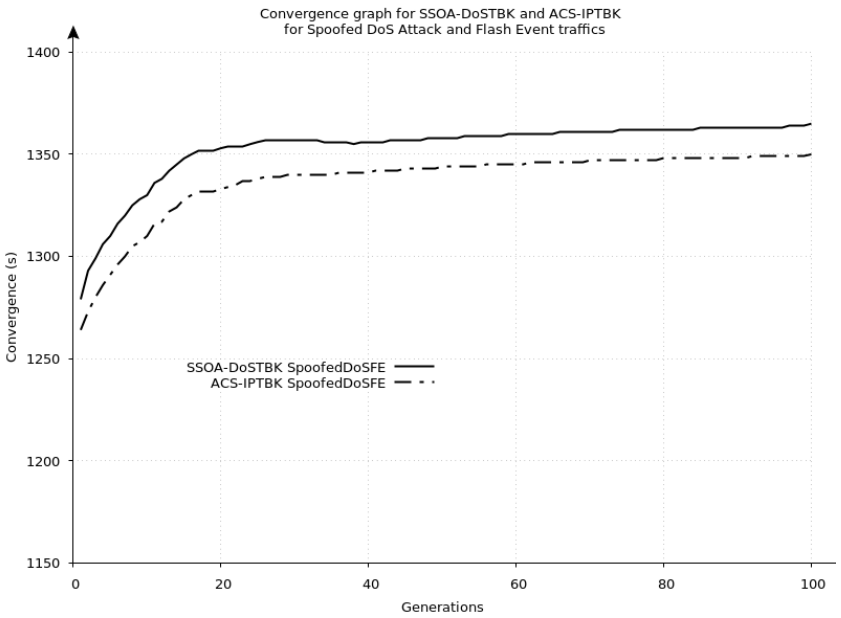


Figure 4.9: SSOA-DoSTBK and ACS-IPTBK convergence time for FE and Spoofed DoS attack.

Convergence time is not a measure of accuracy of a computation results but it may indicate the relative computational efforts expended to compute a result based on different conditions like the number of data involved in the computations. Thus it shows that for SSOA-DoSTBK, computation efforts expend is proportional to the number of traffics encountered on the hops during the trace back process of an attack source.

4.3 Attack path detection results

The results obtained for different tests performed in the simulations are discussed in the following subsections.

4.3.1 FER Tests Results

The results of the tests for performances with respect to false error rate shows that SSOA-DoSTBK results were better than those of ACS-IPTBK by 31.8% on average. SSOA-DoSTBK performs search focusing on the actual attack using the details extracted from the attack packets as heuristics. ACS-IPTBK on the other hand uses flow-based detection and distance metrics to decide the most probable attack path. Since DoS attack is the only traffic in this test flow-based detection may not cause much false error rate. The distance metric may cause error if the routing tables of the upstream routers change frequently due to congestions or nodes going down or for any other reasons. The estimated least cost path at the time of tracing back may not be the same best route chosen by a router during the actual routing of the packets because what caused the route to be avoided might have cleared at the time of traceback. This effect is less pronounced in SSOA-DoSBK because hop-by-hop traceback can detect the actual route followed by the attack traffic and further examination testing the link that carry largest traffic further enhances avoidance of the error due to the topology change.

The performance improvement by SSOA-DoSTBK when flash event occurred during the DoS attack traceback process was 32.06%. It is higher than when DoS attack traffic only was present. The reason for the better performance can be attributed to the fact that ACS-IPTBK been flow-based scheme can mistake flash event traffic for attack traffic as postulated earlier in section (1.3). The flow-base error effect together with the possibility of error due to distance estimation can increase the false error rate.

The performance improvement when the attack was spoofed and there was flash event along the traced attack path was 28.45% higher by SSOA-DoSTBK than ACS-IPTBK. The percentage improvement in this test condition was lower than the other two obtained for the DoS only and DoS with FE. This can be attributed to the greatest difficulty faced by the two IP traceback schemes in this condition. SSOA-DoSTBK does not rely much on the correctness of the source address on the attack packet but it may be possible that most of the flash event traffic were directed towards the victim node and does increase the number of packets tested within the time limit which could not make it be able to exhaust all the available traffics necessary.

4.3.2 Returned Attack Path Correctness Tests Results

The results of the tests for correctness of the paths returned by the developed scheme and the benchmark scheme are discussed here.

SSOA-DoSTBK outperformed the benchmark scheme in returning correct attack path under DoS attack only test by 4.76%. This means that its returned paths were averagely better by 4.76% than those returned by the benchmark scheme. This can be due to the reason that SSOA-DoSTBK use attack packets details as heuristics while ACS-IPTBK use routing packets and distance metrics. The effect of distance metric as discussed in section 4.3.1 can affect the average attack packets on its returned paths.

The improvement of SSOA-DoSTBK results over benchmark scheme in the returned path correctness when there is flash event along the traced attack path was 11.6%. The combined effect of flow-based traceback method and usage of distance metric for estimating attack path discussed in section 4.3.1 can also contribute to this result. Here more packets that are not attack packets are generated on some edges which can be mistaken for attack traffic by

traceback scheme using flow-based method. Path estimation can also result to detection of paths that attack packets did not follow due to topology changes. Thus, SSOA-DoSTBK detected more attack packets on its paths than the benchmark scheme.

In the situation when the attack packets were spoofed and flash event occurred on the attack paths traced, SSOA-DoSTBK returned attack paths that were averagely 5.2% more correct that those returned by the benchmark scheme. Since this calculation was based on the ratio of number attack packets on the path to the total packets on the path it is possible that there were many flash event packets which could reduce the ratio. But since the two schemes were set to detect the attack paths under the same condition, SSOA-DoSTBK was equipped with discernment policy as a template for detecting the actual attack traffic. Thus, it was able to return attack paths that more correct than those returned by benchmark scheme based on its more superior heuristic.

4.3.3 Convergence Time Tests Results

The benchmark scheme was faster by 0.4%, 0.78%, and 1.2% than the developed scheme in DoS only, DoS with FE, and spoofed DoS with flash event tests respectively. The time taken to return results shows corresponding tasks performed under the different conditions. In all the convergence time tests the benchmark scheme was slightly faster than the developed scheme. This is due to the fact that hop-by-hop search by itself is a time consuming and more difficult task than flow-based search method. But the fact that the margin was very negligible confirmed the effectiveness of SSO-DoSTBK using shark smell optimization algorithm (SSOA). Time is very important in IP traceback because the records of traffics on the routers are available for a limited time. SSOA-DoSTBK can be improved for faster convergence by adjusting the SSOA user define parameters or reducing the number of parameters for examining the traffic flows or both as may be necessary.

4.4 Quantified Comparison of Results

This quantified comparison focuses on the improvement of the developed SSOA-DoSTBK over the benchmark scheme, ACS-IPTBK, in the three tests under the three conditions for each test. The comparison is used to evaluate the functionality and efficiency of the SSOA-DoSTBK.

The quantified comparison values in Table 4.1 shows that the proposed scheme outperformed ACS-IPTBK in FER and Performance tests by, at least 5.2% in the worst condition and as much as over 32% in a less challenging condition. It also compared favourably with ACS-IPTBK in terms of convergence time by recording a convergence time with negligible differences compared to the time recorded by ACS-IPTBK as could be seen in Table 4.1.

Table 4.1: SSOA-DoSTBK results compared to ACS-IPTBK

	FER			**Performance**			**Convergence**		
Tests	DoS	DoS+FE	Spoofed DoS+FE	DoS	DoS+FE	Spoofed DoS+FE	DoS	DoS+FE	Spoofed DoS+FE
Percentage Improvement	31.8	32.06	28.45	4.76	11.6	5.2	-0.4	-0.78	-1.2

4.5 Discussions

The developed scheme is still far from been perfect based on the results presented in Table 4.1. Packet level examination of the flow records on the upstream hops along the attack path was performed during the tests to ensure accurate detection of the edges involved. Moreover, hop-by-hop search is a time-consuming process compared to least cost method for route reconstruction. SSO algorithm uses gradient metric. Gradient based algorithms are known to be fast which makes it a good candidate for the packet level examination of flow in hop-by-hop search. Step size is a parameter that is used to establish a balance between accuracy and speed for gradient based algorithms. The number of packet features to be used as parameters

for determining the attack flow on the packet routers can be chosen such as to achieve optimal result within the available time limit.

4.6 Packets Required for Attack Path Reconstruction

As against most of the other IP traceback schemes, including ACS-IPTBK, which require a large number of packets for attack path reconstruction, this scheme requires as few number of packets as much as the attack detector could use to detect an attack. It can be use with as little as one available packet. Hop-by-hop search used in this scheme had been used together with IP logging by Malik & Dutta (2017), for implementing single packet IP traceback.

Another benefit of the proposed scheme is that hop-by-hop search is used to track any data traffic that has been rerouted to find the source of the data.

CHAPTER FIVE

CONCLUSION AND RECOMMENDATIONS

5.1 Summary

Deep search for attack flow was implemented on ingress hops using discernment policy developed based on details extracted from the attack packets. The proposed scheme used the discernment policy in a hop-by-hop search to determine the most probable hops involved in routing the attack packets from the attacker to the victim. This was aimed at reducing errors in the attack path reconstruction that may cause failure to detect the true attack source.

The proposed scheme was implemented in NS2 version known as ns2-allinone-2.35 and was used for attack path reconstruction under different conditions with ordinary DoS attack alone, DoS attack with flash event present on the traced path, and spoofed DoS attack when there is presence of flash event surge traffic on the attack path. The performance of the developed scheme was measured based on FER, amount of the attack packets on the returned path, and convergence time. The developed scheme recorded 31.8%, 32.06%, and 28.45% lower FER for DoS only, DoS with FE, and spoofed DoS with flash event tests, respectively. It also recorded better efficiency in terms of correctness of attack packets detection on the attack path by 4.76%, 11.6%, and 5.2% higher performance in attack path detection for DoS only, DoS with FE, and spoofed DoS with flash event tests, respectively. But the ACS-IPTBK was faster than SSOA-DoSTBK in the attack path reconstruction by 0.4%, 0.78, and 1.2% for DoS only, DoS with FE, and spoofed DoS with flash event tests, respectively.

5.2 Conclusion

This research was aimed at the development of an Internet Protocol traceback scheme for detecting source of DoS attack based on SSO algorithm that cannot be disrupted by flash

event surge traffic. It was tested against ACS-IPTBK developed by (Wang *et al*., 2016) using the same test procedures under the same conditions. The result obtained from the proposed scheme when compared with those of ACS-IPTBK showed improvement over the ACS-IPTBK in terms of FER with as much as 32.06% and Performance tests with as much as 11.6%. ACS-IPTBK converges a little faster than the proposed scheme by the maximum recorded difference of 1.2% in convergence time at the worst test condition when there is flash event on the traceback routes and the DoS attack packets were spoofed.

The tests results show that ACS-IPTBK deviated further than SSOA-DoSTBK from the true attack path and that SSOA-DoSTBK is more effective for detecting source of spoofed IP attacks. The time difference between SSOA-DoSTBK and ACS-IPTBK convergence was negligibly small. The results indicated that SSOA-DoSTBK performed better in the detection of true DoS attack path because ACS-IPTBK works on the basis of parallelism whereby different agents examined different segments of the network concurrently to estimate a probable attack path. However, SSOA-DoSTBK performed a sequential search to detect the most probable attack path which resulted in a little longer convergence time. ACS-IPTBK examined more areas most of which are not on attack path but SSOA-DoSTBK narrowed its search to the most relevant area of the network based on defined heuristics and avoided confusing traffic flows. The detailed examination of the traffics enhanced SSOA-DoSTBK performance to return a more correct attack paths when attack packets were spoofed and flash event traffics were encountered during traceback process.

5.3 Significant Contributions

The significant contributions of this work are;

1. Development of SSOA-DoSTBK IP traceback scheme that can avoid flash event and other legitimate flows that may be symptomatically similar to attack traffic.

2. Discrimination policy created based on details extracted from the detected attack packets was incorporated into SSOA-DoSTBK IP traceback scheme for improved performance and to obtain more accurate results.
3. SSOA-DoSTBK IP traceback scheme outperformed ACS-IPTBK that it was benchmarked against by 31.8%, 32.06%, and 28.45% lower FER for DoS only, DoS with FE, and spoofed DoS with flash event tests, respectively and 4.76%, 11.6%, and 5.2% higher performance in the correctness of attack path detection for DoS only, DoS with FE, and spoofed DoS with flash event tests, respectively. But ACS-IPTBK was faster than SSOA-DoSTBK by just 0.4%, 0.78%, and 1.2% for DoS only, DoS with FE, and spoofed DoS with flash event tests, respectively.

5.4 Recommendations for Further Work

The proposed IP traceback scheme falls short of been perfect as observed from the results of tests conducted with it. There is room for improving it for better performance. The following areas are recommended for possible further research in future research works:

1. The choice of parameters used in the discrimination policy was not based on experimentation but they were just considered as relevant to identifying the traffic. Better method of determining the parameters of discrimination policy can be investigated for better convergence and better path detection. Data that are specific to the traffic and more persistent on the routers can allow more time for the traceback system.
2. The scheme was tested with simple DoS attack. Swarming of artificial sharks can be researched to make it applicable for detection of DDoS attack with large bots whereby different shark can trace different bots.

3. Attackers are information technology professionals and they understand the techniques used by defenders to develop preventive or corrective schemes against their attacks. Investigating the possible ways that attacker can exploit to thwart the operations of the proposed scheme in order to escape detection, e.g. replicating the attack traffic data on a false router to mislead the scheme, will make the proposed scheme more reliable.

References

Abedinia, O., & Amjady, N. (2015). Short-term wind power prediction based on Hybrid Neural Network and chaotic shark smell optimization. *International Journal of Precision Engineering and Manufacturing-Green Technology*, *2*(3), 245-254. DOI: 10.1007/s40684-015-0029-4

Abedinia, O., & Amjady, N. (2016). Net demand prediction for power systems by a new neural network-based forecasting engine, *Complexity, DOI:* 1099-0526 10.1002/cplx.21807 Retrieved from http://dx.doi.org/10.1002/cplx.21807

Abedinia, O., Amjady, N., & Ghasemi, A. (2014). A new metaheuristic algorithm based on shark smell optimization. *Complexity, 00*(00), 1-20. DOI: 10.1002/cplx.21634, Retrieved from http://dx.doi.org/10.1002/cplx.21634

Abedinia, O., Amjady, N., Yousefi, N., & Aramli, M. S. (2016). A Descriptive Study on Mathematical Model of Shark's Capabilities as a Successful Hunter. *Recent Advances in Biology and Medicine, 2*(-), 48-56. DOI: 10.18639/RABM.2016.02.292390, Retrieved from http://dx.doi.org/10.18639/RABM.2016.02.292390

Ahmadigorji, M., & Amjady, N. (2016). A multiyear DG-incorporated framework for expansion planning of distribution networks using binary chaotic shark smell optimization algorithm. *Energy, 102*, 199-215.

Ahmed, I. K., & Fapojuwo, A. O. (2016). Security threat assessment of simultaneous multiple Denial-of-Service attacks in IEEE 802.22 Cognitive Radio networks. Paper presented at the *IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM), Coimbra, 2016, pp.* 1-9. DOI: 10.1109/ *WoWMoM.2016.7523510*

Balachander, K., Dionee, V., Suganya, S. N., & Christina, T. J. (2017). A Viable IP Traceback VIA Dynamic Deterministic Packet Marking. *International Journal of Engineering Science and Computing, 7*(3), 5607-5609.

Bhagat, S., & Pasupuleti, S. K. (2015). Simulated Raindrop Algorithm to Mitigate DDoS Attacks in Cloud Computing. Paper presented at the *Proceedings of the Sixth International Conference on Computer and Communication Technology, Aliahabad, India, 2015, pp.* 412-418. http://dx.doi.org/10.1145/2818567.2818684

Bhandari, A., Sangal, A. L., & Kumar, K. (2016). Characterizing flash events and distributed denial- of- service attacks: an empirical investigation. *Security and Communication Networks, 9*(13), 2222-2239. 10.1002/sec.1472

Bhavani, Y., Janaki, V., & Sridevi, R. (2015). IP traceback through modified probabilistic packet marking algorithm using Chinese remainder theorem. *Ain Shams Engineering Journal, 6*(2), 715-722.

Bhuyan, M. H., Bhattacharyya, D., & Kalita, J. (2016). E-LDAT: a lightweight system for DDoS flooding attack detection and IP traceback using extended entropy metric. *Security and Communication Networks, 9*(16), 3251-3270. 10.1002/sec.1530

Bhuyan, M. H., Kashyap, H. J., Bhattacharyya, D. K., & Kalita, J. K. (2014). Detecting distributed denial of service attacks: methods, tools and future directions. *The Computer Journal, 57*(4), 537-556.

Chawla, S., Sachdeva, M., & Behal, S. (2016). Discrimination of DDoS attacks and Flash Events using Pearson's Product Moment Correlation Method. *International Journal of Computer Science and Information Security, 14*(10), 382-389.

Chen, R., Park, J., & Marchany, R. (2006). TRACK: A novel approach for defending against distributed denial-of-service attacks. *Technical Report TR ECE-06-02.Dept.of Electrical and Computer Engineering*, *14*(10), 382-389.

Claise, B., Vamsidhar Valluri, Martin, D. & Ganesh, S. (2004). Request for Comments: 3954.*Network Working Group;* Internet Engineering Task Force (IETF) Retrieved from https://www.ietf.org/rfc/rfc3954.txt(Accessed on: 2018-05-06)

Cybercrime Act. (2015). CYBERCRIMES (PROHIBITION, PREVENTION, ETC) ACT, 2015.*CENTRE FOR LAWS OF THE FEDERATION OF NIGERIA;* Retrieved from http://lawnigeria.com/LawsoftheFederation/Cyber-Crime-Act,-2015.html(Accessed on: 2018-05-03)

Data Protection Act. (1998). Data Protection Act 1998.*The National Archives;* Retrieved from http://www.legislation.gov.uk/ukpga/1998/29/contents;(Accessed on: 03-05-2018)

Daya, B. (2013). Network security: History, importance, and future.*University of Florida Department of Electrical and Computer Engineering;* Retrieved from http://www.alphawireless.co.za/wp-content/uploads/2013/01/Network-Security-article.pdf;(Accessed on: 2018-3-29)

Dhingra, A., & Sachdeva, M. (2014). *Recent Flash Events: A Study* International Conference on Communication, Computing & Systems. - Retrieved from http://sbsstc.ac.in/icccs2014/Papers/Paper20.pdf

Ehteram, M., Allawi, M. F., Karami, H., Mousavi, S., Emami, M., Ahmed, E., & Farzin, S. (2017). Optimization of Chain-Reservoirs Operation with a New Approach in Artificial Intelligence. *Water Resources Management, 31*(7), 2085-2104.

Ghaffari, S., Aghajani, G., Noruzi, A., & Hedayati Mehr, H. (2016). Optimal economic load dispatch based on wind energy and risk constrains through an intelligent. *Complexity Algorithm, 21*(S2), 494-506.

Gnanasekaran, N., Chandramohan, S., Kumar, P. S., & Imran, A. M. (2016). Optimal placement of capacitors in radial distribution system using shark smell optimization algorithm. *Ain Shams Engineering Journal, 7*(2), 907-916.

Gupta, S. G., Ghonge, M. M., Thakare, P. D., & Jawandhiya, P. (2013). Open-source network simulation tools: An overview. *International Journal of Advanced Research in Computer Engineering \& Technology (IJARCET), 2*(4), pp-1629.

Hamedi-Hamzehkolaie, M., Sanei, R., Chen, C., Tian, X., & Nezhad, M. K. (2014). Bee-based IP traceback. Paper presented at the *Fuzzy Systems and Knowledge Discovery (FSKD), 2014 11th International Conference On*, 968-972.

Hemant Pareek, V. S. (2014). Temporal lensing and its application in pulsing denial-of-service attacks. *International Journal of Advanced Research in Computer and Communication Engineering, 3*(2), 5777-5782.

ISO/IEC. (2009). Information technology — Security techniques — Information security management systems — Overview and vocabulary.*International Standard No. ISO/IEC 27000;* ISO/IEC Switzerland

Kalmijn, A. J. (1982). Electric and magnetic field detection in elasmobranch fishes. *American Association for the Advancement of Science, 218*(4575), 916-918.

Kempster, R. M., & Collin, S. P. (2011). Electrosensory pore distribution and feeding in the megamouth shark Megachasma pelagios (Lamniformes: Megachasmidae). *Aquatic Biology, 11*(3), 225-228.

Kevin, F., & Kannan, V. (2015). *The ns Manual* .ISI Information Sciences Institute. ISI Retrieved from https://www.isi.edu/nsnam/ns/doc/ns_doc.pdf (Accessed on: 2018-04-20)

Latif, R., Abbas, H., Latif, S., & Masood, A. (2016). Distributed denial of service attack source detection using efficient traceback technique (ETT) in cloud-assisted healthcare environment. *Journal of Medical Systems, 40*(7), 1-13.

Maheshwari, R., & Krishna, C. R. (2013). Mitigation of DDoS attacks using probability based distributed hop count filtering and round trip time. *International Journal of Engineering Research & Technology, 2*(7), 1135-1140.

Malik, M., & Dutta, M. (2017). Implementation of single-packet hybrid IP traceback for IPv4 and IPv6 networks. *IET Information Security*, Retrieved from http://digital-library.theiet.org/content/journals/10.1049/iet-ifs.2015.0483 (Accessed: 2018-02-15)

Mary, D. S. N., & Begum, A. T. (2017). An Algorithm for Moderating DoS Attack in Web Based Application. Paper presented at the *Technical Advancements in Computers and Communications (ICTACC), 2017 International Conference On*, 26-31.

Mohamed, M. A., Jamil, N., Abidin, A. F., Din, M. M., Nik, W. N. S. W., & Mamat, A. R. (2018). Entity-based parameterization for distinguishing distributed denial of service from flash events. *International Journal of Engineering & Technology, 7*(2.14), 5-8.

Montgomery, J., & Bodznick, D. (1999). Signals and noise in the elasmobranch electrosensory system. *Journal of Experimental Biology, 202*(10), 1349-1355.

Murugesan, V., Shalinie, M., & Neethimani, N. (2014). A Brief Survey of IP Traceback Methodologies. *Acta Polytechnica Hungarica, 11*(9), 197-216.

NetFlow Commands. (2009). *Cisco IOS NetFlow Command Reference* (Release 12.3 T ed.) CISCO. Retrieved from https://www.cisco.com/en/US/docs/ios/12_3t/netflow/command/reference/nfl_a1gt_ps5207_TSD_Products_Command_Reference_Chapter.html#wp1185290id=2307856&ftid=1257768&dwn=1&CFID=221391783&CFTOKEN=95877803 (Accessed on: 2018-05-06)

Noruzi, A. (2016). Optimal wide-area hybrid fuzzy controller to enhance the stability in power system. *International Journal of Ambient Energy*, , 1-13.

Qian, W., Dunlap, T., Cho, Y., & Qu, G. (2017). DoS attacks and countermeasures on network devices. Paper presented at the *Wireless and Optical Communication Conference (WOCC), 2017 26th}*, 1-6.

Rasti, R., Murthy, M., Weaver, N., & Paxson, V. (2015). Temporal lensing and its application in pulsing denial-of-service attacks. Paper presented at the *Security and Privacy (SP), 2015 IEEE Symposium On*, 187-198.

Roughan, M., Tuke, J., & Parsonage, E. (2015). Estimating the parameters of the Waxman random graph. *arXiv Preprint arXiv:1506.07974*, Retrieved from https://arxiv.org/pdf/1506.07974 (Accessed: 2017-06-25)

Router Data Flow Overview. (2017). Router Data Flow Overview.*Flow of Routing Information;* Juniper networks(Accessed on: 2018-05-06)

Sachdeva, M., & Kumar, K. (2014). A traffic cluster entropy based approach to distinguish DDoS Attacks from flash event using DETER testbed. *ISRN Communications and Networking, 2014LK http://downloads.hindawi.com/journals/isrn.communications.networking/2014/259831.pdf* (Accessed: 2017-08-14)

Saini, A., Ramakrishna, C., & Sachin, K. (2017). A Hybrid Approach to Address IP Traceback Problem using Nature Inspired Algorithm. *International Conference on Information Technology, Communications and Computing*, 10.5281/zenodo.1130621

Saurabh, S., & Sairam, A. S. (2013). A more accurate completion condition for attack-graph reconstruction in Probabilistic Packet Marking algorithm. Paper presented at the *Communications (NCC), 2013 National Conference on}*, 1-5.

Saurabh, S., & Sairam, A. S. (2016). Increasing Accuracy and Reliability of IP Traceback for DDoS Attack Using Completion Condition. JFS IJ Network Security.*18*(2), 224-234.

Schmitz, S. (2013). The US SOPA and PIPA- a European perspective. *International Review of Law, 27*(1-2), 213-229.

Setayeshi, S., Rad, V. B., Noruzi, A., & Yousefi, N. (2017). Optimal Design of Wide Area Based on Fuzzy Controller and Intelligent Method. *International Journal of Software Engineering and Knowledge Engineering, 27*(03), 449-473.

Shayanfar, H., Abedinia, O., Amjady, N., & Rajaei, S. (2016). Wind Power Prediction Model based on Hybrid Strategy. Paper presented at the *Proceedings of the International Conference on Scientific Computing (CSC),* 66. Retrieved from https://pdfs.semanticscholar.org/2eb9/a9270fecd05a4d2198b5cdb9ed6079f53510.pdf (Accessed: 2017-11-24)

Stone, R. (2000). CenterTrack: An IP Overlay Network for Tracking DoS Floods. Paper presented at the *USENIX Security Symposium*, , *21* 114.

Suresh, S., & Ram, N. S. (2018). Feasible DDoS attack source traceback scheme by deterministic multiple packet marking mechanism. *The Journal of Supercomputing,* , 1-15.

Tan, Y., Sengupta, S., & Subbalakshmi, K. (2010). Coordinated denial-of-service attacks in IEEE 802.22 networks. Paper presented at the *International Conference On Communications (ICC), May 2010 IEEE,* 1-5, Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.669.3711&rep=rep1&type=pdf

Tiwari, A., Jaiswal, M., Varma, N., & Joshi, B. (2014). TCP/IP Protocol Suite, Attacks, and Security Tools. *International Research Journal of Management Science \& Technology,* , 14-26. Retrieved from https://www. academia. edu/7134687/TCP_IP_Protocol_Suite_Attacks_and_Security_Tools (Accessed: 2016-12-23)

Venkataramanan, N., & Ravi, T. (2017). An Enhanced IP Trace Back Mechanism by using Particle Swarm System. *International Journal on Future Revolution in Computer Science & Communication Engineering, 3*(9), 48-51.

Wang, P., Lin, H., & Wang, T. (2016). An improved ant colony system algorithm for solving the IP traceback problem. *Information Sciences, 326,* 172-187. 10.1016/j.ins.2015.07.006

Yu, S., Zhou, W., Guo, S., & Guo, M. (2016). A feasible IP traceback framework through dynamic deterministic packet marking. *IEEE Transactions on Computers, 65*(5), 1418-1427.

Zargar, S. T., Joshi, J., & Tipper, D. (2013). A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. *IEEE Communications Surveys & Tutorials, 15*(4), 2046-2069.

APPENDICES

Appendix A-I: C++ Code listing for SSOADOSTBK Header file

Content of SSOADOSTBK.h

```
#ifndef __SSOADOSTBK_H__
#define __SSOADOSTBK_H__

//From NS
#include "agent.h"
#include "node.h"
#include "tclcl.h"

//From SSOADOSTBK Toolkit
#include "Randoms.cpp"

class SSOADOSTBK : public Agent {
public :
        SSOADOSTBK ();
        virtual ~SSOADOSTBK ();
private:
        double AR1, AR2, NVEL;
        int BETA, ALPHA, ETA, nodei,  nodej, nodeEDGESj, ITERATIONS;
        signed int AR3;
        int INITIALNODE, int PSI0;
        double **nodeVel, **PSI,**RHO,**PHI, **normPSI;
        Void SAVERESULT();
        void connectNODES (int nodei, int nodej);
        void objFunction(int itr);
        void objDerivative(int itr);
        void fitEDGES (int nodei, int nodej);
        void init ();
        int **ATKPATH, **EDGE;
        Randoms *randoms;
/**/
protected :
        int command (int, const char*const*);
};
#endif /* __SSOADOSTBK_H__ */
```

Appendix A-II: C++ Code listing for SSOADOSTBK C-plusplus file

Content of SSOADOSTBK.cc

```
/* From SSOA DoSTBK */
#include "SSOADOSTBK.h"

// From NS
//#include "tclcl.h"
#include <stdio.h>
#include <string.h>
#include <iostream>
#include <cstdlib>
//From C++
#include <cmath>
#include <limits>
#include <climits>
using namespace std;
static class SSOADOSTBKClass : public TclClass {
        public:
                SSOADOSTBKClass() : TclClass("Agent/SSOADOSTBKOtcl") {}
                TclObject* create(int, const char*const*) {
                        return (new SSOADOSTBK());
                }
} SSOA_DOSTBK;
SSOADOSTBK::SSOADOSTBK() : Agent (PT_TCP) {
//      Bind variables of C++ and TCL
        bind("victimNode",&INITIALNODE);
        bind("node1",&nodei);
        bind("node2",&nodej);
        bind("iterNum",&ITERATIONS);
        bind("nodegde",&nodeEDGESj);
        bind("psi",&PSI0);
        bind("nvel",&NVEL);
}
/**
 * The command method, used to call the SSOADOSTBK methods
 */

int SSOADOSTBK::command (int argc, const char*const*argv) {

        if(argc >=2) {
       if(strcmp(argv[1], "SetInitialValues") == 0) {
```

```
init ();
            return(TCL_OK);
        }
    }
        if(argc >=2) {
        if(strcmp(argv[1], "fitnessofEDGES") == 0) {
          printf("\n Calculate fitness of edge \n end nodes: Node %d and Node %d \n", nodei, nodej);
                fitEDGES (nodei, nodej);
            return(TCL_OK);
        }
    }
        if(argc >=2) {
        if(strcmp(argv[1], "nodeConnections") == 0) {
          printf("\n Calculate fitness of edge \n end nodes: Node %d and Node %d \n", nodei, nodej);
                connectNODES (nodei, nodej);
            return(TCL_OK);
        }
    }
if(argc >=2) {
        if(strcmp(argv[1], "saveresults") == 0) {
                SAVERESULT();
            return(TCL_OK);
        }
    }
    return(Agent::command(argc, argv));
}
/**/
SSOADOSTBK::~SSOADOSTBK () {
          for(int i=0; i<ITERATIONS; i++) {
                delete [] BETA[i];
                delete [] ALPHA[i];
                delete [] ETA[i];
                delete [] PSI[i];
                delete [] RHO[i];
                delete [] PHI[i];
          }
          delete [] BETA;
          delete [] ALPHA;
          delete [] PHI;
          delete [] ETA;
          delete [] RHO;
          delete [] PSI;
}
```

```
void SSOADOSTBK::init () {
randoms = new Randoms (21);
        BETA                    = new int*[ITERATIONS];
        ALPHA                   = new double*[ITERATIONS];
        PHI             = new double*[NUMBEROFNODES];
        ETA = new double*[ITERATIONS];
        RHO                     = new double*[ITERATIONS];
        PSI                     = new int*[ITERATIONS];
        for(int i=0; i<ITERATIONS; i++) {
                PHI[i]          = new double[NUMBEROFNODES];
                RHO[i]          = new double[ITERATIONS];
                PSI[i]          = new double[ITERATIONS];
                BETA    =  4;
                ALPHA   =  0.1;
                ETA     =  0.9;
        }
        ATKPATH  = new int[20];         //Tabu list
        for (int i=0; i<20; i++) {
                ATKPATH[i] = 0;
        }
}
void SSOADOSTBK::fitEDGES (int nodej,nodeEDGESj) {
  for { int itr=0; itr < ITERATIONS; itr++}
        for (int t=0; t<nodeEDGESi; t++) {                  //Local search
                if {PSI(t) <= 0} {
                        PSI(t) = PSI0;
                }
                        if (GRAPH[nodej][nodet] != 1) {
                        continue;
                        }
                PHI(t) = PSI(t)/nodeEDGESj;
                        sumPSI += PSI(t);
                        sum += pow (PSI(t) - PHI(t), 2);
                        }
                ALPHA(t) = sum/nodeEDGESj;
                RHO(t) = PSI(t)/sumPSI;
                normPSI(t) = objFunction (t);
                normGrad(t) = objDerivative(t);
                double nodeVel1 = ETA*AR1*normGrad(t) + ALPHA*AR2*nodeVel[j][t];
                double nodeVel2 = BETA*nodeVel[j][t];
                if (nodeVel1 > nodeVel2) {
                nodeVel[j][t] = nodeVel2;
                } else {
                nodeVel[j][t] = nodeVel1;
                }
        normPSI(t) = normPSI(t) + nodeVel * 1;   //Change in time is taken as 1 seconds
                (double) fitEDGE INT_MIN;
                        if (normPSI(t) > fitEDGE) {                 //Select the optimal value
                        fitEDGE = normPSI(t);
```

```
ATKPATH(t) = EDGE(t);
                    GRAPH[nodej][nodet] = 0;
                              }
                    PSI(t) = fitEDGE;
          }
  }
}
void SSOADOSTBK::objFunction (int itr) {
          double smellCompo =
(PHI(itr)/ALPHA(itr))*(PSI(itr)/ALPHA(itr))*exp(pow(PSI(itr)/ALPHA(itr),RHO(itr)));
          return globalCandidate;
          }
double SSOADOSTBK::objDerivative(int itr) {
          double b = PHI(itr)/ALPHA(itr);
          double a = ALPHA(itr);
          double c = RHO(itr);
          double  smellGrad  =  (b*exp((PSI(itr)/a)^c))/a  +  (b*c*PSI(itr)*exp((PSI(itr)/a)^c)*(PSI(itr)/a)^(c  -
1))/a^2;
          return smellGrad;
}
void SSOADOSTBK::connectNODES (int nodei, int nodej) {
          GRAPH[nodei][nodej] = 1;
          GRAPH[nodej][nodei] = 0;
          nodeVel[nodei][nodej] = NVEL;   //Product of delay and transmission rate for the edge
          AR1 = randoms -> Uniforme();
          AR2 = AR1 = randoms -> Uniforme();
}

Void SSOADOSTBK::SAVERESULT(){
for (int i=0; i<sizeof(ATKPATH); i++) {
                    cout << ATKPATH[i] << " ";
          string returnedPath= ATKPATH[i] << "\n";
          ofstream os("traceout.txt");
          if (!os) {cerr<<"Error writing to ..."<<endl; } else {
  os << returnedPath;
}
}
```

Appendix A-III: OTCL Code listing for calling SSOADOSTBK TCL file

ACStrc.tcl

```
set ns [new Simulator]
set ACStrc [new Agent/ACSIPtbkOtcl]
set subgroups 2
set Totalants 32
set numofNodes 200
        $ACStrc set antAgents 1
        $ACStrc set popSize 200
        $ACStrc set victimNode $vctm
        $ACStrc set iterNum 100
        set dsv [open dst0.nam r]
        set content [read $dsv]
        close $dsv
        set recs [split $content "\n"]
        set i 0
        foreach rec $recs {
        set flds [split $rec " "]
        lassign $flds evnt  tt tme ss src dd dst aa pkt ee trmsrt oo dly ii idn
        if {$evnt == "l"} {
        $ACStrc set node1 $src
        $ACStrc set node2 $dst
        $ACStrc set nvel $trmsrt * $dly
        $ACStrc connectNODES
        set atkPKTS($src,$dst,pkts) 0
        set srcEdges($src) 0
        foreach rec $recs {
        set flds [split $rec " "]
        lassign $flds evnt0  tt0 tme0 ss0 src0 dd0 dst0 aa0 pkt0 ee0 pktsz oo0 dly0 ii0 idn0
        if {$src0 == $src && $dst0 == $dst && $evnt == "r"} {
        if {$pkt0 == $atktype && $pktsz == $atksz} {
        set $atkPKTS($src,$dst,pkts) [expr $atkPKTS($src,$dst,pkts) + 1]
        }       }
        if {$evnt == "l" && $ss0 == $src} {
        set $srcEdges($src) [expr $srcEdges($src) + 1]
        }       }
        $ACStrc set psi $atkPKTS($src,$dst,pkts)
        $ACStrc set nodegde $srcEdges($src)
        }
        $ACStrc SetInitialValues
        $ACStrc nodeConnections
        $ACStrc fitnessofEDGES
        }
        $ACStrc saveresults
$ns run
```

Appendix A-IV: C++ Code listing for ACS-IPTBK Header file

Contents of ACSIPTBK.h

```
#ifndef __ACSIPtbk_H__
#define __ACSIPtbk_H__

//From NS
#include "agent.h"
#include "node.h"
#include "tclcl.h"

//From ACSIPtbk Toolkit
#include "Randoms.cpp"

class ACSIPtbk : public Agent {
public :
        ACSIPtbk ();
        virtual ~ACSIPtbk ();

private:
        double costs;
        int nodei,  nodej, ITERATIONS, noden, totalFlows;

        int NUMBEROFANTS, NUMBEROFNODES, INITIALNODE, PKTCOUNT, spoofed, update;
        double ALPHA, BETA, C, RO, MU, W, TAUMAX, q, q0, del;

        void connectNODES (int nodei, int nodej);
        void setNODEPOSITION (int node, double x, double y);
        void updatePHEROMONES ();
        void printPHEROMONES ();
        void printGRAPH ();
        void printRESULTS ();
        void init ();
        int node ();
        void route (int antk);
        int valid (int antk, int iteration);
        void optimize (int ITERATIONS);
        double routnPkts (int nodei, int nodej);
        bool exists (int nodei, int nodec);
        bool visited (int antk, int c);
        double PHI (int nodei, int nodej, int antk);
        double edge (int antk);
        double BESTEDGE;
        int **ATKPATH;

        int **GRAPH, **ROUTES, **PKTS, **PHIVALS;
        double **NODES, **PHEROMONES, **DELTAPHEROMONES, **PROBS;
        Randoms *randoms;
/**/
protected :
        int command (int, const char*const*);
};
#endif /* __ACSIPtbk_H__ */
```

Appendix A-V: C++ Code listing for ACS-IPTBK C-plusplus file

Content of ACPIPTBK.cc

```
/* From ACS IPTBK */
#include "ACSIPtbk.h"
// From NS
//#include "tclcl.h"
#include <stdio.h>
#include <string.h>
#include <iostream>
#include <cstdlib>
#include <fstream.h>
//From C++
#include <cmath>
#include <limits>
#include <climits>
using namespace std;
static class ACSIPtbkClass : public TclClass {
        public:
                ACSIPtbkClass() : TclClass("Agent/ACSIPtbkOtcl") {}
                TclObject* create(int, const char*const*) {
                        return (new ACSIPtbk());
                }
} ACS_IPtbk;
ACSIPtbk::
        bind("smllq0",&q0); ACSIPtbk() : Agent (PT_TCP) {
//      Bind variables of C++ and TCL
        bind("antAgents",&NUMBEROFANTS);
        bind("numNodes",&NUMBEROFNODES);
        bind("initNode",&INITIALNODE);
        bind("alfa",&ALPHA);
        bind("betta",&BETA);
        bind("ku",&C);
        bind("w",&W);
        bind("spfd",&spoofed);
        bind("updt",&update);
        bind("smllq",&q);
        bind("rau",&RO);
        bind("picTau",&TAUMAX);
        bind("node1",&nodei);
        bind("node2",&nodej);
        bind("node1",&noden);
        bind("node2",&costs);
        bind("node3",&totalFlows);
        bind("iterNum",&ITERATIONS);
        bind("srctodest",&AtkPktsj);
        bind("srcpkts",&AtkPktsi);
}
```

```
/**
 * The command method, used to call the ACSIPtbk methods
 */
int ACSIPtbk::command (int argc, const char*const*argv) {
          if(argc >=2) {
        if(strcmp(argv[1], "SetInitialValues") == 0) {
                    init ();
            return(TCL_OK);
        }
    }
          if(argc >=2) {
        if(strcmp(argv[1], "connectNODES") == 0) {
          printf("\n From connectNODES \n Nodes to connect: Node %d and Node %d \n", nodei, nodej);
                    connectNODES (nodei, nodej);
            return(TCL_OK);
        }
    }
          if(argc) {
        if(strcmp(argv[1], "setNODEPOSITION") == 0) {
          printf("\n From setNODEPOSITION \n  Number of args: %d \n", argc);
          printf("\n Node %d Packet flows: link cost %d, total packet flows %d \n", costs, flows,
totalFlows);
                    setNODEPOSITION (noden, costs, totalFlows);
            return(TCL_OK);
        }
    }
          if(argc == 2) {
        if(strcmp(argv[1], "printGRAPH") == 0) {
            printGRAPH ();
            return(TCL_OK);
        }
    }
          if(argc == 2) {
        if(strcmp(argv[1], "printPHEROMONES") == 0) {
                    printPHEROMONES ();
            return(TCL_OK);
        }
    }
          if(argc == 2) {
        if(strcmp(argv[1], "optimize") == 0) {
                    optimize (ITERATIONS);
            return(TCL_OK);
        }
    }
   if(argc == 2) {
        if(strcmp(argv[1], "printRESULTS") == 0) {
                    printRESULTS ();
            return(TCL_OK);
        }    }
```

```
return(Agent::command(argc, argv));
}
/**/
ACSIPtbk::~ACSIPtbk () {
	for(int i=0; i<NUMBEROFNODES; i++) {
		delete [] GRAPH[i];
		delete [] NODES[i];
		delete [] PHEROMONES[i];
		delete [] DELTAPHEROMONES[i];
		if(i < NUMBEROFNODES - 1) {
			delete [] PROBS[i];
		}
	}
	delete [] GRAPH;
	delete [] NODES;
	delete [] PHEROMONES;
	delete [] DELTAPHEROMONES;
	delete [] PROBS;
}
void ACSIPtbk::init () {
	randoms = new Randoms (21);
	GRAPH		= new int*[NUMBEROFNODES];
	NODES		= new double*[NUMBEROFNODES];
	PHEROMONES	= new double*[NUMBEROFNODES];
	DELTAPHEROMONES = new double*[NUMBEROFNODES];
	PROBS		= new double*[NUMBEROFNODES-1];
	PHIVALS			= new int*[2];
	PKTS		= new int*[NUMBEROFNODES];
	PHIVALS[1]		= new int[2];
	PHIVALS[2]		= new int[2];
	PKTS		= new int[NUMBEROFNODES];
	for(int i=0; i<NUMBEROFNODES; i++) {
		GRAPH[i]			= new int[NUMBEROFNODES];
		NODES[i]			= new double[2];
		PHEROMONES[i]		= new double[NUMBEROFNODES];
		DELTAPHEROMONES[i]  = new double[NUMBEROFNODES];
		PROBS[i]			= new double[2];
		for (int j=0; j<2; j++) {
			NODES[i][j] = -1.0;
			PROBS[i][j]  = -1.0;
		}
		for (int j=0; j<NUMBEROFNODES; j++) {
			GRAPH[i][j]			= 0;
			PHEROMONES[i][j]		= 0.0;
			DELTAPHEROMONES[i][j]	= 0.0;
		}
	}
	ROUTES = new int*[NUMBEROFANTS];
```

```
for (int i=0; i<NUMBEROFANTS; i++) {
		ROUTES[i] = new int[NUMBEROFNODES];
		for (int j=0; j<NUMBEROFNODES; j++) {
			ROUTES[i][j] = -1;
		}
	}
	BESTEDGE = (double) INT_MAX;		//Best fit edge to be added to tabu list
	ATKPATH  = new int[NUMBEROFNODES];		//Tabu list
	for (int i=0; i<NUMBEROFNODES; i++) {
		ATKPATH[i] = -1;
	}
}
void ACSIPtbk::connectNODES (int nodei, int nodej) {
	GRAPH[nodei][nodej] = 1;
	PHEROMONES[nodei][nodej] = randoms -> Uniforme() * TAUMAX;
	printf("Pheromones for %d and %d is %4.2f", nodei, nodej, PHEROMONES[nodei][nodej]);
	GRAPH[nodej][nodei] = 1;
	PHEROMONES[nodej][nodei] = PHEROMONES[nodei][nodej];
}
void ACSIPtbk::setNODEPOSITION (int node, double flid, double flwpkts) {
	NODES[node][0] = flid;
	NODES[node][1] = flwpkts;
}
void ACSIPtbk::printPHEROMONES () {
	printf(" PHEROMONES: \n");
	printf("    | ");
	for (int i=0; i<NUMBEROFNODES; i++) {
		printf("%5d   ", i);
	}
	printf("\n - | ");
	for (int i=0; i<NUMBEROFNODES; i++) {
		printf("--------");
	}
	printf("\n");
	for (int i=0; i<NUMBEROFNODES; i++) {
		printf( " %d | ", i);
		for (int j=0; j<NUMBEROFNODES; j++) {
			if (i == j) {
				printf ("%5s   ", "x");
				continue;
			}
			if (exists(i, j)) {
				printf ("%7.3f ", PHEROMONES[i][j]);
			}
			else {
				if(PHEROMONES[i][j] == 0.0) {
					printf ("%5.0f   ", PHEROMONES[i][j]);
				}
				else {
```

```
                                        printf ("%7.3f ", PHEROMONES[i][j]);
                                }
                        }
                }
                printf ("\n");
        }
        printf ("\n");
}
double ACSIPtbk::routnPkts (int nodei, int nodej) {
        PKTCOUNT = NODES[nodej][0];
}
bool ACSIPtbk::exists (int nodei, int nodec) {
        return (GRAPH[nodei][nodec] == 1);
}
bool ACSIPtbk::visited (int antk, int c) {
        for (int l=0; l<NUMBEROFNODES; l++) {
                if (ROUTES[antk][l] == -1) {
                        break;
                }
                if (ROUTES[antk][l] == c) {
                        return true;
                }
        }
        return false;
}
double ACSIPtbk::PHI (int nodei, int nodej, int antk) {          //Compute transition rule
        double ETAij = (double) pow (1 / routnPkts (nodei, nodej), BETA);
        double TAUij = (double) pow (PHEROMONES[nodei][nodej],   ALPHA);
        double sum = 0.0;
        for (int c=0; c<NUMBEROFNODES; c++) {
                if (exists(nodei, c)) {
                        if (!visited(antk, c)) {
                                double ETA = (double) pow (1 / routnPkts (nodei, c), BETA);
                                double TAU = (double) pow (PHEROMONES[nodei][c],   ALPHA);
                                PHIVALS(1) = ETA * TAU;
                                sum += ETA * TAU;
                                PHIVALS(2) = PHIVALS(1)/sum
                        }
                }
        }
}
double ACSIPtbk::edge (int antk) {                //Compute routing packets on the edge
        double sum = 0.0;
        for (int j=0; j<NUMBEROFNODES-1; j++) {
                sum += routnPkts (ROUTES[antk][j], ROUTES[antk][j+1]);
        }
        return sum;
}
```

```
int ACSIPtbk::node () {                          //Update pheromone on nodes
        double xi = randoms -> Uniforme();
        int i = 0;
        double sum = PROBS[i][0];
        while (sum < xi) {
                i++;
                sum += PROBS[i][0];
        }
        return (int) PROBS[i][1];
}
void ACSIPtbk::route (int antk) {
        ROUTES[antk][0] = INITIALNODE;
        for (int i=0; i<NUMBEROFNODES-1; i++) {
                int nodei = ROUTES[antk][i];
                int count = 0;
                for (int c=0; c<NUMBEROFNODES; c++) {
                        if (nodei == c) {
                                continue;
                        }
                        if (exists (nodei, c)) {
                                if (!visited (antk, c)) {
                                        if (q <= q0){
                                        } else {
                                PROBS[count][0] = PHIVALS(1);  //Compute probability for nodes
                                        } else {
                                        PROBS[count][0] = PHIVALS(2);
                                        }
                                        PROBS[count][1] = (double) c;
                                        count++;
                                }

                        }
                }
                // deadlock
                if (0 == count) {
                        return;
                }
                ROUTES[antk][i+1] = node();
        }
}
int ACSIPtbk::valid (int antk, int iteration) {
        for(int i=0; i<NUMBEROFNODES-1; i++) {
                int nodei = ROUTES[antk][i];
                int nodej = ROUTES[antk][i+1];
                if (nodei < 0 || nodej < 0) {
                        return -1;
                }
                if (!exists(nodei, nodej)) {
                        return -2;
                }
```

```
for (int j=0; j<i-1; j++) {
                        if (ROUTES[antk][i] == ROUTES[antk][j]) {
                                return -3;
                        }
                }
        }
        if (!exists (INITIALNODE, ROUTES[antk][NUMBEROFNODES-1])) {
                return -4;
        }
        return 0;
}
void ACSIPtbk::printGRAPH () {
        cout << " GRAPH: Number of nodes: " << NUMBEROFNODES << endl;
        cout << "    | ";
        for( int i=0; i<NUMBEROFNODES; i++) {
                printf("%d  ", i);
        }
        printf("\n  - | ");
        for (int i=0; i<NUMBEROFNODES; i++) {
                printf("-  ");
        }
        cout << endl;
        int count = 0;
        for (int i=0; i<NUMBEROFNODES; i++) {
                printf("  %d | ", i);
                for (int j=0; j<NUMBEROFNODES; j++) {
                        if(i == j) {
                                printf("x  ");
                        }
                        else {
                                printf("%d  ", GRAPH[i][j]);
                        }
                        if (GRAPH[i][j] == 1) {
                                count++;
                        }
                }
                printf("\n");
        }
        printf("\n");
        printf("Number of connections: %d \n\n", count);
}
void ACSIPtbk::printRESULTS () {
        BESTEDGE += routnPkts (ATKPATH[NUMBEROFNODES-1], INITIALNODE);
        printf(" ATTACK PATH:\n");
        for (int i=0; i<sizeof(ATKPATH); i++) {
                cout << ATKPATH[i] << " ";
        string returnedPath= ATKPATH[i] << "\n";
        ofstream os("traceout.txt");
```

```
        if (!os) {cerr<<"Error writing to ..."<<endl; } else {
 os << returnedPath;
}
        }
        printf("\n edge: %4.2f \n", BESTEDGE);
}
void ACSIPtbk::updatePHEROMONES () {            //Local updating
        for (int k=0; k<NUMBEROFANTS; k++) {
                double redge = edge(k);                 //Routing packets used in local upadte
                for (int r=0; r<NUMBEROFNODES-1; r++) {
                        int nodei = ROUTES[k][r];
                        int nodej = ROUTES[k][r+1];
                        if (update == "global") {
                        DELTAPHEROMONES[nodei][nodej] += C / redge;
                        DELTAPHEROMONES[nodej][nodei] += C / redge;
                        }
                }
        }
        if (update == 1) {                              //Local update
        for (int i=0; i<NUMBEROFNODES; i++) {
                for (int j=0; j<NUMBEROFNODES; j++) {
                        PHEROMONES[i][j]    =    (1    -    W)    *    PHEROMONES[i][j]    +
W*DELTAPHEROMONES[i][j];
                        DELTAPHEROMONES[i][j] = PHEROMONES[i][j];
                }
        }
        } else if (update == 2) {               //Global update
        for (int i=0; i<NUMBEROFNODES; i++) {
                for (int j=0; j<NUMBEROFNODES; j++) {
                        PHEROMONES[i][j] = (1 - RO) * PHEROMONES[i][j] + DELTAPHEROMONES[i][j];
                        DELTAPHEROMONES[i][j] = 0.0;
                }
        }
        }
        if (spoofed == 1) {                             //spoofed attack
                del = (AtkPktsi - AtkPktsj)/sizeof(ATKPATH);
                for (int i=0; i<NUMBEROFNODES; i++) {
                        for (int j=0; j<NUMBEROFNODES; j++) {
                                if ((routnPkts(i-1,i) - routnPkts(i,j)) < del) {
                                MU = RO*DELTAPHEROMONES[i][j];
                                } else {
                                        MU = -1*RO*DELTAPHEROMONES[i][j];
                                }
                PHEROMONES[i][j] = (1 - RO) * PHEROMONES[i][j] + MU;
                        }
                }
        }
```

```
void ACSIPtbk::optimize (int ITERATIONS) {
	for (int iterations=1; iterations<=ITERATIONS; iterations++) {
		cout << flush;
		printf("ITERATION %d HAS STARTED! \n\n", iterations);

		for (int k=0; k<NUMBEROFANTS; k++) {
			printf(" : ant %d has been released!\n", k);
			while (0 != valid(k, iterations)) {
				printf("  :: releasing ant %d again!n", k);
				for (int i=0; i<NUMBEROFNODES; i++) {
					ROUTES[k][i] = -1;
				}
				route(k);
			}
			for (int i=0; i<NUMBEROFNODES; i++) {
				printf(" %d ", ROUTES[k][i]);
			}
			printf("\n");
			printf("  :: route done\n");
			double redge = edge(k);
			if (redge < BESTEDGE) {
				BESTEDGE = redge;
				for (int i=0; i<NUMBEROFNODES; i++) {
					ATKPATH[i] = ROUTES[k][i];
				}
			}
			printf(" : ant %d has ended!\n", k);
		}
		printf("\nupdating PHEROMONES . . .\n");
		updatePHEROMONES ();
		printf("\n done!\n");
		printPHEROMONES ();
		for (int i=0; i<NUMBEROFANTS; i++) {
			for (int j=0; j<NUMBEROFNODES; j++) {
				ROUTES[i][j] = -1;
			}
		}
		printf("\n ITERATION %d HAS ENDED!", iterations);
	}
}
```

Appendix A-VI: C++ Code listing for ACS-IPTBK C-plusplus file (Random number generator)

Contents of Randoms.cpp (Courtesy: Diogo A. B. Fernandes. Copyright © 2015)

```
#include <cstdlib>
#include <cmath>
#include <limits>

class Randoms {
private:
	long xpto;
public:
	// Generator seed.
	Randoms (long x) {xpto = -x;}
	// Returns a random Gaussian number.
	double Normal (double avg, double sigma)
	{
		return (avg+sigma*gaussdev(&xpto)) ;
	}
	// Returns a uniform random number between 0 and 1.
	double Uniforme()
	{
		return ran1(&xpto);
	}
	// Returns a random number between -m and m.
	double sorte(int m)
	{
		return (1.0*rand())/(1.0*RAND_MAX)*2.0*m-m;
	}
/*
	Taken from Numerical Recipes in C, Chapter 7.
*/
#define	IA 16807
#define	IM 2147483647
#define	AM (1.0/IM)
#define	IQ 127773
#define	IR 2836
#define	NTAB 32
#define	NDIV (1+(IM-1)/NTAB)
#define	EPS 1.2e-7
#define	RNMX (1.0-EPS)
float ran1(long *idum)
```

```
/*
"Minimal" random number generator of Park and Miller with Bays-Durham shuffle and added
safeguards. Returns a uniform random deviate between 0.0 and 1.0 (exclusive of the endpoint values).
Call with idum a negative integer to initialize; thereafter, do not alter idum between successive deviates
in a sequence. RNMX should approximate the largest floating value that is less than 1.
*/
{
        int j;
        long k;
        static long iy=0;
        static long iv[NTAB];
        float temp;
        if (*idum <= 0 || !iy) {         // Initialize.
                if (-(*idum) < 1) *idum=1;     // Be sure to prevent idum = 0.
                else *idum = -(*idum);
                for (j=NTAB+7;j>=0;j--) {     // Load the shuffle table (after 8 warm-ups).
                        k=(*idum)/IQ;
                        *idum=IA*(*idum-k*IQ)-IR*k;
                        if (*idum < 0) *idum += IM;
                        if (j < NTAB) iv[j] = *idum;
                }
                iy=iv[0];
        }
        k=(*idum)/IQ;                 // Start here when not initializing.
        *idum=IA*(*idum-k*IQ)-IR*k;      // Compute idum=(IA*idum) % IM without over-
        if (*idum < 0) *idum += IM;      // flows by Schrage's method.
        j=iy/NDIV;                    //  Will be in the range 0..NTAB-1.
        iy=iv[j];                    // Output previously stored value and refill the
        iv[j] = *idum;                // shuffle table.
        if ((temp=AM*iy) > RNMX)
                return RNMX;                                    // Because users don't expect endpoint
values.
        else
                return temp;
}

float gaussdev(long *idum)
// Returns a normally distributed deviate with zero mean and unit variance,
// using ran1(idum) as the source of uniform deviates.
{
//   float ran1(long *idum);
        static int iset=0;
        static float gset;
        float fac,rsq,v1,v2;
        if (*idum < 0) iset=0;     //    Reinitialize.
        if (iset == 0) {          //    We don't have an extra deviate handy, so
         do {
                        v1=2.0*ran1(idum)-1.0;    // pick two uniform numbers in the square ex-
                        v2=2.0*ran1(idum)-1.0;    // tending from -1 to +1 in each direction,
                        rsq=v1*v1+v2*v2;         // see if they are in the unit circle,
```

```
  } while (rsq >= 1.0 || rsq == 0.0);  // and if they are not, try again.
            fac=sqrt(-2.0*log(rsq)/rsq);
            // Now make the Box-Muller transformation to get two normal deviates.
            // Return one and save the other for next time.
            gset=v1*fac;
            iset=1;             //  Set flag.
            return v2*fac;
  } else {              //   We have an extra deviate handy,
            iset=0;             //   so unset the flag,
            return gset;          //   and return it.
  }
}
};
```

Appendix A-VII: OTCL Code listing for calling ACS-IPTBK TCL file

ACStrc.tcl

```
set ns [new Simulator]
set ACStrc [new Agent/ACSIPtbkOtcl]
set subgroups 2
set Totalants 32
set numofNodes 200
        $ACStrc set antAgents $Totalants/$subgroups
        $ACStrc set numNodes 200
        $ACStrc set w 0.3
        $ACStrc set alfa 0.5
        $ACStrc set betta 0.8
        $ACStrc set ku 80
        $ACStrc set rau 0.2
        $ACStrc set picTau 2
        $ACStrc set initNode 0
        $ACStrc set iterNum 100
        $ACStrc set spfd 1
        set dsv [open dst0.nam r]
        set content [read $dsv]
        close $dsv
        set recs [split $content "\n"]
        set i 0
        foreach rec $recs {
        set flds [split $rec " "]
        lassign $flds evnt  tt tme ss src dd dst aa pkt ee trmsrt oo dly ii idn
        if {$evnt != "l" && $evnt != "A" && $evnt != "c" && $evnt != "n" && $evnt != "V" } {
        $ACStrc set node1 $src
        $ACStrc set node2 $dst
        $ACStrc connectNODES
        set nodePosVals($src,$dst,pkts) 0
        set nodePosVals($src,rcv,pkts) 0
        foreach rec $recs {
        set flds [split $rec " "]
        lassign $flds evnt0  tt0 tme0 ss0 src0 dd0 dst0 aa0 pkt0 ee0 trmsrt0 oo0 dly0 ii0 idn0
        if {$src0 == $src && $dst0 == $dst && $evnt == "r"} {
        set nodePosVals($src,$dst,pkts) [expr $nodePosVals($src,$dst,pkts) +1]
        }
        if {$src0 == $dst && $evnt == "r"} {
        set nodePosVals($src,rcv,pkts) [expr $nodePosVals($src,rcv,pkts) +1]
        }
        }
```

$ACStrc set srctodest $nodePosVals($src,$dst,pkts)

Content of Main_IPtraceback.tcl

```
#Copyright (c) 2017 Salami, Omoniyi Wale of the Ahmadu Bello University, Zaria.  All rights reserved.
# This script generates random network topology using random generator based on Waxman model.
#define communication ports
set com_port 42
# set simulator
set ns [new Simulator]
#       global ns com_port dt dsv
        # Network parameters
        set subD_nodes 8
        set subDs 25
        set nwk_nodes  [expr $subD_nodes * $subDs]
#Define different colors for data flows (for NAM)
        $ns color 1 Blue
        $ns color 2 Green
        $ns color 3 Red
$ #Open trace files
        set dt [open dst0.tr w]
        $ns trace-all $dt
        set dsv [open dst0.nam w+]
        $ns namtrace-all $dsv
## Generate Random Topology based on Waxman model
#creates nodes
        for {set i 0} {$i < $nwk_nodes} {incr i} {
        set nd($i) [$ns node]
}
##Generate random attack node number
# Defining a Random Uniform Generator
        set aRng [new RNG]
        $aRng seed 0
        #Attacker randomly generated
        set r1 [new RandomVariable/Uniform]
        $r1 use-rng $aRng
        $r1 set min_ 1
        $r1 set max_ [expr round($nwk_nodes.0/5.0)+1]
        # Victim randomly chosen
        set r2 [new RandomVariable/Uniform]
        $r2 use-rng $aRng
        $r2 set min_ [expr round($nwk_nodes.0/5.0)+1]
        $r2 set max_ [expr $nwk_nodes-1]
```

Appendix A-VIII: The Main OTCL Code listing for IP Traceback process: TCL file

```
set atckr [expr round([$r1 value])]
        set vctm [expr round([$r2 value])]
        #puts stdout "Attacker: node $atckr      Victim: node $vctm"

#Setup a TCP connection
set tcp [new Agent/TCP]
$tcp set class_ 3
$ns attach-agent $nd($atckr) $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $nd($vctm) $sink
$ns connect $tcp $sink
$tcp set fid_ 1

#Setup Ping connection
 Agent/Ping instproc recv {from rtt} {
        global ns
    $self instvar node_
    $ns trace-annotate "node [$node_ id] sent attack packets to \
        $from. Round-Trip-Time: $rtt ms."
}

#Create ping agents
set pngAgnt1 [new Agent/Ping]
set pngAgnt2 [new Agent/Ping]

#Attach ping agents to nodes
$ns attach-agent $nd($atckr) $pngAgnt1
$ns attach-agent $nd($vctm) $pngAgnt2
$pngAgnt1 set packetSize_ 10k
#Connect ping agents
$ns connect $pngAgnt1 $pngAgnt2
$pngAgnt1 set fid_ 3
#process for contonuous pinging
proc sendPoD {} {
        global ns pngAgnt1
        set iat_PoD [expr (1.0/1000.0)]
        set time [$ns now]
        $ns at [ expr $time + $iat_PoD] "sendPoD"
        #set bytes 512
        #$tcp1 send $bytes
        $ns at [$ns now] "$pngAgnt1 send"
}
#Setup a CBR over TCP connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $tcp
$cbr set type_ CBR
$cbr set packet_size_ 500
$cbr set rate_ 0.5mb
```

```
#$cbr set random_ false
$cbr set random_ true

# create edges connecting the nodes
for {set j 0} {$j < $subDs} {incr j} {
  for {set i 0} {$i < $subD_nodes} {incr i} {
        set cst1 [expr ($i+1)%$subD_nodes]
        if {[expr $j*$subD_nodes+($i+1)%$subD_nodes] == $vctm} {
    $ns duplex-link $nd([expr $j*$subD_nodes+($i+1)%$subD_nodes]) $nd([expr $j*$subD_nodes+$i])
0.03Kb [expr 3-$cst1%3]15ms DropTail
        $ns queue-limit $nd([expr $j*$subD_nodes+($i+1)%$subD_nodes]) $nd([expr
$j*$subD_nodes+$i]) 5
        } else {
    $ns duplex-link $nd([expr $j*$subD_nodes+$i]) $nd([expr $j*$subD_nodes+($i+1)%$subD_nodes])
[expr ($cst1%3+1)*100]Mb [expr 1.0/(3.0-$cst1%3)*0.5]ms DropTail
        $ns queue-limit $nd([expr $j*$subD_nodes+$i]) $nd([expr
$j*$subD_nodes+($i+1)%$subD_nodes]) [expr ($cst1%3+1)*100]
        }
  }
        set cst2 [expr ($j+1)%$subDs]
        if {[expr (($j+1)%$subDs)*$subD_nodes+2]==$vctm} {
        $ns duplex-link $nd([expr (($j+1)%$subDs)*$subD_nodes+2]) $nd([expr $j*$subD_nodes])
0.03Kb [expr 3-$cst2%3]15ms DropTail
        $ns queue-limit $nd([expr (($j+1)%$subDs)*$subD_nodes+2]) $nd([expr $j*$subD_nodes]) 5
        } else {
  $ns duplex-link $nd([expr $j*$subD_nodes]) $nd([expr (($j+1)%$subDs)*$subD_nodes+2]) [expr
($cst2%3+1)*100]Mb [expr 1.0/(3.0-$cst2%3)*0.5]ms DropTail
        $ns queue-limit $nd([expr $j*$subD_nodes]) $nd([expr (($j+1)%$subDs)*$subD_nodes+2]) [expr
($cst2%3+1)*100]
        }
  if {$j%2} {
        set cst3 [expr ($j+3)%$subDs]
        if {[expr (($j+3)%$subDs)*$subD_nodes+1]==$vctm} {
    $ns duplex-link $nd([expr (($j+3)%$subDs)*$subD_nodes+1]) $nd([expr $j*$subD_nodes+3])
0.03KMb [expr 3-$cst3%3]15ms DropTail
        $ns queue-limit $nd([expr (($j+3)%$subDs)*$subD_nodes+1]) $nd([expr $j*$subD_nodes+3]) 5
        } else {
    $ns duplex-link $nd([expr $j*$subD_nodes+3]) $nd([expr (($j+3)%$subDs)*$subD_nodes+1]) [expr
($cst3%3+1)*100]Mb [expr 1.0/(3.0-$cst3%3)*0.5]ms DropTail
        $ns queue-limit $nd([expr $j*$subD_nodes+3]) $nd([expr (($j+3)%$subDs)*$subD_nodes+1])
[expr ($cst3%3+1)*100]
        }
        }
}
# subclass Agent/MessagePassing to make it do flooding

Class Agent/MessagePassing/Flooding -superclass Agent/MessagePassing

Agent/MessagePassing/Flooding instproc send_message {size msgid msg} {
```

```
$self instvar messages_seen node_
   global ns com_port

# $ns trace-annotate "Node [$node_ node-addr] is sending {$msgid:$msg}"

   lappend messages_seen $msgid
   $self send_to_neighbors -1 $com_port $size "$msgid:$msg"
}

Agent/MessagePassing/Flooding instproc send_to_neighbors {skip port size data} {
   $self instvar node_

         foreach x [$node_ neighbors] {
            set addr [$x set address_]
            if {$addr != $skip} {
                  $self sendto $size $data $addr $port
            }
         }
}

Agent/MessagePassing/Flooding instproc recv {source sport size data} {
   $self instvar messages_seen node_
   global ns

   # extract message ID from message
   set message_id [lindex [split $data ":"] 0]


   if {[lsearch $messages_seen $message_id] == -1} {
         lappend messages_seen $message_id
    #   $ns trace-annotate "Node [$node_ node-addr] received {$data}"
         $self send_to_neighbors $source $sport $size $data
   } else {
         #$ns trace-annotate "Node [$node_ node-addr] received redundant copy of message
#$message_id"
   }
}

# attach a new Agent/MessagePassing/Flooding to each node on port $com_port
for {set i 0} {$i < $nwk_nodes} {incr i} {
   set a($i) [new Agent/MessagePassing/Flooding]
   $nd($i) attach  $a($i) $com_port
   $a($i) set messages_seen {}
}

# now set up some message events
set a1q [expr $nwk_nodes/4]
set a2q [expr $nwk_nodes*2/5]
set a3q [expr $nwk_nodes*3/4]
#$ns at 0.1 "$cbr start"
$ns at 0.2 "$a($a1q) send_message 900 1 {first message}"
$ns at 0.5 "$a($a2q) send_message 700 2 {another one}"
```

```
$ns at 1.0 "$a($a3q) send_message 500 abc {yet another one}"
$ns at 0.3 "sendPoD"
#$ns at 1.5 "$cbr stop"

#Schedule events for the CBR and FTP agents
$ns at 0.1 "$cbr start"
$ns at 0.101 "$tcp start"
$ns at 1.5 "$tcp stop"
$ns at 1.53 "$cbr stop"
#}

proc setSearch {} {
        global ns nd dsv dit dstv
        set dsv [open dst0.nam r]
        set content [read $dsv]
        close $dsv
        set recs [split $content "\n"]
        set i 0
        foreach rec $recs {
        set flds [split $rec " "]
        lassign $flds evnt  tt tme ss src dd dst aa pkt ee pktsz oo bb ii idn
        if {$evnt != "V" && $evnt != "A" && $evnt != "c" && $evnt != "n" && $evnt != "l" } {
        incr i
        set srchParas($i,1) $evnt
        set srchParas($i,2) $tme
        set srchParas($i,3) $src
        set srchParas($i,4) $dst
        set srchParas($i,5) $pkt
        set srchParas($i,6) $pktsz
        set srchParas($i,7) $idn

        if {$srchParas($i,1) == "d"} { incr drps
        if {$drps == 1}    { set fstdrp $i
        set trcStrt [expr $srchParas($fstdrp,2) + 0.1969]
        }
         }

        }
        }

set arrsz [expr [array size srchParas]/7]

set idNo $srchParas([expr $arrsz-1],7)

##Retrieve attack packets flow data
foreach k [array names srchParas] {

#Record drop packet event times sorted ascending
if { [regexp \[\d+,1\] $k match] && $srchParas($k) == "d"} {
set drpTme [string range $k 0 [string last , $k]-1]
```

```
set atkDrops($drpTme,2) $srchParas($drpTme,2)
        set atkDrops($drpTme,3) $srchParas($drpTme,3)
        set atkDrops($drpTme,4) $srchParas($drpTme,4)
        set atkDrops($drpTme,5) $srchParas($drpTme,5)
        set atkDrops($drpTme,6) $srchParas($drpTme,6)
        set atkDrops($drpTme,7) $srchParas($drpTme,7)

#Record dropped packets data
incr T
set flwid($T,pkt) $srchParas($drpTme,5)
set flwid($T,sz) $srchParas($drpTme,6)
set flwid($T,flid) $srchParas($drpTme,7)
set flwdrpkts($flwid($T,pkt),pkt) 0
}
##End foreach loop
}

set atkdropsz [expr [array size atkDrops]/6]

if {$atkdropsz > 10} {
set pktdrps 1

##Sort packets drop records based on time ascending

set minDrpTme 0
set t 1
set j 1
set q 1
foreach j [array names atkdropsz] {
set jf [string range $j 0 [string last , $j]-1]
set nxtDrpTme $atkDrops($jf,2)

foreach q [array names atkdropsz] {
set qf [string range $q 0 [string last , $q]-1]
if { $minDrpTme < $atkDrops($qf,2) < $nxtDrpTme } {
set indx $j
set nxtDrpTme $srchParas($qf,2)
}
##End foreach q
}
set atkPathSegs($t) $nxtDrpTme
set minDrpTme $atkPathSegs($t)
incr t
##End foreach j
}
} else {
set pktdrps 0
}
```

```
if { $pktdrps > 0 } {
#Detects maximum flow ID
set maxflw 0
foreach k [array names flwid] {
set IDfl [string range $k 0 [string last , $k]-1]
if { $flwid($IDfl,flid) > $maxflw } {
set maxflw $flwid($IDfl,flid)
}
##End foreach loop
}

#Detect flow that generated highest dropped packets
set flwidsz [expr [array size flwid]/3]
set D 0
while { $D < [expr $maxflw+1]} {
        for {set f 1} {$f < $flwidsz+1} {incr f} {
        if {$flwid($f,flid) == $D } {
        set flwdrpkts($flwid($f,pkt),sz) $flwid($f,sz)
        set flwdrpkts($flwid($f,pkt),flid) $flwid($f,flid)
        incr flwdrpkts($flwid($f,pkt),pkt)
        }
   }
incr D
##End while loop
 }
set maxOccur 0
foreach g [array names flwdrpkts] {
set idxmaxOccur [string range $g 0 [string last , $g]-1]
if { $flwdrpkts($idxmaxOccur,flid) > $maxOccur } {
set maxOccur $flwdrpkts($idxmaxOccur,flid)
}
##End foreach loop
 }
puts "$idxmaxOccur generated $flwdrpkts($idxmaxOccur,pkt) number of dropped packets, the highest.\nDetected DoS attack flow is $idxmaxOccur with packet size $flwdrpkts($idxmaxOccur,sz) and flow ID $flwdrpkts($idxmaxOccur,flid)\n"

##Receive Attack path returned from IP Traceback schemes
set rsltfl [open traceout.txt r]
        set content [read $rsltfl]
        close $dsv
        set links [split $content "\n"]
        foreach rec $links {
        set hops [split $rec " "]
        lassign $hops algorithm src  dst
        set atkPath($r,algo) $algorithm
        set atkPath($r,src) $src
        set atkPath($r,dst) $dst
}##End for loop
```

```
##Mark the attack path segments and nodes
Puts “Trace results for $atkPath($r,algo) IP traceback”
set attksz [expr [array size atkPath]/2]
set gt 0
set n 1
foreach rec $recs {
set flds [split $rec " "]
lassign $flds evnt  tt tme ss src dd dst aa pkt ee pktsz oo bb ii idn
        if { $tme != "*"  && $tme != "" && $trcStrt < $tme } {
        incr gt
if { $gt == 1 && $n < $attksz } {
#puts "gt: $gt"
#puts "Attack nodes: [expr $attksz - $n], n is $n"
puts $dsv "n -t [expr $trcStrt+($tme-$trcStrt)/10] -s $atkPath([expr $attksz],src) -S COLOR -c red -o black
-i red -I black"
puts $dsv "n -t [expr $trcStrt+($tme-$trcStrt)/6] -s $atkPath([expr $attksz - $n],src) -S COLOR -c red -o
black -i red -I black"
puts $dsv "l -t [expr $trcStrt+($tme-$trcStrt)/4] -s $atkPath([expr $attksz - $n],src) -d $atkPath([expr
$attksz - $n],dst) -S DLABEL -l \"atk\" -L \"\""
puts $dsv "l -t [expr $trcStrt+($tme-$trcStrt)/2] -s $atkPath([expr $attksz - $n],src) -d $atkPath([expr
$attksz - $n],dst) -S COLOR -c yellow -o black"

set trcStrt [expr $trcStrt + ($n*0.1)]
incr n
}
}
puts $dsv $rec
#puts "Event Time: $tme and Trace Time: $trcStrt"
##Foreach loop ends
set gt 0
}
$ns at [$ns now] "endsimltn"
} else {
puts "Sorry, attack did not generate packets drops."
exit 0
}
##End of setSearch process
}

proc finish {} {
      global ns dt dsv
#       $ns flush-trace
#       close $dt
      close $dsv
          $ns at [$ns now] "setSearch"
}
```

```
proc endsimltn {} {
	global ns dt dsv
#dit dstv
	$ns flush-trace
	close $dt
	close $dsv
	    puts "running nam ..."
	exec nam dst0.nam &
	exit 0
}

 #$ns at 2.01 "setSearch"
 $ns at 2.0 "finish"
$ns run
```

Appendix A-IX: Data for SSOA-DoSTBK and ACS-IPTBK False Error Rate Measure

Generations	SSOA-DoSTBK			ACS-IPTBK		
	DoS	DoS & FE	Spoofed Dos & FE	DoS	DoS & FE	Spoofed Dos & FE
1	50.07	54.08	59.38	61.51	63.05	65.57
2	41.92	43.34	34.05	42.89	58.73	56.62
3	33.15	34.32	27.37	33.92	46.5	44.96
4	20.01	29.02	22.65	25.66	39.32	38.08
5	17.96	26.32	20.78	21.31	28.11	34.63
6	14.97	23.48	19.93	19.13	25.96	23.98
7	13.87	21.65	18.73	18.29	26.76	22.53
8	12.44	20.14	17.96	18.23	25.8	21.61
9	10.37	18.95	16.09	17.20	22.58	20.56
10	9.69	16.69	15.85	16.13	21.13	20.27
11	9.19	15.53	14.19	15.13	19.82	19.48
12	8.49	14.76	13.57	14.10	18.65	18.73
13	8.27	14.38	12.45	13.49	17.71	18.59
14	7.74	13.28	12.73	12.99	16.64	15.31
15	7.2	11.72	11.87	12.41	15.88	18.14
16	6.98	12.09	11.49	11.83	16.38	16.24
17	6.81	11.53	10.95	11.29	15.63	15.58
18	6.75	11.16	10.46	10.91	15.11	14.99
19	6.58	10.7	10.2	10.44	14.5	14.45
20	6.46	10.32	9.59	10.07	14	13.95
21	6.13	9.95	9.21	9.70	13.48	13.48
22	6.09	9.58	9.15	9.32	12.98	13.05
23	6.3	9.3	9.11	9.06	12.6	12.64
24	6.45	9.02	9.2	9.29	12.23	12.27
25	6.12	8.74	8.91	8.50	11.84	11.92
26	5.65	8.46	8.63	8.22	11.47	11.58
27	5.66	8.28	8.37	8.04	11.22	11.28
28	5.4	8	8.12	7.76	10.84	10.98
29	5.38	7.81	8.08	7.57	10.58	10.68
30	5.21	7.71	7.66	7.26	8.93	9.21
31	5.16	7.37	7.55	8.13	8.59	8.97
32	4.96	7.09	7.45	7.85	8.52	11.13
33	4.71	6.89	7.36	7.77	8.63	8.49
34	3.97	5.81	7.22	7.58	9.35	9.47
35	3.86	4.87	6.17	6.55	9.04	8.06
36	4.52	4.74	6.13	6.90	9.1	10.26
37	4.12	4.69	6.12	5.80	9.39	10.07
38	4.7	4.19	6.11	6.61	10.05	7.75
39	4.16	4.14	6.1	6.50	10.33	8.81
40	3.93	4.01	6.09	6.29	9.87	8.64

41	3.87	3.96	5.85	6.20	9.16	8.46
42	3.74	3.83	5.68	5.98	7.69	8.3
43	3.77	3.77	5.56	5.89	7.55	8.14
44	3.79	3.72	5.54	5.79	7.63	7.98
45	3.6	3.65	5.46	5.70	7.52	7.84
46	3.66	3.59	5.42	5.60	7.38	7.68
47	3.52	3.53	5.32	5.49	7.25	7.55
48	3.42	3.41	5.23	5.28	7	7.41
49	3.28	3.35	5.14	5.18	6.86	7.28
50	3.32	3.29	5.04	5.07	6.73	7.15
51	3.22	3.22	4.96	4.97	6.61	7.04
52	3.15	3.16	4.88	4.87	6.48	6.92
53	3.01	3.12	4.8	4.77	6.38	6.8
54	3.06	3.11	4.72	4.77	6.35	6.69
55	2.92	3.04	4.64	4.68	6.23	6.58
56	2.89	2.98	4.56	4.57	6.08	6.46
57	2.89	2.92	4.49	4.47	5.95	6.36
58	2.76	2.86	4.42	4.37	5.83	6.27
59	2.76	2.95	4.35	4.26	6.01	6.17
60	2.66	2.79	4.28	4.27	5.71	6.06
61	2.6	2.73	4.22	4.17	5.57	5.98
62	2.6	2.67	4.14	4.06	5.43	5.88
63	2.53	2.61	4.08	3.95	5.34	5.79
64	2.53	2.61	4.02	3.96	5.32	5.7
65	2.41	2.55	3.96	3.86	5.18	5.61
66	2.21	2.48	3.9	3.41	5.04	5.53
67	2.46	2.49	3.85	3.76	5.05	5.46
68	2.37	2.42	3.79	3.66	4.92	5.38
69	2.27	2.36	3.73	3.54	4.81	5.29
70	2.22	2.21	3.68	3.55	4.5	5.21
71	2.18	2.3	3.63	3.44	4.66	5.14
72	2.19	2.4	3.57	3.45	4.87	5.07
73	2.11	2.24	3.53	3.34	4.54	5.01
74	2.05	2.33	3.48	3.17	4.71	4.92
75	2.08	2.17	3.42	3.91	4.41	4.85
76	1.96	2.19	3.38	3.76	4.44	4.79
77	1.87	2.14	3.34	3.59	4.36	4.73
78	1.94	2.1	3.29	3.64	4.25	4.67
79	2.03	2.32	3.24	3.72	4.69	4.72
80	1.84	2.18	3.2	3.52	4.42	4.67
81	1.98	1.98	3.16	3.62	4.02	4.6
82	1.79	2.26	3.12	3.41	4.57	4.53
83	1.57	1.92	3.07	2.96	3.89	4.47
84	1.55	1.84	3.03	2.90	3.72	4.41
85	1.62	1.66	2.99	3.01	3.36	4.36

86	1.78	1.86	2.95	3.30	3.75	4.3
87	1.71	1.85	2.91	3.22	3.73	4.24
88	1.44	1.79	2.88	2.77	3.63	4.19
89	1.62	1.78	2.83	3.04	3.59	4.12
90	1.86	1.73	2.8	4.44	3.5	4.07
91	1.56	1.51	2.76	3.79	3.06	4.02
92	1.49	1.67	2.73	3.69	3.37	3.97
93	1.58	1.72	2.69	3.81	3.49	3.92
94	1.39	1.65	2.66	3.47	3.34	3.88
95	1.49	1.61	2.62	3.64	3.23	3.81
96	1.41	1.58	2.6	3.48	3.19	3.78
97	1.3	1.5	2.56	3.24	3.04	3.73
98	1.27	1.54	2.53	3.14	3.1	3.68
99	1.4	1.37	2.5	3.36	2.75	3.64
100	1.32	1.48	2.46	3.29	2.99	3.68

Appendix A-X: Data for SSOA-DoSTBK and ACS-IPTBK Convergence Measure

	SSOA-DoSTBK			ACS-IPTBK		
Generations	**DoS**	**DoS & FE**	**Spoofed Dos & FE**	**DoS**	**DoS & FE**	**Spoofed Dos & FE**
1	1257	1268	1279	1252	1227	1264
2	1258	1278	1293	1258	1263	1273
3	1264	1285	1299	1265	1280	1280
4	1270	1291	1306	1271	1290	1286
5	1275	1296	1310	1276	1297	1291
6	1281	1301	1316	1281	1302	1296
7	1285	1305	1320	1285	1306	1300
8	1290	1310	1325	1290	1309	1305
9	1298	1313	1328	1292	1312	1307
10	1300	1315	1330	1295	1314	1310
11	1306	1321	1336	1300	1316	1316
12	1307	1323	1338	1302	1317	1317
13	1312	1327	1342	1307	1319	1322
14	1314	1330	1345	1309	1320	1324
15	1317	1333	1348	1312	1321	1328
16	1320	1335	1350	1314	1323	1330
17	1322	1337	1352	1316	1324	1332
18	1322	1337	1352	1317	1324	1332
19	1322	1337	1352	1319	1325	1332
20	1323	1338	1353	1323	1326	1333
21	1325	1339	1354	1325	1327	1334
22	1326	1340	1354	1324	1327	1335
23	1328	1341	1354	1323	1328	1337
24	1329	1342	1355	1323	1329	1337
25	1329	1343	1356	1324	1329	1338
26	1330	1343	1357	1324	1330	1339
27	1330	1344	1357	1325	1330	1339
28	1331	1344	1357	1325	1331	1339
29	1331	1344	1357	1326	1331	1340
30	1331	1344	1357	1326	1331	1340
31	1332	1344	1357	1327	1332	1340
32	1332	1345	1357	1327	1332	1340
33	1333	1345	1357	1327	1333	1340
34	1333	1344	1356	1328	1333	1340
35	1333	1345	1356	1328	1333	1340
36	1334	1345	1356	1328	1334	1341
37	1334	1345	1356	1329	1334	1341
38	1334	1345	1355	1329	1334	1341
39	1334	1345	1356	1329	1334	1341

40	1335	1345	1356	1329	1335	1341
41	1335	1346	1356	1330	1335	1342
42	1335	1346	1356	1330	1335	1342
43	1335	1346	1357	1330	1335	1342
44	1336	1346	1357	1330	1336	1342
45	1336	1346	1357	1331	1336	1343
46	1336	1347	1357	1331	1336	1343
47	1336	1347	1357	1331	1336	1343
48	1337	1347	1358	1331	1337	1343
49	1337	1347	1358	1331	1337	1343
50	1337	1348	1358	1332	1337	1344
51	1337	1348	1358	1332	1337	1344
52	1337	1348	1358	1332	1337	1344
53	1338	1348	1359	1332	1338	1344
54	1338	1348	1359	1332	1338	1344
55	1338	1348	1359	1332	1338	1344
56	1338	1349	1359	1333	1338	1345
57	1338	1349	1359	1333	1338	1345
58	1338	1349	1359	1333	1338	1345
59	1338	1349	1360	1333	1338	1345
60	1339	1349	1360	1333	1339	1345
61	1339	1349	1360	1333	1339	1345
62	1339	1349	1360	1334	1339	1346
63	1339	1350	1360	1334	1339	1346
64	1339	1350	1360	1334	1339	1346
65	1339	1350	1360	1334	1339	1346
66	1339	1350	1361	1334	1339	1346
67	1340	1350	1361	1334	1340	1346
68	1340	1350	1361	1334	1340	1346
69	1340	1350	1361	1334	1340	1346
70	1340	1350	1361	1335	1340	1347
71	1340	1351	1361	1335	1340	1347
72	1340	1351	1361	1335	1340	1347
73	1340	1351	1361	1335	1340	1347
74	1340	1351	1362	1335	1340	1347
75	1340	1351	1362	1335	1340	1347
76	1341	1351	1362	1335	1341	1347
77	1341	1351	1362	1335	1341	1347
78	1341	1351	1362	1335	1341	1347
79	1341	1351	1362	1335	1341	1347
80	1341	1352	1362	1336	1341	1348
81	1341	1352	1362	1336	1341	1348
82	1341	1352	1362	1336	1341	1348
83	1341	1352	1362	1336	1341	1348
84	1341	1352	1362	1336	1341	1348

85	1341	1352	1363	1336	1341	1348
86	1341	1352	1363	1336	1341	1348
87	1342	1352	1363	1336	1342	1348
88	1342	1352	1363	1336	1342	1348
89	1342	1352	1363	1336	1342	1348
90	1342	1352	1363	1336	1342	1348
91	1342	1352	1363	1336	1342	1348
92	1342	1353	1363	1337	1342	1349
93	1342	1353	1363	1337	1342	1349
94	1342	1353	1363	1337	1342	1349
95	1342	1353	1363	1337	1342	1349
96	1342	1353	1363	1337	1342	1349
97	1342	1353	1364	1337	1342	1349
98	1342	1353	1364	1337	1342	1349
99	1342	1353	1364	1337	1342	1349
100	1343	1354	1365	1337	1343	1350

Appendix A-XI: Data for SSOA-DoSTBK and ACS-IPTBK Path Correctness Measure

	SSOA-DoSTBK			ACS-IPTBK		
Generations	**DoS**	**DoS & FE**	**Spoofed Dos & FE**	**DoS**	**DoS & FE**	**Spoofed Dos & FE**
1	14.59	14.03	13.19	16.28	12.72	12.66
2	20.21	19.43	18.26	19.43	17.23	17.16
3	25.18	24.21	22.76	24.21	21.51	21.85
4	30.51	29.06	27.32	29.06	25.85	25.95
5	35.48	33.79	32.44	33.79	30.37	30.82
6	41.23	38.9	37.34	38.9	34.95	35.1
7	45.86	43.68	41.93	43.68	39.63	39.83
8	50.46	48.52	46.58	48.52	44.02	43.79
9	56.03	53.36	50.16	53.36	46.88	47.65
10	60.47	58.14	54.65	58.14	52.22	52.46
11	63.27	59.69	57.31	59.69	53.64	55.02
12	63.84	60.8	58.37	60.8	55.77	56.04
13	66.06	62.91	60.39	62.91	56.43	57.97
14	67.16	63.96	60.12	63.96	57.36	57.72
15	67.75	64.52	61.29	64.52	57.37	58.84
16	68.92	66.27	62.96	66.27	58.93	59.81
17	70.69	67.32	63.96	67.32	59.87	60.12
18	70.21	66.87	62.86	66.87	59.30	60.35
19	71.15	68.41	65.68	68.41	62.75	62.4
20	71.6	68.85	64.72	68.85	61.84	60.84
21	72.96	69.49	65.32	69.49	60.46	62.05
22	72.63	68.52	65.1	68.52	61.61	62.5
23	70.22	66.56	63.89	68.56	61.04	61.33
24	72.02	68.59	64.47	68.59	59.67	60.6
25	72.73	68.61	65.18	68.61	61.49	62.57
26	73.12	69.64	66.16	69.64	62.52	62.19
27	74.2	70.67	66.43	70.67	62.08	63.77
28	74.56	71.69	68.82	71.69	65.13	65.38
29	75.09	71.51	68.65	71.51	64.77	65.9
30	75.62	71.34	68.48	71.34	65.43	64.37
31	74.71	71.15	68.31	71.15	64.56	64.21
32	73.6	70.77	66.53	70.77	62.27	63.2
33	73.98	69.79	65.6	70.53	62.68	61.66
34	72.25	68.81	66.06	68.81	62.52	62.76
35	72.96	68.83	66.07	68.53	61.74	62.77
36	72.28	68.84	64.71	67.84	61.24	62.12
37	70.2	66.86	63.85	66.96	58.17	60.34
38	70.58	67.87	64.48	67.87	59.68	60.61
39	71.65	68.89	65.44	68.89	62.43	62.17

40	72.7	69.9	65.71	69.9	63.38	62.42
41	73.06	68.92	66.16	68.92	61.83	62.85
42	72.38	68.28	64.87	68.28	61.30	62.28
43	70.66	67.94	64.54	67.94	60.41	61.31
44	71.71	68.95	64.19	68.95	62.44	62.88
45	71.09	68.36	64.95	68.36	61.28	62.35
46	72.06	67.98	63.9	67.98	60.39	61.34
47	71.22	67.19	64.5	67.19	62.22	61.28
48	70.35	67	64.32	67	59.53	60.46
49	69.73	66.41	63.75	66.41	60.92	59.93
50	69.98	66.02	62.06	66.02	58.73	58.96
51	69.2	65.9	61.95	65.9	58.63	58.85
52	68.29	65.04	61.13	65.04	57.13	57.46
53	67.25	64.05	61.48	64.05	56.91	58.41
54	68.95	65.05	62.45	65.05	59.01	59.95
55	67.69	65.09	62.49	65.09	58.49	58.74
56	68.71	66.07	63.43	66.07	59.84	59.62
57	70.13	66.79	64.12	66.79	60.68	61.56
58	70.44	67.09	63.73	67.09	59.65	59.91
59	70.81	68.09	64.69	68.09	61.03	60.81
60	71.86	69.1	64.96	69.1	62.07	61.71
61	72.91	70.11	65.9	70.11	61.59	63.26
62	73.59	70.76	67.93	70.76	64.20	64.53
63	72.09	68.66	65.92	68.66	63.59	61.96
64	71.46	68.71	64.59	68.71	60.45	61.36
65	73	68.87	64.74	68.87	60.60	62.15
66	72.62	69.16	65.7	69.16	62.09	62.42
67	73.37	69.22	65.07	69.22	61.49	62.47
68	73.48	69.32	65.85	69.32	63.52	63.22
69	73.62	69.45	66.67	69.45	62.41	64
70	71.99	69.22	65.07	69.22	60.23	62.47
71	72.77	69.3	65.14	69.3	60.88	61.23
72	72.74	69.28	66.51	69.28	62.16	63.18
73	71.51	68.76	65.32	68.76	61.62	61.4
74	71.99	69.22	65.07	69.22	60.80	61.82
75	71.6	68.85	65.41	68.85	61.70	61.49
76	71.65	68.89	66.13	68.89	62.50	62.82
77	71.42	68.67	64.55	68.67	60.89	61.32
78	71.55	68.8	66.05	68.8	62.42	63.41
79	71.7	68.94	65.49	68.94	61.78	61.56
80	72.25	68.81	65.37	68.81	61.09	61.45
81	71.58	68.83	66.08	68.83	62.44	62.78
82	71.32	68.58	64.47	68.58	61.60	60.6
83	71.31	68.57	65.83	68.57	63.50	62.54
84	69.53	66.86	62.85	66.86	60.05	59.08

85	69.27	66.61	63.28	66.61	59.70	60.12
86	70.6	66.6	62.6	66.6	59.24	60.1
87	69.96	67.27	64.58	67.27	59.77	61.35
88	69.96	67.27	64.58	67.27	62.29	61.35
89	71.32	67.28	63.24	67.28	60.33	59.45
90	69.97	67.28	63.24	67.28	59.76	60.71
91	70.64	67.28	63.25	67.28	59.20	59.46
92	70.65	67.29	64.6	67.29	60.95	61.37
93	69.98	67.29	63.93	67.29	60.99	60.73
94	70.67	67.3	63.26	67.3	59.78	60.1
95	70.99	68.26	64.85	68.26	60.02	61.61
96	72.37	68.27	64.17	68.27	61.22	61.6
97	71.68	68.27	64.17	68.27	61.32	61.6
98	71.68	68.27	64.86	68.27	60.03	61.62
99	74.48	70.26	66.75	70.26	63.17	64.08
100	74.49	70.27	66.75	70.27	62.38	64.08