

# Detection of Malicious Hosts Against Agents in Mobile Agent Networks

**Jean Tajer**

---

This Thesis is submitted in partial fulfilment of the requirements for  
the award of the degree of Doctor of Philosophy of the University of  
Portsmouth

---

**School of Computing  
University of Portsmouth  
Lion Terrace, Portsmouth, Hampshire  
PO1 3HE, United Kingdom**



August 2018

©Copyright 2018

Jean Tajer

© This copy of the thesis has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with the author and that no quotation from the thesis, nor any information derived therefrom, may be published without the author's prior, written consent.

# Abstract

---

Over the last decade, networks have become increasingly advanced in terms of size, complexity and the level of heterogeneity, due to increase of number of users, devices and implementation of cloud among big enterprises and developing smart cities. As networks become more complicated, the existing client-server paradigm suffers from problems such as delay, jitter, bad quality of service, insufficient scalability, availability and flexibility. The appearance of mobile agents' technology is getting popular as means for an efficient way to access remote resources on computer networks. Mobile Agent- systems usually benefit from the following: asynchronous execution, dynamic adaptation, fault-tolerance improvement in network latency, protocol encapsulation, reduction in network load and robustness.

However, one of the major technical obstacles to a wider acceptance of the mobile agent is security which is the *modus operandi* to protect the mobile agents against malicious hosts.

This work proposes how the Mobile Agents (MA), supported by a new solid models (detection and protection), can present a new way of securing mobile agents against malicious hosts.

The work contributes in proposing a new computing model for protection against malicious hosts. This model is based on *trust*, which is a combination of two kinds of trust: policy enforcement and control and punishment. The originality of this model is the introduction of the concept of setting up an active storage element in the agent space, called as "home away from home", for partial result storage and separation as well as digital signing of the destination of the mobile agent.

An efficient flooding detection scheme is developed by integrating the sketch technique with the Divergence Measures (Hellinger Distance, Chi-Square and Power Divergences). This type of integration can be considered *unique* in comparison with existing solutions over a Mobile Agent network. The sketch data- structure summarizes the mobile agent's process of calls generating into a fixed set of data for developing a probability model.

The Divergence Measures techniques, combined with a Mobile Agent traffic, efficiently identifies attacks, by monitoring the distance between current traffic distribution and the estimated distribution, based on history information. Compared to the previous detection system and existing works, the proposed techniques achieve the advantages of higher accuracy and flexibility, to deal with low intensity attacks and the ability to track the period of attack.

Simulation results are presented to demonstrate the performance of the proposed detection model. This work achieves in outperforming the existing detection solutions by tuning the Divergence Measures. An evaluation of the scheme is done via the receiver-operating characteristic (ROC). The work achieves in outperforming the existing detection solutions by tuning the Power Divergence with a value of  $\beta=2.2$ . With this value of  $\beta$ , the detection scheme leads to a very attractive performance in terms of True Positive Rate (100%), False Positive Rate (3.8%) and is capable of detecting low intensity attacks. Moreover, the Power Divergence with  $\beta=2.2$  presents a better detection accuracy of 98.1% in comparison with Hellinger Distance (60%) and Chi-square (80%).

Since the scenarios in consideration in this work can be reasonably related to any type of network, the strength of the proposed model can alternatively be applied to any enterprise network.

**Keywords:** mobile agents, malicious host, protection, detection, trusted nodes, divergence measures, sketch techniques, false alarm ratio, low intensity attacks.

# Acknowledgments

---

This work would not have been possible without the direct and indirect support of many people who deserve a lot of respect and gratitude. In all honesty, it is impossible to write all of their names here and I do sincerely apologize that I cannot acknowledge everyone by name.

First of all, I would like to thank my family and my wife for their support, especially over the last few years. I must, in all sincerity, acknowledge them; without their love and encouragement. I would not have finished this work. I cannot find enough words to thank them and whatever I say it will still not enough.

I would also like to express my gratitude and appreciation towards my supervisor, Dr. Mo Adda, and co-supervisor Dr. Benjamin Aziz for their overall support, patient guidance and invaluable advice,

for numerous discussions and encouragement throughout the course of the research. I appreciate their vast awareness and skills in the parallel systems and network area.

I would like to thank all my friends and my colleagues, at work who have been around encouraging me to successfully finish this thesis.

# Table of Contents

---

<b>ABSTRACT</b> .....	<b>III</b>
<b>ACKNOWLEDGMENTS</b> .....	<b>V</b>
<b>TABLE OF CONTENTS</b> .....	<b>IV</b>
<b>LIST OF FIGURES</b> .....	<b>VIII</b>
<b>LIST OF TABLES</b> .....	<b>X</b>
<b>ABBREVIATIONS</b> .....	<b>XI</b>
<b>CHAPTER 1 : INTRODUCTION</b> .....	<b>1</b>
1.1.    BACKGROUND .....	1
1.2.    STATEMENT OF THE PROBLEM .....	6
1.3.    OBJECTIVES, METHODOLOGY AND THESIS ORGANISATION .....	7
1.3.1.    Objectives .....	7
1.3.2.    Methodology.....	7
1.3.3.    Thesis Organisation .....	9
<b>CHAPTER 2 : MOBILE AGENT OVERVIEW</b> .....	<b>11</b>
2.1.    INTRODUCTION.....	11
2.2.    MOBILE AGENTS .....	12
2.2.1.    Network Paradigm .....	13
2.2.2.    Mobile Agent Migration.....	15
2.2.3.    Mobile Agent Life Cycle.....	16
2.2.4.    Examples of Mobile Agents.....	17
2.2.5.    Adopting the Mobile Agent .....	21
2.3.    AGENT PLATFORM OVERVIEW WITH AGLET .....	22
2.3.1.    Aglet System Architecture .....	23
2.3.2.    Aglet Model .....	24
2.3.3.    A tour of the Aglet API .....	25
2.3.4.    Security.....	26
2.4.    MALICIOUS HOSTS ON MOBILE AGENTS AND THEIR COUNTERMEASURES .....	27
2.4.1.    Malicious Hosts.....	27
2.4.2.    Countermeasures for Security Services .....	28
2.5.    SUMMARY.....	29
<b>CHAPTER 3 : LITERATURE REVIEW</b> .....	<b>30</b>
3.1.    INTRODUCTION.....	30
3.2.    EXISTING PROTECTION TECHNIQUES AND THEIR DRAWBACKS.....	30

3.2.1.	<i>Distributed Technique for Multi-Agents</i> .....	30
3.2.2.	<i>Environmental Key Generation Technique</i> .....	31
3.2.3.	<i>Secure Software technique for MA Execution</i> .....	31
3.2.4.	<i>DDOS Protection Technique</i> .....	32
3.2.5.	<i>Multi-Facet Security Approach</i> .....	33
3.2.6.	<i>Obfuscation technique</i> .....	33
3.3.	EXISTING DETECTION TECHNIQUES AND THEIR DRAWBACKS .....	35
3.3.1.	<i>Watermarking and Fingerprinting Detection Techniques</i> .....	35
3.3.2.	<i>Recording and Tracking Technique</i> .....	35
3.3.3.	<i>Itinerary Recording with Replication and Voting Technique</i> .....	36
3.3.4.	<i>State Appraisal Technique</i> .....	36
3.3.5.	<i>Enhanced Reference Monitor based Security Framework</i> .....	37
3.3.6.	<i>Trusted Hardware technique</i> .....	37
3.3.7.	<i>Intrusion Detection Techniques</i> .....	38
3.3.7.1.	Multi-agent-based IDS .....	38
3.3.7.2.	MAIDS Technique .....	38
3.3.7.3.	Distributed IDS Technique .....	39
3.3.7.4.	Intrusion Detection Processor (IDP) & Sensors Technique .....	39
3.3.7.5.	Simple System for Multi-Agents for IDS .....	39
3.3.7.6.	Multilayer Detection Technique .....	40
3.4.	CURRENT DIVERGENCE MEASURES AND SKETCH TECHNIQUE .....	41
3.5.	MOTIVATION FOR IMPROVING THE PROTECTION OF MOBILE AGENTS .....	42
3.6.	SUMMARY.....	45
<b>CHAPTER 4 : DETECTION PROPOSED MODEL .....</b>		<b>46</b>
4.1.	INTRODUCTION .....	46
4.2.	THEORETICAL BACKGROUND .....	47
4.2.1.	<i>Sketch Techniques</i> .....	47
4.2.2.	<i>Threshold</i> .....	49
4.2.3.	<i>Mathematical Algorithms</i> .....	50
4.2.3.1.	Chi-Square.....	50
4.2.3.2.	Hellinger Distance .....	51
4.2.3.3.	Power Divergence .....	51
4.3.	PROPOSED APPROACH .....	53
4.4.	CONCLUSION .....	56
<b>CHAPTER 5 : PROTECTION PROPOSED MODEL .....</b>		<b>57</b>
5.1.	INTRODUCTION .....	57
5.2.	DESIGN GUIDELINES.....	58
5.3.	COMPONENTS OF THE PROPOSED COUNTERMEASURE MODEL .....	58
5.3.1.	<i>Home of the Mobile Agent (Home)</i> .....	59

5.3.2.	<i>Mobile Agent</i> .....	59
5.3.3.	<i>Trusted Node</i> .....	59
5.3.4.	<i>Active Storage Element</i> .....	59
5.3.5.	<i>Host</i> .....	59
5.4.	PROPOSED COUNTERMEASURE MODEL .....	60
5.5.	SECURITY MODEL .....	62
5.5.1.	<i>At Home</i> .....	63
5.5.2.	<i>At Trusted Server</i> .....	64
5.5.3.	<i>At ith Host</i> .....	64
5.5.4.	<i>At Trusted Server</i> .....	65
5.5.5.	<i>At Home</i> .....	66
5.5.6.	<i>Security Model Summary for N Hosts</i> .....	67
	EVALUATION OF THE APPROACH .....	68
5.6.	CONCLUSION .....	69
<b>CHAPTER 6 : EXPERIMENTAL WORK.....</b>		<b>70</b>
6.1.	INTRODUCTION .....	70
6.2.	PROTECTION MODEL RESULTS .....	70
6.2.1.	<i>Developed Classes</i> .....	70
6.2.1.1.	<i>Destn</i> .....	70
6.2.1.2.	<i>ProxyH</i> .....	71
6.2.1.3.	<i>CipherCls</i> .....	72
6.2.2.	<i>Components of the Prototype Application</i> .....	72
6.2.2.1.	<i>statA</i> .....	72
6.2.2.2.	<i>MobileA</i> .....	75
6.2.2.3.	<i>AgentApp</i> .....	80
6.2.2.4.	<i>pathDIg</i> .....	80
6.2.2.5.	<i>MobileUIF</i> .....	81
6.2.2.6.	<i>infoDIg</i> .....	81
6.2.3.	<i>Setting up the laboratory</i> .....	85
6.2.4.	<i>Evaluation Strategy</i> .....	86
6.2.4.1.	<i>Eavesdropping</i> .....	86
6.2.4.2.	<i>Alteration</i> .....	87
6.2.4.3.	<i>Performance Comparison</i> :.....	87
6.3.	DETECTION MODEL TEST RESULTS .....	89
6.3.1.	<i>Experiments Set Up</i> .....	89
6.3.2.	<i>Evaluation Strategy</i> .....	90
6.3.2.1.	<i>Detection Techniques</i> .....	90
6.3.2.2.	<i>Injection of SYN Flooding Attacks</i> .....	92
6.3.2.3.	<i>Comparison between HD &amp; Chi-square</i> .....	93
6.3.2.4.	<i>Power Divergence</i> .....	95
6.4.	CONCLUSION .....	100



<b>CHAPTER 7 : CONCLUSION AND FURTHER WORK .....</b>	<b>103</b>
7.1.    CONCLUSION .....	103
7.2.    FUTURE WORK .....	106
<b>BIBLIOGRAPHY .....</b>	<b>107</b>
<b>APPENDIX A – PUBLICATIONS.....</b>	<b>113</b>
<b>APPENDIX B – EXPERIMENT ENVIRONMENT.....</b>	<b>114</b>
<b>APPENDIX C - AGLETS INSTALLATION AND CONFIGURATION INSTRUCTIONS FOR WINDOWS .....</b>	<b>117</b>
<b>APPENDIX D – INSTALLATION PROVIDERS IN AGLET .....</b>	<b>120</b>
<b>APPENDIX E – AGLET CODE .....</b>	<b>121</b>
<b>APPENDIX F – SKETCH TECHNIQUE AND DETECTION ALGORITHM CODES .....</b>	<b>132</b>
<b>APPENDIX G – UPR16 FORM.....</b>	<b>150</b>

# List of Figures

---

FIGURE 1-1 APPROACH TO SOLVE THE PROBLEM OF MA.....	9
FIGURE 2-1: CLIENT-SERVER COMPUTING MODEL .....	13
FIGURE 2-2: REMOTE EVALUATION.....	14
FIGURE 2-3 :MOBILE AGENT COMPUTING MODEL .....	15
FIGURE 2-4: MOBILE AGENT LIFE CYCLE .....	17
FIGURE 2-5: TAHITI GRAPHIC USER INTERFACE (GUI) .....	23
FIGURE 2-6: AGLET ARCHITECTURE .....	23
FIGURE 4-1 : SKETCH DATA STRUCTURE.....	48
FIGURE 4-2 : PROPOSED ALGORITHM FOR NETWORK ANOMALY DETECTION. ....	53
FIGURE 5-1: EXISTING MOBILE AGENT COMPUTING MODEL .....	60
FIGURE 5-2 : PROPOSED NEW MODEL .....	61
FIGURE 5-3: PROPOSED MODEL AT HOME .....	63
FIGURE 5-4: PROPOSED MODEL AT TRUSTED SERVER.....	64
FIGURE 5-5: SECURITY MODEL AT THE ITH HOST .....	65
FIGURE 5-6: PROPOSED MODEL AT TRUSTED SERVER, AFTER THE REACH OF THE NTH HOST .....	66
FIGURE 5-7: SECURITY MODEL BACK AT HOME.....	67
FIGURE 6-1: FUNCTIONS OF THE DESTN CLASS.....	71
FIGURE 6-2: FUNCTIONS OF THE PROXYH.....	71
FIGURE 6-3: FUNCTIONS OF THE CIPHERCIS .....	72
FIGURE 6-4: CLASS STATA .....	73
FIGURE 6-5: CLASS PROXYH.....	73
FIGURE 6-6 CODE: HANDLEMESSAGE (MESSAGE MSG) .....	74
FIGURE 6-7 CLASS: MOBILEA .....	75
FIGURE 6-8 CLASS: MOBILITYLISTENER.....	75
FIGURE 6-9 CLASS: MOBILITYLISTENER.....	76
FIGURE 6-10 CODE: ONARRIVAL(MOBILITYEVENT .....	77
FIGURE 6-11 CODE: IF ELSE FOR THE DSTATHOME.....	79
FIGURE 6-12 CODE: TOURING THE DIFFERENT HOSTS .....	79
FIGURE 6-13: PATH DIG BOX .....	81
FIGURE 6-14: DIAL BOX .....	81
FIGURE 6-15 CLASS: AGENTAPP .....	82
FIGURE 6-16 CODE: MAIN VARIABLES FOR AGENTAPP CLASS.....	82
FIGURE 6-17 CODE: ONCREATION METHOD .....	83
FIGURE 6-18 CODE: BEHAVIOUR OF THE APPLICATION .....	84
FIGURE 6-19 : TEST ENVIRONMENT SET UP.....	85
FIGURE 6-20: CAPTURED PACKET ANALYSIS FOR DS MA AND NORMAL MA .....	86

FIGURE 6-21: CAPTURED PACKET ANALYSIS FOR PROPOSED MA .....	86
FIGURE 6-22: DETECTION OF MALICIOUS HOST .....	87
FIGURE 6-23: PERFORMANCE COMPARISON BETWEEN DIFFERENT TYPES OF MOBILE AGENTS .....	88
FIGURE 6-24: PERFORMANCE TIME TREND AS THE NUMBER OF NODES VISITED INCREASES .....	89
FIGURE 6-25: SYN FLOODING ATTACKS .....	90
FIGURE 6-26 : TOTAL NUMBER OF MOBILE AGENTS' PACKETS .....	93
FIGURE 6-27 TOTAL NUMBER OF MOBILE AGENTS' PACKETS AFTER SYN FLOODING ATTACKS INJECTION..	93
FIGURE 6-28: HELLINGER DISTANCE BEFORE ATTACKS .....	94
FIGURE 6-29 : CHI-SQUARE BEFORE ATTACKS .....	94
FIGURE 6-30: HELLINGER DISTANCE AFTER ATTACKS.....	95
FIGURE 6-31: CHI-SQUARE AFTER ATTACKS .....	95
FIGURE 6-32: POWER DIVERGENCE FOR $\beta=0.5$ .....	96
FIGURE 6-33: POWER DIVERGENCE FOR $\beta= 1.5$ .....	96
FIGURE 6-34: POWER DIVERGENCE FOR $\beta= 2$ .....	97
FIGURE 6-35: POWER DIVERGENCE FOR $\beta= 2.2$ .....	97
FIGURE 6-36: RECEIVER OPERATING CHARACTERISTIC FOR $x^2$ , HD AND PD .....	99
FIGURE B-1 TEST ENVIRONMENT SET UP .....	115
FIGURE B-2 WIRESHARK .....	115
FIGURE B-3 NETWORK PACKET GENERATOR .....	116

# List of Tables

---

TABLE 3-1 COMPARATIVE ANALYSIS OF SECURITY SOLUTIONS TO PROTECT MOBILE AGENTS.....	40
TABLE 3-2 DETECTION AND PROTECTION DRAWBACKS TECHNIQUES .....	44
TABLE 6-1 THRESHOLD PARAMETERS .....	92
TABLE 6-2: FALSE POSITIVE ALARM FOR DIFFERENT VALUES OF $b$ .....	97
TABLE 6-3: FLOODING RATE RESULT.....	98
TABLE 6-4 : ROC BASED ON DIVERGENCE MEASURES .....	99
TABLE 6-5: CONFUSION TABLE FOR PD OF $B=2.2$ .....	100

# Abbreviations

---

ABA	Agent Based Application
API	Application Programming Interface
ASE	Active Storage Element
ATP	Agent Transfer Protocol
BC	Bouncy Castle
COD	Code on Demand
CS	Client-Server
CUSUM	Cumulative Sum
DSMA	Digitally Signed Mobile Agent
DOS	Denial of Service Attack
FN	False Negative
FP	False Positive
GUI	Graphic User Interface
HD	Hellinger Distance
HN	Home Node
IETF	Internet Engineering Task Force
IP	Internet Protocol group
IDS	Intrusion Detection System
JCA	Java Cryptography Architecture
JVM	Java Virtual Machine
LAN	Local Area Network
MA	Mobile Agent
MAS	Multi-Agents System
MSE	Mean Square Error
NM	Network Management
OSI	Open System Interconnection

PMA	Proposed Mobile Agent
PD	Power Divergence
ROC	Receiving Operating Characteristic
TCP	Transmission Control Protocol
TN	True Negative
TS	Trusted Server
TP	True Positive
$\chi^2$	Chi Square

**DECLARATION:**

Whilst registered as a candidate for the above degree, I have not been registered for any other research award. The results and conclusions embodied in this thesis are the work of the name candidate and have not been submitted for any other academic award.



# Chapter 1 : Introduction

---

## 1.1. Background

Next generation enterprises will be definitely characterized by new organizational structures, such as increased reliance on virtual operations and “electronically glued” organizations. Furthermore, it is also clear that the next-generation enterprises will have an increasing percentage of personnel, be they employees, suppliers or customers, who would use, a wide range of wireless technology, devices and the implementation of cloud, smart cities and internet of things. One of the key facilities personnel in next generation enterprises will require is the ability to efficiently retrieve, organize, manage and leverage information, do the job anywhere, any time; and at last saving time compared to their real world counterparts and knowledge from widely dispersed sources within, as well as from outside, viz. the virtual organization. This activity represents a significant challenge as it tends to require a great deal of human involvement, that increases cost and delay associated with it. Nevertheless, it is unfortunate that all of these services are based on the traditional client server approach.

Thus, any tool, technology, or alternative distributed paradigm that can help automate this activity has a potential for a significant payoff. The distributed paradigm is modeled much like what top business-persons, famous coaches and sporting personalities are used to. As a single person could not manage or handle many tasks at a time, he would delegate part of his authority to his agent, who could autonomously make a decision on his behalf.

There has been tremendous interest in the past few years in using mobile agent technology for next-generation enterprises. In particular, mobile agents seem have been proposed for automating the task of retrieving, organizing and filtering information located at widely dispersed sites. Mobile agent systems have been built and demonstrated, that indicate mobile agents can be used for information retrieval activities.



Any of the following questions can be considered:

Can the mobile agent be “better” than using traditional client-server computing for information retrieval tasks? Yes, the answer is emphatic affirmative. The questions that immediately arise are: ‘what is a mobile agent? Is not the mobile agent paradigm an old tool? Can mobile agent reduce the network load and thus increase the responsiveness and speed of the network?’

Due of increase in demand of applications and the number of users who wish to use wireless and portable devices increases significantly, it was realized that the traditional client-server approach was lacking in terms of bandwidth use and server flexibility, load balancing, availability and delay in traffic network.

In lieu of this, a new paradigm is needed and certainly the most promising among the new paradigms is the use of mobile agents.

A Mobile Agent (MA) is a software entity which basically exists in a software environment (Liotta, 2001, Pleisch and Schiper, 2004a, Richard and Lipperts, 2002). Until now, the research community has not agreed on a common definition for MAs. This has raised controversial arguments for nearly a decade and only recently have features been identified (Liotta, 2001) such as migration, cloning and autonomy. Various sets of required capacities for MAs have been devised, but none of these sets has been commonly accepted as a standard definition of an MA (Pleisch and Schiper, 2004). In general, the agents used focus on software systems and consequently, the term agent will be defined and used for software systems only, i.e. each agent is a software agent and thus an independent program. Subsequently, a MA is an agent that is able to move between different machines or, in a more abstract sense, between various individual network locations. An MA system is a software artefact, specified in a language that provides sufficient expressiveness and flexibility to allow the construction of multiple interacting MAs. Thorough background information, generally about software agents, in particular about mobile agent systems is presented in the succeeding chapter.

However, despite how fascinating the idea of mobile agent computing is, it is not without a glitch when it comes to the implementations and real world deployment. Mobile agents could be subjected to attacks launched by hostile hosts while visiting different hosts in the net. Sander and Tschudin raised two types of security problems that need to be solved. The first is host protection against malicious agents (Sander and Tschudin, 1998). The second is agent protection against malicious hosts. Many techniques have been developed for the first problem, such as access control, password protections, sand boxes etc.

But the second problem appears to be difficult to solve. The fact is that computers have complete control over all the executing programs. As a consequence, it becomes very difficult to protect mobile agents from malicious hosts.

Several studies have been done to provide a way for protecting malicious hosts on mobile agent networks. These solutions present many drawbacks like reducing the efficiency of the network, being unable to protect against denial of service. Furthermore, researchers are focusing their studies on detection of attacks by including an intrusion detection system. IDSs are hardware and software systems that monitor events occurred on computers and computer networks in order to analyse security problems. IDS and firewalls have become key components in ensuring the safety of network systems. Intrusions and invasions inside computer networks are called as “attacks” and these attacks threaten the security of networks by violating privacy, integrity and accessibility mechanisms. Attacks can be originated from users, who login to the computer using Internet, trying to gain administrator rights and other users who misuse the rights they have. IDSs automate monitoring and analysing the attacks. IDS can recognize the patterns of typical attacks, analyze the abnormal activity pattern and track the user’s policy violation. So it often called as the “last line of defense”.

A new semi-supervised learning method is introduced by Chein, Wen and Lee for false alarm reduction using only a very small amount of labelled information. This made the alarm filter more practical for the real systems. Numerical comparison with the conventional supervised learning approach with the same small portion of the labelled data has been done. Analysis shows this method as having the significantly superior detection rate as well as in the false alarm detection rate (Chein, Wen & Lee, 2010).

There are four basic techniques that are used to detect intruders: Anomaly detection, Misuse detection (signature detection), Target monitoring and Stealth Probes. The security of a local networking environment, or a single computer system, is a very important aspect of its infrastructure. A properly secured system is able to guarantee confidentiality, integrity, and availability of its resources and services. Thus, the ability to protect a system from outside interference is of most prime importance. However, flaws in computer systems can be specifically attacked by individuals. These attacks result in rendering the system vulnerable, compromising its entire security scheme. A significant aspect of the general security scheme is to detect if and when an attack against protected resources is attempted. The topic of network security that deals with this field is IDS. The Intrusion Detection Technology, employed in order to monitor resources distributed among several nodes in a local network, adopts a distributed approach to its design.

However, this way of detection encounters many problems, like the inability to handle large amounts of network traffic; centralized structure of IDS can be harmed in the case of high speed network, increase of false alarm ratio, incapability of detection of low intensity attacks. Lots of work have been done in detection of intruders. But the solutions are far from satisfactory.

In this respect, the thesis will concentrate on and provide a new detection model for:

- Reduction of high false alarm ratio.
- Improve the detection of low intensity attacks in mobile agents' network.

In fact, a natural idea for flooding detection is to identify changes in traffic volume or rate as mentioned by Chen (2006). In such schemes, alarms are raised if the traffic volume during a time interval is larger than a threshold, predicted according to past normal conditions. A major issue of volume/rate monitoring is that the detection accuracy can be severely degraded if the normal rate is dynamic in the observation window, due to the random nature and the flooding attack rate being not very high. The use of Divergence Measures (Hellinger, Chi-Square and Power Divergence), which describes the deviation between two probability distributions, have been proposed as a detection method. These divergence measures schemes have shown their strong capability to detect flooding attack, because the low-rate flooding is likely to have different probability distributions from the

normal traffic. Moreover, studies (Sengar and Wijesekera, 2008) do not address how to maintain an accurate threshold reflecting the normal condition under attacks, which is critically important to the divergence measure based detection system.

The work distinguishes among other works in terms of integrating the sketch technique with divergence measures, in order to achieve a more efficient and flexible flooding detection scheme. The sketch is a technique for random data aggregation, which can summarize the traffic associated with one or more physical attributes into a pre-determined number of states by the hash operation. In the proposed scheme, the probability model is based on the sketch data, rather than directly on the physical attributes.

In summary, the work has contributed in securing the mobile agent inside their network using the following four aspects:

- 1) Apply sketch to establish a behaviour probability model, which enables our detection scheme to efficiently deal with flooding attacks; the sketch data structure is a probabilistic data summary technique. It randomly aggregates high dimensional data streams into smaller dimensions.
- 2) Perform a high accuracy of detection of attacks while limiting the false alarm ratio as maximum. This is done by tuning the parameter of Divergence Measures to optimize the performance.
- 3) Capability of the detection model to identify low intensity attacks; the experiments will be repeated several times on Divergence measures by changing the attack rates accordingly.
- 4) Providing a new model for protecting mobile agents. This will be achieved by providing a new computing model based on trust for the protection against malicious attacks over mobile agents -network. A combination of two kinds of trust will be applied: policy enforcement and control and punishment.

## 1.2. Statement of the Problem

Mobile agent systems need an open environment to operate and perform their job.

More specifically, mobile agents and mobile agent platforms are subjected to attacks from malicious hosts and malicious mobile agents respectively. Hence, the major technical obstacle, to a wider acceptance of the mobile agent paradigm is security.

There are two types of security problems that must be addressed:

- Mobile Agent platform protection
- Mobile Agent protection

Many techniques have been developed for the first kind of problem but it is believed that the execution environment (host) has full control over executing programs; hence, protecting a mobile agent from malicious hosts is difficult to be achieved.

The work emphasises on the Mobile Agent protection in relation to data collecting agents, which can be divided in to two parts:

- To protect against malicious host from attacking incoming mobile agent is difficult. To work around this problem, the use of closed system or only trusted servers are needed.
- To detect an attack against mobile agent with the capability of reducing the false alarm and detecting low intensity attacks. The following techniques have been used: sketch, divergence measure and dynamic threshold.

Evolving from this, the research problem / thesis statement can be described as given below:

How can the new proposed protection model, outperforming the existing models in terms of reducing the false positive ratio and capable of detecting low intensity attacks, be used to secure the functionality of Mobile Agents inside its network against malicious host?

## 1.3. Objectives, Methodology and Thesis Organisation

### 1.3.1. Objectives

The objectives of this thesis are to:

- Find a new solution (for the protection and detection), different from the existing ones, to secure the trip of Mobile Agents inside its network.
- Study the mobile agent systems in general and analyse the existing countermeasures on malicious hosts' threats.
- Propose a new computing model for protecting malicious attacks over mobile agents' world, using trusted server which is a combination of two kinds of trust: policy enforcement and control and punishment.
- Propose a new model for detecting any threats over mobile agents. This model has two main objectives: limitation of false alarm ratio and the capability of detection of low intensity attacks in mobile agents' network.

The power of this new model is the combination of several techniques: sketch technique, Divergence Measures (Hellinger Distance, Chi-Square and Power Divergence) and use of a dynamic threshold to differentiate network anomalies from normal behaviour on the mobile agent.

### 1.3.2. Methodology

Different methodologies are used for the various phases of the thesis work.

The first phase involves the study of the issues and areas closely related to the thesis work, in order to get a deeper understanding of the problem. This is accomplished mainly through a literature review, viz. reading journal papers, articles, books and supplementary reading materials.

Major activities performed in this phase are:

- Studying about software agents in general, their characteristics and mobile agent systems.
- Studying the various types of distributed computing paradigms which include client server, remote evaluation, code on demand and mobile computing.
- Studying threats posed on mobile computation, specifically on mobile agents. Various propositions available far are studied and analysed as well, which helped us to come up with our own perception of the problem.

Overall, the major tasks performed during this phase were focused on gaining all the necessary background information that is needed to understand the broader picture of the problem as well as its surroundings.

The last phase of the work deals with the development and realization of models that will address some of the threats posed to mobile agents. The major activities carried out include:

- Development of models (detection and protection) to mitigate some of the problems posed on mobile agents, specifically to that of data collecting agents.
- Evaluation and selection of different mobile agent platforms that would be used to realize the concept.
- Further refining the concept developed.
- Collection of data by implementing a mobile agent network.
- Validation and Evaluation of the work:
  - 1- Tuning the parameter of Divergence Measures to optimize the performance. A comparison between different divergence measures will be done to choose the best detection with low false alarm ratio. A dynamic threshold will be used to differentiate network anomalies from normal behaviour on the mobile agent.
  - 2- Conducting performance analysis over real IP traces publicly available, in Mobile Agent Network, integrated with flooding attacks.
  - 3- Testing the detection of low intensity attacks over different divergence measures with different flooding rate.

- 4- Evaluating the performance of the proposed divergence measures via the False positive alarm, false negative alarm, receiver operating characteristic (ROC), and confusion matrix.

The diagram given below presents the approach to solve the security issue of mobile agents.

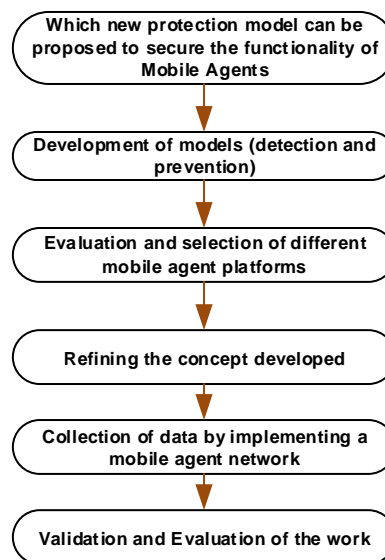


Figure 1-1 Approach to solve the problem of MA

### 1.3.3. Thesis Organisation

The main chapters in this thesis are organised as follows:

- Chapter 2 gives an overview of Mobile Agent systems (MAs) and also has addressed the main features of the MAs such as mobility, portability, communication, etc. It also shows that the MAs offer several advantages for distribution management over the traditional solutions, for instance; communication latency and bandwidth, off-line processing, reaction time, asynchronous behaviour by request aggregation and software-distribution on demand, which put MAs in better position to replace or to coexist with the traditional technology in network management to cope with the up-to-date users needs.
- Chapter 3 discusses the background and related work in the context of computer networking and system management and has addressed the main issues in the area of such threats over mobile agents.



- 
- Chapter 4 explains the proposed model for detecting attacks over mobile agent network. It will combine sketch technique and divergence measure (Hellinger Distance, Chi square and Power Divergence) in order to find the best way for the detection.
  - Chapter 5 presents a new computing model for protection of malicious mobile agent in a mobile agent world. It will show the components of the proposed countermeasure model. indeed, it will propose a security model that defines how components of the system should interact with each other as well as what necessary tasks need to be performed at each level in order to secure the network from malicious attack.
  - Chapter 6 shows tests and verifies the strength and performance of the proposed new approaches for the detection and protection of attacks over mobile agent network.
  - Chapter 7 concludes the work in this thesis and offers suggestions for future work are outlined as well.



## Chapter 2 : Mobile Agent Overview

---

### 2.1. Introduction

In the previous chapter, the concepts of protection of mobile agents have been discussed. It has been shown that the CS paradigm is the most popular paradigm for current networks; however, with the rising demands on processing power and the need to conserve bandwidth on large and slow networks, the limitations of this approach are easily perceptible. The concept of distributed CS computing can be too restrictive for certain applications (Pleisch and Schiper, 2004a). For instance, the functionality of a server is defined by its computing interface and generally cannot be adapted to the client's needs without reconfiguring that interface. Undoubtedly, this limits the flexibility with which a client can use a server and forces the client to process the server data locally according to the client's requests. Consequently, the mobile code-computing paradigm was introduced. With this paradigm, not only, data but also the code to act on the data, are transmitted between the client and the server. Transmitting the code makes the applications more flexible and actively allows the client to modify the functionality provided by the server to its fundamental needs. When using Mobile Agents (MAs), it is possible to bring the code close to the resources, which is not possible with the CS paradigm.

This chapter starts with a short overview of Mobile Agents. A definition of a Mobile Agent is given in Section 3.2. An introduction of a MA platform, which it is adopted in this research, has been given in Section 3.3. Malicious Hosts on Mobile Agents and their Countermeasures are discussed in Section 3.4.

## 2.2. Mobile Agents

According to the Oxford English Dictionary, an agent is “one who (or that which) acts or exerts power, as distinguished from the patient, and also from the instrument. For example, James Bond (007) is a fictional British agent created by the novelist Ian Fleming in 1952 (Siemens, 2002). Therefore, an agent is a person or an object that has the power to thoroughly act on others' behalf with official authority. In the case of Mobile Agents, there are several definitions by different authors. A Mobile Agent is defined as an agent capable of moving between machines (Pham and Karmouch, 1998). It was also described as an agent that has the unique ability of transporting itself from one network to the other (Lange and Oshima, 2003). *In a nutshell, a Mobile Agent is a form of software agent, capable of travelling across a network.* For a comprehensive overview about MAs, the reader can be referred to (Graesser,2006).

To be precise, Mobile Agents are Software agents that automate tasks. What distinguishes software agents from other utility software programs are both the ability to perform in distributed computing environments and to supply some domain knowledge to automating tasks for users' (Watson, 1997).

Furthermore, the state of a mobile agent consists of the attributes and values of the mobile agent that assist in the executions of its functions and the code is, for example, the Java classes or compiled/uncompiled C code.

The mobile agent needs to maintain the same state and code, while travelling down a network, for unique identification and for execution of its functions at any point on the network. It is only capable of executing its functions or interacting with an object across a network with the same host.

### 2.2.1. Network Paradigm

Mobile agent computing is just one of the four network computing paradigms, used to develop distributed applications. Each one paradigm has its own mechanism of performing the computation as well as area of applicability. These paradigms are:

- Client-Server
- Remote Evaluation
- Mobile Agent

#### Client - Server

This paradigm is the first one for implementing distributed applications, using remote procedure call approach. It utilizes centralized servers and distributed clients. The servers publish methods to be used by clients to access data and services it offers. Whenever the client wants to get some kind of service from the server, it calls the right published interface with appropriate arguments.

The server after processing the call returns the result to the client. Figure 2-1 shows a typical client-server environment, e.g. HTTP and FTP.

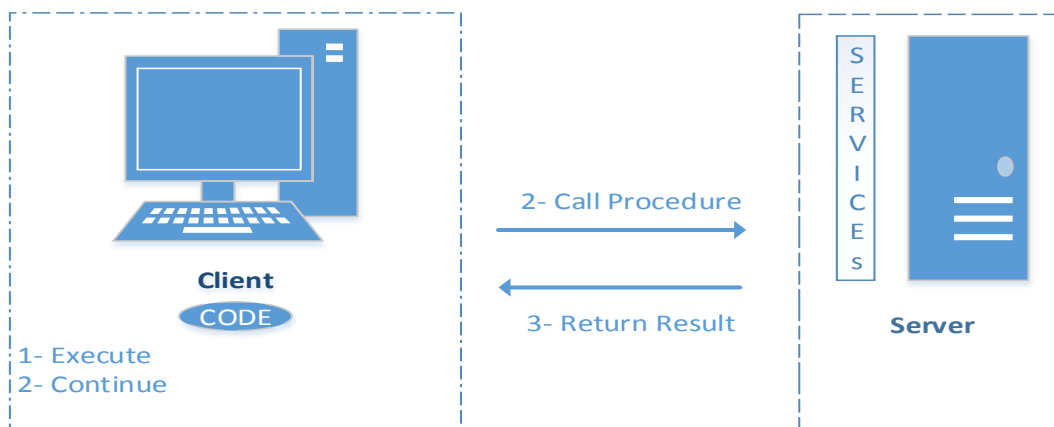


Figure 2-1: Client-server computing model

### Remote Evaluation

Unlike the client-server paradigm, this paradigm not only transfers data between the client and the server, but also the code. Once this code is transferred to the server, it will be executed there and only the result will be returned to the client. In contrast to the client – server paradigm, only one connection to the server has to be made to transfer the code to the server instead of several, as in client – server.

Figure 2-2 shows the typical remote evaluation environment, e.g. rsh command in Unix.

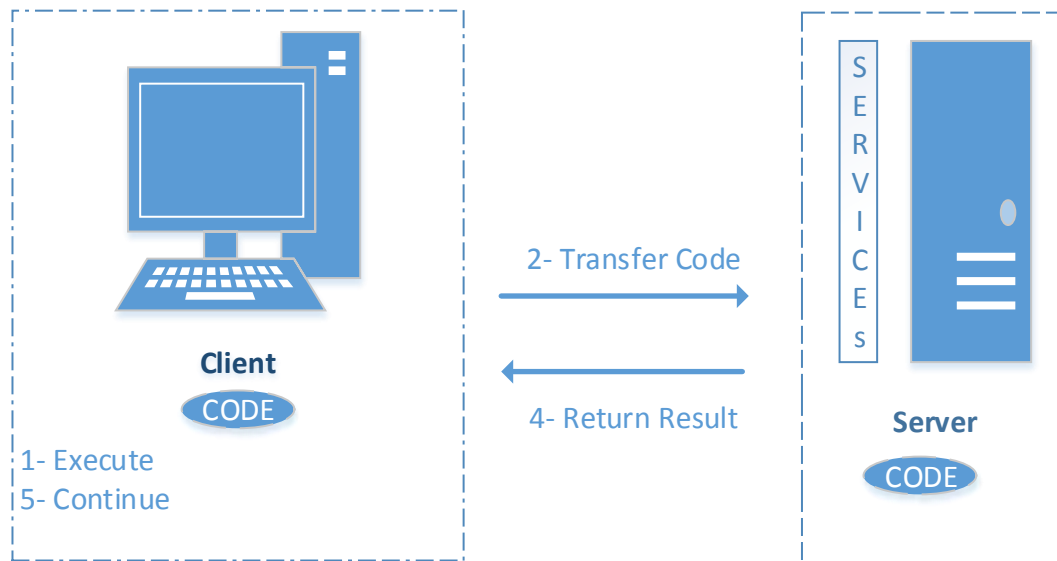


Figure 2-2: Remote Evaluation

### Mobile Agent

In this paradigm, the code (agent) moves from place to place to perform computation on behalf of its possessor. It is somewhat similar to remote evaluation, except that the code has autonomy and could visit multiple nodes before returning to its home.

Figure 2-3, given below shows typical mobile agent computing environment.

An MA could migrate between nodes, carrying the necessary code to be executed (Nikaein, 1999). MAs offer a real benefit to the system developers and the users of the applications based on MA, specifying which parameters can be improved upon. The importance of the key parameters such as performance, connectivity, reliability and modularity varies from one application to another.

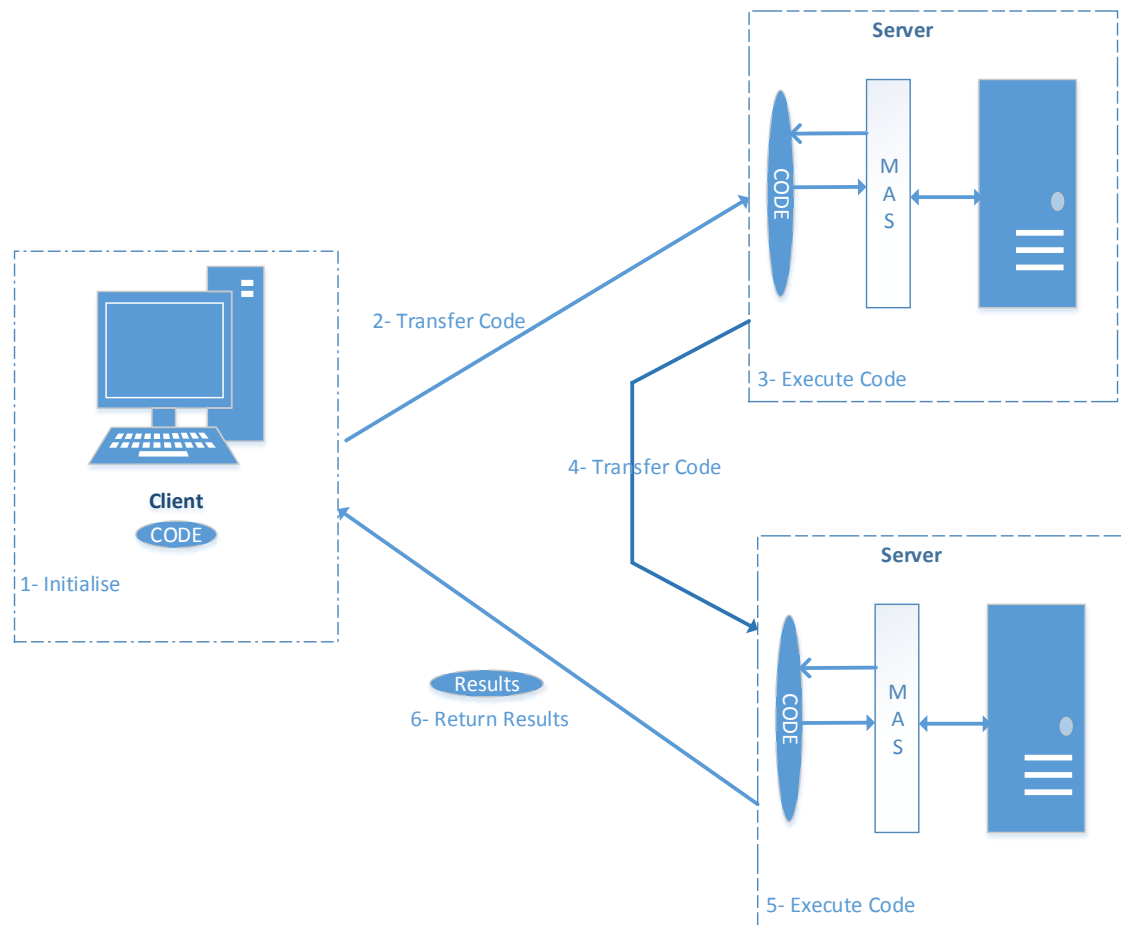


Figure 2-3 :Mobile Agent Computing Model

### 2.2.2. Mobile Agent Migration

Mobile agents perform their functions by travelling from one node to another. Their autonomous characteristic allows them to travel to any node at any time. Alternatively, pre-coded itineraries allow them to travel to the specific nodes. MAs have little or no external influence and always travel with their state of execution.

The following requirements that make it possible for mobile agent migration across a network are mentioned in (Reilly, 1998):

- *Common Execution Language*: The problem of having a common execution language persists due to the fact that the migrated agent can only perform its function if the host at the source and the destination have a common executable language. This

requirement best suits heterogeneous network instances, as the network may be composed of different network architectures.

- *Process Persistence*: Process persistence as a property of the platform (Aglet, see Section 2.3) is normally designed in the mobile agent architecture. This helps maintain its execution state during travel across the network, by changing it to a transferable form of data that could be sent in a particular state before its execution at the destination.
- *Communication Mechanism Between an Agent's Hosts* - Using techniques like shared code systems or caching is very effective in agent transfer, as it uses a minimal bandwidth. It is also suitable, as the transfer of an agent executable code or its persistent state requires the use of large network bandwidth.

The migration process can involve different considerations of how to get the best use of this feature. This process with all its issues, especially in the area of interest, will be discussed in the next chapter.

### 2.2.3. Mobile Agent Life Cycle

A Mobile Agent contains only code and status. It can only be created in a local server and each agent has a unique identifier. Both the serialisation and deserialisation of the mobile agent are possible, allowing for the suspension of its execution and resumption later on in the same or different host. The communication layer is responsible for agent remote messaging and transportation. It does so by using an application-level protocol called the Agent Transfer Protocol (ATP) which is also an internet protocol (Lange and Oshima, 1998).

The life cycle of the MA is depicted in Figure 2-4. After the MA has been created in the home node and its state configured, the MA starts performing its duties. A home node (HN) is the where the MA is being created.

During its life, the agent can travel and visit different hosts. After finishing its tasks, it can suspend, clone or terminate itself. Usually, the agent comes back to the home host to give the collected results before terminating itself.

A home node is the where the MA is being created.

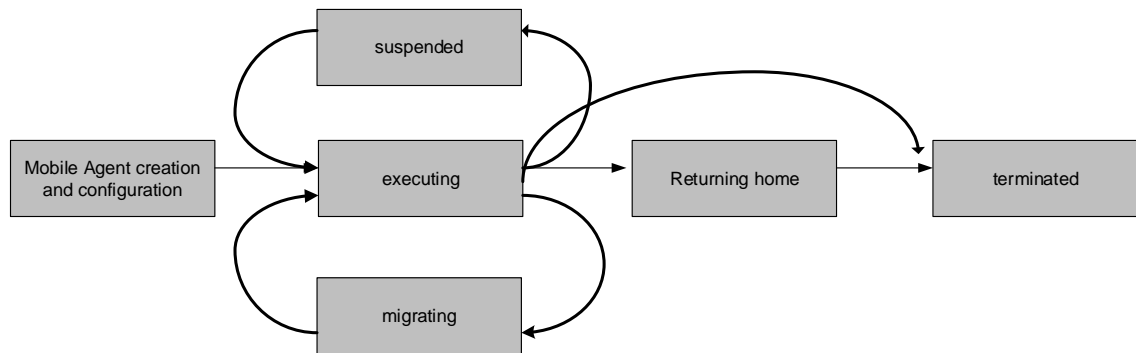


Figure 2-4: Mobile Agent Life Cycle.

#### 2.2.4. Examples of Mobile Agents

Many Mobile Agent platforms have been developed during the last decade. The platforms have been programmed using different programming languages such as Java, TCL, C/C++ and other miscellaneous codes. Some of these platforms have been developed for commercial purposes and others for research.

In this section, seven of the popular platforms are presented and compared according to (Gupta & Kansal, 2011). The platforms are as follows: Aglet, Voyager, Grasshopper, Tryllian, JADE, Tracy and SPRINGS. The work of Altmann, Gruber, Klug & Weipll shows that the Aglet, Voyager and Grasshopper platforms are the best Mobile Agent platforms (Altmann, Gruber, Klug & Weipll, 2001).



Below is a comparison between the seven popular platforms based on (Gupta & Kansal, 2011), (Trillo, Ilarri & Mena, 2007) and (Silva, Soares, Martins, & Santos, 2000).

- **Voyager**<sup>1</sup> developed initially by ObjectSpace in 1997 and currently by Recursion Software (last version: Voyager Edge 6.0.1, August 2006), is a distributed computing middleware, focused towards simplifying the management of remote communications of traditional CORBA and RMI protocols. The main drawback of Voyager is that it is a commercial product not available for free, which could/might protect many researchers from using it in favour of other alternatives available.
- **Grasshopper**<sup>2</sup> was developed by IKV++ in 1999 (last version: 2.2.4, January 2003), then became part of the commercial Enago Mobile and today its development has probably been abandoned. The main disadvantage of Grasshopper is that it is not available anymore and new versions will not appear in the future in all probability. The region server could become a bottleneck, as it must update every proxy right before using it.
- **Tryllian**<sup>3</sup> developed by the Homonym company in 2001 (last version: 3.2.0, released as open source in November 2005), is based on a sensing-reasoning action mechanism. It allows programmers to define a reactive (based on incoming messages) and proactive (based on heartbeats) behavior of agents. The main disadvantage of Tryllian is that it does not offer location transparency (the current location of the target agent of a message must be known in advance). Furthermore, its taskbased and asynchronous model could be difficult to use, due to its differences with the classical procedural programming.

---

<sup>1</sup> <http://www.recursionsw.com/>,2006

<sup>2</sup> [www.ikv.de/products/grasshoper](http://www.ikv.de/products/grasshoper)

<sup>3</sup> <http://www.tryllian.org>,2005

- **JADE**<sup>4</sup> developed by Telecom Italia Lab since July 1998, was released as open source in February 2000 (last version: JADE 3.4.1, November 2006). It is a very popular FIPA-compliant agent platform. An agent is composed of different concurrent (and non-preemptive) behaviors, which can be added dynamically. Probably, the main disadvantage is that mobility is not a key element in JADE. Thus, it focuses on other functionalities relevant to the development of multi-agent systems.
  
- **Tracy**<sup>5</sup> developed at the University of Jena in Germany (last version: 1.0.1-52, April 2005), has a plugin-oriented architecture. Plugins are software components that can be added dynamically to a running agency (the context where agents execute), if required, in order to provide high-level services (e.g., inter-agent communication, migration, security, etc.). A key disadvantage of Tracy is that it does not support remote communications between agents: an agent must travel to the agency where another agent is running in order to communicate with it. The platform was probably developed mainly as a test environment, where different migration and class loading strategies could be evaluated, and in that aspect the authors have performed a meritorious and interesting research work.
  
- **SPRINGS**<sup>6</sup> developed recently by the Distributed Information Systems Group at the University of Zaragoza in Spain , focuses on scalability and reliability in scenarios, with a moderate and high number of mobile agents. Its development has been inspired by the features of other popular platforms, such as Voyager and Grasshopper. The main disadvantage of SPRINGS is perhaps that it does not support agent communication, using the standard FIPA. In addition, it does not provide sophisticated security mechanisms. Despite the fact that it is easy to use, it does not offer any graphical tool to the user. Unfortunately as it is a new platform, there is little documentation available about it.

---

<sup>4</sup> <http://jade.tilab.com/>,2006

<sup>5</sup> <http://www.mobile-agents.org>,2005

<sup>6</sup> <http://sid.cps.unizar.es/SPRINGS/>,2006

- **Aglets**<sup>7</sup> initially developed by IBM and maintained by the open source community since 2001 (last version: 2.02, June 2004), is probably the most popular mobile agent platform developed so far. Aglets is built around a single-thread model for agents and a communication infrastructure based on message passing. Both synchronous and asynchronous messages are supported. Agents in Aglets use proxies as a convenient abstraction to refer to remote agents (e.g., to send them messages).

One of its strongest points is that it follows the MASIF specification. Aglets is built around a single-thread model for agents, and a communication infrastructure based on message passing. Aglets has contributed significantly to the field of mobile agents in terms of classes, and objects. Aglet is an open source. Moreover, it does offer any graphical tool to the user. It can be with BOUNCY Castle (a security mechanism).

From the mentioned positive points of Aglets, the Aglet platform, for this research, has been selected.

---

<sup>7</sup> <http://aglets.sourceforge.net,2004>

### 2.2.5. Adopting the Mobile Agent

Mobile agents have been accepted, tested and proven to be the generally effective way of managing networks facilitating flexibility, automation and security. The following situations that made mobile agents more suitable are:

- **Multi-processor Calculations:** Breaking calculations down into discrete units is the only way large calculations are done by processors on a large network. The discrete units are simultaneously processed by a pool of processors or servers. The function of the Mobile Agent here is to transport the discrete units to the new hosts, initiate calculations and return, with the results for possible aggregation.
- **Low-reliability or Partially Disconnected Networks:** Low reliability network connections to devices, such as laptops, make it difficult to download large volumes of data. The mobile agent given specific instructions, is capable of travelling to the source of the data and calculate, or filter out, all unnecessary data and bring the desired data back. The source of the agent does not need to be online for this execution.
- **Communication:** An agent is always capable of communicating with both the master agent of the host and other agents. Collaborating through communication with these agents helps agents to reach their goals.
- **Learning:** Mobile agents have the ability to learn. They learn from their environments while executing their functions. Learning can be done deliberately by storing large volumes of data for knowledge or by reflexive learning, where the mobile agents learn gradually from their environments as they progress. There are also traces of achievements of semi-intelligence when agents collaborate among each other.
- **Robustness:** A mobile agent must be robust enough to be able to withstand unfavourable or unexpected conditions which might cause breakdowns, damages or entire destruction itself.
- **An activity called the Checkpoint Restore mechanism** is always used to restart agents and check state information before and after execution on any server. Any agents left on the server at the time of shutdown are capable of being restarted with the execution of a recovery process when that server is restarted.

According to Pleisch and Schiper (2004a), all the problems that were considered as examples for the use of MAs could also be solved by using the traditional CS paradigm. Nevertheless, MAs allow a general solution to all problems that need to be distributed. Instead of having to create different solutions to every problem, MAs can present a standard solution to all problems. In fact, agents are already used in new software products such as Microsoft Outlook Express, Microsoft Office and other software like operating systems. Outlook Express has been developed as an email and news client bundle. It has some definitions of an agent, as previously described, by attempting to observe user actions and to offer help based on these actions. Also, the user can customise these agents to do specific jobs, such as filtering the incoming emails and blocking junk emails. The use of MAs offers several advantages over traditional solutions, for instance: communication latency and bandwidth, off-line processing, reaction time, asynchronous behaviour by request aggregation, software-distribution on demand, scalability due to dynamic deployment, heterogeneity fault tolerance, etc. These advantages make MAs an appropriate and helpful technology for various application domains.

### 2.3. Agent Platform Overview with Aglet

In this work, the Aglet platform has been used in order to design the proposed model and to test the real behaviour of the Mobile Agent in the network area. In this section, an overview about the platform will be provided.

In early 1995, the research laboratory of IBM Tokyo developed the *Aglet API* as an agent development kit. Danny B. Lange, a member of the research team, initiated this effort as cited by Lange and Oshima (2003). An Aglet can simply be defined as Java mobile agents. In other words, it is a Java object capable of transporting itself from one host to another on a network while maintaining their code and data. The earlier version of Aglet developed was Version 1.0.3 which later, through the open source project, became available as the later versions (Aglet 2.0.2). It is also mentioned in Lange and Oshima, (2003) that someone can think of an Aglet being a generalized and extended version of a Java applet and servlet. It also acknowledges the concept of having its itinerary dynamically generated and executed autonomously.

Aglet executes in an Aglet server host. The default Aglet server host is called Tahiti. Figure 2-5 shows the Graphic User Interface (GUI) of the Tahiti server running on default port number of 4434.

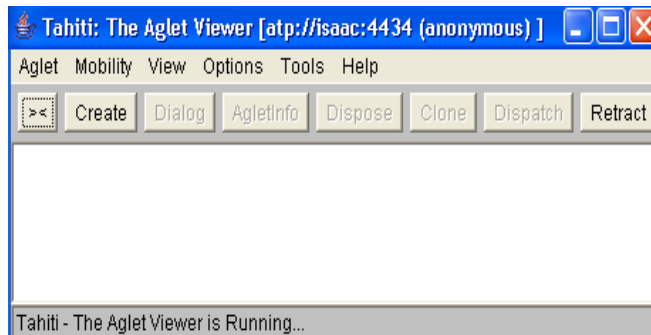


Figure 2-5: Tahiti Graphic User Interface (GUI)

### 2.3.1. Aglet System Architecture

The Aglets architecture consists of two layers (runtime layer and the communication layer) and two Application Program Interfaces (APIs) that define interfaces for accessing their functions. Figure 2-6 shows the Aglet architecture with the two layers and the API.

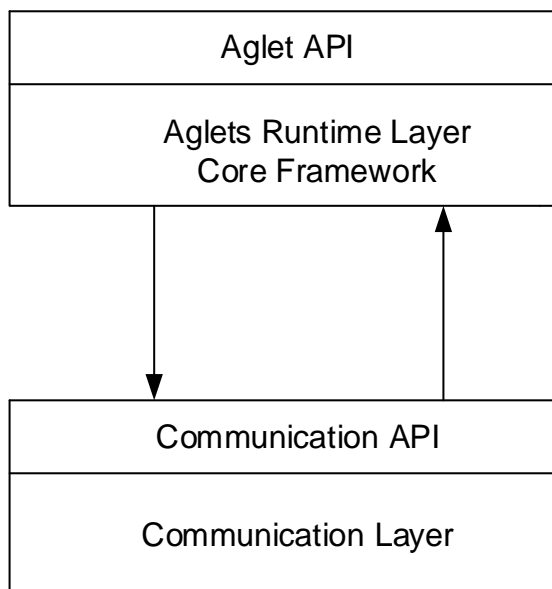


Figure 2-6: Aglet Architecture

The Aglets runtime layer implements Aglets interfaces such as AgletContext and AgletProxy. It consists of a core model and subcomponents as well. The core model provides the fundamental mechanisms to Aglet execution, i.e. serialization and deserialization of Aglets, class loading and transfer, and reference management and garbage collection.

The Aglets runtime itself has no communication mechanism for transferring the serialized data of an Aglet to destinations. Instead, the Aglets runtime uses the communication API that abstracts the communication between agent systems. This API defines methods for creating and transferring agents, tracking agents, and managing agents in an agent-system in an independent way.

The current Aglets uses the Agent Transfer Protocol (ATP) as the default implementation of the communication layer.

### 2.3.2. Aglet Model

A closer look at the model underlying the Aglets API is taken care of in this subsection. The key abstractions are aglet, proxy, context and message. Definitions are as follows:

- **Aglet:** An aglet is a mobile java object that visits aglet enabled hosts in a computer network.
- **Proxy:** A proxy is a representative of aglet. It serves as a shield for the aglet to protect the aglet from direct access to its public methods. The proxy also provides location transparency for the aglet.
- **Context:** A context is aglet's workplace. It is a stationary object that provides a means for maintaining and managing the running aglet in a uniform execution environment, where the host system is secured against the malicious aglet. One node in a computer network may run multiple servers and each server may host multiple contexts. Contexts are named and can be located by the combination of their server address and their name.
- **Message:** A message is an object exchanged between aglets. It allows for synchronous as well as asynchronous messages passing between aglets. Message passing can be used by aglets to collaborate and exchange information in a loosely coupled fashion.

### 2.3.3. A tour of the Aglet API

The aglet API is used by programmers to create and operate aglets. It contains methods for initializing an aglet, message handling, dispatching, retracting, deactivating, activating, cloning and disposing of the aglet. The aglet API is a Java package (aglet) consisting of classes and interfaces, most notably, Aglet, AgletProxy, AgletContext and Message.

- **Aglet Class:** The Aglet class is the key class in the Aglet API. This is the abstract class that the aglet developer uses as a base class, when he or she creates customized aglets. The Aglet class defines methods for controlling its own life cycle, viz. methods for cloning, dispatching, deactivating and disposing of itself. It also defines methods that are supposed to be overridden in its subclasses by the aglet programmer and provides the necessary "hooks" to customize the behaviour of the aglet. These methods are systematically invoked by the system when certain events take place in the life cycle of an aglet.
- **AgletProxy Interface:** The AgletProxy Interface acts as a handle of an aglet and provides a common way of accessing the aglet behind it. Since an aglet class has several public methods that should not be accessed directly from other aglets for security reasons, any aglet that wants to communicate with other aglets has to first obtain the proxy and then interact through this interface. In other words, the aglet proxy acts as a shield object that protects an aglet from malicious aglets. Another important role of the AgletProxy interface is to provide the aglet with location transparency. If the actual aglet resides at a remote host, the proxy forwards the requests to that remote host and returns the result to the local host.
- **AgletContext Interface:** An aglet spends most of its life in an aglet context. It is created in the context, it goes to sleep there, and it dies there. When it travels in a network, it really moves from context to context. In other words, the context is a uniform execution environment for aglets in an otherwise heterogeneous world. The AgletContext interface is used by an aglet to get information about its environment and to send messages to the environment, including other aglets,



currently active in that environment. It provides means for maintaining and managing running aglets in an environment where the host system is secured against malicious aglets.

- **Message Class:** Aglets communicate by exchanging objects of the Message class. A string field named "kind" distinguishes messages. This field is set when the message is created. The second parameter of the message constructor is an optional message argument.

#### 2.3.4. Security

As pointed out in the proposed solution, the prototype development exploits cryptographic concepts to some extent. Hence cryptographic algorithms need to be incorporated in the developed program. Fortunately, Java is currently perceived as the most secure object-oriented programming language. Security in Java was enhanced by reviewing other object-oriented programming languages like C and C++. Additionally, it has a support for a seamless integration of a third party implementation of cryptographic services providers (just providers) through its Java Cryptographic Architecture (JCA)<sup>8</sup>. Various providers exist for the Java platform. Some are free for academic purposes while others are extremely expensive. In the developed prototype, a cryptography provider called Bouncy Castle<sup>9</sup> is used.

The Bouncy Castle, also called BC provider, offers a pure Java implementation of different cryptographic algorithms. A range of other crypto- based products can also be found on the site. It is a free cryptographic package, where its running and maintenance are carried out by volunteers. It has a support for a wide range of various cryptographic algorithms at various levels of strength. True to its reputations, its jar file is signed by Sun, so that it can be easily integrated into the platform. This provider is integrated into the Java platform and its various crypto packages are extensively used to provide cryptographic services for the various parts of the agent system.

---

<sup>8</sup> <https://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html>, 2002

<sup>9</sup> <http://www.bouncycastle.org>, 2016

## 2.4. Malicious Hosts on Mobile Agents and their Countermeasures

The previous chapter has provided a detailed view of the makeup of mobile agent systems. Security is one of the key factors of MAS. In fact, a MA is one of the potential threats to computer systems and vice versa, ie. , from the host system to the MAS itself. In this part of the dissertation, the main security issues related to MAS will be described in details.

### 2.4.1. Malicious Hosts

The four main forms of attacks by malicious hosts on mobile agents are:

- **IP spoofing:** consists of sending packets with a faked IP source address. The server should believe that the packets come from another host, probably a host that is allowed to establish connections with the attacked host, if the real one is not allowed.
- **Sniffing:** it is the observation and analysis of network traffic in order to obtain relevant information (such as IP addresses and host functionalities) to perform other attacks.
- **Denial of Services:** A host may deny an agent a specific service provided by the host. It is possible for a host to both intentionally and unintentionally deny an agent a service. A host may deny an agent service so that the agent is not able to complete its task. Another possible attack is where the host could terminate the agent altogether. Furthermore, a host may deny a request from an agent on a time-sensitive task, so that the agent is unable to complete its task in its allotted time.
- **UDP flood attack:** this kind of flooding attack consists on sending many UDP packets to a different port of a target in a random fashion. This target will check if there is any application on the relevant port; if not, he will be occupied to send ICMP replies and cannot treat requests from legitimate clients.

- **SYN flood attack:** it consists of sending many TCP connection requests to a target. The latter will accept the establishment of the connection and notify the client. Except for this, this one will never use them. Thereby, the server will be drown by unused connections and, eventually, will not reply to legitimate users requests.
- **Eavesdropping:** The classical eavesdropping threat involves the interception and monitoring of secret communications. The threat of eavesdropping, however, is further exacerbated in mobile agent systems because the agent platform can not only monitor communications, but also can monitor every instruction executed by the agent, all the data it brings to the platform, and all the subsequent data generated on the platform. Since the platform has access to the agent's code, state, and data, the visiting agent must be wary of the fact that it may be exposing proprietary algorithms, trade secrets, negotiation strategies, or other sensitive information. Even though the agent may not be directly exposing secret information, the platform may be able to infer meaning.
- **Alteration:** The final form of attack by a host on an agent is the alteration of the agent. The host can alter an agent by changing the data, code and control flow. A malicious host may try to change the code of an agent so that the agent performs other tasks than were intended by its creator. A host may also try to change the data contained in the agent.

#### 2.4.2. Countermeasures for Security Services

There are many security services that can be used for securing the agents systems, for example, authentication, integrity, confidentiality and authorization.

In case of the **authentication**, the host needs to know the sender of the delivered agent. The agent authentication process includes verifying the entity that programmed the agent and also verifying the entity that dispatched it to the host. Basically, the agent and the host need to know whom they are talking to and dealing with; here the public-key encryption or passwords can be used.

Regarding **integrity**, checking the integrity of the agents is a technique that makes sure no one has made any changes to the agents, the agents travelling from one host to another, and communicates and exchanges their data with other hosts and other agents. In this case, it is required to make sure that the agents have not been tampered with in relation to their state, code or data. Moreover, the agents could carry different types of data, for example some private data. These data should only be readable from a specific host or agents. This technique is very important to avoid an eavesdropping threat.

The last service which helps to protect the agents and the hosts is **authorization**; the incoming agents should have a specific right to access the host information, so different agents have different authority, to protect the hosts and also to protect themselves.

## 2.5. Summary

This chapter has given an overview of Mobile Agent systems and has addressed the main features of Mobile Agents such as mobility, portability, communication, etc. It has also shown that the MAs offer several advantages over traditional solutions of network management, for instance, communication latency and bandwidth, off-line processing, reaction time, asynchronous behaviour by request aggregation on demand which give MAs a substantial advantage over traditional technology, in detection and protection against malicious hosts in the mobile agent world.



## Chapter 3 : Literature Review

---

### 3.1. Introduction

For wide scale applications, the approaches to protect an agent can be broadly classified as two main mechanisms:

- Detection Technique attempt to detect unauthorized modification of code, state or execution of mobile agent.
- protection technique attempt to try to make it impossible to access or modify code, state or data of mobile agent.

The purpose of this chapter is to give an overview of existing detection and protection mechanisms on mobile agents and their drawbacks. Consequently, this will explain the reasons that motivate and push us to provide this work.

### 3.2. Existing protection Techniques and their Drawbacks

Several ways of existing protection techniques are used in order to protect the mobile agents from malicious hosts.

#### 3.2.1. Distributed Technique for Multi-Agents

E.Abolfazl and M.R.Ali have proposed a novel distributed protocol for multi agent environments to improve the communication security in packet-switched networks. This approach makes use of distribution, double encryption and some other traditional methods such as digital signature (Abolfazl and Ali, 2009). In this approach, the encrypted message and encrypted private key are broken into different parts carried by different agents, which makes it difficult for malicious entities to extract the private key for message encryption, while the private key for the encrypted key is allocated on the predetermined destination nodes. Every part is assembled and decrypted by different mobile agents along different routes to the destination.

The main advantages are that double encryption used in this approach prepares an appropriate infrastructure for today's critical areas such as e-commerce or NCW.

The main drawback of this technique is that the computation load of the approach is larger.

### 3.2.2. Environmental Key Generation Technique

Environmental Key Generation describes a scheme for allowing an agent to take predefined action when some environmental condition is true. The approach centers on constructing agents in such a way that upon encountering an environmental condition (for example, string match in search) a key is generated which is used to unlock some executable code cryptographically. The environmental condition is hidden through either a one-way hash or public key encryption of the environmental trigger. The technique ensures that a platform or an observer of the agent cannot uncover the triggering message or response action by directly reading the agent's code. The procedure is somewhat leading to the way in which passwords are maintained in modern operating systems and used to determine whether login attempts are valid.

One weakness of this approach is that a platform that completely controls the agent could simply modify the agent to print out the executable code upon receipt of the trigger, instead of literally executing it. Another drawback is that an agent platform typically limits the capability of an agent, to execute code created dynamically, since it is considered an unsafe operation. An author of an agent can apply the technique in conjunction with other protection mechanisms for specific applications on appropriate platforms.

### 3.2.3. Secure Software technique for MA Execution

Mousa and Ljiljana have proposed a software solution for secure execution of mobile agents under untrusted execution environment (Mousa and Ljiljana, 2005). As per their proposal, the agent home made its code hidden from the remote host on which it executes to maintain privacy. Home platform has an algorithm in form of function  $f$ . At remote sites, the target host has data (input)  $x$  and it computes  $f(x)$  to provide services to agent. To secure the function  $f$  so that remote host cannot read this, home platform encrypts the

function  $f$  to get  $E(f)$  and then embodies encrypted function within program. Home platform inserts this program within agent code and sends it to the remote host platform for execution. The target platform runs program on input  $x$  and produces  $E(f(x))$ , then the generated output (final results) is sent back to its home platform. Home platform decrypts it and gets  $f(x)$ . The main advantage is that this mechanism enables the agent to execute in secure manner at remote untrusted platforms.

The main drawback is that it is not capable of protecting the system from denial of service attack and replay attack.

#### 3.2.4. DDOS Protection Technique

Timcenko proposes a protection mechanism for distributed denial of service (DDoS) attacks in Mobile Ad hoc Networks (MANET) environment (Timcenko, 2014). This approach relies on the investigation of widespread bandwidth attacks, with focus on Distributed Denial of Service (DDoS) attacks, which are extremely dangerous, are hard to detect and challenging to be protected. DDoS represents a coordinated activity of a group of attackers aiming to protect legitimate users the access to network resources. The paper analyses the new proposed mechanism (Flexible MANET Prevention Algorithm (FMPA)) under two cases: 5 % and 10 % intruders and was successfully able to detect three consecutive attacks, generated by a synchronous activity of a group of nodes.

The power of DDoS attacks is intensified with use of multiple attack sources, thus providing favourable conditions for jeopardizing network security. The factor of attack duration and repetition can additionally reinforce the attack effect, weaken network performances and protect legal users the access to network services.

The main drawback is the extra cost in memory and computation time necessary to execute the technique.

### 3.2.5. Multi-Facet Security Approach

The multi-facet security approach is proposed by (Ngreki and Kahonge, 2015) in order to protect against malicious host. This security strategy is purposely to contain most of the threats dogging multi-agent systems, especially for the malicious hosts attack. In their work, Time-To-Live TTL and heartbeat methods were adopted as technique to tackle the DoS attack, encryption method provided confidentiality and integrity of agents' code.

The strength of the work is tested by proposing an environment of 6 hosts where the job of some of them is to corrupt other multi-agent systems' states and data by injecting random data and setting the agents' states to random values. Hosts are simulated by having them listen on different ports. Results have shown that traffic data is encrypted and thus cannot be easily interpreted.

However, the mechanism failed to offer protection against the repudiation attack.

### 3.2.6. Obfuscation technique

Another technique for protection is Obfuscation; it is a technique in which the mobile code producer enforces the security policy by applying a behaviour preserving transformation to the code before it sends it to run on different platforms that are trusted to various degrees. Obfuscation aims to protect the code from being analysed and understood by the host. Consequently, the host should/ would not be able to modify the mobile code's behaviour or expose sensitive information that is hidden inside the code such as a secret key, credit card number, etc. Typically, the transformation procedure that is used to generate the obfuscated code aims to make the same very hard to understand or be analysed by malicious parties. There are different, useful obfuscating transformations. Layout obfuscation tries to remove or modify some information in the code, such as comments and debugging information, without affecting the executable part of the code. Data obfuscation concentrates on obfuscating the data and data structures in the code without modifying the code itself.

Hohl suggests using the obfuscation technique to obtain a time limited black box agent that can be executed safely on a malicious platform for a certain period of time but not forever (Hohl, 1998). The central idea is not to allow an attacker to build such a mental model of the agent in advance (before the agent arrives, and to make the process of building this model a time-consuming task). The first goal is reached by creating a new



“form” of the agent dynamically, in an unpredictable, manner at the start of the protection interval. The second goal is reached by using conversion algorithms that produce a new form that is hard to analyse. In this context hard means that the analysis should take as much as time as possible. These conversion algorithms are therefore called obfuscating or mess-up algorithms.

D’Anna points out that obfuscation could delay but not protect the attacks on agent via reverse engineering (Larry D’Anna, 2003). They also argue that an attacker with enough computational resources, such as enough time, can always de-obfuscate the code. Barak et al studied the theoretical limits of obfuscation techniques and showed that in general achieving completely secure obfuscation is impossible. In addition to protecting a mobile agent, obfuscation can also be used for other applications such as protecting digital watermarking, enforcement of software licensing and protecting protocols from spoofing. As far as the performance is concerned, some obfuscation techniques reduce the size of the code and thus speed up its execution, while others achieve the opposite (control obfuscation). Obfuscation is considered resistant to impersonation and denial of service attacks. This technique is flexible and inexpensive. Depending on the need for security, an application can be obfuscated accordingly. In this technique, there is a possibility to create different instances of one software application to battle global attacks. It has low maintenance cost, due to automation of the transformation process and compatibility with systems. This technique is platform independent.

This technique has a number of drawbacks: for instance, every transformation introduces extra cost in memory and computation time necessary to execute the obfuscate program. Obfuscation does not provide waterproof security. Most of these code transformations are not one-way and it is hard to decide where to use which transformations.

Sandya defined obfuscation as an approach where an agent owner (home agent platform) enforces a security policy by applying a behavior preserving transformation to the code before it is dispatched to run on unknown and known hosts. The code obfuscation creates a blackbox entity where agent’s program is illegible and hides data thereby rendering it difficult for attackers (malicious hosts) to read and modify the agent’s code, its data and partial results (Sandya, 2015).

However, this approach offers short-lived code obfuscation as given time, the malicious platform will be able to de-obfuscate the code.

### 3.3. Existing Detection Techniques and their Drawbacks

Several ways of existing Detection Techniques are used to detect unauthorized modifications of code, state or execution of mobile agent.

#### 3.3.1. Watermarking and Fingerprinting Detection Techniques

Oscar, Marcel and Miguel have proposed two traceability techniques, viz., watermarking and fingerprinting (Oscar, Marcel and Miguel, 2003). In these techniques, a mark is embedded into the agent and the agent's execution creates marked results. When an agent returns to its original host, these results are examined. If the mark has changed or has disappeared, this means that the executing host has modified the agent. In the agent's watermarking scheme, the mark is embedded into mobile agent's code, because all executing hosts in the agent's itinerary must run the same marked code. But in agent's fingerprinting scheme, the embedded mark is different for each host because mark is embedded into agent's data and data is usually different for each host.

However, the main drawback of these techniques are: increase in its code and data size. This is because embedding a mark always means that some overhead is added to the mobile agent. Moreover, a TP (Trusted Third Party) is needed in order to punish malicious behaviour.

#### 3.3.2. Recording and Tracking Technique

Wayne has proposed a general technique that allows an agent's itinerary to be recorded and tracked by another cooperating agent and vice-versa, in a mutually supportive arrangement (Wayne, 2000). When moving between agent platforms, an agent conveys the last platform, the current platform, and next platform information to the cooperating peer through an authenticated channel. The peer maintains a record of the itinerary and takes appropriate action when inconsistencies are noted. Attention is paid in this scheme so that an agent avoids platforms already visited by its peer.

The main advantages of this technique is that by dividing up the operations of the application between two agents, certain malicious behaviour of an agent platform can be detected. Moreover, this scheme can be incorporated into any appropriate application.

The main drawback of this technique includes the cost of setting up the authenticated channel and the inability of the peer to determine which of the two platforms is responsible, if the agent is killed.

### 3.3.3. Itinerary Recording with Replication and Voting Technique

Wayne has proposed a technique (Itinerary Recording with Replication and Voting) for detecting malicious behaviour of an agent platform by replicating mobile-agents and voting on results of their computation (Wayne, 2000).

This technique is based on the idea that instead of using a single copy of an agent to perform a computation, multiple copies are used. Although a malicious platform may corrupt a few copies of the agent, enough replicas avoid the encounter to successfully complete the computation. This technique seems appropriate for specialized applications where agents can be duplicated without problems, and the task can be formulated as a multi-staged computation.

The most noticeable advantage of this technique is that this approach is taken similar to path histories, but extended with fault tolerant capabilities.

The major drawback of this technique is that additional resources are consumed by replicate agents.

### 3.3.4. State Appraisal Technique

A state appraisal technique has been proposed, wherein an architecture through which changes in a mobile-agent execution states are monitored; it was initially suggested by Farmer et al. (Farmer, Guttman and Swarup, 1996). This scheme was specifically designed for multi-hop agent architectures, in which an agent moves through several platforms (with various levels of trust) in-order to perform the required tasks. State appraisal can be used to detect tampering on the agent by malicious platforms, so that further damage to agent code and information disclosure is minimized. State appraisal requires agents to decide on privileges they will need at an intended destination platform. A state appraisal

function is used to compute a set of privileges to be requested for, based on the state of the agent when it arrives at the new destination platform. Using this state information, dangerous modifications to the agent's state can be detected and appropriate counter actions triggered. This technique is implemented as a "layer" above the authentication phase during mobile-agent execution, to provide input to any implemented authorization mechanisms, as previously indicated by (Jansen,2000).

The main challenge to this technique is the large space of appraisal states for an agent, which makes it difficult to foresee all forms of attacks that a malicious platform may launch.

### 3.3.5. Enhanced Reference Monitor based Security Framework

Akomolafe and Honesty introduced a Trusted Third Party technique designed to allow the offloading of the computation-intensive verification mechanism for execution. The Trusted Third Party fulfils the requirements of an efficient computing environment to mobile agents. Altogether, with the enhanced security framework, it is still possible for a mobile agent to migrate to any agent platform (AP) with the assurance of security. The enhanced security framework is most suitable for open systems where protection or security against malicious hosts and support for mobile agent computing with static as well as dynamic itineraries are required (Akomolafe and Honesty, 2017). The results of the approaches have reduced the resource consumption: the memory usage is decreased to 35.80% and the time to complete the verification mechanism downgraded to 53.54%

However, this framework is not tested and integrated with multi-agent systems where remote hosts are unknown and untrusted.

### 3.3.6. Trusted Hardware technique

A trusted hardware technique tries to enforce the notion of trust between an agent and a host by physically adding, secure - tamper detecting and responding hardware to conventional computing systems. The hardware encapsulates the entire environment in which the agent executes, creating a safe haven within hosts in the agent space. It protects the visiting agent from any possible attack that could be launched by the entertaining host. The major drawback of this proposition is that it comes up with another prerequisite for an agent, to execute at a given host. After all, all that is all needed for an agent to execute

at a given host is the availability of Mobile Agent Platform. But now, the agent also needs the existence of the “Safe Heaven” (literally speaking tamper resistant and detecting hardware) at each host it is going to visit. This has the effect of constraining the itinerary the agent could follow, dramatically limiting the systems’ openness.

### 3.3.7. Intrusion Detection Techniques

Several approaches of mobile agents to intrusion detection have been executed.

#### 3.3.7.1. *Multi-agent-based IDS*

One of the well-known multi-agent-based IDS is the Autonomous Agents for Intrusion Detection (AAFID) (Spafford and Zamboni, 2000). Within this architecture, a set of agents monitors specific aspects of a machine requiring security. These agents send information to the transceivers, which in turn, amalgamate the information and re-send it to the monitors. The monitors process this information and decide whether or not an attack has occurred. To make the system more scalable, the monitors may be built in layers.

Although this system was very innovative in its time, its main drawback is the extreme rigidity of its architecture, as this makes the introduction of new agents very complicated. Moreover, its hierarchy is such that, if an attack manages to deactivate an agent in the upper layers, the entire system might be deactivated.

#### 3.3.7.2. *MAIDS Technique*

One of the studies worthy of mention was the MAIDS architecture, proposed in Li et al., (2004). The MA-IDS system employed mobile agents to coordinately process information from each monitored host. Its architecture includes the Assistant and the Response mobile agents. The Assistant mobile agent is dispatched by the manager component to gather information in the network. The mobile agents within the MA-IDS system are capable of evading attackers and resurrecting themselves if they are attacked. Moreover, the mobility of agents makes it possible that distributed intrusion can be detected by means of data correlation and cooperative detection.

However, whenever the location of manager was found by attackers, the IDS faces a dangerous situation. Thus, MA-IDS suffers from the drawback of single point of failure.

### 3.3.7.3. *Distributed IDS Technique*

Wang et al. proposed a distributed IDS, which includes a Manager and a Host monitor (Wang and Hu 2006). The components in such a model are designed as mobile agents for the purpose of high adaptability and security of the system. It is claimed that the mobile agents of the proposed system can evade intrusion and recover by themselves if they suffer from intrusion. Nevertheless, the system uses a control center to carry out the major part of the intrusion detection. Consequently, if the location of this center is discovered, then the system collapses.

### 3.3.7.4. *Intrusion Detection Processor (IDP) & Sensors Technique*

Mo et al. implemented a misused mobile agent-based IDS (Mo et al., 2009), which incorporates the SNORT system (Roesch, 1999). The architecture consists of three different components: (1) an intrusion detection processor (IDP), (2) a mobile agent platform (MAP), and (3) distributed sensors or sniffer. Indeed, the IDP is responsible for monitoring network segments. Besides, the MAP is responsible for accepting requests made by the IDP and generating mobile agents as well as sending them into the network to start sniffing activities within the local network, to stop it when necessary, and to send the collected data to the IDP for further analysis. Finally, the sniffer is responsible for gathering data. The mobile agents in this work are fully managed and network resources utilization is saved when there is no attack. However, the proposed system suffers from a high false positive rate, since many attacks could be missed.

### 3.3.7.5. *Simple System for Multi-Agents for IDS*

Ionita describes a simple system based on multi agents for Intrusion Detection, utilized to run on general purpose devices such as web servers; this system with considerable knowledge is capable to detect port scans and “syn” attacks against a specific port, and it is capable of performing actions to prevent the suspicious attack. Also, by using its feature of mobility, which is characteristic to mobile agents, the system has capability to protect several hosts in a small time frame, by triggering agents which go onto the hosts and reply the protective actions has done on the place where the attack was detected initially (Ionita, 2013).

### 3.3.7.6. Multilayer Detection Technique

Abdurrazaq, Bambang and Rahardjo propose a multilayer technique with a Multi-agent and multilevel technique (Abdurrazaq, Bambang and Rahardjo, 2014). In the 1st level, the information is collected; then this information is divided among the multiple agents. They have their own rule to verify this. In the second level, the resulting information is gathered and passed to the different layers such as application, session and transport. In the third level, another algorithm is to detect the intrusion by agent. The benefit of multilevel detection process here is that detection is error-free and all the low-level and high-level intrusion is detected by the different level.

However, the drawback of this technique is that it is appropriate to any application.

The following table 3-1 summarizes the comparative analysis of approaches of securing mobile agents.

Table 3-1 Comparative Analysis of Security Solutions to Protect Mobile Agents

Proposed Mechanisms	Parameters						
	Protect Masquerading	Protect Eavesdropping	Protect Unauthorized Access	Protect DOS	Protect Copy & Replay	Detect Tampering	Implementation Available
Trusted Hardware	Yes	No	Yes	Yes	Yes	Yes	Yes
Mutual Itinerary Recording	No	No	No	No	No	Yes	Yes
Itinerary Recording with Replication & Voting	No	No	No	No	No	Yes	Yes
Computing with Encrypted Functions	No	YES	Partial	NO	NO	NO	YES
Partial Result Encapsulation	YES	Yes (Encryption)	Partial	Yes (Encryption)	Yes	Partial	YES
Partial Result Authentication Codes	No	N	Partial	No	No	Partial	YES
Environmentl Key Generation	Yes	Yes	Partial	No	No	Partial	No
Execution Tracing	Yes	No	No	Yes	Yes	Yes	No
Obfuscated code	No	Yes	Yes (Limited Time)	No	No	Yes (Limited Time)	Yes

### 3.4. Current Divergence Measures and Sketch Technique

Several researches have been done on detection of anomalies in IP networks using divergence measures and sketch technique.

Existing methods for anomaly detection are based on different techniques, such as Haar-wavelet Analysis Entropy based method and Holt-Winters Seasonal Forecasting Method. Haussler and Opper compare two different algorithms (CUSUM and Adaptive Threshold) for the detection of SYN flooding attack. They conclude that CUSUM performs better than Adaptive Threshold, in terms of detection accuracy of low intensity attacks. However, both of these algorithms face problems of false alarm ratio under normal IP traffic variation.

Other work aggregates the whole traffic in one time series, and applies a change point detection algorithm to detect the instant of anomaly occurrence. The latter has good performance in terms of spatial and temporal complexities, but presents the drawback of aggregating all traffic in one flow, where low intensity attacks cannot be detected. Furthermore, these methods use static threshold for detecting anomalies, which is not adequate with traffic variations, and may induce false alarm and miss detection.

Sketch data structure uses the random aggregation for more grained analysis than aggregating the whole traffic in one time series. It has been used to summarize monitored traffic in a fixed memory, and to provide scalable input for time series analysis. Tang and Zhou propose the use of Cumulative SUM (CUSUM) over the sketch for network anomaly detection, (Tang and Zhou, 2009). Furthermore, they propose a new mechanism for Sketch inversion and malicious flows identification. Then, the Sketch data structure to derive probability distributions is exploited.

In addition, recent work experiments the histogram-based detector in order to detect the anomaly behaviours and changes in traffic distributions. They apply Kullback-Leibler divergence between the current and previous measurement distributions.



Wijesekera, Duminda and Jjodia apply Hellinger distance (HD) on Sketch data structure, in order to detect divergence between current and previous distributions of the number of SIP INVITE request (Wijesekera, Duminda and Jjodia, 2008). In fact, HD must be near zero when probability distributions are similar, and it increases up to one, whenever the distributions diverge (e.g. under Invite flooding attacks). In addition, they used the dynamic threshold proposed during their experimental analysis.

Ehlert, Wang, Magedanz and Sisalem presented a scheme to detect the SIP flooding DoS attacks. SIP transactional models are built to detect deviations from normal behaviors (Ehlert, Wang, Magedanz and Sisalem, 2008). However, these schemes are customized specifically to the SIP protocol and cannot be easily generalized to be applied in other flooding detection case.

All the above mentioned studies used the divergence measures and sketch technique to detect anomalies in IP Networks.

Based on these techniques, the aim of the thesis is to integrate these methods for detection of malicious hosts, not only in IP/SIP Network, but in a very specific one which is the mobile agent network .

### 3.5. Motivation for Improving the Protection of Mobile Agents

Based on what is shown in the previous section, it is clear that the existing solutions for protection and detection of mobile agents present many limitations.

For the protection system the many drawbacks can be summarised as below:

- Computation load and code is large (Abolfazl and Ali, 2009)
- Some computing with encrypted functions is not capable to Protect the system from denial of service (Mousa and Ljiljana, 2005).
- Efficiency of some these solutions, i.e generated sub-agent mechanism, are very low (D'Anna, 2003).

For the Intrusion Detection System, many disadvantages can be summarised as below:

- Lack of Efficiency: Host-based IDSs often slow down a system and network-based IDSs drop network packets that they don't have time to process (Spafford and Zamboni, 2000).
- High Number of False Positives: False alarms are high and attack recognition is not perfect (Duraipandian and Palanisamy, 2014).
- Spend a huge amount of money and time to be deployed (Mirkovic and Reiher, 2005).
- Limited Flexibility: Intrusion detection systems have typically been written for a specific environment (Abdurrazaq, Bambang and Rahardjo, 2014).
- Limited Response Capability: IDSs have traditionally focused on detecting attacks. While detection serves a useful purpose, often times a system administrator is not able to immediately analyse the reports from IDS and take appropriate action.
- No Generic Building Methodology: In general, the cost of building IDS from available components is considerable, due, in large part, to the absence of a structured methodology.

These drawbacks motivate to propose solutions for the protection of mobile agents over malicious hosts. Table 3-2 presents a summary of drawbacks and advantages of the discussed detection and protection techniques.

Table 3-2 Detection and protection Drawbacks Techniques

Techniques	Category	Advantages	Drawbacks
Distributed Technique for Multi-Agents	Protection	double encryption used is used	Increase in Computation
Environmental Key Generation	Protection	Passwords are maintained in modern operating systems	Limitation of MA to execute the code dynamically
Software solution for secure execution of mobile agents under untrusted execution environment	Protection	MA executes in secure manner at remote untrusted platforms.	Not able to protect against DDOS
Obfuscation	Protection	Low maintenance due to automation of the transformation process	Required additional memory and computation time
Watermarking and Fingerprinting	Detection	Agent's execution creates marked results	Increase in code and data size
Recording and Tracking	Detection	Malicious behavior of an agent platform can be detected & incorporated into any application	Required additional materials for setting up the authenticated channel
Itinerary Recording with Replication and Voting	Detection	Fault tolerant capabilities	Additional memory required and increase in computation
State Appraisal	Detection	Detect tampering on the agent	Difficult to foresee all forms of attacks
Multi-agent-based IDS	Detection	Capable of evading attackers and resurrecting themselves if they are attacked	Rigidity of its architecture
Distributed IDS	Detection	Can evade intrusion and recover by themselves if they suffer from intrusion	location of this If the center is discovered, then the system collapses

The proposed new computing model, for protecting against malicious hosts, is based on trust where it is combination of two kinds of trust: policy enforcement and one based on control and punishment. A detailed description of this approach, as well as its implementation and a test against malicious hosts will be performed in order to verify the robustness of the approach.

This new model for detecting these malicious hosts, is proposed and is based on Divergence measures (Power Divergences, Hellinger Distance and Chi-square) and sketch technique. This type of integration can be considered as unique in comparison with all existing solutions. The performance of the proposed model is investigated in terms of detection probability and false alarm ratio. A focus on tuning the parameter of Divergence Measures to optimize the performance will be provided. Then, performance analysis over publicly available real IP traces, in Mobile Agent Network, integrated with flooding attacks is conducting. Finally, an evaluation of the performance of the proposed divergence measure via the receiver operating characteristic (ROC) will be performed.

### 3.6. Summary

This chapter has given an overview of existing detection and protection mechanisms on mobile agents. It has addressed the main issues in the area. Moreover, it has shown the necessity of providing a new approaches of protecting mobile agent's world against malicious hosts. An overview of the previous works in the area of protection, detection of mobile agents as well as the existing divergence measures and sketch technique has been given. The next chapter will give an overview of MA systems.



## Chapter 4 : Detection Proposed Model

---

### 4.1. Introduction

As mentioned in the previous chapters, networks have become more and more advanced in terms of their size, complexity, increased number of users and devices and the level of heterogeneity. Delay, jitter, quality of service, availability are the most significant challenges in any network.

The Client-Server (CS) model is suitable for small and limited applications or network. Undoubtedly, this limits the flexibility with which a client can use a server, by processing the server data locally according to the client's needs.

The concept of a Mobile Agent (MA) has been introduced, which could migrate between nodes, carrying the necessary code to be executed (Nikaein, 1999). MAs offer a real benefit to the system developers and the users; this application specifying which parameters can be improved upon, for example performance, connectivity, reliability and modularity. The importance of these key parameters may vary from one application to another. MAs with integral intelligence can present a reasonably new technology that will generously help to achieve distributed management; several researchers have embraced these approaches. However, as mentioned in the previous chapter, MA has a big concern in the security realm.

In this Chapter, the techniques to be used in order to detect against flooding attacks in mobile agent network will be introduced, viz., Sketch technique, Divergence measures (Chi-square, Power Divergence and Hellinger Distance) as well as the Dynamic Threshold. A new model for detection of attacks is proposed. Experimental results will be shown in detail in Chapter 6. We will expressly propose how to build up an analytical model, which can be used to quantitatively assess the performances of the MA and CS paradigms under different scenarios.

## 4.2. Theoretical Background

### 4.2.1. Sketch Techniques

In this section, the K-ary Sketch data structure is going to be reviewed. Using the Sketch data structure makes the model flexible and scalable for grained analysis. No matter how many flows exist in the traffic, Sketch generates fixed-number of time instances for anomaly detection. Sketch provides more grained analysis than aggregating whole traffic in one-time instances (Thorup and Zhang, 2004).

The Sketch data structure is used for dimensionality reduction. It is based on random aggregation of traffic attribute (e.g. number of packets) in different hash tables. A Sketch  $S$  is a 2D array of  $H \times K$  cell (as shown in Figure 4-1), where  $K$  is the size of the hash table, and  $H$  is the number of mutual independent hash functions (universal hash functions). Each item is identified by a key  $Kn$  and associated with a reward value  $Vn$ . For each new arriving item  $(Kn, Vn)$ , the associated value will be added to the cell  $S[i][j]$ , where  $i$  is an index used to represent the hash function associated with  $i$ th hash table ( $0 \leq i \leq d - 1$ ), and  $j$  is the hash value ( $j = H_i(Kn)$ ) of the key by the  $i$ th hash function.

Data items, whose keys are hashed to the same value, will be aggregated in the same cell in the hash table, and their values will be added up to existing counter in the hash table. Each hash table (or each row) is used to derive probability distribution as the ratio of the counter in each cell to the sum of whole cell in the line.

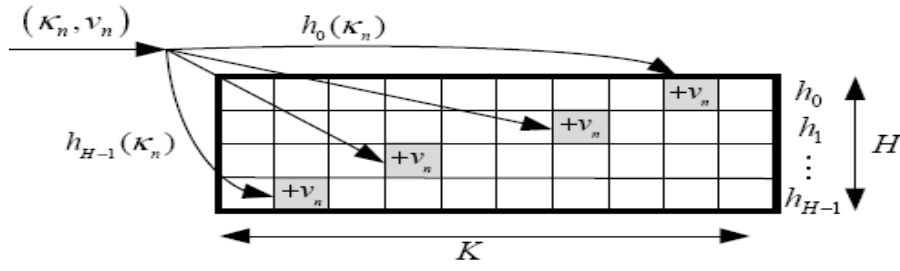


Figure 4-1 : Sketch Data structure

Let the Mobile Agents be the input stream to the detection system. Time is divided into discrete intervals and each interval is of a fixed length  $\Delta t$ . In each detection cycle, the data being analyzed is organised into a training set and a test set. The training period consists of  $n$  consecutive time intervals and the test period is the interval right after.

Assuming, there is no attack in the initial training set. An  $H \times K$  sketch from the set for the Mobile Agents' messages is built. The Mobile Agents addresses of the senders are used as the hash keys. For messages hashed to the same position in a sketch table, the total number of them will be the value of that entry.

For the  $H \times K$  table of the training set, a probability distribution  $P1$  from the first row of it is obtained. In order to achieve this, suppose the values of the  $K$  entries are  $(n_1, n_2, \dots, n_K)$ , These values are added up and get a number  $N$ . Thus, the distribution  $P1$  can be defined as:

$$P1 = \left( \frac{n_1}{N}, \frac{n_2}{N}, \dots, \frac{n_K}{N} \right) \quad (1)$$

Similarly, for the  $H \times K$  table of the test set, a distribution  $Q1$  from its first row is obtained. Suppose the values of the  $K$  entries are  $(m_1, m_2, \dots, m_K)$ . These values are added up and get another number  $M$ . Thus, the distribution  $Q1$  can be defined as:

$$Q1 = \left( \frac{m_1}{M}, \frac{m_2}{M}, \dots, \frac{m_K}{M} \right) \quad (2)$$

The derived probability distributions ( $K$  probability set, one per line) are used as inputs for divergence measures; they will be described in the following sections.

### 4.2.2. Threshold

In order to differentiate network anomalies from normal behaviour on the mobile agent, the use of a detection threshold for Power Divergence ( $PD$ ) is mandatory.

Instead of using a static threshold, the Exponential Weighted Moving Average (EWMA) is adapted.

Let  $PD(n)$  be the current value of the Power Divergence.  $PD(n)$  and  $PD(n+1)$  are respectively the current and next exponentially weighted average estimates of Power Divergence. Let  $\alpha(n)$  be the deviation between the current Power measure  $PD(n)$  and the average measure  $\bar{PD}(n)$ . The exponentially weighted average of  $\sigma(n)$  is denoted by  $\bar{\sigma}(n)$ .  $\bar{\sigma}$  measures how much  $\bar{PD}(n+1)$  deviates from  $PD(n)$ .

The estimated threshold  $h(n+1)$  is then given as follows:

$$\bar{PD}(n+1) = (1 - \alpha) \cdot \bar{PD}(n) + \alpha \cdot PD(n) \quad (3)$$

$$\alpha(n) = \bar{PD}(n) - PD(n) \quad (4)$$

$$\bar{\sigma}(n+1) = (1 - \alpha) \cdot \bar{\sigma}(n) + \alpha \cdot \sigma(n) \quad (5)$$

$$h(n+1) = \lambda \cdot \bar{PD}(n+1) + \mu \cdot \bar{\sigma}(n+1) \quad (6)$$

The principle of using EWMA here is to forecast future values based on current values. Given  $\bar{PD}(n+1)$  and  $\bar{\sigma}(n+1)$ , the estimated Threshold  $h(n+1)$  can then be computed. In order to avoid false alarms, the threshold should be greater than divergence measure in normal situation.

Two parameters  $\lambda$  and  $\mu$  then come into play to set a safe margin for the threshold. The parameters  $\alpha$ ,  $\beta$ ,  $\lambda$  and  $\mu$  are all tunable parameters in the model. Proper values will be set for them in the experiments, in order to improve the detection accuracy.



### 4.2.3. Mathematical Algorithms

These measures are used to detect the DDoS attacks based on the deviation of traffic distribution. In fact, the idea is to compare the prior distribution derived from Sketch counters in previous time slot, with the currently obtained distribution. One can use this change to detect flooding attack, because the counter of one cell will increase significantly with the number of sent requests, and the probability distribution deviates at the start and stop instances of the flooding attack.

#### 4.2.3.1. Chi-Square

$x^2$  divergence is used to measure distance between two discrete probability distributions ( $P$  and  $Q$ ). For 2 probability sets  $P = (P_1, P_2, P_3 \dots P_n)$  and  $Q = (Q_1, Q_2, Q_3 \dots Q_n)$ , with  $P_i \geq 0$ ,  $Q_i \geq 0$  &  $\sum_{i=1}^n p_i = \sum_{i=1}^n q_i = 1$

The Pearson  $x^2$  divergence between  $P$  and  $Q$  is given by:

$$x^2 (P||Q) = \sum_{i=1}^n \frac{(P_i - Q_i)^2}{Q_i} \quad (7)$$

where  $Q$  is the estimated probability distribution and  $P$  is the measured probability.

Distribution, and  $x^2 (P ||Q)$  is the distance between distributions  $P$  and  $Q$ .

For hypothesis testing, such as  $H_0$  (normal traffic hypothesis) and  $H_1$  (traffic with anomalies),  $x^2$  values can run from zero into infinity.  $x^2$  will be zero if  $P$  and  $Q$  are identical ( $P_i = Q_i$ ) under hypothesis  $H_0$ , and  $x^2$  increases as the distributions become dissimilar, and eventually so high (infinity) when the two distributions are independent ( $P \neq Q$ ) under hypothesis  $H_1$ . It is important to note that  $x^2$  divergence is non-negative and the division  $0/0$  is treated as 0, and the division by zero is replaced by a very small value  $\epsilon$ .

The  $x^2$  divergence between 2 probability distributions  $P$  and  $Q$  must be near zero under normal traffic, with a large deviation (one spike) when distributions change occurs.  $x^2$  is asymmetric ( $x^2 (P ||Q) \neq x^2 (Q ||P)$ ), and its symmetric version raises two spikes. One spike at the beginning and the second at the end of the attack.

$$x^2 (P||Q) + x^2 (Q||P) = \sum_{i=1}^n \frac{(P_i - Q_i)^2 + (P_i + Q_i)}{P_i * Q_i} \quad (8)$$

Pearson chi-square divergence (asymmetric) is used to detect anomaly through the detection of deviations from normal traffic profile. Then, the input time series is modified to constrain  $x^2$  to raise alarms (spikes) for the whole duration of attack. Leorato and Broniatowski (M. Broniatowski and S. Leorato, 2006) proved that  $x^2$  divergence behaves better than all classical divergences (Hellinger distance, Kullback-Leibler, Likelihood, etc.

#### 4.2.3.2. Hellinger Distance

Hellinger Distance (*HD*) is used to measure the divergence between two sets of probability values.

For two discrete probability distributions  $P = (P_1, P_2, P_3 \dots P(k - 1))$  and  $Q = (Q_1, Q_2, Q_3 \dots Q(k - 1))$ , with  $P_i \geq 0, Q_i \geq 0$  and

$$\sum_{i=0}^{k-1} p_i = \sum_{i=0}^{k-1} q_i = 1$$

The HD between current distribution  $P$  and prior distribution  $Q$  is defined as:

$$HD (P, Q) = \frac{1}{2} \sum_{i=0}^{k-1} (\sqrt{p_i} - \sqrt{q_i})^2 \quad (9)$$

where *HD* satisfies the inequality  $0 \leq HD (P, Q) \leq 1$ , and  $HD (P, Q) = 0$  if  $P = Q$ . *HD* is a symmetric distance (e.g.  $HD (P, Q) = HD (Q, P)$ ) and induces two spikes, one at the beginning of change, and the second at the end of the change (Hemant Sengar, Duminda Wijesekera, Sushil Jjodia, June, 2008).

#### 4.2.3.3. Power Divergence

Rathie and Kannappan have defined the Power Divergence and equivalent variants (up to a scale factor  $\beta$ ) of this divergence (P. N. Rathie and P. Kannappan, 1972). The divergence measure is therefore the decision measure that generalizes the Kullback-Leibler measure and Hellinger distance to a broad class of divergence of order  $\beta$ . In fact, the Power Divergence is a measure of distance between two probability measures of order  $\beta$  given as follows:

$$PD (P||Q) = \frac{\sum_{i=1}^W P_i (P_i/Q_i)^{\beta-1} - 1}{\beta (\beta-1)} \quad (10)$$

where  $P$  is the posterior probability distribution and  $Q$  is the prior probability distribution. This divergence presents some interesting special cases.

For  $\beta = 0.5$ , this divergence is  $4 \times HD(P || Q)$ , and for  $\beta = 2$  it is equal to  $0.5 \times x^2(P || Q)$  divergence. Obviously, this power divergence outperforms then the  $x^2$  and HD measures (some results will be provided later in the paper). In fact, by changing the values of  $\beta$ , one can optimize the detection of attacks compared to the  $x^2$  and HD measures.

The experiments work will show numerically that for different values of  $\beta$ , the detection efficiency changes. The optimal value of  $\beta$  then can then be obtained.

$P$  and  $Q$  are derived from the Sketch data structure in two consecutive discrete intervals. Firstly, the shared counters of the sketch are continuously updated from ongoing traffic during a time interval  $T$ . At the end of each interval, the probability  $P_{i,j}$  is calculated as the ratio of each counter to the sum of the whole counters in one hash table:

$$P_{i,j} = \frac{S^{[i][j]}}{\sum_{j=1}^w S^{[i][j]}} \quad (11)$$

The probability distributions  $d$  in each interval ( $P_1 \dots P_d$ ) is obtained, where  $P_i$  is the distribution ( $P_1 \dots P_d$ ) resulted from the  $i^{th}$  hash table.  $Q_i$  is the probability distributions resulted from previous interval. The probability distributions of  $Q_i$  is calculated in the same manner as  $P_i$  (Eq. 2).

When the Power Divergence is larger than dynamically updated threshold, an alarm will be raised. However, Power Divergence induces only two spikes (at the start and at the end of attack). As the goal is to continuously raise alarms for whole duration of the attack, the distribution  $Q_i$  will stop sliding by keeping its value until the end of the attack. However, with the variations of normal traffic and the similarity of DDoS attacks with flash crowd, Flooding attacks is supposed to span for many intervals, in contrast to flash crowd and normal variation. Thus, the false alarms is reduced. Therefore, an alarm will be triggered only if the deviation lasts more than  $\Delta$  intervals.

### 4.3. Proposed Approach

This section explains, in detail, the contribution on how to integrate Sketch technique and divergence measures (Hellinger Distance, Power Divergence and Chi-square) in favour of detecting anomalies in Mobile Agent networks (Figure. 4-2).

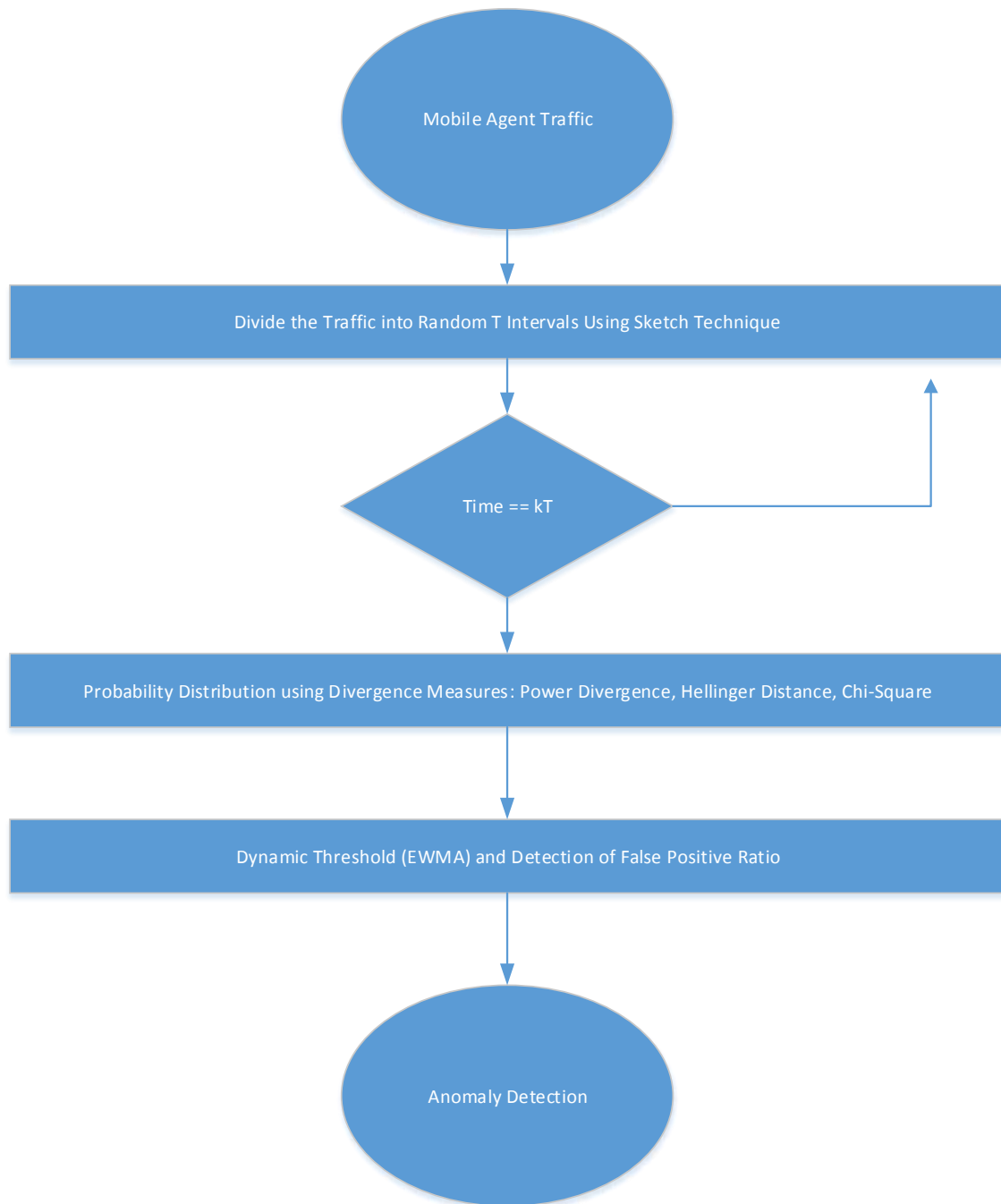


Figure 4-2 : Proposed Algorithm for Network Anomaly Detection.

1. The detection system begins first by recording the number of monitored points (e.g. #packets, #SYN, #flows, etc.) in the Sketch for each discrete time interval  $T$ . Random aggregation of traffic flows in Sketch is the first step of the processing, followed by time instances forecasting with divergence measures.
2. During each interval, the destination IP address (DIP) for each packet containing a SYN segment, is hashed by  $H$  hash functions. The resulting hash value by the  $i^{th}$  function ( $J = H_i(\text{DIP})$ ) is used as index of the associated counter  $S_{i,j}$  with DIP. Each arriving SYN segment increments the associated counter.

The analysis will be focused on TCP SYN flooding by counting the number of SYN.

At the end of each epoch  $T$ , the probability distributions is derived via the Sketch technique. At first, the sum of the counter in each line is developed, and the probability  $P_{i,j}$  in each cell is calculated as the ratio of each counter to the total number of SYN:

$$P_{i,j} = \frac{S_{i,j}.Counter}{\sum_{j=0}^{k-1} S_{i,j}.Counter} \quad (12)$$

Each cell  $S_{i,j}$  becomes a data structure that contains current counter, current and previous probabilities. Therefore, each line (or hash table) provides two probability distributions: the first one is from previous interval and used as reference distribution  $Q_i$ . The second one is from current interval  $P_i$ , and used to measure the divergence from the reference distribution, in order to detect anomalies. Divergence measures between the current ( $P_i$ ) and reference probability ( $Q_i$ ) distributions is calculated for each line in the Sketch, at the end of each time interval (i.e. at  $n.T$ ). During malicious activities, the divergence measure  $D(P_i || Q_i)$  produces spikes, and when more than  $L$  ( $L < H$ ) divergences resulted from different hash tables exceed a dynamic threshold, an alarm is raised.

To detect deviations in the time instances resulted from divergence measures, a subsequent time instances containing the values of  $D(P_i || Q_i)$  without spikes is derived.

3. In this last time instances (without large values), a dynamic bound of  $\mu_i + \alpha\sigma_i$  is defined. Significant deviations are larger than the dynamic bound:

$$D(P_i||Q_i) > \mu_i + \alpha\sigma_i \quad (13)$$

where  $D(P_i || Q_i)$  is the divergence measure in the time interval  $n.T$  for the  $i^{th}$  line in the Sketch, and  $\mu_i$  &  $\alpha\sigma_i$  are the mean and the standard deviation respectively of smoothed time instances that do not contain spikes ( $D^{\wedge}(P_i || Q_i)$ ).  $\mu_i$  and  $\sigma_i$  are updated dynamically using the Weighted Average (EWMA):

$$\mu_i = \beta\mu(i-1) + (1-\beta)D^{\wedge}(P_i||Q_i) \quad (14)$$

$$\sigma_i^2 = \beta\sigma(i-1)^2 + (1-\beta)(D^{\wedge}(P_i||Q_i) - \mu_i)^2 \quad (15)$$

The threshold is updated dynamically with the value of  $\mu_i$  and  $\sigma_i$  as shown in above equations.  $\alpha$  is a parameter used for calibrating the sensitivity of the detection algorithm to variations. It is also used to reduce the false alarm rate. Under normal traffic, divergence  $D(P_i || Q_i)$  falls inside the bound of  $\mu_i + 2\sigma_i$ . When  $D(P_i || Q_i)$  exceeds the dynamically updated threshold over  $L$  lines, an alarm is triggered.

The solution for the detection of attacks is developed using C++ Code. It is located in Appendix F.

## 4.4. Conclusion

This chapter has introduced the techniques that will be used for the new proposed approach for the detection of attacks and anomalies in the mobile agent world.

This model is based on sketch technique (Hash functions), divergence measure (Hellinger Distance, Chi-square and Power Divergence) and dynamic threshold (Jacobson Fast algorithm for RTT (Round - Trip Time)). The last section of this chapter has described in detail, the process and modelling of the proposed solution.

The strength of this model, in comparison of the existing solutions, is the accuracy of the detection of anomalies in terms of false alarm. This will be shown and verified in Chapter 7. The above described divergence measures will be tested via several scenarios in order to know which one can provide better performance in detection of anomalies, with less false alarm ratio. These evaluations will be done via the receiver operating characteristic (ROC).



# Chapter 5 : Protection Proposed Model

---

## 5.1. Introduction

The previous chapter has presented a new method for the detection of attacks against mobile agents. This chapter will focus on and develop a new way of protection against malicious host in mobile agents' network. This new approach will be delineated in detail in the following sections.

The proposed countermeasure is based on trust. A mobile agent can host either blind folded, based on policy enforcement, or based on control and punishment.

A Mobile Agent is blind folded, which "simply need to trust its entertaining host". The host is free to do whatever it wants while giving services to the Mobile Agent. But it is trusted that it neither has malicious behaviour nor does it collaborate with other hostile hosts that perform some bad actions on the agent.

Policy enforcement is another trust. In this case, the Mobile Agent and the host have a prior contractual relationship in the form of policy. Both parties need to sign for their rights and obligations.

The last trust is control and punishment. There is no needed policy to be signed between the two parties as well as there is no contract signed. The trust assumes that hosts are not by nature malicious and give them a chance to behave accordingly. But it still uses control mechanism to punish the host if found guilty of misbehaviours.

The proposed solution is a combination of *policy enforcement* and *control and punishment*.

The below section outlines the guidelines used to develop the countermeasure.



## 5.2. Design Guidelines

The following points are used as guideline when the proposition is being developed:

- Convenience to the home of the agent: Once the home of the agent sends the MA, he does not have to worry anymore about whereabouts of the agent. All that he needs is the agent to return with its result; nothing more.
- Abstraction of the modification: The home of the agent does not have to feel the difference between the proposed and the normal mobile agent operation and computation
- No pre-negotiation with hosts: The mobile agent should be able to freely roam inside the agent space, without a prior agreement between the set of hosts to be visited and the user of the mobile agent.
- Ease of access of information gathered: Once the mobile agent returns to its home, the user should be able to retrieve the information gathered from the data carried back by the mobile agent, without any need to contact the hosts involved.

## 5.3. Components of the Proposed Countermeasure Model

This section will look at the main components of the proposed countermeasure approach and how should the components interact according to the security model.

Figure 5-2 shows the overall view of the model; it shows that the countermeasure constitutes various components at various degree of multiplicity.

Below is the component of the proposed approach:

- Home of the Mobile Agent
- Mobile Agent (MA)
- Trusted Node
- Active Storage Element
- Home

### 5.3.1. Home of the Mobile Agent (Home)

It is the computer running mobile agent platform and has sent the mobile agent to carry out a task on its behalf. It can also be defined as *a computer running a mobile agent based distributed application*. The application as a part of its mission packs a task into the mobile agent and sends it to the agent space. The mobile agent after completing its task will eventually return to the home, carrying the result.

### 5.3.2. Mobile Agent

As defined throughout this thesis, it is a program that migrates from one node to another node in a computer network to accomplish a task.

### 5.3.3. Trusted Node

It is similar in composition to the home of the mobile agent, but differs in the function it provides. It is there to provide support and service to the mobile agent while it is in the agent space.

### 5.3.4. Active Storage Element

It is a temporary storage element that is created by each mobile agent, at the trusted server, that is sent to visit nodes in the agent space. It actively participates in the process of temporary information storage and handing over of all the information to the mobile agent.

### 5.3.5. Host

It is a computer in the agent space running mobile agent platform and entertains any visiting mobile agent which would like to gather information from it. This component is at the center of the controversy, which could be hostile. The host provides all the necessary resources for the agent to execute there.

## 5.4. Proposed Countermeasure Model

Mobile Agents are subjected to any type of attacks because they are a 'lonely figure', once sent to the agent space. Hence, the proposition modifies the computing model of the mobile computation in order to address hostile host threats.

The computational model of the original mobile agent system consists of home of the mobile agent, computers in the agent space (hosts) and the mobile agent itself. In this model, the mobile agent visits nodes (hosts) that are in the agent space, dictated by its list of destinations, gets the data, carries it around as it visits other nodes and will eventually return to its sending node with the result. This is an overall view of the computing model of the original setup as shown in Figure 5-1.

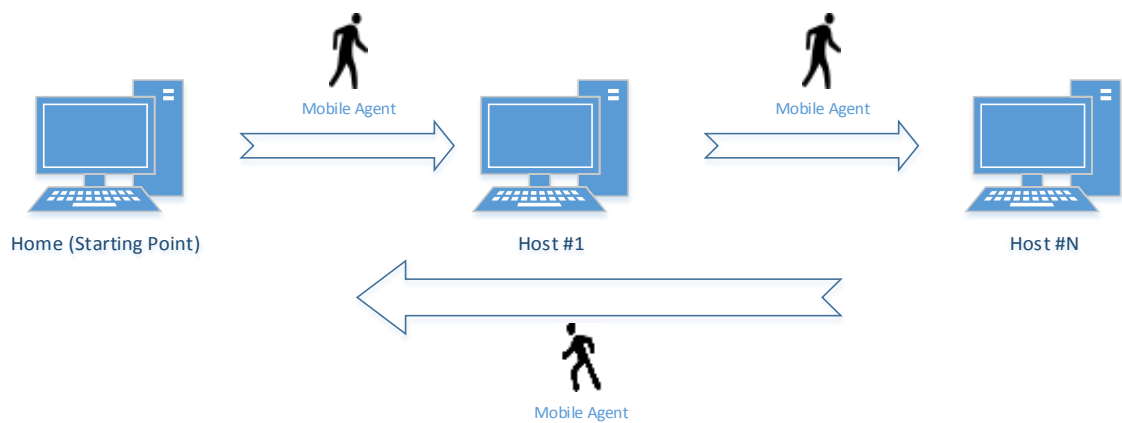


Figure 5-1: Existing Mobile Agent Computing Model

Figure 5-2 shows an overview of the mobile computing model. It is mandatory that the home of the Mobile Agent has a public-private key pair at its disposal, the public key is published to the world so that the Mobile Agent could retrieve this key while it is visiting hosts. These keys are used by the security model to protect the confidentiality and the integrity of parts of the Mobile Agent.

The model modifies the way by which the mobile computation is done.

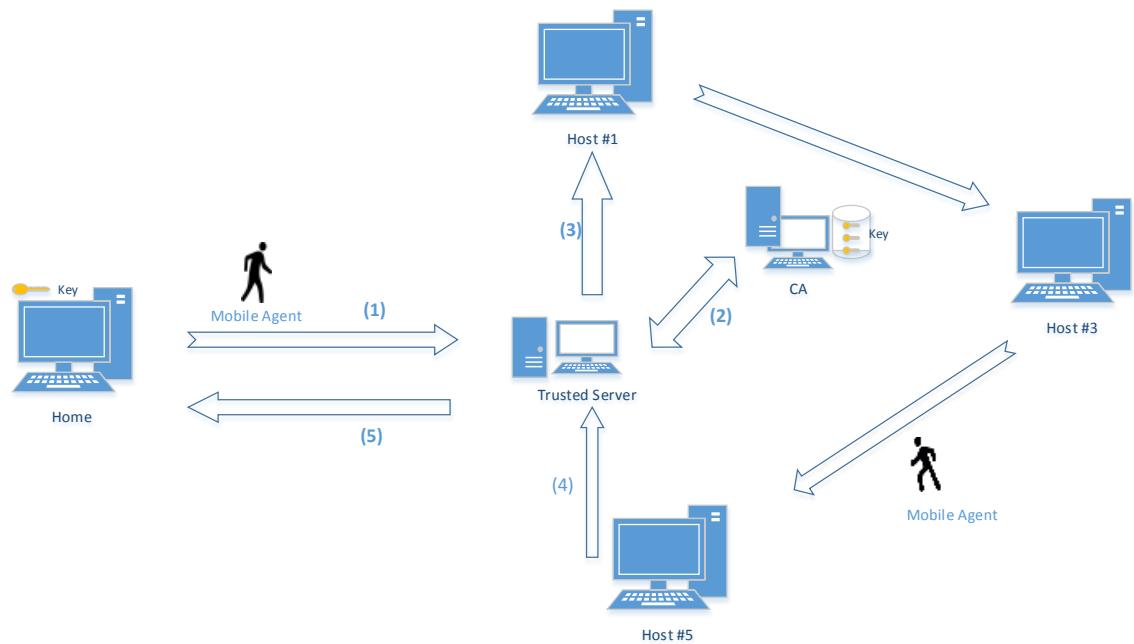


Figure 5-2 : Proposed new model

The process is described as below:

- (1) The arrows dictate that the Mobile Agent first goes to the trusted node.
- (2) Creates a temporary storage element called active storage element (ASE).
- (3) Then moves to the first host to be visited. It goes there, sends the information it has retrieved from the corresponding host to be stored temporarily at ASE. The trusted node accepts the information and stores it. Each Mobile Agent that has a trust relationship with this node does the same, creates its own ASE at the trusted node and uses it to store the partial information it retrieves from each hosts.
- (4) In the end, the Mobile Agent returns to the trusted node and asks the corresponding ASE to hand it over the results it has been accumulating so far.
- (5) It carries the result back to its home as if it has been doing the job also.

It is assumed that the agent space is divided into regions; within each region, a node called *trusted server* is setup. These servers provide various services to the Mobile Agent while the agent is in the agent space. The Mobile Agent supported by these trusted nodes should be able to avert some of the evil acts from hostile hosts. The division of the agent space into regions is analogous to the cells in the mobile communication systems.

Nowadays, it is common to introduce trusted server setup in a network handled by a third party. Plenty of servers deployed in the internet world uses a trusted server like Web servers, mail servers and Domain Name Services (DNS) servers. Trusted servers do not have the right to modify the Mobile Agent's content, as web servers do not modify the web page they host. But here the security model provides further protection to the Mobile Agent content at the trusted server. It is such a similar concept that the proposition wants to exploit. The nodes and the trusted servers could be set up, in a similar style as nodes of root Web servers, by the Mobile Agent user community.

## 5.5. Security Model

A security model defines how components of the system (Home, Trusted Nodes and Active Storage Element) should interact with each other as well as, what the necessary tasks need to be performed are at each level in order to secure the network from any malicious actions.

The security model is free to take or alter any action on the Mobile Agent. While the Mobile Agent is travelling between its home and trusted nodes, the usual composition is deemed. But when the agent is visiting different nodes, it is assumed to be composed of only the two out of the three components that is usually associated with code and state, to give hostile hosts no chance of disclosure of information collected from previous hosts.

Another function of the security model is to develop a mechanism that lets the user of the Mobile Agent to digitally sign the list of destinations it wants the Mobile Agent to visit. After creating the list of destination, it digitally signs the destination object using its private key, then the destination object is passed down to the Mobile Agent. The Mobile Agent, upon its arrival at each host in the network, verifies that it has a valid copy of the destination object before putting that object into use. By this, the Mobile Agent avoids the possibility that it would be directed to visit other hosts by altering the list of paths it has carried from its home, as any malicious host could not forge the digitally signed destination object.

The security model can be found in every step in the agent trip in the network. It is assumed that, the home node has a public-private key pair (HPubK-HPrvK).The public key could be retrieved by the hosts from relevant authorities.

### 5.5.1. At Home

The user of the Multi-Agents System (MAS) specifies the address of the list of hosts that will visit, using the Agent Based Application (ABA).

The ABA takes the list and forms a destination object. The destination object contains the list of hosts to be visited, the address of the trusted server (TS) and the home. The ABA then digitally signs the destination object and passes it to the MAS. The MAS accepts the signed object. By using HPubK, the MAS verifies that it has the right un-signed destination object, from which the address of the next node to be visited is determined and dispatches itself to that node; as pointed out in the previous section, it goes first to the Trusted Node, as given in Figure 5-3.

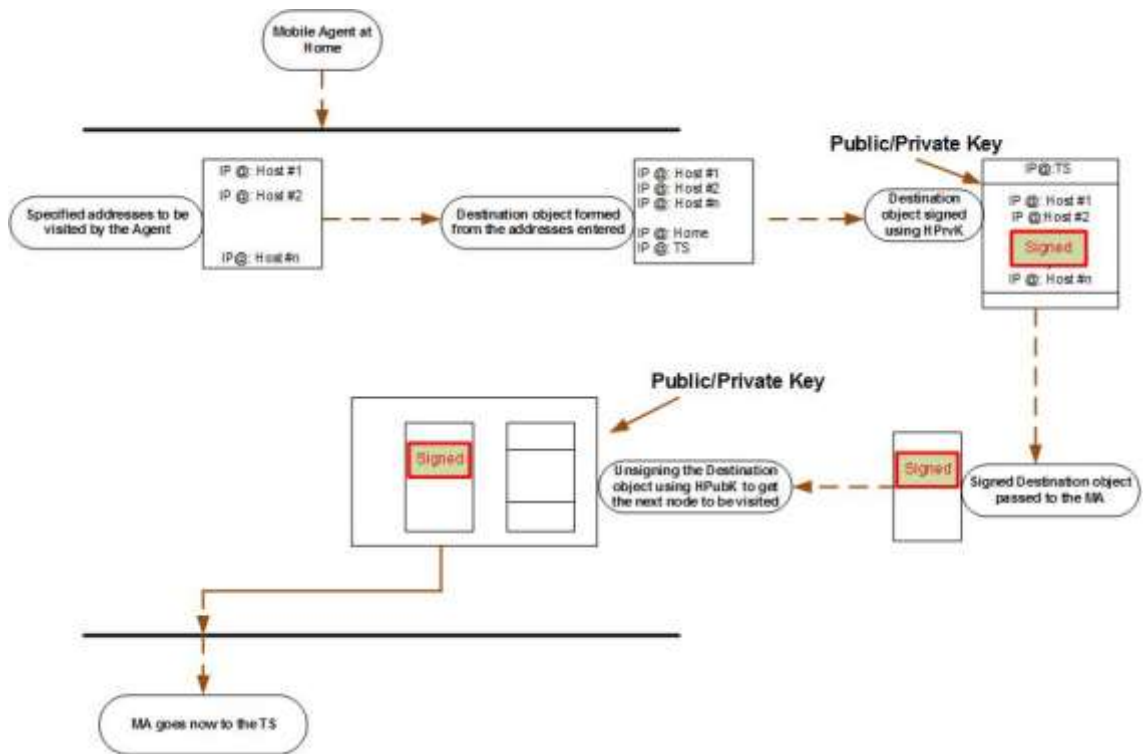


Figure 5-3: Proposed Model at Home

### 5.5.2. At Trusted Server

The MAS arrives at the TS. It creates its own Active Storage Element (ASE). The MAS passes down the necessary information to the ASE, so it can communicate with it. The MAS retrieves the public key of the home, HPubK. Using this key, the MAS un-signs the digitally signed- destination object and determines the next node to be visited. In this case, it is the first host in the list and the MA Dispatches itself to that node, as delineated in Figure. 5-4.

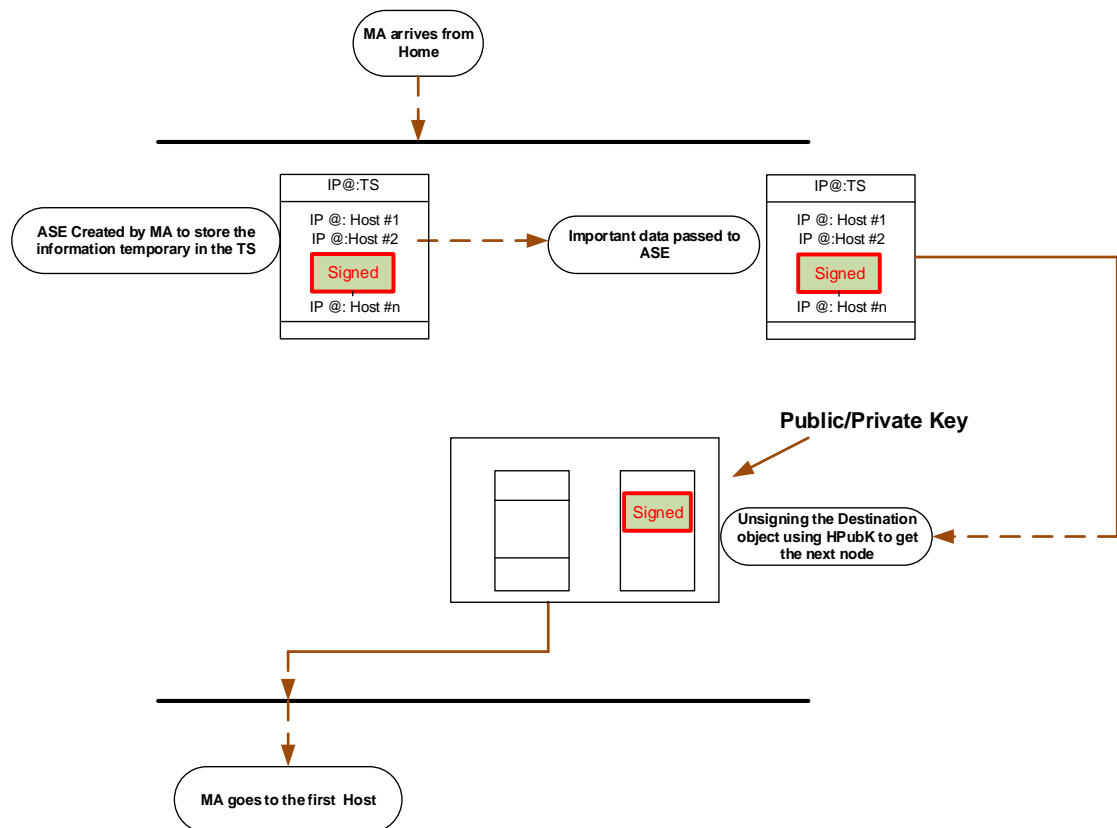


Figure 5-4: Proposed Model at Trusted Server

### 5.5.3. At ith Host

- The MA arrives at the ith host, Host<sub>i</sub>.
- It generates a random symmetric key, SymK<sub>i</sub>.
- The MA retrieves the public key of the home, HPubK.
- Asks the host about the information it wants, Info<sub>i</sub>.
- Encrypts the information using the symmetric, SymK<sub>i</sub>(Info<sub>i</sub>).

- Encrypts the randomly generated symmetric key using the public key,  $HPubK(SymK_i)$ .
- Sends both of these information,  $HPubK(SymK_i)$  and  $SymK_i(Info_i)$ , to its ASE at the TS to be stored temporarily.
- The MA un-signs its destination object, looks the address of the next node to be visited and dispatches itself to that node. At Trusted Server, after reach of the trusted server. Please see figure 5-5.

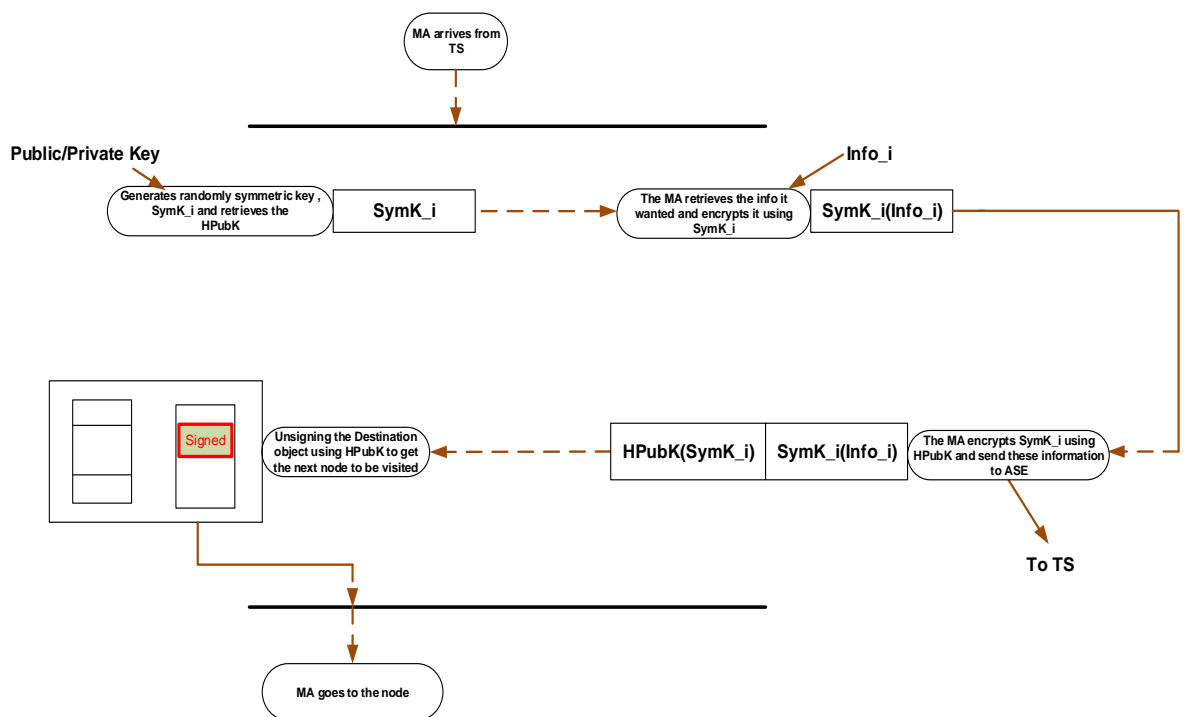


Figure 5-5: Security Model at the ith Host.

#### 5.5.4. At Trusted Server

If the next node is another host; it does the same task as indicated above. Or else, if it is a trusted server, the following set of actions follows.

The MAS arrives at the TS. It asks the corresponding ASE to hand it the information it has been accumulating (a pair of  $HPubK(SymK_i)$  and  $SymK_i(Info_i)$  retrieved from each host), and takes these pieces of information. After un-signing its destination object, it looks



for the address of the next node to be visited. In this case for sure, it is the home node and the MAS dispatches itself to its home as shown in Figure 5-6.

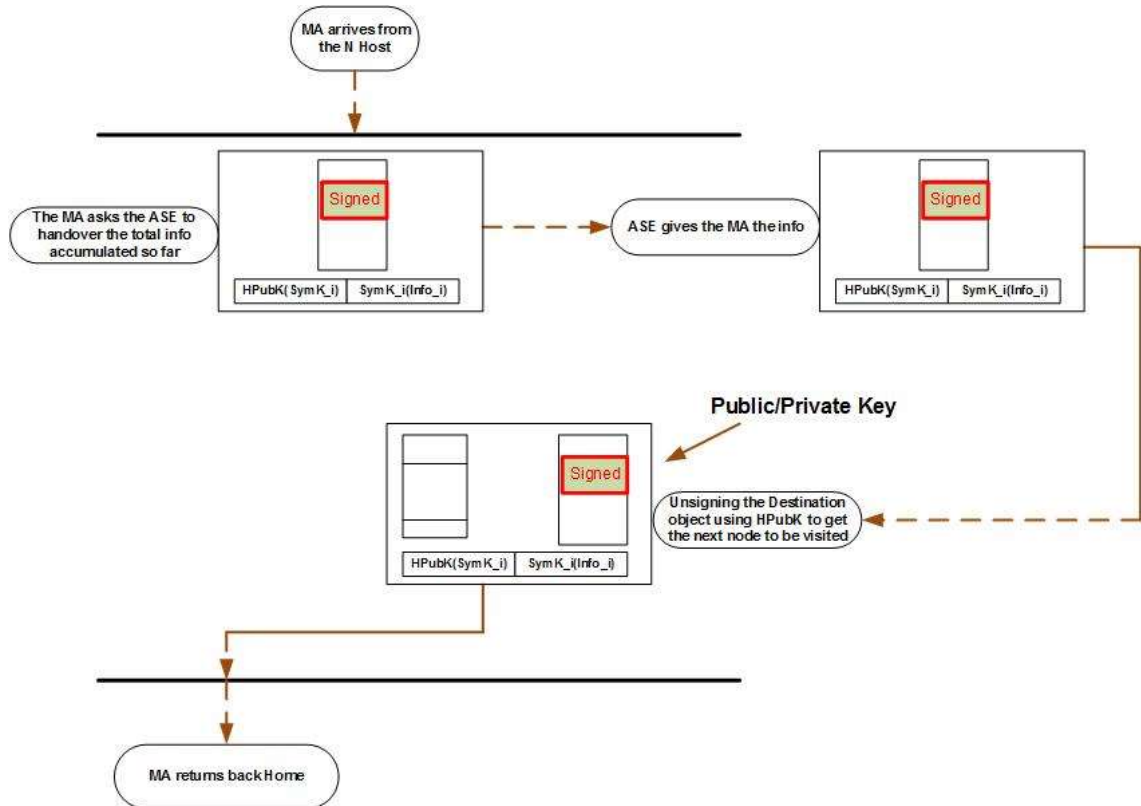


Figure 5-6: Proposed Model at Trusted Server, after the Reach of the Nth Host

### 5.5.5. At Home

The MAS arrives back at home after completing its mission. The MAS contains a pair of encrypted information. The MAS hands the overall information to the ABA. For each pair of encrypted information retrieved from each host, the ABA does the following as in figure 5-7:

- First, using its private key (HPrvK) to decrypt the encrypted symmetric key, HPubK (SymK\_i).
- Second, using the decrypted symmetric key, SymK\_i, it decrypts the information which is encrypted using the same key, SymK\_i (Info\_i).
- The ABA does the same process for each pair of information retrieved from every host the agent goes to collect information. At last, the ABA displays the result to the user, Info\_i.

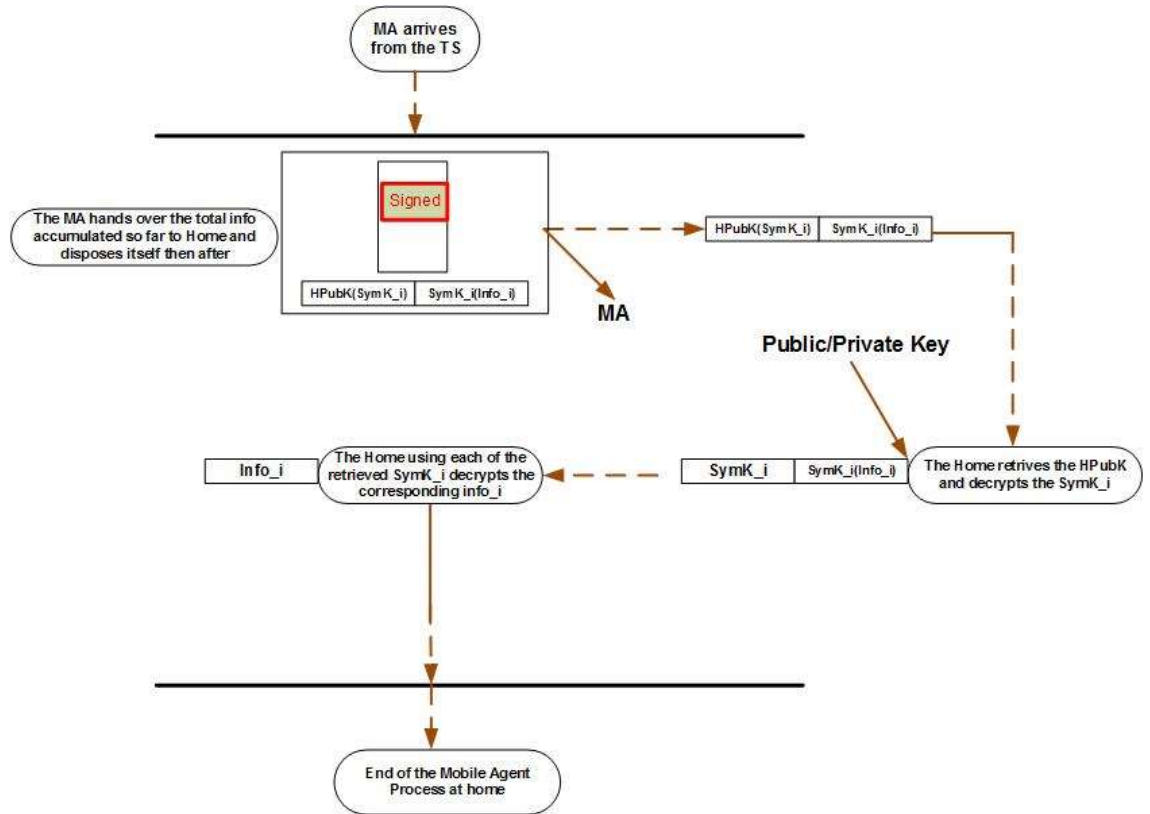


Figure 5-7: Security Model Back at Home.

### 5.5.6. Security Model Summary for N Hosts

You can find below a summary for the security model:

- N hosts addresses digitally signed by the home node.
- One ASE created at TS.
- N symmetric random keys generated at each host.
- N information retrieved will be encrypted by the corresponding N symmetric keys.
- The N symmetric keys will be encrypted by the public key (RSA) of the home.
- The encrypted N information and encrypted keys stored at the ASE.
- Decryption at home node and displaying the plain text result to the user.

## Evaluation of the Approach

This section will deal with the malicious threats (presented in Chapter 3), using the various functionality of the countermeasure, presented above and how this is done.

The work will focus on two types of threats: Alteration and Eavesdropping

### **Alteration**

The countermeasure addresses alteration of carried threats from malicious hosts. Unlike the previous mobile computation model, the mobile agent does not carry the result it has gathered from the previous hosts while visiting the next host. It gives no chance to a malicious host, interested to hack into the information provided by other competing hosts. Furthermore, the security model through usage of randomly generated symmetric keys protects the confidentiality of the information retrieved while the information is being sent to the TS for temporary storage.

The countermeasure through separation of the destination part from the composition of the mobile agent also provides protection of alteration of the destination object. Based on the proposed approach, the user of the mobile agent creates a destination object, which consists of the list of all hosts to be visited in the agent space as well as the home and TS addresses; then the user digitally signs this object using its private key. This makes it a difficult task for a malicious host, which would like to misdirect the mobile agent by supplying a wrong destination object, as it cannot counterfeit the digitally signed destination objects. Any attempt on replacing the destination object while the mobile agent is in the way of the agent space will be detected by the mobile agent itself, as it always checks the validity of the destination object it carries, on its arrival at each and every host, before using it to select the next host to be visited.

### **Eavesdropping**

The countermeasure addresses the threat of eavesdropping of the collected information, using the security model to randomly generate a symmetric key that will be used to encrypt the collected information. The sending of information in encrypted form protects external malicious eavesdropper from learning anything substantial while the information is sent for temporary storage. The computing model also protects internal eavesdropping

to be an impossible task. A malicious host which would like to look into the information collected so far, simply cannot do so, due to the adapted new computing model, as the information is not available there.

The protection model against malicious hosts is developed using JAVA Code. It can be found in Appendix E.

## 5.6. Conclusion

This chapter has presented a new approach of protection against malicious hosts in mobile network world. This new approach is based on trusted servers. Its strength is that it relies on the existing protection solutions. The evaluation of this approach will be done, based on Eavesdropping and Alteration. Chapter 7 will show in detail, the results of this new approach.



# Chapter 6 : Experimental Work

---

## 6.1. Introduction

The previous chapters described, in details, the new approaches for detection of SYN flooding attacks using the sketch techniques, and the protection method against malicious hosts based on trusted servers. The robustness, performance and strength of these new proposed models of mobile agents: both protection and detection will be tested in this chapter. (Appendix C will delineate in detail the experiment environment.

## 6.2. Protection Model Results

This section presents the result of the implementation of the security policy as well as the result of the performance comparison between different types of mobile agents.

### 6.2.1. Developed Classes

To realize the proposed model in the Chapter 4, a number of classes are developed. Some of them form the basic components of the prototype: like statA, mobile and Agentapp, while other classes are developed to implement the Graphical User Interface (GUI).

#### 6.2.1.1. *Destn*

This class encapsulates the addresses of the nodes to be visited by the mobile agent, including the addresses of its home and its trusted server. In this class, the separation of the other parts of the mobile agent from its list of destinations is implemented. The object of this class is constructed based on the user's preference and order of list of hosts it wants the mobile agent to visit. The functions are listed in figure 6-1

Destn
<pre> Destn(ArrayList, ArrayList) getHosts(): ArrayList getTS(): ArrayList getHome(): URL isAtHome(Aglet): Boolean isAtTS(Aglet): Boolean goToNext(Aglet) goBackHome(Aglet) getHomeURL(): URL getAgletURL(Aglet): URL </pre>

Figure 6-1: Functions of the Destn Class

### 6.2.1.2. ProxyH

This class holds the proxy of the main interacting parties of the countermeasure: Mobile Agent, ASE & Agent based application. As mentioned in Destn Class, Aglets encapsulates the information in a message, then uses the proxy of the other party to send the message object. By this action, the ProxyH serves as the container of the proxy of the main interacting components of the system. It provides a number of methods to keep its instance variables valid.

In the current implementation of aglet, the proxy of a moving agent obtained while it is at one host becomes invalid as it moves from one host to another, and hence needs to be constantly updated as the mobile agent migrates. But the proxy of a stationary agent remains valid throughout its life time. The functions of this class are listed in figure 6.2.

ProxyH
<pre> proxyH (Aglet, Aglet, Aglet) proxyH (Aglet, Aglet) proxyH (Aglet) proxyH (AgletProxy, AgletProxy, AgletProxy) proxyH (AgletProxy, AgletProxy) proxyH (AgletProxy) updateAProxy(Aglet) updateSProxy(Aglet) updateMProxy(Aglet) updateAProxy(AgletProxy) updateSProxy(AgletProxy) updateMProxy(AgletProxy) getSProxy(): AgletProxy getAProxy(): AgletProxy getMProxy(): AgletProxy </pre>

Figure 6-2: Functions of the ProxyH

### 6.2.1.3. CipherCls

This class provides cryptographic services to the agent- based application and to the mobile agent using Legion of Bouncy Castle provider implemented algorithms. It provides various methods to sign object, verify a signed object, encrypt and decrypt.

The functions of the class are listed in figure 6-3.

CipherCls
getPrivateKey(String, char[] , String): PrivateKey
getPublicKey(String , String ): PublicKey
sealObject(Serializable, PublicKey): SealedObject
unsealObject(SealedObject, PrivateKey): Object
signObject(Serializable, PrivateKey): SignedObject
getSignedObject(SignedObject, PublicKey): Object
generateSessionKey(): SecretKey
sealSecretKey(SecretKey , PublicKey): SealedObject
unsealSecretKey(SealedObject, PrivateKey): SecretKey

Figure 6-3: Functions of the CipherCls

## 6.2.2. Components of the Prototype Application

This section will describe in detail, how the components (AgentApp, mobileA, statA) are implemented, in order to execute the proposed protection approach.

### 6.2.2.1. statA

This class creates the temporary storage element (ASE) concept.

A statA is created by each MA inside a trusted server on its first visit there. It is used to temporarily store results that the mobile agent retrieves from its different hosts in the agent space. The information in the statA is encrypted, so even it doesn't know what it is storing about.

Then, the statA transfers the results it has created, to the corresponding mobile agent at the end of the mobile agent's missions on its way back to its home. After that, it disposes itself to handover the resources it has been using, back to the trusted servers.

An example of the code can be found below in figure 6-4.

```
public class statA extends Aglet
public ArrayList statRes;
public ArrayList statKeys;
public proxyH prxy;
```

Figure 6-4: Class statA

- The instance variable statRes stores temporarily, the information the agent sends, while visiting different nodes in the agent space.
- statKeys instance variable holds a symmetric key that the mobile agent generates randomly, as the MA visits various hosts in the agent space. This key is used to protect both the confidentiality and integrity of the partial information as it is sent to the ASE.
- The instance variable prxy is of type ProxyH as shown in figure 6-5. It holds the proxy of the three interacting parties in the model (the mobile agent, stationary agent and application agent) to facilitate the exchange of information among themselves.

```
public void onCreate (Object init)
{
prxy = (proxyH) init;
statKeys=new ArrayList ();
statRes=new ArrayList ();
prxy.updateSProxy (getProxy ());
}
```

Figure 6-5: Class ProxyH

onCreation(Object init) is a method that every agent that extends Aglet inherits and this method is called only once in the life cycle of the agent.

Its role is described below:

- It initializes instance variables of the agent. It also provides a mechanism to pass an array of objects containing important information from the creator of the agent, down to agent being created.
- The MA that is responsible for the creation of the statA passes to it a ProxyH object, which contains important proxy information. In the code that follows, the statA updates its own proxy instance variable, holds inside the proxy object, using the updateSProxy() method.



- `getProxy()` returns the proxy of the current aglet, i.e. the stationary agent.

The next section of block of code is shown in figure 6-6 given below:

```
public boolean handleMessage (Message msg)
{
    if (msg.sameKind("MobieAway"))
    {
        SealedObject
        sos=(SealedObject)msg.getArg("keys");
        statKeys.add(sos);
        SealedObject
        sor=(SealedObject)msg.getArg("infos");
        statRes.add(sor);
        return true;
    }
    else if (msg.sameKind("MobileBackV"))
    {
        Message newmsg = new Message("MobileH");
        newmsg.setArg("keys", statKeys);
        newmsg.setArg("infos", statRes);
        msg.sendReply(newmsg);
        return true;}return false;}

```

Figure 6-6 Code: `handleMessage (Message msg)`

The above code shows the behaviour of the agent, as it receives various kinds of messages from the mobile agent.

In mobile agents that implement aglet, every message that is sent to the aglet is handled inside `handleMessage (Message msg)`.

- The first if case is satisfied, if a message object of kind "MobileAway" is sent to the stationary agent. In the proposed model, such a kind of message object is sent by the mobile agent, while it is visiting different hosts in the agent space. This same message object encapsulates the information the agent retrieves from the corresponding hosts. The `statA` uses `msg.getArg()` method to retrieve this information, which is a sealed object. This same information, as stated in the proposition, is stored in the `statA` temporarily, using `statRes.add(sor)` statement. Similar techniques are used to retrieve the second argument sent by the mobile agent and stored in the corresponding temporary storage.

- The second if case will be satisfied, when the message object of kind "MobileBackV" is sent to the statA. In the proposed model, such a kind of message object is sent to the statA by the mobileA, when the MA has finished touring various hosts in the agent space and returns to collect all the information it has stored temporarily while it is on the move. The statA hands over all the information it has accumulated, by constructing a message object of kind "MobileH" encapsulating the pieces of information.

### 6.2.2.2. MobileA

This class is used to realize the actual mobile agent, which moves from one node to another, to perform the computation.

A detailed discussion of the code is shown below in figure 6-7.

```
public class mobileA extends Aglet
implements MobilityListener
{
```

Figure 6-7 Class: MobileA

This class by virtue of extending the base class (Aglet) and implementing mobilityListener, becomes an agent that can listen and react to mobility events (a mobile agent) as shown in below figure 6-8.

```
public SignedObject dsDstn;
public String
alias="certvirtual";
public int startTrip=1;
public String agletName="statA";
public Message msgInfo= null;
public proxyH prxy;
```

Figure 6-8 Class: MobilityListener

- The MA uses the instance variable startTrip while it is at the trusted server to take on appropriate action, depending on whether it is there first or after completing its mission. The instance variable prxy of ProxyH as in the statA is used to hold the proxy of agent that it wants to interact with, in this case the proxy of both the statA and the AgentApp.

- The instance variable `agletName` holds the name, specifically the class name, of the stationary agent that this mobile agent will create on the TS, as shown in figure 6-9 given below.

```

public void onCreate(Object init)
{
    Object [] axcptd =(Object [])init;
    prxy =(proxyH) axcptd[0];
    dsDstn=(SignedObject) axcptd[1];
    Destn dstn=null;
    dstn=(Destn) CipherCls.getSignedObject (dsDstn,
    CipherCls.getPublicKey (alias));
    addMobilityListener (this);
    dstn.goToNext (this);
}

```

Figure 6-9 Class: MobilityListener

- As mentioned before, the `onCreation(object init)` method is called once the aglet, `MobileA`, is created. In the prototype, the MA is created by the agent application. It, the application, passes to the MA two objects, one that is a digitally signed destination object and the other, a `ProxyH`.
- Each of these objects is passed as an array of objects and hence need to be retrieved using appropriate index and casted to appropriate type. Next, the MA un-signs the digitally signed destination object, using the utility class's `CipherCls.getSignedObject()`. To un-sign the signed object, it needs the public key of the corresponding private key at its disposal. This public key is obtained using the utility class's `CipherCls.getPublickey(alias)` method. The object retrieved, unsigned, is casted to `Destn` object and it is ready to be used.
- `goToNext()` method is invoked to dispatch the MA to the next node in its list of destinations. Literally, according to the proposition, it goes to the TS and doesn't have any thing to do when it is created.
- The line of code, in Figure 6-10, `addMobilityListener` permits this mobile agent to listen to mobility events and react according to the way 'this' is coded.

```

public void onArrival(MobilityEvent event)
{
    Destn dstn=null;
    dstn=(Destn)CipherCls.getSignedObject(dsDstn,
    CipherCls.getPublicKey(alias));
    if (dstn.isAtTS(this))
    {
        if(startTrip==1)
        {
            prxy.updateSProxy(getAgletContext().createAglet(getC
            ode
            Base(), agletName, prxy));
            startTrip=0;
            dstn.goToNext(this);
        }
        else{
            Message msg= new Message ("MobileBackV");
            msgInfo=(Message)prxy.getSProxy().sendMessage(msg);
            dstn.goBackHome(this);
        }
    }
}

```

Figure 6-10 Code: onArrival(MobilityEvent

The above code shows a section of code inside onArrival (MobilityEvent event) methods. This method is one of the three methods including onDispatching (MobilityEvent event) and onReverting(MobilityEvent event), that agents implementing mobility Listener interface needs to define. It is executed every time a mobile agent arrives at another node. In the proposed model, once the MA arrives to a node, a number of test conditions are utilized.

The section below will describe when each of the test condition is satisfied and a range of tasks that will be performed.

- The first if case if (dstn.isAtTS (this)) tests whether the mobile agent is currently at its TS home or not. This condition is tested using isAtTS(this) method of Destn class. The method compares the URL of the current mobile agent using the argument this with the URL of the TS it holds inside. The condition will be satisfied if the mobile agent is at the TS. But as per the proposition, the mobile agent could be there at two different instances, ie. at the first and the last.

In each case, it performs various kinds of tasks. The if's inside uses the variable startTrip to determine whether the mobile agent is there at first with a set startTrip value, or at the end of its mission with a reset startTrip value.

If the agent is there at first, it performs the following range of tasks:

- Gets the context of the platform it is running at, using the getAgletContext() method of Aglet;
- Creates its temporary data storage (ASE) inside the trusted server, using AgletContext's create aglet method;
- Passes all the necessary information the method needs through(code-based, class name of the agent and others), its arguments to the agent being created using object init argument.
- Updates instance variables corresponding to the proxy of the stationary agent, using the proxy of the stationary object just returned, and
- Uses the dstn go to next method to select the next node next to it.

If the agent is at TS after completing its task, it performs a range of tasks as pointed out below:

- The mobile agent constructs a message object of "MobileBackV", mimicking the fact that it is back at TS;
- Sends the constructed message object, using the proxy of the stationary agent it retrieved from the proxy instance variable's getSProxy();
  - The stationary agent when receives message of such type will send all information it has stored;
  - The information is sent as a message object by the stationary object and stored in a similar message object instance variable;
- The mobile agent invokes the method dstn.goBackHome () to return home ;  
'MISSION ACCOMPLISHED'.

The next if else is shown in below figure 6-11.

```

else if (dstn.isAtHome(this))
{
prxy.getAProxy().sendOnewayMessage(msgInfo);
dispose();
}

```

Figure 6-11 Code: if else for the dstathome

When the condition is satisfied it performs the following tasks:

- Hands over all the result is has collected from the visited hosts, back to the user (agent based application), by retrieving its proxy from the proxy object;
- Disposes of itself, as it is no longer needed.

The last if case will be run when either of the above conditions is not met ; this means that the agent is neither at home nor at TS; it is doing its job at some other context, touring the different hosts for the information.

The major tasks performed are shown in the code fragment given below in Figure 6-12.

```

else{
PublicKey pubk=CipherCls.getPublicKey(alias);
SecretKey secKey = CipherCls.generateSessionKey();
SealedObject
encKey=CipherCls.sealSecretKey(secKey, pubk);
String result =
System.getProperty("os.name"); SealedObject
encRes=CipherCls.sealObject(result, secKey);
Message msg = new Message("MobieAway");
msg.setArg("keys", encKey);
msg.setArg("infos", encRes);
prxy.getSProxy().sendOnewayMessage(msg);
dstn.goToNext(this);
}

```

Figure 6-12 Code: Touring the Different Hosts

The agent executes the security model developed to protect parts of the mobile agent, while the agent is touring the potentially malicious hosts in the agent space. There, it performs the following range of tasks:

- It retrieves the public key of the Agentapp using the utility method's CipherCls.getPublicKey();

- Generates a random symmetric key using generateSecret () method of CipherCls;
- Next; the symmetric key generated is sealed(encrypted), using sealSecetKey() method of the CipherCls object;
- Does its job;
- Encrypts the data it obtained, using sealobject() method of CipherCls class;
- Constructs a message object that will be sent to the stationary agent that encapsulates both the retrieved object and the sealed symmetric key; and
- Moves on either to a next host or back to TS, depending on whether it's at the middle of its journey or at the end of its host visit mission.

#### 6.2.2.3. AgentApp

This class creates the agent based application located at the home computer. It is used by the user to create and dispatch a mobile agent that acts on its behalf to perform a range of tasks on the hosts' list provided. This class constitutes a number of other classes that collect input information from the user, about the mobile agent to be dispatched through GUI's.

#### 6.2.2.4. pathDIg

This class implements a dialog box that lets users to enter the address of the list of hosts the mobile agent is going to visit. It allows the users to enter the address of the TS , as shown in figure 6-13 given below.

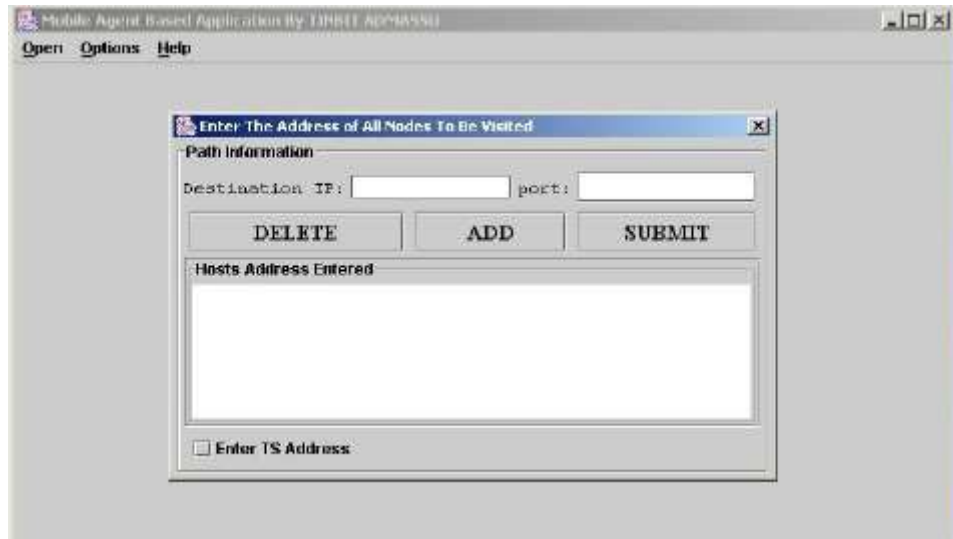


Figure 6-13: Path Dig box

#### 6.2.2.5. MobileUIF

This class is the GUI; it is the face of the application.

#### 6.2.2.6. infoDIg

This class shows the information retrieved by the agent as shown below in figure 6-14.

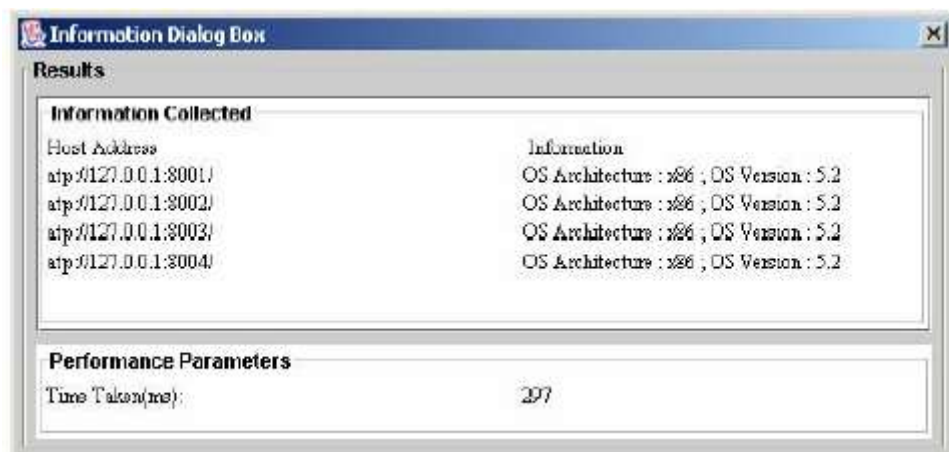


Figure 6-14: Dial Box

Retracing the steps to describe the major parts of the main class, AgentApp, is shown in the below figure 6-15.



```
public class AgentApp extends Aglet implements
ActionListener
{
```

Figure 6-15 Class: AgentApp

The above code describes the class declaration of the AgentApp. It extends Aglet as in the case of mobileA and statA, implying that it is an agent too. This is only done to exploit the famous information exchange mechanism supported by aglet's platform: the message object. The declaration shows that the class implements the interface ActionListener, so that it can listen and react to events as the user interacts with the GUI components of the user interface as shown below in figure 6-16.

```
public mobileUIF mobui ;
public JMenuItem crtMenuI , DispaMenuI
, setinMenuI;
public ArrayList clrKeys=null;
public ArrayList clrInfos=null;
public ArrayList abahsts=null;
public ArrayList abavhms=null;
public SignedObject dsDstn=null;
public Destn dstn;
public PrivateKey prvk;
public String agletName = "mobileA";
public String alias="kprvirtual";
public String password="virtual";
public long send =0;
public long arrived =0;
public long lived =0;
public proxyH prxy;
```

Figure 6-16 Code: Main Variables for AgentApp Class

The code above shows the most important instance variables of the AgentApp class.

- mobui is an instance of mobileUIF class and is used to show up the main GUI as the agent based application runs.
- The instance variables of JMenuItem, crtMenuI, DispaMenuI, setinMenuI, are used to get a reference to some of the most important menu items inside the mobileUIF class. The AgentApp is using these instance variables codes, how each should behave as the user interacts with the corresponding menu items.
- Abahsts and abavhms instance variables are of ArrayList types and holds the list of hosts and TS the mobile agent visits respectively.

- Instance variables `ArrayList clrKeys` and `ArrayList clrInfos` are used to hold the unscrambled keys and results respectively, to show it up to the user.
- Instance variables `SignedObject dsDstn` and `Destn dstn` hold the digitally signed and the unsigned destination object respectively.
- The instance variable `AgletName = "mobileA"` holds the class name of the mobile agent that is going to be dispatched by the `AgentApp`.
- The next three instance variables are used to access and store cryptographic keys from the keystore as described as follows. `Password="virtual"` holds the password of the whole of the keystore. `alias="kpvirtual"` holds a 'name or alias' that differentiates the public key of the application with the other public keys available in the keystore and finally, `PrivateKey prvK` holds the private key to be retrieved from the key store and will be stored in the `prvk` instance variable. The last instance variables of type `long` are used to capture performance parameter of the mobile agent as shown in figure 6-17 given below.

```
public void onCreate(Object init)
{
    prxy = new proxyH(this);
    abavhms = new ArrayList();
    mobui = new mobileUIF() ;
    crtMenuI = mobui.getCreateMI();
    crtMenuI.addActionListener(this);
    DispaMenuI = mobui.getDispatchMI();
    DispaMenuI.addActionListener(this);
    setinMenuI = mobui.getSettingMI();
    setinMenuI.addActionListener(this);
    clrKeys=new ArrayList();
    clrInfos=new ArrayList();
    mobui.show();
}
```

Figure 6-17 Code: onCreate Method

The above code describes the major actions undertaken as the agent based application's `onCreation()` method is called as.

The last line of code, as shown in figure 6-18, `mobui.show()`, brings up the main GUI or face of the application.

```

public boolean handleMessage(Message msg)
{
    arrived = new Date().getTime();
    lived = arrived - send;
    prvK=CipherCls.getPrivateKey(alias,
    password.toCharArray());
    Iterator
    kitr=((ArrayList)msg.getArg("keys")).iterator();Iter
    ator
    resitr=((ArrayList)msg.getArg("infos")).iterator();
    while (kitr.hasNext())
    {
        SealedObject sldKey=(SealedObject)kitr.next();
        SecretKey secKey=CipherCls.unsealSecretKey(sldKey,
        prvK);
        clrKeys.add(secKey);
        SealedObject
        sldRes=(SealedObject)resitr.next();Object res = new
        Object();
        res=CipherCls.unsealObject(sldRes, secKey);
        clrInfos.add(res);
    }
    infoDlg inf= new infoDlg(mobui ,"Information Dialog
    Box",
    abahsts ,clrInfos ,lived);
    return true;
}

```

Figure 6-18 Code: Behaviour of the Application

The above code defines the behaviour of the application as it interacts with other agents. The first line of code records the time the agent returns home, to compute a performance parameter. Next, the private key of the agent application is retrieved to unscramble the information that the mobile agent will be delivering to the AgentApp shortly. The mobile agent passes the information it has collected from the hosts in the agent space using the message object to Iterator for easy traversal. The while loop using the iterator object traverses the scrambled information handed over, to unscramble and store it in a plain form to the instance variables `clrKeys` and `clrInfos`. The utility class's `CipherCls` method `unsealSecretKey()` and `unsealSealedObject()` plays a crucial role in the process. At last the information is passed to `infoDlg` dialog box to be displayed to the user.

### 6.2.3. Setting up the laboratory

A specific set up is made in order to initiate the mobile agent network.

Computer A is considered to act as trusted server (TS) and computer B runs many host nodes simulated through various port numbers as well as the home node in a virtualized mode as shown in figure 6-19.

Wireshark Network Packet Analyser will be running regularly over computer A. its job is to capture, sniff packets in a network and store them.

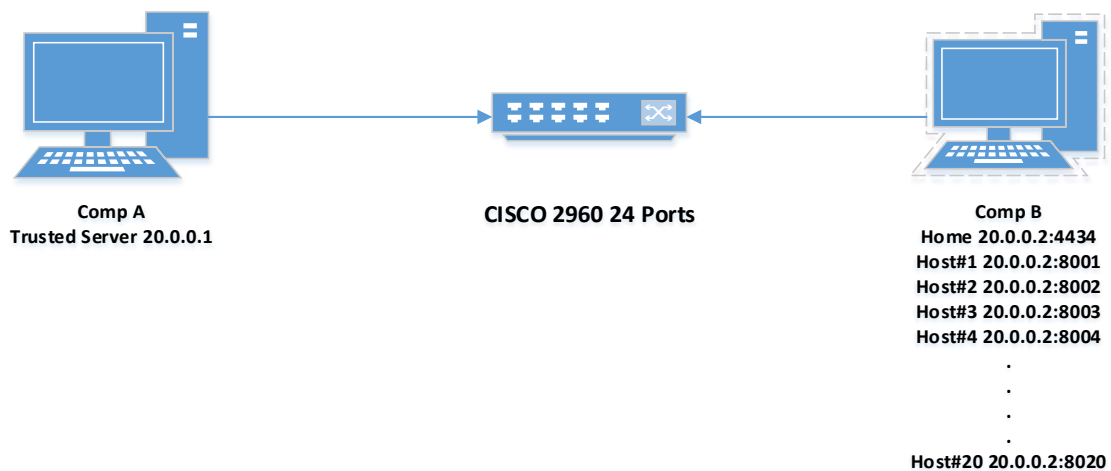


Figure 6-19 : Test Environment Set Up

The setup is amply described in Appendix B.

Several types of Mobile Agents (Proposed MA, DS MA and Normal MA) are generated as described below:

- Normal Mobile Agent (Normal MA): An Agent that executes mobile computation in the usual way.
- Proposed Mobile Agent (Proposed MA): A Mobile Agent which is directed by the security model mentioned in section 4.
- Digitally Signed Mobile Agent (DSMA): A Mobile Agent that supports digital signing of the destination object while still performing computation.

## 6.2.4. Evaluation Strategy

This section will measure the capability of the model towards eavesdropping and alteration threats between several types of Mobile Agents (Proposed MA, DS MA and Normal MA).

### 6.2.4.1. Eavesdropping

Figure 6-20 shows analysis of the packet captured while the Normal MA and DS MA are in execution.

As it is shown in either cases (DS MA and Normal MA), it is possible to eavesdrop what information is retrieved and exchanged at each host: "OS Architecture: x86; OS Version: 5.2".

```

00e0 75 74 69 6c 2e 48 61 73 68 74 61 62 6c 65 13 bb util.Hashtable..
00f0 0f 25 21 4a e4 b8 03 00 02 46 00 0a 6c 6f 61 64 .%!J....F..load
0100 46 61 63 74 6f 72 49 00 09 74 68 72 65 73 68 6f FactorI..thresho
0110 6c 64 78 70 3f 40 00 00 00 00 08 77 08 00 00 00 ldxp?@..w...
0120 00 0b 00 00 00 01 74 00 05 69 6e 66 6f 73 74 00 .....t infnst
0130 28 4f 53 20 41 72 63 68 69 74 65 63 74 75 72 65 (OS Arch itecture
0140 20 3a 20 78 38 35 20 3b 20 4f 53 20 56 65 72 73 : x86 : OS Vers
0150 69 6f 6e 20 3a 20 35 2e 32 78 74 00 09 4d 6f 62 ion : 5. 2xt..Mod
0160 69 65 41 77 61 79 ieAway

```

Figure 6-20: Captured Packet Analysis for DS MA and Normal MA

Figure 6-21 shows the analysis of the captured packet while the Proposed MA is in operation. As it can be seen from the figure, unlike the above case, it is not possible to look into its content since the information is sent to the TS in encrypted form.

```

01a0 00 02 78 70 70 75 72 00 02 5b 42 ac f3 17 f8 06 ..xppur. [B....
01b0 08 54 e0 02 00 00 78 70 00 00 00 40 5c 89 a6 04 .T....xp ...@...
01c0 c0 63 90 f9 1c 71 c5 38 da 3d cf f5 db 9a fe 97 .c...q.8 .=.....
01d0 04 88 ce f4 65 be 06 8b 1e 06 50 be 25 17 39 e9 .....e...P.%:9.
01e0 d6 79 f2 6a 93 40 27 5f b8 b3 03 45 b5 c4 38 38 .y.j.@' ...E..88
01f0 5b d8 e2 88 f2 26 6e ad 79 08 aa 12 70 74 00 03 k....&n. y...pt..
0200 52 53 41 74 00 05 69 6e 66 6f 73 73 71 00 7e 00 RSAt..in fassq.~.
0210 08 70 75 71 00 7e 00 0b 00 00 00 30 03 83 e1 e9 .puq.~. ...0....
0220 04 73 9e a5 fa 97 86 f6 67 2b 73 fc 66 46 7b 7b .s.....g+s.fF[{
0230 0c 6a 89 92 d7 34 8f 77 e6 2a 0b 3e 54 2b c2 62 .j...4.w .#.>T+.b
0240 03 42 7b 16 b8 d3 25 71 c8 cd 48 ba 70 74 00 03 .B[...%q ..H.pt..
0250 41 45 53 78 74 00 09 4d 6f 62 69 65 41 77 61 79 AESxt..M obieAway

```

Figure 6-21: Captured Packet Analysis for Proposed MA

Hence, the security model provides the required confidentiality of the information while it is being stored at ASE.

#### 6.2.4.2. Alteration

To test the strength of the proposed approach against alteration, a malicious host is included.

A malicious node is interfering on a different port number in the Computer B. This node is planned to behave maliciously towards the Proposed MA; its goal is to supply a wrong public key to the MA as it arrives there and is in the process of un-signing its digitally signed destination object. But fortunately, the MA cannot un-sign the signed object using the public key just supplied. This is because the destination object is signed by the private key of the home node, not by a private key which corresponds to the public key supplied by the hostile node. Therefore, any attempt of alteration of destination object will be detected by the MA as shown in figure 6-22.

```
E:\Documents and Settings\Administrator\Desktop>cd\
E:\>agletsd -port 8003 -f E:\Extracted\cnf\aglets.props
[Warning: The hostname seems not having domain name.
Please try -resolve option to resolve the fully qualified hostname
or use -domain option to manually specify the domain name.]
The following error occurred: java.lang.RuntimeException: Verification failed!
java.lang.RuntimeException: Verification failed!
    at CipherCls.getSignedObject(CipherCls.java:94)
    at mobileA.onArrival(mobileA.java:87)
    at com.ibm.aglet.Aglet.processMobilityEvent(Unknown Source)
    at com.ibm.aglet.Aglet.dispatchEvent(Unknown Source)
    at com.ibm.aglets.LocalAgletRef.dispatchEvent(Unknown Source)
```

Figure 6-22: Detection of Malicious Host

#### 6.2.4.3. Performance Comparison:

To measure the performance of every mobile agent (Normal, DS, Proposed MA), a similar test environment, as above is used. Their performance is compared in terms of their average turnaround time, measured in milliseconds (ms).

This performance parameter is the average time in milliseconds (ms) each Mobile Agent requires to do the job, after being dispatched, till it returns and handovers the result to the user.

As expected DS Mobile Agent takes in between of the two. Comparing the execution time of the Normal MA with the Proposed MA, the latter needs approximately 5x more time as shown in Figure 6-23.

This substantial amount of time is the price to pay to achieve the corresponding security: Generation of the keys, encryption of partial information, verification of destination object at each visited host and at last collecting the results back to the Mobile Agent from the TS all add up to form a big turnaround time.

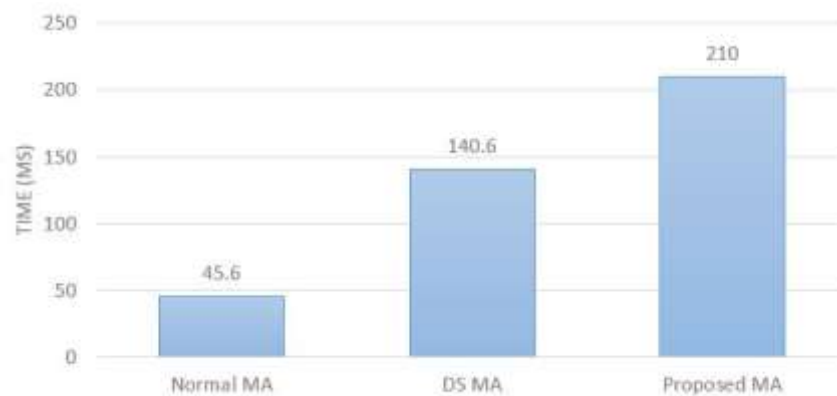


Figure 6-23: Performance Comparison between Different Types of Mobile Agents

Comparing the performance time between DS MA and Proposed MA, the DS MA apparently needs less time. This is due to the fact that DS MA does not carry out some of the functions the Proposed MA performs like Generation of keys and Encryption. It takes time since it verifies that the destination object is a valid copy on its arrival at each and every host.

Figure 6-24 compares the execution time for all Mobile Agent cases. As the number of nodes to be visited is steadily increased, it can be noticed that the turnaround time increases. This is a tribute to the fact that there are more jobs to be done.

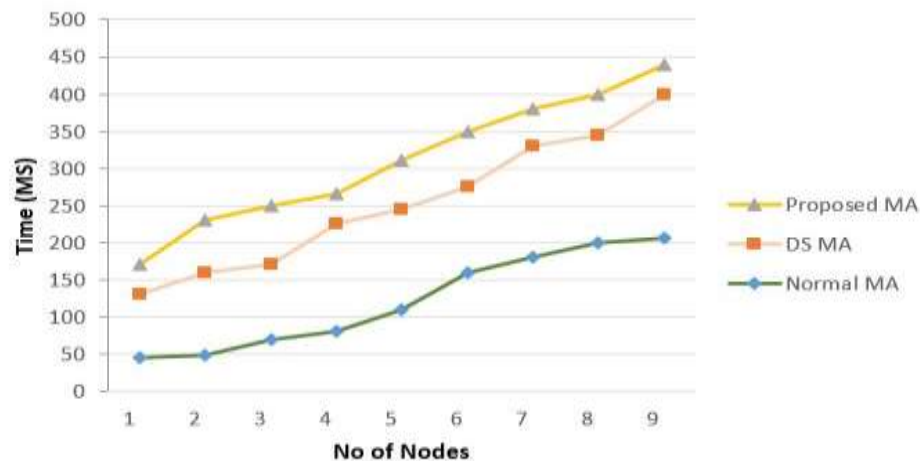


Figure 6-24: Performance Time Trend as the Number of Nodes Visited Increases

### 6.3. Detection Model Test Results

This section presents the performance analysis results for integrating divergence measures over Sketch, for detecting SYN flooding attacks in a mobile agent network. A mobile agent network is going to be implemented. Hence, the comparison of three divergence measures (HD, PD &  $\chi^2$ ) over Sketch for the detection of flooding attacks will be tested.

For the sake of simplicity, the analysis on the detection of SYN flooding attacks is focused, as it is the widely used attack for DDoS in these days.

#### 6.3.1. Experiments Set Up

The above described mobile agents will have to execute the similar path. To measure the capability of the model towards eavesdropping threat, a test environment is set up using the above mentioned computers as shown in Figure 6-19. Computer A is considered to act as trusted server (TS) and computer B runs many host nodes simulated through various port numbers as well as the home node in a virtualized mode. Appendix B presents the tools used to assure the set up.

Ethereal will be running regularly over computer A. Its job is to collect packets in the mobile agent network and store them for a period of 4h00 from 18/02/2017 07h30 to 11h30. These traces are used to test the efficiency of divergence measures. IP addresses in the traces are scrambled by a modified version of tcpdriv tool, but correlation between



addresses are conserved. These 4h traces using Sketch data structure are analysed using a key of the Sketch ( $\kappa_n = \text{DIP}$ ), and a reward  $v_n = 1$  for SYN request only, and  $v_n = 0$  otherwise. The Sketch is set with a width  $K$  to 1024, and the number of hash  $H$  with a value of 5.

For simplicity, 12 real distributed SYN flooding attacks with different intensity inside this trace are injected. These attacks are inserted each 30 minutes (on instants  $t=30, 61, 90, 127, 157, 187, \text{etc.}$ ) span for 10 minutes. These different intensity attacks are shown in Figure 6-25. The first attack begins with a value of 700 SYN/min and decreases until 80 SYN/min.

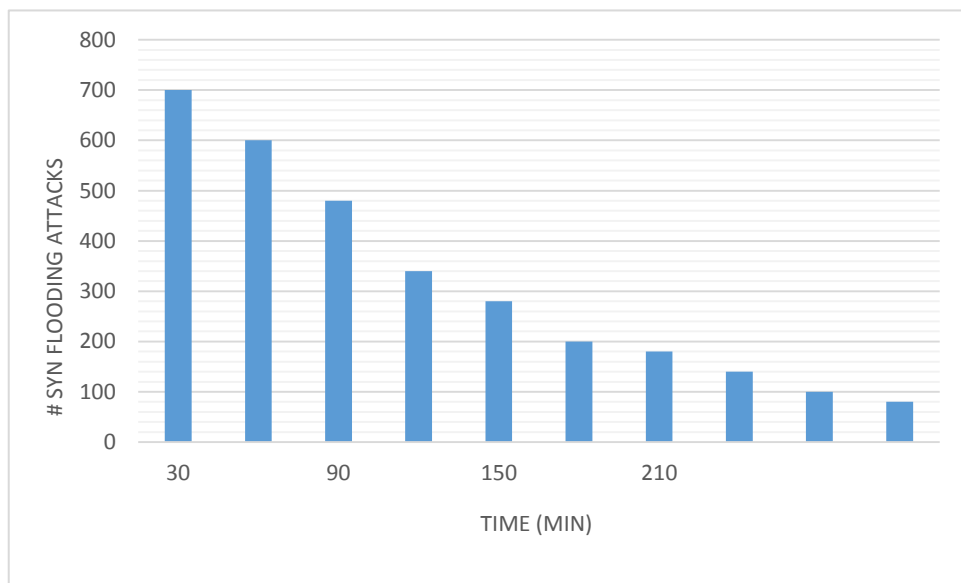


Figure 6-25: SYN Flooding Attacks

### 6.3.2. Evaluation Strategy

This section evaluates the performance of the proposed detection scheme.

#### 6.3.2.1. Detection Techniques

Below are the techniques to detect the anomaly in mobile agent networks. They rely on the techniques described in Chapter 4.

- Sketch technique
- Dynamic Threshold
- Divergence Measures
- Receiver Operating Characteristic (ROC)

A focus on tuning the parameter of Divergence Measures to optimize the performance is applied. The performance analysis over available IP traces, in Mobile Agent Network integrated with flooding attacks, is conducted. An evaluation of the performance of the proposed divergence measure via the Receiver Operating Characteristic (ROC) is conducted.

#### 6.3.2.1.1. Receiver Operating Characteristic (ROC) Technique

To evaluate the performance of Power Divergence with different value of  $\beta$ , an investigation between the False Positive Rate (FPR) and True Positive rate (TPR), which is the Detection Rate (DR), is performed. A confusion matrix and ROC will be used to evaluate the detection. Receiver Operating Characteristic (ROC) is used for accuracy analysis when varying the value of the threshold  $h$ . ROC curve shows the variation of the true positive rate (Eq. 16) in term of false positive rate (Eq. 17):

$$DR = TPR = \frac{TP}{TP + FN} \times 100 \quad (16)$$

The false positive rate is defined as the ratio of false alarms to the number of raised alarms:

$$FPR = \frac{FP}{TP + FP} \times 100 \quad (17)$$

where,

- **True positives (TP):** refer to the positive tuples that were correctly labelled by the classifier; an alarm is triggered because there is anomaly in the network.
- **True negatives (TN):** are the negative tuples that were correctly labelled by the classifier; an alarm is triggered but there is no anomaly in the network.
- **False positives (FP):** are the negative tuples that were incorrectly labelled as positive; an alarm is not be triggered when there is anomaly in the network.
- **False negatives (FN):** These are the positive tuples that were mislabeled as negative; an alarm is not be triggered since there is not any anomaly in the network.

#### 6.3.2.1.2. Dynamic Threshold

Instead of using a static threshold, the Exponential Weighted Moving Average (EWMA), defined in the equations (3, 4, 5 &6) in Chapter 4, is used.

The parameters  $\alpha$ ,  $\lambda$  and  $\mu$  are all tunable parameters in the mode.

After several experiments on real attacks, numerous trial & errors are conducted. This is to find the best parameter of the threshold to improve the detection accuracy as mentioned in below table 6-1.

Table 6-1 Threshold Parameters

Threshold Parameter	
$\alpha$	0.125
$\beta$	0.25
$\lambda$	5
$\mu$	1

#### 6.3.2.1.3. Divergence Measures and Sketch Technique

The detection scheme will rely on the sketch technique and three types of divergences measures: Hellinger Distance, Chi-Square, and Power Divergence as described in Chapter 4.

#### 6.3.2.2. Injection of SYN Flooding Attacks

Figure 6-26 & Figure 6-27 show the variation of total number of mobile agents' packets before and after the injection of SYN flooding attacks.

By comparing these variations, the differences between both figures without deep inspection might not be noticed. Inserted attacks do not induce heavy deviations in the time series of the total number of SYN requests. This can be explained by the fact that the intensity of SYN flooding attacks is not large compared to the intensity of the total number of SYN segments. In such cases, the detection of attacks is very challenging, because no heavy changes in the time series describing the variations of the total number of SYN, and the intensity of the SYN flooding attacks is buried by the large number of SYN (as shown in Figure 6-25) before attacks injection.

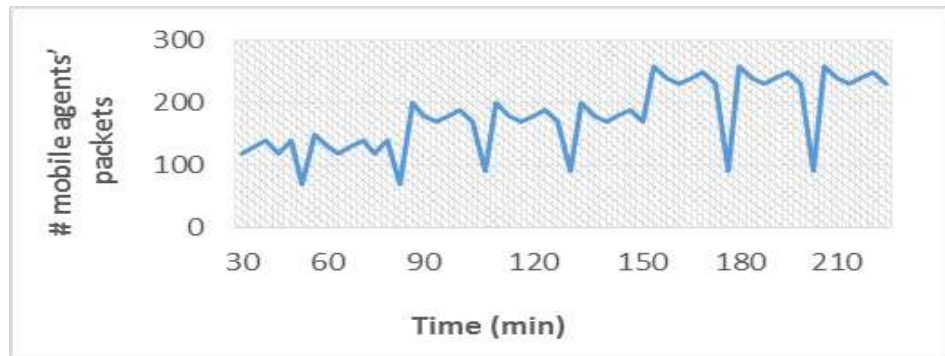


Figure 6-26 : Total Number of Mobile Agents' Packets

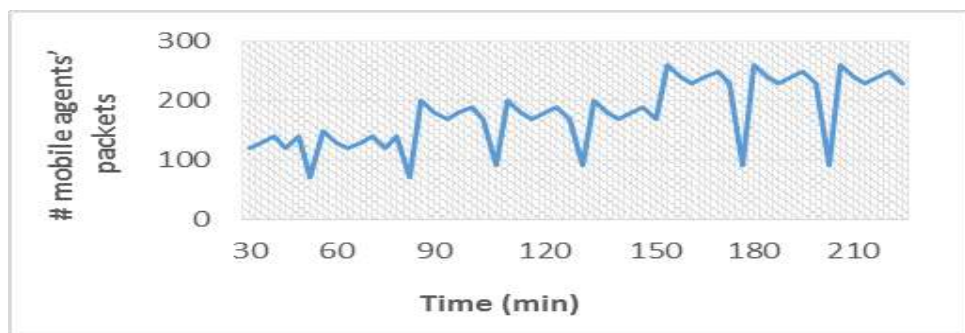


Figure 6-27 Total Number of Mobile Agents' Packets after SYN Flooding Attacks Injection

### 6.3.2.3. Comparison between HD & Chi-square

This section presents the evaluation results of the application by comparing Hellinger Distance & Chi-Square response on the mobile agent IP traces.

First, the analysis begins by applying HD &  $\chi^2$  divergence over over the mobile agent IP traces (before injection SYN flooding attacks). The dynamic threshold, as given in Equation 6, is set.

Figures 6-28 & 6-29 show the variation of these 2 divergence algorithms as well as the dynamic threshold before the injection of attacks. When the value of divergence measures is larger than the threshold in at least 3 hash tables in the Sketch, an alarm is triggered. It can see that both algorithms were able to detect anomalies at different times (t=90, 127,157,180 etc.). These anomalies are temporary and they do not persist for more than

many minutes. However, there are more anomalies that can be detected by using the source IP address as the key of the Sketch, but the analysis will be restricted to SYN flooding attacks. In fact, after the manual verification of traces, it is found that HD triggers 4 false positive alarms, and the  $\chi^2$  divergence achieves very high detection accuracy with 2 positive alarms.

Indeed, the analyses are conducted by applying the HD and Chi-Square over the mobile agent IP traces (after injection SYN flooding attacks). It is noticed that in case of Hellinger Distance using a dynamic threshold, 4 false positive alarms with a detection of 100% are obtained (Figure 6-30). However, in the case of Chi-Square, 2 false positive alarms with a detection of 100% are obtained (Figure 6-31).

As conclusion, Chi-square divergence performs better than HD in terms of reducing false positive alarm, with less effort for tuning the dynamic threshold. In conclusion, Chi-Square outperforms Hellinger Distance in terms of detection.

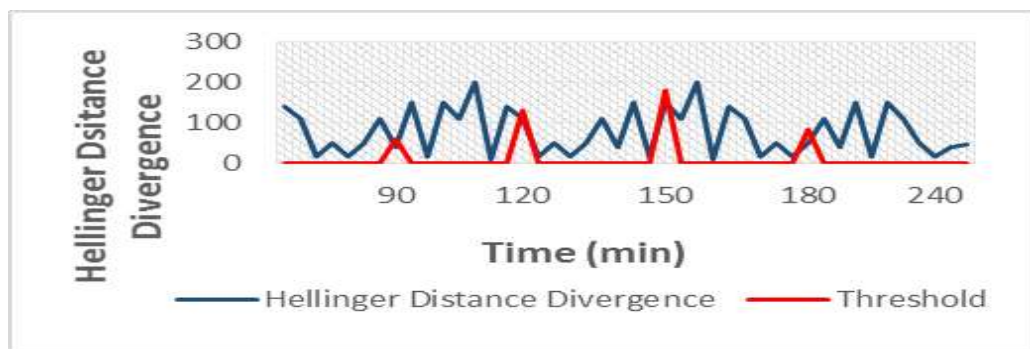


Figure 6-28: Hellinger Distance before attacks

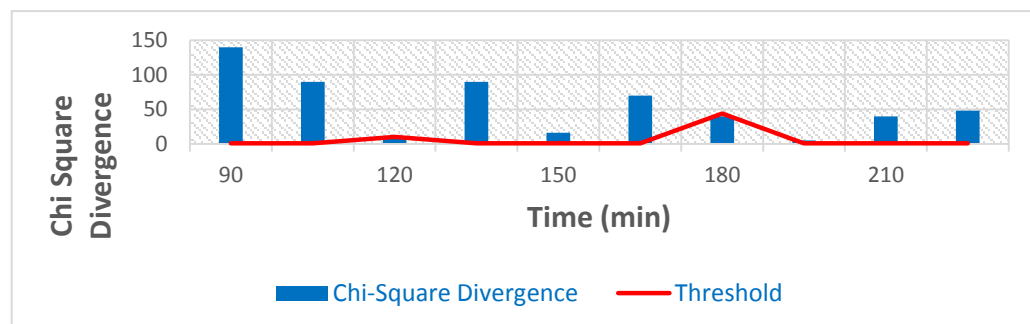


Figure 6-29 : Chi-square before attacks

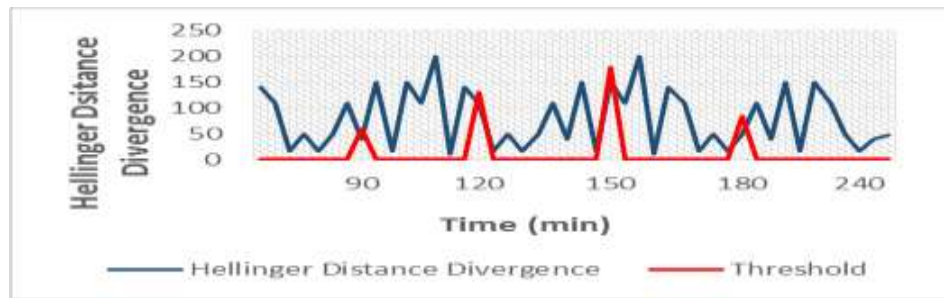


Figure 6-30: Hellinger Distance after attacks

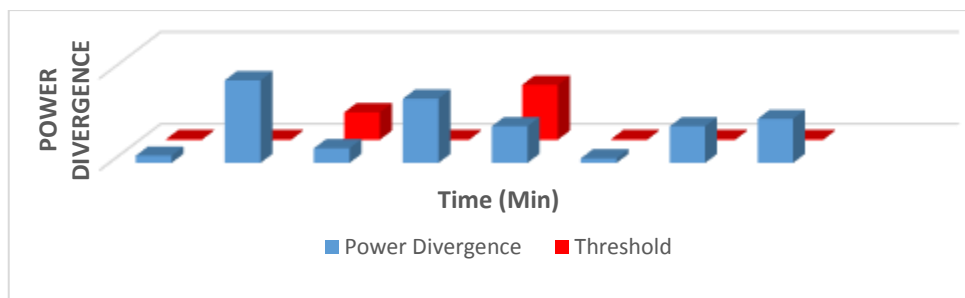


Figure 6-31: Chi-square After Attacks

#### 6.3.2.4. Power Divergence

##### 6.3.2.4.1. Comparison of PD Using Different Values of $\beta$

This part will focus this study on the Power Divergence.

Due to space limitation, the results for the following of values  $\beta$  are tested  $\beta = 0.5, 1.5, 2$  and  $2.2$ .

This divergence presents some interesting special cases as discussed in Chapter 4.

For  $\beta = 0.5$ , this divergence is  $4 \times HD (P || Q)$ , and for  $\beta = 2$  it is equal to  $0.5 \times x^2 (P || Q)$  divergence. Obviously, this power divergence then outperforms the  $x^2$  and HD measures. This will be tested and proved in this section.

The value of  $\beta = 0.5$  makes the Power Divergence (PD) proportional to Hellinger Distance (HD). Figure 6-32 shows the variation of Power Divergence for  $\beta = 0.5$  with the dynamic threshold given in Equation 6. Power Divergence is able to detect all the SYN flooding attacks but with 4 false alarms.

For the value of  $\beta = 1.5$ , Figure 6-33 shows the variation of Power. It is noticed that via this value of  $\beta$ , all the attacks have been detected (100%) with only 3 false positive alarm. The intensity of spike is proportional to the intensity of the attack.

For the value of  $\beta = 2$ , Figure 6-34 shows the variation of Power. It can be noticed that via this value of  $\beta$ , all the attacks have been detected (100%) with only 2 false positive alarms.

For the value of  $\beta = 2.2$ , Figure 6-35 shows the variation of Power. It can be noticed that via this value of  $\beta$ , all the attacks have been detected (100%) with only false positive alarm.

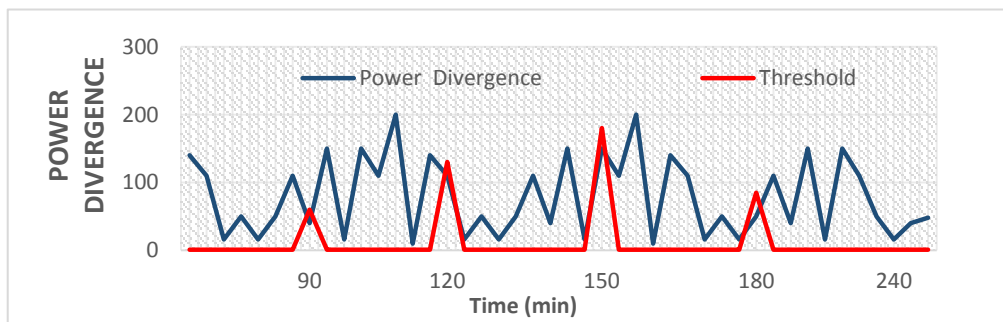


Figure 6-32: Power Divergence for  $\beta=0.5$

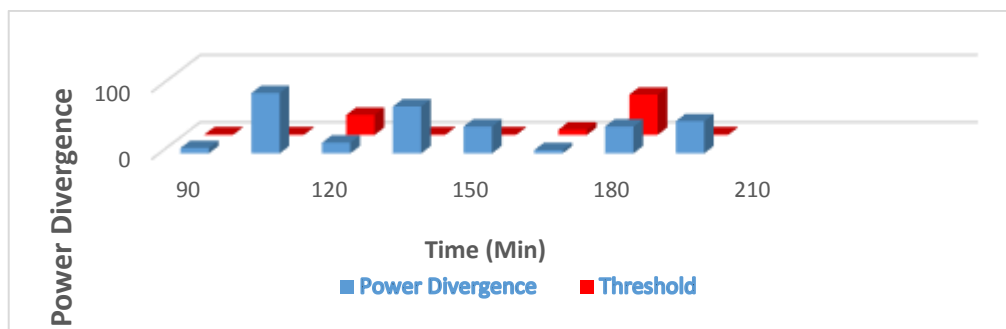


Figure 6-33: Power Divergence for  $\beta = 1.5$

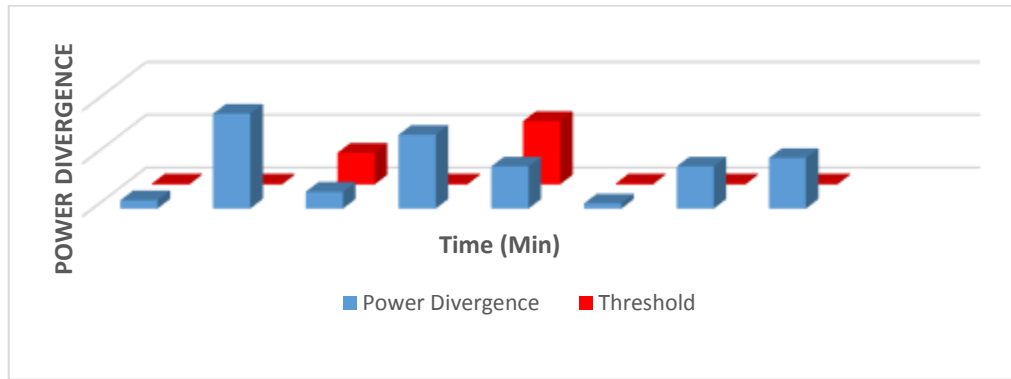
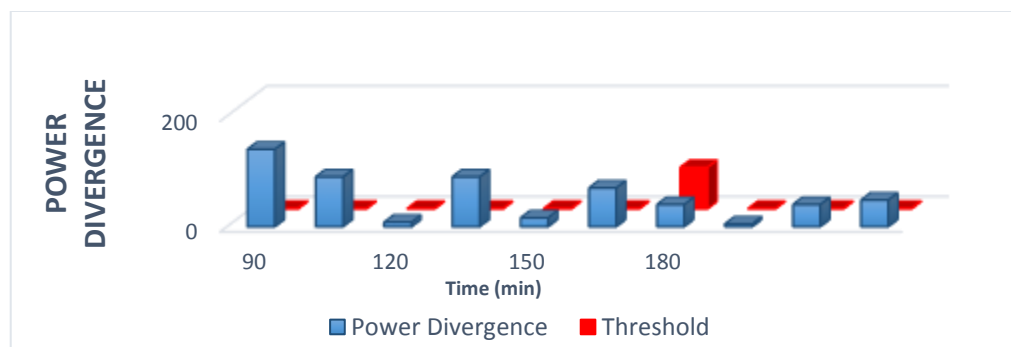
Figure 6-34: Power Divergence for  $\beta = 2$ Figure 6-35: Power Divergence for  $\beta = 2.2$ 

Table 6-2 presents a summary of the result of detections:

Table 6-2: False Positive Alarm for Different Values of  $\beta$ 

Different Values of $\beta$ of PD	False Positive Alarm	Notes
0.5	4	For $\beta=0.5$ , $PD=4 * HD$
1.5	3	
2	2	For $\beta=2$ , $PD=0.5 * x^2$
2.2	1	

Based, on the above experiments, it can be concluded that the value of  $\beta = 2.2$  outperforms the values of  $\beta = 0.5, 1.5,$  and  $2$  in terms of true detection and false positive rate.

By this action, Power Divergence outperforms Hellinger Distance and Chi-Square. Moreover, the best value of  $\beta$  for limiting the false positive alarm is  $2.2$ .



#### 6.3.2.4.2. Flooding Rate Comparison

The experiments are repeated for several times on Hellinger Distance, Chi-square and PD  $\beta=2.2$  and change the attack rates accordingly. This specific value of  $\beta$  is chosen for PD since, as shown previously, it presents better detection with low false positive alarm rate. Even when the attack rate is as low as 70 per mn, Chi-Square can still identify it with the accuracy of 80%, where HD accuracy is 60%. Moreover, PD with  $\beta=2.2$  presents an accuracy of 98.1% as shown in Table 6-3.

Table 6-3: Flooding Rate Result

Flooding Rate	Number of Experiments	Detection Probability PD with $\beta=2.2$	Detection Probability of Chi-Square	Detection Probability of HD
70	40	98.1%	80%	60%
80	40	100%	100%	100%
120	40	100%	100%	100%
480	40	100%	100%	100%
700	40	100%	100%	100%

The intensity of raised spikes in PD increases with the intensity of attacks and the dynamic threshold becomes useless.

As conclusion, PD  $\beta=2.2$  represents better detection accuracy (of a value of 98.1%) in comparison with Hellinger Distance and Chi-Square in terms of low intensity level of attack rate.

#### 6.3.2.4.3. Receiver Operating Characteristic (ROC) Result

To evaluate the performance of Divergence Measures (PD, Chi-Square & PD), an investigation between the true positive rate and the false positive rate is conducted. Receiver Operating Characteristic (ROC) is used for accuracy analysis when varying the value of the threshold  $h$ ; A high true positive and a low false alarm rate are desired to achieve good performance.

The performance of the three divergence measures over Sketch:  $x^2$ , HD and PD with  $\beta=2.2$  are compared. As mentioned previously, only this specific value of  $\beta$  is chosen, since this value presents better detection of false positive alarm rate.

Figure 6-35 given below shows that the ROC curves (TPR versus FPR) when varying the parameters in the threshold.

With the use of dynamic threshold, and the Halt technique of forecasting after the detection of attack, HD achieves a TPR=100% with a FPR of 43%,  $\chi^2$  achieves a TPR=100% with FPR=20%, and PD ( $\beta=2.2$ ) achieves a TPR=100% with FPR=3.8%.

Thus, when comparing the performance of these algorithms, it is found that PD achieves better than HD and Chi-square. PD outperforms both measures for anomaly detection with the lowest FPR. In conclusion, PD leads to very attractive performance in terms of True Positive Rate and False Positive Rate.

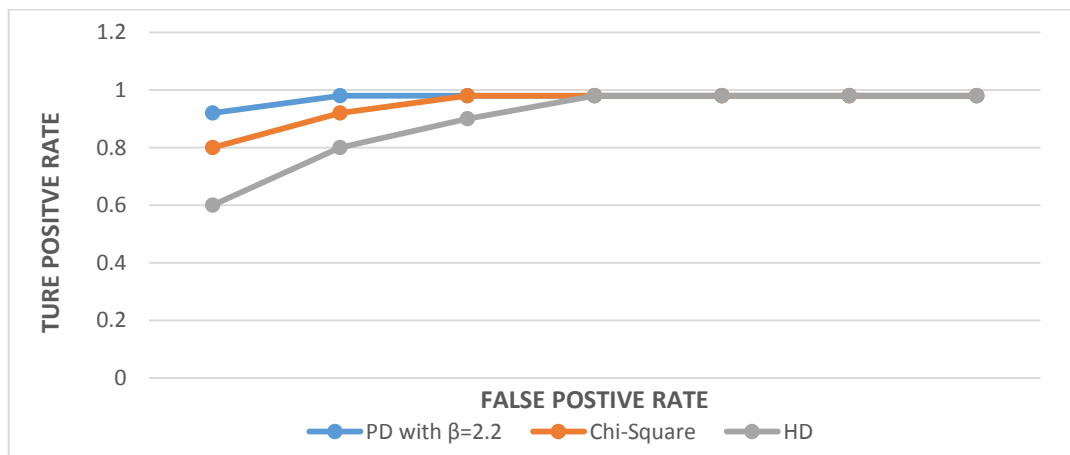


Figure 6-36: Receiver Operating Characteristic for  $\chi^2$ , HD and PD

Table 6-4 summarizes the result of ROC based on different values of  $\beta$ :

Table 6-4 : ROC Based on Divergence Measures

Divergence Measures	False Positive Alarm	True Positive Alarm
HD	43%	100%
Chi-Square	20%	100%
PD for $\beta=2.2$	3.8%	100%

As per the results of the performed experiences, the best detection (100%) of anomaly with a reduced false alarm ratio (3.8%) is Power Divergence with a value of  $\beta=2.2$ . Its confusion table is provided in Table 6-5 given below.

Table 6-5: Confusion Table for PD of  $\theta=2.2$ 

PD with $\beta=2.2$	Predicted: No	Predicted: Yes
Actual: NO	96.20%	3.80%
Actual: yes	2.90%	98.10%

## 6.4. Conclusion

The main goal of this chapter is to test the strength of both the proposed approaches for the protection and detection of attacks in a mobile agent platform.

The protection approach has evaluated how different types of mobile agents reacts, based on real traces with attacks. The experimental results have shown that the proposed security approach provides the required confidentiality of the information while it is being stored at ASE, where it is not possible to look into its content as information is forwarded to the TS in an encrypted form. Additional test has shown that this security approach is capable of protecting against alteration attacks.

Then a measure of the performance of the proposed approach is performed. The results indicate that this new model for mobile agent needs approximately four times more than a normal agent to execute its job. This is caused due to the Generation of the keys, encryption of partial information, verification of destination object at each visited host and finally collecting the results back to the Mobile Agent from the TS all this add up to form a big turnaround time. Moreover, a comparison with the execution time for all Mobile Agent cases was carried out as the number of nodes to be visited is increased it was noticed that the turnaround time also increased. This is a tribute to the fact that there are more jobs to be done.

In conclusion, the proposed approach is capable of providing the confidentiality of the mobile agent in addition to being able to protect against eavesdropping and alteration attacks.

Flooding attack is very severe in mobile agent networks. A flooding detection scheme is proposed by integrating the following techniques: *Sketch* and three kinds of Divergence Measures viz: *Hellinger Distance*, *Chi-Square* and *Power Divergence*.

Sketch is used to build fixed-size compact summaries of the mobile agents' flows. Divergence Measures (HD, PD for  $\beta=2.2, x^2$ ) will profile normal traffic behaviours and detect attacks based on probability distributions defined from the sketch tables.

The goal of the flooding detection scheme is to find the best way to limit the high false alarm ratio and the capability of detecting low intensity attacks in mobile agents' network. Simulation results demonstrates the performance of the proposed technique. A focus on tuning the parameter of Divergence Measures is performed to optimize the performance. The three Divergence Measures (HD, PD for  $\beta=2.2, x^2$ ) are compared.

As per the experiments, Chi-square divergence performs better than HD in terms of reducing false positive alarm, with less effort for tuning the dynamic threshold.

Then the simulation is continued by introducing Power Divergence with different values of  $\beta = 0.5, 1.5, 2$  and  $2.2$ . This divergence presents some interesting special cases as discussed in Chapter 4.

For  $\beta = 0.5$ , this divergence is  $4 \times HD (P || Q)$ , and for  $\beta = 2$  it is equal to  $0.5 \times x^2 (P || Q)$  divergence. Obviously, this power divergence then outperforms the  $x^2$  and HD measures. Based on the performed experiments, the value of  $\beta = 2.2$  outperforms the values of  $\beta = 0.5, 1.5$  and  $2$  in terms of false positive rate.

In conclusion, Power Divergence outperforms Hellinger Distance and Chi-Square. The best value of  $\beta$  for low false positive alarm is  $2.2$  where only one false positive alarm is detected.

Further test is performed to know which Divergence Measure (HD, PD for  $\beta=2.2, x^2$ ) is capable of detecting the low intensity attacks by changing the attack rates accordingly.

Even if the attack rate is as low as 70 per mn, Chi-Square can still identify it with the accuracy of 80% where HD accuracy is 60%. Moreover, PD with  $\beta=2.2$  presents an accuracy of 98.1%.

To conclude, PD  $\beta=2.2$  represents better detection accuracy in comparison with Hellinger Distance and Chi-Square in terms of low intensity attack.

In short, the performance of Divergence Measures (PD, Chi-Square & PD for  $\beta=2.2$ ) has been evaluated by investigating between the true positive rate and the false positive rate. Thus, when comparing the performance of these algorithms, all Divergence Measures present a TPR of 100 % but PD of  $\beta=2.2$  presents the lowest FPR of 3.8%.

PD outperforms both measures (HD and  $x^2$ ) for anomaly detection with the lowest FPR.

To conclude, it can be discerned that the work has achieved in outperforming the existing detection solutions by tuning the Power Divergence with  $\beta=2.2$ . With this value of  $\beta$ , the proposed detection scheme is leading to a very attractive performance in terms of True Positive Rate (100%), limiting the False Positive Rate (3.8%) and is capable of detecting low intensity attacks.



# Chapter 7: Conclusion and Further Work

---

## 7.1. Conclusion

This work presents and analyses a new model of protection (protection and detection) against threats, that can be used for the evaluation and analysis of the performance of the MA paradigm, in comparison to the CS models.

MAs are simply put, just tools; but in actuality, they can be more. If they are uploaded with simple and suitable methods for different tasks, they can become highly skilled agents. This work tries to postulate that the Sketch techniques and Divergence measures integrated with Mobile Agent technology can effectively be combined to detect flooding attacks in communication networks. Indeed, the integration of mobile agents, with trusted server can protect the network against malicious attacks; it can easily facilitate the trip of mobile agents safely in the network.

The MAs are deployed using Aglet from IBM platform and the cryptographic functionality into the system, so as to provide crypto services; these are implemented using the freely available Java cryptography service provider called “BC provider.

The new proposed approach for protection against malicious hosts introduces the concept of setting up an active storage element in the agent space, as called “home away from home”, for partial result storage and separation as well as digital signing of the destination of the mobile agent.

This approach clearly addresses the alteration and eavesdropping issues. The issue of alteration of carried result through encryption of partial results which are collected from the visited hosts and are sent to the TS. This protects the malicious host from altering the carried information. Furthermore, the integrity and the confidentiality of the information, while it is sent to the trusted server and stored there, is achieved. The separation of the destinations from the other parts of the mobile agent helps the home to digitally sign the list of hosts it wants to visit. In addition, the mobile agent on each destination could verify whether its signed object is tampered with or not, before putting that object into use. This

mechanism gives a malicious host no chance to alter the list of host addresses the mobile agent has carried and is going to visit.

The work also addresses the threats of Eavesdropping of collected information, both from the internal and external malicious host perspective, through partially storing the encrypted information at the TS.

Moreover, the performance of this approach has been tested. The experimental results show that the new model of the mobile agent needs approximately 4x more time than a normal agent to execute its job. This is cause of the Generation of the keys, encryption of partial information, verification of destination object at each visited host and at last collecting the results back to the Mobile Agent from the TS all add up to form a big turnaround time. The execution time for all the Mobile Agent cases has definitely been compared; it is noticed that as the number of nodes to be visited is measured, the turnaround time increases. This is an indicator and a tribute to the fact that more jobs can be and need to be done in this respect.

The new proposed approach for detection of SYN flooding attacks is based on Sketch and Divergence Measures for anomaly detection on a mobile agent's network. The proposed approach evaluated on SYN flooding attacks. Performances were compared in terms of true positive and false alarm ratio, over mobile agents IP traces with injected real distributed SYN flooding attacks at known instants.

The accuracy of 3 divergence measures (HD, Power Divergence & Chi-square Divergence) over Sketch data structure has been analysed. Afterwards, dynamic threshold is used for achieving the best tradeoff between false alarm and true detection. It was found that HD performs a good detection, but with higher false alarm ratio than Chi-square divergence. Hence, it can be concluded that that Chi-square conducts better detection than HD for mobile agents' network. Furthermore, it is found that the intensity of triggered spikes by Chi-square divergence increases significantly with the intensity of attacks. It is important to note that these divergence measures with Sketch are computationally efficient for handling traffic on mobile agents' traffic.

It is shown that Power Divergence presents some interesting special cases. For  $\beta = 0.5$ , this divergence is  $4 \times \text{HD} (P || Q)$ , and for  $\beta = 2$  it is equal to  $0.5 \times x^2 (P || Q)$  divergence. Obviously, this power divergence outperforms then the  $x^2$  and HD measures. In fact, by changing the values of  $\beta$ , one can optimize the detection of attacks compared to the  $x^2$  and HD measures. For PD, different values of  $\beta = 0.5, 1.5, 2 \& 2.2$  are proposed and studied. Results have shown that for  $\beta = 2.2$ , PD outperforms the HD and  $x^2$ .

Based on the performed experiments, it can be concluded that the value of  $\beta = 2.2$  outperforms the values of  $\beta = 0.5, 1.5$  and  $2$  in terms of false positive rate. Only 1 false positive rate for  $\beta = 2.2$  is detected.

Moreover, experimental results show the capacity of PD in the detection of low intensity attacks. PD with  $\beta = 2.2$  presents a better detection accuracy of 98.1% than HD (60%) and  $x^2$  (80%).

Finally, when comparing the performance of these algorithms in terms of false positive alarm and true positive alarm, it is found that all Divergence Measures present a TPR of 100 %, but PD presents the lowest FPR of 3.8%. it can be concluded that PD outperforms both measures (HD &  $x^2$ ) for anomaly detection with the lowest FPR.

In conclusion, the work has achieved in outperforming the existing detection solutions by tuning the Power Divergence with  $\beta = 2.2$ . With this value of  $\beta$ , the detection scheme is leading to very attractive performance in terms of True Positive Rate (100%), False Positive Rate (3.8%) and is capable of detecting low intensity attacks.



## 7.2. Future Work

Through this work, several aspects of the design and development of the MA have been addressed. This work has also come across several points which require further investigation, for example:

- The setup of the protection proposition could also be used in the future to address the issue of denial of service (DOS) to some extent, with respect to mobile agent computation. To send the information back, the mobile agent has so far accumulated at its active storage element as last minute retaliation by the trusted server based on some kind of timing mechanism if the TS hasn't heard about the MA for quite a large amount of time.
- Include a mechanism in the protection approach to flag malicious hosts when detected so as to be avoided by successive agents. This would reduce latency.
- For the detection of attacks in mobile agent network, a focus on providing additional information to pinpoint malicious flows will be considered, in order to trigger automatic reaction against ongoing attacks. It is also intended to provide a method for reducing the amount of monitoring data on high speed networks, and analysing the impact of sampling on the precision of this divergence measure need to be developed.
- Develop an estimation freeze scheme that can protect the HD threshold estimation from being impacted by the attacks. A side benefit of the estimation freeze scheme is that the duration of attacks can be traced.
- Aiming to develop a software based on these two approaches and implement it in real time monitoring system, for a network in an enterprise since the scenarios considered in this work can be related to any type of networks.



# Bibliography

---

- Abdurrazaq, M.N., Bambang, R.T. and Rahardjo, B. (2014). *Distributed Intrusion Detection System Using Cooperative Agent Based on Ant Colony Clustering*. International Conference on Electrical Engineering and Computer Science (ICEECS), Kuta, 24-25 November 2014, 109-114. <http://dx.doi.org/10.1109/ICEECS.2014.7045229>.
- Alkasasbeh, M., & Adda, M. (July 2009). *Network Fault Detection with Wiener filter-based Agent*. Journal of Network and Computer Applications 32(4) (4):824-833.
- Alfalayleh, M., & Brankovic, L. (2005). *An Overview of Security Issues and Techniques in Mobile Agents*. 8th IFIP TC-6 TC-11 Conference on Communications and Multimedia Security.
- Altmann, J., Gruber, F., Klug, L., Stockner, W. & Weippl, E. (2001). *Using Mobile Agents in Real World: A Survey and Evaluation of Agent Platforms. Infrastructure for MAS, and Scalable MAS*. Montreal, Canada.
- Akomolafe, O. & Honesty, A. (2017). *Protecting Mobile Agent using Enhanced Reference Monitor based Security Framework*. International Journal of Computer Applications (0975 – 8887) Volume 161.
- Anna, L.D., Matt, B., Reisse, A., Vleck, S., Schwab, & LeBlanc, P. (June 2003). *Self-Protecting Mobile Agents Obfuscation Report*. Network Associates Laboratories.
- Armoogum, S. & Cully, A. (2011). *Obfuscation Techniques for Mobile Agent Code Confidentiality*. Journal of Information & Systems Management Vol.1 Number 1.
- Bace, R. G. (2000). *Intrusion detection: Sams*.
- Base, R., & Mell, P. (2001). *Intrusion Detection Systems. National Institute of Standards and Technology (NIST). Special Publication, vol. 51, pp. 800-831*.
- Bieszcad, A., Pagurek, B., & White, T. (1997). *Mobile Agents for Network Management*. IEEE Communications Surveys, 1, pp 2-9.
- Bishop, M. (October 2004). *Introduction to Security Network*. Addison Wesley, 1st edition.
- Broniatowski, M., & Leorato. (July, 2006). *An Estimation Method for the Neyman Chi-square Divergence with Application to test of Hypotheses*. J. Multivar. Anal. pp. 1409-1436.
- Carzaniga, A., Picco, G., & Vigna, G. (2007). *Designing Distributed Applications with Mobile Code Paradigms*. International Conference on Software Engineering, 19, pp 22-33.
- Chein-Yi, C., Lee, & Wen. (2010). *Semi-supervised learning for false alarm reduction*. Springer-Verlag Berlin Heidelberg, pp.595-605
- Chen, B., Cheng, H., & Palen, J. (2009). *Integrating Mobile Agent Technology with Multi-agent Systems for Distributed Traffic Detection and Management Systems*. Transportation Research Part C: Emerging Technologies 17, no. 1 1-10.
- Chen, E. (2006). *Detecting DoS Attacks on SIP Systems*. 1st IEEE Workshop on VoIP Management and Security.
- Collberg, C., Thomborson, C. & Low, D. (1997). *A Taxonomy of Obfuscating Transformations*. Technical Report 148, Department of Computer Science, University of Auckland.

- Cormode, G. & Muthukrishnan, S. (2004). *An Improved Data Stream Summary: The count-min sketch and its applications*. J. Algorithms, vol. 55, pp. 29–38.
- Deeter, K., Wilson, S., & Vuong, S. *APHIDS: A Mobile Agent-based Programmable Hybrid Intrusion Detection System*. International Workshop on Mobility Aware Technologies and Applications MATA'04, Florianopolis, Bresil, 2004, pp 244–253.
- Dagiukldz, T., Markl, J., & Rokos, M. (2006). *Low Cost Tools for Secure and highly Available Voip Communication Services*. Snocer 2.
- Denning, D. (1987). *An Intrusion Detection Model*. IEEE Transactions on Software Engineering, 13(2) pp 222– 232.
- Duraipandian, M., & Palanisamy, C. (2014). *An intelligent agent based defense architecture for ddos attacks*. In Electronics and Communication Systems (ICECS), 2014 International Conference on, pp 1–7.
- Ehlert, S., Wang, C., Magedanz, T., & Sisalem, D. (2008.) *Specification based Denial-of-Service Detection for SIP Voice-over-IP Networks*. the Third International Conference on Internet Monitoring and Protection.
- Eid, M., Artail, H., Kayssi, A. & Chehab, A. (2008). *A Lightweight Adaptive Mobile Agent based Intrusion Detection System*. International Journal of Network Security, 6(2), pp 145–1570.
- Esfandi, A., & Movaghar, A. (2009). *Mobile Agent Security in Multi Agent Environments Using a Multi Agent-Multi key Approach*. 2<sup>nd</sup> IEEE International Conference on Computer Science and Information Technology, pp. 438-442.
- Esparza, O., Fernandez, M., & Soriano, M. (2003). *Protecting Mobile Agents by Using Traceability Techniques*. IEEE International Conference on Information Technology, pp.264-268.
- Farid, D., & Rahman, M. (2010). *Anomaly network intrusion detection based on improved Self-adaptive Bayesian Algorithm*. Journal of computers, Vol. 5(1), Jan. pp. 23–31.
- Farmer, W.M., Guttman, J.D., & Swarup, W. (1996). *Security for mobile agents: Authentication and state appraisal*. 4th European Symposium on Research in Computer Security (ESORICS), pp. 118-130, Rome, Italy, September. Springer-Verlag Lecture Notes in Computer Science No. 1146.
- Fengxiang, Z., & Shunji, Z. (2006). *DoS/DDoS Attacks Detection Scheme Based on In/Out Traffic Proportion*. Information and Communication Engineers, 105, pp 7-11.
- Forgy, C. (1982). *A Fast Algorithm for the Many Pattern/many Object Pattern Match Problem*. Artificial Intelligence, 19(1), pp 17–37.
- Gavalas, D., Greenwood, D., & Ghanbari, M. (2003). *Advanced Network Monitoring Applications based on Mobile/intelligent Agent Technology*. Computer Communications, 23, pp 720-730.
- Gray, R., Kotz, D., Cybenko & Rus, G. (1998). *Security in a Multiple Language, Mobile Agent Systems*. LNCS 1419. Springer-Verlag.
- Graesser, S. F. (2006). *Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents*. Retrieved from <http://www.msci.memphis.edu/~franklin/AgentProg.html>
- Griffel, M., & Lamersdorf, W. (2003). *Integration of Intelligent and Mobile Agent for E-Commerce*.
- Gupta, R., & Gaurav, K.(2011). *A Survey on Comparative Study of Mobile Agent Platforms*. International Journal of Engineering Science and Technology (IJEST).

- Hachez, G. (2003). *A Comparative Study of Software Protection Tools Suited for Ecommerce with Contributions to Software Watermarking and Smart Cards*. Universite Catholique de Louvain.
- Hacini, S. (2013). *A Survey of Self-protected Mobile Agents*. International Journal of Computer Applications (0975 – 8887) Volume 61– No.19, January
- Hausssler, D., & Opper, M. (1997). *Mutual Information, Metric Entropy, and Cumulative Relative Entropy Risk*. Ann. Statist. vol. 25, pp. 2451–2492.
- Havrda, J., & Chavrat, F. (1967). *Quantification Method of Classification Processes: The concept of structural  $\alpha$ -entropy*. Kybernetika, vol. 3, pp. 30–35.
- Hohl, F. (1998). *Time Limited Blackbox Security: Protecting Mobile Agents from Malicious Hosts*. G. Vigna, editor, Mobile Agents and Security, LNCS, Springer-Verlag, Vol.1419, pp.92-113.
- HU, J.P., Zhi-Xin, L., & WANG, J.H. (2013). *Estimation, Interovention and Interaction of Multi-agent Systems*. Acta Automatica Sinica 39, no. 11 pp 1796-1804.
- Ionita, I., & Ionita, L. (2013). *An Agent-Based Approach for Building an Intrusion Detection System*. RoEduNet International Conference 12th Edition on Networking in Education and Research, Iasi, 26-28 September 2013, 1-6. <http://dx.doi.org/10.1109/RoEduNet.2013.6714184>
- Jansen, W. (November 2000). *Countermeasures for Mobile Agent Security*. Computer Communication. Special issue on Advances in Research and Application of Network Security.
- Jansen, W., & Karygiannis, T. (1999). *Mobile Agent Security*. National Institute of Standards and Technology, Gaithersburg, MD 220899.
- Jha, R. (2002). *Mobile Agents for e-commerce*. M. Tech. Dissertation, IIT Bombay, India.
- Karnik, N. (1998). *Security in Mobile Agent Systems*. PhD thesis, University of Minnesota
- Kowalczyk, R., Ulieru, M., & Unland, R. (2003). *Integrating Mobile and Intelligent Agents in Advanced e-Commerce: A Survey* Agent-Oriented Information Systems. Volume 3030 of the series Lecture Notes in Computer Science pp 45-60.
- Lange, D., & Oshima, M. (September 1998). *Mobile Agents with Java: The Aglet API*. Volume 1, Issue 3, pp 111–121.
- Lange, D.B & Oshima, M. (1997). *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesley Reading, Mass.
- Liotta, A. (2001). *Towards Flexible and Scalable Distributed Monitoring with Mobile Agents*. degree of Doctor of Philosophy Department of Computer Science, University of London.
- Manzoor, U., & Nefti, S. (May 2012). *iDetect: Content Based Monitoring of Complex Networks using Mobile Agents*. Applied Soft Computing, Volume 12, Issue 5, pp 1607-1619.
- Manzoor, U., & Nefti, S., & Rezgui, Y. (May 2013). *Categorization of Malicious Behaviours using Ontology-based Cognitive Agents*. Data & Knowledge Engineering. Volume 85, pp 40-56.
- Michelle, S., & Obelheiro, R. (2013). *A Security Scheme for Agent Platforms in Large-Scale Systems*. IFIP International Conference on Communications and Multimedia Security Mobile. pp 104-116.

- Mirkovic, J., & Reiher, P. (2005). *D-WARD: a source-end defense against flooding Denial-of-Service attacks*. IEEE Transactions dependable and secure computing, vol.2, no.3, pp. 216-232.
- Mohamed, T.A. (2012). *Generate Sub-Agent Mechanism to Protect Mobile Agent Privacy*. IEEE Symposium on Computers & Informatics, pp.86-91.
- Moore, D., Voelker, G. M., & Savage, S. (2001). *Inferring Internet Denial-of-Service Activity*. Proceedings of USENIX Security Symposium (SSYM'01). pp. 9-22.
- Nassar, M., State, R., & Festor, O. (May 2007). *Voip Honeypot Architecture*. Integrated Network Management (IM 2007). pp 109-118. IEEE, Munich.
- Nikaein, N. (1999). *Reactive Autonomous Mobile Agent*.
- Ngereki, A.M & Kahonge, A.M. (2015). *A MultiFaceted Approach to Mobile Agent Security*. International Journal of Computer Applications, vol. 120, no. 21. pp. 20-25
- Northcutt, S. (1999). *Network Intrusion Detection: An Analysts' Handbook*. Second Edition, New Riders Publishing.
- Noordende, G., Van't, J., Frances, M. T., Brazier, A., & Tanenbaum, S. (2004). *Security in a Mobile Agent System*. IEEE Symposium on Multi-Agent Security and Survivability.
- Pham, V.A., & Karmouch, A. (1998). *Mobile software agents: an overview*. Communications Magazine, IEEE, 36, 26-37.
- Pleisch, S. & Schiper, A.E. (2004a). *Approaches to Fault-Tolerant and Transactional Mobile Agent Execution-An Algorithmic View*. Computing Surveys, 36, 219-262.
- Pleisch, S. & Schiper, A.E. (2004b). *Approaches to Fault-Tolerant and Transactional Mobile Agent Execution – An Algorithmic View*. Pages 219-262.
- Rahwan, T., Rahwan, T., Rahwan, I., & Ashri, R. (YEAR). *Agent-based Support for Mobile Users using AgentSpeak*. Agent-Oriented Information Systems Volume 3030 of the series Lecture Notes in Computer Science pp 45-60.
- Ran, Z. (2012). *A Model of Collaborative Intrusion Detection System Based on Multi-Agents*. International Conference on Computer Science & Service System (CSSS), Nanjing, August 2012, pp 789-792.
- Richard, S. & Lipperts, G. (2002). *Mobile Agent Support Services, Department of Mathematic, Informatics and Nature Sciences*. Rhenisch-westflischen technischen hochschule Achen.
- Rathie, P.N., & Kannappan. P. (1972). *A directed-divergence function of type  $\beta$* . Inform. Contr. vol. 20, pp. 38-45.
- Riordan, J., & Schneier, B. (1998). *Environmental Key Generation towards Clueless Agents*. G. Vinga, editor, Mobile Agents and Security, LNCS, Springer-Verlag, Vol.1419.
- Reilly, D. (1998). *Mobile Agents-Process migration and its implications*. Retrieved from <<http://www.davidreilly.com/topics/softwareagents/mobileagents>>
- Roesch, M. (1999). *Snort- Lightweight Intrusion Detection for Networks*. LISA '99 Proceedings of the 13th USENIX conference on System administration, pp. 229-238
- Sahota, R. (2012). *An Overview of Security Techniques to Protect Mobile Agent from Malicious Host*. International Conference on Computing and Control Engineering, 12 & 13 April.

- Salem, O., Vaton, S., & Gravey, A. (2009). *A novel approach for anomaly detection over high – speed networks*. Proceedings of the 3rd European Conference on Computer Network Defense (ECND'07). vol. 30, pp. 49–68.
- Saidi, A., Bendriss, E.M., Kartit, A. & ElMarraki, M. (2015). *A Mobile Agent System to Enhance DoS and DDoS Detection in Cloud Computing*. European Journal of Scientific Research. Volume 131 No 2. April.
- Saidi, A. Bendriss, E.M., Kartit, A., & ElMarraki, M. (2017). *Techniques to Detect DoS and DDoS Attacks and an Introduction of a Mobile Agent System to Enhance it*. Cloud Computing, Special Issue on Advances and Applications in the Internet of Things and Cloud Computing.
- Sandya, A. (2015). *Mobile Agent Security using Reference Monitor-based Security Framework*. Nineth International Conference on Emerging Security Information Systems and Technologies.
- Scarfone, K., & Mell, P. (2007). *Guide to Intrusion Detection and Prevention Systems (Idps)*. NIST Special Publication, vol. 8, pp. 800-894
- Sander, T., & Tschudin, C.F. (1998). *Protecting Mobile Agents Against Malicious Hosts*. G. Vigna (Ed.), *Mobile Agents and Security*, Springer-Verlag, pp. 44-60.
- Sengar, H., Wijesekera, D., & Jodia, S. (2008). *Detecting VOIP Floods Using the Hellinger Distance*. IEEE, Vol.19.
- Silva, L., Soares, G., Martins, P., Batista, V., & Santos, L. (2000). *Comparing the performance of mobile agent systems: a study of benchmarking*. Journal Computer Communications archive. Volume 23 Issue 8, pp. 769-778
- Spafford, E., & Zamboni, D. (2000). *Intrusion detection using autonomous agents*. Computer Networks 34, pp 54-570
- Sun: Java 2 SDK security documentation. (2003).
- Taneja, I.J. (2006). *Bounds on Triangular Discrimination, Harmonic Mean and Symmetric Chi-Square Divergences*. Journal of Concrete and Applicable Mathematics, vol. 4, no. 1, pp. 91–111.
- Tang, J., Cheng, Y., & Zhou, C. (2009). *Sketch-based sip flooding detection using hellinger distance*. Proceedings of the 28th IEEE conference on Global telecommunications (GLOBECOM'09). pp. 3380–3385.
- Timcenko, V.V. (2014). *An Approach for DDoS Attack Prevention in Mobile ad hoc Networks*. Elektronika Ir Elektronika, ISSN 1392–1215, VOL. 20, NO. 6.
- Thorup, M., & Zhang, Y. (2004). *Tabulation Based 4-Universal Hashing with Applications to Second Moment Estimation*. The Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms.
- Trillo, R., Ilarri, S., & Mena, D. (2007). *Comparison and Performance Evaluation of Mobile Agent Platforms*. Third International Conference on Autonomic and Autonomous Systems
- Vigna, G. (1998). *Mobile Agents and Security*. Springer-Verlag London, UK.
- Vigna, G. (1997). *Protecting Mobile Agents through Tracing*. Proceedings of the 3rd ECOOP Workshop on Mobile Object Systems, Jyväskylä, Finland.
- Watson, M. (1997). *Intelligent Java Applications for the Internet and Intranets*. Morgan Kaufmann San Francisco, Calif.

- Wayne, & Jansen, A. (2000). *Countermeasures for Mobile Agent Security*. *Computer Communications* 23 (17), 1667-1676, pp. 1-14.
- Wroblewski, G. (2002). *General Method of Program Code Obfuscation*. PhD Dissertation, Wroclaw University of Technology, Institute of Engineering Cybernetics.
- Zubair, M., & Manzoor, U. (2016). *Mobile Agent based Network Management Applications and Fault-Tolerance Mechanisms*. The Sixth International Conference on Innovative Computing Technology (INTECH 2016)

## Appendix A - Publications

As a result of this work, the following papers have been presented at the conferences and published in the proceedings:

- C.1 Jean Tajer, Mo Adda, Benjamin Aziz (2017) "Comparison Between Divergence Measures for Anomaly Detection of Mobile Agents in IP Networks" at AIRCSSE - International Journal Wireless & Mobile Networks (IJWMN)
- C.2 Jean Tajer, Mo Adda, Benjamin Aziz (2017) "Detection of Flooding Attacks on Mobile Agents Using Sketch Technique and Divergence Measures" at International Journal of Engineering Sciences & Research Technology (IJESRT)
- C.3 Jean Tajer, Mo Adda, Benjamin Aziz (2017) "Flooding Attacks Detection of Mobile Agents in IP Networks" presented at 4<sup>th</sup> International Conference on Computer Networks & Communications 2017 (CCNET)
- C.4 Jean Tajer, Mo Adda, Benjamin Aziz (2017) "New Computing Model for Securing Mobile Agents in IP Networks" presented at 2<sup>nd</sup> International Conference on Internet of Things, Big Data and Security 2017 (IoT BDS)



# Appendix B – Experiment Environment

---

This section describes the laboratory setup and the software used for the thesis.

The hardware and software are listed below:

- **PC Router:**

A PC with two network interface cards are installed. The overall information is as follows:

Processors	: Intel (R) Pentium (R) 4 CPU 2.80GHZ
Memory	: 8 GB
Network adapters	: Intel® PRO/100 VE Network Connection
Routing protocol	: RIP
OS	: Windows server 2003

- **PC (Virtualised)**

The experiment deployed one computer running on virtualised machine. The overall information about them is as follows:

Processors	: Intel (R) Pentium (R) 4 CPU 2.80GHZ
Memory	: 8 GB
Network adapters	: Intel® PRO/100 VE Network Connection
OS	: Windows XP & VMWARE

- **Switch :**

Cisco Systems Catalyst 2900 Series XL 24 Port Switch 24-Port 10 Base T / 100 Base X.

Computer A is considered to act as trusted server (TS) and computer B runs many host nodes simulated through various port numbers as well as the home node in a virtualized mode

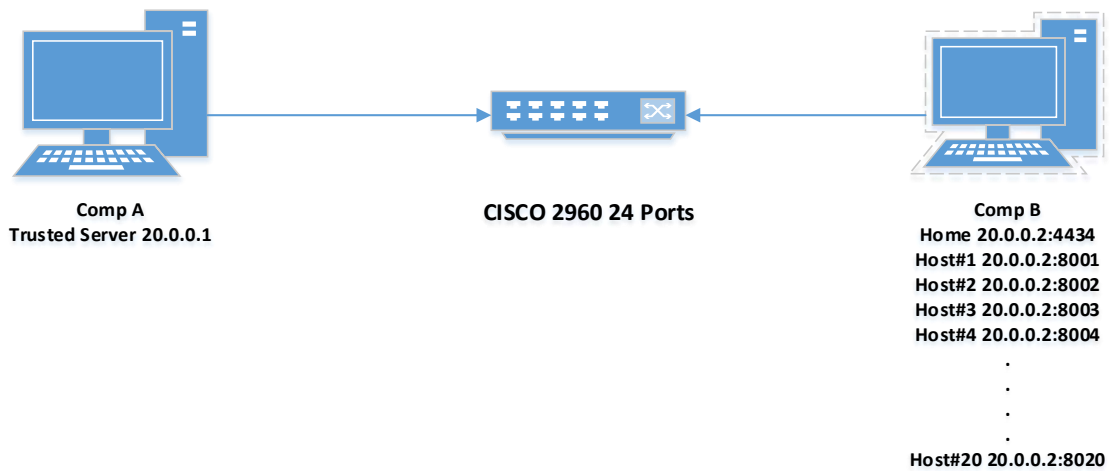


Figure B-1 Test Environment Set Up

- **The software used in the experiments works are described below:**
  - 1- Wireshark Network Packet Analyser will be running regularly over computer A. its job is to capture, sniff packets in a network and store them as shown in below Figure B-2<sup>10</sup>

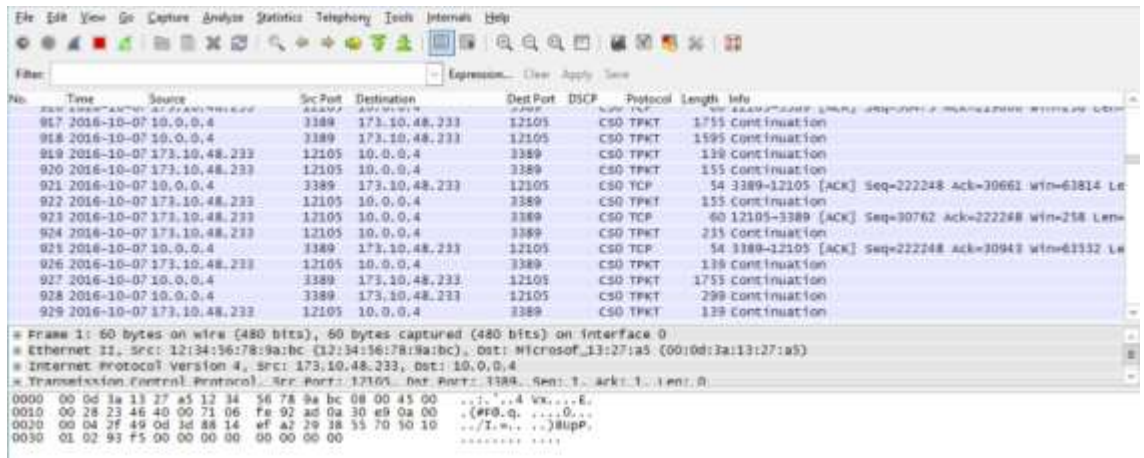


Figure B-2 Wireshark

- 2- Hyenae<sup>11</sup> is a highly flexible platform independent network packet generator. It allows you to reproduce several MITM, DoS and DDoS attack scenarios. It

<sup>10</sup> <https://sourceforge.net/projects/hyenae/>,2016

<sup>11</sup> <https://sourceforge.net/projects/hyenae/>,2016

can be used simultaneously on multiple PCs to follow a Distributed Denial of Service attack. As shown in below Figure B-3, to start the flooding only a few options are required, such as the Target URL and the socket number (<https://sourceforge.net/projects/hyenaef/>)

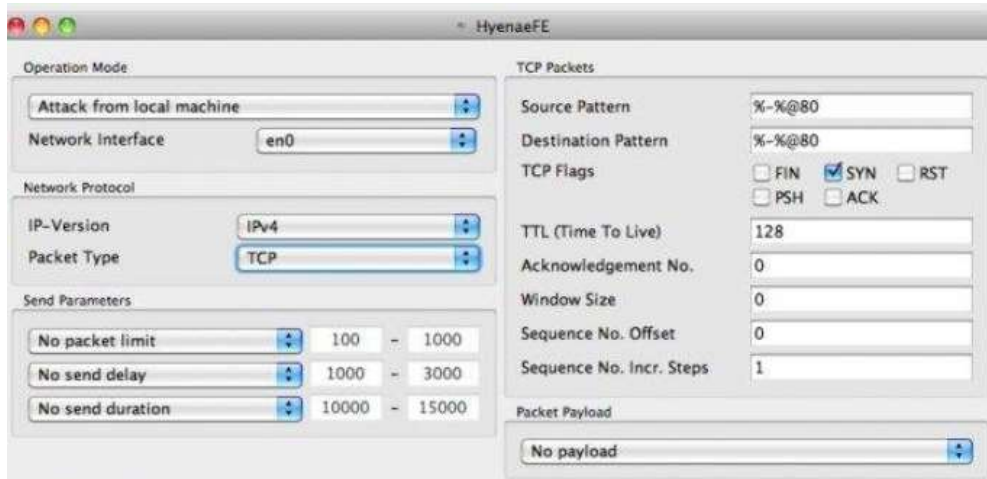


Figure B-3 Network Packet Generator

# Appendix C - Aglets Installation and Configuration Instructions for Windows

---

To setup and run aglets successfully one needs an Aglets Software Development Kit (ASDK). Installing the Aglet WorkBench also required the installation of JDK SDK1.1 or later which in this case was J2SDK1.4.2\_09.

Operating Systems considered for the installation guide are;

- Windows XP Professional
- Windows 2000 Professional

## Step1

Download and install the following packages from the internet;

- Aglets-2.0.2
- J2SDK1.4.2\_09

## Step2

Setting up the AGLET\_HOME and PATHS environment variables;

The following sequence of the installation guides need to be done in the command prompt.

*AGLET\_HOME*

*Set the AGLET\_HOME variable to the directory in which you installed the Aglets. It should look as follows if it is at the root directory of C:\;*

```
C:\> set AGLET_HOME=C:\AWB\Aglets
```

*PATH variables are set to include the subdirectory of the Aglet WorkBench(now AGLET\_HOME). It should look as follows;*

```
C:\> set PATH=%AGLET_HOME%\bin;%PATH%
```

**Step3**

Setting up environmental variables for Aglet program compilation; (CLASSPATH, AGLET\_PATH, AGLET\_EXPORT\_PATH)

## III.CLASSPATH

The CLASSPATH variable locates the directory of the aglet library ('lib'). Below is the command;

```
C:\> set CLASSPATH=%AGLET_HOME%\lib;%CLASSPATH%
```

## AGLET\_PATH

No codebased aglets uses AGLET\_PATH as a default path when Aglets API is created on the AgletContext class as an argument with codebase. Set it as follows;

```
C:\> set AGLET_PATH=C:\aglets\public
```

## AGLET\_EXPORT\_PATH

The AGLET\_EXPORT\_PATH variable locates files remotely for retrieval and referencing purposes. It can be set as follows;

```
C:\> set AGLET_EXPORT_PATH=%AGLET_PATH%
```

After this installation procedure, install JAVA\_HOME before TAHITI server installation. Set as follows;

```
C:\>set JAVA_HOME=C:\j2sdk1.4.2_09
```

Installing Tahiti Server:

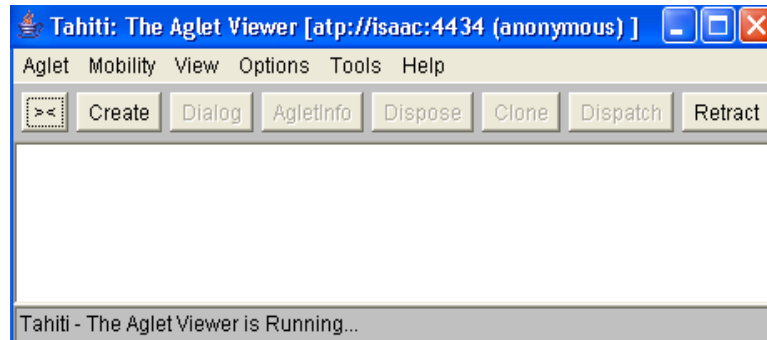
The following instructions need to be done at the command prompt:

Go to the bin directory of aglets (C:\>Aglets\bin) at the command prompt.

Type *ant*

Right after that run *ant install-home*

Typing *agletsd -port 4434* at the 'C:\>Aglets\bin' prompt brings up the Tahiti server as shown below;



The default port for this server is 4434.

# Appendix D – Installation Providers in Aglet

---

## INSTALLING PROVIDERS

Installing a provider is done in two steps: Installing the provider package classes and configuring the provider.

### Installing the Provider Classes:

First make sure that the provider classes are available so that they can be found when requested.

Provider classes are shipped as a JAR (Java ARchive) file.

To install the provider classes, install the JAR file containing the provider classes as an “installed” or “bundled” extension.

The provider JAR-file will be considered as an installed extension if it is placed in the standard place for the JAR files of an installed extension:

```
<java-home>/lib/ext
```

Where <java-home> refers to the directory where the runtime software is installed, which is the top level directory of the Java 2 Runtime Environment (JRE) or the jre directory in the Java 2 SDK (Java 2 SDK) software.

### Configuring the Provider:

The next step is to add the provider to a list of approved providers. This is done statically by editing the security properties file:

```
<java-home>\lib\security\java.Security
```


For each provider, this file should have a statement of the following form:

```
security.provider.n=masterClassName
```

This declares a provider and specifies its preference order *n*. The preference order is the order in which providers are searched for requested algorithms when no specific provider is requested.

The order is 1-based; 1 is the most preferred, followed by 2, and so on.

*masterClassName* must specify the fully qualified name of the provider's "master class". The provider vendor should supply you this name.



## Appendix E – Aglet Code

---

### AgentApp.java

```
/*
 * This program is a mobile agent based application,prototype,
 * that sends a mobile agent to the agent space to perform some
 * task
 */
import com.ibm.aglet.*;
import java.security.*;
import javax.crypto.SecretKey;
import javax.crypto.SealedObject;
import java.net.URL;
import java.util.Iterator;
import java.util.ArrayList;
import java.util.Date;
import java.awt.event.*;
import javax.swing.*;
import java.security.cert.Certificate;
import java.security.*;
import javax.crypto.*;
import java.io.*;
import javax.crypto.spec.SecretKeySpec;

public class AgentApp extends Aglet implements ActionListener
{
    public mobileUIF mobui ;
```



```

public JMenuItem crtMenuI , DispaMenuI , setinMenuI;
public ArrayList clrKeys=null;
public ArrayList clrInfos=null;
public ArrayList abahsts=null;

public ArrayList abavhms=null;
public SignedObject dsDstn=null;
public Destn dstn;
public PrivateKey prvk;
public String agletName = "mobileA";
public String alias="kprvirtual";
public String password="virtual";
public long send =0;
public long arrived =0;
public long lived =0;
public proxyH prxy;
public URL url;
public int pt;
public String prt;
public void onCreate(Object init)
{
try{
prxy = new proxyH(this);
}catch(Exception ex){
System.out.println("The following error occurred" +
ex.toString());
ex.printStackTrace();
}
}

```

```

abavhms = new ArrayList();
mobui = new mobileUIF() ;
crtMenuI = mobui.getCreateMI();
crtMenuI.addActionListener(this);
DispaMenuI = mobui.getDispatchMI();
DispaMenuI.addActionListener(this);
setinMenuI = mobui.getSettingMI();

setinMenuI.addActionListener(this);
clrKeys=new ArrayList();
clrInfos=new ArrayList();
url=getAgletContext().getHostingURL();
pt=url.getPort();
prt=Integer.toString(pt);
mobui.show();
}

public boolean handleMessage(Message msg)
{
try{
arrived = new Date().getTime();
lived = arrived - send;
prvk=CipherCls.getPrivateKey(alias,
password.toCharArray(),prt);
Iterator kitr=((ArrayList)msg.getArg("keys")).iterator();
Iterator resitr=((ArrayList)msg.getArg("infos")).iterator();
while (kitr.hasNext())
{
SealedObject sldKey=(SealedObject)kitr.next();

```

```

    SecretKey secKey=CipherCls.unsealSecretKey(sldKey, prvk);
    clrKeys.add(secKey);
    SealedObject sldRes=(SealedObject)resitr.next();
    Object res = new Object();
    res=CipherCls.unsealObject(sldRes, secKey);
    clrInfos.add(res);
}
}catch(Exception ex)
{System.out.println("The following error occurred" +
ex.toString());
ex.printStackTrace();
return false;
}
infoDlg inf= new infoDlg(mobui ,"Information Dialog Box",
abahsts ,clrInfos ,lived);
return true;
}
public void actionPerformed(ActionEvent event)
{
    Object source = event.getSource();
    if (source == crtMenu1)
    {
        pathDlg pdlg = new pathDlg(mobui , "Enter The Address of All
Nodes To Be Visited");
        pdlg.show();
        abahsts = (ArrayList)pdlg.getHsts();
        abavhms = (ArrayList)pdlg.getVHm();
        return;
    }
}

```

```

}
else if (source == DispaMenuI)
{
try{
dstn = new Destn(abahsts , abavhms);
prvk=CipherCls.getPrivateKey(alias,
password.toCharArray(),prt);
dsDstn = CipherCls.signObject(dstn,prvk);

Object[] pass = {prxy,dsDstn};
send = new Date().getTime();
getAgletContext().createAglet(getCodeBase(), agletName, pass);
mobui.hide();
}catch(Exception ex)
{
System.out.println("The following here here exception types
captured: " + ex.toString());
return;
}
return ;
}
return;
}
}

```

## mobileA.java

```
/*
 * This program is the proposed mobile agent implementation
 * it carries out its task as pointed out in the proposal
 */
import com.ibm.aglet.*;
import com.ibm.aglet.event.*;
import java.security.SignedObject;
import java.security.PublicKey;
import javax.crypto.SealedObject;
import javax.crypto.SecretKey;import java.net.URL;
import java.net.InetAddress;
import java.util.Vector;
import java.util.ArrayList;
import java.util.Date;
public class mobileA extends Aglet implements MobilityListener
{
public SignedObject dsDstn;
public String alias="certvirtual";
public int startTrip=1;
public String agletName="statA" ;
public AgletProxy slave;
public Message msgInfo= null;
public proxyH prxy;
public URL url;
public int pt;
public String prt;
```

```

public void onCreate(Object init)
{
try{
Object[] axcptd=(Object [])init;prxy=(proxyH)axcptd[0];
dsDstn=(SignedObject)axcptd[1];
Destn dstn=null;
url=getAgletContext().getHostingURL();
pt=url.getPort();
prt=Integer.toString(pt);
dstn=(Destn)CipherCls.getSignedObject(dsDstn,
CipherCls.getPublicKey(alias,prt));
addMobilityListener(this);
dstn.goToNext(this);
}catch(Exception ex)
{
System.out.print("The following error occurred" +
ex.toString());
ex.printStackTrace();
}
}

public void onArrival(MobilityEvent event)
{
Destn dstn=null;

try{
url=getAgletContext().getHostingURL();
pt=url.getPort();prt=Integer.toString(pt);
dstn=(Destn)CipherCls.getSignedObject(dsDstn,
CipherCls.getPublicKey(alias ,prt));

```

```

if (dstn.isAtTS(this))
{
if(startTrip==1)
{
prxy.updateSProxy(getAgletContext().createAglet(getCodeBase(
), agletName, prxy));
startTrip=0;
dstn.goToNext(this);
}
else
{
Message msg= new Message("MobileBackV");
msgInfo=(Message)prxy.getSProxy().sendMessage(msg);
dstn.goBackHome(this);
}
}
else if (dstn.isAtHome(this))
{
prxy.getAProxy().sendOnewayMessage(msgInfo);
dispose();
}
else
{
PublicKey pubk=CipherCls.getPublicKey(alias ,prt);
SecretKey secKey = CipherCls.generateSessionKey();
SealedObject encKey=CipherCls.sealSecretKey(secKey,pubk);
String tmp1 = "OS Architecture : " +
System.getProperty("os.arch");
String tmp2 = " ; OS Version : " +

```

```
System.getProperty("os.version");
String result=tmp1 + tmp2;
SealedObject encRes=CipherCls.sealObject(result,secKey);
Message msg = new Message("MobieAway");
msg.setArg("keys", encKey);
msg.setArg("infos", encRes);prxy.getSProxy().sendOnewayMessage(msg);
dstn.goToNext(this);
}
}catch(Exception ex)
{
System.out.println("The following error ocured" +
ex.toString());
ex.printStackTrace();
}
}
public void onDispatching(MobilityEvent e){}
public void onReverting(MobilityEvent e){}
}
```



## statA.java

```
/*
 *This class implements a temporary storage place as pointed out in the proposal
 */
import com.ibm.aglet.*;
import com.ibm.aglet.event.*;
import javax.crypto.SealedObject;
import java.util.ArrayList;
public class statA extends Aglet
{
public ArrayList statRes;
public ArrayList statKeys;
public proxyH prxy;
public void onCreate(Object init)
{
try{
prxy = (proxyH)init;
statKeys=new ArrayList();
statRes=new ArrayList();
prxy.updateSProxy(getProxy());
}catch(Exception ex)
{
System.out.println(ex.toString());
}}
public boolean handleMessage(Message msg){
if (msg.sameKind("MobieAway"))
{
```

```

try{
SealedObject sos=(SealedObject)msg.getArg("keys");
statKeys.add(sos);
SealedObject sor=(SealedObject)msg.getArg("infos");
statRes.add(sor);
}catch(Exception ex)
{
System.out.print("The following error occurred"+
ex.toString());
ex.printStackTrace();
}
return true;
}
else if(msg.sameKind("MobileBackV"))
{
try{
Message newmsg = new Message("MobileH");
newmsg.setArg("keys", statKeys);
newmsg.setArg("infos", statRes);msg.sendReply(newmsg);
}catch(Exception ex)
{
System.out.print("The following error occurred"+
ex.toString());
ex.printStackTrace();
}
return true;
}
return false;}

```

# Appendix F - Sketch Technique and Detection Algorithm Codes

---

## Divergence Measures Code

```

clear all
close all
f=fopen('osmansalem','w');
% a = 40; b = 100;

%beta=[-1 -0.75 -0.5 -0.25 0.25 0.5 0.75 1.25 1.5 1.75 2 2.25 2.5 2.75 3 3.25 3.5 3.75 4 4.25
4.5 4.75 5 5.25 5.5 5.75 6]
%beta=2.25
%beta=[18:19]

% S=100
IScore=[]
JEAN=[]
%load ALL.dat
%P=ALI

A=1024
B=511

file_name='./jean1.dat'

%load -ascii ./2048/jean.dat
%NB_SYN = jean
fid = fopen (file_name,'r');
for j=1:B
for i=1:A
    NB_SYN(j,i) = fscanf(fid,'%e',1);
end
% % fscanf(fid,'%s',1);
end

for i=1:B
    for j=1:A

        P(i,j)= NB_SYN(i,j)./sum(NB_SYN(i,:));
    end
end

```

## APPENDIX F - SKETCH TECHNIQUE AND DETECTION ALGORITHMS CODES

```

%%%%%%%%%%
%%%%%%%%%%
%%%%%%%%%
%Chi-square

%%%%%%%%%%
%%%%%%%%%%
%%%%%%%%%

for i=1:size(P,1)-1
    for j=1:size(P,2)
        if (P(i+1,j) ~= 0 && P(i,j) ~= 0)
            Rapport(i,j) = (P(i+1,j)-P(i,j))^2/ (P(i,j));

        else Rapport(i,j) = 0;
        end
    end

end

for i=1:size(Rapport,1)
    X(i)=sum(Rapport(i,:));

    IScore=[IScore X(i)];
end

figure(1)
axes('FontSize',18)
plot(IScore,'LineWidth',1.5);

axis([1 509 0 250])
xlabel('Time(min)');
ylabel('Chi-square Divergence');

tajer=0
for makke=10:10
    tajer = tajer + 1;
    for i=1:length(IScore)-1
        diff(1,i)=IScore(i+1) - IScore(i);
    end
    ho=[]
    meo= mean(IScore(1:9));
    varo= var(IScore(1:9));
    ho=[ho meo+2*sqrt(varo)];
    distosman=IScore(1:9)
    for i=10:length(IScore)

```

```

meo= mean(distosman(1:(i-1)));
varo= var(distosman(1:(i-1)));

ho=[ho (makke*meo)+3*sqrt(varo)];
if (IScore(i) < ho(end))
    distosman(i)=IScore(i);
else
    distosman(i)=distosman(i-1)
end

end

%figure(tajer);
%plot(dists);
hold on
T=9:510
plot(T,ho,'r--','LineWidth',1.5)
legend('Chi-square Divergence','Threshold');
box on
print -depsc2 CS.eps
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Hellinger distance

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
HD1=[];
for i=1:size(P,1)-1
    for j=1:size(P,2)
        %if (P(i+1,j) ~= 0 && P(i,j) ~= 0)
            HD(i,j) = (sqrt(P(i+1,j))-sqrt(P(i,j)))^2;

        %else Rapport(i,j) = 0;
        end
    end

for i=1:size(HD,1)
    Y(i)=0.5*sum(HD(i,:));

    HD1=[HD1 Y(i)];
end

```

```

figure(2)
axes('FontSize',18)
plot(HD1,'LineWidth',1.5);

axis([0 508 0 0.5])
xlabel('Time(min)');
ylabel('Hellinger Distance');

tajer=0
for makke=1:1
tajer = tajer + 1;
    for i=1:length(HD1)-1
        diff(1,i)=HD1(i+1) - HD1(i);
    end
    ho=[]
    meo= mean(HD1(1:9));
    varo= var(HD1(1:9));
    ho=[ho meo+2*sqrt(varo)];
    distosman=HD1(1:9)
    for i=10:length(HD1)
        meo= mean(distosman(1:(i-1)));
        varo= var(distosman(1:(i-1)));

        ho=[ho (makke*meo)+3*sqrt(varo)];
        if (HD1(i) < ho(end))
            distosman(i)=HD1(i);
        else
            distosman(i)=distosman(i-1)
        end
    end

end

%figure(tajer);
%plot(dists);
hold on
T=9:510
plot(T,ho,'r--','LineWidth',1.5)
legend('Hellinger Distance','Threshold');
box on
print -depsc2 HD.eps
end

%%%%%%%%%%
%%%%%%%%%%
%%%%%%%%%%

%%%%%%%%%%
%%%%%%%%%%

```

APPENDIX F - SKETCH TECHNIQUE AND DETECTION ALGORITHMS CODES

```

%%%%%%%%%%
%%%%%%%%%%

%JEAN=[JEAN IScore']
%IScore=[]

%end
% [XX,YY]=meshgrid(beta, 1:99)
% mesh(XX,YY,JEAN)
% axis([5 6 0 100 0 10^13])

%
% hold on
% % apprentissage
%
% mBr=zeros(1,length(IScore)+1);
% Err=zeros(1,length(IScore)+1);
% S=zeros(1,length(IScore)+1);
% H=zeros(1,length(IScore)+1);
%
% mBr(1)=0
% mBr(2)= IScore(1);
% mBr(3)=mBr(2);
% Err(3)=abs(mBr(3)-IScore(3));
% S(3)=0.25*Err(2);
% H(3)=5*mBr(3)+S(3); % estimation
%
%
% for i=4:length(IScore)
%   mBr(i)=(7/8)*mBr(i-1)+(1/8)*IScore(i-1);
%   Err(i)=abs(mBr(i)-IScore(i));
%   S(i)=0.75*S(i-1) + 0.25*Err(i-1);
%   H(i)=2*mBr(i)+1*S(i);
%
% end
%
%
% plot(H,'r')

return

%PD2= [100 100 100 98 95 85 74 52 20 0 0 0 0 0 ]
PD1=[100 100 100 100 98 95 91 80 70 45 30 10 0 0 0 0 0 0 0 0]
FR1=[100 95 90 80 50 23 15 6 2 1 0 0 0 0 0 0 0 0 0 0]
x=0:2:18
figure(1)

```

```

plot(PD1,'r*-')
hold on
plot(FR1,'bo-')
plot(PD2,'bo-');
set(gca,'XTickLabel', 0:2:20)
xlabel('Threshold h');
ylabel('True positive/False positive');
legend('True positive','False detection');
FAR1=(FR1./100)
PDR1=PD1./100
figure(2)
plot(FAR1, PDR1,'ro-')
xlabel('False positive');
ylabel('True positive');
set(gca,'XTickLabel', 0:10:100)
legend('True positive');

%return
figure(3)
PD2=[100 100 98 95 91 82 75 66 48 29 14 5 2 0 0 0 0 0 0 0]
FR2=[100 80 50 15 5 2 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
x=0:2:18
%figure(1)
plot(PD2,'r*-')
hold on
plot(FR2,'bo-')
set(gca,'XTickLabel', 0:2:20)
xlabel('Threshold h');
ylabel('True positive/False positive');
legend('True positive','False detection');
FAR2=(FR2./100)
PDR2=PD2./100
figure(4)
plot(FAR2, PDR2,'ro-')
hold on
plot(FAR1,PDR1,'bo-');
xlabel('False positive');
ylabel('True positive');
set(gca,'XTickLabel', 0:10:100)
legend('True positive');

figure(5)
PD3=[85 85 82 80 77 65 55 46 31 10 2 0 0 0 0 0 0 0 0 0]
FR3=[100 100 90 85 77 55 40 36 22 18 13 13 11 9 8 7 5 5 5 5]
x=0:2:18
%figure(1)
plot(PD3,'r*-')
hold on
plot(FR3,'bo-')

```



```

set(gca,'XTickLabel', 0:2:20)
xlabel('Threshold h');
ylabel('True positive/False positive');
legend('True positive','False detection');
FAR3=(FR3./100)
PDR3=PD3./100
figure(6)
plot(FAR2, PDR2,'ro-')
hold on
plot(FAR1,PDR1,'bo-');
hold on
plot(FAR3,PDR3,'bo-');
xlabel('False positive');
ylabel('True positive');
set(gca,'XTickLabel', 0:10:100)
legend('True positive');

return
f=fopen('osmansalem','w');
for i=1:21

    fprintf(f,'%d\t %d \t %d \t %e \t %e \t %d \t %d \t %e \t %e \n', i-
1,PD1(i),FR1(i),PDR1(i),FAR1(i),PD2(i),FR2(i),PDR2(i),FAR2(i));
end
fclose(f);

```

## Attack Generator

```

include <iostream>
#include <string>
#include <cstdlib>
#include <cmath>
#include <fstream>
#include <sstream>
#include <iomanip>
#include <cstdio>
#include <stdlib> /* for double = atof() and atoi() functions */

#include <cstring>
#include <cmath>

//-----
// Main program
//-----

typedef struct
{

```

```

    int timestamp_second;
    int timestamp_float_part;
    int ipv4_source;
    int src_port;
    int ipv4_dest;
    int dest_port;
    int ip_len;
    int proto;
    int flags_tcp;
} oscarflow_id;

using namespace std;

int main(int argc, char *argv[])
{
//-----
// Initialisation des variables
//-----
char line[1500],line2[1500];
int retVal;
    double tsec, tusec,tfsec, inter_tsec;

    int ipv4_src[4], ipv4_dst[4];
    int sport,dport;
    int volume, fenetre,nseq, nack;
    char flags, c;
    int len, flag;
    int syn, fin,rst;
    oscarflow_id ofp;
    int nomber;
    int i;
int x, z;
    char proto;
string inputLine, stipv4_src, stipv4_dst; // Input line

    ifstream inputFile;
    ofstream outputFile;
    char filename[100];
    int interval_cnt=31 ;

string attack="1271300401.260961 222.199.15.145 10.0.0.1 18669 80 T S";

    ofstream attackfile;

for (z=1;z<=10;z++)
{
    sprintf(filename,"attack%d",interval_cnt);

```

```

attackfile.open(filename);

cout << setprecision(6) ;
    cout.setf(ios::fixed,ios::floatfield);

    for (x=1;x<15000;x++)

        attackfile << attack<< endl;

        attackfile.close();
interval_cnt++;

}
    attackfile.close();

return 0;
}

```

## Sketch

```

/*****/
/** NADA-CRESBALA V1.0 Gamma **/
/** Implementation of Counting REversible Sketch Based At Linear Algebra **/
/** as Network Anomaly Detection Algorithm **/
/** for DoS/DDoS Attack **/
/** Developed by Osman Salem, March 2007 **/
/** Copyright (c) 2007 ENST Bretagne. All rights reserved. **/
/** **/
/** The NADA-CRESBALA software package is distributed in open-source **/
/** under the terms of the GNU general public license agreement. **/
/** To document the usage of NADA-cresbala, we ask each user to cite our **/
/** scientific publications mentioned on our Web site. **/
/*****/

/*****
LAST Updated: Sunday 18/03/2007 Ã 5:06
This work is under evaluation phase, and its distribution is not allowed for
any organisation or any person without the permission of the author

```

Great thanks to Frederic grasset for his indenial help in the developpement of this code

```

and I wish that he had enjoy his black pizza
*****/

//-----
//Counting REversible Sketch Based At Linear Algebra
//-----

#include <iostream>
#include <stdio.h>
#include <stdlib.h>          /*for rand() and srand() */
#include <malloc.h>
#include <inttypes.h>
#include <string.h>
#include <unistd.h>
#include <math.h>
#include <time.h>
#include <limits.h>        /* for time() function */
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "includes/cresbala.h"
using namespace std;

const uint64_t prime = (((uint64_t)1)<<61)-1;
const uint64_t lowones = (((uint64_t)1)<<32)-1;

uint64_t low32of64(uint64_t x) { return(x&lowones); }
uint64_t high32of64(uint64_t x) { return(x>>32); }

uint64_t MultAddMod(uint64_t x, uint64_t a, uint64_t b) {
    // Compute (ax+b)%prime + possibly 2*prime using prime=2^61-1

    uint64_t a0 = low32of64(a)*x;
    uint64_t a1 = high32of64(a)*x;
    uint64_t c0 = a0+(a1<<32);
    uint64_t c1 = (a0>>32)+a1;
    uint64_t c = (c0&prime)+(c1>>29)+b;
    return(c);
}

/*****/
/* Reversible sketches based on linear algebra */
/*****/
cresbala * CRES_Init(uint64_t width, int depth, int U)
{
    cresbala * cm;
    int i,j,k;
    char t[100];
    FILE *fin[10], *infin;
    uint64_t a,b;

```

```

srand(time(NULL));
    if (U <= 0 || U > P) return(NULL);
cm=(cresbala *) malloc(sizeof(cresbala));
if (cm)
{
    cm->totale.totale = 0;
    cm->depth=depth;
    cm->width=width;
    cm->U=U;
    cm->counts=(counter **)calloc(cm->depth,sizeof(counter *));
        for (i=0;i < cm->width;i++)
        {
            cm->counts[i]=(counter *)calloc(cm->width,sizeof(counter));
        }
    cm->prob=(float **)calloc(cm->depth,sizeof(float *));
        for (i=0;i < cm->width;i++)
        {
            cm->prob[i]=(float *)calloc(cm->width,sizeof(float));
        }
    cm->hasha=(uint64_t *)calloc(cm->depth,sizeof(uint64_t));
    cm->hashb=(uint64_t *)calloc(cm->depth,sizeof(uint64_t));
    if (cm->counts && cm->hasha && cm->hashb && cm->counts[0])
    {
        for (j=0;j<depth;j++)
        {
            b((((uint64_t)rand())<<32)+((uint64_t)rand()))%prime;
            do {a((((uint64_t)rand())<<32)+((uint64_t)rand()))%prime;}
while(a == 0);
            cm->hasha[j]= a;
            cm->hashb[j]=((((uint64_t)rand())<<32)+((uint64_t)rand()))%prime;
            // pick the hash functions

        }
        for (i = 0; i < cm->depth; i++)
            for (j = 0; j < cm->width; j++)
            {
                cm->counts[i][j].X = 0;
                for (k = 0; k < 5; k++)
                    cm->counts[i][j].WIND[k]=0;
            }
        }
    else cm=NULL;
}
return cm;
}

// ***** /
/* Initial update function for sketch */
/***** /

```

```

void CRES_Update(cresbala *cm, uint64_t item, uint32_t differ)
{
    int i,j;
    uint64_t bjectv=0;
    uint32_t test;
    if (cm==NULL) return;
    cm->totale.totale += differ;

    /*char tchr[1500];
    FILE *fosman;
    sprintf(tchr,"sketchosman");
    fosman=fopen(tchr,"a");*/

    for (i = 0 ; i < cm->depth; i++)
    {
        //test = (MultAddMod(item,cm->hasha[i], cm-
>hashb[i])%prime)%(cm->width);
        //fprintf(fosman, "item=%llu \t hash=%u\n",item,test) ;
        cm->counts[i][(MultAddMod(item,cm->hasha[i], cm-
>hashb[i])%prime)%(cm->width)].X += differ;
    }
    //fprintf(fosman, "\n\n\n");
    //fclose(fosman);
return;
}

//*****
/* Initial update function for sketch */
//*****

void Calc_Proba(cresbala *cm)
{
    FILE *fileid1, *fd;
    fileid1=fopen("kldiv1.dat","a");
    fd = fopen("jean.dat","a");
    int i,j;
    for (i = 0; i < cm->depth; i++)
        for (j = 0; j < cm->width; j++)
        {
            if (cm->totale.totale)
                cm->prob[i][j]= (float)cm->counts[i][j].X/(cm->totale.totale);
            else printf("error\n");
        }
    fprintf(stdout,"%ld \t", cm->width);
}

```

```

for (j = 0; j < cm->width; j++)
    fprintf(fd,"%u ",cm->counts[1][j].X);
fprintf(fd,"\n");

fprintf(fileid1,"%lld \t", cm->totale.totale);
for (i = 0; i < cm->depth; i++)
    {
        for (j = 0; j < cm->width; j++)
            fprintf(fileid1,"%u \t", cm->counts[i][j].X);
        fprintf(fileid1,"\n");
    }
fclose(fileid1);
fclose(fd);

return;
}

void print_file(cresbala *cm)
{
    static int ik = 1;
    FILE *fileid1;
    char tch[1500];
    sprintf(tch,"./osman/blaise%03d",ik);
    fileid1=fopen(tch,"w");
    int i,j;
    for (i = 0; i < cm->depth; i++)
        {
            for (j = 0; j < cm->width; j++)
                {
                    fprintf(fileid1,"%u ",cm->counts[i][j].X);
                }
            fprintf(fileid1,"\n");
        }
    fclose(fileid1);
    ik++;
return;
}

int CRES_Est(cresbala * cm1, cresbala * cm2 )
{
    static int i1 = 0;
    static int osman=0;
    double R = 0, Q= 0;
    unsigned int X, compt;
    double alarm[32]={0};
    float KLDiv[5]={0};

```

```

FILE *fileid, *fileid1;
fileid=fopen("kldiv.dat","a");
fileid1=fopen("ALI.dat","a");
uint64_t key;
uint64_t i,j,k,t,t1, r, result = 0, bjectv = 0;
//double seuil[3], threshold_stat[3][3]={{150,125,100},{100,75,50},{100,75,25}};
uint64_t seuil[3];//1000000000ULL;
uint64_t thresh=UINT_MAX, threshold_stat[3];
node *Lghp[3];
if (!cm1 && !cm2) return 0;
srand(time(NULL));
int limit = 3;
i=rand()%limit;
compt = 0;

for (i = 0; i < cm1->depth; i++)
    {
        for (j = 0; j < cm1->width; j++)
            {
                R = cm1->prob[i][j];
                Q = cm2->prob[i][j];
                //R= sqrt(R);
                //Q= sqrt(Q);
                //fprintf(fileid,"R=%f \t Q=%f \t ", R,Q);
                if (R && Q)
                    {
                        KLDiv[i] += R*log(R/Q);// - Q*log10(R));
                        //KLDiv[i] += 0.5*pow(R-Q,2) ;

                    }
                /*else
                    KLDiv[i] += 0;*/

            }
        //fprintf(fileid,"\n");
    }
double totale_ali = 0;
for (j = 0; j < cm2->width; j++)
    {
        R= cm2->prob[1][j]; //(double)cm1->counts[1][j].X/(cm1-
>totale.totale);
        //if (R)
            {
                fprintf(fileid1,"%e ", R);
                totale_ali += R;
            }
    }

```



```

cout << "La somme totale de la ligne " << j << " = " << totale_ali << endl;
totale_ali = 0;
fprintf(fileid1, "\n");

    fprintf(fileid, "%d \t %lld \t %lld \t ", osman++, cm1->totale.totale, cm2-
>totale.totale);
    for (i = 0; i < cm1->depth; i++)
        fprintf(fileid, "%f \t ", KLDiv[i]);
    fprintf(fileid, "\n");
    fclose(fileid);

//exit;
    cm1->totale.totale = cm2->totale.totale;
    cm2->totale.totale = 0;
    for (i = 0; i < cm1->depth; i++)
        for (j = 0; j < cm1->width; j++)
            {

                cm1->counts[i][j].X = cm2->counts[i][j].X;
                cm2->counts[i][j].X = 0;
                cm1->prob[i][j] = cm2->prob[i][j];
                cm2->prob[i][j] = 0;

            }

return 0;

}

// ***** /
/* Initial update function for sketch */
// ***** /

/***** /
/* Free allocated memory for sketch */
/***** /
void CRES_Destroy(cresbala * cm)
{
    int i,j;
    if (!cm) return;
    for (i=0;i<cm->depth;i++)
        {
            free(cm->counts[i]);
        }
    free(cm->counts);
    cm->counts=NULL;
    free(cm->hasha);
}

```

```

    free(cm->hashb);
    cm->hasha=NULL;
    cm->hashb=NULL;
    free(cm);
    cm=NULL;
}
/*****
*****/
/* Duplicate sketch : // create a new sketch with the same parameters as an existing
one */
/*****
*****/
cresbala * CRES_Copy(cresbala * cmold)
{
    cresbala * cm;
    int i,j;
    if (!cmold) return(NULL);
    cm=(cresbala *) malloc(sizeof(cresbala));
    if (cm)
    {
        cm->totale.totale=cmold->totale.totale;
        cm->depth=cmold->depth;
        cm->width=cmold->width;
        cm->U=cmold->U;

        cm->counts=(counter **)calloc(sizeof(counter *),cm->depth);
        for (i=0;i<cm->depth;i++)
        {
            cm->counts[i]=(counter *)calloc(sizeof(counter ),cm->width);
        }
        cm->prob=(float **)calloc(cm->depth,sizeof(float *));
        for (j=0;j<cm->width;j++)
        {
            cm->prob[i]=(float *)calloc(cm->width,sizeof(float));
        }
        cm->hasha=(uint64_t *)calloc(cm->depth,sizeof(uint64_t));
        cm->hashb=(uint64_t *)calloc(cm->depth,sizeof(uint64_t ));

        for (i=0;i<cm->depth;i++)
            for (j=0;j<cm->width;j++){
                cm->counts[i][j].X=cmold->counts[i][j].X;
                cm->prob[i][j]=cmold->prob[i][j];
            }

        for (i=0;i<cm->depth;i++)
        {
            cm->hasha[i]=cmold->hasha[i];
            cm->hashb[i]=cmold->hashb[i];
        }
    }
}

```

```

    }
    else cm=NULL;

    return cm;
}

/*****
/* RESET counting sketch */
*****/
int CRES_Reset(cresbala *cm)
{
    int i,j;
    if (cm==NULL) return 0;
    cm->totale.totale = 0;
    for (i=0;i<cm->depth;i++)
        for (j=0;j<cm->width;j++){
            cm->counts[i][j].X= 0;
            cm->prob[i][j]= 0;
        }
    return 1;
}

```

## ROC

```

clear all
close all
figure(1)
axes('FontSize',18)

%Jensen-Shannon ROC
DR=[100 100 83 75 66 41 25 16 8 0]
FAR=[100 40 28 26 20 13 0 0 0 0]
plot(FAR,DR,'b--','LineWidth',4)

axis([0 100 0 105])
xlabel('False Alarm Rate');
ylabel('Detection Rate');

hold on
%beta=0.5, HD ROC
PD=[100 100 91 83 75 58 41 25 16 8 0]
FR=[100 43 35 23 18 12 8 0 0 0]
plot(FR,PD,'r:','LineWidth',4)

hold on

```

## APPENDIX F - SKETCH TECHNIQUE AND DETECTION ALGORITHMS CODES

```
%beta=1, KL ROC
PD=[100 100 91 83 75 66 58 50 41 33 25 16 8 0]
FR=[100 40 31 16 0 0 0 0 0 0 0 0 0 0]
plot(FR,PD,'g--','LineWidth',4)

hold on
%beta=1.5 ROC
PD=[100 100 91 83 75 66 58 50 41 33 25 16 8 0]
FR=[100 25 21 11 0 0 0 0 0 0 0 0 0 0]
plot(FR,PD,'m-','LineWidth',4)

hold on
%beta=2, CS ROC
PD=[100 100 91 83 75 66 58 50 41 33 25 16 8 0]
FR=[100 20 17 9 0 0 0 0 0 0 0 0 0 0]
plot(FR,PD,'c-','LineWidth',4)

hold on
%beta=2.5 ROC
PD=[100 100 91 83 75 66 58 50 41 33 25 16 8 0]
FR=[100 13 8 0 0 0 0 0 0 0 0 0 0 0]
plot(FR,PD,'k-','LineWidth',4)

legend('JSD','Beta=0.5 (HD)','Beta=1 (KLD)','Beta=1.5','Beta=2 (CSD)','Beta=2.5');
```

# Appendix G – UPR16 Form

## FORM UPR16 Research Ethics Review Checklist



Please include this completed form as an appendix to your thesis (see the Postgraduate Research Student Handbook for more information)

<b>Postgraduate Research Student (PGRS) Information</b>		<b>Student ID:</b>	828996
<b>PGRS Name:</b>	Jean Tajer		
<b>Department:</b>	School of Computing	<b>First Supervisor:</b>	Dr. Mo'Adda
<b>Start Date:</b> (or progression date for Prof Doc students)	October 2016		
<b>Study Mode and Route:</b>	Part-time <input checked="" type="checkbox"/>	MPhil <input type="checkbox"/>	MD <input type="checkbox"/>
	Full-time <input type="checkbox"/>	PhD <input checked="" type="checkbox"/>	Professional Doctorate <input type="checkbox"/>
<b>Title of Thesis:</b>	Detection of Malicious Hosts Against Agents in Mobile Agent Networks		
<b>Thesis Word Count:</b> (excluding ancillary data)	33551		
<p>If you are unsure about any of the following, please contact the local representative on your Faculty Ethics Committee for advice. Please note that it is your responsibility to follow the University's Ethics Policy and any relevant University, academic or professional guidelines in the conduct of your study</p> <p>Although the Ethics Committee may have given your study a favourable opinion, the final responsibility for the ethical conduct of this work lies with the researcher(s).</p>			
<b>UKRIO Finished Research Checklist:</b>			
(If you would like to know more about the checklist, please see your Faculty or Departmental Ethics Committee rep or see the online version of the full checklist at: <a href="http://www.ukrio.org/what-we-do/code-of-practice-for-research/">http://www.ukrio.org/what-we-do/code-of-practice-for-research/</a> )			
a) Have all of your research and findings been reported accurately, honestly and within a reasonable time frame?	YES	<input checked="" type="checkbox"/>	
	NO	<input type="checkbox"/>	
b) Have all contributions to knowledge been acknowledged?	YES	<input checked="" type="checkbox"/>	
	NO	<input type="checkbox"/>	
c) Have you complied with all agreements relating to intellectual property, publication and authorship?	YES	<input checked="" type="checkbox"/>	
	NO	<input type="checkbox"/>	
d) Has your research data been retained in a secure and accessible form and will it remain so for the required duration?	YES	<input checked="" type="checkbox"/>	
	NO	<input type="checkbox"/>	
e) Does your research comply with all legal, ethical, and contractual requirements?	YES	<input checked="" type="checkbox"/>	
	NO	<input type="checkbox"/>	
<b>Candidate Statement:</b>			
I have considered the ethical dimensions of the above named research project, and have successfully obtained the necessary ethical approval(s)			
<b>Ethical review number(s) from Faculty Ethics Committee (or from NRES/SCREC):</b>			
If you have <i>not</i> submitted your work for ethical review, and/or you have answered 'No' to one or more of questions a) to e), please explain below why this is so:			
<b>Signed (PGRS):</b>			<b>Date:</b> 6 August 2018

APPENDIX F - SKETCH TECHNIQUE AND DETECTION ALGORITHMS CODES

