

**School of Information Systems**

**DoSTDM: A Denial of Service Detection Model Using  
Firewall Data Traffic Pattern Matching**

**Mohammed Ali Mohammed Ahmad Salem**

**This thesis is presented for the Degree of  
Doctor of Philosophy  
of  
Curtin University**

**March 2013**

## DECLARATION

To the best of my knowledge and belief this thesis contains no material previously published by any other person except where due acknowledgment has been made.

This thesis contains no material which has been accepted for the award of any other degree or diploma in any university.

Signature:

A handwritten signature in blue ink, appearing to read 'Ms. Salem', is written over a horizontal dotted line. Below the signature, the date '12/03/2013' is handwritten in blue ink over another horizontal dotted line.

Date:

Parts of this thesis been previously published as listed below:

- Salem, Mohammed and Armstrong, Helen. 2008. Identifying DOS attacks using data pattern analysis, in Craig Valli and Andrew Woodward (ed), The 6th Australian Information Security Management Conference, Dec. 1 2008, pp. 118-129. Perth, Australia: SECAU - Security Research Centre.  
[http://espace.library.curtin.edu.au/R?func=dbin-jump-full&local\\_base=gen01-era02&object\\_id=118194](http://espace.library.curtin.edu.au/R?func=dbin-jump-full&local_base=gen01-era02&object_id=118194)

## **ABSTRACT**

This research deals with Denial of Service (DoS) flooding attacks, sometimes referred to as Distributed Denial of service (DDoS) attack. These types of attacks toward internet connected networks are on the rise. During a DDoS attack it is very difficult for a firewall to differentiate between legitimate packets that can be accepted and rogue packets that should be rejected due to packet flooding, particularly in large networks carrying substantial levels of traffic. Therefore, large networks commonly use a form of host based or network based Intrusion Detection Systems (IDS) and / or Intrusion Prevention Systems (IPS) to identify DoS / DDoS attacks. However some new viruses, malware and worms can escape detection until their signatures are known and classified as an attack.

Commonly used IDS / IPS systems are rule based or using attacking signatures, and they can produce a high number of false positive alerts. Therefore, many organisations all over the world are tracing new DoS / DDoS attacks, including the US government and other organisations that are trying to reduce the DoS / DDoS effect on the internet by publishing recent DoS / DDoS incident statistics. These publications helped in identifying new DDoS attacking methods by either developing new detection techniques or improving current detection techniques to reduce future DDoS attacking impact on internet connected networks. The current trends toward cloud computing and intrusion prevention approach the problem by moving the protection from the client side and centralizing it on the cloud provider or data centre side of the network. This approach requires a lot of virtualization of both hardware and software components. However, the security problem remains the same, if it is not even larger, due to the complex nature of these implementations. Changes made to the internet communication protocols (IPV4 and IPV6) included new security measures at packet levels, however these security measures and enhancements do not help when the network is under a flooding attack by a DoS / DDoS attacker. In addition, new emerging wireless technologies making it very hard to provide complete immunity to the network making the security hazards of a DoS / DDoS attacks even more difficult to trace back to the attacking device using current IDS and IPS tools.

Therefore, a simpler approach to tackle DoS / DDoS attacks is needed without the need for IDS / IPS. Such an approach needs to be aligned toward both networks and users behaviours (i.e. network traffic levels generated by user actions in normal situations). The significance of this research revolves around the idea of simplifying the solution in a complex growing network structure. The simplicity is achieved by using the network firewall as the main source of information that can draw a picture of the network activities under normal and abnormal situations. The research explored the causal relationship between DoS / DDoS flooding attacks phenomena and firewalled networks behaviour under attack using a positivist research paradigm with a quantitative approach. This research methodology utilised the statistical and neural network modelling to predict and forecast the amount of rejected packets within the network using the firewall records before, during and after the occurrence of a DoS / DDoS attack. In addition to and inline with the simplicity approach, the research provides a model rather than a software and / or hardware product like most IDS / IPS implementations. Therefore, the proposed DoS / DDoS protection model can be implemented using a variety of computer languages at the firewall level and does not require agents like most IDS and IPS systems. The aim of this research was to determine if it is possible for a firewall to analyse its own traffic patterns to identify an attempted denial of service attack. In the context of this research, statistical analyses of firewall logs for a large network were carried out and a baseline determined. Estimated traffic levels were projected using Linear Regression and Holt-Winter statistical forecasting methods for comparison with the network firewall data traffic baseline. In addition, a simulation network was created to compare with the results that are obtained from a real network firewall logs. The research also proposes a Multi Layer Perceptron (MLP) method that implements a Back Propagation Neural Network (BPNN) for forecasting rejected traffic falling outside the projected level for the network under study that could indicate an attack. This triangulation approach between statistical and neural network forecasting approaches provided the research model with multi prediction techniques with high accuracy.

The results of the research were positive with variance and residuals error analysis of the projected rejected packet levels successfully indicating an attack toward the targeted network.

## **ACKNOWLEDGMENTS**

This research was not to be completed without the help of God then the support of many people who helped me to reach this point in my life. I dedicate this research to my mother and my late father for giving me a lot of things that made me who I am.

I'm very sure I have a long list of individuals that I would like to thank within Curtin School of Information Systems, so thank you every one who helped. However, I would like to have a special thanks to my supervisor Dr. Helen Armstrong for her continuous support and encouragement over the past years. Also, my thanks go to Dr. Peter Dell for his support in my final stage of writing this thesis.

I finally would like to thank my wife Reema and my kids Lara, Ali and Sara for all the warm support during my journey to complete this research.

# TABLE OF CONTENTS

<b>DECLARATION .....</b>	<b>II</b>
<b>ABSTRACT .....</b>	<b>III</b>
<b>ACKNOWLEDGMENTS.....</b>	<b>V</b>
<b>TABLE OF CONTENTS.....</b>	<b>VI</b>
<b>LIST OF FIGURES.....</b>	<b>IX</b>
<b>LIST OF TABLES &amp; CODE LISTINGS .....</b>	<b>XI</b>
<b>KEYWORDS AND ABBREVIATIONS .....</b>	<b>XIV</b>
<b>CHAPTER 1 .....</b>	<b>1</b>
<b>INTRODUCTION .....</b>	<b>1</b>
1.1 RESEARCH BACKGROUND.....	1
1.2 RESEARCH PROBLEM DEFINITION .....	2
1.3 RESEARCH PROBLEM IMPLICATIONS.....	6
1.4 RESEARCH OBJECTIVES.....	7
1.5 ADDRESSING THE RESEARCH PROBLEM .....	7
1.6 RESEARCH SIGNIFICANCE AND CONTRIBUTION OVERVIEW.....	8
1.7 THESIS STRUCTURE OVERVIEW.....	9
<b>CHAPTER 2 .....</b>	<b>10</b>
<b>PROBLEM DOMAIN LITERATURE REVIEW.....</b>	<b>10</b>
2.1 INTRODUCTION TO INFORMATION SYSTEMS SECURITY.....	11
2.2 NETWORK SECURITY THREATS.....	12
2.2.1 <i>Network Security Threats Overview.....</i>	<i>12</i>
2.2.2 <i>DoS and DDoS Attacks Definitions.....</i>	<i>16</i>
2.2.3 <i>DoS and DDoS Attack Types &amp; Classifications.....</i>	<i>17</i>
2.2.4 <i>Impact and Motivation of DoS / DDoS Attacks.....</i>	<i>21</i>
2.2.5 <i>Initiation Models for DoS and DDoS Attacks.....</i>	<i>24</i>
2.2.6 <i>DoS &amp; DDoS Detection Algorithms.....</i>	<i>26</i>
2.2.7 <i>DoS &amp; DDoS Defence Mechanisms.....</i>	<i>27</i>
2.3 FIREWALL DEFENCE MECHANISMS .....	29
2.3.1 <i>Firewall Implementations.....</i>	<i>30</i>
2.3.2 <i>Firewall Quality &amp; Performance.....</i>	<i>32</i>
2.3.3 <i>Firewalls Limitations.....</i>	<i>34</i>
2.4 INTRUSION DETECTION AND PREVENTION SYSTEMS .....	36
2.4.1 <i>IDS Background.....</i>	<i>36</i>
2.4.2 <i>IDS Detection Techniques .....</i>	<i>37</i>
2.4.3 <i>IDS Implementations.....</i>	<i>39</i>
2.4.4 <i>Issues with IDS Implementations .....</i>	<i>42</i>
2.4.5 <i>Improving IDS Implementation Using Self-Learning Approach.....</i>	<i>43</i>
2.4.6 <i>Intrusion Prevention Systems.....</i>	<i>44</i>
2.5 RELEATED WORK.....	46
2.5.1 <i>Anomaly Based Related Research.....</i>	<i>47</i>
2.5.2 <i>Anomaly Detection Top Used Techniques.....</i>	<i>61</i>
2.6 CHAPTER CONCLUSION .....	63

<b>CHAPTER 3 .....</b>	<b>65</b>
<b>RESEARCH METHODOLOGY .....</b>	<b>65</b>
3.1 RESEARCH OBJECTIVES.....	66
3.2 RESEARCH QUESTIONS .....	66
3.3 INTRODUCTION TO RESEARCH METHODOLOGY.....	67
3.4 RESEARCH PARADIGMS AND APPROACHES.....	67
3.4.1 <i>Research Methods - Positivist vs. Interpretivist</i> .....	67
3.4.2 <i>Research Approaches</i> .....	70
3.4.3 <i>DoS / DDoS Problem Domain &amp; Research Paradigm</i> .....	72
3.4.4 <i>Applied Research Approaches</i> .....	74
3.5 RESEARCH CHARACTERISTICS .....	77
3.6 RESEARCH SIGNIFICANCE .....	77
3.6.1 <i>Contribution to Theory</i> .....	78
3.6.2 <i>Contribution to Practice</i> .....	79
3.7 RESEARCH PROCESS DESIGN.....	80
3.7.1 <i>Phase 1 – Research Literature and Design</i> .....	80
3.7.2 <i>Phase 2 – Research Case Study</i> .....	80
3.7.3 <i>Phase 3 – Experimental &amp; Simulation Research Bed</i> .....	81
3.7.4 <i>Phase 4 – DoS Tracking and Detection Model Design &amp; Testing</i> .....	81
3.7.5 <i>Phase 5 – Research Findings &amp; Discussion</i> .....	82
3.7.6 <i>Phase 6 – Research Documentation &amp; Submission</i> .....	82
3.8 CHAPTER CONCLUSION .....	84
<b>CHAPTER 4 .....</b>	<b>85</b>
<b>DATA COLLECTION: SOURCES &amp; STRUCTURE ANALYSIS.....</b>	<b>85</b>
4.1 INTRODUCTION.....	86
4.2 RAW DATA SOURCE.....	87
4.2.1 <i>Generic Firewall Logs</i> .....	87
4.2.2 <i>The Case Study Network Infrastructure</i> .....	89
4.2.3 <i>The Simulated Network Infrastructure</i> .....	91
4.3 PROPOSED MODEL DATA STRUCTURES.....	93
4.3.1 <i>Basic Data Structures &amp; Data Conversion</i> .....	93
4.3.2 <i>Data Measurements &amp; Pre-Processing</i> .....	94
4.4 DATA COLLECTION & STORAGE.....	97
4.4.1 <i>Data Collection Process</i> .....	97
4.4.2 <i>Data Storage</i> .....	97
4.5 DATA PRE-PROCESSING & PREPARATION PROCESS.....	98
4.6 CHAPTER FINDINGS.....	102
4.7 CHAPTER CONCLUSION .....	103
<b>CHAPTER 5 .....</b>	<b>104</b>
<b>THE DOSTDM DESIGN STRATEGY .....</b>	<b>104</b>
5.1 INTRODUCTION.....	105
5.2 DoSTDM DESIGN PROCESS.....	106
5.2.1 <i>Model Design - Theoretical Aspects</i> .....	106
5.2.2 <i>Model Design - Business &amp; Practical Aspects</i> .....	107
5.2.3 <i>Model Design Process Overview</i> .....	107
5.2.4 <i>DoSTDM Design Planning Phase</i> .....	109
5.2.5 <i>DoSTDM Design Analysis Phase</i> .....	109
5.2.6 <i>DoSTDM Design Engineering Phase</i> .....	110
5.2.7 <i>DoSTDM Design Evaluation Phase</i> .....	121
5.2.8 <i>DoSTDM Model Final Design Structure</i> .....	121
5.3 DoSTDM PROCESS IMPLEMENTATION GUIDE.....	124
5.4 DoSTDM MODEL CHARACTERISTICS AND LIMITATIONS .....	125
5.5 CHAPTER CONCLUSION .....	125

<b>CHAPTER 6 .....</b>	<b>129</b>
<b>DOSTDM IMPELENTATION &amp; FINDINGS ANALYSIS .....</b>	<b>129</b>
6.1 INTRODUCTION.....	130
6.2 DOSTDM IMPLEMENTATION .....	130
6.2.1 <i>Simulation Network Setup</i> .....	130
6.2.2 <i>Statistical Forecasting Engine</i> .....	132
6.2.3 <i>Neural Network Pseudo Code</i> .....	137
6.2.4 <i>DoSTDM Code Implementation in Java</i> .....	143
6.3 DATA ANALYSIS & FINDINGS .....	146
6.3.1 <i>Holt-Winter Statistical Analysis &amp; Findings from CSDS1</i> .....	147
6.3.2 <i>Regression Statistical Analysis &amp; Findings from CSDS1</i> .....	153
6.3.3 <i>DoSTDM Analysis &amp; Findings from 1<sup>st</sup> Data Set (CSDS1)</i> .....	161
6.3.4 <i>DoSTDM Analysis &amp; Findings from 2<sup>nd</sup> Data Set (CSDS2)</i> .....	163
6.3.5 <i>DoSTDM Analysis &amp; Findings from Simulation Data Set (SNDS)</i> .....	168
6.4 DoSTDM THRESHOLD MANAGEMENT.....	176
6.5 CHAPTER CONCLUSION .....	178
<b>CHAPTER 7 .....</b>	<b>180</b>
<b>RESEARCH DISCUSSION &amp; CONCLUSIONS.....</b>	<b>180</b>
7.1 RESEARCH FINDINGS DISCUSSION.....	181
7.2 RESEARCH CONTRIBUTION .....	183
7.2.1 <i>Contribution to Theory</i> .....	183
7.2.2 <i>Contribution to Practice</i> .....	185
7.3 RESEARCH LIMITATIONS.....	186
7.4 FUTURE RESEARCH DIRECTION .....	187
<b>REFERENCES .....</b>	<b>189</b>
<b>APPENDIX – A: DOSTDM PROGRAMMING CODE .....</b>	<b>200</b>
A.1 DATA PREPREPARATION PROGRAMMING CODE.....	200
A.1.1 <i>Data Collection Shell Scripts</i> .....	200
A.1.2 <i>Data Processing Code in Java</i> .....	202
A.2 PROGRAMMING CODE LISTING .....	214
A.2.1 <i>Simulation Network Packets Generator Scripts</i> .....	214
A.2.2 <i>DoSTDM Java Applet &amp; Associated Classes</i> .....	216
<b>APPENDIX – B: PROCESSED FIREWALL LOGS.....</b>	<b>299</b>
<b>APPENDIX – C: COMPILED DOSTDM JAVA APPLET CLASSES .....</b>	<b>300</b>

## LIST OF FIGURES

FIGURE 1 – 1: THE INCREASE OF ATTACK SIZES IN GIGA BITS PER SECOND .....	3
FIGURE 1 – 2: TOTAL NO. OF DDOS ATTACKS FROM 2006 TO 2011.....	3
FIGURE 1 – 3: INTERNET TRAFFIC IN PETABYTES FROM 1990 TO 2011.....	4
FIGURE 1 – 4: ATTACK SPIKE OF 100 GIGA BITS PER SECOND.....	5
FIGURE 1 – 5: CSI ATTACK SURVEY, ATTACKS EXPERIENCED BY PERCENTAGE OF RESPONDERS.....	5
FIGURE 1 – 6: THESIS RESEARCH APPROACH .....	8
FIGURE 2 – 1: TAXONOMY OF INFORMATION SECURITY TECHNOLOGIES .....	11
FIGURE 2 – 2: NETWORK BASED IDS / IPS VS. HOST BASED IDS / IPS ARCHITECTURE .....	12
FIGURE 2 – 3: ATTACK SOPHISTICATION VS. INTRUDER TECHNICAL KNOWLEDGE .....	15
FIGURE 2 – 4: VULNERABILITY EXPLOIT CYCLE .....	16
FIGURE 2 – 5: DDOS ATTACKS TAXONOMY .....	20
FIGURE 2 – 6: TOTAL ATTACK TYPES (Q1 2012) .....	22
FIGURE 2 – 7: CODE RED, SLAMMER AND BLASTER ATTACKS GROWTH OVER 24 HOURS .....	23
FIGURE 2 – 8A: DDOS AGENT-HANDLER ATTACK MODEL.....	25
FIGURE 2 – 8B: DDOS IRC-BASED ATTACK MODEL .....	25
FIGURE 2 – 9: TAXONOMY OF DDOS DEFENCE MECHANISMS .....	28
FIGURE 2 – 10: TYPICAL FIREWALL CONFIGURATION WITH DMZ .....	29
FIGURE 2 – 11: FIREWALL IMPLEMENTATION IN COMPUTER NETWORKS ARCHITECTURES.....	31
FIGURE 2 – 12: TYPICAL FIREWALL PACKET FILTERING POLICY EXAMPLE .....	31
FIGURE 2 – 13: GARTNER RESEARCH MAGIC QUADRANT FOR ENTERPRISE NETWORK FIREWALLS .....	34
FIGURE 2 – 14: ANOMALY BASED GENERIC IDS ARCHITECTURE.....	38
FIGURE 2 – 15: DARPA 1999 DETECTION RESULTS FOR KNOWN VS. NEW ATTACKS .....	39
FIGURE 2 – 16: TIMELINE OF SIGNIFICANT EVENTS IN DEVELOPMENT OF NIDS.....	41
FIGURE 2 – 17: FIREWALL WITH IDS IN INFORMATION TECHNOLOGY SECURITY INFRASTRUCTURE .....	41
FIGURE 2 – 18: PLACEMENT OF INTRUSION PREVENTION LAYER.....	46
FIGURE 2 – 19: ANOMALY DETECTION TECHNIQUES DISTRIBUTION (1999 – 2011).....	62
FIGURE 3 – 1: RESEARCH DESIGN PROCESS .....	83
FIGURE 4 – 1: DoSTDM CASE STUDY LAN / WAN NETWORK LAYOUT .....	89
FIGURE 4 – 2: EXPERIMENTAL SIMULATION NETWORK .....	91
FIGURE 5 – 1: DoSTDM MODEL PROCESS DESIGN USING SPIRAL DEVELOPMENT PROCESS .....	109
FIGURE 5 – 2: PACKETS TREND COMPONENT IN CASE STUDY FIREWALL LOGS .....	111
FIGURE 5 – 3: PACKETS SEASONALITY COMPONENT IN CASE STUDY FIREWALL LOGS .....	111
FIGURE 5 – 4: NEURAL NETWORK ARCHITECTURE EXAMPLE WITH ONE HIDDEN LAYER.....	117
FIGURE 5 – 5: MLP SYSTEM FLOOR PLAN .....	119
FIGURE 5 – 6: DoSTDM MLP NEURAL NETWORK STRUCTURE.....	120
FIGURE 5 – 7: DoSTDM FINAL DESIGN STRUCTURE.....	123
FIGURE 6 – 1: DoSTDM BASELINE DATA EXPLORER JAVA APPLET .....	144
FIGURE 6 – 2: DoSTDM REJECTED PACKETS FORECASTING ANALYSER JAVA APPLET.....	144
FIGURE 6 – 3: DoSTDM REJECTED FORECASTING COMPARISON USING DoSTDM APPLET.....	145
FIGURE 6 – 4: AVERAGE RESIDUALS PATTERN IN CSDS1.....	147
FIGURE 6 – 5: COMPARISON OF INBOUND PACKETS VS. REJECTED PACKET IN CSDS1.....	150
FIGURE 6 – 6: HOLT-WINTER RESIDUALS PLOT FOR ALL INBOUND PACKETS IN CSDS1.....	151
FIGURE 6 – 7: HOLT-WINTER RESIDUALS PLOT FOR TCP INBOUND PACKETS IN CSDS1.....	152
FIGURE 6 – 8: HOLT-WINTER RESIDUALS PLOT FOR ALL REJECTED PACKETS IN CSDS1.....	152
FIGURE 6 – 9: HOLT-WINTER RESIDUALS PLOT FOR TCP REJECTED PACKETS IN CSDS1.....	153
FIGURE 6 – 10: SUB-SET REGRESSION EQUATION RESIDUALS PLOT WITHOUT SEASONALITY FOR CSDS1 .....	160
FIGURE 6 – 11: DoSTDM MLP VS. HOLT-WINTER VS. REGRESSION FOR CSDS1.....	162
FIGURE 6 – 12: DoSTDM MLP VS. HOLT-WINTER VS. REGRESSION FOR CSDS2.....	165
FIGURE 6 – 13: CSDS2 INBOUND TCP PACKETS FOR SEPTEMBER 2005.....	166
FIGURE 6 – 14: CSDS2 REJECTED TCP PACKETS FOR SEPTEMBER 2005.....	167
FIGURE 6 – 15: CSDS2 REJECTED TCP PACKETS OVER 12 MONTHS .....	167
FIGURE 6 – 16: DoSTDM MLP VS. HOLT-WINTER VS. REGRESSION FOR SNDS .....	169
FIGURE 6 – 17: DoSTDM MLP REJECTED PACKETS PREDICTION OF THE 1 <sup>ST</sup> SIMULATED DoS / DDOS ATTACK ON DECEMBER 4, 2010 FOR SNDS.....	171

FIGURE 6 – 18: DoSTDM MLP REJECTED PACKETS PREDICTION OF THE 2 <sup>ND</sup> SIMULATED DoS / DDoS ATTACK ON DECEMBER 11, 2010 FOR SNDS.....	172
FIGURE 6 – 19: DoSTDM MLP REJECTED PACKETS PREDICTION OF THE 3 <sup>RD</sup> SIMULATED DoS / DDoS ATTACK ON DECEMBER 17, 2010 FOR SNDS.....	173
FIGURE 6 – 20: DoSTDM MLP REJECTED PACKETS PREDICTION OF THE 4 <sup>TH</sup> SIMULATED DoS / DDoS ATTACK ON DECEMBER 18, 2010 FOR SNDS.....	174
FIGURE 6 – 21: DoSTDM MLP REJECTED PACKETS PREDICTION OF THE 5 <sup>TH</sup> SIMULATED DoS / DDoS ATTACK ON DECEMBER 27, 2010 FOR SNDS.....	175
FIGURE 6 – 22: DoSTDM MLP REJECTED PACKETS PREDICTION OF THE 4 <sup>TH</sup> SIMULATED DoS / DDoS ATTACK ON DECEMBER 18, 2010 FOR SNDS WITH 10 WEEKS TRAINING HISTORY ONLY.....	177
FIGURE A – 1: DoSTDM JAVA APPLET CODING DESIGN.....	217

## LIST OF TABLES & CODE LISTINGS

TABLE 2 – 1: TOPOLOGY AND COMPLEXITY OF COMPUTER INTRUSIONS .....	14
TABLE 2 – 2: ACTIVE AND PASSIVE THREATS .....	15
TABLE 2 – 3: DoS ATTACK CLASSIFICATIONS .....	17
TABLE 2 – 4: DDoS STATS 2006 - 2011.....	23
TABLE 2 – 5: SUMMARY OF ANOMALY BASED AND MISUSE BASED DETECTION MODELS.....	26
TABLE 2 – 6: SAMPLE OF ANOMALY BASED RESEARCH TECHNIQUES (1999 – 2011).....	61
TABLE 3 – 1: IS RESEARCH APPROACHES & PHILOSOPHIES (POSITIVIST VS. INTERPRETIVIST) .....	67
TABLE 3 – 2: SUMMARY OF SOFT VS. HARD RESEARCH DICHOTOMIES .....	69
TABLE 3 – 3: IS RESEARCH APPROACHES REVISED TAXONOMY .....	71
TABLE 3 – 4: IS RESEARCH METHODOLOGIES ASSOCIATED WITH STRATEGIES .....	73
TABLE 3 – 5: PROCESS OF BUILDING THEORY FROM CASE STUDY RESEARCH .....	75
TABLE 3 – 6: A MORPHOLOGIC BOX CLASSIFICATION OF SIMULATION RESEARCH METHOD .....	76
TABLE 4 – 1: DIFFERENT RAW FIREWALL LOG SAMPLES .....	88
TABLE 4 – 2: MODEL DATA STRUCTURE SAMPLE.....	93
TABLE 4 – 3: MODEL PROPOSED DATA STRUCTURE PER PROTOCOL FOR AN INBOUND SAMPLE.....	95
TABLE 4 – 4: PROCESSED DATA STRUCTURE MATRIX.....	96
TABLE 4 – 5: DATA COLLECTION STAGGED APPROACH.....	97
TABLE 4 – 6: FILECLAB LOG FIELDS NAMES MAPPING TO COMMON FIELDS NAMES .....	98
TABLE 4 – 7: DATA PREPARATION PROCESSOR PSEUDO CODE .....	100
TABLE 4 – 8: RAW FIREWALL LOG DATA FORMAT .....	101
TABLE 4 – 9: TCP PACKETS CLASSIFIED BY DIRECTION AND STATUS OVER A PERIOD OF 24 HOURS .....	101
TABLE 5 – 1: HOLT-WINTER MULTIPLICATIVE FORECASTING EQUATIONS .....	112
TABLE 5 – 2: MULTIPLE LINEAR REGRESSION FORECASTING EQUATIONS .....	114
TABLE 5 – 3: REGRESSION MODEL DUMMY VARIABLES .....	115
TABLE 5 – 4: PROGRAMMING APPROACH VS. NEURAL COMPUTING APPROACH.....	116
TABLE 5 – 5: NEURAL NETWORK FOR MLP .....	118
TABLE 5 – 6: NEURAL NETWORK FOR MLP .....	119
TABLE 6 – 1: NEMESIS SERVER SETUP .....	130
TABLE 6 – 2: FIREWALL SERVER SETUP .....	131
TABLE 6 – 3: NEMESIS PACKETS GENERATOR SCRIPT PSEUDO CODE .....	131
TABLE 6 – 4: FIREWALL HTTP GENERATOR PSEUDO CODE .....	132
TABLE 6 – 5: HOLT-WINTER PSEUDO CODE .....	135
TABLE 6 – 6: PSEUDO CODE FOR MULTIPLE LINEAR REGRESSION INTERFACE TO FLANAGAN REGRESSION CLASS.....	136
TABLE 6 – 7: NEURAL NETWORK (MLP) PSEUDO CODE .....	143
TABLE 6 – 8: REJECTED PACKETS COUNTS PER HOUR IN CSDS1 (JANUARY – MARCH, 2004) .....	149
TABLE 6 – 9: REJECTED PACKETS COUNTS PER HOUR IN CSDS1 (APRIL – MAY, 2004) .....	149
TABLE 6 – 10: REGRESSION PARAMETERS DEFINITIONS (SOURCE: MINITAB STATISTICAL GUIDE) .....	155
TABLE 6 – 11: REGRESSION EQUATION WITHOUT SEASONALITY IN CSDS1 .....	155
TABLE 6 – 12: REGRESSION P-VALUES TABLE FOR INBOUND AND OUTBOUND FACTORS WITHOUT SEASONALITY IN CSDS1.....	156
TABLE 6 – 13: ANOVA ANALYSIS TABLE FOR INBOUND AND OUTBOUND FACTORS WITHOUT SEASONALITY FROM MINITAB Using CSDS1 .....	156
TABLE 6 – 14: REGRESSION EQUATION WITH SEASONALITY IN CSDS1.....	157
TABLE 6 – 15: REGRESSION P-VALUES TABLE WITH SEASONALITY IN CSDS1.....	157
TABLE 6 – 16: ANOVA ANALYSIS TABLE FOR INBOUND & OUTBOUND FACTORS WITH SEASONALITY IN CSDS1.....	158
TABLE 6 – 17: BEST SUBSET REGRESSION EQUATION WITH SEASONALITY IN CSDS1 .....	158
TABLE 6 – 18: BEST SUBSET OF REGRESSION FACTORS WITH SEASONALITY IN CSDS1 .....	158
TABLE 6 – 19: ANOVA ANALYSIS OF BEST SUBSET REGRESSION FACTORS WITH SEASONALITY IN MINITAB Using CSDS1 .....	159
TABLE 6 – 20: ANOVA ANALYSIS AND COMPARISON TABLE OF ALL MINITAB REGRESSION EQUATIONS USING CSDS1 .....	159
TABLE 6 – 21: RMSE COMPARISON OF ALL METHODS BY DoSTDM FOR CSDS1 .....	161
TABLE 6 – 22: RMSE COMPARISON OF ALL METHODS BY DoSTDM FOR CSDS2 .....	164
TABLE 6 – 23: RMSE COMPARISON OF ALL METHODS BY DoSTDM FOR SNDS.....	168
TABLE 6 – 24: SIMULATED NETWORK DoS ATTACKS TRAFFIC PERIODS WITHIN SNDS .....	170

TABLE 6 – 25: MLP SEEDING EFFECT ON RMSE WITH DoS ATTACKS .....	179
LISTING A – 1: UNIX SHELL & PERL SCRIPT TO COLLECT RAW DATA FROM FIREWALL LOGS.....	202
LISTING A – 2: JAVA CLASS BASEDATATOKENIZER.JAVA .....	205
LISTING A – 3: JAVA CLASS HOURLYDATACLASSIFIER.JAVA .....	213
LISTING A – 4: SCRIPTS SCHEDULER FOR RANDOM TRAFFIC PATTERN GENERATOR IN THE SIMULATION NETWORK ....	214
LISTING A – 5: ARP TRAFFIC RANDOM GENERATOR SCRIPT .....	214
LISTING A – 6: DNS TRAFFIC RANDOM GENERATOR SCRIPT.....	215
LISTING A – 7: ETHERNET TRAFFIC RANDOM GENERATOR SCRIPT .....	215
LISTING A – 8: ICMP TRAFFIC RANDOM GENERATOR SCRIPT .....	215
LISTING A – 9: IP TRAFFIC RANDOM GENERATOR SCRIPT .....	215
LISTING A – 10: RIP TRAFFIC RANDOM GENERATOR SCRIPT .....	216
LISTING A – 11: TCP TRAFFIC RANDOM GENERATOR SCRIPT .....	216
LISTING A – 12: UDP TRAFFIC RANDOM GENERATOR SCRIPT .....	216
LISTING A – 13: DoSTDM WEB APPLET CODE (DoSTDM.HTML) .....	218
LISTING A – 14: DoSTDM JAVA CLASS (DoSTDM.JAVA).....	255
LISTING A – 15: JAVA CLASS HOLT-WINTER (HWF.JAVA) .....	259
LISTING A – 16: JAVA CLASS REGRESSION INTERFACE (REG.JAVA).....	262
LISTING A – 17: JAVA CLASS MLP (MLP.JAVA, BASED ON PHIL BRIERLEY MLP CODE).....	272
LISTING A – 18: JAVA CLASS ANALYSIS TABLE (ANALYSIS.JAVA) .....	274
LISTING A – 19: JAVA CLASS ALIGNED LIST CELL RENDERER (ALIGNEDLISTCELLRENDERER.JAVA) .....	275
LISTING A – 20: JAVA CLASS SORT (SORT.JAVA) .....	276
LISTING A – 21: JAVA CLASS DAY OF THE WEEK (DAYOFTHEWEEK.JAVA).....	277
LISTING A – 22: JAVA CLASS DAILY GRAPH (RESIDUALSGRAPH.JAVA).....	280
LISTING A – 23: JAVA CLASS HOLT-WINTER GRAPH (HWGRAPH.JAVA) .....	283
LISTING A – 24: JAVA CLASS REGRESSION GRAPH (REGGRAPH.JAVA).....	286
LISTING A – 25: JAVA CLASS MLP GRAPH (MLPGRAPH.JAVA).....	288
LISTING A – 26: JAVA CLASS RESIDUALS GRAPH (RESIDUALSGRAPH.JAVA) .....	289
LISTING A – 27: JAVA CLASS COMPARE GRAPH (COMPAREGRAPH.JAVA) .....	298

## **APPENDIXES**

Appendix – A: DoSTDM Programming Code.

Appendix – B: Processed Firewall Logs (Soft Copy).

Appendix – C: Compiled DoSTDM Java Applet Classes (Soft Copy).

## KEYWORDS AND ABBREVIATIONS

Keywords	Abbreviation
Artificial Immune Systems	AIS
Artificial Neural Networks	ANN
Address Resolution Protocol	ARP
Back Propagation Neural Network	BPNN
Denial of Service	DoS
Distributed Denial of Service	DDoS
Domain Name System	DNS
Denial of Service Tracking and Detection Model	DoSTDM
Delimited Separated Value	DSV
Ethernet Protocol (Ethernet frames, Physical Layer and MAC operation)	Ethernet
File Transfer Protocol	FTP
Host-based Intrusion Detection System	HIDS
Hyper Text Transfer Protocol	HTTP
Human Immune System	HIS
Internet Control Message Protocol	ICMP
Internet Protocol	IP
Intrusion Detection Systems	IDS
Intrusion Prevention System	IPS
Java Virtual Machine	JVM
Local Area Network	LAN
Media Access Control address	MAC
Mining Audit Data for Automated Models for Intrusion Detection	MADAM ID
Multi Layer Perceptron	MLP
Network Interface Card	NIC
Network Intrusion Detection System	NIDS
Network Operating Systems	NOS
Reflective Denial of Service	RDoS
Remote Procedure Calls	RPC
Routing Information Protocol	RIP
Secure Internet Protocol	IPSec
Secure Shell Layer	SSL
Secure Shell Protocol	SSH
Server Message Block	SMB
Simple Mail Transfer Protocol	SMTP
State Transition Analysis Technique	STAT
Quality of Firewall	QoF
Transmission Control Protocol	TCP
User Datagram Protocol	UDP
Virtual Private Network	VPN
Wide Area Network	WAN

# CHAPTER 1

## INTRODUCTION

### 1.1 RESEARCH BACKGROUND

Network system architectures are becoming more complex and difficult to manage especially from a security point of view. The mix of multiple Network Operating Systems (NOS) platforms makes it harder to prevent new hacking attacks by external sources from the Internet.

Security has been major problem for networked computers from day one and has continued to be a problem with the expansion of networked systems over the past decade. This has led these networks into a security dilemma. On the one hand network resources needed to be protected from external access not allowed by the network policies and on the other hand, keeping the network up and running to provide these resources in 24x7 bases without interruptions.

While these goals seem to be reasonable, they are difficult to achieve, requiring a lot of design work to produce a secure model for any network. However, achieving such a goal utilising the Internet is an even more complex and tedious process. Network administrators spend weeks and months trying to secure their network from known and unknown threats due to the open nature of the Internet.

As it is difficult to predict attacks it is necessary to base-line firewall activities. This process works well for blocking illegal access to network resources but it cannot stop other sorts of network attacks by external and internal entities to the open ports, as they are considered to be legitimate activities by the firewall. In some cases even the most hardened firewalls can still fall into the trap of an attacker by responding to the attack packets rather than dropping them and continuing to process normal network activities.

## 1.2 RESEARCH PROBLEM DEFINITION

This research concentrates on the problem of Denial of Service attack (DoS) and Distributed DoS (DDoS). This type of security attack is normally hard to isolate before and during the attack due to the stress it places on the attacked network infrastructure (i.e. the firewall and network gateway to the internet). The identification of such an attack depends on the number of attackers. Although firewall technologies do incorporate some of the required elements for intrusion protection, these technologies have yet to mature. Security companies such as Check Point and Juniper Networks are in the lead in the quest to improve these tools (Noakes-Fry 2004, Young & Pescatore 2008). Therefore, Intrusion Detection Software (IDS) was introduced as an important part of network infrastructure security, adapting to discover new threatening network attacks (Lee & Stolfo 2000). However, most IDS systems rely on pre-defined set of rules and signatures to identify these attacks making the detection of new attacks nearly as hard as predicting the attack itself.

In a practical networking environment, DoS / DDoS can be defined in different ways based on the target (i.e. specific application or service) as follows:

1. Attacks against application servers: for example web servers causing servers to be unavailable for users.
2. Flooding network gateways and firewalls: for example flooding with a large number of Transmission Control Protocol / Internet Protocol Suite (TCP / IP) packets causing either slowness in network activities or the network to become completely unusable till all packets are dropped from the network.
3. Attacking mail gateways: for example sending mail ware Simple Mail Transfer Protocol (SMTP) packets that can block email systems for a long period of time before they can be cleared.

This research is dealing with DoS / DDoS attacks from the context of the second type, flooding the firewall with an enormous number of packets making it either unstable or unusable. Such a DoS attack will flood the network with randomly generated packets, where the firewall will respond by rejecting these packets if it has been configured properly, however, it will not serve the legitimate network users as it

is busy rejecting these randomly arriving packets. The evolution of DoS / DDoS over the past decade had consumed a lot of efforts and resources. Arbor Networks Inc. monitoring of DDoS (part of Networks 2007 Worldwide Infrastructure Security Report) has shown a significant increase of attack sizes that affected the internet bandwidth utilisation as shown in Figure 1 – 1 (Nazario 2008).

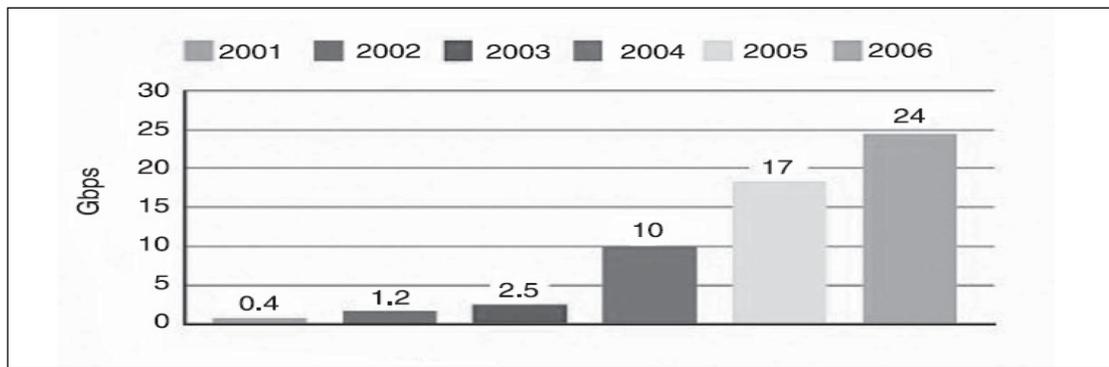


Figure 1 – 1: The Increase of Attack Sizes in Giga Bits per Second  
Source: (Nazario, 2008, 8)

However, In recent years (2006 – 2011), the frequency of DDoS attacks has also been increased to reach it is highest peak in 2009 with nearly 7 millions DDoS attack as shown in Figure 1 – 2.

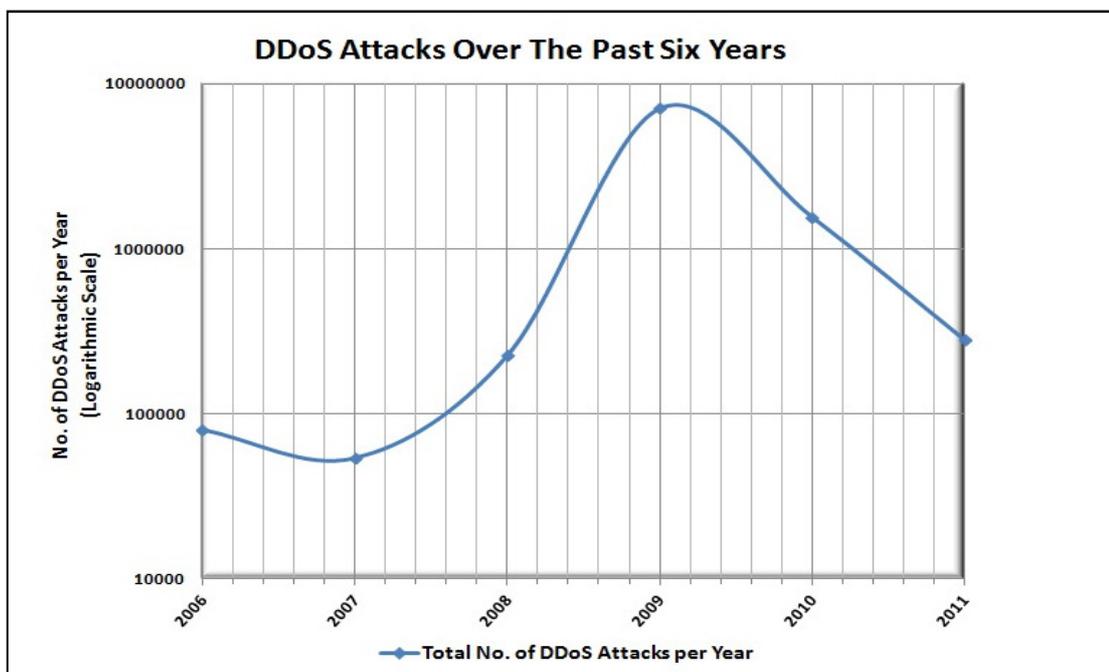


Figure 1 – 2: Total No. of DDoS Attacks from 2006 to 2011

Source: (Based on Data from Shadowserver Foundation, 2012)

Therefore, some DDoS attacks can be easily overlooked by the enormous growth in current internet traffic as shown in Figure 1 – 3.

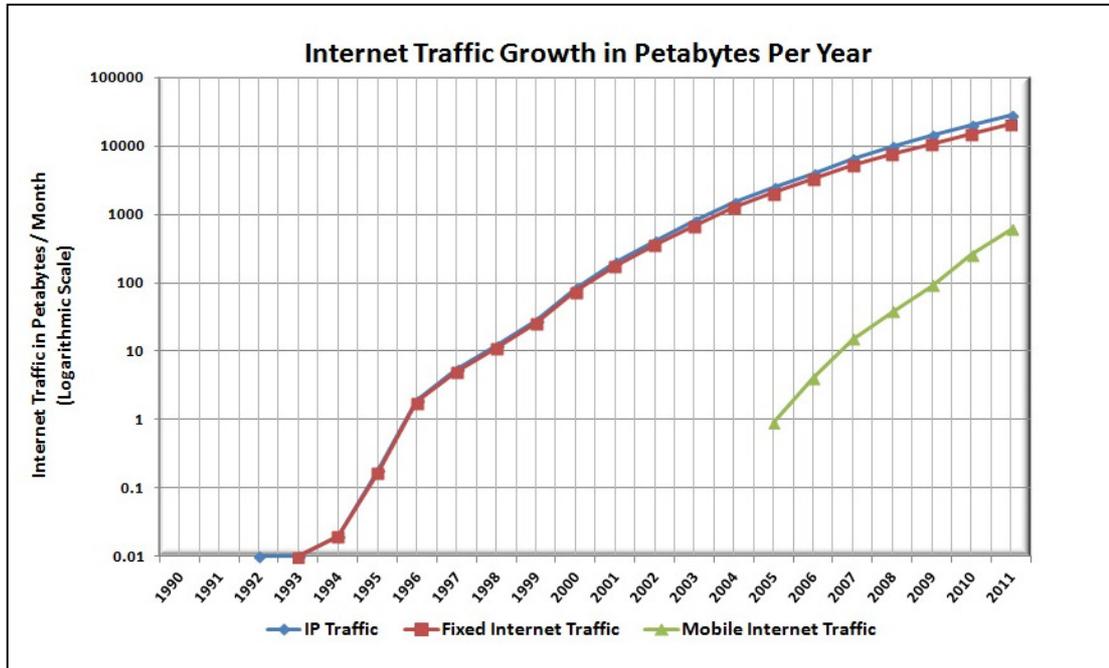


Figure 1 – 3: Internet Traffic in Petabytes from 1990 to 2011  
Source: (Based on Cisco Systems Data obtained from Wikipedia, 2012a)

Figure 1 – 3 shows the internet utilisation for IP, fixed internet and mobile internet subscribers globally per month for every year since 1990 in Petabytes (1 Petabyte = 1 million GB). Under this rate of growth the attacks can be easily missed especially with traffic levels of nearly 25+ Petabytes in 2011 and subject to expand in coming years. Therefore, this increase in the global internet traffic bandwidth has given the attackers the ability to target more victims globally while the attack mechanism remains simple in design.

Nazario (2008) described one of these early simple forms of attacks used back in the 1990's called the Smurf attack in which the attacker sent a request to significant number of servers and internet services using the victim's IP address. As a result of the Smurf attack, every server that had been contacted by the attacker will do its best to reply to the victim IP causing large traffic flooding toward the victim network as shown in Figure 1 – 4.

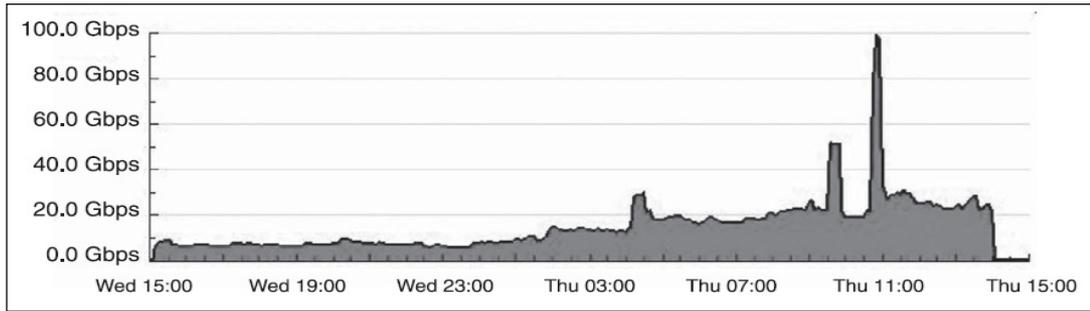


Figure 1 – 4: Attack Spike of 100 Giga Bits per Second  
 Source: (Nazario, 2008, 9)

Therefore, DoS / DDoS attacks has been on the rise and jumped to become a significant threat among other types of attacks. In a survey questionnaire by Computer Security Institute (CSI), DoS attacks have been experienced by 16.8% of the survey responders in 2010 (CSI 2011) as shown in Figure 1 – 5.

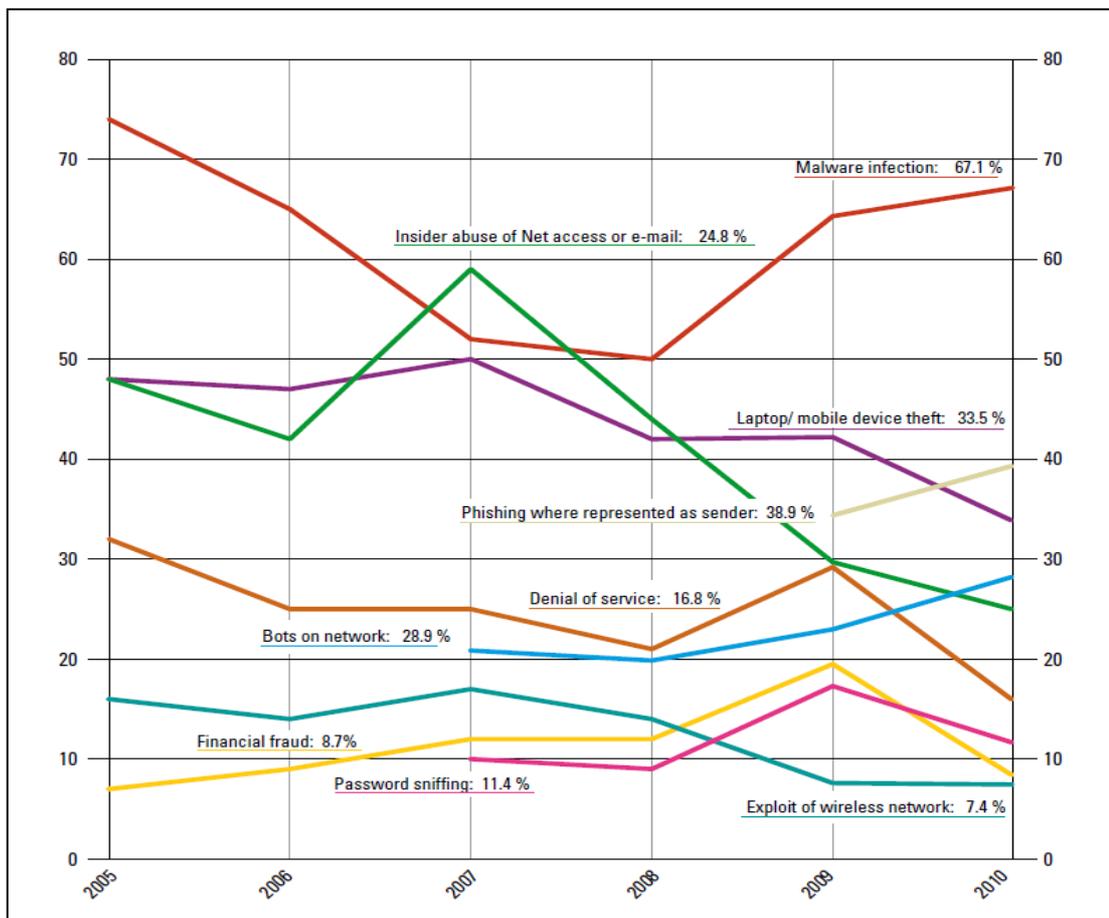


Figure 1 – 5: CSI Attack Survey, Attacks Experienced by Percentage of Responders  
 Source: (CSI, 2011, 15)

With the recent DoS attacks on Wikileaks, Al-Jazeera and many other web sites, it is obvious that the DoS / DDoS threat is far from disappearing from the internet and computer networks. Historically these attacks were not driven by financial gain or profit, however, more recent attacks observed in 2012 whereby Australian financial companies had been targeted for extortion purposes (CERT Australia 2012).

### **1.3 RESEARCH PROBLEM IMPLICATIONS**

The implications of DoS / DDoS attacks can vary from a minor service outage to a complete network shutdown. Therefore, depends on the attack capacity and continuity before identification, the recovery process also vary from a quick services restart (e.g. restarting WWW services server) to a complete network restart, that's include firewall, routers, switches and servers.

During these attacks most of the services (even if they are not directly affected) will struggle to service the users. Let's take a hospital as an example, if a patient record cannot be retrieved while the network has been flooded with DoS / DDoS attacking packets, the live of such a patient might be in danger when he needs immediate medical service. The attackers can also target the economical stability by shutting down the banking systems.

DoS / DDoS attacks can be less profitable than other type of attacks and might not capture attention (unless the attack is directed toward a high profile web service). However, considering the amount of time and resources required to restore the victim network services, the value of restoration can be considered a profit for the victim competitors since they will have a bigger share of business while the victim is trying to recover.

Other examples do exist; however, the bottom line is that most if not every thing these days run on a computerised network with access to the internet. Therefore, every network can be subject to DoS / DDoS attacks from either internal or external sources.

## **1.4 RESEARCH OBJECTIVES**

This research aims to study the causal relationship between firewall logs and DoS / DDoS attacks. While a firewall needs to be maintained by humans it also records its own activities in the form of log files. These logs can collect, store and analyse firewall activity data to subsequently provide a proactive mechanism to defend the network from future attacks. This research objective has been achieved by the development of a model called DoS Tracking and Detection Model (DoSTDM). DoSTDM does not remove the need for misuse IDS that deals with known intrusion attacks, however, it can complement the security of IDS to detect new attacks that IDS systems have no pre-defined rule or signature to detect. DoSTDM is trying to close the gap between firewalls and IDS when it comes down to new threats and attacks generated by a DoS / DDoS flooding of packets.

## **1.5 ADDRESSING THE RESEARCH PROBLEM**

This research addresses the problem using a four phased approach as shown in Figure 1 – 6. Phase one investigates the past and current research papers and studies within the problem domain. Phase two involved the design of the DoS Tracking and Detection Model (DoSTDM) architecture. Phase three involved testing and revising the model architecture using a simulated case study network structure. Finally Phase four encompassed documenting the research findings. The research followed a systematic process to design, test and analyse the proposed DoSTDM architecture. In summary the following steps were taken to achieve the new model:

- Building a simulated WAN environment with firewall capabilities.
- Data sampling from a live firewall environment.
- Preparation of collected data before presenting it to the detection model.
- Designing of different data mining models using both statistical and neural network algorithms.
- Quantitative analysis of each model's performance.
- Comparison of the different models' performance using the same data ranges.
- Selection and discussion of model limitations that lead to accept or reject the research questions.

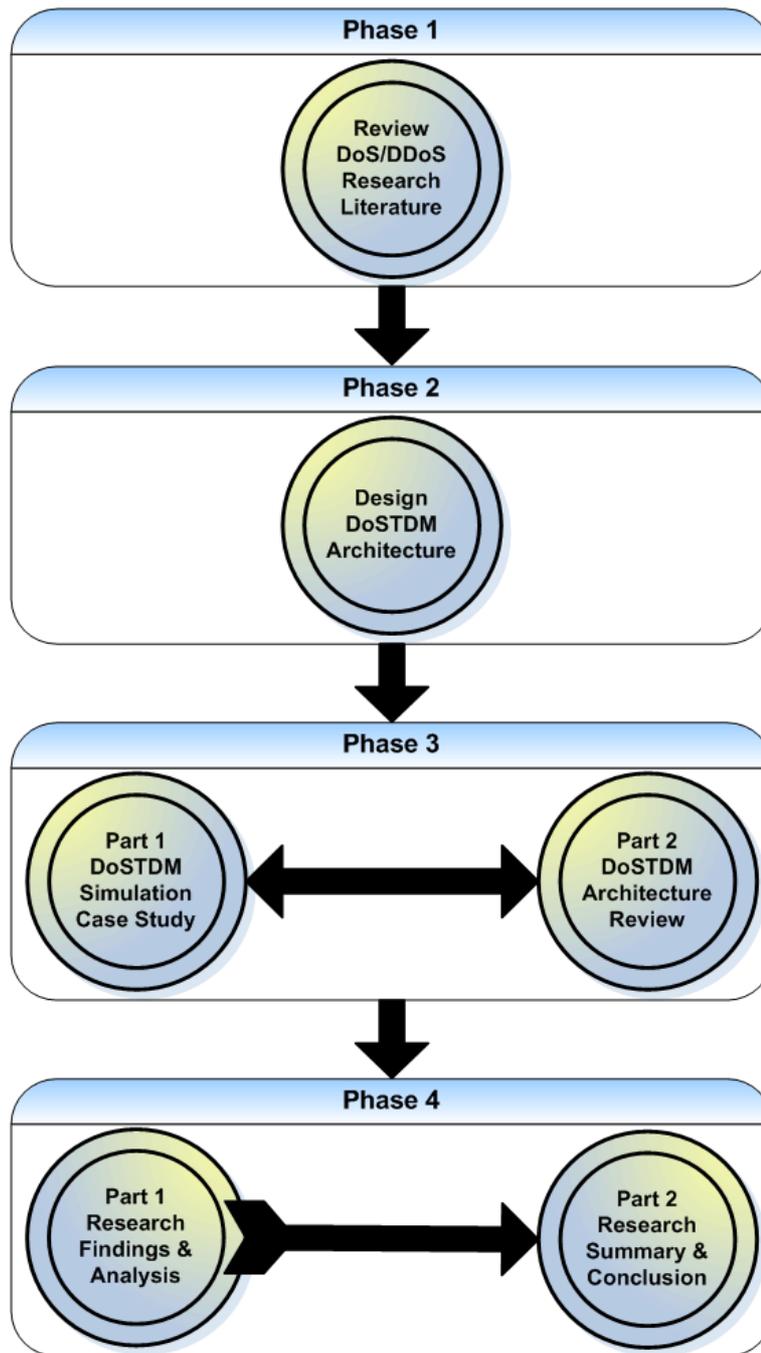


Figure 1 – 6: Thesis Research Approach

## 1.6 RESEARCH SIGNIFICANCE AND CONTRIBUTION OVERVIEW

The reduction of DoS / DDoS attacks will give computer networks attached to the internet more stability and reliability to deal with other problems caused by internal network activities. Firewalls are good, however, they need to be continually monitored and analysed in order to be more efficient against attacks (especially DoS attacks).

Early detection of an attack against the network aids fast elimination and more effective network protection, and researching this area can enhance our understanding of how to provide more protection to large networks that are connected directly to the Internet.

## **1.7 THESIS STRUCTURE OVERVIEW**

This thesis is divided into seven chapters. Chapter two is a literature review studying past and current research in information systems security emphasising the need for researching into DoS and DDoS prevention and existing research models used to approach and resolve the problem domain.

Chapter three describes the research methodology used to carry this research, discussing the quantitative positivist approach applied using a case study combined with a simulation experiment.

Chapter four describes the data collection process and data structure analysis. The chapter explore firewall logs TCP / IP fields as a base line for data collected from the test case network and also from the simulated network to pre-process DOSTDM data structures.

Chapter five describes the proposed architecture for the new DoSTDM model as an alternative DoS protection design to existing role based IDS / IPS. The new model designs use a mixed approach of statistical and neural network models as a forecasting engine.

Chapter six describes the implementation phase of the DoSTDM model in a simulated network, and it's testing with pre-collected data from a real firewall network.

Chapter seven discusses the research contribution, findings, limitations and future research directions.

# CHAPTER 2

## PROBLEM DOMAIN LITERATURE REVIEW

This chapter investigates network security for computer networks with emphases on the way Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks target and affect the security of information systems and their availability. This literature review aims to understand existing DoS and DDoS attacks and defence mechanisms and provides the theoretical background for designing the DoSTDM model (discussed in Chapter 4) for tracking and detecting flooding types of DoS and DDoS attacks.

Chapter two is divided into six main sections:

- Section one introduces a summary of information systems security and the aim of this thesis from a DoS problem context.
- Section two examines computer networks attacks and threats in general with a focus on DoS and DDoS threats and includes an overview of required defence mechanisms components.
- Section three investigates existing computer network systems firewall technologies design, implementations and limitations.
- Section four examines Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS) implementations and the role they play in protecting against DoS and DDoS attacks.
- Section five presents related research on DoS and DDoS defence models and how the thesis selected the appropriate anomaly detection technique for a DoS Tracking and Detection Model (DoSTDM) was selected.
- Section six provides a conclusion of this chapter.

## 2.1 INTRODUCTION TO INFORMATION SYSTEMS SECURITY

With the introduction of Internet in the mid 1990s and the subsequent wide-spread use of it, information systems security has become a major challenge to any organisation that participates and communicates openly over the Internet. Venter and Eloff (2003) categorise information security approaches into either reactive or proactive types (see Figure 2 – 1) with a variety of structured levels of protection that can and need to be implemented to maintain information security.

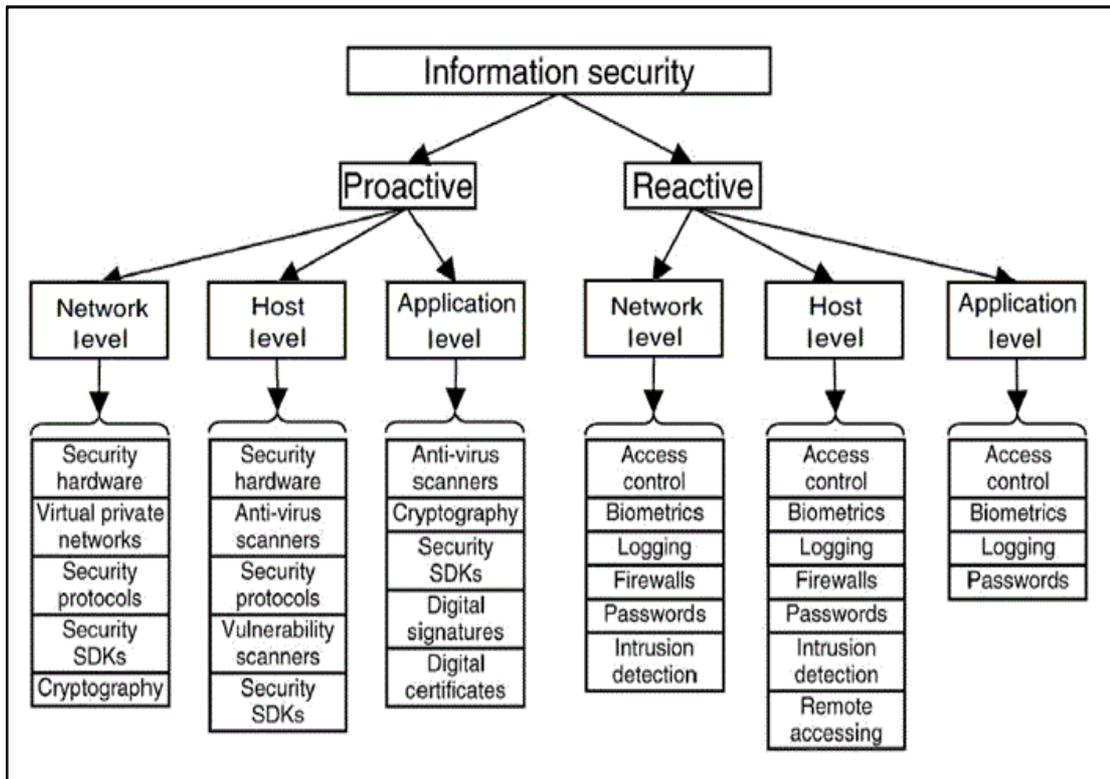


Figure 2 – 1: Taxonomy of Information Security Technologies  
Source: (Venter & Eloff, 2003, 299)

A proactive approach is an approach where all security measures have been taken and enforced before any security breach or attack can occur toward the network. Cryptography, security protocols, anti-virus and vulnerability scanners are some examples of this information and network security approach. A reactive approach is an approach where additional security measures will be activated as soon as the system security has been breached. Firewalls and IDS are examples of a reactive security approach. (Venter & Eloff 2003).

An example of how reactive and proactive approaches work can be seen when comparing Network-Based and Host-Based implementations of intrusion monitoring agents (Pescatore & Stienon 2003). The IDS and IPS monitors can be installed on either network or host infrastructures as illustrated by Figure 2 – 2.

The Network-Based monitors can be considered as a proactive approach as it continues to monitor the network before and after attacks. On the other hand, Host-Based monitors are more of a reactive approach (from a network point of view) that monitor the host server and react to attacks before the reach networks workstations.

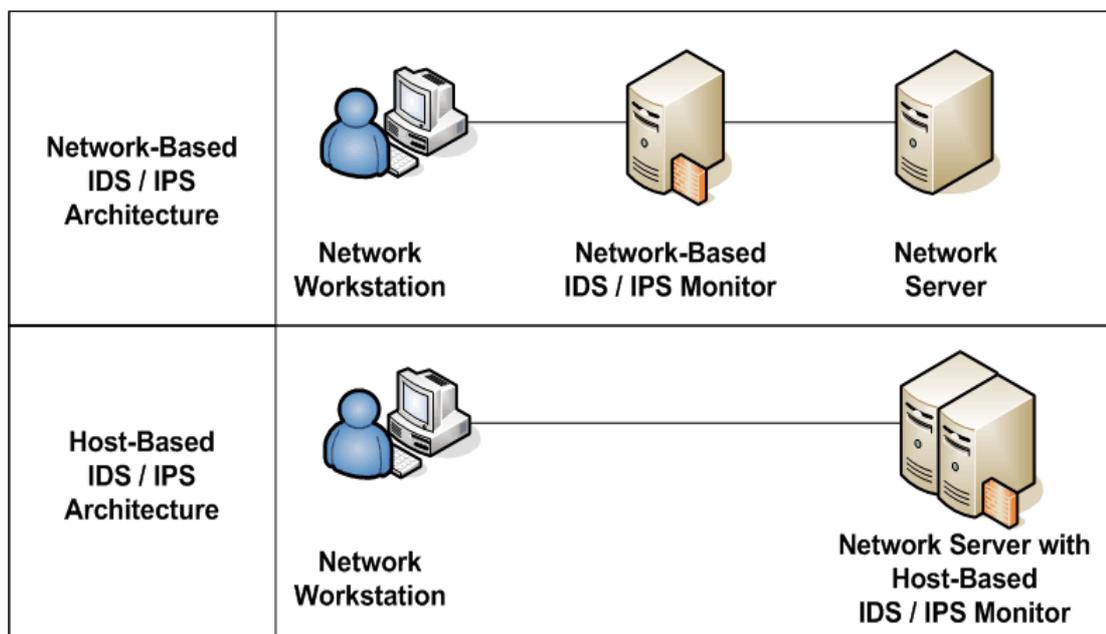


Figure 2 – 2: Network Based IDS / IPS vs. Host Based IDS / IPS Architecture

## 2.2 NETWORK SECURITY THREATS

### 2.2.1 Network Security Threats Overview

Early computer networks used a classic host based security model developed around access control policies and authentication databases model to protect information from intruders (Escamila 1998). This was appropriate for the time, however with the ubiquitous penetration of networks for the majority of business functions, host based solutions are no longer the most effective means of providing networks with protection against attack.

McEachen and Zachary (2007) identify three types of security penetration attacks:

- External penetration: Occurs when a non-legitimate user accesses a computer that he / she is not authorised to access.
- Internal penetration: Occurs when a legitimate computer user accesses data and resources that are prohibited by local computer security policies.
- Misfeasance: Occurs when a legitimate computer and resource user abuses local computer and resources security policies.

The most dangerous type of attacks is the misfeasance attack and the attacker can be identified by abnormal activities (McEachen & Zachary 2007).

McEachen and Zachary (2007) and Stallings (2006) classify intruders to three classes of computer and resource users:

- Masquerades: An unauthorized user.
- Misfeasors: A legitimate users who have access but misuses it.
- Clandestine: A user who gains supervisory control of a system to bypass auditing and security control.

Intruders can be either internal or external to networked systems and firewalls cannot protect against every type of attack, including those that incorporate malicious programs, trap doors, trojan horses, viruses and worms. Most security attacks will be attached to valid packet datagrams that pass through the firewall and possibly also to valid users and network-attached nodes in the case of an internal attack. Therefore, other mechanisms and techniques have been proposed to protect against these types of networking attacks (Stallings 2006, Escamila 1998). Durst et al. (1999) classified computer systems attacks based on attack topology and complexity (see Table 2 – 1).

Newman (2006, 69) classifies attacks against enterprise organisation networks as either active or passive (see Table 2 – 2) where:

- *“An active attack involves some modification of the data stream or attempts to gain unauthorized access to computer and networking systems”*
- *“A passive attack would include monitoring and eavesdropping on transmission”*

<b>Attack Topology</b>	<b>Description</b>
Local host attack	Attack occurs on the local host by a user on that host. This kind of attack generally involves a legitimate user seeking unauthorized privileges. The entire attack takes place on the same machine and does not travel across the network. It is detectable only by host-based IDS on the same machine.
Local subnet attack	Attack occurs on a host by an intruder using another host on the same subnet. This attack takes place over the same network segment but never goes through a router. It can be detected by the victim's host-based IDS and by network based IDS on the same subnet.
Local domain, remote subnet attack	Attack occurs from a host on one subnet to a victim on another subnet, both of which are still in the same network domain. This attack can be seen by the victim's host-based IDS, by network-based IDS on the affected subnets, and by router-based IDS on the bridge(s) between the affected subnets. This attack does not cross the boundary of the domain, which is where many IDS exclusively monitor.
Foreign domain attack	Attack that originates from a machine outside the domain to compromise a machine on the inside. This attack is visible to the victim's host-based IDS, to network monitors on all network segments leading to the victim, to IDS on routers bridging those subnets, and to any domain-wide IDS.
<b>Attack Complexity</b>	<b>Description</b>
One-to-one	Attacker uses a single machine to attack a single target machine. Example: sendmail bug.
One-to-many	Attacker uses a single machine to attack many targets. Example: probes, denial of service attacks.
Many-to-one	Attacker divides assault among multiple outside machines to attack a single victim. This is difficult to detect because multiple connections from multiple sources look more innocent than multiple connections from a single source. Example: SYN flood using IP spoofing to deny services.
Many-to-many	Many collaborating attackers divide the tasks of probing and attacking multiple victims. This poses the same challenge as the "many-to-one" cases, with the added complexity of multiple target machines. This kind of attack is very difficult to detect. Example: "Smurf" attack from multiple sources.

Table 2 – 1: Topology and Complexity of Computer Intrusions

Source: (Durst et al., 1999, 58)

Active Threats	Passive Threats
Denial of service Masquerade Modification of message contents Replay	Release of message contents Traffic analysis

Table 2 – 2: Active and Passive Threats

Source: (Newman, 2006, 69)

The complexity of network threats has grown over the past decades from simple to complex attacks while the technical knowledge of average attackers is declining (Lipson 2002). The technical knowledge seems to be needed by the developers of the attack scripts but the majority of novice attackers are only using these scripts to perform the attacks (see Figure 2 – 3).

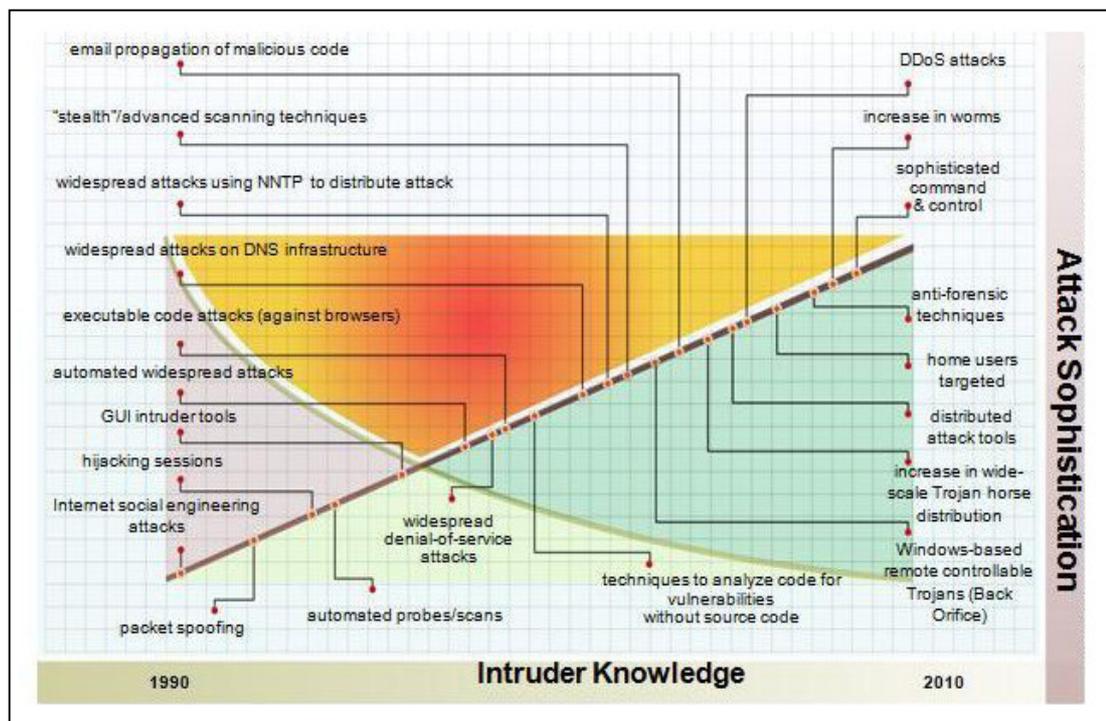


Figure 2 – 3: Attack Sophistication vs. Intruder Technical Knowledge

Source: (SIA 2011)

Software developers and hackers continue to discover new vulnerabilities and exploit them on regular basis. Therefore, patching targeted victims against these vulnerabilities turn to be a tedious exercise that can only provide short term protection. Lipson (2002) show this as a cycle that will continue to evolve over time as shown in Figure 2 – 4.

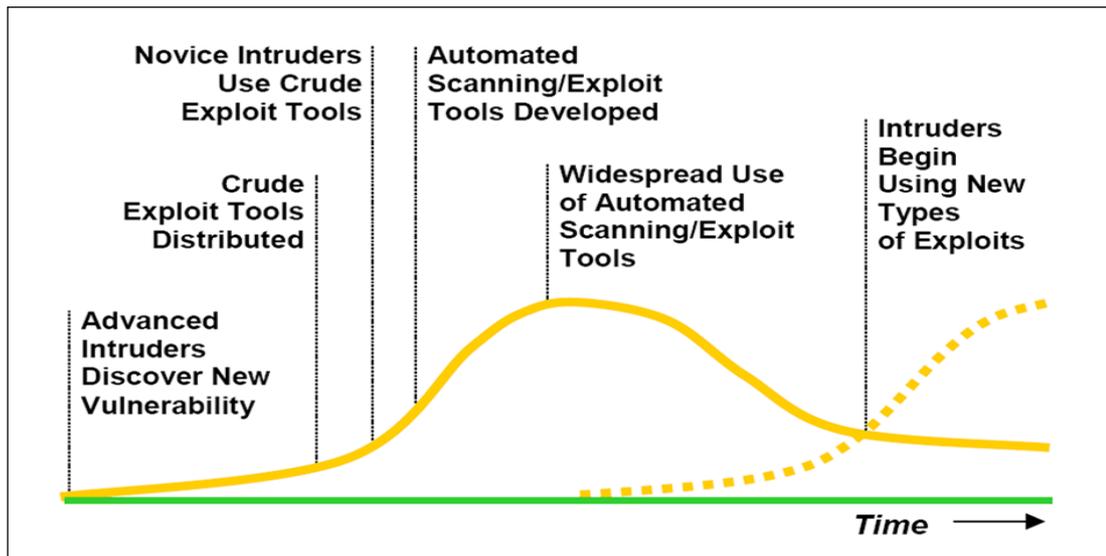


Figure 2 – 4: Vulnerability Exploit Cycle

Source: (Lipson, 2002, 11)

### 2.2.2 DoS and DDoS Attacks Definitions

DoS and DDoS attacks are among the most disruptive types of attacks that can be and have been waged against the Internet (Mitrokotsa & Douligieris 2007). They can be categorised into three types of attacks: flood attacks, protocol attacks and logical attacks (Li 2006). An examination of DoS attack definitions from various sources can provide a good understanding of DoS threats:

- FitzGerald & Dennis (2007, 387-388): “With a DoS attack, an attacker attempts to disrupt the network by flooding it with messages so that the network cannot process messages from normal users”
- Mirkovic & Reiher (2004, 40): “A denial of service attack is characterized by an explicit attempt to prevent the legitimate use of service”
- Newman (2006, 70): “Denial of Service attack is defined as an attack that attempts to deny computer and network resources to legitimate users”.
- Specht & Lee (2004, 543): “A Denial of Service (DoS) attack is an attack with the purpose of preventing legitimate users from using a specified network resource such as a website, web service, or computer system”
- Stallings (2006, 614)” “A denial of service (DoS) attack is an attempt to prevent legitimate users of a service from using that service.”

Distributed Denial of Service (DDoS) attack is a many to one case of DoS in which many sources target the same network resource. In this type of attack hundreds of thousands of Internet servers are under the control of a hacker targeting a single system to overload it and block any legitimate user's access to the target (Chen & Hwang 2006, Lee et al. 2008, Newman 2006). A DDoS attack can consume the attacked target system resources making it unable to provide any service (Stallings 2006). First appearing in June 1998 (Lee et al. 2008) DDoS attacks are much harder to trace than DoS, more severe and difficult to prevent and therefore mitigate (FitzGerald & Dennis 2007). The amount of flooding traffic that is generated by DDoS can crash a victim network server by exhausting its resources (Chen & Hwang 2006). DDoS damage was seen on a large scale back in 2000 when major web sites like Yahoo, eBay, CNN and Amazon were targeted by DDoS attackers (Lee et al. 2008, Malliga & Tamilarasi 2007). The effect of these DDoS attacks continued for several hours causing a complete service outage. Another form of DDoS attack style is called Reflective DoS (RDoS). This special case of DoS can occur while tracing back the attacker's packet sources using non attacking internet hosts as reflectors, so when the trace occurs, IP spoofing is used to mislead the trace to these reflectors causing more packets to flow from these hosts back to the attacked network from the internet (Wang & Schulzrinne 2004, Lee et al. 2005).

### 2.2.3 DoS and DDoS Attack Types & Classifications

Mitrokotsa & Douligeris (2007) classify DoS attacks into five categories depending on the attacked protocol level (see Table 2 – 3).

<b>DoS Attack</b>	<b>Example</b>
Network device level	Targeting software bugs in network devices
Operating System (OS) level	Attacking protocols implementations within OS like ping attacks to crash targeted machine.
Application level	Exploiting network application bugs to drain all available resources.
Data flooding attacks	Exhausting network bandwidth and resources by packets flooding (specially seen with DDoS attacks)
Specific protocol features	Targeting Domain Name Servers (DNS) cache and IP address spoofing.

Table 2 – 3: DoS Attack Classifications  
(Based on Classification by Mitrokotsa & Douligeris 2007)

The most common type of attacks are the TCP / IP protocols misuse DoS attacks, FitzGerald & Dennis (2007) provide a summary of six types of these attacks designed to create the maximum level of damage (including flooding attacks) to targeted network victims as follow:

1. *Internet Control Message Protocol (ICMP) ping flooding attacks*: The attacker pings the targeted network with a fake source IP address causing many ICMP reply messages to occur from the network computers that cannot reach the source generating a flood of message packets across the network.
2. *User Datagram Protocol (UDP) floods attack*: Similar to ICMP attack except that it uses UDP messages as a vehicle to carry the flooding attack.
3. *TCP SYN flood attack*: Attacker with a fake IP address swamps the network with repeated Synchronize (SYN) messages to establish a TCP connection with the targeted network. Since the network replies to the fake address it will never receive any replies, therefore, all of the victim memory resources will gradually be exhausted.
4. *UNIX process table attack*: Similar to the TCP SYN attack but it targets UNIX system processes by opening many connections that will never be completed leading targeted UNIX systems to run out of memory.
5. *Finger of Death attack*: Similar to the TCP SYN attack by initiating an endless finger requests that will never disconnect.
6. *DNS recursion attack*: Works by spoofing both source and target IP addresses when sending DNS requests causing the DNS to respond to a non-existing requester IP, this attack is stronger than the previous attacks as the size of DNS messages are normally larger than ICMP, UDP and SYN packets.

Patrikakis et al. (2004) described additional forms of DDoS attack like:

- Central Source Propagation that use protocols such as HTTP, FTP and Remote Procedures Callas (RPC) to propagate malicious code to target vulnerable systems turning these systems to Zombies (Zombies also known as Demons, however, in networked systems that's running on Unix platform, the word Daemons is another name for an active Unix services or process).
- Address Resolution Protocol (ARP) poisoning attack that cause known IP hosts to be identified with wrong Media Access Control (MAC) address at the switched LAN level.

- Secure Shell (SSH) process table attack that initiate hundreds of connection without completing the login process till the target is exhausted.

However, in order to understand DDoS attacks and the way to defend against these attacks, it is important to understand the way these attacks work. Mitrokotsa & Douligieris (2007) used a cross-set of classification criteria to classify different types of DDoS attacks based on the following characteristics (see Figures 2 – 5):

- Attack degree of automation: Early DDoS attacks were manually initiated by scanning remote machines to find vulnerabilities and weakness in order to break into these machines and install the attacking agents. Current attacks are either semiautomatic or totally automated in the way they scan and break through. This automation process includes hard-coding all the features of the attack within the attacking code such as the IP address of the attack handler, the target address and the attack duration. Therefore, this provides some saving in the breaking through time with a little exposure from the attacking side to the target.
- Exploited vulnerabilities and weakness: This aspect of DDoS defines the type of the attack by targeting victim resource weaknesses to initiate DDoS attack. The victim's weakest points are mostly the network bandwidth and the machine's local resources; if these two can be exhausted then the DDoS can successfully be completed. The network bandwidth can be exhausted by flooding attacks where the attacker sends a large number of UDP and ICMP packets causing network saturation, another way is to use amplification attacks by broadcasting the victim's IP address to all routers within the broadcasting range that will direct all traffic toward the victim's network connection. The victim's local resources can be exhausted by targeting a specific protocol bug (e.g. TCP SYN attacks that exploit the TCP handshake process) or by sending a malformed packet that confuses the victim's operating system.
- Attacking rate dynamics: DDoS attacks can be divided to either continuous or variable rate attacks. Increasing the attack rate gradually will exhaust victim resources slowly and delay the attack detection while continuous rate attack impact is very quick and faster to detect.

- DDoS attack impact: The impact of DDoS can be either disruptive or degrading to the victim. Disruptive attacks will render the victim services unavailable after the attack. Degrading attacks will consume victim resources gradually causing the loss of victim services over a longer period of time and delaying the detection process, which in return will increase the damage over the attack period.

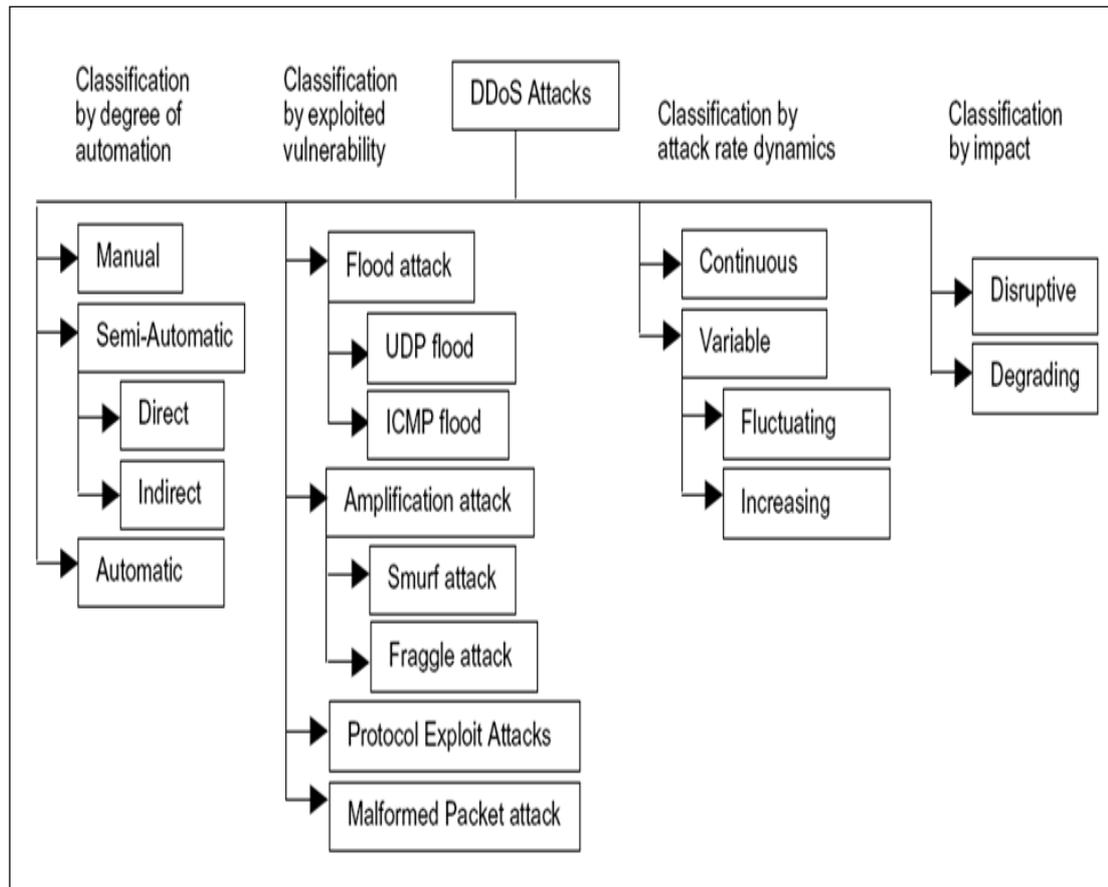


Figure 2 – 5: DDoS Attacks Taxonomy  
 Source: (Mitrokotsa & Douligeris, 2007, 124)

Mirkovic & Reiher (2004) described additional characteristics while classifying DDoS attacks as listed below:

- Secure address validity: This specifies the attack by investigating the attacking address to see if it is a spoofed or valid source address in order to establish attack accountability and responsibility.

- DDoS characterization possibility: This classification studies the attack to see if it can be characterised against specific protocols or not (e.g. TCP SYN attack). In addition it checks if the DDoS attack can be filtered from the network or not using firewalls (e.g. UDP flooding attack).
- DDoS victim type: This can be network, application, host, resource or the complete infrastructure.
- DDoS agent type: The agent set can be a constant or variable number of attacking agents.

#### **2.2.4 Impact and Motivation of DoS / DDoS Attacks**

The attacking speed of these worms causing DDoS attacks have resulted in huge financial losses of billions of dollars. A good example of that is an attacker called Mafiaboy (Wikipedia, 2012b) who managed to launch a series of DDoS attacks in February 2000 under operation Rivolta (the Italian word for riot). The DDoS targeted web sites like Yahoo, Fifa, Amazon, Dell, E-TRADE, eBay, and CNN. The initial cost of the DDoS attack was estimated to be US\$1.2 Billion, however, during an official trial that figure was downgraded to US\$7.5 millions by prosecutors.

However, It is very hard to differentiate the real cost from actual cost due to the fact of potential revenue being lost or estimated to be lost while the victim site is under attack. In the other hand, the cost to launching an attack can be as low as a two digits figure, such a low cost was offered by a hacker who offered a six hours outage for only US\$60 and the price can vary depending on the target victim (Golubev 2005). Some attacks can cost more to launch, for example it cost a US\$100 to rent a botnet for one day to launch a DDoS attack (Weiss 2012).

Therefore, the reactive approach of Internet users is insufficient and ineffective due to the following factors described by Pethia (2003):

- The large and increasing number of Internet connected computers.
- The ability to control large numbers of vulnerable Internet connected computers to launch wide scale attacks.

- New fully automated and complex attacking technologies have been developed and implemented by attackers.
- With the complexity of the attacks the required time to detect and react is also increasing.
- High dependency on the Internet by users and online business mean that such attacks can be highly damaging.

The majority of attacks implementing DoS / DDoS developed in the early years of DoS / DDoS as identified by Lin & Tseng (2004) were concentrated on flooding the network to cause services outages. However, flooding DDoS methods like ICMP, UDP and DNS floods are still current and represent the most common attack mechanisms used to launch DDoS attacks (Weiss 2012). The Prolexic (2012) attack report for the first quarter of 2012 has identified that the DDoS attacks increased by 25% compared to same quarter in 2011. The Prolexic (2012) report also identified that out of the total number of attacks in Q1 2012: 24.66% of attacks were SYN floods, 19.65% of attacks were ICMP floods and 15.41% of attacks were UDP floods. Therefore, these types of attacking mechanisms are still active and in use by DoS / DDoS attackers as shown in Figure 2 – 6.

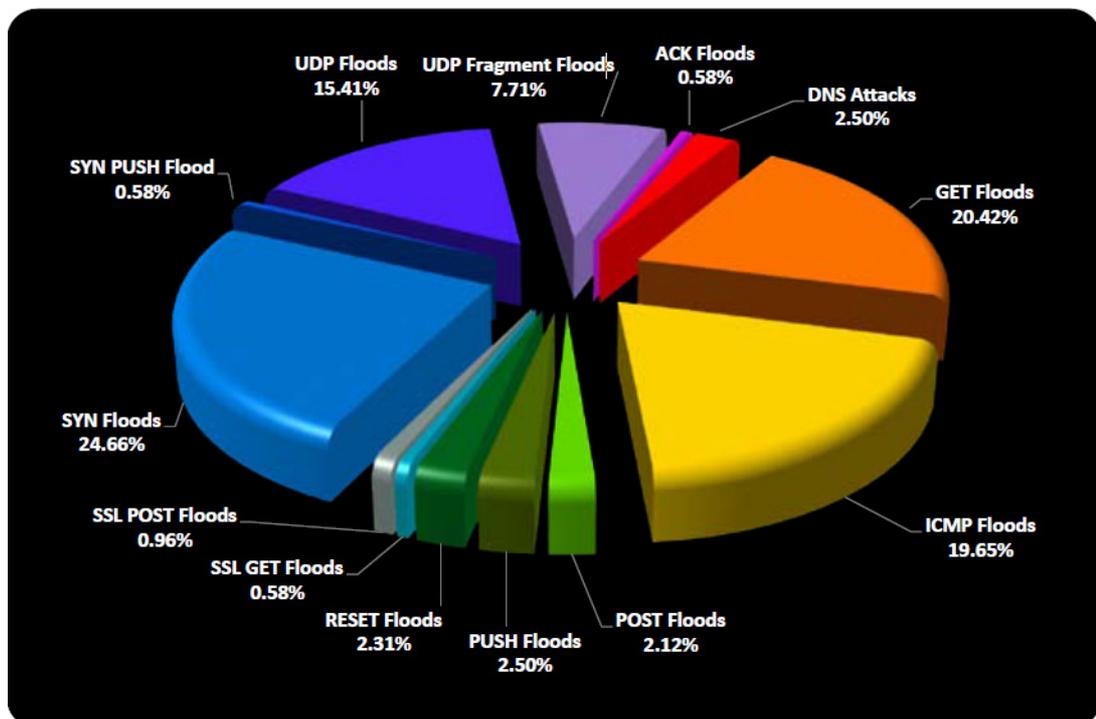


Figure 2 – 6: Total Attack Types (Q1 2012)  
(Source: Prolexic 2012, 4)

In addition, the attack speed was on the increase and spreading fast, Figure 2 – 7 illustrates a comparison of the attacking speed of three major worm attacks made by Code Red, Slammer and Blaster over a period of one day from attack establishment.

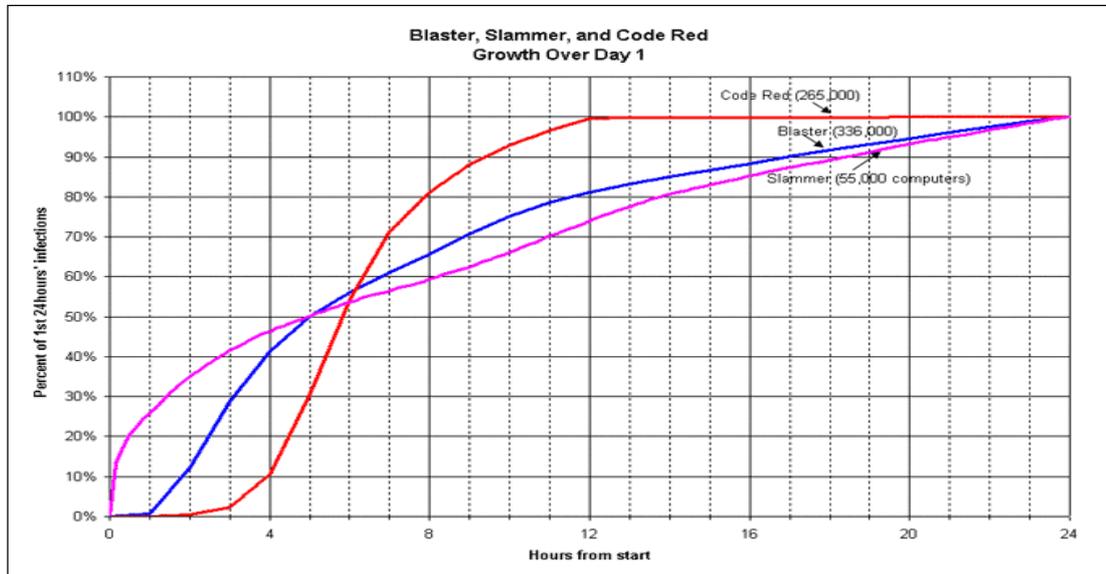


Figure 2 – 7: Code Red, Slammer and Blaster Attacks Growth over 24 Hours  
Source: (Pethia, 2003)

Latest statistics from Shadowserver Foundation (2012) show an increase of DDoS counts in 2009 over the previous years (see Table 2 – 4). The 2009 figures indicate that the total number of DDoS attacks for that year increased more than thirty-folds when compared to 2008. This indicates that the rate of DDoS attacks has not diminished, along with the losses to business associated with such attacks.

DDoS Statistics Description	Year					
	2006	2007	2008	2009	2010	2011
Unique Command and Controls conducted DDoS attacks	414	848	618	590	430	322
Unique ASN's did the Command and Control servers reside in that participated in DDoS attacks	214	390	332	272	157	98
Unique countries did the Command and Control servers reside in that participated in DDoS attacks	40	67	66	53	41	30
DDoS attacks occurred during that period	50650	35566	202678	7058221	1545208	275459
Different targets were DDoS'ed	25953	15755	21312	10991	13757	5327
Unique ASN's were affected by the DDoS's	3079	1633	1870	1491	1697	756
Unique countries were affected by the DDoS's	133	107	117	110	106	72

Table 2 – 4: DDoS Stats 2006 - 2011  
Source: (Shadowserver Foundation, 2012)

The Network Security (2002, volume 9, page 3-3) news article titled “*who has been hit by the big bad DDoS* “ reports how attackers using DDoS targeted the Internet Service Provider (ISP) CloudNine in January 2002 forcing this small UK ISP to close. This is a pertinent example of the impact of DDoS attacks on the business e-community. In addition, the same report mentioned that 15000 servers were targeted by the Code Red worm causing governments and business to fall victims of DDoS attacks. The list provided in the report did not show any direct relation between these targets, providing no common motive behind the attacks other than that the majority were e-commerce entities. One of the largest sites that were hit by the Code Red DDoS attack was USA `www1.whithouse.gov` that was forced to change its IP to mitigate the attack (Lamp 2001). Recent attacks included individual (e.g. the attack against Gary Kasparov political party websites during 2008 elections in Russia) and governments as well attacks like the massive scale DoS / DDoS politically motivated attacks reported against Estonia in 2007 and Georgia in 2008 (Talihärm 2010). The main aim of DoS and DDoS attacks is to damage the target victim server, network services and data availability. Nazario (2008) and Mirkovic & Reiher (2004) categorise the motivations behind DDoS attacks (as they vary from one case to another) into three main categories:

1. Retaliation and/or anger toward specific victims such as home users with access to Internet or sites that support and produce scams, phishing and spam.
2. Financially motivated attacks against e-commerce, online gambling and pornography sites demanding ransoms to prevent further attacks.
3. Politically motivated attacks.

### **2.2.5 Initiation Models for DoS and DDoS Attacks**

Nazario (2008) mentioned the DoS / DDoS attacks can be initiated simply by asking as many people as possible to visit a website which will generate a flood of requests putting the website stability in danger. However, a DDoS attack normally follows two initiation models: the Agent-Handler model (see Figure 2 – 8A) or the IRC-Based model (see Figure 2 – 8B). The Agent-Handler model uses handlers which are a set of communication software living in the Internet servers providing a communication link over TCP, UDP and ICMP protocols between the attacker and the agents. Agents are networked machines attached to the Internet that are unaware of their role of attacking the victim (Mitrokotsa & Douligeris 2007, Specht & Lee

2004). The relationship between handlers and agents are referred to as “master” and “daemons” (Specht & Lee 2004). The IRC-Based model uses Internet Relay Chat (IRC) protocol servers to establish communication between the attacker or attackers and the attacking agent’s machines (Specht & Lee 2004, Mitrokotsa & Douligeris 2007). The first DDoS attack used a simple script that listened to IRC channels in order to command multiple hosts to attack the targeted victim machine using ping commands that could be executed while channel management scripts (e.g. eggdrop) were running (Nazario 2008).

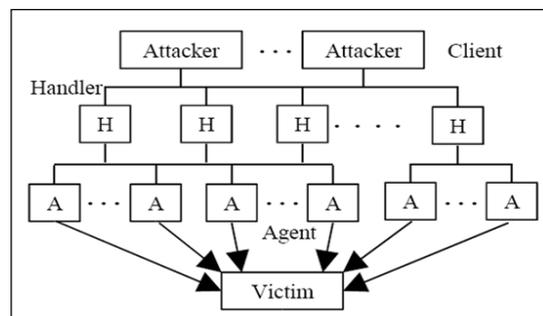


Figure 2 – 8A: DDoS Agent-Handler Attack Model  
Source: (Specht & Lee, 2004, 544)

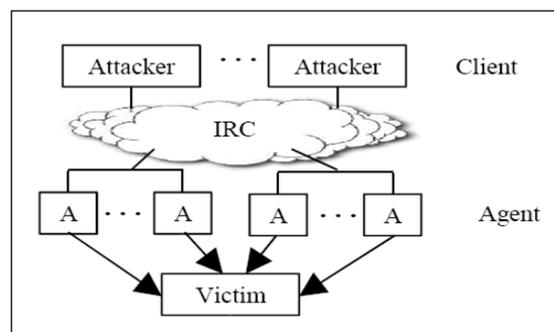


Figure 2 – 8B: DDoS IRC-Based Attack Model  
Source: (Specht & Lee, 2004, 544)

Tracing DDoS attacks in general from both models is a difficult task since DDoS agents normally spoof the source IP address to hide their identity (Lee et al. 2008). However, tracing and detecting IRC-Based DDoS attacks is a more challenging task due to the fact that identifying a participating agent will only reveal the IRC server used as a communication channel but not the rest the agents or the attacker(s) identities. In addition, IRC-Based DDoS models provide anonymity to both attacker(s) and agents (Mitrokotsa & Douligeris 2007).

## 2.2.6 DoS & DDoS Detection Algorithms

Verwoerd & Hunt (2002) identified the two general mainstream models used for DoS / DDoS detection as either anomaly based or misuse based models. Table 2 – 5 list a summary of each model with its implementation approaches.

Detection Model	Implementation Approach
Anomaly Based Detection	<p><b>Statistical Modelling:</b> Analyse network events using any of the following:</p> <ul style="list-style-type: none"> <li>a) Measuring traffic threshold</li> <li>b) Calculate mean and standard deviation</li> <li>c) Multivariate study of events correlation and relativity to expectations.</li> <li>d) Markov process study of events state as variables (Markov Process Model).</li> <li>e) Vector representation of events to group and classify based on events behaviours and patterns (Cluster Analysis).</li> </ul>
	<p><b>Immune System Approach:</b> Modelling applications behaviour in the network to establish a base line of application events. However, it is limited to traditional attacking techniques but can not detect attacks that corrupt time critical interaction between different process and systems (also known as Race Condition Attack), policy violations and masquerading attacks.</p>
	<p><b>Protocol Verification:</b> Check for malformed fields and compare it to established protocol behaviour</p>
	<p><b>File Checking:</b> Searching for data changes in files to detect anomalies.</p>
	<p><b>Taint Checking:</b> Used to check embedded shell commands in the application traffic.</p>
	<p><b>Neural Networks:</b> Used to classify events as either normal (that matched pre-trained data) or anomalies.</p>
	<p><b>Whitelisting:</b> This is a data reduction filtering technique that passes the events through multiple filters of known network activities and the remaining events are treated as novel or suspicious.</p>
Misuse (Signature Based) Detection	<p><b>Expression Matching:</b> This approach searches network events (logs or network sessions streams) for specific patterns and signatures.</p>
	<p><b>State Transition Analysis:</b> This is a complex approach that traces network events states and transitions. Every event is checked against a set of attacks, when an event reaches a final state of acceptance it is then considered to be an attack.</p>
	<p><b>Dedicated Language:</b> This involves writing attack detection signatures by different programming languages, however, this can vary from being a simple to a complex exercise to isolate the attack within network events.</p>
	<p><b>Genetic Algorithms:</b> Search events for combinations of known attacks as binary vectors to weight attacking risk and re-test matching vectors to reduce false positive and negative alerts.</p>
<p><b>Burglar Alarms:</b> This approach uses dedicated monitors to find policy violations and any thing that should not exist within network events to trigger alarms. High understanding of the network environment is needed to configure the appropriate triggers.</p>	

Table 2 – 5: Summary of Anomaly Based and Misuse Based Detection Models  
(Summary Based on Verwoerd & Hunt 2002)

In addition, Chen et al. (2004) characterize the detection algorithms for detecting DoS and DDoS attacks by adding congestion and source based detection algorithms to anomaly based detection algorithm as follow:

1. *Congestion based detection algorithm*: Can only be applied while the network is congested and can cause a lot of false positive alerts raised by normal network traffic at the time of the attack.
2. *Anomaly based detection algorithm*: Can only be applied to TCP SYNC packets and need to have a TCP protocol and sub-protocols (e.g. ICMP and UDP) thresholds to be known before it can declare a DoS attacks. This method can also generate false positives if the normal network traffic reaches the pre-defined thresholds.
3. *Source based detection algorithm*: Can be used if the source attacker utilises a spoofed IP address, however, if the system cannot distinguish between an original trusted IP address and a spoofed IP it will generate a false alert.

### **2.2.7 DoS & DDoS Defence Mechanisms**

The Mitrokotsa & Douligieris (2007) taxonomy describes two criteria to achieve DoS and DDoS attack protection (as shown in Figure 2 – 9), the first criteria is based on the attack activities and the second on attack location of the targeted network. However, IDS systems need to be included within the network firewall design phase. For example, IDS are normally placed in the Demilitarized Zone (DMZ) of the network to act as a buffer zone between external and internal firewalls implemented normally at hardware level using routers (Keromytis & Prevelakis 2007).

Figure 2 – 10 illustrates the design of a firewall with a DMZ buffer zone that can accommodate IDS and other types of add-on services like DNS, mail gateways and Network Address Translation (NAT). In addition, Intrusion Prevention Systems (IPS) solutions are usually marketed in the form of a firewall with an IDS system (Cardoso 2007). Therefore, DoS and DDoS defence mechanism structures need to include firewalls, IDS and IPS mechanisms.

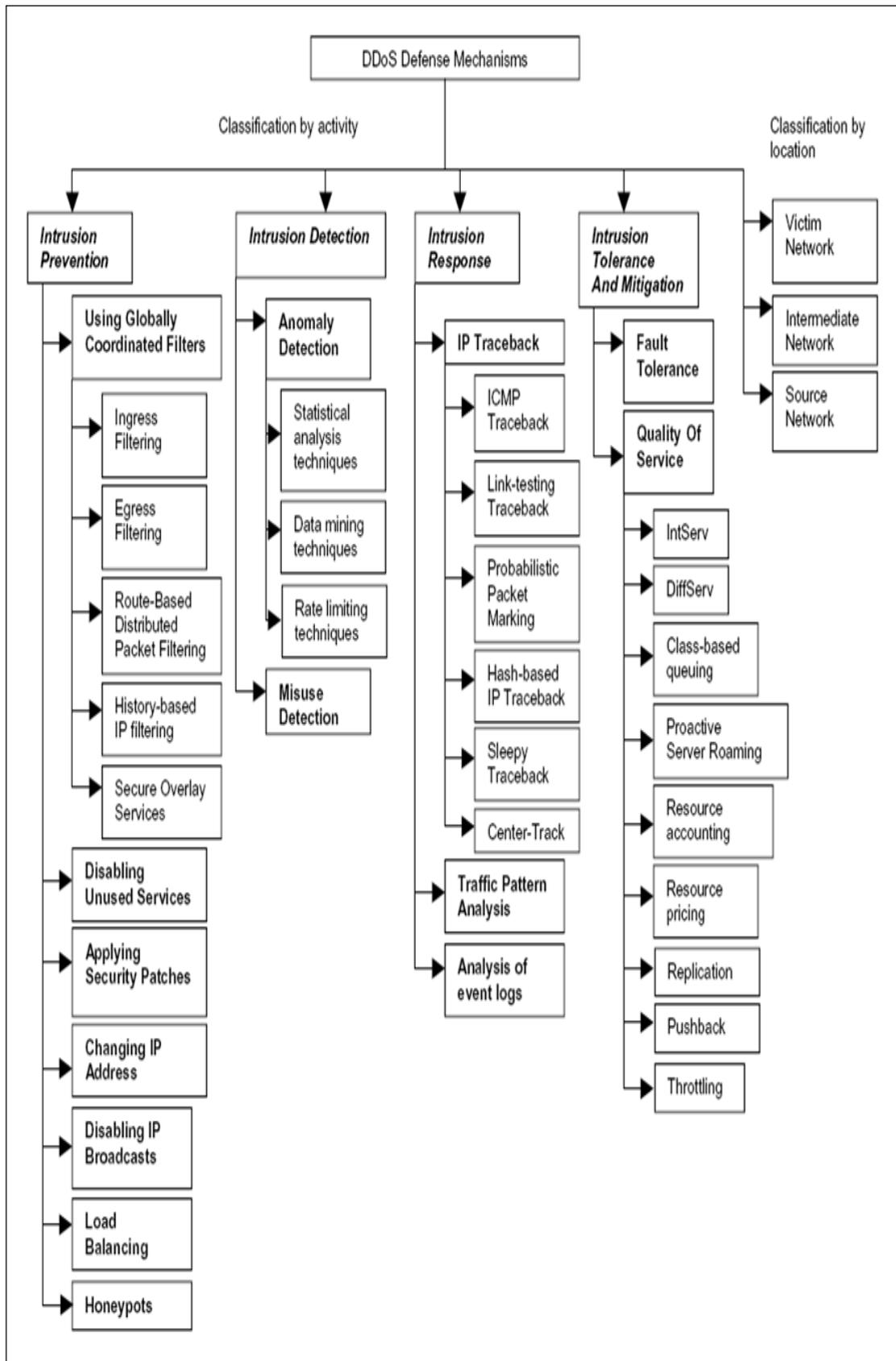


Figure 2 – 9: Taxonomy of DDoS Defence Mechanisms

Source: (Mitrokotsa & Douligieris, 2007, 128)

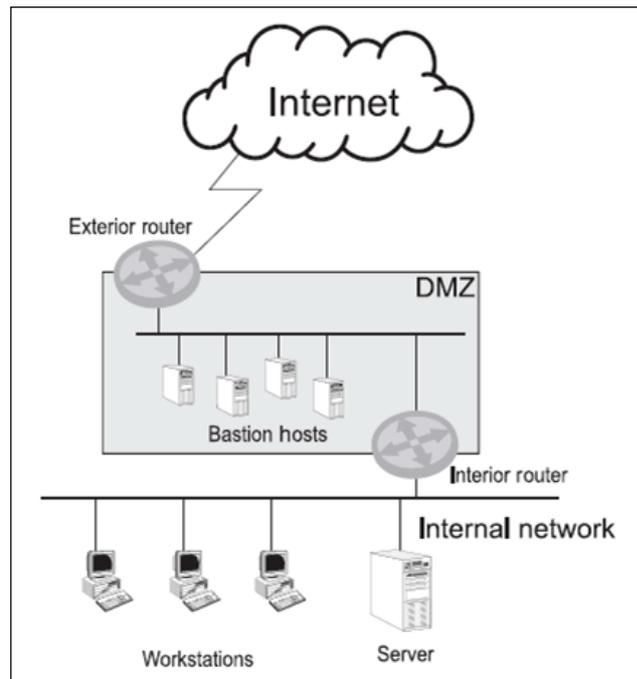


Figure 2 – 10: Typical Firewall Configuration with DMZ  
 Source: (Keromytis & Prevelakis, 2007, 34)

### 2.3 FIREWALL DEFENCE MECHANISMS

Kizza (2013) define the firewall function as: “a tool that provides a filter of both incoming and outgoing packets”. Sudhir S. et al. (2013) defines the firewall as: “a device that allows multiple networks to communicate with one another according to a defined security policy”. The primary rule of any firewall is to protect the network based on pre-defined rules and policies designed as per the computer network local security best practice policy (Smith & Bhattacharya 1999, Sudhir S. et al. 2013). Firewalls can achieve there security protection function by using either packets filtering (to accept or deny access) or application proxy gateways that provide protection to users from outside bad users (Kizza 2013). Firewalls use a screening process for every packet using rules and policies to detect attacks; therefore, this process is very important and sensitive to the security of the whole network infrastructure. The configuration of firewall rules and policies is very important to differentiate between normal network activities and attacks. A packet travelling across the network firewall using TCP / IP represents a network communication activity that can be an attack against any network host sitting behind this network firewall (Oppliger 1997, Sudhir S. et al. 2013).

Network activities (normally logged by the firewall) are very rich in data and can be analysed later to obtain further information about existing and possible future attacks. It can also be used to evaluate firewall security, performance and management levels (Noureddien & Osman 2000).

In addition, firewall should have a set of management tools (provided by firewall vendor or third party vendors) that provides a reporting functionality for the firewall packet screening process and activities. This reporting function can help in identifying:

- Intruders attempts.
- Illegal access to internet resources by internal users
- Firewall overall screening and performance.

Therefore, firewall reporting process is another important function of firewalls to alert the firewall system administrator of any possible issues that can affect firewall environment.

### **2.3.1 Firewall Implementations**

Even though current implementations of firewall systems can be done on a variety of software and hardware systems, the basic idea behind firewalls remains the same.

The hardware type of firewall can be found on routers which represent a packet filtering firewall implementation, while a dedicated “Check Point Firewall-1” is a fully featured capable software firewall that can run on a variety of UNIX, Linux and Microsoft Windows operating systems platforms (Wool 2004). Figure 2 – 11 shows a typical implementation of a firewall within a computer network. Packets inspected by the firewall normally contain the source IP address, source port number and destination IP address (Hamed & Al-Shaer 2006). Packet filtering firewalls work by reading the source IP address from each packet header (Stallings 2006). Packet filtering design scans incoming and outgoing packets to either allow or deny the traffic as follows:

- If the packet is allowed to or from the network it will be forwarded to the next hub as per the routing table information for this packet’s destination.
- If the packet is denied it will be discarded.

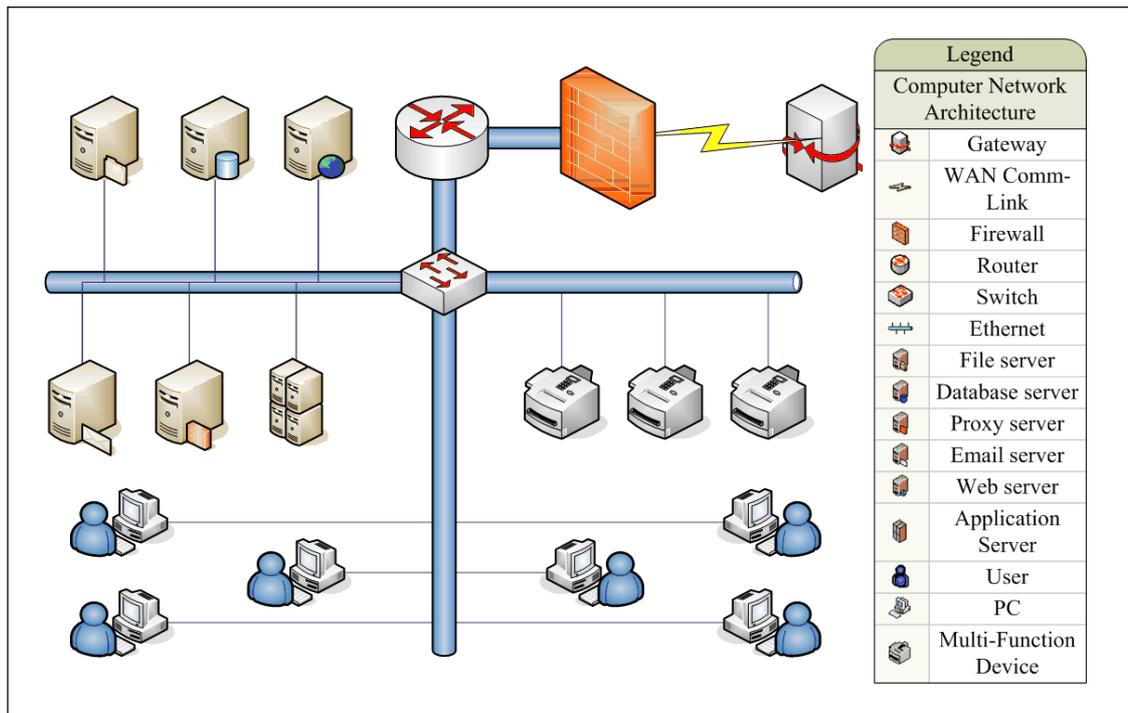


Figure 2 – 11: Firewall Implementation in Computer Networks Architectures

Firewall packet filtering policies are applied within the firewall to “allow” or “deny” any packet passing through the network, see Figure 2 – 12 for a firewall policy example.

protocol	source address:port	destination address:port	action
1 tcp,	*.*.*.*:any,	161.120.33.41:25,	allow
2 tcp,	140.192.37.30:any,	*.*.*.*:21,	deny
3 tcp,	*.*.*.*:any,	161.120.33.*:21,	deny
4 tcp,	140.192.37.*:any,	*.*.*.*:21,	allow
5 tcp,	*.*.*.*:any,	161.120.33.*:22,	allow
6 tcp,	140.192.37.*:any,	*.*.*.*:80,	deny
7 tcp,	*.*.*.*:any,	161.120.33.40:80,	allow
8 tcp,	*.*.*.*:any,	161.120.33.43:53,	allow
9 udp,	*.*.*.*:any,	161.120.33.43:53,	allow

Figure 2 – 12: Typical Firewall Packet Filtering Policy Example

Source: (Hamed & Al-Shaer, 2006, 333)

The second type of firewall is an application level firewall that works at the application level to allow network services to be established and used within predefined policy criteria controlled by firewall policies (e.g. allow or deny services such as HTTP, Telnet, FTP, mail, etc.).

The main difference between packet filtering and application level firewalls is that packet filtering firewalls do not understand specific application access policies as packet filtering works at the transport layer of the network like mail gateways using Simple Mail Transfer Protocol (SMTP) and Internet access using Hypertext Transfer Protocol (HTTP) and many other application level protocols (Keromytis & Prevelakis 2007).

Other forms of firewall technologies exist and are implemented focusing on protocol activities like Stateful Firewalls and packet contents like Deep Packet Inspection (DPI) firewalls (Keromytis & Prevelakis 2007, Noakes-Fry 2004). Stateful Firewalls is a form of packet filtering firewall (Noakes-Fry 2004) that is required for some protocols like FTP which relies on a secondary connection to exchange and send information.

In general firewalls tend to reject the secondary connection as it deals with connections and packets individually. Therefore, Stateful firewalls are used to maintain the status of connection for specific protocols in order to avoid rejecting legitimate secondary connections (Keromytis & Prevelakis 2007). DPI firewalls still look at individual packets and inspect the packet contents in an attempt to identify harmful activities within the packet based on pre-defined signature and traffic anomalies (Noakes-Fry 2004).

### **2.3.2 Firewall Quality & Performance**

As with any technology, firewalls are designed by humans and need human intervention in order to be kept up-to-date with latest security patches and rules configuration. The issues with human intervention start at the design level of firewall security for different network types. This requires defining the best system of firewall services that can be provided.

Firewall technologies alone can't provide complete security for any organization and human firewall administrator interaction cannot be avoided. However, firewall quality can be measured by its policy efficiency of allowing legal traffic only and disallowing any illegal traffic (Cavusoglu et al. 2004, Al-Haj & E. Al-Shaer 2011).

In addition, the term Quality of Firewall (QoF) (Xu & Singhal 1999) has been proposed as a novel firewall design concept for high performance networks. A major part of QoF in protecting networks with firewalls is traffic monitoring and packet filtering as the screening process component undertaken by the firewall.

Sheth & Thakker (2011) identified the need behind a quality design for a firewall to be driven by current IT and business regulatory standards such as:

- Sarbanes–Oxley Act of 2002 (also known as the 'Public Company Accounting Reform and Investor Protection Act').
- COBIT business framework for the governance and management of enterprise IT.
- Payment-Card Industry Data Security Standard (PCIDSS)
- National Institute of Standards and Technology (NIST).

With the introduction of some new protocols to the Internet such as Internet Protocol Version 6 (IP V6) to replace existing IP V4, IPSec and secure certificate systems used with Secure Shell Layer (SSL), it becomes more important to revise the way current data pattern matching is used by firewalls. These need to be up-to-date in order to be able to screen and filter datagram packets that use these new protocols (Smith & Bhattacharya 1999). This adds complexity to how firewalls are managed, however, attack patterns in principal remain the same but the method of carrying out the attack over TCP / IP has changed.

Finally, only a small set of vendors is considered to be market leaders in producing enterprise level network firewalls (Checkpoint Software Technologies and Palo Alto Networks).

Figure 2 – 13 identifies current enterprise level firewall providers as classified by Gartner research (Young 2013).

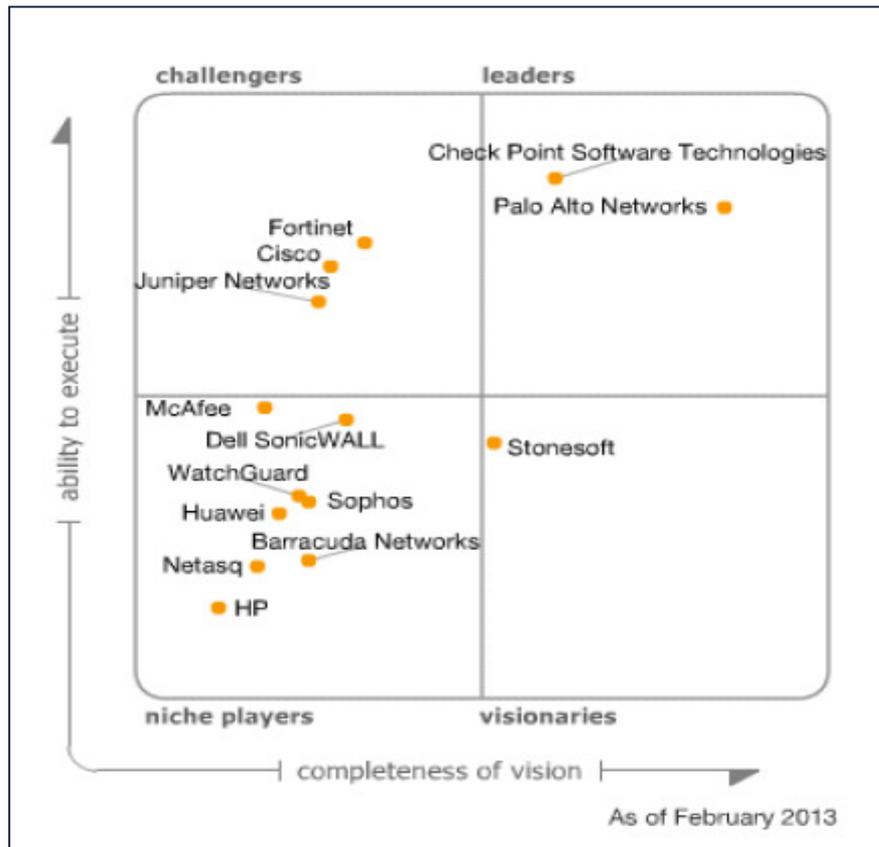


Figure 2 – 13: Gartner Research Magic Quadrant for Enterprise Network Firewalls  
 Source: (Young 2013, 2)

### 2.3.3 Firewalls Limitations

The firewall level of security can be limited and degraded under certain circumstances such as:

- End to End Encryption Channels: firewalls can provide secure network communication by applying encryption; however, this normally leads to an overhead on the firewall’s ability to process requests unless high-speed links are available to the network. A good example of encryption protocols is the Secure Internet Protocol (IPSec) which provides a secure communication channel that can be used to implement a Virtual Private Network (VPN) using the firewall when accessing network resources via an external host from the Internet (Stallings 2006). However, using these encryption channels will prevent the firewall from filtering packet contents as it is a trusted channel of traffic (Keromytis & Prevelakis 2007).

- **Internal Attacks:** The amount of security that can be provided by firewalls is very limited when an attacker initiates an internal attack as this falls within the Local Area Network (LAN) boundaries (Keromytis & Prevelakis 2007).
- **Self Learning:** firewalls alone still lack a self-learning mechanism and cannot learn from past attacks dynamically (Noureldien & Osman 2000). Some firewall implementations based on Stateful firewalls also known as Deep Packet Inspection (DPI) firewalls can detect changes made to the packets status using pre-defined signature and traffic anomalies (Noakes-Fry 2004). However, DPI firewalls react to the anomaly based on the pre-defined signature but do not update these signatures dynamically, therefore, a self-learning process is needed to update the baseline referencing these attacks and anomalies to achieve high levels of firewall protection. Xiaobo et al. (2010) attribute this limitation to the fact that firewalls operate at switching and routing layer (i.e. layer 2 and layer 3) of the Open System Interconnection (OSI) architecture and cannot identify malicious contents of upper layers in OSI (e.g. application layer). However, the self learning intelligence can be achieved by combining firewall and IDS (Xiaobo et al. 2010).
- **Traffic Congestion:** due to the increasing speed of the networks firewalls can become a congestion bottleneck with more possibility of increasing the attack scale causing the network communication performance with the Internet to be degraded or unavailable (Keromytis & Prevelakis 2007).
- **IP Spoofing:** firewall protection becomes useless if an external attacker manages to spoof an IP header with an IP address that the firewall rules will allow to pass through to the network. This case is called false negative, however, the firewall might also produce false positives preventing legal users from accessing the networked system. In both cases this can be too expensive for any business when measuring the cost and value of their security systems (Cavusoglu et al. 2004). For IP spoofing technique to be considered a DoS flooding attack, it needs to generate an enormous number of traffic, however, a small set of packets will be an intrusion by the spoofer

but it might not trigger any alarm unless the packets header contains pre-defined attack signature known by the firewall rules to identify the attack.

- **Wireless Networks:** accessing the network through wireless communications mean that firewall administrators have no physical control over the network access points that can be used by attackers (Keromytis & Prevelakis 2007).

## **2.4 INTRUSION DETECTION AND PREVENTION SYSTEMS**

### **2.4.1 IDS Background**

Firewalls can provide an authorized flow of traffic to inside the network and all traffic to pass to outside the network with a convenient service, direction, user and behaviour control, however, this type of protection is no longer a valid type of protection when it comes to internal or external attacks, worms and viruses (Stallings 2006). Therefore, no complete immunity can be obtained from a firewall because it is a role based system since the attack has to happen first to be identified and configured as a static role in the firewall in order to be used for future filtration and protection. This opens the door for IDS systems trying to overcome the previous firewall static role limitation with unknown attacks and therefore, IDS are commonly used to supplement the firewall in identifying network attacks. Early implementations of IDS came into the equation after worms started to attack networks and flood the networks with packets that stopped any other activity from happening until the network became clean of all these additional packets. This was originally defined as a DoS attack by Stallings (2006).

In DoS attacks, the IDS requires pattern updates to recognise new attacking agents sent by foreign networks it needs to stop in order to be reconfigured (Vigna et al. 2003). By the early 1990's several IDS systems were introduced in an ad-hoc implementation concentrating on host based intrusion detection using the detection rules and based on State Transition Analysis Technique (STAT). However, these roles are purely static and need to be modified manually to be updated and require a special language to be programmed to establish Transmission Control Protocol (TCP) monitoring sessions (Vigna et al. 2003, Verwoerd & Hunt 2002). There was a clear need to shift IDS to the enterprise network level instead of an ad-hoc host based security and also to cover mobile units within the network using lightweight agents

in order for the IDS to perform the required tasks with a minimum amount of coding (Helmer et al. 2003). More recent IDS design proposed a distributed host based IDS mechanism to attach some informational labels (based on taint marking) to system objects (e.g. files, sockets, process,...etc). This IDS approach aim to define how hosts and application can access the information and control the flow of communication between each group of connected hosts (Hauser et al. 2013).

#### **2.4.2 IDS Detection Techniques**

McEachen & Zachary (2007) and Ghosh & Schwartzbard (1999) classify IDS into two classes based on the method used to detect the intrusion as follow:

- Anomaly based IDS systems - where a network base line is compared to normal network behaviour at detection time. This type of IDS produces a high level of false alerts depending on the thresholds used to identify normal network behaviour. However, it can deal with new types of attacks.
- Signature based IDS systems – (also known as misuse IDS) where a pre defined set of signatures and/or patterns of known attacks are searched for in network traffic at detection time. This is a good method for known attacks but can't handle new forms of a malicious attack with an unknown signature and/or pattern.

McEachen & Zachary (2007) extend the above classification of IDS systems to add a new type of IDS called Specification-Based Detection (SBD IDS). These systems work using the anomaly detection method but are different in the way the network normal behaviour is measured or defined.

SBD IDS use a manual set of standards. Estevez-Tapiador et al. (2004) described generic anomaly detection architecture for IDS using three subsystem components: Sensor, Modeling and Detection as shown in Figure 2 – 14. This architecture was mainly proposed for wired systems.

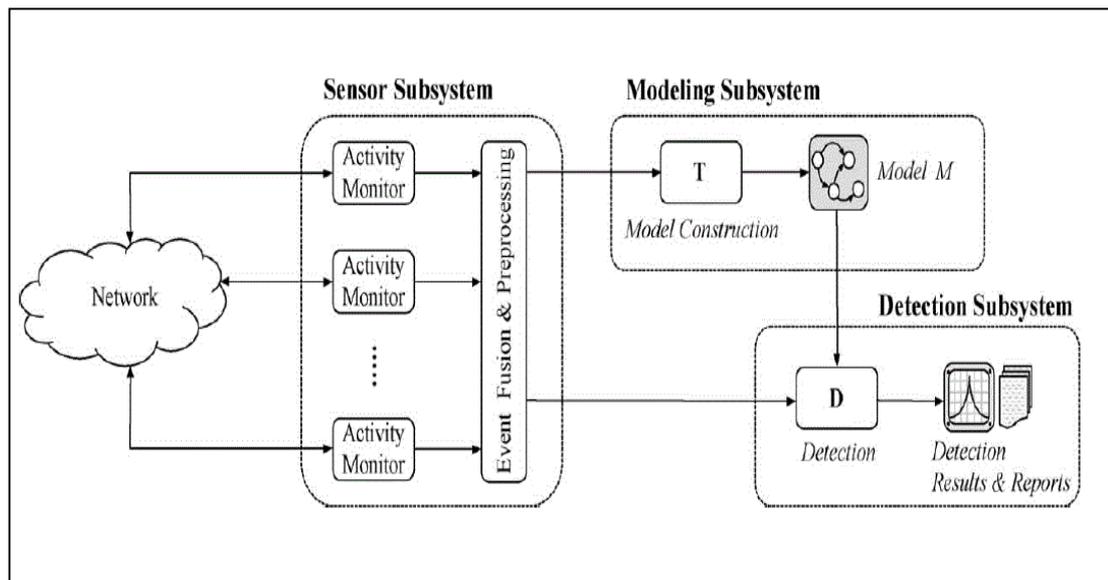


Figure 2 – 14: Anomaly Based Generic IDS Architecture  
 Source: (Estevez-Tapiador et al., 2004, 1571)

The most utilised type of IDS is misuse detection (e.g. Snort) where firewall logs are compared with signatures or patterns of known attacks (Ghosh & Schwartzbard 1999, McEachen & Zachary 2007). Current misuse IDS models search for a matched pattern or signatures of information in network activities that can be analysed and categorised as malicious behaviour by the IDS in order to protect the network from intruders (Vigna et al. 2003). Misuse detection can efficiently detect known attacks if the signatures are known, resulting in a fast match with attack patterns stored in the IDS. Looking for patterns is reasonably straightforward for IDS which compares network traffic with attack data to pick intrusion patterns (Lee & Stolfo 2000). Misuse detection systems commonly exhibit low false positive rates, as an alarm is only raised if the log data possesses an attack signature. Ghosh & Schwartzbard (1999) categorise misuse detection approaches into expert systems, model-based reasoning, state transition analysis and keystroke dynamics monitoring. Although the misuse approach is a static process not able to identify and recognise novel attack mechanisms the majority of research into intrusion detection (and the commercial products available) are misuse based, requiring signature matching. Estevez-Tapiador et al. (2004) tests made on 1999 DARPA/MIT Lincoln Labs evaluations (using top three IDS systems at the time has shown that IDS could detect 80% known attacks, however, it only manage to detect 22% of unknown attacks as shown in Figure 2 – 15 .

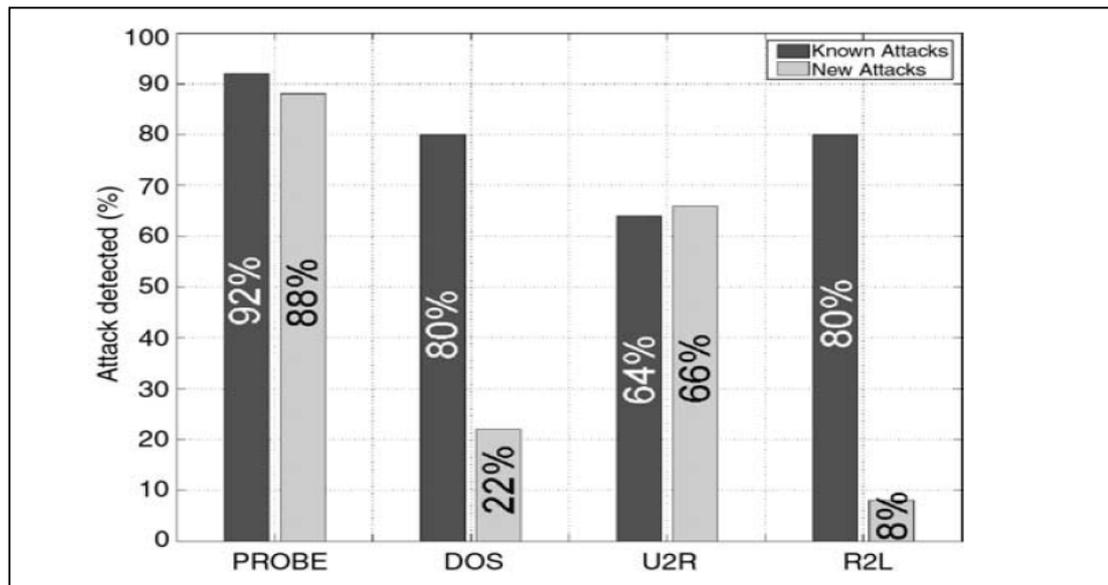


Figure 2 – 15: DARPA 1999 Detection Results for Known vs. New Attacks  
 Source: (Estevez-Tapiador et al., 2004, 1570)

This implies the fact that misuse detection will not be able to detect all new attacks as it will not be able to match it to a pattern or a signature of known attacks. Therefore, anomaly detection method is more powerful in detecting novel or unforeseen attacks in comparison to misuse detection (Estevez-Tapiador et al. 2004).

### 2.4.3 IDS Implementations

IDS implementations are generally classified based on the location of the IDS system, being either Host-based Intrusion Detection System (HIDS) on a single host or a Network-based Intrusion Detection System (NIDS) placed on the network (Allan 2003, Holden 2004). HIDS systems are installed directly into a host machine and audit host system logs with an eye on both internal and external attacks (Durst et al. 1999). HIDS suffer from certain problems and limitations, and McEachen & Zachary (2007) summarise these problems as follow:

- Scalability: HIDS need to be installed on every server for full protection for an organization network.
- Resources: HIDS share a dedicated portion of hosting server resources that can be used to perform business requirements.

- Infrastructure Protection: HIDS can detect attacks toward network infrastructure like switches.
- Forensics: Compromising HIDS by infection also means that the hosting server is compromised; therefore, server logs are disputed from a legal perspective.

On the other hand, the second type NIDS uses a more broad approach of IDS toward networked systems by dedicating a specific network device to act as a network activity monitor and sensor. When a malicious activity is detected it alerts the network and blocks the intruders from doing further damage to the network participating hosts (Allan 2003). NIDS development has been an evolving task ever since James P. Anderson published his famous paper “Computer Security Threats Monitoring and Surveillance” back in 1980.

A major difference between HIDS and NIDS is that NIDS cannot deal with encrypted packets when Virtual Private Networks (VPNs) are used within the network. On the other hand, HIDS can handle encryption but with an overhead of host resource utilisation to inspect host communication packets, application logs and system calls (Goh et al. 2010).

The solution for this problem in the past was to place the NIDS within the VPN gateway, however, Goh et al. (2010) proposed a novel method to resolve this problem with NIDS implementations using Secret-sharing Schemes to deal with encrypted traffic over VPNs without affecting the confidentiality and integrity of the data passing through the VPN.

Figure 2 – 16 illustrates the timeline for NIDS significant developments since 1980 as presented by McEachen & Zachary (2007) and based on previous work by Innella (2001). Also, Figure 2 – 17 shows IDS implementation with firewalls in security infrastructure used by enterprise networks as described by Cavusoglu et al. (2004). This design can contain anomaly detection or pattern matching misuse IDS techniques to detect intruders with the minimum amount of false alerts.

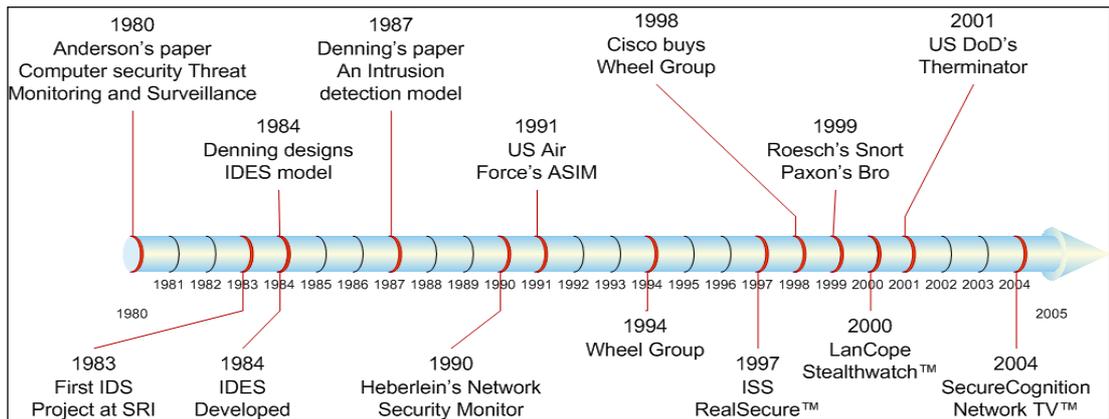


Figure 2 – 16: Timeline of Significant Events in Development of NIDS  
 Source: (McEachen & Zachary, 2007, 88)

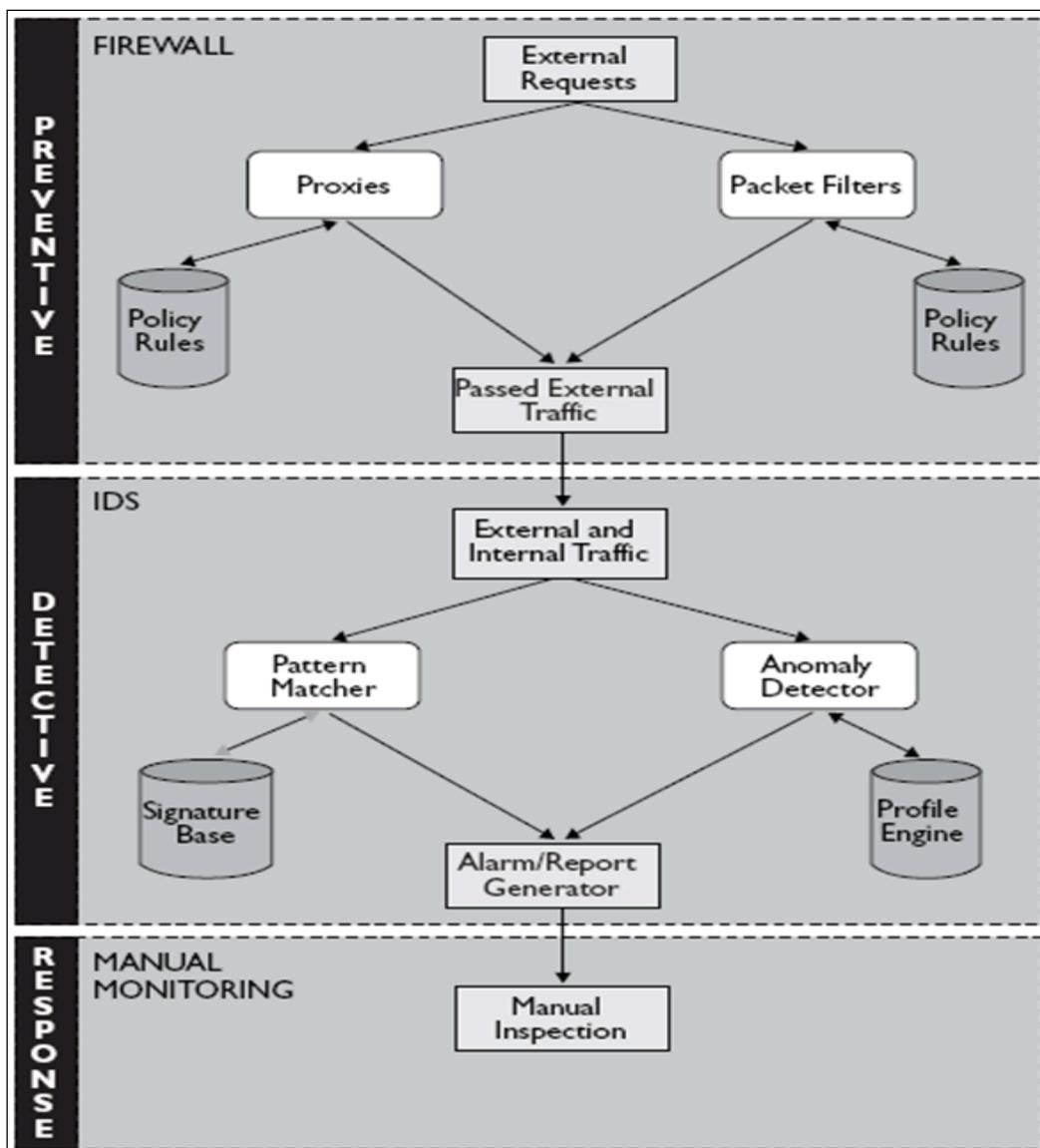


Figure 2 – 17: Firewall with IDS in Information Technology Security Infrastructure  
 Source: (Cavusoglu et al., 2004, 89)

#### **2.4.4 Issues with IDS Implementations**

In spite of enhancements to traditional NIDS implementations to expedite their ability to detect, alert and block intruders, this type of system still suffers from re-configuration problems that make it very hard to generalize and apply to all network configurations. This is due to protocol and configuration language dependencies while re-configuring classification methods in different NOS software and hardware environments (e.g. Cisco, UNIX and Windows) and the bandwidth cost of implementing IDS systems (Iheagwara & Blyth 2002, Ollmann 2003, Rozenshine-Kemelmakher et al. 2010). In wired networks, some IDS implementations went down to the hardware monitoring level of every Network Interface Card (NIC) to do packet classification on all traffic passing through the firewall interfaces in order to avoid bypassing an attack at the firewall level. This means the firewall cannot pass any packet to the network without being scanned first against all known attacks (Otey et al. 2003, Singh & Silakari. 2009). Also, with the introduction of wireless networks, IDS has more challenges to protect the network from internal intruders as wireless networks are more vulnerable to hackers and intruders attacks than wired networks (Zhang et al. 2003, Pelechrinis et al. 2011).

The main disadvantage of misuse detection IDS is that they cannot detect novel attacks that leave unknown signatures (Ghosh & Schwartzbard 1999, Wu & Banzhaf 2010). This means that a new type of DoS attack will go undetected by the misuse detection IDS until its signature is known and recorded. There is, therefore, a gap between the time the new attack appears and the time it is recognised as an attack by the coding of its signature in the IDS. This is one of the main limitations in the ability of misuse detection IDS to identify new attacking techniques without being attacked first. Therefore, the faster the IDS can detect the attack the faster an intrusion can be stopped before causing more damage to the network (Chang 2002, Wu & Banzhaf 2010). A common problem with IDS is the rate of inaccuracy and the high number of false positive alerts leading to management overheads without a valid source of an attack toward the network, however, this is due to the IDS using multiple algorithms when classifying an attack (Stiennon & Easley 2002, Cheng 2012).

Finally, the expected value of networked IDS implementation from a business perspective can be affected by many factors as described by Cardoso (2007):

- *Overreliance on firewall implementations:* Networks rely on firewalls as the first line of defence, however, firewalls alone cannot protect against DoS and DDoS attacks been the target at the same time.
- *Inaccuracy with overwhelming false alerts:* High levels of false alerts generated by IDS implementation reduce confidence in the level of protection. In addition, it requires manual intervention by system administrator to filter these false alerts.
- *Low level of management compare with high cost of maintenance:* Poorly implemented IDS consume more time to configure and manage increasing maintenance cost.
- *Perceived need to outsource:* Adding IDS solution to the network had been perceived as a very complex solution that required additional skills to manage and drive successfully. Therefore, the perception is that outsourcing security management is needed to obtain the maximum value of IDS implementation.
- *Attack detection but no prevention:* Currently implemented IDS solutions can only protect against attacks but cannot prevent these attacks from occurring specially DoS and DDoS. Therefore, the solution is to combine a firewall with IDS to prevent the system and that how IPS solutions are marketed.

#### **2.4.5 Improving IDS Implementation Using Self-Learning Approach**

The current trends of defending against attacks in the past few years tend to be more dependent on learning from the past data of intrusion scenarios. A framework that uses Data Mining techniques to automate IDS systems called Mining Audit Data for Automated Models for Intrusion Detection (MADAM ID) was developed to shift IDS from static classification based on patterns and association roles to dynamically construct new classifier in case of new attacks (Lee & Stolfo 2000).

As a result of the MADAM ID framework, the DARPA evaluation program implementation of MADAM ID framework has been developed and tested, showing an equivalent efficiency compared to manually configured IDS models. The advantage here is that MADAM ID can construct the classifier pattern or role in real time while detecting the intrusion but a static IDS will need to be taken out of service to be reconfigured. This allows attackers to gain access to the network as no active IDS is available. DARPA shows that dynamic learning from past data can be helpful and more efficient compared to static as it saves re-configuration time (Lee & Stolfo 2000).

IDS can also make use of decision roles to decide, and Neural Networks (NN) and data mining theories have been used to enhance the performance of IDS systems against network attacks by providing an intelligent mechanism to learn more from past audit data collected from IDS (Lee & Heinbuch 2001). Although extensive research has been carried out in misuse detection, research into pre-emptive approaches is less prolific. Categories of anomaly based research identified by Ghosh and Schwartzbard (1999) include rules for normal behaviour, statistical models of user or program profiles, and machine learning to recognise anomalous user or program behaviour.

More recent research includes areas such as intelligent firewalls with egress security (Xiaobo et al. 2010), specifically designed architectures (Bolzoni et al. 2006, Zanero 2008), web application profiles and protocols (Estevez-Tapiador et al. 2005, Kruegel & Vigna 2003, Robertson et al. 2006), data mining (Sequeira & Zaki 2002), neural networks (Mukkamala et al. 2002, Ordonez-Cardenas & Romero-Troncoso 2008, Zhang et al. 2005) and privileges flows and system calls (Cho & Park 2003, Kruegel et al. 2003).

#### **2.4.6 Intrusion Prevention Systems**

The success of the MADAM ID framework can be extended to the advantage of creating IPS as an advanced and intelligent IDS implementation. IPS can be considered to be a mix of an IDS and network routing functions of any firewall router that controls packet trafficking across the network providing a mechanism for prevention of an attack rather than having an attack to be detected (Ollmann 2003), it

needs to shift in a new direction to work as a single self intelligent component for new emerging networks.

The IDS has recently been considered more of defensive mechanism and therefore, the focus has been shifted to IPS systems that can integrate within a firewall implementation and can also exist independently; however, it is preferable to have the IPS as part of the firewall for enterprise networks implementations. Figure 2 – 18 illustrates how IPS works within a layered network environment.

Emerging IPS systems combined the firewall with all the aspects of previous generations of NIDS systems and can effectively prevent intrusion attacks based on understanding the intruder attacking technique using expert analysis (Cardoso 2007). Unfortunately, IPS still can't predict the DoS and DDoS flooding attack until the attack has actually started.

Cardoso (2007) set some architectural requirements for IPS systems in order to provide a reliable level of service, these are: high availability, high performance, manageability and scalability.

Pescatore & Stiennon (2003) describe the need for successful IPS network protection from a practical implementation point of view. They suggest that IPS should be able to overcome the decision problem of which algorithm to use and which activities to classify as an attack by:

- Using multiple logical algorithms rather than classification patterns or roles used by an IDS in a real time mode.
- Distinguishing between attacks and normal network activities (self and non-self activities) done by legitimate applications and users of the network.

Additional IPS success factors might be needed such as high network throughput and the ability to control and stop malicious traffic activities acting at a secure sockets layer level in the network (Stiennon 2003).

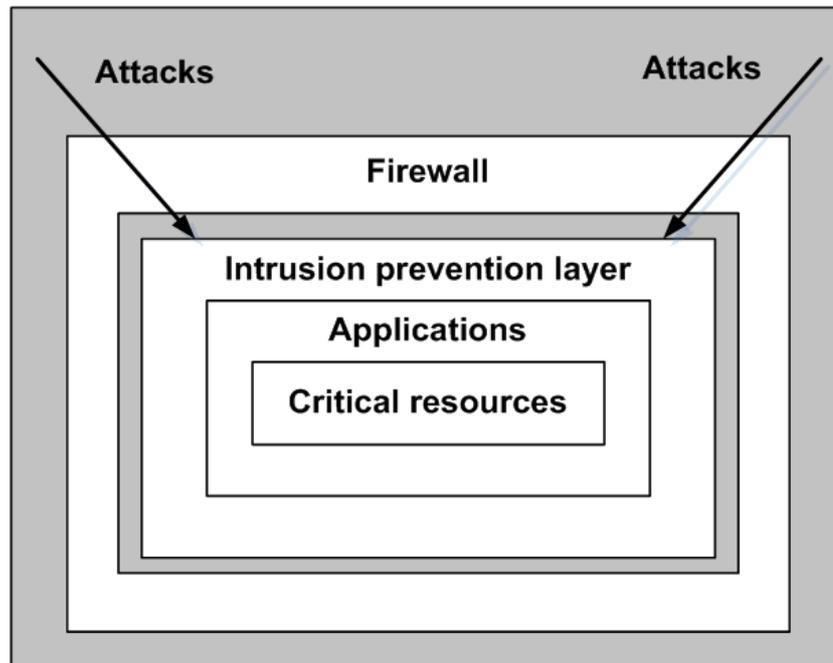


Figure 2 – 18: Placement of Intrusion Prevention Layer

Source: (Cardoso, 2007, 111)

The most important part of selecting any IPS system is why the network needs an IPS, the answer to such a question leads to more specific selection criteria based on security coverage level, performance, management, and deployment platform (Snyder 2006).

The selection criteria can also be affected by the amount of resources to be invested in a complex IPS system in comparison to a cheap system since IPS are generally very expensive and require adequate level of training to manage (Cardoso 2007).

## 2.5 RELEATED WORK

Over the past decade, numerous researchers have worked on many detection models and approaches to mitigate the DoS / DDoS security problems. This was done either by using misuse or anomaly based detection models.

However, this research is more interested in the use of an anomaly detection model within the firewall borders for both incoming and outgoing traffic patterns. Therefore, the following section summarises related research work in this area of attack detection from past to recent years.

### 2.5.1 Anomaly Based Related Research

The following is a list of previous work related to anomaly detection over the past years track the usage of multiple approaches to resolve the DoS / DDoS problem and follows Table 2 – 5 classifications in a chronological order.

[1] *Ghosh and Schwartzbard (1999)*: used neural networks experimental approach to enhance the detection process and reduce false alerts, however, no complete results of the experiment had been provided to compare different neural network models for specific DoS / DDoS flooding attacks. The researchers used DARPA data set to evaluate research findings for both anomaly detection and misuse techniques. This researchers contributed to the fact that neural network can be used for both type of detection type (i.e. misuse and anomaly detection). The researchers also observed two main issues with DoS / DDoS detection, these are the detection capability and false positive ratio to judge the IDS performance using an experimental approach. Overall, the researchers show the viability of using neural network approach to resolve the DoS / DDoS problem.

[2] *Michael & Ghosh (2000)* described an approach for IDS using two-state anomaly detection applied to data from Sun Solaris Basic Security Mode (BSM) auditing system. The researchers describe how to create an automatic “finite automata” function that can analyse the data and change of state from one event to another within the BSM data set. The researchers use a comparison approach with other methods of detection (i.e. n-gram) to conclude that the detection algorithm performance can be affected by the quantity of the trained data. The researchers presented two algorithms to solve the IDS problem using program behaviour traces. The researchers raised the possibility of enhancing the results obtained from a finite state machine using better learning algorithms suggesting more learning can provide better modelling with long term capabilities, however, no complete experimental results provided at the time of the study to support this assertion.

[3] *Lee and Heinbuch (2001)* used experimental setup to train hierarchy neural network as an anomaly detector of protocol and services activities by representing every activity with a neural network in order to classify the

anomalies within these activities. Having multiple neural networks at every protocol level can produce an accurate detection levels, however, it can also be time consuming especially if the system is under a flooding attack. The research focuses on the TCP protocol well defined behaviour to be monitored by the IDS experiment. This research contributed to the data type that should be monitored at a host based IDS (e.g. TCP protocol activities) in order to detect attacks. The test data was used in a simulation environment to detect and classify anomalies within the data after training the neural network. Therefore, this research shows that monitoring network protocol activities can be used to detect anomalies within the network traffic.

[4] *Ye and Chen (2001)* used multivariate statistical techniques based on chi-square analysis for DoS detection by event auditing of system process achieving 0% of false alerts on normal events but up to 75% with attack events. The researchers suggested an aggregate level behaviour to achieve a more accurate detection using a decision threshold to distinguish between normal sessions and intrusion sessions. The researchers also highlight the need to detect the intrusion as early as possible and while it is still in progress using three detection levels: normal, alert and alarm levels. Therefore, this research study helped in staging the detection process of multiple sessions using statistical decision threshold. However, the testing data set was very small and it can be argued that the results can be applied to a large-scale network.

[5] *Mukkamala et al. (2002)* suggested the use of both neural networks and Support Vector Machines (SVM) for IDS systems to detect intrusions (including DoS). The model produced shorter data training time, however, it was only implemented on DARPA data set and needs further testing done on real life data. The researchers show that SVM can produce a slightly higher rate of accurate detection compared to neural network detection. It has also been observed that the performance by both approaches is very high; however, the training rate of SVM was much less than neural network which is an advantage of SVM over neural networks. The research study identified the disadvantage for using SVM whereby it can only do binary classification while IDS required multiple classifications to identify different types of attacks.

[6] *Sequeira & Zaki (2002)* introduced ADMIT (Anomaly based Data Mining for Intrusions) that searches within shell commands history to detect anomalies of intrusion. This is a host monitoring system that monitors commands activities only. It also suffers from the large size of training data that can slow down the processing of intrusions. The researchers achieved 80% success rate of identification but it also produces 15% of false positive. Future works suggested to reduce the training for ADMIT and different parameters in order to improve detection performance.

[7] *Cho & Park (2003)* proposed an IDS model based on anomaly detection of behaviour changes on system calls using Markov modelling. Sun Solaris Basic Security Mode (BSM) auditing was used to provide the source data, results shows that a model based on privileged changes of transition state flows can produces less errors without losing its detection capabilities while system is under intrusion attack. The proposed detection system used Solaris operating systems logs to identify normal behaviour and abnormal behaviour of users. The collected activities included but not limited to programming activities, web navigation and FTP data transfers for one month for 10 users. The researchers shows that training detection model for privileged user activities is faster than conventional data training and has less detection errors. The researchers also suggested that identifying privileged activities as an anomaly detection indicator can be applied in real life networks to support behavioural security models.

[8] *Feinstein et al. (2003)* used chi-square analysis for DoS detection and applied filtering rules upon detecting an attack. This approach used automatic baseline and threshold calculation to identify an attack. However, no triangulation with another detection method had been used to measure the accuracy of the detection. The researchers used multiple publicly available network traces data to test the detection approach on multiple environments for efficiency purpose on high speed routers with low computational resources with Snort and the chi-square detector as a Snort plug-in. The chi-square detectors could identify DDoS attacks and apply packet filtering and classification rules. Future work suggested by this research to enhance the detectors by applying a tight integration of detection and response rules.

- [9] *Kruegel & Vigna (2003)* used statistical Markov model to detect anomalies in web based attacks by the means of HTTP queries to web sites data. The validation process concluded that web attacks should be addressed with signature based detection algorithm, however, the detection process should also use anomaly detection to optimise the detection process. The data was collected from Google and couple of universities in USA and Europe. The only issue with this research study is that it relay on the HTTP server access logs, if the attack targets these logs it will be impossible to identify an attack because these logs could be altered or removed and therefore, the attack cannot be identified. However, the researchers shows that web server logs can be used to identify attacks and that support the data mining approach for detecting anomalies for systems under attack.
- [10] *Kruegel et al. (2003)* investigated the anomalies with system call arguments instead of signature based algorithms using Markov model to detect anomalies within applications (such as WUFTP servers) to monitor malicious behaviour of these applications as an application level IDS system. The researchers developed a host based IDS to identify malicious behaviour. The researchers highlighted the importance of adding system calls arguments to the anomaly detection methods and not to rely only on the sequence of system call invocations as the only measure of detection.
- [11] *Estevez-Tapiador et al. (2004)* presented the criteria whereby anomaly detection can be used to detect network-based attacks. It identified the possibility of using anomaly-based detection but with some challenges like cryptographic issues which is also a problem for misuse detection IDS systems. More research was proposed to over come the limitations in that field of detection. The researchers suggested improving the anomaly detection by scaling the analysis of protocol fields, apply multiple and hybrid detection methods and the use specific anomaly detection. Overall the researchers emphasis the importance of further work to improve anomaly detection techniques and present it as a hard problem that still need more knowledge to be resolved.

- [12] *Li (2004)* introduced a statistical detection algorithm for IDS to detect new signs of DoS attacks and calculate the threshold. The researcher identified that previous research on statistical modelling in the field of anomaly detection has some issues in the statistical features of the traffic used to recognise the attack pattern, the probability of raising false alarm or missing on real alarms and how to reduce false alarms and produce accurate identification of the attack. The researcher uses a statistical detection schema centred on the probability of attack identification, false alarms, miss and decision making factors. However, the detection algorithm template lacked the ability to adapt for new devices added to the network environment such as the internet.
- [13] *Takei et al. (2004)* used pattern matching and data mining techniques to detect illegal access attacks on the internet transmitted over TCP / IP packets and represented as a time-series. This approach produced good results of less than 1ms of detection for 25 interconnected networks; however, this can also become a problem with large interconnected networks since the calculation time can be up to the square value of the connected networks. The researchers used a mix of data sets to evaluate the proposed detection method and validate the detection parameters result against a known data set to trace illegal access. The researchers study shows that illegal access patterns in the internet can be detected by adjusting the detection threshold to detect SYN flood and Smurf attacks.
- [14] *Estevez-Tapiador et al. (2005)* used Markovian protocol parsing to detect web servers attack by monitoring HTTP protocol achieving high detection capabilities with low false positives. However, this researchers was only limited to a specific part of the network (web servers) and dealing with a single protocol (HTTP). Therefore, this study can be improved by including more traffic protocols to be monitored to enhance detection capabilities. The work also doesn't address the re-training process, threshold tuning and additional protocols used by users such as DNS and SMTP activities.
- [15] *Lee et al. (2005)* used IP trace back against RDoS with NS-2 simulator to detect spoofed IP's. This method uses a classification technique to identify attacking packets and then uses packets path marking by modifying the time to

live (TTL) value to trace back RDoS. The proposed method used the fact that malicious ICMP messages are normally dropped at the router level causing the DDoS attack to become a passive attack. However, trace back methods cause the routers to send some data to the RDoS attacker. The researchers study in that field had shown that it is possible to defend against RDoS attacks using a trace back function.

[16] *Mitrokotsa and Douligieris (2005)* used Kohnan's Self-organising Maps (KSOM) neural network model to classify network traffic attacks by using preselected features of attacks such as duration, source, destination, count and other features as identified by Mukkamala et al. (2002). This approach is powerful and produces accurate measures. However, the main disadvantage of the KSOM model is that the required high training computational overhead when the neural network training records exceeded the 10000 records boundaries. Therefore, to re-train the network on a real time scenario when ever new data arrive can be time consuming and will reduce this model's efficiency. The researchers study supported the use of neural network as a DoS detection approach and proposed it to solve other intrusion detection problems.

[17] *Zhang et al. (2005)* used a Multi Layer Perceptron (MLP) learning algorithm and a Radial Basis Functions (RBF) to classify network attacks showing that neural network tools can be used with both known and unknown attacks. The researchers study illustrated the ability to use more than one neural network to classify and detect unknown anomalies with RBF producing better results than MLP when trained as classifier methods to detect known attacks and novel intrusions. However, the training process of RBF can be very complex in comparison to MLP due to the simplicity of the MLP training process which gives it an advantage over RBF.

[18] *Bolzoni et al. (2006)* used a 2-tier anomaly detector that includes a Self-organising Maps (SOM) neural network as the first tier and clustering packet payload data analysis algorithm (PAYL). The model was tested with 1999 DARPA test data. The results shows the data profiles reduced to achieve accurate results from PAYL when using neural network SOM. This highlights the

efficiency that can be added using neural networks in a packet classification scenario by reducing the levels of false positive at a higher detection rate. The researchers use of more than one anomaly detection technique highlighted the importance of triangulating between neural networks and statistical methods to resolve the IDS anomaly detection problem.

[19] *Carl et al. (2006a and 2006b)* compared the use of activity profiling using backscatter analysis, sequential change of point detection and wavelet analysis. The results vary depending on the test conditions applied and attack rate dynamics. This work highlighted the level of complexity within the three approaches showing change point detection with chi-square analysis as the lowest complex approach while activity profiling is the most complex approach. The researchers uses NS-2 simulator with 100 nodes three flooding attacks over TCP, UDP, ICMP and SYN protocols reaching up to 20% of the total traffic. All three attacks were detected with false alert ranging from 1 to 6 alerts for every 100 packets with generated background traffic of 75%. A superimposed DoS attack was applied to a real world traffic showed that real world data can provide better confidence in output test results compared to NS-2 simulation. Therefore, the researchers suggested using realistic network traffic for accurate DoS detection.

[20] *Chen & Hwang (2006)* proposed the use of aggregate detection trees on routers for early detection of traffic pattern anomalies. The simulation experiment was tested with TCP, UDP and ICMP flooding attacks with three evaluation metrics: detection rate, alarm rate and receiver operating characteristics using NS-2 simulation. The model claiming to have accurate results with low false alerts in simulated environment. However, if multiple routers got compromised the construction of the proposed aggregate tree pattern can be affected and the wrong pattern will be constructed which mean that the detection process will eventually fail.

[21] *Globa et al. (2006)* completed an analysis of using neural networks for anomaly detection using experimental traffic to train the network and then compare with new traffic and re-train the neural network. The issue was the

analytical proposed model was using an attack signature which means that the proposed model cannot handle unknown attack signatures to find the anomalies. However, the researchers also confirm the ability of neural network to detect DDoS attacks.

[22] *Robertson et al. (2006)* presented an architecture model for web intrusion detection system using anomaly signature generation and attack characterisation class. The proposed web intrusion detection system architecture collect access logs from the web servers and examine these logs in a learning phase to create an access profile. When the system changes to detection mode it detects unmatched patterns and generates an anomaly signature if the detection threshold had been exceeded. The architecture has been evaluated on two universities data sets based on false positive alerts to generalise attack patterns (without further statistical analysis to reduce false positive alerts) showing good performance results. The researchers also suggested that the proposed web access intrusion detector to be generalised to other intrusion detection fields like system calls arguments anomalies.

[23] *Schadwinkel and Dilger (2006)* proposed an anomaly detector based on an artificial immune system using neural networks. The Immune System on Self-Organising Map system (IS-SOM) treats internal patterns as antibodies and environmental patterns as antigens in the same way as a human immune system (HIS). The system trains itself on internal patterns to counter external environment patterns as foreign bodies to its internal anomaly-free patterns. The researchers presented the HIS based approach of artificial immune system toward solving pattern anomalies detection problem using SOM neural network and suggested further research in that field is still needed. However, the researchers study seems to be theoretical with no experimental results provided.

[24] *Siris and Papagalou (2006)* presented a comparison between two anomaly detection algorithms to detect SYN TCP flooding using statistical analysis and investigated the detection probability performance in these algorithms and the trade of between attack detection and false alerts. The researchers used adaptive threshold algorithms for detecting DoS attacks considering the tradeoffs between detection capabilities and false alarm ratios. These methods used change point

detections like CISUM to detect anomalies during the DoS attack. The aim of the researchers was to simplify the detection process while keeping a good performance. Future work discussed the enhancement of the DoS detection methods by packet marking specially in the case attacks that apply of IP spoofing techniques. The researchers study had shown the importance of threshold management to obtain accurate performance of DoS anomaly detectors.

[25] *Znati et al. (2006)* used data sampling and packet inspection models to isolate normal network behaviour from suspicious activities in a distributed defence scheme in order to distribute the load on the detection process while filtering and inspecting the traffic packets. The accuracy of obtained results relied heavily on the throughput and buffer sizes of the network firewall bandwidth to drop malicious packets. The researchers used simulation to experiment to investigate a distributed global adaptive DDoS defence infrastructure using NS-2 simulator. The experiment proved that this type of defence scheme is providing better defence than existed schemes by distributing the workload between when packet filtering methodology applied. The proposed schemes tested both routes and firewalls performance while filtering the packets upon detecting an attack. The researchers study contributed to the fact that firewalls and routers can be enhanced for better detection using packet sampling to mitigate DDoS attacks.

[26] *Alsaleh and Alfantookh (2007)* highlighted the efficiency of using the neural network feed forward algorithm as a significant tool to detect unknown attacks by introducing the gray area improvements. The researchers try to solve the problem of new DoS attacks detection using training methods that initialize the network weights to either zero values, training initials values or random values. The measurement unit used to measure the detection performance is mean square error (MSE). However, the research study did not define the amount of history that the neural network can use as minimum value to train the network in order to produce accurate results. In addition the data used DARPA data set known attacks to train the neural network with output results that shows a reduction of false alerts 1.42% under normal circumstances showing the significance of using neural nets as detection algorithm.

[27] *Onut and Ghorbani (2007)* introduced SVision a visual network anomaly detector using protocol inspection of network traffic to visualize anomalies within a community of independent roaming hosts. The proposed technique uses the DARPA database with multiple sets of pre-set attacks including UDP, ICMP, SSH, Mailbomb, DNS and Nachi worm. The researchers used TCP dumps tool data logs to identify historical traffic anomalies and graphically visualise flooding attacks. The proposed graphical algorithm combined with an anomaly detection algorithm aimed to detect the attack based on the service provided by different protocols. The main issue that this detection system has is it is currently passive rather than interactive with speed and accuracy under questioning for high speed networks. However, this study highlighted the importance of data mining within TCP / IP suite of protocols traffic activities and anomalies that can contribute to detecting DoS attacks.

[28] *Lee et al. (2008)* suggested the use of cluster analysis using entropy theory. This produced very good results. However, the results obtained were limited to certain parameters such as source and destination IP addresses and port numbers with every packet type occurrence rate and number of packets. The research paid no attention to the status of the packet from a historical point of view (i.e. how many packets of this type in this direction of traffic had been accepted or rejected). In addition the model was only tested on a pre-provided data set from DARPA designed for specific attacking scenarios as described by DARPA. The researchers contributed to the fact that traffic activities can be used to detect abnormal activities within the network which indicates occurrences of DoS / DDoS attacks. In addition this research study proposed more future work is needed to investigate network traffic activities and extract more variables to develop an advanced detection techniques based on these variables.

[29] *Malliga et al. (2008)* proposed a model to use routers to monitor source IP addresses and calculate a traffic trends per source IP at certain time intervals. It then compare the normal trend with new data to calculate this source IP threshold before declaring any traffic from this source as an attack and reduce the traffic from this source IP. However, if the attacker managed to spoof a legitimate IP address or a legitimate user manages to create multiple simultaneous connections

it can confuse this model. Therefore the alerting mechanism is not accurate from both false negative and false positive points of view. The researchers identified that more work is needed to include other parameters to the selection process such as ICMP traffic and high packets fragmentation in order to improve the detection methods since the research study used a single source network with emphasis on spoofed attacks only.

[30] *Xu and Zhan (2008)* suggested the use of an adaptive system state transition model. The model uses linear functions to estimate the mean and standard deviation from normal behaviour by self-learning approach. However, the model is so general and does not cover every scenario especially the DoS / DDoS flooding attacks. In addition it does not cover how the selected function parameters can be selected when dealing with specific network under attack (e.g. constant or variables). The researchers shows that transition status changes which can dynamically adapt to changes in the network can be used to improve IDS without the need to be re-trained in order to improve detection performance. This research study had contributed to the need of dynamically adaptive techniques when dealing with attacks in a dynamic network environment.

[31] *Yu et al. (2008)* discussed traffic flooding detection using support vector machine implementation (SVM) with SNMP management information base (MIB) traffic records data mining as an anomaly detection model. This model's performance and accuracy depends on the selected SNMP MIB variables to be collected in order to obtain accurate measurements for attack detections during the variables collection time. The research paper proposed building a real time prototype for future work using this approach. The researchers had shown that data mining approach of traffic protocols flooding over TCP, UDP and ICMP in SNMP MIBs can be used for detecting DoS / DDoS attacks as anomalies within the data.

[32] *Zanero (2008)* introduced ULISSE (a network IDS system) that uses a model similar to Bolzoni et al. (2006) that reduces false positives further. However, the model speed needs improvements. The ULISSE was integrated it with other host based systems to filter and further reduce false positive alerts. The researcher

developed an anomaly detector based on unsupervised learning techniques of self organised maps (SOM) to reduce false positive detections. The researcher study highlighted the need for future work to improve the speed of the proposed detection system.

[33] *Oshima et al. (2009)* highlighted the increasing values of chi-square statistical analysis results for a system under DDoS attack using IP source and destination addresses. The research used an experimental network consisting of web server, DMZ network and a firewall. Using the web server as an attack access point, the system was configured to reject any packet calls outside port 80 toward the testing experiment web server. The researchers identified that the amount of TCP / IP packets and the activity of these packets toward the web server can be used as detection parameters to build a DoS / DDoS detection scheme with statistical chi-square analysis. However, further work is still needed to cover other characteristics of these packets such as packets number of bytes and the planning of packets receiving durations.

[34] *Viinikka et al. (2009)* presented an approach to analyse and filter alerts within network alert flows from SNMP, SMP, ICMP and other network packet traffic patterns. This approach uses time series statistical modelling to filter abnormal level of alerts as an anomaly within the residual of the time series compared to normal system behaviour. However, the approach also uses Kalman filtering which has been identified as a risk to over smooth the alerts data. In addition, the applied detection algorithm used in this research study can miss an anomaly if it was preceded by another anomaly. The researchers approach had shown that more work is needed to handle the flow of large quantity of alerts to detect IDS alerting anomalies. Therefore, future work was suggested by the researchers to use time-frequency to further enhance the detection algorithm of online alerts processing.

[35] *Hashim et al. (2010)* proposed a security framework based on anomaly detection that follows the human immune system (HIS) and danger theory (DT) to mitigate DoS / DDoS attacks. The framework targets malicious anomalies in networks and recovery processes inspired by human diseases and pandemics in real life. The main limitation of this model is that it leaves the responsibility of

determining the detection thresholds to the network administrator and that can be out of date for any new attacks. The researchers study using heterogeneous network supported the use of HIS lymphatic laws as a valid approach to mitigate the DoS / DDoS anomaly detection problem and therefore, reduces the impact of malicious attacks.

[36] *KrishnaKumar et al. (2010)* introduced an algorithm called Hop Count for packet filtering to counter DDoS on routers using the time to live (TTL) hop counts for TCP / IP packets travelling through the routers. The researchers proposed four modules to mitigate DoS / DDoS attacks, these are: packet marking, packet analysing, attack path construction and packet filtering. The proposed algorithm requires encoding the data within the packets using a hash function and a secret key code. Therefore, all participating routers in this proposed algorithm should know how to authenticate in order to communicate and filter the packets. The main issue with that algorithm is mainly to do with encryption overhead and security if the key is leaked. Overall the proposed algorithm highlights the useability of distributed methods to tackle the DoS / DDoS problem at the packet level of within the attacked network.

[37] *Li et al. (2010)* presented a DDoS detection based on neural networks. The proposed approach uses Learning Vector Quantization (LVQ) artificial neural network algorithm and compares with Back Propagation (BP) neural networks. The study has shown an improved detection level with lower false alerts using neural networks. The researchers observed the benefits of using multiple neural network modeling to detect attack patterns. However, the assessment criteria was designed for host anomaly detection and widely moved beyond network packets activities to include more parameters like host processor usage, memory consumption and system time. The triangulation between two types of neural networks had shown a detection improvement ranging from 89.9% with PB compared to 99.7% with LVQ. Future work suggested the use of other neural networks to enhance the hybrid detection architecture of DoS / DDoS host based detection method.

[38] *Lo et al. (2010)* proposed a cooperative IDS system for cloud computing designed to detect DoS and minimise its effect. The proposed system consists of

multiple IDS agents as a distributed mix of IDS systems whereby the IDS can either use misuse or anomaly detection. The simulation made by the researchers used Snort with three additional modules: block, communication and corporation modules. However, this added some computational overhead to the Snort IDS in comparison to a pure Snort implementation. The idea is based on multi staging attack prevention in which one IDS can be affected while the rest can make a judgment and mitigate attack risk. The main issues with this approach are false alerting and if the communication between the agents is lost the overall cloud security will be at risk. The research study had shown that the problem of DoS / DDoS is still an important issue for new emerging technologies like cloud computing and changing the way how computer networks work doesn't change the damaging effect of these attacks.

[39] *Tzur-David et al. (2010)* presented a Statistical Packet Acceptance Defense Engine (SPADE) detection engine design to prevent from attacking threats of DoS and DDoS with the aim targeting threats prevention, accuracy and scalability. The proposed design uses IP packets and bandwidth resource measurements to measure the attack impact on victim server and / or network resource usage. The research design uses random attacks of ICMP flood, SYN flooding and DNS floods to test SPADE performance. However, the design did not target attacks that consume large resources and can slow down the attacked server. The SPADE performance measurement was based on the algorithm scalability and accuracy. The researchers study had shown that host / network IP destination, the protocol type and protocol specific information can be used to identify the DoS / DDoS attacks with negligible number of false positives. This implies the fact that expanding the research on protocol types and activities can lead to better detection results.

[40] *Xiaobo et al. (2010)* proposed an intelligent firewall by combining firewalls with egress security that include IDS engine, identity authentication, context auditing filter and access control policy. The implemented prototype model uses Artificial Neural Network (ANN) in the form of a Back Propagation (BP) neural network to determine the intrusion as an anomaly. In addition it uses Snort to implement rule based detection for misuse intrusion. The prototype model

showed the effectiveness of merging all these technologies for more security. However, the issue with this proposed model is resource overload of egress security on top of the detection engines combined with the firewall overall workload performance which was not clearly explained within the research.

[41] *Vijayasathy et al. (2011)* proposed a Naive Bayesian Classifier model to detect DDoS in TCP networks traffic patterns. The model uses training data to classify traffic patterns as normal if it is below a particular threshold probability level using cross validation technique. The issue with this proposed model was the packets selection as it was only limited to TCP and UDP, therefore, it ignores anomalies whereby ICMP, IP and other packets can be used to flood the network during a DDoS attack. The researchers identify the necessity to improve the proposed classification model to catch all attacking traffic.

### 2.5.2 Anomaly Detection Top Used Techniques

Using Verwoerd & Hunt (2002) classification shown in Table 2 – 5, a summary of past anomaly based detection techniques used by other researchers in this field over the past decade can be presented as shown in Table 2 – 6 and Figure 2 – 19 (both table and figure are based on the previously listed related work mentioned earlier in section 2.5.1):

Anomaly Based Technique							
Year	Statistical Modelling	Immune System	Protocol Verification	File Check	Taint Check	Neural Network	White Listing
1999						[1]	
2000	[2]						
2001	[4]					[3]	
2002					[6]	[5]	
2003	[7],[8],[9],[10]						
2004	[12]		[13]				
2005	[14]		[15]			[16],[17]	
2006	[19],[24]	[23]	[20],[25]			[18],[21],[23]	
2007			[27]			[26]	
2008	[28],[30],[31]		[29]			[32]	
2009	[33]						[34]
2010	[39]	[35]	[36],[38]			[37],[40]	
2011	[41]						

Table 2 – 6: Sample of Anomaly Based Research Techniques (1999 – 2011)

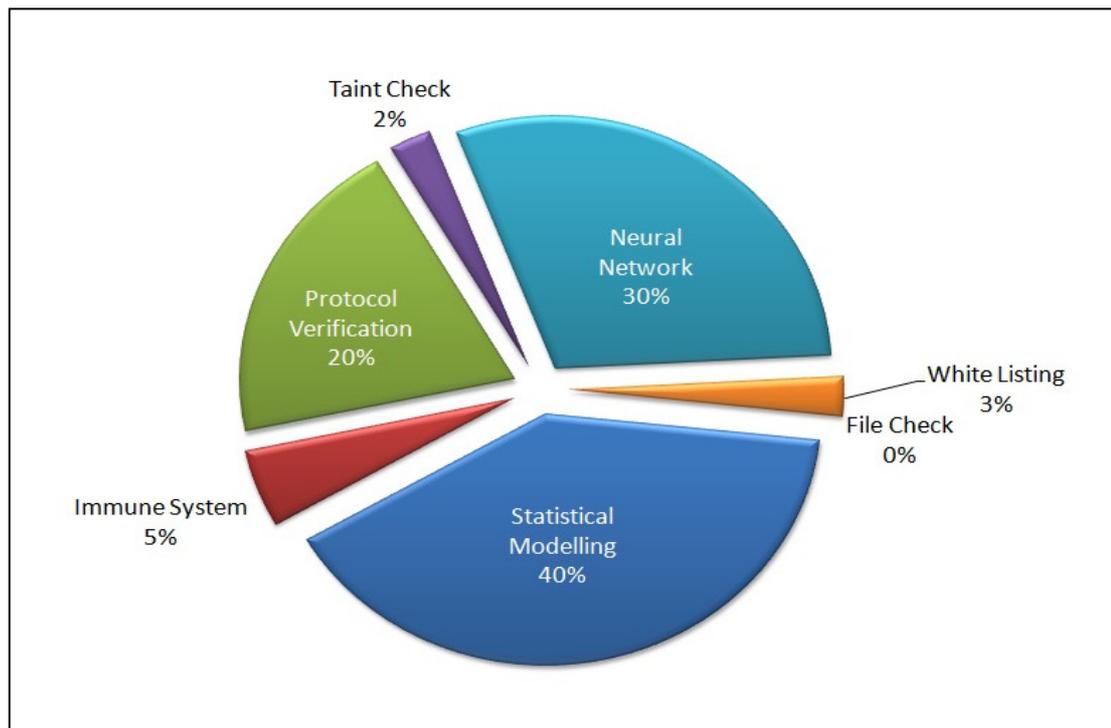


Figure 2 – 19: Anomaly Detection Techniques Distribution (1999 – 2011)

From Table 2 – 6 and Figure 2 – 19, it has been found that the most common techniques in anomaly detection are Statistical Analysis (40%) and Neural Networks (30%) followed by Protocol Verification (20%). This implies that the effectiveness of a model can be optimised if two or more techniques can be merged in one detection model. Therefore, it is desirable that any new model utilises the features of statistical analysis and neural network combined techniques with all the traffic pattern parameters that can be collected and observed during the DoS / DDoS flooding attacks. The previous work had identified certain types of protocols as parameters that can be used for flood detection, these are: TCP, IP, ICMP and UDP. The selection of the DoS / DDoS anomaly detection technique was also driven by the fact that flooding and many other attacking methods and techniques do exist and can be used to launch a DoS / DDoS attacks, however, the flooding technique seems to be the preferable method of attack. This is because flooding is more effective due to the simplicity of launching an attack and the impact on the targeted environment. This was clearly observed as shown before in Figure 2 – 6 whereby only 2.5% of the launched attacks in the first quarter of 2012 were not caused by a flooding technique (DNS attack, Prolexic Q1 2012 report).

## 2.6 CHAPTER CONCLUSION

As seen from previous sections within this chapter, a lot of research has been done on the field of DoS / DDoS addressing the flooding problem toward computer networks using anomaly detection approach.

The available published research addresses the problem as a fact of life for computerised networks within the internet age. DoS / DDoS exist within new emerging network structures connected to internet whereby any participating computer networked to the internet can launch a DoS / DDoS attack toward any victim network causing the network service to be either unavailable or unstable for a period of time.

The literature review also indicated the need for an accurate and simpler detection system to be adopted as attacks methods and targets dynamically change over time. Therefore, the main problems identified with previous work are:

1. Most used detection techniques cannot cover for unknown attacks that lead to DoS / DDoS flooding of the network with the exception of anomaly detection models.
2. Previous research is difficult to implement in real networks as tested work data sets were mainly generic set(s) of published attacking test cases from DARPA. This eventually means that when implemented it might not cover for individual network unique patterns of activities. Therefore, there are generalisability and availability issues within the proposed solutions that are independent from the used approach.
3. The number of false alerts that can be generated from existing protection methods including but not limited to IDS and IPS systems when the attack is unknown to these protection systems.

In addition, some approaches are very complex to be implemented specially in practical world (e.g. Snort implementation with large set of pre-defined and new rules applied by some approaches). Therefore, this research present a new detection model that make use of the KISS principle “keep it simple and stupid” by targeting all traffic protocols to forecast the rejected packets value as a DoS / DDoS attack measurement.

Finally, this chapter has shown the importance of further research relating to DoS / DDoS flooding on internet connected networks. Therefore, this research proposed a new detection model to overcome the previous limitation and address the DoS / DDoS attack patterns in timely response.

The new model is contributing by dynamically adjusting the rejected traffic natural growth detection threshold using a network traffic baseline database that is continuously updated. The proposed detection model baseline data structures are presented Chapter 4 of this thesis. The threshold dynamic adjustment feature of the proposed detection model is discussed in Chapter 5 of this thesis as part of the proposed detection model design. Such a design is also implemented in Chapter 6 of this thesis.

The next chapter will discuss the thesis research methodology and design used to complete this thesis and to drive the development of a new model.

# CHAPTER 3

## RESEARCH METHODOLOGY

This chapter investigates the methodology and design used to complete this research. Selecting the best research method, paradigm and approach are crucial aspects of any research to be successfully completed.

Chapter three is divided into eight main sections:

- Section one set the research objectives within the context of DoS / DDoS problem domain.
- Section two discusses research questions to be answered in accordance with the research objectives.
- Section three introduces the importance of research methodologies and the role they play in IS research.
- Section four investigates different research paradigms, methods and approaches. This section covers the selected research methodology, paradigm and approach used to complete this research.
- Section five examines the characteristics of this research.
- Section six discusses research significance and the way it contributes to theory, the body of knowledge and practice.
- Section seven examines the research process carried out using six modular phases.
- Section eight provides a conclusion for this chapter.

### **3.1 RESEARCH OBJECTIVES**

This research aims to produce a simple model for tracking and detecting DoS / DDoS flooding attacks by using anomaly detection in a dynamic integration with the firewall environment. The proposed DoS / DDoS model does not remove the need for misuse IDS or IPS systems that deal with known intrusion attacks, however, it can complement the security of the network as the first detection line of DoS and DDoS flooding attacks. Xu & Zhan (2008) suggested the use of simple mathematical function(s) to detect the attacks simply because existing methods are more static and cannot learn from past network activities fast enough to detect the attacks. Therefore, the research proposed model objective is to close the gap between static and dynamic detection of DoS / DDoS attacks by dynamically setting the detection threshold(s) using a recursive learning process to forecast the rejected level of packets travelling through the firewall interfaces.

### **3.2 RESEARCH QUESTIONS**

This research develops a model (called DoSTDM model as discussed in Chapter 5) to explore the relation between firewall patterns and the ability to track and prevent network DoS / DDoS attacks using data statistical and neural network pattern modelling by producing a dynamic network activity baseline from active firewall logs. The DoSTDM compares new data with the baseline to track changes in new logs produced daily in order to provide answers for the following research questions:

*Q1. Can new DoS / DDoS flooding attacks against the network be identified as an anomaly within firewall traffic patterns of rejected packets using the dynamic baseline only and without the need for a pre-installed sophisticated IDS / IPS components?*

*Q2. What are the accepted accuracy and threshold margins before a DoS / DDoS flooding attack is defined in order to lower the number of false alerts generated by the DoSTDM design?*

The expectation is that changes within the firewall traffic patterns can be interpreted as flooding attacks as long as the minimum margin can be dynamically adjusted to eliminate false alerts using the dynamic baseline generated.

### 3.3 INTRODUCTION TO RESEARCH METHODOLOGY

It is important to relate research methodologies to research objectives and their context with a full understanding of these methodologies' strengths and weaknesses (Jenkins 1985). Choosing a research methodology appropriate to the research at hand provides a firm foundation, guiding the researcher through the process and adding strength to the content of the research. Methodology best selection sets up the foundation for success, contributing to positive research outcomes. The application of an approach that enables the researcher to reach the desired outcomes through dependable and academically rigorous steps will ensure the research outcome is valid, consistent and has the ability to survive extensive testing with rigour (Sarker & Lee 1998). Different research methods have been used within the field of Information Systems (IS) and Eisenhardt (1989) suggests that research methods and techniques can be used in two ways; to drive new theories from data collected by empirical observations, and alternatively to support existing theories.

### 3.4 RESEARCH PARADIGMS AND APPROACHES

#### 3.4.1 Research Methods - Positivist vs. Interpretivist

Two existing paradigms are predominantly in use with IS research, these are the Positivist paradigm also known as scientific (Galliers 1991) and the Interpretivist paradigm. These two research philosophies can be applied and used with different research approaches to conduct research in the field of IS. Table 3-1 illustrates the range of research methods appropriate to these two main paradigms.

		<b>RESEARCH PARADIGMS</b>	
		<b>POSITIVIST (Scientific)</b>	<b>INTERPRETIVIST</b>
<b>RESEARCH APPROACHES</b>	Laboratory Experiments		Subjective / Argumentative
	Field Experiments		Reviews
	Surveys		Action Research
	Case Studies		Descriptive / Interpretive
	Theorem Proof		
	Forecasting		Future Research
	Simulation		Role / Game Playing

Table 3 – 1: IS Research Approaches & Philosophies (Positivist vs. Interpretivist)

Source: (Based on Galliers, 1991, 332)

Galliers (1991) suggested the use of case studies as a research approach within the Positivist paradigm only; however, it can also be used for both Positivist and Interpretivist paradigms (Shanks & Parr 2003). Fitzgerald & Howcroft (1998) characterise the two main paradigms of research (i.e. positivist and interpretivist) as follow:

- The Positivist (scientific) paradigm is based on fixed and hard tangible structures with complexity handling via reductionist thinking. It uses objectivity, quantitative measurements and testing repeatability for prediction and generalisability purposes. The positivist research is known as the “*hard*” research paradigm where the researcher is separated from the research system by hard boundaries in order to maintain total objectivity. Therefore, this type of research has been found to be more suitable for scientific research with quantitative analysis of the researched system variables.
- The Interpretivist paradigm assumes multiple realities do exist. This type of research can be criticized for its subjectivity as it depends on the interpretation of the researcher and their analysis of the researched situation using qualitative measures. The interpretivist paradigm is also referred to as a “*soft*” research paradigm as the boundaries between the researcher and researched system are soft and not clearly defined. Therefore, Interpretivist research is more suitable for social research where the boundaries cannot be defined and the researcher is part of the research.

Table 3–2 shows a complete comparison between the *positivist* and *interpretivist* paradigms at different research levels as defined by Fitzgerald and Howcroft (1998). The table shows the differences at the following research levels:

- Paradigm level
- Ontological level
- Epistemological level
- Methodological level
- Axiological level

<b>PARADIGM LEVEL</b>	
<b>Interpretivist:</b> No universal truth. Understand and interpret from researcher's own frame of reference. Uncommitted neutrality impossible. Realism of context important.	<b>Positivist:</b> Belief that world conforms to fixed laws of causation. Complexity can be tackled by reductionism. Emphasis on objectivity, measurement and repeatability.
<b>ONTOLOGICAL LEVEL</b>	
<b>Relativist:</b> Belief that multiple realities exist as subjective constructions of the mind. Socially-transmitted terms direct how reality is perceived and this will vary across different languages and cultures.	<b>Realist:</b> Belief that external world consists of pre-existing hard, tangible structures which exist independently of an individual's cognition.
<b>EPISTEMOLOGICAL LEVEL</b>	
<b>Subjectivist:</b> Distinction between the researcher and research situation is collapsed. Research findings emerge from the interaction between researcher and research situation, and the values and beliefs of the researcher are central mediators.	<b>Objectivist:</b> Both possible and essential that the researcher remain detached from the research situation. Neutral observation of reality must take place in the absence of any contaminating values or biases on the part of the researcher.
Emic / Insider / Subjective: Origins in anthropology. Research orientation centred on native / insider's view, with the latter viewed as the best judge of adequacy of research.	Etic / Outsider / Objective: Origins in anthropology. Research orientation of outside researcher who is seen as objective and the appropriate analyst of research.
<b>METHODOLOGICAL LEVEL</b>	
<b>Qualitative:</b> Determining what things exist rather than how many there are. Thick description. Less structured and more responsive to needs and nature of research situation.	<b>Quantitative:</b> Use of mathematical and statistical techniques to identify facts and causal relationships. Samples can be larger and more representative. Results can be generalized to larger populations within known limits of error.
<b>Exploratory:</b> Concerned with discovering patterns in research data, and to explain / understand them. Lays basic descriptive foundation. May lead to generation of hypotheses.	<b>Confirmatory</b> Concerned with hypothesis testing and theory verification. Tends to follow positivist, quantitative modes of research.
<b>Induction:</b> Begins with specific instances which are used to arrive at overall generalizations which can be expected on the balance of probability. New evidence may cause conclusions to be revised. Criticized by many philosophers of science, but plays an important role in theory / hypothesis conception.	<b>Deduction:</b> Uses general results to ascribe properties to specific instances. An argument is valid if it is impossible for the conclusions to be false if the premises are true. Associated with theory verification / falsification and hypothesis testing.
<b>Field:</b> Emphasis on realism of context in natural situation, but precision in control of variables and behaviour measurement cannot be achieved.	<b>Laboratory:</b> Precise measurement and control of variables, but at expense of naturalness of situation, since real-world intensity and variation may not be achievable.
<b>Idiographic:</b> Individual-centred perspective which uses naturalistic contexts and qualitative methods to recognize unique experience of the subject.	<b>Nomothetic:</b> Group-centred perspective using controlled environments and quantitative methods to establish general laws.
<b>AXIOLOGICAL LEVEL</b>	
<b>Relevance:</b> External validity of actual research question and its relevance to practice vital, rather than constraining the focus to that researchable by "rigorous" methods.	<b>Rigor:</b> Research characterized by hypothetico-deductive testing according to the positivist paradigm, with emphasis on internal validity through tight experimental control and quantitative techniques.

Table 3 – 2: Summary of Soft vs. Hard Research Dichotomies

Source: (Fitzgerald & Howcroft, 1998, 160)

### 3.4.2 Research Approaches

Due to continuous progress in technological aspects of the IS field, it is very important to select the best research approach in a way that links the research output to practice within the IS industry. Therefore, selecting an approach like quantitative vs. qualitative as the best selection of a preferred research approach for a specific research application relies upon the stated objectives for the research plus the researcher's understanding of the application domain together with any underlying theories and models obtained from the literature.

Research methods in IS can be classified under the positivist and interpretivist paradigms based on their contribution to research object (Galliers 1991). This can be seen in the research methods taxonomy presented in Table 3 – 3. Galliers (1991) taxonomy links the research objects to the research approaches including:

- Theorem Proof
- Laboratory Experiments
- Field Experiments
- Surveys
- Case Studies
- Forecasting & Future Research
- Simulation & Game / Role Playing
- Subjective / Argumentative
- Descriptive / Interpretive and Reviews
- Action Research

In the past, there has been a debate about combining these different research approaches to conduct IS research. This mixed approach has been deemed as a failure within the scientific field (Fitzgerald et al. 1985). However, some researchers within the IS field think that a combination of different approaches from different research methods (qualitative and quantitative) can perform better to support research results (Eisenhardt 1989). This had been seen clearly within the use of case study approaches (Shanks & Parr 2003).

Object	Modes for traditional empirical approaches (observational)						Modes for newer approaches (interpretation)			
	Theorem Proof	Laboratory Experiments	Field Experiments	Case Study	Survey	Forecasting and Future Research	Simulation and Game / role Playing	Subjective / Argumentative	Descriptive / Interpretive (inc. Reviews)	Action Research
<b>Society</b>	No	No	Possibly	Possibly	Yes	Yes	Possibly	Yes	Yes	Possibly
<b>Organisation/ Group</b>	No	Possibly (small groups)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
<b>Individual</b>	No	Yes	Yes	Possibly	Possibly	Possibly	Yes	Yes	Yes	Possibly
<b>Technology</b>	Yes	Yes	Yes	No	Possibly	Yes	Yes	Possibly	Possibly	No
<b>Methodology</b>	No	No	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
<b>Theory Building</b>	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
<b>Theory Testing</b>	Yes	Yes	Yes	Possibly	Possibly	No	Possibly	No	Possibly	Possibly
<b>Theory Extension</b>	Possibly	Possibly	Possibly	Possibly	Possibly	No	No	No	Possibly	Possibly

Table 3 – 3: IS Research Approaches Revised Taxonomy  
Source: (Galliers, 1991, 339)

### 3.4.3 DoS / DDoS Problem Domain & Research Paradigm

The research focuses on a problem that has already been defined by past research literature (i.e. firewall enabled systems are not suitable to provide neither large nor small computer networks with all the required protection against network attacks). In addition the research objectives and questions have been identified to explore the causal relationships between firewall patterns (obtained from day-to-day network traffic logs) and DoS attacks to answer the research questions.

The researched system under study comprises a set of programs and equipment rather than an organisation composed of humans with a tendency towards bias. Therefore, empirical testing within this research context can provide a hard, clear explanation of the links between the DoS phenomena and its causal relationship by using the high rate of rejected data patterns captured from the firewall traffic patterns.

Therefore, the selection criteria for the best research paradigm found that the positivist paradigm with a quantitative approach as a hard research approach is more suitable for this scientific research for the following reasons:

1. The practical aspects of the research problem context as understood by the researcher from past literature such as network complexity and security management overhead associated with these complex networks.
2. The research data complexity as every packet travelling across the network represents an observation (with up to  $2 \times 10^6$  packets per day).
3. The need to carry out statistical analysis to determine relationships and patterns in the collected data.

In this research, the object as described in the context within Table 3 – 3, is a large real time computerised network used by a large organisation to service thousands of users. The case study model was used to collect the data and analyse the results obtained to answer the research questions. In addition, an experimental modelling approach with a simulation computer network was used to triangulate with case study results. The research uses the quantitative method to perform data analysis of the case study and the simulated approach.

Triangulating within the case study model and the experimental model helped in testing the data mining theory in the field of IS to unleash DoS patterns within firewall logs and use Neural Network theories to learn from these patterns. Therefore, choosing these empirical research approach models with triangulation helped this research to build a methodological model that can be implemented as a technology application device in multiple computer programming languages.

Gallier's (1991) taxonomy has been adapted and extended over the years to account for new research methodologies such as the Critical Research method (Alexander 2002, p. 42) as shown in Table 3-4.

<b>RESEARCH STRATEGY</b>	<b>POSITIVIST</b>	<b>INTERPRETIVE (HERMENEUTICS)</b>	<b>CRITICAL (HERMENEUTICS)</b>
<b>Epistemology</b>	Positivism	Anti Positivism	Anti Positivism
<b>Type of Research Goal</b>	Prediction & Control	Understanding	Emancipation
<b>Research Outcomes</b>	Laws	Models	Emancipatory Models
	Hypotheses	Frameworks	Emancipatory Frameworks
	Theorem Proofs	New Concepts, Insights or Theories	New Emancipatory Concepts, Insights or Theories
	Tools	New Applications	Changed Organisation or System
	IS Instruments		
	Techniques		
	Methods		
	Application of Models		
<b>Methodology</b>	Measurement	Participant Observation, Discussion and Textual Analysis	Critical Observation and Textual Analysis Particularly of Power Structures, Discussion and Intervention.
<b>Example of Methodologies</b>	Hypothetico-Deductive Experiments.	Interpretive Case Study, Action Research, Holistic Ethnography.	Interpretive Case Study, Action Research, Critical Ethnography, Phenomenology

Table 3 – 4: IS Research Methodologies Associated with Strategies

Source: (Alexander, 2002, 42)

#### **3.4.4 Applied Research Approaches**

This research used a quantitative research methodology by combining two research approaches. The first is a case study approach with a single case organization, where the data collected consists of a set of cross-sectional past data log sets in different time frames for the case study organisation's computer network.

The second is an experimental simulation approach to overcome the limitation of generalisability related to case studies and to be able to extend theoretical knowledge with practical experiments under a set of controlled environmental conditions.

Case study research approach has been identified by Yin (1994, pp.13) as "*An empirical enquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between the phenomenon and context are not clearly evident*".

The importance of studying phenomenon in the field provides the opportunity to analyse the object of interest within its day-to-day environment where interactions and influences can be identified and studied. This provides a richer and more comprehensive perspective of the object under study, contributing to the building of new knowledge and theory. Shanks (2002, pp.78) added that "*Case study research can be used to achieve various research aims: to provide descriptions of phenomena, develop theory, and test theory*".

Case study research can be undertaken using a quantitative approach, a qualitative approach or a mix of both, in some cases it can be even limited to quantitative evidence only (Yin 1994).

This research follows the process described by Eisenhardt (1989) to conduct the case study research as shown in Table 3-5. Paré & Elam (1997) indicated that Eisenhardt's (1989) process can be used to build theories and test these theories in the information system field rather than borrowing theories from other disciplines.

<b>STEP</b>	<b>ACTIVITY</b>	<b>REASON</b>
<b>Getting Started</b>	Definition of research question	Focuses efforts
	Possibly a priori constructs	Provides better grounding of construct measures
	Neither theory nor hypotheses	Retains theoretical flexibility
<b>Selecting Cases</b>	Specified population	Constrains extraneous variation and sharpens external validity
	Theoretical, not random, sampling	Focuses efforts on theoretically useful cases i.e., those that replicate or extend theory by filling conceptual categories
<b>Crafting Instruments and Protocols</b>	Multiple data collection methods	Strengthens grounding of theory by triangulation of evidence
	Qualitative and quantitative data combined	Synergistic view of evidence
	Multiple investigators	Fosters divergent perspectives and strengthens grounding
<b>Entering the Field</b>	Overlap data collection and analysis, including field notes	Speeds analyses and reveals helpful adjustments to data collection
<b>Analysing Data</b>	Flexible and opportunistic data collection methods	Allows investigators to take advantage of emergent themes and unique case features
	Within-case analysis	Gains familiarity with data and preliminary theory generation
<b>Shaping Hypotheses</b>	Cross-case pattern search using divergent techniques	Forces investigators to look beyond initial impressions and see evidence thru multiple lenses
	Iterative tabulation of evidence for each construct	Sharpens construct definition, validity, and measurability
<b>Enfolding Literature</b>	Replication, not sampling, logic across cases	Confirms, extends, and sharpens theory
	Search evidence for "why" behind relationships	Builds internal validity
	Comparison with conflicting literature	Builds internal validity, raises theoretical level, and sharpens construct definitions
<b>Reaching Closure</b>	Comparison with similar literature	Sharpens generalisability, improves construct definition, and raises theoretical level
	Theoretical saturation when possible	Ends process when marginal improvement becomes small

Table 3 – 5: Process of Building Theory from Case Study Research

Source: (Eisenhardt, 1989, 533)

Simulation research using a good experimental design can enhance the efficiency of the research discovery process by generating a good set of data, making research analysis much easier and cleaner (Dooley 2002). The use of simulation in IS research allows the researcher to not only investigate complex situations, but also diagnose the effects of influencing factors and make real life decisions in a safe and isolated testing environment. In many circumstances simulation is also used to test theories by simulating environments in future settings, having the ability to change the value of parameters and study the subsequent effects. In simulation the researcher needs to study the real world scenario (under investigation) and model an experiment based on the assumption(s) captured from the real scenario (Becker et al. 2005). The classification of a simulation research method depends on the specifications and the criteria of the conducted research problem domain as shown in Table 3-6.

<b>CRITERIA</b>	<b>SPECIFICATIONS</b>		
<b>Time Reference</b>	Static		Dynamic
<b>Intention</b>	Descriptive		
	Description	Explanation	Prediction
<b>Change of State</b>	Deterministic		Stochastic
<b>Time Model</b>	Discrete		Continuous
<b>Procedure Control</b>	Event-Driven		Time-Driven

Table 3 – 6: A Morphologic Box Classification of Simulation Research Method

Source: (Becker et al., 2005, 3)

It is important to note that the problem definition can determine the complexity level for the simulation (Becker et al. 2005). The simulation and forecasting model for this research is based upon an experimental design of the computer network that represents the basic elements of a secure network. The experiment employs a firewall as its protection mechanism for a continuous six month experiment. Therefore, the simulation design used is a time model simulation as the research captured packet activities over a continuous period of time.

### **3.5 RESEARCH CHARACTERISTICS**

As the research aims to explore the DoS problem domain, the research questions can be answered using the baseline models generated from the case study results and compared to simulation experimental results to explore the traffic patterns anomalies. The previous combined methodologies provide the following research characteristics to support the research outcomes:

- The research reliability was measured using the ability to obtain consistent forecasted data results when using multiple forecasting methods. In addition, cross referencing the results with real time detected attacks within the same time line period has been used to support the case study findings. This is very important in order to accept or reject any produced results based on the baseline model comparison.
- Triangulation between quantitative analysis results using multiple statistical forecasting techniques and neural network forecasting techniques (i.e. Holt-Winter forecasting and Linear Regression analysis) was used to provide internal validity for this research. Therefore, the produced baseline model can be generalised to other network environments. The only condition is that any tested environment needs to be running the standard TCP / IP protocol suite and not a proprietary communication method to connect to the Internet.
- The ability exists to generalise the model between different network configurations and security policies implemented in different operating systems and different hardware platforms as long as the baseline model variables are shared among these networks.
- Research rigour is ensured by the use of deductive testing and triangulation between multiple quantitative analysis results.

### **3.6 RESEARCH SIGNIFICANCE**

This research contributes to both theory and practice. The contribution to theory involved the development of the DoSTDM model, while the contribution to practice

involves setting up procedural guidelines to implement the model to any network that conforms to the same networking standards of security devices using firewalls as the first line of defence to reduce the risk of DoS flooding attacks.

### **3.6.1 Contribution to Theory**

This research develops and uses the proposed research model to explore the relation between firewall patterns and the ability to track and prevent network DoS / DDoS by using case study and simulation to understand the relationships between traffic patterns and DoS /DDoS attacks.

The proposed research model uses Artificial Neural Networks (ANN) theories to dynamically learn from past experience. It also uses the Human Immune System (HIS) and Artificial Immune System (AIS) analogies by detecting foreign network activities (in the form of high levels of rejected packets) and reacting to these activities by raising alerts or disabling certain network functions in the same way the human body detects foreign objects (viruses and germs) using antigens and reacts by attacking these foreign objects with antibodies.

In addition, statistical data mining and mathematical modelling are used to triangulate results and obtain accurate measurements. The proposed research model findings can be an application extension or expansion of these scientific theories within the field of DoS and DDoS protection in a dynamic format.

The expansion of Information Systems theories by scaffolding from HIS and AIS theories can help in solving many problems and not just DoS and DDoS attacks. These expansions have been theoretically studied, however, the main aspects were to deal with complex mathematical decision making by computer systems and not specifically toward DoS and DDoS ground work (De Castro & Timmis 2002).

Therefore, by using the HIS theoretical knowledge and combining this knowledge with data mining and Neural Networks algorithms, a new dimension of knowledge represented by a dynamic modelling and implementation for network security can be observed.

### 3.6.2 Contribution to Practice

The existing Internet open environment has generated substantial security concerns within the business community and government entities. As a result organisations have been forced to implement some very expensive systems in both software and hardware to manage trust in the security domain.

However, these systems are still suffering from the following problems:

1. The implemented detection and prevention system comes with an additional cost tag to operate computer networks effectively.
2. Most commercial security products are proprietary and cannot be transferred from one hardware and / or software platform to another.
3. There is a significant management overhead with high rates of false alerts.

From a practical point of view, it is believed that intrusion detection and prevention is not a product as much as it is a function. Therefore, working in the field of DoS and DDoS requires a function that can evolve with these attacks dynamically rather than a static product that needs updates and / or upgrades to continue protecting the network after every new attack (as it is the case with many commercial security appliances and implementations). The most significant practical contribution of this research is the simplicity approach to resolve the DoS / DDoS problem. This contribution is presented by choosing one aspect of the problem to be targeted (i.e. the amount of rejected packets by the firewall and how this amount changes between normal and abnormal situation). Therefore, tracking and predicting this aspect presents a model with a simple and basic rule of protection that is easy to manage and track. The effect of the proposed detection model simplistic approach can be seen when dealing with existing IDS products like Snort and managing an enormous number of rules that control the detection process. Therefore, this research attempts to minimise the impact of DoS and DDoS flooding attacks by using existing firewall network technology within the network to protect itself by using the proposed model self-learning approach as an additional firewall function rather than a new product. The proposed model can be implemented in any network environment as long as it uses a firewall and complies with the standard TCP / IP communication protocol using a set of guidelines (described in details in Chapter 4).

## **3.7 RESEARCH PROCESS DESIGN**

The research follows a systematic process to collect, design and analyse the proposed DoS detection model. The research process involved six phases to complete the research as explained in the following.

### **3.7.1 Phase 1 – Research Literature and Design**

This phase covers the steps used to initiate the research by investigating literature within the field of the research. The literature review covers the theoretical and practical background of problem domain, associated protection technologies and past solutions to DoS / DDoS attacks. It also covers the selected research methodology, approach and the research process design to carry on this research. Chapters 2 and 3 cover this phase of the research process design.

### **3.7.2 Phase 2 – Research Case Study**

This phase covers the selected case study network and the investigation made to collect and prepare firewall log data. This phase of the research process contributes to the overall process by transforming raw firewall data that can be obtained from any firewall log from any network to a structured data set (i.e. traffic characteristics per packet containing date, time, protocol, direction and firewall action).

This phase highlights the important contribution of the case study network to understand traffic patterns by data mining firewall logs to:

1. Explore and understand the case study network structure and components.
2. Collect the raw data from the firewall logs
3. Design the research instrumentation tools: these are a set of Java and shell scripts to re-structure firewall log raw data.
4. Generate the case study restructured data set using research instrumentation from the previous step.

The selected case study data collection, preparation and re-structuring process will be used in the following phase of the research process to aid building the simulation experiment network test bed. Chapter 4 covers this phase of the research process.

### **3.7.3 Phase 3 – Experimental & Simulation Research Bed**

This phase covers the setup and building of a simulated network to create a baseline set of data that was used to train the proposed model. The simulation network represents a set of network components within the LAN premises and communicates with external networks using a WAN gateway while all network traffic passes through the firewall. External simulation of traffic was achieved using a TCP / IP packet generator tool to simulate normal and abnormal network activities. In addition, the simulation network was used later for testing and comparing different model results with the case study data set.

The simulation network uses an open source utility called Nemesis which is a packet injection tool to replicate a real world network traffic activity from a network structural point of view in order to construct the simulation environment (please refer to Section 6.2.1 for full details of the simulation network structure used with this research). However, packets injected by Nemesis had been randomized using the random function in Linux so that the traffic packets level is not pre-determined during the simulation testing.

This phase is introduced in Chapter 4 and implemented in detail in Chapter 6. The simulation network helped in testing the proposed model design introduced in Chapter 5.

### **3.7.4 Phase 4 – DoS Tracking and Detection Model Design & Testing**

This phase covers the proposed model design process and the steps used to evaluate the model on the simulated network. The model is called DoSTDM (Denial of Service Tracking and Detection Model).

This phase also includes the iterative steps used to enhance DoSTDM model performance for fast detection and testing triangulation with statistical methods. This phase is covered in detail in Chapter 5. The testing is conducted in Chapter 6 using all data sets collected from a real case study network and simulation network firewall logs to compare the model prediction based on the rejected packet levels.

### **3.7.5 Phase 5 – Research Findings & Discussion**

This phase covers DoSTDM model performance analyses and discusses research findings to answer the research questions. It also highlights any implementation limitations discovered during the testing phase of the DoSTDM model. Evaluation of the research approach is then performed and reflections on the research process and resulting content are discussed.

The research findings were analysed by triangulating between different data sets from the case study and simulation tests using error measurements in predicting the rejected packet levels within the network in normal and abnormal situations as presented in the design of DoSTDM within Chapter 5.

This phase of the research process is covered in Chapter 6 and the research discussion in Chapter 7 elaborates on these findings.

### **3.7.6 Phase 6 – Research Documentation & Submission**

This final phase involved writing up the research thesis, including research literature review, research methodology and design, DoSTDM design and implementation, DoSTDM results analysis and discussion of research findings. The final step upon completing this research is the submission of research thesis for thesis examination purposes.

Finally, Figure 3 – 1 shows how design process six phases and how they integrate with each other to deliver the research findings and complete this thesis.

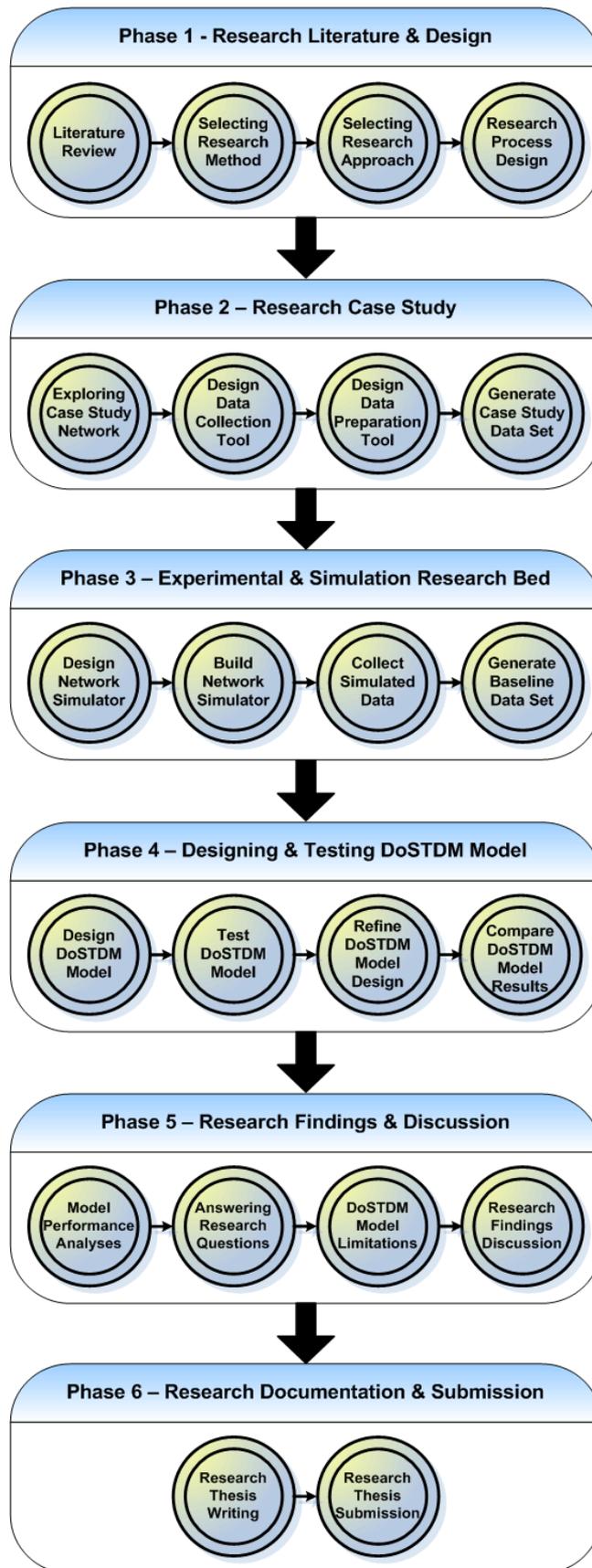


Figure 3 – 1: Research Design Process

### **3.8 CHAPTER CONCLUSION**

This chapter has emphasised the research aims to explore the theoretical grounds of existing causal relationships between DoS attacks and past network history to develop the DoSTDM model. In addition, the research model borrows from other scientific theories to model DoSTDM as a computer immune system protecting against DoS flooding attacks using native firewall log data as a baseline for immunity and defence. Therefore, DoSTDM facilitates the use of past firewall logs as a baseline to isolate DoS flooding attacks toward the network (as per the DoSTDM implementation guidelines).

In this chapter the scientific empirical nature of this research is emphasised to justify the research paradigm, methodology and approach. Therefore, in conclusion, this research follows the positivist paradigm, case study and experimental simulation approaches embracing a quantitative methodology. The developed research process has been used and implemented systematically to produce the model as a functional enhancement to any firewall to predict and detect DoS flooding attacks without adding another piece of equipment to the network.

The next chapter will discuss the design and implementation of the DoSTDM model as a direct result of the research process developed in section 3.6 of this chapter.

The following chapters will also answer the research questions based on quantitative statistical analyses that can be applied to any computer network upon adding the same model structure and process to existing firewall environment using DoSTDM model implementation guidelines.

# CHAPTER 4

## **DATA COLLECTION: SOURCES & STRUCTURE ANALYSIS**

This chapter covers the steps used to collect, format and re-structure the firewall logs data obtained from the case study and the simulation network to construct research data variables and initiate the process of designing and implementing the proposed DoS / DDoS protection model.

Chapter four is divided into seven main sections:

- Section one introduces to data collection and the way it follows the research process as explained earlier in Chapter 3.
- Section two identifies the nature of firewall logs data sources used to model the data structure design. It also discusses the infrastructure of the case study and simulation networks.
- Section three discusses model data structures used within the context of this research.
- Section four discusses the model data collection staged process and storage of research data.
- Section five discusses the data pre-processing and preparation process.
- Section six discusses the findings from data mining the firewall logs.
- Section seven provides a conclusion to this chapter.

## 4.1 INTRODUCTION

The literature review indicated the need for a simpler and more efficient detection and prevention system to handle new DoS / DDoS attacks. Therefore, this model design process is based around reutilization of existing network resources. The main resource to be used is firewall logs, and firewall logs are recorded by every network mainly for auditing and access tracking purposes. It is important to relate the model design to network resources in order to minimise changes to the network environment and to optimise the performance of the model. Firewall logs are rich with information that can be used to track computerised network activities in various ways. The data obtained from these logs can determine the way the network firewall is configured to protect itself and other network resource settings behind this firewall. In an enterprise environment these are normally load balanced firewalls that can provide fault tolerance to each other to provide 24x7 system protection. However, the network firewall implementation complexity level does not change the principle of communication method in use and it remains the same in small, medium and large networks using the standard TCP / IP communication suite.

In Chapter 3, the proposed research design phases of this research were discussed (as shown in Figure 3 – 1) which shows the reliance in design phases 2, 3 & 4 on the data collection and analysis process. Therefore, data collection and structuring has been implemented to design the proposed model as part of the following research phases:

- Research phase 2: Data collection and structuring using case study network data to help understand firewall data components used within the proposed model.
- Research phase 3: Building the training and testing simulation environment with a variety of forecasting methods using firewall traffic protocols data which is identified from research phase 2 (i.e. TCP, IP, UDP and ICMP).
- Research phase 4: Forecasting model design based on the rejected packets within the firewall and the comparison of these levels between real network data and simulated data obtained from the simulation networks.

Therefore, the following sections investigate the building blocks of data used to design the proposed DoS / DDoS protection model as obtained from both non-controlled (case study) and controlled environments (simulation network).

## **4.2 RAW DATA SOURCE**

### **4.2.1 Generic Firewall Logs**

Firewall logs can be logged to either the firewall itself or to an external log server. In both cases the format of the log is the same and can be extracted to multiple formats of either binary or text format. Binary formats required special tools to analyse these logs (normally provided by the firewall manufacturer).

However, extracting the data in text format will make it easier to handle the logs with any open source or in-house development tools to study and analyse these logs. In the context of this research, the text format was chosen to simplify future implementations of the DoS / DDoS proposed protection model using any analysis tool of choice.

Different firewall vendors have different raw formats of these log files as shown in Table 4 – 1. Typical firewall logs can be huge and hard to explain, however, the basic information collected by the firewall log data can be classified as a set of large number of cases where every line in the log represent a case study by itself. Each line contains the records of a packet transaction passing through the firewall.

These packets travelling across the firewall adhere to firewall rules used to control traffic passing through the firewall. The recorded activity per packet contains the following information (among other information recorded by the firewall):

- Date and time of packet transaction.
- Source packet IP address.
- Target packet IP address.
- Direction of transaction (Inbound or Outbound).
- TCP / IP suit sub-protocol (TCP, IP, ICMP & UDP) used for this transaction.
- Firewall reaction to this packet transaction (Accepted or Rejected / Dropped).

Additional information recorded by the firewall can be used for a specific packet investigation. However, these are not common across multiple firewalls and are not part of the data collected and used by the DoS / DDoS proposed protection model.

Checkpoint Firewall								
date	time	orig	action	if_name	if_dir	src	dst	proto
30Apr2004	0:00:00	192.45.0.71	accept	qfe0	outbound	78.153.234.99	192.45.0.99	tcp
30Apr2004	0:00:00	192.45.0.71	accept	qfe0	inbound	78.153.234.99	192.45.0.99	tcp
30Apr2004	0:01:13	192.45.0.72	accept	qfe1	outbound	192.45.0.72	10.56..1.40	udp
30Apr2004	0:00:00	192.45.0.72	drop	qfe0	inbound	217.217.37.94	10.56..114.1	tcp
30Apr2004	0:00:00	192.45.0.71	accept	qfe0	inbound	78.153.234.99	192.45.0.99	udp
30Apr2004	0:00:00	192.45.0.72	accept	qfe0	inbound	78.153.234.99	192.45.0.99	tcp
30Apr2004	0:00:00	192.45.0.72	accept	qfe1	inbound	10.56.97.12	210.150.254.124	tcp
30Apr2004	0:00:00	192.45.0.72	accept	qfe0	inbound	78.153.234.99	192.45.0.99	tcp
30Apr2004	0:00:00	192.45.0.72	accept	qfe1	inbound	10.56.97.12	67.28.114.32	icmp

Juniper Networks Firewall
Time of Log: 2004-10-13 10:37:17 PDT, Filter: f, Filter action: accept, Name of interface: fxp0.0 Name of protocol: TCP, Packet Length: 50824, Source address: 172.17.22.108:829, Destination address: 192.168.70.66:513
Time of Log: 2004-10-13 10:37:17 PDT, Filter: f, Filter action: accept, Name of interface: fxp0.0 Name of protocol: TCP, Packet Length: 1020, Source address: 172.17.22.108:829, Destination address: 192.168.70.66:513
Time of Log: 2004-10-13 10:37:17 PDT, Filter: f, Filter action: accept, Name of interface: fxp0.0 Name of protocol: TCP, Packet Length: 49245, Source address: 172.17.22.108:829, Destination address: 192.168.70.66:513
Time of Log: 2004-10-13 10:37:17 PDT, Filter: f, Filter action: accept, Name of interface: fxp0.0 Name of protocol: TCP, Packet Length: 49245, Source address: 172.17.22.108:829, Destination address: 192.168.70.66:513
Time of Log: 2004-10-13 10:37:17 PDT, Filter: f, Filter action: accept, Name of interface: fxp0.0 Name of protocol: TCP, Packet Length: 49245, Source address: 172.17.22.108:829, Destination address: 192.168.70.66:513

Microsoft Windows Server Firewall
#Version: 1.5
#Software: Microsoft Windows Firewall
#Time Format: Local
#Fields: date time action protocol src-ip dst-ip src-port dst-port size tcpflags tcpsyn tpack tcpwin icmptype icmpcode info path
2010-07-17 12:55:44 ALLOW TCP 192.168.0.3 209.62.182.33 57290 80 0 - 0 0 0 - - - SEND
2010-07-17 12:55:52 ALLOW TCP 127.0.0.1 127.0.0.1 57291 6999 0 - 0 0 0 - - - SEND
2010-07-17 12:55:52 CLOSE TCP 127.0.0.1 127.0.0.1 57291 6999 0 - 0 0 0 - - - RECEIVE
2010-07-17 12:55:52 ALLOW TCP 192.168.0.3 69.147.86.169 57292 80 0 - 0 0 0 - - - SEND
2010-07-17 12:55:54 ALLOW UDP 192.168.0.3 192.168.0.255 137 137 0 - - - - - SEND
2010-07-17 12:56:03 ALLOW TCP 127.0.0.1 127.0.0.1 57293 6999 0 - 0 0 0 - - - SEND
2010-07-17 12:56:03 ALLOW TCP 127.0.0.1 127.0.0.1 57293 6999 0 - 0 0 0 - - - RECEIVE
2010-07-17 12:56:03 ALLOW UDP 192.168.0.3 192.168.0.1 56892 53 0 - - - - - SEND
2010-07-17 12:56:03 ALLOW ICMP 192.168.0.3 67.195.145.137 - - 0 - - - - 8 0 - SEND
2010-07-17 12:56:03 ALLOW TCP 192.168.0.3 69.147.86.169 57294 80 0 - 0 0 0 - - - SEND

Filseclab Firewall
Filseclab Personal Firewall Log files
Time RuleId Action Application Protocol Direction Local Port Remote Port Status
30-Dec-2006 23:53:20 10001 Pass SYSTEM TCP In 192.168.3.2 63156 127.245.247.66 4772 RDSD
30-Dec-2006 23:53:20 10001 Pass SYSTEM TCP In 192.168.3.2 19084 127.230.130.109 16230 RDSD
30-Dec-2006 23:53:30 27 Pass svchost UDP Out 192.168.3.2 137 192.168.3.255 137 RDSD
30-Dec-2006 23:54:05 2 Deny SYSTEM ICMP In 192.168.3.2 60707 185.82.238.33 31226 RECV
30-Dec-2006 23:54:05 2 Deny SYSTEM ICMP In 192.168.3.2 10502 7.198.42.4 51054 RECV
30-Dec-2006 23:54:05 2 Deny SYSTEM ICMP In 192.168.3.2 48925 22.127.188.31 54999 RECV

Table 4 – 1: Different Raw Firewall Log Samples

The proposed DoS / DDoS protection model required the data to be extracted from raw format and structured in a unified format that contains common data across different firewalls. Section 4.3 shows how these raw formatted logs were used to construct the basic elements for the model in a way that can generalize the useability of the DoS / DDoS protection model.

#### 4.2.2 The Case Study Network Infrastructure

The case study of this research was a large government organisation networked facility. It incorporates a mix of large numbers of networked systems incorporated from different software vendors. This large overall network is currently protected with one of the leading international vendors of firewall software, Check-Point Firewall-1. Figure 4 – 1 illustrates the general layout of this firewall within the organisation’s network infrastructure:

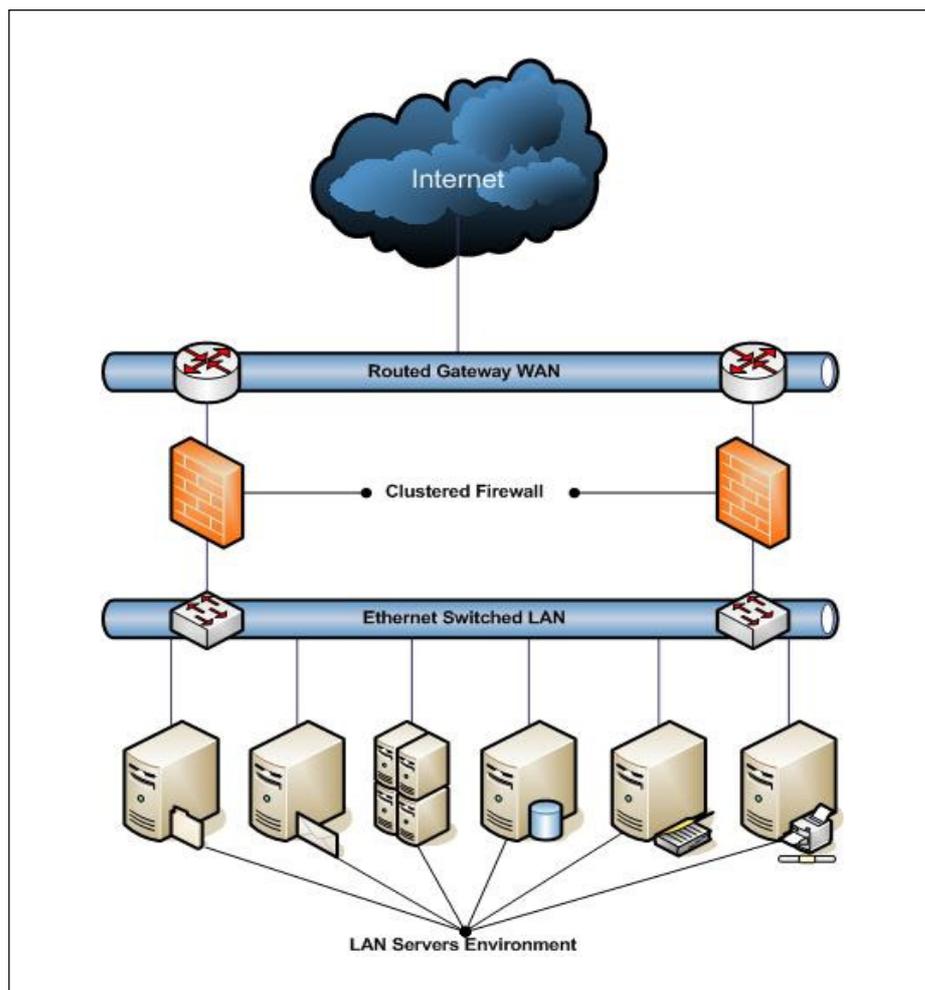


Figure 4 – 1: DoSTDM Case Study LAN / WAN Network Layout

The firewall runs on a clustered Sun Microsystems UNIX platform that serves multiple sites within the organisation. Both Servers and hosts sitting behind the firewall represent a wide range of networked clients. The case study network overall supported:

- 50000 Users.
- 40000 Workstations (some workstations are shared between multiple users).
- 4000 Cisco switches and routers distributed across both LAN and WAN infrastructures.
- Two large data centres and more than 400 remote sites.
- 1000 Microsoft Windows servers.
- 200 Unix / Linux servers.
- 1.4 Petabytes (1 Petabyte = 1,000,000,000,000,000 bytes) of shared data hosted in a Storage Area Network (SAN).

As with any large network the firewall configuration had been hardened to eliminate any possibility for network attacks to gain access behind the firewall, therefore, the Check-Point firewall was only logging activities of open ports as required by the applications used within the environment. In addition logging did include logging of dropped packets from LAN to WAN and vice versa and these were classified as rejected packets. The collected data from this network included firewall logs for two periods: January 2004 to May 2004 and January 2005 to December 2005. This data range of the data was carefully selected due to changes made to the case study network to improve security in 2005 by adding an IDS system. The modification was made to support the existing Checkpoint firewall security levels and to prevent any further attacks similar to the attacks observed in 2004. The data collected was on Checkpoint firewall log format containing all packets passed through the firewall on these two periods:

- The date the packet passed through the firewall interface.
- The time the packet passed through the firewall interface.
- The IP address of the firewall interface that recorded the log entry.
- The action taken by the firewall interface (i.e. accept or reject the packet)
- The firewall interface name that the packet pass through

- The direction of the traffic (i.e. inbound toward the firewalled network from the internet or outbound from the firewalled network to the internet)
- The IP address of the packet source.
- The IP address of the packet destination.
- The protocol used by the packet to transfer the data (i.e. TCP, IP, UDP, ICMP, etc.).

#### 4.2.3 The Simulated Network Infrastructure

The simulated network consists of both hardware and software components that simulates a complete network that incorporate LAN, WAN users and computers in a real network. The hardware component includes two servers. The first server represents the Internet cloud and the WAN interfaces while the second server represents the LAN services network protected by the firewall. The simulation network was assembled in the first quarter of 2010 and tested from June 1<sup>st</sup>, 2010 and December 31<sup>st</sup>, 2010. The software components include the operating systems that run the environment, the firewall protecting the environment and the attacking software to launch DoS / DDoS attacks using multiple protocols from randomly generated addresses to simulate attacking hosts as shown in Figure 4 – 2.

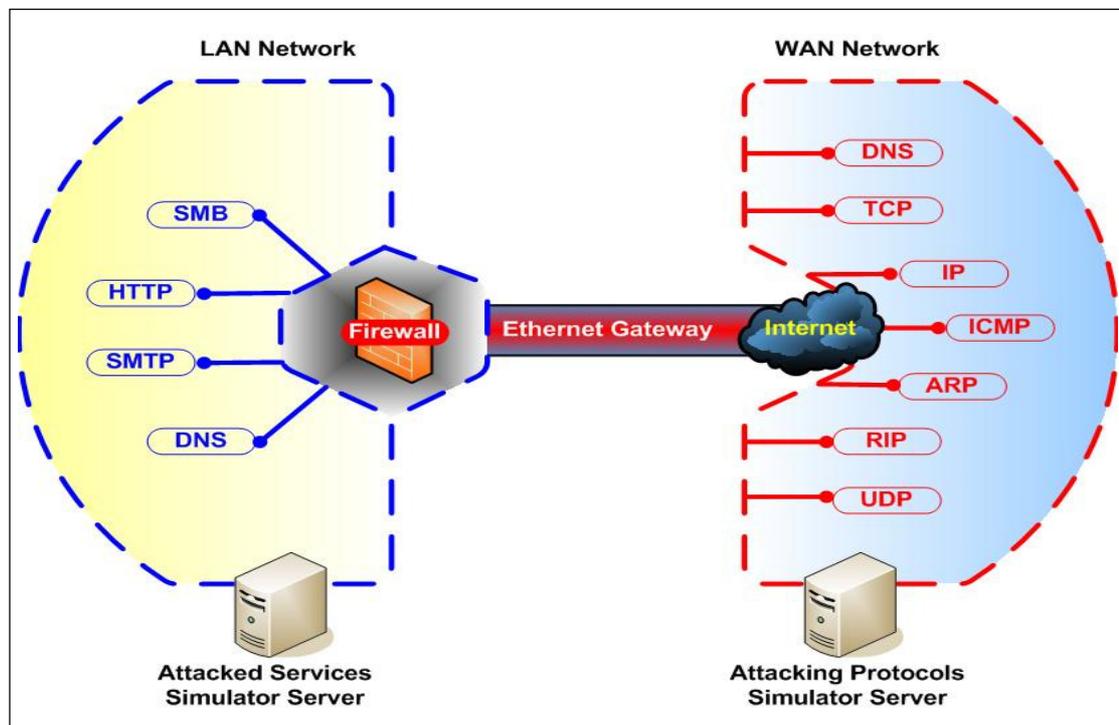


Figure 4 – 2: Experimental Simulation Network

Simulating the packets travelling from attacking hosts to the attacked network was achieved by using an open source packet injection software from Nemesis (<http://nemesis.sourceforge.net>) running on the attacking server platform of Linux. The attacked network was running on a Microsoft server with the firewall installed on it. This incorporate multi protocols and services like:

- Address Resolution Protocol (ARP)
- Domain Name services (DNS),
- Internet Protocol (IP)
- Internet Control Message Protocol (ICMP)
- Routing Information Protocol (RIP)
- Transmission Control Protocol (TCP)
- User Datagram Protocol (UDP)

A simple open source firewall implementation had been used from Filseclab (<http://www.filseclab.com/eng/products/firewall.htm>). The reason behind using this firewall was to show that complex firewalls (i.e. Checkpoint firewall used with the case study network) and simple firewalls like Filseclab can generate the same firewall log data but with different formats. The attacked network runs some standard services controlled by the firewall like:

- E-Mail Services with Simple Mail Transfer Protocol (SMTP)
- File sharing with Server Message Block (SMB)
- Name resolution with Domain Name Services (DNS)
- Web services with Hyper Text Transfer Protocol (HTTP)

The data collection process helped in constructing the simulation experimental network using the data format obtained from the case study network. This was achieved by collecting the first stage of data from the case study network. The objective behind using the simulation network is to triangulate the prediction model error analysis between the real network data (obtained from the case study large network) and the simulation data (obtained from a small simulation network) in order to measure the proposed model detection accuracy between different networks.

Therefore, this simulation network approach had contributed to the proposed model findings by measuring the proposed model results consistency between large and small network. The next sections explain the detailed structure of the firewall log data and the staged approach used to collect and re-construct the data to support the proposed model approach.

### 4.3 PROPOSED MODEL DATA STRUCTURES

#### 4.3.1 Basic Data Structures & Data Conversion

As explained earlier in the literature review, a lot of IDS systems concentrate in investigating the full contents of every packet header using the pre-defined rules.

However, the research proposed model approach uses the status of the packet to build up the current picture of the system DoS level by using: date, time, source, destination, direction, network protocol and firewall reaction to any passing packet through the firewalled network.

This data can be obtained from raw firewall logs and organised before being accessed by the proposed model forecasting engines as shown in Table 4 – 2.

Date	Time	Action	Direction	Source	Destination	Protocol
.....	.....	.....	.....	.....	.....	.....
dd/mm/yy	hh:mm:ss	Accept	inbound	192.168.114.24	121.10.12.121	icmp
dd/mm/yy	hh:mm:ss	Drop	inbound	192.168.12.35	62.25.100.32	tcp
dd/mm/yy	hh:mm:ss	Accept	inbound	192.168.1.50	203.166.25.130	tcp
dd/mm/yy	hh:mm:ss	Accept	inbound	192.168.114.24	121.10.12.120	icmp
dd/mm/yy	hh:mm:ss	Drop	inbound	218.57.92.23	192.168.87.210	tcp
dd/mm/yy	hh:mm:ss	Accept	inbound	192.168.114.24	121.10.12.122	icmp
dd/mm/yy	hh:mm:ss	Accept	inbound	192.168.114.24	121.10.12.23	icmp
dd/mm/yy	hh:mm:ss	Accept	inbound	192.168.1.50	17.250.248.32	tcp
dd/mm/yy	hh:mm:ss	Accept	outbound	192.168.1.41	207.88.7.248	udp
dd/mm/yy	hh:mm:ss	Accept	outbound	121.11.14.98	204.179.240.7	udp
dd/mm/yy	hh:mm:ss	Accept	outbound	192.168.114.24	121.10.12.27	icmp
dd/mm/yy	hh:mm:ss	Accept	inbound	192.168.1.50	203.36.194.181	tcp
dd/mm/yy	hh:mm:ss	Accept	inbound	121.10.12.20	203.18.56.43	udp
dd/mm/yy	hh:mm:ss	Drop	inbound	80.8.120.76	192.168.10.115	tcp
dd/mm/yy	hh:mm:ss	Accept	inbound	192.168.114.24	121.10.12.29	icmp
dd/mm/yy	hh:mm:ss	Accept	inbound	192.168.114.24	121.11.14.113	icmp
.....	.....	.....	.....	.....	.....	.....

Table 4 – 2: Model Data Structure Sample

Therefore, a conversion process is required to convert the raw data from Table 4 – 1 to the structured data in Table 4 – 2. Re-formatting and structuring of the data can be achieved with a variety of tools to extract the required elements (i.e. date, time, action, direction, source, destination and protocol). The aim was to format the data in a way that makes it simpler to handle different types of firewall logs no matter how large or small is the firewall network in use. These tools are either unique to the Firewall platform or common across all platforms as follow:

- Operating system scripting environments that depend on a specific platform such as UNIX / Linux shell scripts or Microsoft Windows command line scripting tools.
- Common scripting tools like multi platform programming and scripting languages such as Perl or Java.

In this research, a mix of shell, Perl and Java scripts has been used to achieve this conversion process. This has provided a portable conversion option between different hardware and software implementations of firewalls that support generalising the DoSTDM implementation process.

#### **4.3.2 Data Measurements & Pre-Processing**

The data measurement unit used by the proposed model is the number of rejected packets of a single protocol per hour. This means that the number of cases generated from a single day of data collection is 24 cases. The choice of such a measurement unit has been based upon understanding the nature of DoS and DDoS attacks response from firewalls; in other words, when an attack is launched the firewall reacts to the high volume of packets by either rejecting or dropping these packets.

This behaviour by the firewall causes the flow of packets transmission across the network to be stopped but at the same time the network can be unusable either completely or partially depending on the packets volume in comparison with the network bandwidth in use.

The data sample in Table 4 – 2 shows the direction of the packets and the firewall response to these packets based on pre-set firewall rules. The measurement unit of

this data is the rejected packets which can be referenced as dropped packets. The difference between the two terms of reject vs. drop is based upon the fact that a rejected packet will also mean an ICMP packet to be sent back to the source address. Silent dropping of packets is more helpful as it does not tell the attacker about the firewall port status. Therefore, the data preparation process re-distributes the logged data based on the activity of these protocols as follow:

- **Inbound:** Any packets traffic generated from LAN to WAN environment.
- **Outbound:** Any packets traffic passing from WAN to LAN environment.
- **Accepted:** Any packets accepted by the firewall rules as legitimate transaction.
- **Rejected:** Any packets rejected by firewall rules restrictions.

Merging protocols with their corresponding direction and firewall reaction can give a clear picture of what is occurring inside the network before, during and after an attack. Therefore, a correctional matrix of these data structures is built with the additional fields for date and time. Table 4 – 3 shows the re-distribution of inbound packets for a sample set of data over a period of time.

Date	Time	TOTAL INBOUND	TCP IN	IP IN	ICMP IN	UDP IN	OTHER IN
.....	.....	.....	.....	.....	.....	.....	.....
dd/mm/yy	hh:mm:ss	149208	136744	0	805	11630	29
dd/mm/yy	hh:mm:ss	21483	9194	0	804	11457	28
dd/mm/yy	hh:mm:ss	27899	16394	0	804	10674	27
dd/mm/yy	hh:mm:ss	51371	40073	0	804	10465	29
dd/mm/yy	hh:mm:ss	18896	7503	0	800	10564	29
dd/mm/yy	hh:mm:ss	19648	8011	0	806	10803	28
dd/mm/yy	hh:mm:ss	90797	78269	0	849	11651	28
dd/mm/yy	hh:mm:ss	25002	11386	0	883	12705	28
dd/mm/yy	hh:mm:ss	39215	25656	0	856	12674	29
dd/mm/yy	hh:mm:ss	97676	84421	0	869	12358	28
dd/mm/yy	hh:mm:ss	82523	68258	0	876	13358	31
dd/mm/yy	hh:mm:ss	23228	9080	0	889	13229	30
dd/mm/yy	hh:mm:ss	24725	9571	0	865	14261	28
dd/mm/yy	hh:mm:ss	24441	9770	0	869	13772	30
dd/mm/yy	hh:mm:ss	25534	10488	1	834	14182	29
dd/mm/yy	hh:mm:ss	27657	14082	0	853	12694	28
.....	.....	.....	.....	.....	.....	.....	.....

Table 4 – 3: Model Proposed Data Structure per Protocol for an Inbound Sample

The final data structure contains some extra fields added for any additional protocols that might be travelling across the network but represent special secure or private network services (referenced as Other Protocols). Section 5 of this chapter provides a listing of all scripts and classes used for data pre-processing using Java to reconstruct firewall log data and prepare it in the required data structure format.

Table 4 – 4 shows the matrix of all data variables created by merging packets protocol, status and direction.

<b>Data Field Name</b>	<b>Data Field Description</b>
DAY	Date of occurrence in dd-mmm-yyyy format
TIME	Time in 24 hours unit format
INBOUND	Total inbound number of all packet types per hour
TCP_IN	Total inbound number of TCP packets per hour
IP_IN	Total inbound number of IP packets per hour
ICMP_IN	Total inbound number of ICMP packets per hour
UDP_IN	Total inbound number of UDP packets per hour
OTHER_IN	Total inbound number of other types of packets per hour
OUTBOUND	Total outbound number of all packet types per hour
TCP_OUT	Total outbound number of TCP packets per hour
IP_OUT	Total outbound number of IP packets per hour
ICMP_OUT	Total outbound number of ICMP packets per hour
UDP_OUT	Total outbound number of UDP packets per hour
OTHER_OUT	Total outbound number of other types of packets per hour
ACCEPTED	Total accepted number of all packet types per hour
TCP_ACPT	Total accepted number of TCP packets per hour
IP_ACPT	Total accepted number of IP packets per hour
ICMP_ACPT	Total accepted number of ICMP packets per hour
UDP_ACPT	Total accepted number of UDP packets per hour
OTHER_ACPT	Total accepted number of other types of packets per hour
REJECTED	Total rejected number of all packet types per hour
TCP_RJCT	Total rejected number of TCP packets per hour
IP_RJCT	Total rejected number of IP packets per hour
ICMP_RJCT	Total rejected number of ICMP packets per hour
UDP_RJCT	Total rejected number of UDP packets per hour
OTHER_RJCT	Total rejected number of other types of packets per hour

Table 4 – 4: Processed Data Structure Matrix

## 4.4 DATA COLLECTION & STORAGE

### 4.4.1 Data Collection Process

Data collection was done in four stages. These stages were used to initialise the data structure, baseline both case study network and the simulation network and finally compare the results obtained in both networks. These stages can be listed based on their functionality as follow:

- Stage 1: Initializing raw data and format it in the advanced data structure format as described in Table 4 – 4.
- Stage 2: Create a network baseline from the Case Study network.
- Stage 3: Create a network baseline from the Simulation network.
- Stage 4: Study the affect of applying controlled DoS / DDoS attack pattern against the simulation network.

Table 4 – 5 below shows this staged data collection approach and relate it to the context of the multi phase research process described earlier in Chapter 3.

Data Collection Stage	Data Source		Data Size (in Days)	Number of Cases (24 per day)
	Case Study Network	Simulation Network		
1	X		150	3600
2	X		365	8760
3		X	153	3672
4		X	31	744

Table 4 – 5: Data Collection Staged Approach

### 4.4.2 Data Storage

For simplicity and portability, the data has been stored in a Comma Separated Value (CSV) format as a flat file database. The data can be also stored in a relational database for advanced and complex queries; however, this will add a level of complexity as it will make the proposed model implementation platform highly coupled with the database platform of choice.

#### 4.5 DATA PRE-PROCESSING & PREPRATION PROCESS

The data collection process collected the raw data from native firewall logs format similar to the formats presented in Table 4 – 1 as they have been extracted from the logs. However, to transfer this data to the proposed data structures used in this research as presented in Table 4 – 4, some programming and scripting data pre-processing scripts were needed. The following section discusses the algorithm used to re-construct this data and provide a script implementation tool that can be tailored to different types of firewall logs depending on the format of the extracted firewall logs and the type of the firewall in use.

The collected log files stored by the case study network or by the proposed simulation network are normally exported to a Delimited Separated Value (DSV). The delimiter can be either a comma or a tab. Therefore, the firewall log raw data format needs to be restructured (as presented in Table 4 – 1) to a packet classified format (as presented in Table 4 – 2). The proposed model involves a data preparation class that converts daily raw log file data to the needed hourly format that will be used later by the model forecasting engines. This means one day of firewall logging is converted to 24 segments that correspond to 24 hours within a single day. It follows the following steps:

1. Read the exported firewall log file name for a certain day.
2. Set data fields names for Inbound, Outbound, Accept and Reject for any data set obtained from the firewall logs to a common format since different firewalls use different notations for these fields. Table 4 – 6 shows a conversion example for the Filseclab firewall log notation and the converted common names used.

Filseclab Field Nam	Common Field Name Notation
In	Inbound
Out	Outbound
Pass	Accept
Denied	Reject

Table 4 – 6: Filseclab Log Fields Names Mapping to Common Fields Names

3. Break down each day's data to 24 segments representing each hour of the day in the format presented earlier in Table 4 – 2.

4. Format the data to calculate protocol activity per hour in the format presented earlier in Table 4 – 3.
5. Loop through data to re-read protocol specific activities and packet counts as presented in Table 4 – 4.

The following pseudo code in Table 4 – 7 shows the core elements of steps 3, 4 and 5 of the previous five steps approach used to re-construct the data. The pseudo code below had been implemented in Java and can be found in Appendix – A.

```

START
SET Log File Name
SET index, counter values to zero
SET Arrays (Date_List, Time_List) to null string values
SET Arrays (Inbound, Outbound, Accepted, Rejected, Protocol_Accept,
Protocol_Reject, Protocol_Inbound, Protocol_Outbound) to Zero values
SET (Total_Inbound, Total_Outbound, Total_Accepted, Total_Rejected,
Total_Protocol_Accept, Total_Protocol_Reject, Total_Protocol_Inbound,
Total_Protocol_Outbound) to Zero value
FOR each line in Log File while line data is not equal to null
    READ Log File lines with delimiter
    WRITE date and time to Date_List and Time_List arrays
    IF packet direction is Inbound THEN SET Inbound value and to 1 and SET
    Protocol_Inbound to 1
    END IF
    IF packet direction is Outbound THEN SET Inbound value and to 1 and SET
    Protocol_Outbound to 1
    END IF
    IF packet status is Accepted THEN SET Accepted value and to 1 and SET
    Protocol_Accept to 1
    END IF
    IF packet status is Rejected THEN SET Rejected value and to 1 and SET
    Protocol_Reject to 1
    END IF
    INCREMENT index value by one
END LOOP

```

```

FOR each counter while (counter < index+1)
    READ Arrays index equal to counter
    IF packet direction is Inbound and Accepted THEN
        SET Total_Inbound value to Total_Inbound +1,
        SET Total_Accepted value to Total_Accepted +1
        SET Total_Protocol_Accept to value to Total_Protocol_Accept +1
        SET Total_Protocol_Inbound to value to Total_Protocol_Inbound +1
    END IF
    IF packet direction is Outbound and Accepted THEN
        SET Total_Outbound value to Total_Outbound +1
        SET Total_Accepted value to Total_Accepted +1
        SET Total_Protocol_Accept to value to Total_Protocol_Accept +1
        SET Total_Protocol_Outbound to value to Total_Protocol_Outbound +1
    END IF
    IF packet direction is Inbound and Rejected THEN
        SET Total_Inbound value to Total_Inbound +1
        SET Total_Rejected value to Total_Rejected +1
        SET Total_Protocol_Reject to value to Total_Protocol_Reject +1
        SET Total_Protocol_Inbound to value to Total_Protocol_Inbound +1
    END IF
    IF packet direction is Outbound and Rejected THEN
        SET Total_Outbound value to Total_Outbound +1
        SET Total_Rejected value to Total_Rejected +1
        SET Total_Protocol_Reject to value to Total_Protocol_Reject +1
        SET Total_Protocol_Outbound to value to Total_Protocol_Outbound +1
    END IF
    INCREMENT counter value by one
END LOOP
PRINT (Date, Time, Total_Inbound, Total_Outbound, Total_Accepted, Total_Rejected,
Total_Protocol_Accept, Total_Protocol_Reject, Total_Protocol_Inbound,
Total_Protocol_Outbound)
END

```

Table 4 – 7: Data Preparation Processor Pseudo Code

Table 4 – 8 shows a sample of the collected firewall log data in raw format before applying the data preparation process described in steps one to five.

Date	Time	Origin	Action	Direction	Source	Destination	Protocol
5-Jan-04	11:00:00	192.168.114.24	Reject	inbound	61.172.195.164	192.168.135.118	tcp
5-Jan-04	11:00:02	192.168.114.24	Accept	outbound	192.168.114.24	121.10.12.118	icmp
5-Jan-04	11:00:03	192.168.114.24	Accept	inbound	192.168.114.24	121.10.12.121	icmp
5-Jan-04	11:00:03	192.168.114.24	Reject	inbound	192.168.12.35	62.25.100.32	tcp
5-Jan-04	11:00:03	192.168.114.24	Accept	inbound	192.168.1.50	203.166.25.130	tcp
5-Jan-04	11:00:03	192.168.114.24	Accept	inbound	192.168.114.24	121.10.12.120	icmp
5-Jan-04	11:00:03	192.168.114.24	Reject	inbound	218.57.92.23	192.168.87.210	tcp
5-Jan-04	11:00:03	192.168.114.24	Accept	inbound	192.168.114.24	121.10.12.122	icmp
5-Jan-04	11:00:03	192.168.114.24	Accept	inbound	192.168.114.24	121.10.12.23	icmp
5-Jan-04	11:00:03	192.168.114.4	Accept	inbound	192.168.1.50	17.250.248.32	tcp
5-Jan-04	11:00:03	192.168.114.4	Accept	outbound	192.168.1.41	207.88.7.248	udp
5-Jan-04	11:00:03	192.168.114.4	Accept	outbound	121.11.14.98	204.179.240.7	udp
5-Jan-04	11:00:03	192.168.114.4	Accept	outbound	192.168.114.24	121.10.12.27	icmp
5-Jan-04	11:00:03	192.168.114.24	Accept	inbound	192.168.114.24	121.10.12.29	icmp
5-Jan-04	11:00:03	192.168.114.34	Accept	inbound	192.168.114.24	121.11.14.113	icmp
5-Jan-04	11:00:03	192.168.114.34	Accept	inbound	192.168.114.24	121.11.14.98	icmp
5-Jan-04	11:00:03	192.168.114.4	Accept	inbound	192.168.1.50	203.36.194.181	tcp

Table 4 – 8: Raw Firewall Log Data Format

After applying the data preparation steps, the data is now presented in a 24 hour data set which is classified by packet type, direction and status (i.e. accepted vs. rejected).

Table 4 – 9 shows a sample of the new data format for TCP packets.

DAY	TIME	INBOUND	TCP_IN	IP_IN	ICMP_IN	UDP_IN	OTHER_IN	OUTBOUND
1-Jan-04	1:00	23433	10522	0	817	12065	29	474
1-Jan-04	2:00	23595	9830	1	807	12929	28	544
1-Jan-04	3:00	149208	136744	0	805	11630	29	460
1-Jan-04	4:00	21483	9194	0	804	11457	28	424
1-Jan-04	5:00	27899	16394	0	804	10674	27	325
1-Jan-04	6:00	51371	40073	0	804	10465	29	439
1-Jan-04	7:00	18896	7503	0	800	10564	29	498
1-Jan-04	8:00	19648	8011	0	806	10803	28	595
1-Jan-04	9:00	90797	78269	0	849	11651	28	506
1-Jan-04	10:00	25002	11386	0	883	12705	28	505
1-Jan-04	11:00	39215	25656	0	856	12674	29	460
1-Jan-04	12:00	97676	84421	0	869	12358	28	599
1-Jan-04	13:00	82523	68258	0	876	13358	31	503
1-Jan-04	14:00	23228	9080	0	889	13229	30	476
1-Jan-04	15:00	24725	9571	0	865	14261	28	491
1-Jan-04	16:00	24441	9770	0	869	13772	30	545
1-Jan-04	17:00	25534	10488	1	834	14182	29	539
1-Jan-04	18:00	27657	14082	0	853	12694	28	413
1-Jan-04	19:00	54652	40672	0	856	13090	34	466
1-Jan-04	20:00	22730	9074	0	848	12779	29	242
1-Jan-04	21:00	22820	8958	3	828	13001	30	248
1-Jan-04	22:00	22056	9156	0	785	12088	27	382
1-Jan-04	23:00	22229	9237	0	794	12169	29	423
1-Jan-04	24:00:00	19992	7551	0	864	11549	28	531

Table 4 – 9: TCP Packets Classified by Direction and Status over a Period of 24 Hours

The results obtained in Table 4 – 9 will be the input data for the proposed model of detection. This formatted data using the developed tools is justified given the facts described in the beginning of this chapter about the native layout of the firewall logs and how they are presented in different formats by different firewall implementations (please refer to Section 4.2.1, Table 4 – 1 and Section 4.3.1 of this chapter).

#### **4.6 CHAPTER FINDINGS**

The investigation within the firewall logs had shown that at least four main protocols need to be monitored (as identified from the literature review of different DoS / DDoS attacking methods) these are:

- TCP protocol that controls the transmission method of the packet data.
- IP that defines the source and target address of every packet.
- ICMP protocol that is used to diagnose and exchange messages between hosts.
- UDP protocol which is similar to TCP but much faster due to the fact that the transmission with UDP is not guaranteed to be delivered in comparison to TCP.

These firewall logs contain full details of the above protocols activity and many other protocols that are travelling across the firewall (e.g. DNS, SMTP, SNMP), the packet direction (inbound or outbound) and the decision made by the firewall rules to either accept or reject these packets.

This chapter has shown that the information obtained from the literature identifying various attacking techniques for DoS / DDoS can be observed by simply monitoring rejected protocols packets while passing through the firewall.

Therefore, this chapter provides a visibility of packet type, status and direction to the proposed DoS / DDoS prevention model in order to detect abnormal activities of these protocols.

## **4.7 CHAPTER CONCLUSION**

The chapter builds on this research by describing the process used to handle selected data patterns of TCP / IP suite of protocols from the firewall logs (as identified in Chapter 2). These protocols represent the way traffic can pass through the firewall and carry DoS / DDoS attacks. Therefore, this chapter has collected and restructured the activities of these protocols specially TCP, IP, ICMP and UDP which in turn present a picture of how the firewall handles the traffic before, during and after an attack has been launched. The firewall log data presents a reach environment for researching using a simple approach of data collection and pre-processing as identified by the research phases from Chapter 3.

In addition, the chapter introduced a pseudo code for data preparation to tailor the data toward the simplistic approach of the research model which is based on the collection of rejected packets statistics and the ability of these to present an attack at the early stages of existence within the network. The data preparation pseudo code programming implementation can be found in Appendix – A. The full firewall log data collection and the transformation from raw logs to re-structured logs (as proposed in Table 4 – 4) had been completed using these tools and can be found in Appendix – B of this thesis.

The next chapter will discuss the proposed DoS Tracking and Detection Model (DoSTDM) design using statistical forecasting and neural network modelling analysis with an implementation guideline of the DoSTDM model.

# CHAPTER 5

## THE DoSTDM DESIGN STRATEGY

This chapter presents the design of the new DoS Tracking and Detection model (DoSTDM). This new model has its foundation in findings from previous chapters and especially from Chapter 4 whereby the data structure of the firewall logs and the protocols that governed the traffic before had been constructed. The TCP / IP suite of protocols (TCP, IP, ICMP and UDP) activities collected in Chapter 4 data pre-processing process helped in generating and shaping the design strategy for the new conceptual model. The design incorporates both neural network and statistical analysis and forecasting methods for tracking DoS / DDoS attack anomalies within the firewall logs of TCP / IP protocol suite activities.

Chapter five is divided into five main sections:

- Section one introduces the DoSTDM model design strategy.
- Section two examines the theoretical and practical aspects of the model design and outlines the process used in the development driven by the pre-defined data structures as obtained from Chapter 4.
- Section three discusses model implementation guidelines in existing computerised network (with existing firewall).
- Section four examines the model characteristics and limitations.
- Section five provides a conclusion for this chapter.

## 5.1 INTRODUCTION

This research is not about a specific product or commercial implementation, it is about simplifying the way computer networks can defend themselves against flooding DoS and DDoS flooding attacks using existing tools and infrastructures firewall data.

The existing tools that implement IDS and IPS systems (like Snort and other proprietary commercial protection applications) suffer from certain limitations that inhabit their existence in every network, these are:

1. High costs associated with design, implementation, training and maintenance.
2. The majority of these systems follow the misuse approach to identify known attacks with high number of rule sets which in turn add more management overhead to the system in order to manage the rules.
3. Low levels of accuracy with high levels of false alerts due to the fact that these systems are rule based and the accuracy level is highly coupled with the way the rules are configured.
4. The overall performance of these systems is affected by the number of rules that are needed to be managed and created over time.
5. The majority of these implementations are vendor proprietary in both hardware and software implementations. Therefore, it is very hard to identify the methodology used for detecting unknown attacks.

Therefore, the selection criteria of a new model are based on:

1. *Simplicity*: Simplifying the detection process of flooding DoS and DDoS attacks can be achieved by using the firewall only without having to use a complex IDS and IPS solution as both rely on the firewall to filter the traffic from any attack and not necessarily flooding attacks. This is following the Keep it Simple and Stupid (KISS) approach of keeping things simple and uses the simplest way to achieve the required results.
2. *Portability*: The model can be carried from one network to another as it uses standard TCP / IP protocol traffic estimators and therefore, it can be implemented in many ways and shapes of development tools.

3. *Efficiency*: Low resource usage compared to complex IDS / IPS resource aggressive systems that require high processing and memory capabilities. This will improve the new model overall performance.
4. *Accuracy*: The threshold estimated from the network traffic history determines the model level of accuracy.
5. *Cost of Operation and Management*: No need for new skills as existing skills of managing and operating the firewall should be sufficient to operate and maintain the model as it integrate with the firewall logs to track and alert when DoS / DDoS flooding are detected.

Therefore, the proposed model is driven in design by existing limitations of every detection technique. The new model is a hybrid structured model that uses both statistical and neural network detection algorithms to:

1. Close the gap of anomaly based detection by dynamically defining the anomaly detection threshold.
2. Avoid pre-defined attacking signatures by creating a historical baseline of the network for future estimation of traffic without being under attack. This can later be used to isolate DoS / DDoS flooding attack patterns.
3. Implement source based detection techniques by monitoring existing traffic passing through the firewall both inbound and outbound directions.

This chapter describes in detail how the DoSTDM hybrid-design model detects and tracks flooding attacks caused by DoS and DDoS threats.

## **5.2 DoSTDM DESIGN PROCESS**

### **5.2.1 Model Design - Theoretical Aspects**

DoSTDM uses anomaly detection to track past DoS / DDoS attacks and to detect any new attacks accordingly. The two main aspects of this method are the ability to detect new attacks and a reduction of the high level of false alerts that might be generated if the detection threshold boundaries were not accurate for the network.

Therefore, for successful detection using this method an enhanced threshold setup needs to be in place based on dynamic baseline rebuilding techniques of past firewall data analysis. DoSTDM uses other fields in Computer Science and Artificial Intelligent to build a picture of the network before, during and after a flooding DoS / DDoS attack.

The computer immunology principle is used in the same way human immunology applies to humans. This means any foreign activities within the network where DoSTDM is applied will be detected and dealt with in the same way when germs enter a human body and anti bodies are applied as a means of protection.

The artificial intelligence part uses a neural network to compare past experience within the network with new traffic activities in the same network. The main aim is to maximise the protection by detecting new network traffic anomalies that present a flooding behaviour of a network under an attack.

### **5.2.2 Model Design - Business & Practical Aspects**

For any business or government sector that uses computerised networks attached to the internet, the network up time is critical. These networks should include what's called Business Continuity Plans (BCP). The BCP defines the way a business should react to any loss of resources. DoS and DDoS attacks are centred on making the network unusable for a certain amount of time and are aimed at eliminating crucial computing resources and services. The cost of maintaining a BCP can be tremendous if it is to create a redundant network in case the main network is unavailable. Therefore, DoS / DDoS flooding attacks can't be ignored by such a business.

### **5.2.3 Model Design Process Overview**

DoSTDM is a software driven model independent from any hardware, therefore, it follows software engineering lifecycle design principles. The main-stream software engineering lifecycle models include the Waterfall model, Prototyping model, Evolutionary model and Spiral model (Mall 2000). These models have the same process goal, which is to develop a model, process and/or software products, however, the methods are not the same.

Boehm (1986) was the first to introduce the Spiral model using four phases to complete the modelling process. The spiral model is considered to be the most flexible development approach as it incorporates aspects of the Waterfall model, Evolutionary model and the Prototype cyclic development model with the ability to assess the risk that might appear during the development of the model (Guimarães & Vilela 2005).

The Spiral model is also called the Meta Model as it encompasses all of the other three models in one using the prototype model as a risk reduction mechanism (Mall 2000). Mall (2000) and Guimarães & Vilela (2005) summarise these four phases as follows:

**Stage 1:** Identify objectives, limitations and alternatives.

**Stage 2:** Evaluate alternatives and resolve any risks.

**Stage 3:** Development of next level model prototype including design, specifications and verifications.

**Stage 4:** Evaluate the prototype in this cycle and plan for the next phase cycle.

The spiral design process model has been staged in four phases to complete DoSTDM design and development from start to end. These phases are: (1) Planning, (2) Analysis, (3) Engineering and (4) Evaluation (as shown below in Figure 5 – 1). These phases are the same original spiral stages with customized names for DoSTDM design process referencing.

DoSTDM spiral model process design provides:

- Early detection of errors and limitations by iterative prototype testing.
- The ability to modify and expand the model while developing it.
- Risk assessment visibility to identify and mitigate any limitations within the produced model.
- No limitations on the number of iterations that can be applied to achieve a robust model.

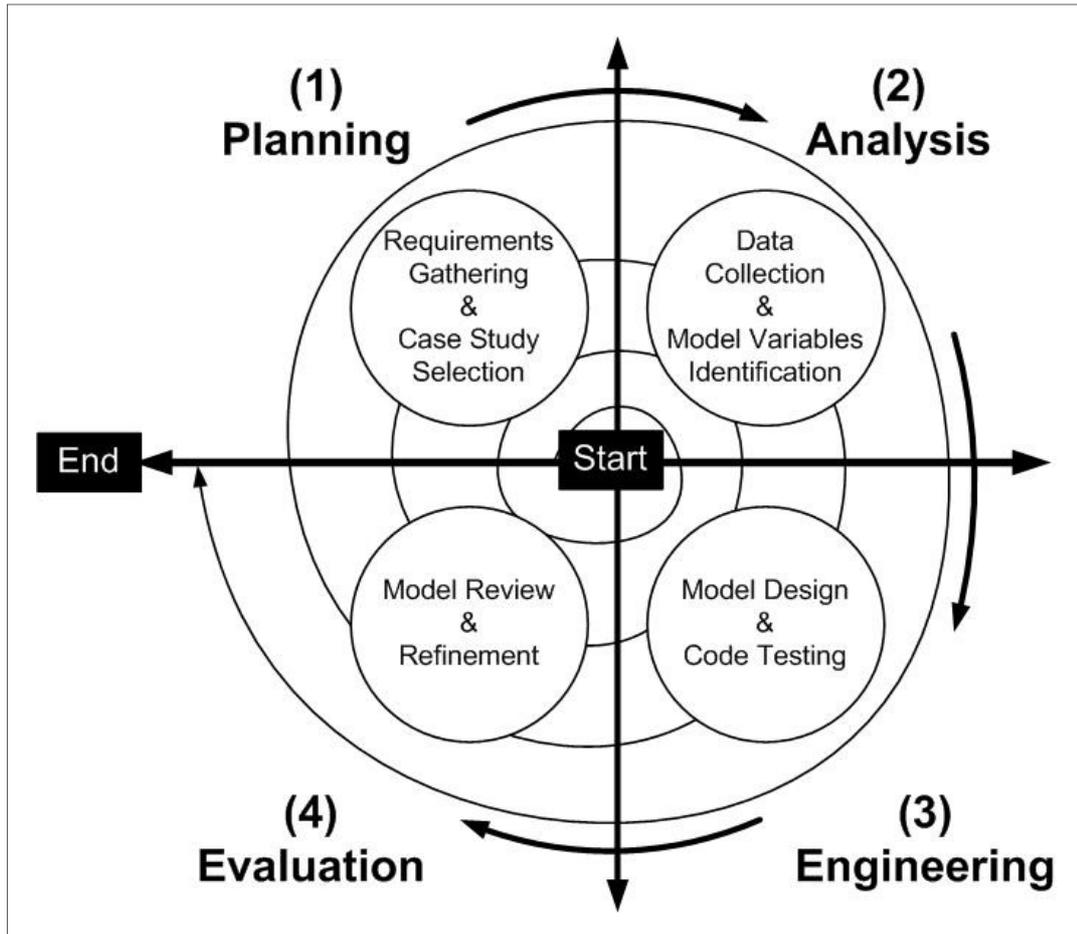


Figure 5 – 1: DoSTDM Model Process Design Using Spiral Development Process

Figure 5 – 1 illustrates the spiral design process of DoSTDM and the development cycle over the four phases of planning, analysis, engineering and evaluation. These phases are discussed in detail in the following sections of this chapter.

#### 5.2.4 DoSTDM Design Planning Phase

In this phase the design requirements to develop the DoSTDM model were locked down. These requirements had been narrowed down to the existence of a firewall (with traffic logging capabilities) that use the TCP / IP network protocol to communicate within itself internally and to the internet externally. This phase dealt with the raw firewall data described in Chapter 4 to identify the basic data structures within the case study network.

#### 5.2.5 DoSTDM Design Analysis Phase

The design phase analysed the provided data obtained from the case study firewall logs. The data set included TCP / IP sub-protocols that were used in the case study

network to communicate over the internet such as TCP, IP, ICMP and UDP. This phase generated the modified data structures described earlier in Chapter 4 to merge the type of protocol with its direction based on the classification of firewall reaction of accepted or rejected. The output data was used to construct the simulation network.

### **5.2.6 DoSTDM Design Engineering Phase**

The engineering phase of this design process uses the outcome of the analysis phase to craft the first prototype of the model using statistical modelling at the first iteration. The statistical modelling uses two statistical forecasting techniques, the first is Holt-Winter method and the second is a Multiple Linear Regression method.

These two quantitative statistical forecasting methods were found to be more suitable than some other methods for this time-series data (i.e. Moving Averages, Simple Regression, etc.) (Makridakis et al. 1998). The main reason behind selecting these two methods was that both methods can cover for firewall log data trend and seasonality components at the same time. The engineering phase also designed the simulated network that will be used to test the DoSTDM model in an experimental controlled environment. The second iteration of the spiral development process incorporates a neural network algorithm using the Multi Layer Perceptron (MLP) algorithm.

#### **5.2.6.1 Holt-Winter Forecasting Method**

Holt-Winter method of forecasting uses an exponential smoothing forecasting technique that can allow discovery of the underlying pattern within the time series data while eliminating the effects of any trend and seasonal components. A three months data sample obtained from the case study data collection (as shown in Figure 5 – 2) illustrated both trend and seasonal components within this data set. The trend component marked by a yellow line passing through the centre of the data shows an upward long-term trend activity. The seasonal pattern was also seen repeating itself with the start of every week. Therefore, to see the seasonal factor better, it is necessary to zoom inside the data by weeks rather than by months. Figure 5 – 3 shows the seasonal part of the data pattern activity (marked by a yellow circle). The seasonality is occurring in a multiplicative way every day of the week (as working hours are 8:00AM to 5:00PM except for weekends).

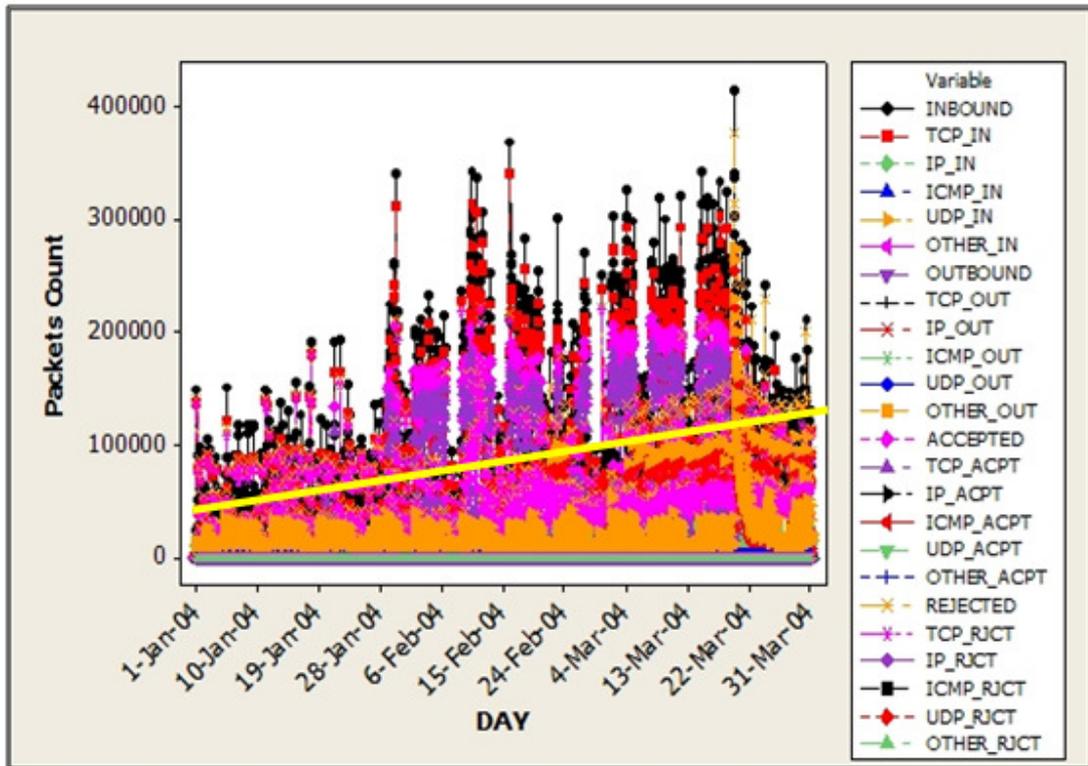


Figure 5 – 2: Packets Trend Component in Case Study Firewall Logs

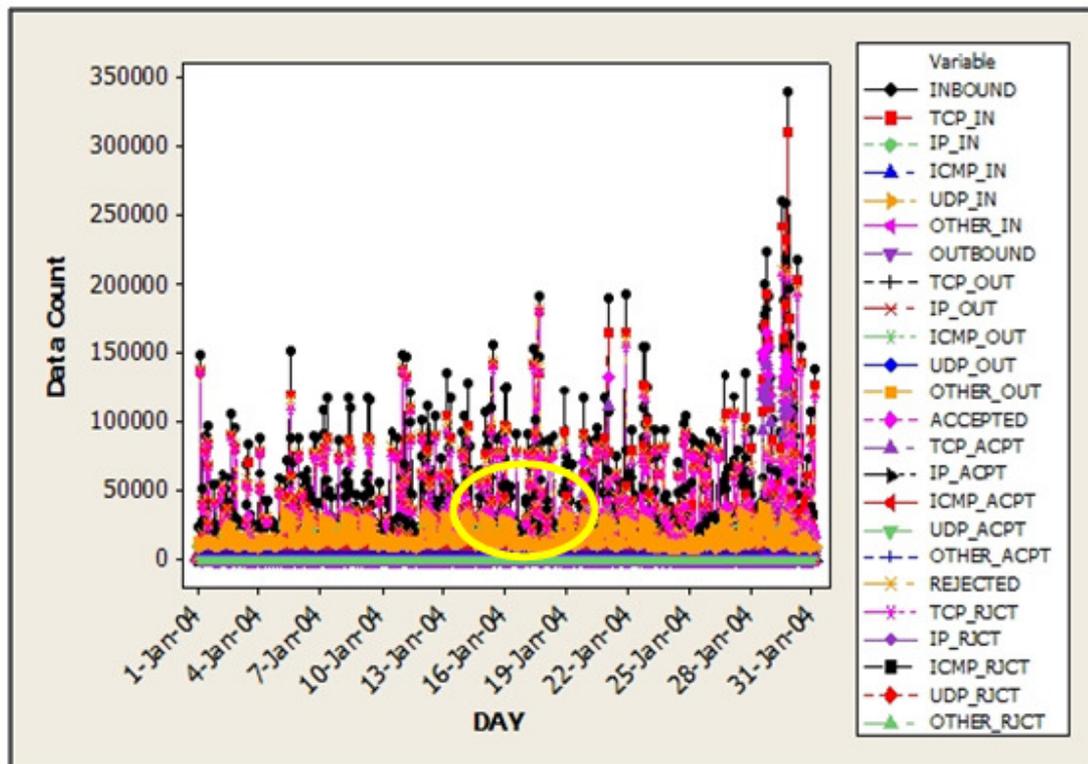


Figure 5 – 3: Packets Seasonality Component in Case Study Firewall Logs

Holt-Winter is an exponential smoothing forecasting method that caters for both trend and seasonality within the data. A special Holt-winter multiplicative method has been used to decompose the data and smooth the forecasts eliminating any effect of seasonality and trend. This was achieved by using the Holt-Winter set of multiplicative forecasting equations in Table 5 – 1 in order to forecast future values of rejected packets:

Linear Trend Equation:	$Y(t) = a + bt$
Level Adjustment Equation:	$L_t = \alpha \left( \frac{Y_t}{S_{t-s}} \right) + (1 - \alpha)(L_{t-1} + b_{t-1})$
Trend Adjustment Equation:	$b_t = \gamma(L_t - L_{t-1}) + (1 - \gamma)b_{t-1}$
Seasonal Adjustment Equation:	$S_t = \delta \left( \frac{Y_t}{L_t} \right) + (1 - \delta)S_{t-s}$
Forecast Equation:	$F_{t+m} = (L_t + b_t m)S_{t-s+m}$
Where:	
• $Y(t)$	Represents the value of the trend line at time “t”
• $a, b$	Represents the linear trend coefficients.
• $t$	Represents the time value.
• $L_t$	Represents the level component of the data series at time “t”.
• $b_t$	Represents the trend component forecasted value at time “t”.
• $S_t$	Represents the seasonal component value at time “t”.
• $m$	Represents the forecasted period range in time.
• $s$	The seasonality period length (i.e. 24, corresponding to 24 hours / day)
• $F_{t+m}$	Represents the forecasted value at time “t+m”
• $Alpha$ “ $\alpha$ ”	Represents the smoothing factor for the data level component.
• $Delta$ “ $\delta$ ”	Represents the smoothing factor for the data seasonal component.
• $Gamma$ “ $\gamma$ ”	Represents the smoothing factor for the data trend component.

Table 5 – 1: Holt-Winter Multiplicative Forecasting Equations

Source: (Makridakis et al., 1998, 165)

Notes:

- 1) The trend coefficients can be calculated as:

$$a = \frac{\text{Total counts of packets for the training period}}{\text{Total number of readings in this training period}}$$

$$b = \frac{\text{Total of (packets count} \times \text{time period)}}{\text{Total of ((time point)}^2)}$$

- 2) The smoothing factors “ $\alpha$ ”, “ $\delta$ ” and “ $\gamma$ ” need to be selected in the range from “0” to “1” by minimising the forecasting error by iterating through different values of these parameters then selecting the one with the lowest error level of RMSE (Root Mean Square Error).

Please refer to Appendix – A and Appendix – C for code listings and Java classes designed to implement this method in the context of this research.

#### **5.2.6.2 Multiple Linear Regression Forecasting Method**

Multiple Linear Regression smooths the data by representing the forecasting outcome (in this case the rejected level of packets) as a formula of the most contributing variables.

The aim from this statistical model is to see if the rejected packets can actually correspond to the results found by Holt-Winter forecasts in order to provide a reliable conclusion from the results of both methods. Linear multiple regression can express the relation between different factors based on the causal relationship and the correlation between these factors.

The difference between real data and regression forecasted data (i.e. residuals) should follow a normal distribution bell shape with mean value equal to zero. The basic multiple regression equation described by Makridakis et al. (1998) in Table 5 – 2.

Multiple Linear Regression Equation:  $Y_i = \beta_0 + \beta_1 X_{1,i} + \dots + \beta_k X_{k,i} + \varepsilon_i$

Where:

- $[Y_i]$  The regression value at time “ $i$ ”.
- $[X_{1,i}, \dots, X_{k,i}]$  The regression explanatory factors value at time “ $i$ ” (i.e. the packet count of TCP\_IN, IP\_RJCT, ....etc at specific hour)
- $[\beta_0, \beta_1, \dots, \beta_k]$  Fixed regression parameters for each explanatory factor
- $\varepsilon_i$  Random variable with zero mean distribution

Table 5 – 2: Multiple Linear Regression Forecasting Equations

Source: (Makridakis et al., 1998, 248)

However, Table 5 – 2 equation works fine if the time series data doesn’t have any seasonal component included. Therefore, when dealing with seasonality this problem can be fixed by adding seasonal dummy variables to include the seasonality component within the data in the final regression model. Because the seasonal period is fixed (i.e. the re-occurrence of the season is every 24 hours), the seasonal intervals can be added as 23 dummy variables as shown in Table 5 – 3.

The reason for adding 23 dummy variables instead of 24 is to address a regression computational problem called “Multicollinearity” (Makridakis et al. 1998). Multicollinearity occurs when the correlation between the regression factors is very high and it can be a real issue when looking at all of these factors’ error measurements as a whole. These 23 dummy variables are now considered as regression factors within the regression equation as shown below:

$$Y_i = \beta_0 + \beta_1 X_{1i} + \lambda_1 D_{1i} + \dots + \beta_k X_{ki} + \lambda_k D_{ki}$$

Where “ $\lambda$ ” represents a fixed regression parameter for each dummy factor (i.e.  $D_1, D_2, \dots, D_{23}$ ) used in the regression equation, however, the final regression model might include all or some of these factors. Multi Linear Regression can be calculated and generates a static equation for a specific period within the time series as in the example below calculated using the MINITAB statistical package to represent the

number of rejected packets as an equation that consists of inbound and outbound protocols :

$$\text{REJECTED} = 16057 + (0.289) \text{TCP\_IN} - (139) \text{IP\_IN} - (6.93) \text{ICMP\_IN} + (0.969) \text{UDP\_IN} - (358) \text{OTHER\_IN} - (27.2) \text{TCP\_OUT} - (2711) \text{ICMP\_OUT} + (8.98) \text{UDP\_OUT} - (804) \text{OTHER\_OUT}$$

Sample Regression Equation Using the Case Study Data)

Table 5 – 3 shos the distribution of dummy variables across 24 hours to be used with the final multiple regression equation.

Hourly Time Interval	Dummy Variables Value																						
	D 1	D 2	D 3	D 4	D 5	D 6	D 7	D 8	D 9	D 10	D 11	D 12	D 13	D 14	D 15	D 16	D 17	D 18	D 19	D 20	D 21	D 22	D 23
1 - 2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2 - 3	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3 - 4	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4 - 5	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5 - 6	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6 - 7	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7 - 8	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8 - 9	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9 - 10	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10 - 11	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
11 - 12	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
12 - 13	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
13 - 14	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
14 - 15	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
15 - 16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
16 - 17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
17 - 18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
18 - 19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
19 - 20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
20 - 21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
21 - 22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
22 - 23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
23 - 24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
24 - 1	All dummy variables values are set to zero to avoid Multicollinearity between the first interval and the last interval (i.e. first hour and last hour in the same day)																						

Table 5 – 3: Regression Model Dummy Variables

The regression method uses the predictor significance value ( P-Value) to construct the regression equation. The “P” value provides the degree of significance about the relation between the predicting factor (e.g. TCP inbound TCP\_IN activities) and expected response of the regression model (i.e. number of REJECTED packets per hour) with a pre set confidence (e.g. 95%). Therefore, for any predictor to remain in the regression model the “P” value needs to be less than 5% (i.e. 100% -95%). However, DoSTDM model uses a real time Java class to produce the regression calculation on the fly (please refer to Appendix – A and Appendix – C for code listings and compiled Java classes).

### 5.2.6.3 Neural Network Modeling

Neural networks are very useful and can be applied to solve multiple types of problems. Swingler (1996) identified the differences between the neural network approach and the traditional programming approach as shown in Table 5 – 4.

Programming Approach	Neural Computing Approach
<ul style="list-style-type: none"> <li>• Follows rules</li> <li>• Solution formally specifiable</li> <li>• Cannot generalise</li> <li>• Not error tolerant</li> </ul>	<ul style="list-style-type: none"> <li>• Learns from data</li> <li>• Rules are not visible</li> <li>• Able to generalise</li> <li>• Copes with noise</li> </ul>

Table 5 – 4: Programming Approach vs. Neural Computing Approach

Source: (Swingler, 1996, 9)

The DoSTDM design uses a neural network Multi Layer Perceptron (MLP) engine to predict the rejected packets level in parallel with the statistical Holt-Winter and Multiple Linear Regression methods. The MLP neural network is a type of Back Propagation Neural Networks (BPNN) algorithm. BPNN models are suitable for data mining pattern recognition and provide good performance while handling complex patterns (Suh 2012).

The structure of an MLP neural network provides a non-linear relation between the network real value input and the expected real value output using a relatively simple optimization backpropagation training method (Hand et al. 2001).

The MLP neural network can be constructed from an interconnected set of nodes with multiple layers as shown in Figure 5 – 4 as identified by Kennedy et al. (1997). The first layer is the input layer, the hidden layer where all inputs are transformed to the next layer based on their weights and the final layer is the output layer.

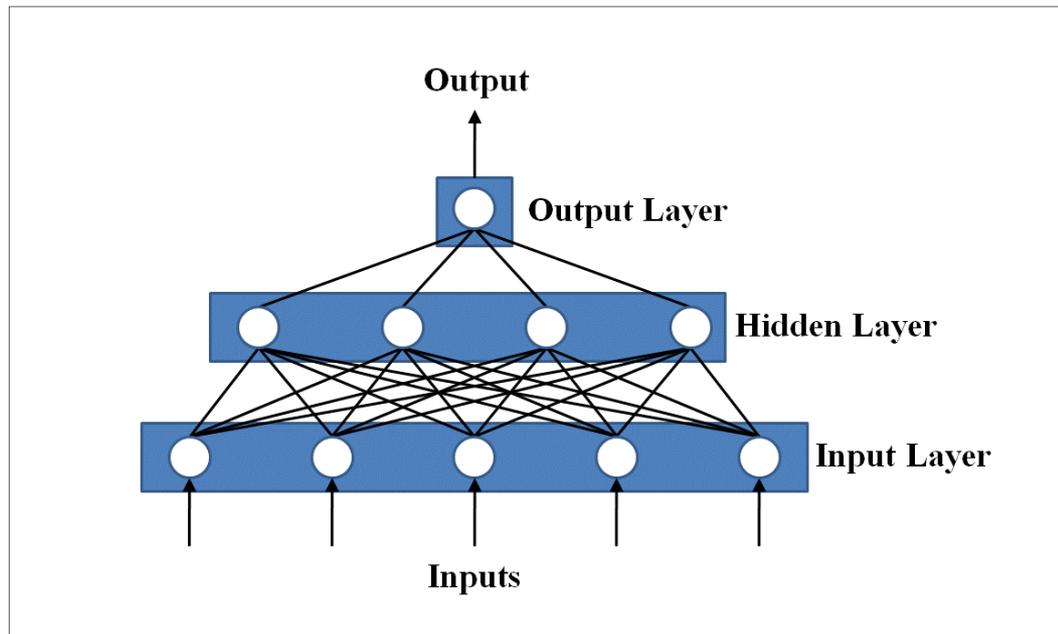


Figure 5 – 4: Neural Network Architecture Example with One Hidden Layer  
Source: (Kennedy et al., 1997, 10-23)

The MLP algorithm utilises the neural network inputs to calculate the output using a hyperbolic tangent transformation function for all hidden neurons (nodes). Therefore, the output neuron “y” is the sum of all the connection weights between the hidden neurons and the output neuron inside the network (Kennedy et al. 1997).

For this research, MLP network inputs are the packets statics per a time unit (hour / day / week) for every type of packet that enters or exits the computer network firewall (e.g. TCP, IP, ICMP, UDP, etc.) regardless as to whether it has an accepted or rejected status. The MLP neural network output represents the total number of rejected packets forecast for the next time period (hour / day / week).

Kennedy et al. (1997) defined the hyperbolic tangent transformation function to obtain the hidden neurons (nodes) output as shown in Table 5 – 5.

Predicted Output:	$y = \frac{(1 - e^{-net})}{(1 + e^{-net})}$
Neural Network <i>net</i> :	$y(x) = w_0 + \sum_i w_i x_i$
Where:	
	<ul style="list-style-type: none"> <li>• <i>net</i> is the output node (<i>y</i>) as a function of input nodes (<i>x</i>) that can be obtained from the summation of the weights multiplied by the inputs.</li> <li>• <math>w_0</math> is the bias of the node</li> <li>• <math>w_i</math> is the connection weight from hidden neuron node <math>i^{th}</math> with input value <math>x_i</math></li> </ul>

Table 5 – 5: Neural Network for MLP

Source: (Kennedy et al., 1997, 10-24)

Swingler (1996) defined the learning steps for MLP algorithm with multiple recurrent backpropagation and is quoted as follows:

1. *Build a network with a context layer equal in size to the hidden layer for each sequence to be learned.*
2. *Set the network weights to low random values.*
3. *Assign a context layer to each sequence.*
4. *Reset all the context layer values to indicate the start of a sequence*
5. *Choose a single sequence at random from those chosen in step 2.*
6. *Present the next element in the chosen sequence along with the context layer values associated with that sequence to the network as input, along with the target category or required values as output.*
7. *Cycle the network and adjust the weights using the standard backpropagation method.*
8. *Copy the contents of the hidden layer back into the context layer used in step 4.*
9. *If the end of a sequence is reached, reset the associated context layer and return to the start of the sequence.*
10. *Repeat steps 4 to 9 until the error is sufficiently low or the network settles.*

The error minimisation process in step 10 can be obtained by back-propagating the difference between real training output values and predicted output values of the

MLP using Mean Square Error (MSE) of the overall training set as shown in Table 5 – 6. However, due to the use of random weights at the beginning of the learning algorithm as described by Swingler (1996), the error values obtained every time the neural network is re-trained can vary due to the initial weights randomisation.

Error Calculation: 
$$E = \sum_{i=1}^{N_{train}} \left\{ \frac{1}{2} \sum_{j=1}^J (d_{ij} - y_{ij})^2 \right\}$$

Where:

- $E$  is the total root mean square error of the network at training pattern “ $i$ ”
- $i$  is the index of the training pattern input value
- $N_{train}$  is the total number of training patterns with total number of outputs “ $J$ ”
- $j$  is the index of the training pattern output value
- $J$  is the training patterns outputs
- $d_{ij}$  is the desired value of output “ $j$ ” at training pattern “ $i$ ”
- $y_{ij}$  is the actual output of the training pattern

Table 5 – 6: Neural Network for MLP  
Source: (Kennedy et al., 1997, 10-25)

Figure 5 – 5 shows the flow of data within MLP with recursive backpropagation processing of the weights that is used to find the optimal value of the connection weights between MLP input and output nodes (Ordonez-Cardenas & Romero-Troncoso 2008). The input scaling process is a pre-processing step to prepare the inputs in order to match the hyperbolic tangent range.

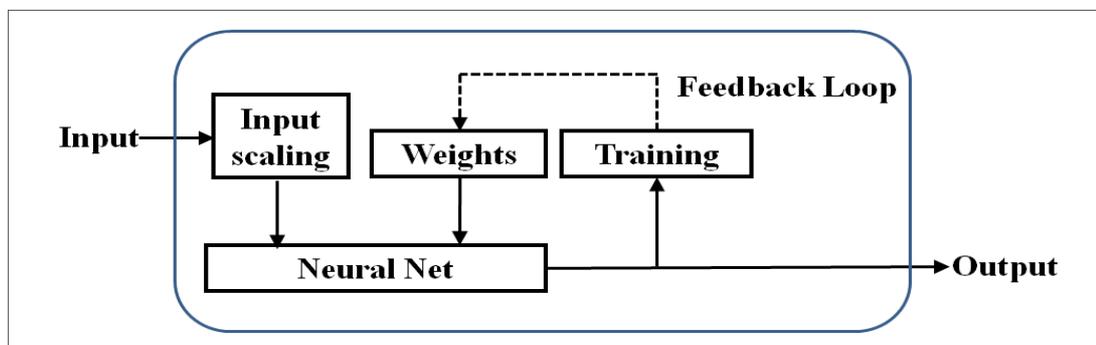


Figure 5 – 5: MLP System Floor Plan  
Source: (Ordoez-Cardenas & Romero-Troncoso, 2008, 334)

This research MLP design uses all inputs collected from the firewall logs as classified earlier in Chapter 4 Table 4 – 4 (DoSTDM Advanced Data Structure Matrix) to construct the MLP neural network. However, with the type of data used in this research, it has been found that the output neural node value can be nearly the same for all training patterns as the MSE (Mean Square Error) and connection weights were minimum which leads the implemented programming interface to round up the results due to the hyperbolic tangent. Therefore, an additional step was added to expedite model performance by normalising the inputs and then De-Normalise the output. This means the values entering the neural network are transformed to values between one and zero and then the output is retransformed by reversing the process of the normalisation. The normalisation and de-normalisation layers can be seen in Figure 5 – 6 which shows the complete neural network designed for DoSTDM.

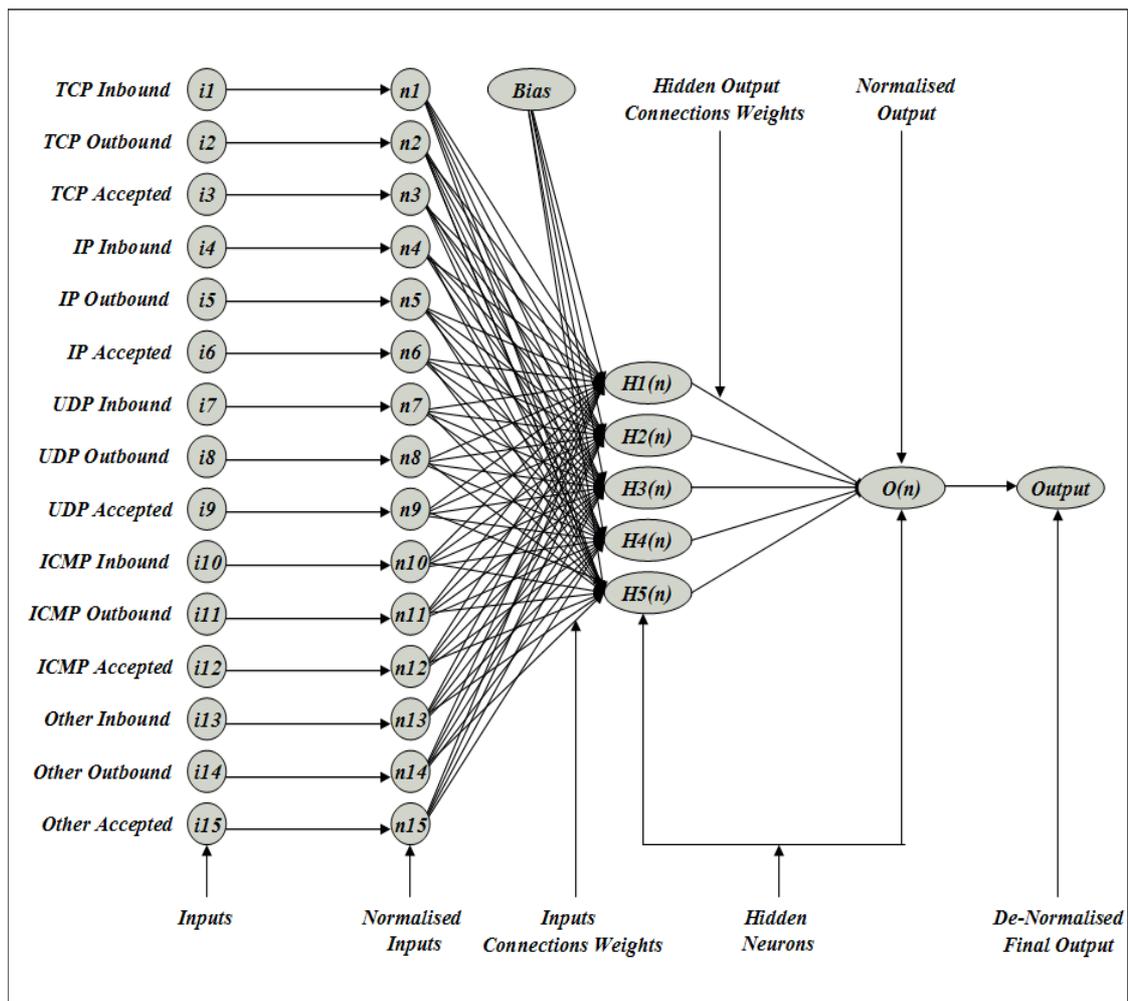


Figure 5 – 6: DoSTDM MLP Neural Network Structure

The Java class's code designed to implement MLP algorithm and the normalization and de-normalisation functions embedded can be seen in Appendix – A code listings and are provided in a compiled format as part of Appendix – C.

#### **5.2.6.4 Baseline Threshold Definition Method**

The baseline detection threshold represents the network's natural growth in rejected packets and anything higher than this threshold can be considered as an attack with the intention to flood the network with traffic that can lead to a DoS situation. Therefore, comparing the rejected packets forecast obtained from statistical forecasting methods and MLP neural network output with real rejected data for a specific day can show if the difference is lower or higher than the dynamically calculated threshold.

#### **5.2.7 DoSTDM Design Evaluation Phase**

The evaluation phase compares the results of different forecasting techniques to determine the most important set of data that can be used to produce the baseline for the model. In addition, it re-iterates within the process to eliminate any detected errors within the prototype model and re-feed these to the engineering phase to enhance the model build. Special attention been paid to implemented code debugging to enhance the final model by minimising false alerts within DoSTDM output.

#### **5.2.8 DoSTDM Model Final Design Structure**

The model build incorporated the existing technologies in use by many organisations and networks; these are the existing firewalls and standard protocols within the TCP / IP suite. It merges these technologies with the ability of system administrators to script data extraction and handle data manipulation to produce the foundation baseline of firewall activities from firewall logs of past days, months and maybe years. Therefore, the model's final design includes the following components:

1. A database system or file to store the log extracted data; this will be used as a historical data source for the model.
2. A pro-active programmable routine to determine the real time threshold of all rejected TCP / IP packets passing through the firewall assuming all rejected and dropped packets are the same under a DoS / DDoS flooding attack.

3. A forecasting engine to predict future rejected packet activities based on past and current activities obtained from the historical database and current activities of the firewall.
4. A comparison real time routine that can find anomalies within the firewall activities and identify up-normal activities.
5. An alerting system to produce alerts to system administrators and / or handle certain functions to disable certain ports or shutdown certain functions on the firewall in order to stop the DoS / DDoS flooding.
6. A real time data collector to continue updating the historical database and therefore force a dynamic adjustment of the rejected packets threshold.

The above components have been merged in real time loop design as shown in Figure 5 – 7 to complete the final design structure for a dynamic DoSTDM implementation.

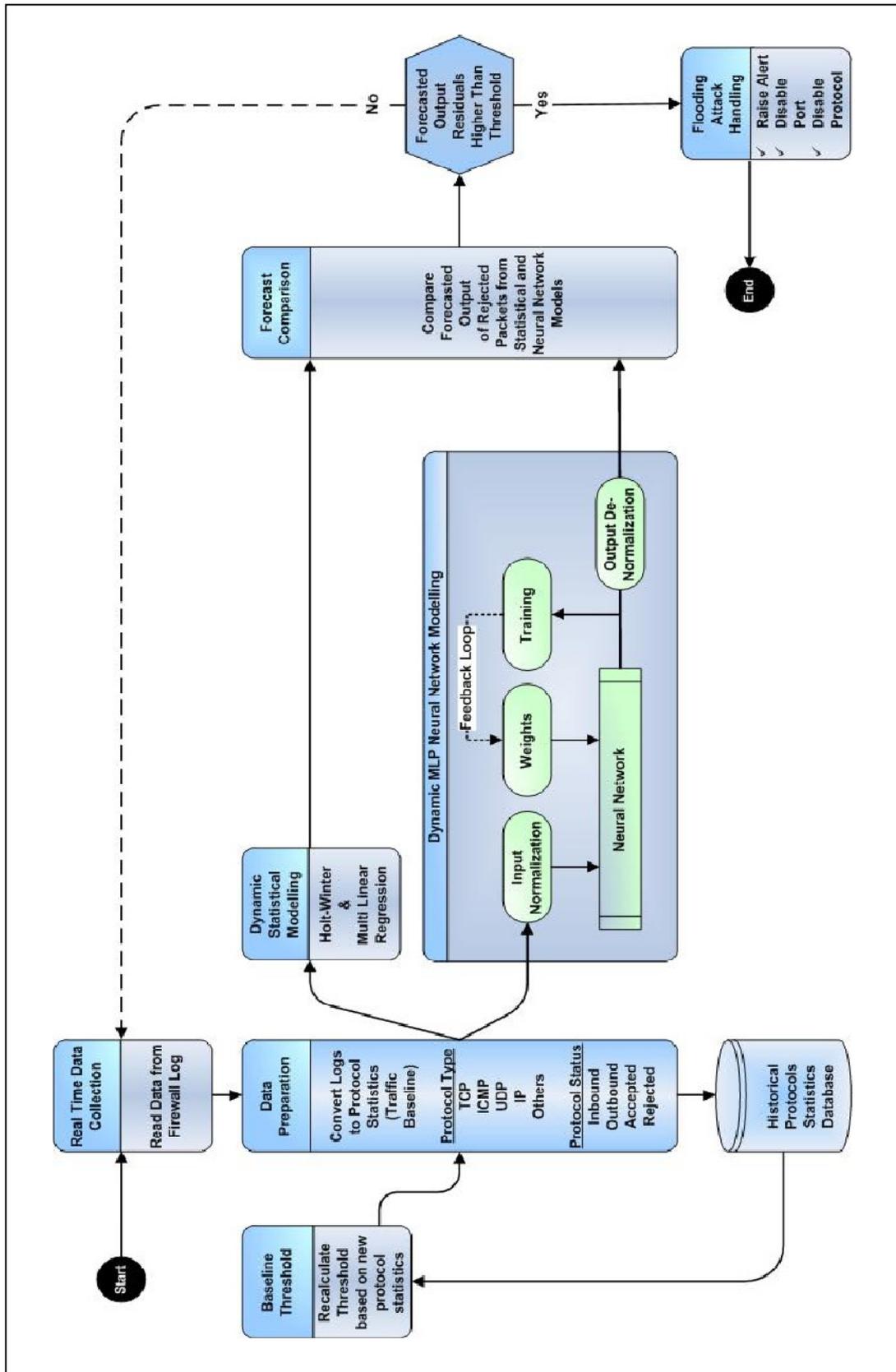


Figure 5 – 7: DoSTDM Final Design Structure

Figure 5 – 7 illustrates the DoSTDM design with flow of data from start to end. The system collects real-time data by reading through the firewall logs. It then passes this information to the data processor section whereby the data is re-presented in the format required per protocol activity and direction. The re-formatted data is also stored in the historical database to be re-baseline the network activity and decide the current detection threshold. In the same time the model will forward the formatted data to the forecasting engines (statistical modeling and neural network modelling engines). The forecasted data of rejected packets is then compared and measured against the last threshold value. If the rejected packets are higher than the threshold, an alarm will be raised. Otherwise, the process continues and returns back to recollect real-time data and repeats the process again. The process can be repeated based on pre-set interval that can be decided by the firewall administrator.

### **5.3 DoSTDM PROCESS IMPLEMENTATION GUIDE**

DoSTDM design approaches the solution of the problem from a network management point of view. Therefore, instead of setting the threshold to raise an alert by configuration rules, the model will re-calculate the threshold based on the history of average rejected packets as recorded by the firewall logs. The proposed solution model is different from existing IDS / IPS in that it can change the detection threshold dynamically using the following steps that follows design in Figure 5 – 7:

**Step 1:** Analyse firewall logs and obtain protocol statistics using any type of scripting language available within the firewall to re-structure the log data.

**Step 2:** Calculate the baseline natural growth threshold of rejected packets over the past available log history using averages of the data per protocol collected in step 1.

**Step 3:** Forecast future levels of rejected packets. DoSTDM calculates this using both statistical and neural network engines.

**Step 4:** Compare forecasted data with real data within units of hours or days and if the difference between real data and forecasted data exceeds the threshold then raise appropriate alerts. Otherwise, the process repeats all previous steps dynamically.

These steps offer the basic guidelines that are required to implement DoSTDM design; however, the implementation steps can vary depending on the implementation hardware and software platform.

#### **5.4 DoSTDM MODEL CHARACTERISTICS AND LIMITATIONS**

DoSTDM collects data in a numerical format; therefore, the quantitative empirical approach for data analysis was the most suitable approach to study the causal relationship between dependent factors within the produced network baseline. This helped in facilitating the dynamic threshold function of DoSTDM which represent it as dynamic behaviour to predict and detect DoS / DDoS attacks. Another aspect of DoSTDM was the use of the MLP neural network engine to ensure the accuracy of the measurement unit (i.e. the total number of rejected packets per hour) obtained by the statistical modeling. This triangulation of quantitative analysis of forecasted data added more stability and reliability to the final results predicted by DoSTDM over a larger baseline that can contain weeks and months of log data for a specific network.

In addition, DoSTDM analysis approach provided the ability to generalise the model between different network configurations and security policies implemented in different operating systems and different hardware platforms as long as the baseline model variables (described earlier in Chapter 4, Table 4 – 4) are shared among these networks using the TCP / IP protocol suit. However, DoSTDM was only tested on a closed network simulated experimental environment with controlled DoS / DDoS attacks on variable lengths and was not implemented on a real life network due to the fact that no production networks were available to test it.

Finally, it is worth noting the fact that MLP neural network processing algorithm is considered to be a universal approximation neural network like any other neural network (e.g. Radial Basis Function) that has the tendency to overtrain the non-linear model data (Nisbet et al. 2009)

#### **5.5 CHAPTER CONCLUSION**

This chapter describes the design criteria of a simpler DoS / DDoS attack detection model by addressing the problems of previous work as explained earlier in Chapter 2. It addresses first problem of handling unknown attacks by using an anomaly detection approach that can handle for both known and unknown techniques and remove the assumption that every network can have an IDS or an IPS protection as it is not always the case and concentrates only on firewall systems that exist in every

network attached to the internet. This means that the proposed solution will be available for every type of network whether it is small, medium or enterprise level network to handle unknown attacks using packets travelling through the firewall only and independent from pre-defined attack signatures. Therefore, every attack is considered to be a new attack as long as it is not normal network traffic behaviour that existed before for this network using the network baseline.

Addressing the second problem of real data patterns was achieved by creating a network baseline before implementing DoSTDM on a simulated network that can be applied to any real life network firewall data using the network traffic patterns activities of known and commonly used protocols like TCP, IP, ICMP and UDP. This will eliminate any dependency pre-defined sets of attacks as in the case of testing with DARPA data sets. It will make the network data unique to every implantation of DoSTDM model and therefore, DoSTDM can be made available to any firewalled network and generalised the implementation across small, medium and enterprise level networks as long as they use TCP / IP suite of protocols.

The third problem of false alerting management has been addressed in section 5.3 of this chapter by identifying how to setup the detection threshold where by the proposed solution (i.e. DoSTDM) can identify new unknown attacks and reconfigure the detection threshold dynamically. This research achieved that by building on previous published research which has shown that some techniques like neural networks can produce accurate alerts better than other tools or techniques like statistical models or packet analysis.

Due to the simplicity of the DoSTDM approach following the KISS principle compared to complex commercial products and solutions, it is practical and can be integrated with any existing network without the need to re-organise the way the network normally operates and with less need to spend more time and money on top of the existing organisational IT budget. This will provide both sides of security handling technologies to existing network infrastructures (i.e. the reactive side and the preventive side as explained in Chapter 2, Figure 2 – 1).

Therefore, this research proposes a joining of techniques to better reduce false alerts by identifying the best threshold value dynamically of the attacked victim. This aspect will be established by creating a baseline profile for all network traffic activities passing through the attacked network firewall.

The main advantage of DoSTDM proposed solution comes from the fact that it uses a double triangulation approach between neural network and statistical analysis which also triangulate internally between two statistical techniques (Holt-Winter analysis and Multiple Linear Regression). This approach will generate a more accurate attack detection threshold value and optimise the detection model results by reducing false alerts.

The DoSTDM models a solution for the DoS / DDoS problem using Artificial Neural Networks (ANN) theory, statistical modelling and forecasting theories and follows the Artificial Immune System (AIS) analogies to investigate and detect network anomalies in the same way a human react to viruses and germs attacking the human body. However, as complex as these theory might looks like, DoSTDM scaffolds and build from these theories a very basic model to detect the number of rejected packets and to use a quantitative analysis approach to develop the baseline history that is updated and analysed dynamically with any new changes to the packets level in order to adjust the network threshold of rejected packets as shown in Figure 5 – 7.

From a theoretical point of view, the DoSTDM modelling process has been successful in following a research theoretical principal to target the DoS / DDoS problem using a simplistic approach. This approach aimed to enhance the existing approaches in resolving the DoS / DDoS problem in order to deliver a fast detection technique based on self learning and the sense of danger toward the network under attack.

Therefore, in comparison with previous work in the field of DoS / DDoS detection, the main advantage that DoSTDM achieved over previous work (from a theoretical perspective) is the ability to scale down the problem domain solution theory to a single attack indicator (i.e. the rejected packets level) making the detection process easy to achieve.

While DoSTDM detection process can deliver a detection with less complexity and more accuracy due to its reliance on past network experience, it is worth noting that every DoSTDM implementation is considered to be a special case by itself as it is a mirror representation of the network it lives within. Therefore, DoSTDM usability and ability to integrate with firewall data logs (as explained previously in Chapter 4) is another aspect of this model.

Finally, the work within this chapter also adheres to the research design methodology (as described in Chapter 3) and highlights the advantages that quantitative approaches can offer when dealing with high amounts of unprocessed firewall logs and the handling of this data.

The next chapter will discuss the simulated DoSTDM implementation driven by the implementation guide explained earlier in section 5.3 of this chapter.

# CHAPTER 6

## DOSTDM IMPLEMENTATION & FINDINGS ANALYSIS

This chapter describes in detail the practical implementation steps of DoS Tracking and Detection model (DoSTDM). It covers the coding and integration of DoSTDM model components previously designed in Chapter 5 and the programming and analysis steps taken to analyse research findings.

Chapter six is divided into five main sections:

- Section one introduces the DoSTDM implementation strategy.
- Section two examines the practical parts including software design that implements DoSTDM model by using the data structures described previously in Chapter 4 and the design cycle described in Chapter 5 with a guideline process to re-implement DoSTDM in any firewall protected network.
- Section three examines data analysis and findings from statistical models compared to DoSTDM neural network MLP forecasting of rejected packets by the firewall.
- Section four discusses the model threshold handling process of rejected packets to manage and minimise false alerts obtained by DoSTDM.
- Section five provides a conclusion to this chapter and emphasises the relation between model implementation and findings in the context of this research.

## 6.1 INTRODUCTION

In this chapter, the research discusses the implementation taken to collect, format, construct and analyse DoSTDM findings as per the design discussed in Chapter 5 and presented in Figure 5 – 7. The tools and programmes chosen have a native format to enable security and firewalls administrators to minimise the risk of software integration issues that may arise during the implementation phase of DoSTDM in any firewalled network, large, medium or small in size. However, many tools and programming languages can be used to implement the DoSTDM model. The research analyse the case study network using a quantitative approach and build on this analysis to construct the simulation network. In this chapter DoSTDM's analyse forecasted data in comparison to baseline data for both networks using firewall logs over a period of many months.

## 6.2 DOSTDM IMPLEMENTATION

### 6.2.1 Simulation Network Setup

The simulated network created using the diagram explained in Figure 4 – 2 mimics a real network with two major components:

1. The simulated network internet Nemesis server generating the traffic and attacks over TCP, IP, DNS, RIP, ICMP, UDP, ARP and Ethernet protocols packets mimics the outside internet world targeting the firewall system. The hardware and software configuration is listed in Table 6 – 1.

Hardware Setup		
Processor	Memory	IP Address
3.0 GHz	1.5 GB	192.168.3.1
Software Setup		
Operating System	Kernel Level	Research Related Installed Packages
Ubuntu Linux	2.6.35-28 SMP Kernel	Apache version 2.2.16 Samba version 3.5.4 Libnet version 1.0.2a Nemesis version 1.4 Build 26 Java JVM version 1.6.0 update 20

Table 6 – 1: Nemesis Server Setup

- The simulated network Filseclab firewall server responded to all received packets generated by the Nemesis server. This server also mimics internal user activities passing through the firewall to the outside network such as HTTP requests to the internet server (Nemesis server). Table 6 – 2 shows this server hardware and software setup.

Hardware Setup		
Processor	Memory	IP Address
1.7 GHz	2.0 GB	192.168.3.12
Software Setup		
Operating System	Kernel Level	Research Related Installed Packages
Microsoft Windows	2003-SP1 Kernel	Filseclab version 3.0.0.8686 Professional Java JVM version 1.5.0 update 5

Table 6 – 2: Firewall Server Setup

The Nemesis server runs on a Linux platform using the CRON scheduler service to run a set of scripts to generate the needed inbound traffic patterns. The scripts run with one protocol repeated every 7 minutes per hour continuously for 24 hours.

These scripts are basically a set of loops that uses a randomly generated number of cycles every minute to send the needed packets as shown in the pseudo shown in Table 6 – 3. Please refer to Appendix – A for the shell scripts used for each protocol.

<pre> <b>Script Nemesis_Packet_Generator ()</b> SET Count value to Zero SET Random value to pseudo-random numbers divided by two FOR each Count while Count less than or equal to Random value     Call Nemesis to generate TCP / IP traffic packets     SET Count incremental by one END LOOP <b>END</b> </pre>
--

Table 6 – 3: Nemesis Packets Generator Script Pseudo Code

The firewall server also runs Filseclab continuously with a standard set of rules similar to any other type of firewall like Checkpoint or IP Chains to block known unauthorised access to the simulated network environment. It also generates end users HTTP packets to the Nemesis default web server (Apache) on default port 80 to mimic WWW requests. In addition a scheduled task to copy firewall collected logs to the Nemesis server mimics activities of file sharing using Samba over SMB protocol between the two servers. The pseudo code shown in Table 6 – 4 was implemented in Java to achieve WWW web requests. This code runs only between 7:00AM and 5:00PM every working day to simulate working time when users normally access the internet.

<pre> <b>CLASS Traffic (Servers_List)</b> SET Server Name to null FOR each line in Servers_List     READ Server Name     CALL HTTP Driver to open a SOCKET on PORT 80 with Nemesis server     GET WWW web page contents     PRINT web page END LOOP <b>END</b> </pre>
---

Table 6 – 4: Firewall HTTP Generator Pseudo Code

Note: The HTTP driver and Traffic Class was originally written in Java by Vikram Rangnekar [yr@udel.edu](mailto:yr@udel.edu) and distributed under GNU (GENERAL PUBLIC LICENSE Version 2, June 1991) to simulate HTTP traffic with Java.

### 6.2.2 Statistical Forecasting Engine

The statistical forecasting engine has two modules, the first is Holt-Winter and the second is Multiple Linear Regression. The Holt-Winter algorithm was implemented from scratch on Java to be used with DoSTDM, Table 6 – 5 shows the Holt-Winter forecasting class pseudo code with all related sub functions for this algorithm to work.

```

CLASS Holt_Winter (No_of_Training_Patterns, Training_Patterns_Array,
Testing_Patterns_Array)
SET No_of_Training_Patterns value
SET Training_Array to Training_Patterns_Array
SET Testing_Array to Testing_Patterns_Array
SET Time_Period to No._of_Training_Patterns value
SET Season_Length to No. of Hours per day
SET No_of_Seasons to Training_Array size divided by Season_Length
CALL Training_Data_Constructor
CALL Testing_Data_Constructor
CALL Holt_Winter_Parameters_Constructor
CALL Trend_Component_Calculator
CALL Seasonal_Component_Calculator
SET Alpha, Gamma and Delta initial values to zero
FOR each value of Alpha with increments between "0" and "1"
    FOR each value of Gamma with increments between "0" and "1"
        FOR each value of Delta with increments between "0" and "1"
            SET Alpha, Gamma, Delta
            CALL Forecasting_Calculator with Alpha, Gamma and Delta
            CALL Training_Errors
        END LOOP
    END LOOP
END LOOP
SELECT Alpha, Gamma and Delta with lowest RMSE returned by Training_Errors
CALL Forecasting_Calculator with selected Alpha, Gamma and Delta values
PRINT results in stored in Forecasting_Output Array with lowest RMSE
END

=====

FUNCTION Training_Data_Constructor
FOR each count in No_of_Training_Patterns
    SET Training_Array values
    SET Time_Period to No_of_Training_Patterns divided by two
    SET Time_Squared value to square value of Time_Period
END LOOP
END

```

**FUNCTION Testing\_Data\_Constructor**

FOR each count in No\_of\_Testing\_Patterns

    SET Testing\_Pattern value in array

END LOOP

**END**

=====

**FUNCTION Holt\_Winter\_Parameters\_Constructor**

SET Trend\_Coefficient\_A value to

$$a = \frac{\text{Total count of packets for the training period}}{\text{Total number of readings in this training period}}$$

SET Trend\_Coefficient\_B value to

$$b = \frac{\text{Total of (packets count} \times \text{time period)}}{\text{Total of ((time point)}^2)}$$

**END**

=====

**FUNCTION Trend\_Component\_Calculator**

FOR each training pattern in Training\_Array

    SET Calculated\_Trend to  $Y(t) = a + bt$

    SET Multi\_Seasonal\_Index to Real\_Reject divided by Calculated\_Trend

END LOOP

**END**

=====

**FUNCTION Seasonal\_Component\_Calculator**

FOR each count in the Season\_Length

    FOR each training pattern count in Training\_Array / Season\_Length

        SET Average\_Seasonal\_Index to the total of Multi\_Seasonal\_Index  
        divided by No\_of\_Seasons

    END LOOP

END LOOP

**END**

=====

**FUNCTION Forecasting\_Calculator**

FOR each count in Season\_Length

SET the forecasting period “m” forward in time

SET the season period “s” (i.e. 24 hours)

```

SET the time period “t” using last reading in the training period

SET Level Adjustment to  $L_t = \alpha \left( \frac{Y_t}{S_{t-s}} \right) + (1 - \alpha)(L_{t-1} + b_{t-1})$ 

SET Trend Adjustment to  $b_t = \gamma(L_t - L_{t-1}) + (1 - \gamma)b_{t-1}$ 

SET Seasonal Adjustment to  $S_t = \delta \left( \frac{Y_t}{L_t} \right) + (1 - \delta)S_{t-s}$ 

SET Forecasted_Value in Forecasting Array to  $F_{t+m} = (L_t + b_t m)S_{t-s+m}$ 

END LOOP

END

=====

FUNCTION Training_Errors

SET Root_Mean_Square_Error to zero

FOR each Training pattern in No_of_Training_Patterns
    SET Mean_Error to difference between Forecast and Training_Pattern
    values at the same Time_Period
    SET Mean_Absolute_Error to absolute value of the Mean_Error
    SET Mean_Square_Error to square value of Mean_Absolute_Error
END LOOP

SET Root_Mean_Square_Error = Square_Root(Mean_Square_Error)

STORE Alpha, Gamma and Delta values for this RMSE in array

END

=====

FUNCTION Forecasting_Errors

SET Root_Mean_Square_Error to zero

FOR each Testing pattern in No_of_Testing_Patterns
    SET Mean_Error to difference between Forecast and Testing_Pattern
    values at the same Time_Period
    SET Mean_Absolute_Error to absolute value of the Mean_Error
    SET Mean_Square_Error to square value of Mean_Absolute_Error
END LOOP

SET Root_Mean_Square_Error = Square_Root(Mean_Square_Error)

END

```

Table 6 – 5: Holt-Winter Pseudo Code

The Linear Multiple Regression algorithm has been implemented using a Java interface calling mathematical analysis Java classes published by Dr. Michael Thomas Flanagan from UCL Department of Electronics & Electrical Engineering ([www.ee.ucl.ac.uk/~mflanaga](http://www.ee.ucl.ac.uk/~mflanaga)). Please see Table 6 – 6 for a pseudo code of the regression interface that uses the Flanagan mathematical classes.

```

CLASS REGRESSION (Training Starting Day, Training Ending Day)
SET Array Length to Regressed Days (Days between Starting Day and End Day)
SET Regression Array to hold data Network Protocols
SET Dummy Variables Initial Values to Zero to handle Seasonality in the Data
FOR each count in “i”
    (where “i” is the Training Data Patterns within the Regressed Days Period)
    SET Regression Array at day “i” with:
        TCP Protocol Inbound traffic values in day “i”
        ICMP Protocol Inbound traffic values in day “i”
        UDP Protocol Inbound traffic values in day “i”
        Other Network Protocols Inbound traffic values in day “i”
        No. of Rejected Packects in day “i”
        Dummy Variable per working hours (8:00AM to 6:00PM) in day “i”
    CALL REGRESSION_FUNCTION Class to hold the Regression Data
    Function “Y” values at training day “i” ( $Y_i = \beta_0 + \beta_1 X_{1,i} + \dots + \beta_k X_{k,i}$ )
END LOOP
CALL External Regression Class (i.e. Flanagan Mathematical Class) to calculate
Regression Parameters ( $\beta_0, \beta_1, \dots, \beta_k$ ) and the Rejected forecasting values of “Y”
END
=====
CLASS REGRESSION_FUNCTION implement Regression Function ( )
    SET Initial Regression Parameter  $\beta_0 = 0$ ;
    SET Regression Function  $Y_i = \beta_0 + \beta_1 X_{1,i} + \dots + \beta_k X_{k,i}$ 
    RETURN  $Y_i$ 
END

```

Table 6 – 6: Pseudo Code for Multiple Linear Regression Interface to Flanagan Regression Class

### 6.2.3 Neural Network Pseudo Code

The original implementation of the MLP Neural Network model has been implemented after the free MLP basic programming code made free by Phil Brierley (published in his web site (<http://www.philbrierley.com/code.html>)).

However, more functions were added in the context of this research to modify and improve his basic MLP code in order to use it with the firewall logs of data. The initial code was freely offered by the author to be used and modified in many programming languages (including C / C++, C#, Java, Perl and PHP).

One of the most important modifications made to the original code was the Normalization and De-normalization function which takes care of setting up the data for the neural network nodes during input and output processing stages as illustrated in Figure 5 – 6 within the previous chapter. Table 6 – 7 shows the MLP Neural Network implementation classes in pseudo code.

```
CLASS MLP-NN (numInputs, numHidden, numTrainingPatterns,  
numTestingPatterns)  
SET No._of_Input_Nodes = numInputs  
SET No._of_Epochs = 100 (range from 100 to 1000)  
SET No._of_Training_Patterns = numTrainingPatterns (decided by user)  
SET No._of_Testing_Patterns = numTestingPatterns (decided by user)  
SET No._of_Hidden_Nurons = numHidden (range from 1 to 15)  
SET Learning_Rate_for_Hidden_Input_Nurons = 0.7 (range from 0.1 to 1.0)  
SET Learning_Rate_for_Hidden_Output_Nurons = 0.07 (range from 0.01 to 0.1)  
SET Training_Inputs = Array [numTrainingPatterns][numInputs]  
SET Training_Output = Array [numTrainingPatterns]  
SET Testing_Inputs = Array [numTestingPatterns][numInputs]  
SET Testing_Real_Output = Array [numTestingPatterns]  
SET Testing_Prediction_Output = Array [numTestingPatterns]  
SET Input_Hidden_Node_Weights = Array [numInputs][numHidden]  
SET Output_Hidden_Node_Weights = Array [numHidden]  
SET Maximum_Input_Training_Pattern_Value = Array [numTrainingPatterns]  
SET Minimum_Input_Training_Pattern_Value = Array [numTrainingPatterns]
```

```

SET Normalised_Pattern_Input = Array [numTrainingPatterns]
SET Normalised_Pattern_Output = Array [numTrainingPatterns]
CALL Hidden_Nodes_Weight_Generator
CALL Training_Data_Constructor
CALL Pattern_Minimum_Constructor
CALL Pattern_Maximum_Constructor
CALL Normaliser(Training_Pattern_Input)
CALL Normalizer(Training_Pattern_Output)
FOR each epoch count in (numEpoch)
    FOR each training pattern in (numTrainingPatterns)
        Select a pattern randomly
        CALL Calculate_Neural_Network
        CALL Change_Hidden_Node_Output_Weight
        CALL Change_Hidden_Node_Input_Weight
    END LOOP
    CALL Total_Training_Error
END LOOP
CALL Normaliser(Testing_Pattern_Input)
CALL Testing_Data_Constructor
CALL Testing_Data_Neural_Network_Constructor
SET Output_Error = Testing_Prediction_Output - Testing_Real_Output
PRINT Testing_Prediction_Output and Corresponding Output_Error Value
END

```

```

=====
FUNCTION Hidden_Nodes_Weight_Generator

```

```

FOR each count in (numHidden)
    SET Output_Hidden_Node_Weights = (Random_Number - 0.5) / 2
    FOR each count in (numInputs)
        SET Input_Hidden_Node_Weights = (Random_Number - 0.5) / 5
    END LOOP
END LOOP
END

```

```

=====

```

**FUNCTION Calculate\_Neural\_Network**

```
FOR each hidden node input in (numHidden)
    SET Hidden_Input_Node_Value to zero
    FOR each Training_Input in (numInputs)
        SET Hidden_Input_Node_Value = Hidden_Input_Node_Value +
            (Training_Input x Input_Hidden_Node_Weights)
    END LOOP
    CALL TANH_Transformation(Hidden_Input_Node_Value)
    SET Hidden_Input_Node_Value to TANH_Transformation value
END LOOP
SET Hidden_Output_Node_Value to zero
FOR each hidden node output in (numHidden)
    SET Hidden_Output_Node_Value = Hidden_Output_Node_Value +
        (Training_Output x Output_Hidden_Node_Weights)
END LOOP
SET Training_Pattern_Output_Error = Hidden_Output_Node_Value - Training_Output
END
```

=====

**FUNCTION Change\_Hidden\_Node\_Input\_Weight**

```
FOR each hidden output node in (numHidden)
    FOR each Training_Input in (numInputs)
        SET Weight_Change_Value = 1 - (Hidden_Input_Node_Value)2
        SET Weight_Change_Value = (Weight_Change_Value x
            Training_Pattern_Output_Error x
            Learning_Rate_for_Hidden_Output_Nurons)
        SET Weight_Change_Value = (Weight_Change_Value x Training_Input)
        SET Input_Hidden_Node_Weights = (Input_Hidden_Node_Weights -
            Weight_Change_Value)
    END LOOP
END LOOP
END
```

=====

**FUNCTION Change\_Hidden\_Node\_Output\_Weight**

```
FOR each hidden output node in (numHidden)
    SET Weight_Change_Value = (Learning_Rate_for_Hidden_Output_Nurons x
                               Training_Pattern_Output_Error x
                               Hidden_Output_Node_Value)

    SET Output_Hidden_Node_Weights = (Output_Hidden_Node_Weights -
                                       Weight_Change_Value)

    IF (Output_Hidden_Node_Weights < -5) THEN
        Output_Hidden_Node_Weights = -5
    ELSE
        IF (Output_Hidden_Node_Weights > 5) THEN
            Output_Hidden_Node_Weights = 5
        END IF
    END IF
END LOOP
END
```

=====

**FUNCTION Training\_Data\_Constructor**

```
FOR each count in (numTrainingPatterns)
    FOR each input in (numInputs)
        CALL Normliser(Training_Input)
        SET Training_Inputs[numTrainingPatterns][numInputs] to the Normalised
        value of Training_Input
        SET Training_Inputs[numTrainingPatterns][numInputs+1] to 1 as input bias
        CALL Normliser(Training_Output)
        SET Training_Output[numTrainingPatterns] to the Normalised
        value of Training_Output
    END LOOP
END
```

=====

**FUNCTION TANH\_Transformation(Hidden\_Input\_Node\_Value)**

IF (Hidden\_Input\_Node\_Value &gt; 20) THEN

RETURN {1}

ELSE

IF (Hidden\_Input\_Node\_Value &lt; -20) THEN

RETURN {-1}

ELSE

        RETURN TANH Transformation Value as:  $\left\{ \frac{e^x - e^{-x}}{e^x + e^{-x}} \right\}$ 

END IF

END IF

**END****FUNCTION Total\_Training\_Error**

SET Root\_Mean\_Square\_Error to zero

FOR each training pattern in (numTrainingPatterns)

CALL Calculate\_Neural\_Network

SET Root\_Mean\_Square\_Error = Root\_Mean\_Square\_Error +

(Training\_Pattern\_Output\_Error x Training\_Pattern\_Output\_Error)

END LOOP

SET Root\_Mean\_Square\_Error =  $\frac{\text{Root\_Mean\_Square\_Error}}{\text{numTrainingPatterns}}$ **END****FUNCTION Pattern\_Minimum\_Constructor(Pattern\_Array)**

SET Minimum to first item in Pattern\_Array

FOR each item in (Pattern\_Array)

IF item in current position is less than first item THEN

SET Minimum to current item in Pattern\_Array

END IF

END LOOP

RETURN Minimum

**END**

**FUNCTION Pattern\_Maximum\_Constructor(Pattern\_Array)**

```
SET Maximum to first item in Pattern_Array
FOR each item in (Pattern_Array)
    IF item in current position is larger than first item THEN
        SET Maximum to current item in Pattern_Array
    END IF
END LOOP
RETURN Maximum
END
```

=====

**FUNCTION Testing\_Data\_Constructor**

```
FOR each count in (numTestingPatterns)
    FOR each input in (numInputs)
        CALL Normliser(Testing_Input)
        SET Testing_Inputs[numTestingPatterns][numInputs] to the Normalised
        value of Testing_Input
        SET Testing_Inputs[numTestingPatterns][numInputs+1] to 1 as input bias
        SET Testing_Output[numTestingPatterns] to Testing_Output
    END LOOP
END
```

=====

**FUNCTION Normaliser(Pattern\_Value)**

```
SET Min to used pattern (training or test) Minimum
SET Max to used pattern (training or test) Maximum
SET Normlised_Value =  $\left[ \left\{ \left( \frac{Pattern\_Value - Min}{Max - Min} \right) - \left( \frac{Max - Min}{Max + Min} \right) \right\} x \left\{ \frac{Max + Min}{Max - Min} \right\} \right]$ 
RETURN Normlised_Value
END
```

=====

**FUNCTION De\_Normaliser(Pattern\_Value)**

```
SET Min to used pattern (training or test) Minimum
SET Max to used pattern (training or test) Maximum
```

```

SET De_Normalised_Value =  $\left\{ \left( \frac{(Normalised\_Value + 1) \times (Max - Min)^2}{Max + Min} \right) + Min \right\}$ 
RETURN De_Normalised_Value
END
=====
FUNCTION Testing_Data_Neural_Network_Constructor
FOR each hidden output node in (numHidden)
    SET Hidden_Input_Node_Value to zero
    FOR each Testing_Input in (numInputs)
        SET Hidden_Input_Node_Value = Hidden_Input_Node_Value +
            (Testing_Input x Input_Hidden_Node_Weights)
        SET Hidden_Input_Node_Value = TANH_Transformation(
            Hidden_Input_Node_Value)
    END LOOP
SET Hidden_Output_Node_Value to zero
FOR each hidden node output in (numHidden)
    SET Hidden_Output_Node_Value = Hidden_Output_Node_Value +
        (Hidden_Input_Node_Value x Output_Hidden_Node_Weights)
    CALL De_Normaliser(Hidden_Output_Node_Value)
    SET Testing_Prediction_Output Array with returned De_Normalisation Value
END LOOP
END

```

Table 6 – 7: Neural Network (MLP) Pseudo Code

#### 6.2.4 DoSTDM Code Implementation in Java

Java was used to implement the DoSTDM model and the DoSTDM Network Analyser Java applet was created to analyse and implement the previously described forecasting engines in Sections 6.2.2 and 6.2.3.

The Java applet shown in Figures 6 – 1 and 6 – 2 uses data collected from the simulation network described in Section 6.2.1 and pre-processed by methods described discussed in Chapter 4. In addition this Java applet was used to analyse data from the case study network for comparison purposes.

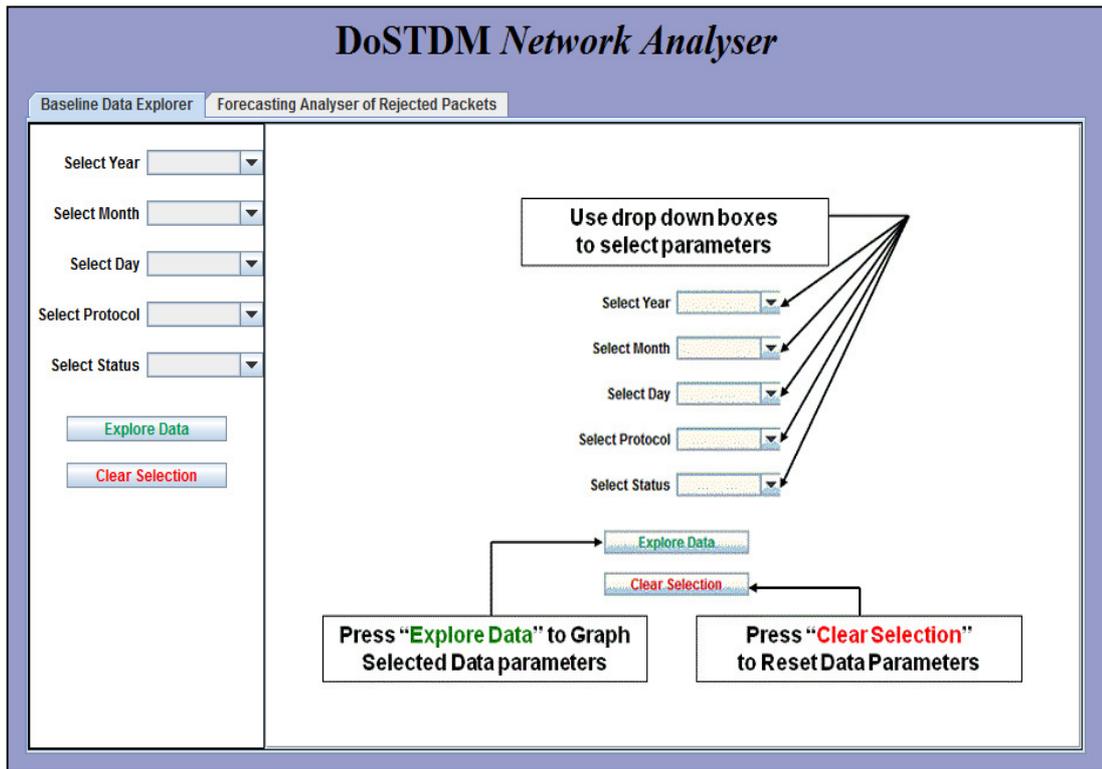


Figure 6 – 1: DoSTDM Baseline Data Explorer Java Applet

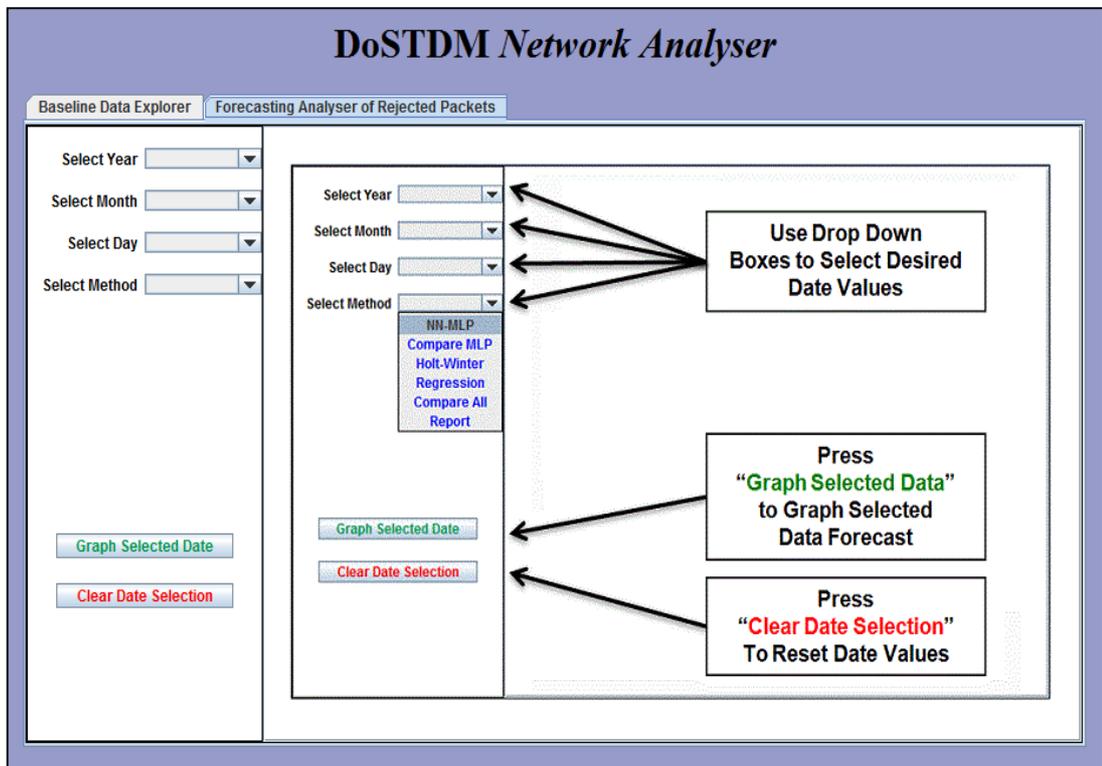


Figure 6 – 2: DoSTDM Rejected Packets Forecasting Analyser Java Applet

This applet has been used to explore and investigate the collected data using DoSTDM model to compare forecasting results. The forecasting results analysis was based on Root Mean Square Error (RMSE) as the measurement unit of DoSTDM forecasting accuracy using the forecasting engines (i.e. statistical and neural network forecasting engines). The next section uses this applet to construct DoSTDM model findings. Please refer to Appendix – A for the Java code listing used to construct this applet and the compiled code in Appendix – C.

Figure 6 – 3 shows an example of DoSTDM model dynamic forecasting in a formatted comparison produced by the DoSTDM Java applet forecasting engines.

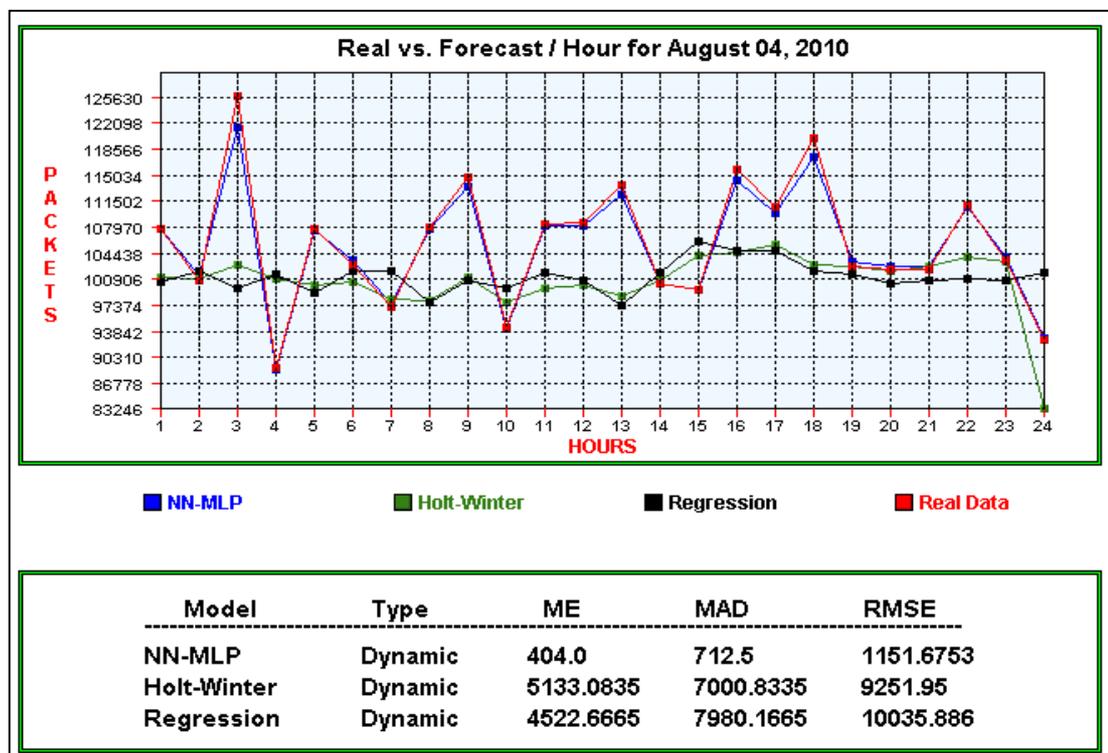


Figure 6 – 3: DoSTDM Rejected Forecasting Comparison Using DoSTDM Applet

Figure 6 – 3 illustrates a comparison graph and the error analysis between the three forecasting engines using the difference between real data and forecasted data. These are called the residuals or errors. However, to measure these errors statistically, the DoSTDM applet uses the following statistical metrics:

1. Mean Error (ME) which is a simple average of errors within the forecasted data using the DoSTDM applet.

2. Mean Absolute Deviation (MAD) which is similar to ME but it uses the absolute value of the error to calculate the overall error of the forecasted data.
3. Root Mean Square Error (RMSE) which is the average of the square root of errors.

Section 6.3.3 of this chapter analyses the results obtained using DoSTDM from the data sets using the RMSE value as it is more commonly used for time series forecasting analysis.

### **6.3 DATA ANALYSIS & FINDINGS**

DoSTDM model has been used to analyse three data sets distributed across two network settings. The first and the second data sets are from the real network case study firewall environment and the third was produced from the simulation network. These data sets were analysed using residual analysis (i.e. the difference between real data and forecasted data). The data sets were used to triangulate the analysis as per the following criteria:

- Case Study First Data Set (CSDS1) from case study network firewall logs (January – May 2004 logs) has been used to perform statistical analysis on firewall data under attack using Holt-Winter and Regression methods and triangulate with DoSTDM forecasting results using MLP method.
- Case Study Second Data Set (CSDS2) from case study network firewall logs (January – December 2005 logs) has been used to study long term data effects on the DoSTDM forecasting results using Holt-Winter, Regression and MLP methods.
- Simulation Network Data Set (SNDS) from simulation network firewall logs (June – December 2010 logs) has been used to study a hypothetical simulation of a real network firewall using DoSTDM to forecast rejected packets under a simulated network traffic using Holt-Winter, Regression and MLP methods.

### 6.3.1 Holt-Winter Statistical Analysis & Findings from CSDS1

In the case study network a period of data from Jan 2004 to May 2004 was collected and analysed as it contains a DoS attack made at the time by the SASSER worm attack. However, due to the need to classify the packets status as either accepted or rejected, only inbound and outbound statistics would be suitable for such a prediction of rejected amount of packets over time. Therefore, by inspecting the forecasting residuals graphs from Holt-Winter statistical method the common pattern in the data that is not seen in any other residuals graphs can be obtained.

Before applying the DoSTDM model to CSDS1, a manual statistical Holt-Winter model forecast was applied using the MINITAB statistical tool. The residual pattern can be easily seen if by plotting the average of all of the forecasted traffic packets (including rejected packets) residuals data in one graph as seen in Figure 6 – 4.

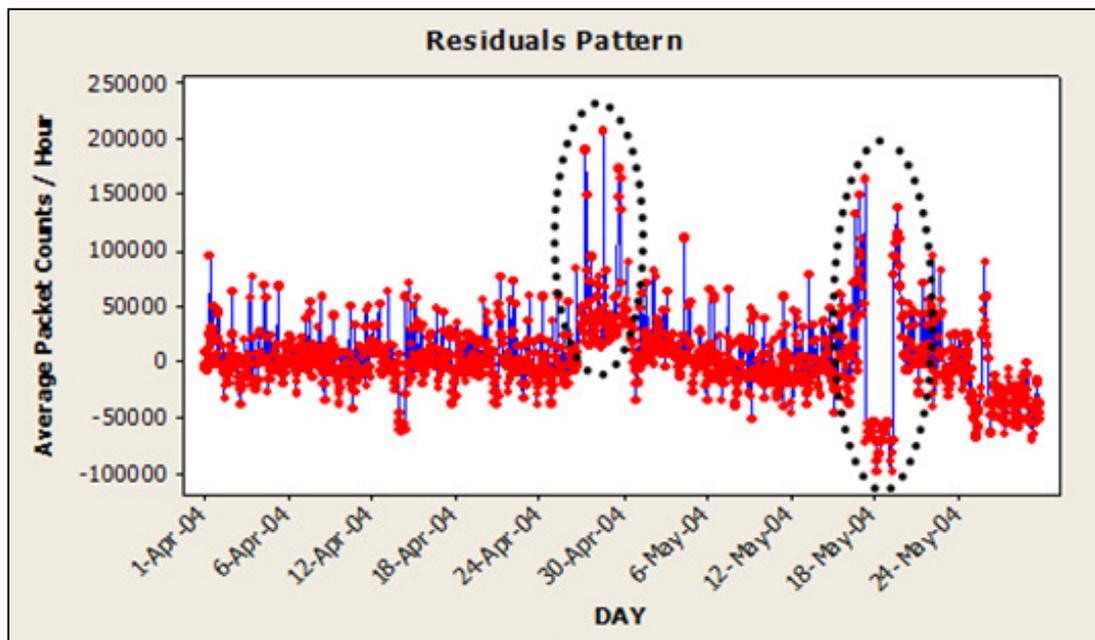


Figure 6 – 4: Average Residuals Pattern in CSDS1

From Figure 6 – 4, two anomalies can be observed (marked by the dashed black circles) as follow:

1. Large spike in the amount of packets over the period from 27<sup>th</sup> April – 1<sup>st</sup> May 2004. Investigating the firewall audit logs shows no explanation for this large spike in the residuals.

2. A mix of an increase and a decrease in the packet levels within the period from 17<sup>th</sup> May – 21<sup>st</sup> May 2004 (after investigating firewall audit logs, it was found that the loss of data was due to introducing a new firewall cluster that affected the logging process causing the loss of some logging data, this was rectified on the 21<sup>st</sup> May 2004).

Therefore, further investigation of the internet activities within the period from 27<sup>th</sup> April – 1<sup>st</sup> May 2004 reveals that there was a widespread infection by an internet malware worm named “SASSER” and only reported to the internet community as early as the 1<sup>st</sup> May 2004 at 14:24 GMT time (Source: <http://isc.sans.org/>). However, the network firewall had been overwhelmed by the SASSER for nearly four days before it was discovered. The SASSER malware uses a vulnerability within the Microsoft Windows 2000 operating system called the Local Security Authority Subsystem Service (LSASS) that was discovered first in 26<sup>th</sup> April, 2004. However, nobody linked the LSASS vulnerability to SASSER till the 30<sup>th</sup> April 2004. SASSER uses certain ports within TCP to spread itself (Ports 445, 5554 and 9996).

In this research, the case study network firewall was not based on a Microsoft Windows operating system and it would block the SASSER packets requests from passing to the LAN network as these ports are not usually opened. Therefore, the only effect of SASSER was limited to flooding the network with a large number of TCP packets passing through the gateway to the firewall interfaces.

This was a typical network DoS attack launched randomly from a SASSER infected Windows systems targeting any network connected to the internet. In order to declare a DoS attack based on the information provided by the residuals obtained from both forecasting baseline models (i.e. Holt-Winter and Regression models), the maximum accepted margin of rejected packets per hour needs to be defined. Therefore, using the real data (not forecasted data) the number of rejected packets can be calculated from both inbound and outbound packets travelling across the firewall between January 1<sup>st</sup> and March 31<sup>st</sup>, 2004. This assisted in obtaining the average number of rejected packets from the network (without having an attack) then compares it with the average in the period from April 1<sup>st</sup> to May 29<sup>th</sup>, 2004.

The difference is the maximum number of rejected packets that can be allowed before declaring a DoS attack. In addition, the total rejected packets per hour can be divided by the total number of packets travelling in and out of the firewall per hour to obtain a percentage representation of the accuracy margin. Table 6 – 8 and Table 6 – 9 shows these measurement and corresponding calculations.

	<b>Inbound Traffic</b>	<b>Outbound Traffic</b>	<b>Rejected Traffic</b>
<b>Average Number of Packets / Hour</b>	93748.41	342.12	51658.11
<b>Total</b>	94090.53		51658.11
<b>Rejected to Total Traffic Percentage</b>	54.9 %		

Table 6 – 8: Rejected Packets Counts per Hour in CSDS1 (January – March, 2004)

	<b>Inbound Traffic</b>	<b>Outbound Traffic</b>	<b>Rejected Packets</b>
<b>Average Number of Packets / Hour</b>	108924.92	366.91	89888.85
<b>Total</b>	109291.83		89888.85
<b>Rejected to Total Traffic Percentage</b>	82.2 %		

Table 6 – 9: Rejected Packets Counts per Hour in CSDS1 (April – May, 2004)

From Tables 6 – 8 and 6 – 9 calculations the following has been observed:

1. The amount of outbound traffic is very low, even negligible compared to the inbound traffic passing through the network at the same period. Therefore, it is clear that the inbound traffic influences the number of rejected packets more significantly than the outbound traffic.
2. An increase of approximately 38,231 packets in the rejected packets per hour on average. Therefore, if the number of rejected packets per hour increased by 38,231 packets then it is save to say that this network is under attack.

In order to verify the above findings from the graphs, the period from 27<sup>th</sup> April – 1<sup>st</sup> May 2004 needs to be expanded with all inbound traffic real data (not forecasted) compared to the average number of rejected packets at the same period (i.e. approximately 89,889 packets per hour obtained from Table 6 – 9) as the maximum limit of allowed rejected packets.

Figure 6 – 5 shows a plot of all inbound protocols data compared to the average rejected data limit.

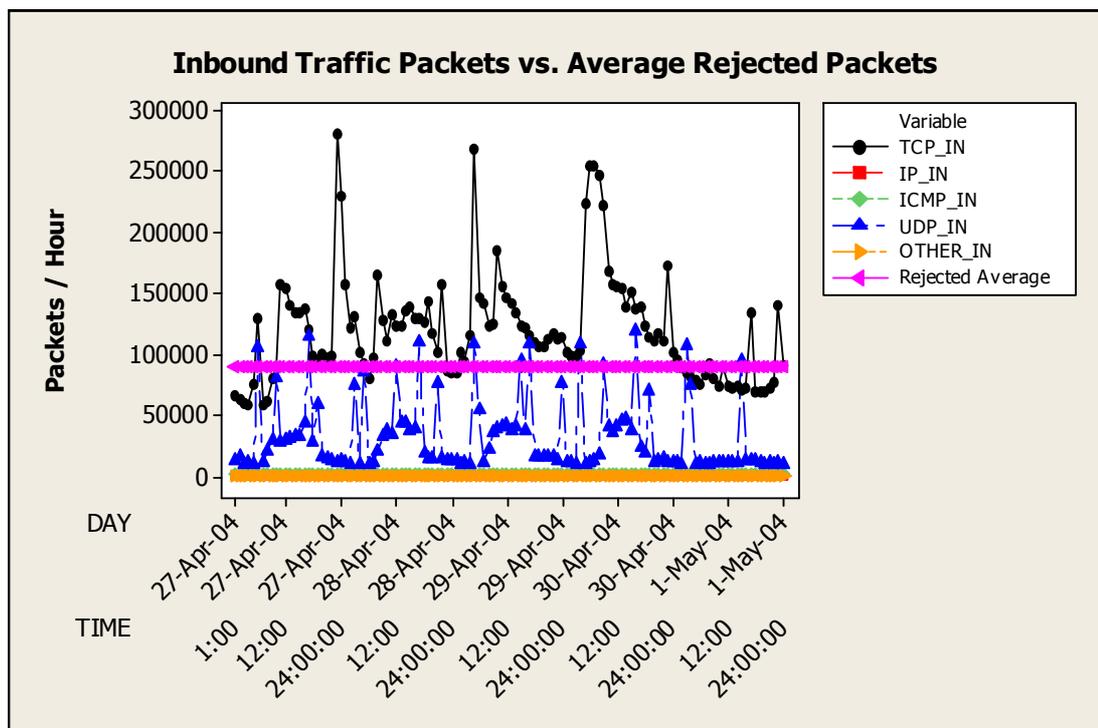


Figure 6 – 5: Comparison of Inbound Packets vs. Rejected Packet in CSDS1

Figure 6 – 5 indicates that the inbound TCP traffic as the largest contributor of traffic in this period (from 27<sup>th</sup> April – 1<sup>st</sup> May 2004) and above the average number of rejected packets. Some random UDP packets have shown some activity; however, in comparison to TCP spikes these are minimal.

Therefore, further analysis and investigation of Holt-Winter forecasting results was carried to confirm the relationship between forecasted TCP inbound traffic residuals and rejected packets residuals shown earlier in Figure 6 – 4.

The Holt-Winter further investigation of forecasting residuals analysis showed a correlation between inbound TCP traffic forecasting residuals and rejected TCP forecasting residuals as shown in Figure 6 – 6 and Figure 6 – 7.

This correlation was very clear at the exact time of the SASSER attack and matched the attacking spike within the forecasting residuals mentioned earlier. Such a correlation identifies inbound TCP traffic as the highest contributor to all rejected packets by the firewall at the time of the SASSER attack.

The following MINITAB residuals analysis figures have been used to draw the high relationship between inbound TCP packets and rejected packets levels in CSDS1 (please see Figure 6 – 6, Figure 6 – 7, Figure 6 – 8 and Figure 6 – 9).

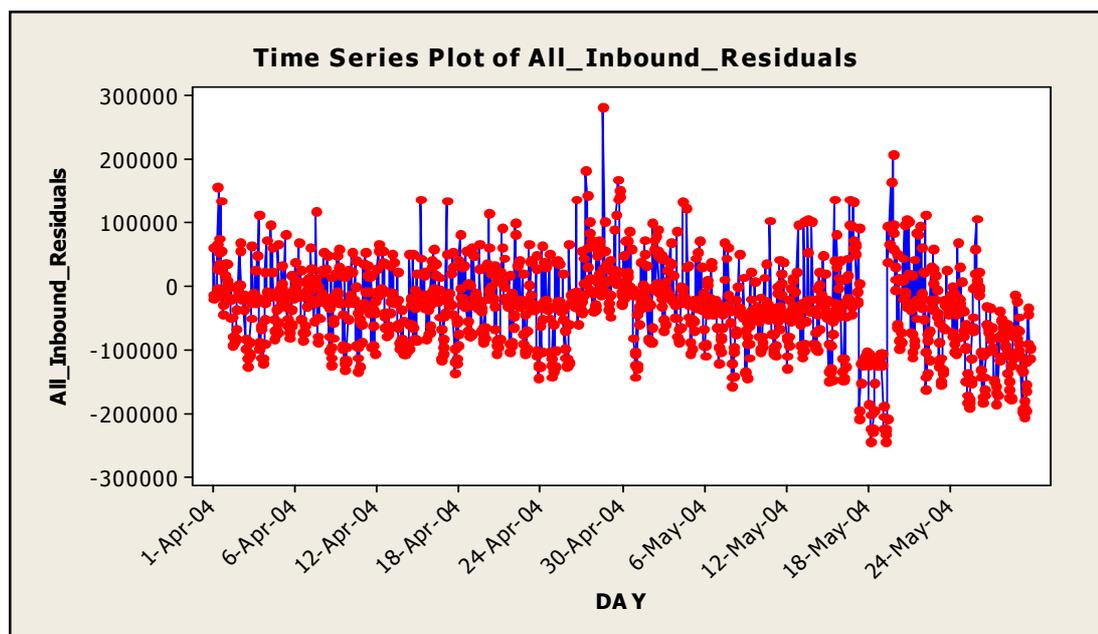


Figure 6 – 6: Holt-Winter Residuals Plot for All Inbound Packets in CSDS1

Figure 6 – 6 illustrates the forecasting residuals for all inbound traffic during the SASSER attack. The forecasting residuals from 27<sup>th</sup> April – 1<sup>st</sup> May 2004 show a spike in the incoming traffic that can only be explained as the SASSER attack. This figure also indicates the correlation between inbound traffic and the attack hence the attack was coming from the internet toward the firewall.

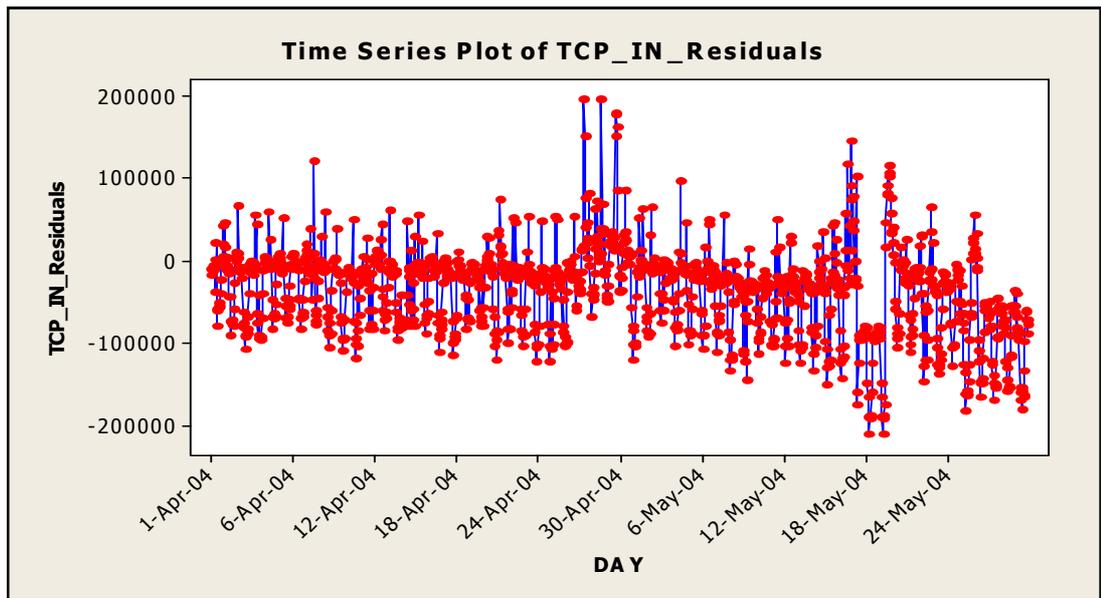


Figure 6 – 7: Holt-Winter Residuals Plot for TCP Inbound Packets in CSDS1

Figure 6 – 7 shows the same anomaly in the same period (27<sup>th</sup> April – 1<sup>st</sup> May 2004) largely represented by TCP inbound protocol forecasting residuals during the SASSER attack. This indicates that the TCP protocol is the contributing highly to the inbound traffic volume of packets.

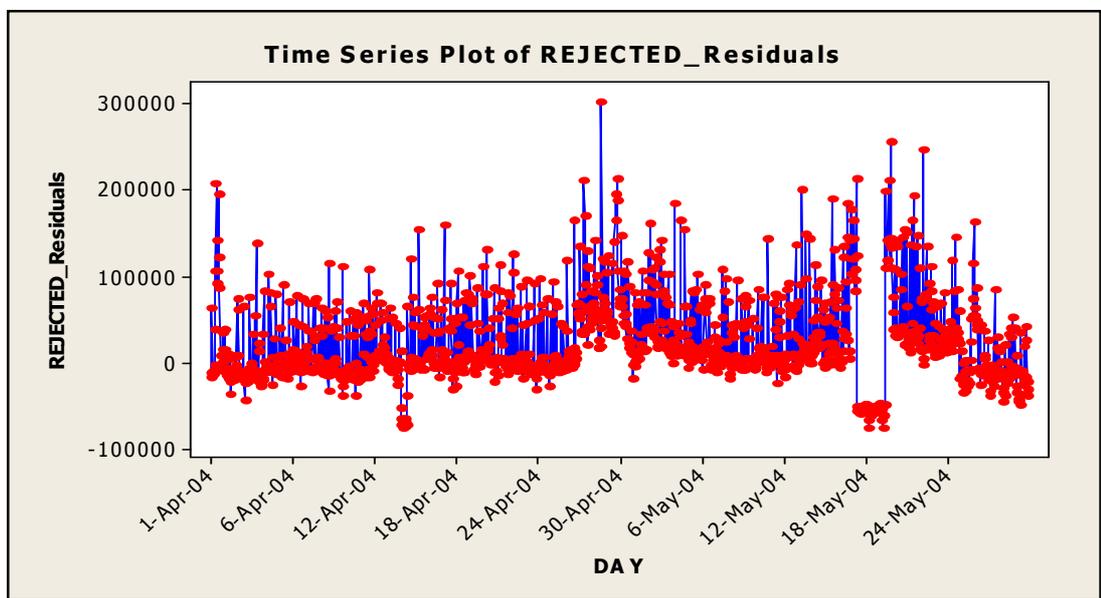


Figure 6 – 8: Holt-Winter Residuals Plot for All Rejected Packets in CSDS1

Figure 6 – 8 shows the total rejected packets residuals (for all protocols) during SASSER attack. The correlation between this figure and Figure 6 – 7 indicated that most of the inbound TCP traffic had been rejected by the firewall.

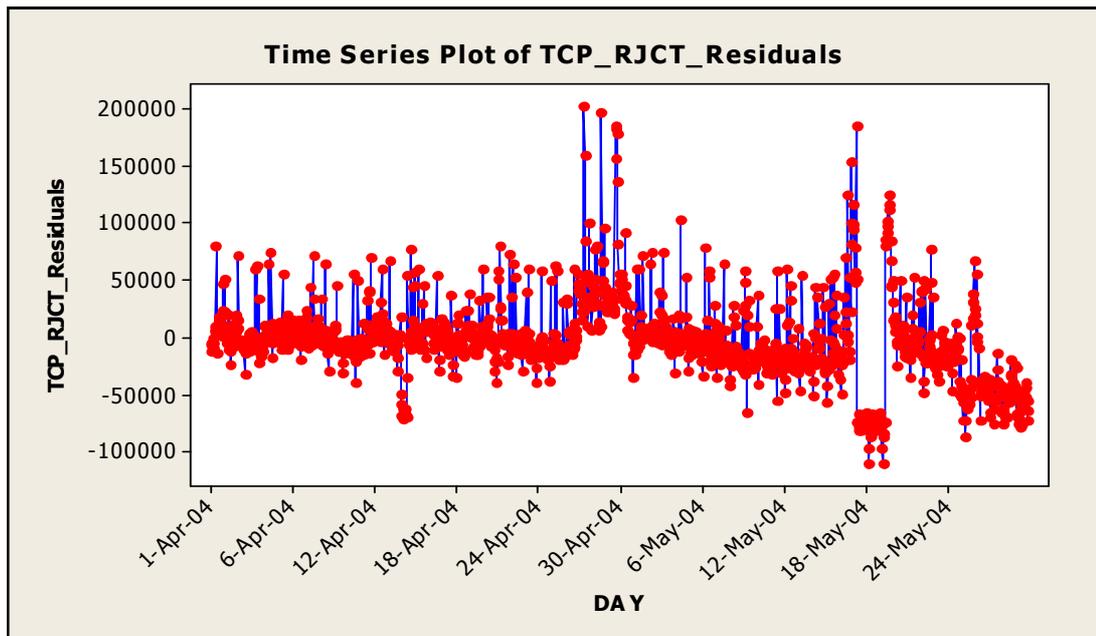


Figure 6 – 9: Holt-Winter Residuals Plot for TCP Rejected Packets in CSDS1

Figure 6 – 9 shows the rejected total TCP rejected packets had increased during SASSER attack. Therefore, this figure confirms the correlation observation between inbound TCP and rejected packets from Figure 6 – 7 and Figure 6 – 8.

This Holt-Winter manual analysis with MINITAB showed the link between inbound traffic, TCP protocol and rejected packets levels. However, the next section using the regression manual analysis in MINITAB will drill further to identify the causal relationship between the protocols, the traffic direction and the rejected packets volume in CSDS1.

### 6.3.2 Regression Statistical Analysis & Findings from CSDS1

In this section the first data set was used to produce a regression forecasting equation of all rejected packets with Analysis of variance (ANOVA) in MINITAB. This is a manual analysis prior to use DoSTDM dynamic analysis. The regressed factors that have been used are the inbound and outbound protocols factors only. The reason for doing so, is to predict the status of the packet been defined as rejected from incoming and outgoing packets passing through the firewall. Neither accepted nor rejected protocols factors can be used (i.e. if the packet status is accepted or rejected it cannot be used to forecast a rejected status since it has already been classified).

MINITAB can generate the regression equation; however, the selection of which factor to be regressed (as part of the total regression equation) needs to be made manually. In order to select which factors can remain in the equation and which can be ignored, the regression method requires the “T” and “P” values to be checked at the beginning of the regression process. Where all regressed values are selected the ANOVA table needs to be checked to see if the regression model is suitable or not. Therefore, when dealing with regression its necessary to setup the confidence level for the regression “T” values. In this research a 95% confidence level is used, this means that the regression function should only contain predictors with “T” absolute values higher than 5% or 0.05 to be incorporated in the model. In other words predictors with T-values higher than 0.05 are considered to be significant in providing information about the rejected number of packets (Source: MINITAB State Guide). The “P” value provides the degree of significance about the relation between the predicting factor and expected response of the regression model (i.e. number of rejected packets per hour); therefore, since the confidence has been set to 95%, for any predictor to remain in the regression model the “P” value needs to be less than 0.05. MINITAB uses special notation in describing the results, these notations might be a bit confusing when looking at the output tables, therefore, Table 6 – 10 includes a description of every parameter used to select the best regression model as defined by MINITAB Statistical Guide.

The statistical manual regression modeling with MINITAB iterates to produce the best subset of factors including dummy variables (mentioned earlier in Chapter 5, Table 5 – 1) that can predict the amount of rejected packets from the firewall traffic pattern as follows:

- Stage One - Regressing inbound and outbound data to produce the initial multiple linear regression equation.
- Stage Two - Regressing inbound and outbound data with dummy seasonality variables to produce the include seasonality factors in the multiple linear regression analysis.
- Stage Three: Select the best subset of protocol factors with seasonality as the final regression equation factors for rejected packets prediction.

<b>MINITAB Parameter Name</b>	<b>Parameter Description</b>
<b>[Coef.]</b>	Coefficient is the estimated constant for every factor used in the regression equation.
<b>[SE-Coef.]</b>	This is the standard error of the coefficient that represents the precession of the estimated coefficient value.
<b>[T-Value]</b>	This is the ratio of the coefficient “Coef.” to its standard error value “SE-Coef.” ( T-Values > 0.05 ).
<b>[P-Value]</b>	Predictor significance to regression response ( P-Value < 0.05 ).
<b>[S]</b>	The estimate of the variance in the data after the linear relationship between the response and the predictor has been taken into account.
<b>[R-Sq]</b>	(R <sup>2</sup> ) Represents the amount of variation in the observed response values that are explained by the regressed predicting factors.
<b>[R-Sq (adj.)]</b>	(Adjusted R <sup>2</sup> ) is the modified R-Sq that has been adjusted for the number of factors in the model, including any unnecessary factors will make R-Sq high. However, adjusted R-Sq may get smaller when factors to the regression model.
<b>[DF]</b>	Degrees of Freedom (DF) for the predictor.
<b>[SS]</b>	The sums of squares are the summation of the squared deviations of the fitted response values from the mean response value. It measures the amount of variation in the response data that is explained by the regression model.
<b>[MS]</b>	This is the mean squares obtained by dividing the sum of squares by the degree of freedom for both regression and regression residuals.
<b>[F]</b>	The F-value for regression is used to test the hypothesis that all the coefficients in a regression model are less than 95% of the “F” distribution with DF regression

Table 6 – 10: Regression Parameters Definitions (Source: MINITAB Statistical Guide)

### 6.3.2.1 Regression of Inbound & Outbound Data

Regressing the first dataset TCP, IP, ICMP, UDP and other protocols with MINITAB produced the following initial regression equation shown in Table 6 – 11 that can predict the amount of rejected packets in the firewall data as a function of inbound and outbound network traffic protocols.

$\text{REJECTED} = 16057 + (0.289) \text{TCP\_IN} - (139) \text{IP\_IN} - (6.93) \text{ICMP\_IN} + (0.969) \text{UDP\_IN} - (358) \text{OTHER\_IN} - (27.2) \text{TCP\_OUT} - (2711) \text{ICMP\_OUT} + (8.98) \text{UDP\_OUT} - (804) \text{OTHER\_OUT}$
---

Table 6 – 11: Regression Equation without Seasonality in CSDS1

Further analysis of regression parameters show more details about the “T” and “P” values that helped in eliminating some variables (marked in red) from the initial regression equation as shown in Table 6 – 12.

Predictor	Coef.	SE Coef.	T	P
Constant	16057	2096	7.66	0.000
TCP_IN	0.289065	0.009182	31.48	0.000
IP_IN	-138.9	416.7	-0.33	0.739
ICMP_IN	-6.932	1.271	-5.45	0.000
UDP_IN	0.96856	0.02505	38.66	0.000
OTHER_IN	-358.48	38.46	-9.32	0.000
TCP_OUT	-27.22	88.24	-0.31	0.758
ICMP_OUT	-2711	2755	-0.98	0.325
UDP_OUT	8.982	4.071	2.21	0.027
OTHER_OUT	-803.9	387.8	-2.07	0.038
S =	24269.7			
R-Sq =	60.9%			
R-Sq(adj) =	60.8%			

Table 6 – 12: Regression P-Values Table for Inbound and Outbound Factors without Seasonality in CSDS1

Table 6 – 12 shows that the “T” and “P” values of regressed factors. Therefore, factors with high P-Values will be taken out of the final regression model using MINITAB search function for best subset of regression factors. Further analysis in Table 6 – 13 shows the ANOVA analysis of “P” values within the produced regression equation.

Source	DF	SS	MS	F	P
Regression	9	1.99782E+12	2.21980E+11	376.87	0.000
Residual Error	2174	1.28052E+12	589016067	N / A	N / A
Total	2183	3.27834E+12	N / A	N / A	N / A

Table 6 – 13: ANOVA Analysis Table for Inbound and Outbound Factors without Seasonality from MINITAB Using CSDS1

### 6.3.2.2 Regression of Inbound & Outbound Data with Seasonality

In this stage of the regression analysis, the seasonal dummy factors are introduced to the regressing process and used MINITAB to re-calculate the “T” and “P” values. The produced regression equation contains dummy variables D1 to D23 to cover for seasonality over 24 hours (i.e. one day) as shown in Table 6 – 14.

$$\text{REJECTED} = 16918 + (0.355) \text{TCP\_IN} + (102) \text{IP\_IN} - (3.73) \text{ICMP\_IN} + (1.04) \text{UDP\_IN} - (240) \text{OTHER\_IN} - (14.7) \text{TCP\_OUT} - (4215) \text{ICMP\_OUT} + (5.49) \text{UDP\_OUT} - (566) \text{OTHER\_OUT} + (1024) \text{D1} + (613) \text{D2} + (878) \text{D3} + (364) \text{D4} - (1359) \text{D5} + (784) \text{D6} - (2397) \text{D7} - (10165) \text{D8} - (21496) \text{D9} - (26170) \text{D10} - (18677) \text{D11} - (18903) \text{D12} - (24436) \text{D13} - (25887) \text{D14} - (25341) \text{D15} - (23412) \text{D16} - (21936) \text{D17} - (7216) \text{D18} - (6030) \text{D19} - (327) \text{D20} - (1631) \text{D21} - (448) \text{D22} + (2286) \text{D23}$$

Table 6 – 14: Regression Equation with Seasonality in CSDS1

The “P” values > 0.05 had been selected and any prediction variable with “P” values < 0.05 has been marked in red to be removed as shown in Table 6 – 15.

Predictor	Coef.	SE Coef.	T	P
Constant	16918	3040	5.56	0.000
TCP_IN	0.355423	0.009265	38.36	0.000
IP_IN	101.7	388.8	0.26	0.794
ICMP_IN	-3.729	1.189	-3.14	0.002
UDP_IN	1.0374	0.02351	44.12	0.000
OTHER_IN	-239.68	36.2	-6.62	0.000
TCP_OUT	-14.69	82.83	-0.18	0.859
ICMP_OUT	-4215	2561	-1.65	0.100
UDP_OUT	5.487	3.799	1.44	0.149
OTHER_OUT	-566.2	360.9	-1.57	0.117
D1	1024	3349	0.31	0.760
D2	613	3344	0.18	0.855
D3	878	3356	0.26	0.794
D4	364	3353	0.11	0.913
D5	-1359	3351	-0.41	0.685
D6	784	3349	0.23	0.815
D7	-2397	3350	-0.72	0.474
D8	-10165	3351	-3.03	0.002
D9	-21496	3380	-6.36	0.000
D10	-26170	3395	-7.71	0.000
D11	-18677	3410	-5.48	0.000
D12	-18903	3413	-5.54	0.000
D13	-24436	3406	-7.17	0.000
D14	-25887	3419	-7.57	0.000
D15	-25341	3412	-7.43	0.000
D16	-23412	3411	-6.86	0.000
D17	-21936	3378	-6.49	0.000
D18	-7216	3354	-2.15	0.032
D19	-6030	3352	-1.80	0.072
D20	-327	3357	-0.10	0.922
D21	-1631	3351	-0.49	0.626
D22	-448	3348	-0.13	0.894
D23	2286	3351	0.68	0.495
S =	22469.5			
R-Sq =	66.9%			
R-Sq(adj) =	66.4%			

Table 6 – 15: Regression P-Values Table with Seasonality in CSDS1

Table 6 – 16 shows the ANOVA analysis of “P” after adding seasonality to the regression equation.

Source	DF	SS	MS	F	P
Regression	32	2.19234E+12	68510668796	135.70	0.000
Residual Error	2151	1.08600E+12	504880048	N / A	N / A
Total	2183	3.27834E+12	N / A	N / A	N / A

Table 6 – 16: ANOVA Analysis Table for Inbound & Outbound Factors with Seasonality in CSDS1

### 6.3.2.3 Best Regression Subset Equation Prediction Variables with Seasonality

In this stage, MINITAB best subset regression uses Table 6 – 15 best “P” values prediction variables to produce the final regression equation with seasonality as shown in Table 6 – 17. The new regressed subset prediction variables will include variables with “P” values of less than 0.05 as shown in Table 6 – 18.

$$\begin{aligned} \text{REJECTED} = & (18599) + (0.353) \text{TCP\_IN} - (3.82) \text{ICMP\_IN} + (1.03) \text{UDP\_IN} - (240) \text{OTHER\_IN} \\ & - (9386) \text{D8} - (21038) \text{D9} - (25635) \text{D10} - (18355) \text{D11} - (18428) \text{D12} \\ & - (24038) \text{D13} - (25490) \text{D14} - (25051) \text{D15} - (23037) \text{D16} - (21554) \text{D17} \\ & - (6590) \text{D18} \end{aligned}$$

Table 6 – 17: Best Subset Regression Equation with Seasonality in CSDS1

Predictor	Coef.	SE Coef.	T	P
Constant	18599	1290	14.41	0.000
TCP_IN	0.353093	0.009062	38.96	0.000
ICMP_IN	-3.821	1.187	-3.22	0.001
UDP_IN	1.02969	0.02326	44.27	0.000
OTHER_IN	-239.64	35.96	-6.66	0.000
D8	-9386	2446	-3.84	0.000
D9	-21038	2483	-8.47	0.000
D10	-25635	2503	-10.24	0.000
D11	-18355	2520	-7.28	0.000
D12	-18428	2529	-7.29	0.000
D13	-24038	2516	-9.56	0.000
D14	-25490	2536	-10.05	0.000
D15	-25051	2520	-9.94	0.000
D16	-23037	2522	-9.13	0.000
D17	-21554	2494	-8.64	0.000
D18	-6590	2453	-2.69	0.007
S =	22465.9			
R-Sq =	66.6%			
R-Sq(adj) =	66.4%			

Table 6 – 18: Best Subset of Regression Factors with Seasonality in CSDS1

Furthermore, the ANOVA analysis shows an improvement in the mean squares error of the residuals obtained by the subset regression equation as shown in Table 6 – 19.

Source	DF	SS	MS	F	P
Regression	15	2.18412E+12	1.45608E+11	288.50	0.000
Residual Error	2168	1.09422E+12	504714948	N / A	N / A
Total	2183	3.27834E+12	N / A	N / A	N / A

Table 6 – 19: ANOVA Analysis of Best Subset Regression Factors with Seasonality in MINITAB Using CSDS1

### 6.3.2.4 Regression Equations Comparison

The statistical regression process has shown that the regression equation can be optimized by iterating through the regression predicting variables selection using the “P” values. In this section a comparison driven by the “MS” error of the residuals can show this improvement as shown in Table 6 – 20.

Measurement	MINITAB Regression Model		
	Without Seasonal Factors	With Seasonal Factors	Best Sub Set
R-Sq	60.9%	66.9%	66.6%
R-Sq (adj.)	60.8%	66.4%	66.4%
MS	589016067	504880048	504714948
F	376.87	135.70	288.50
DF	(9, 2174)	(32, 2151)	(15, 2168)
P	0.000	0.000	0.000

Table 6 – 20: ANOVA Analysis and Comparison Table of All MINITAB Regression Equations Using CSDS1

Table 6 – 20 shows a major drop (“MS” changed from 589016067 to 504714948) in the mean square “MS” values between the best subset of regression predictors and regression without seasonality. However, the difference in the “MS” value between the subset regression equation and regression with seasonality using all predictors is minimal (“MS” changed from 504880048 to 504714948).

Furthermore, when using the subset regression equation in Table 6 – 17 to produce regression forecasts of the rejected packets, the residuals analysis of this forecasted data shows similarity to Holt-Winter residuals as shown in Figure 6 – 10.

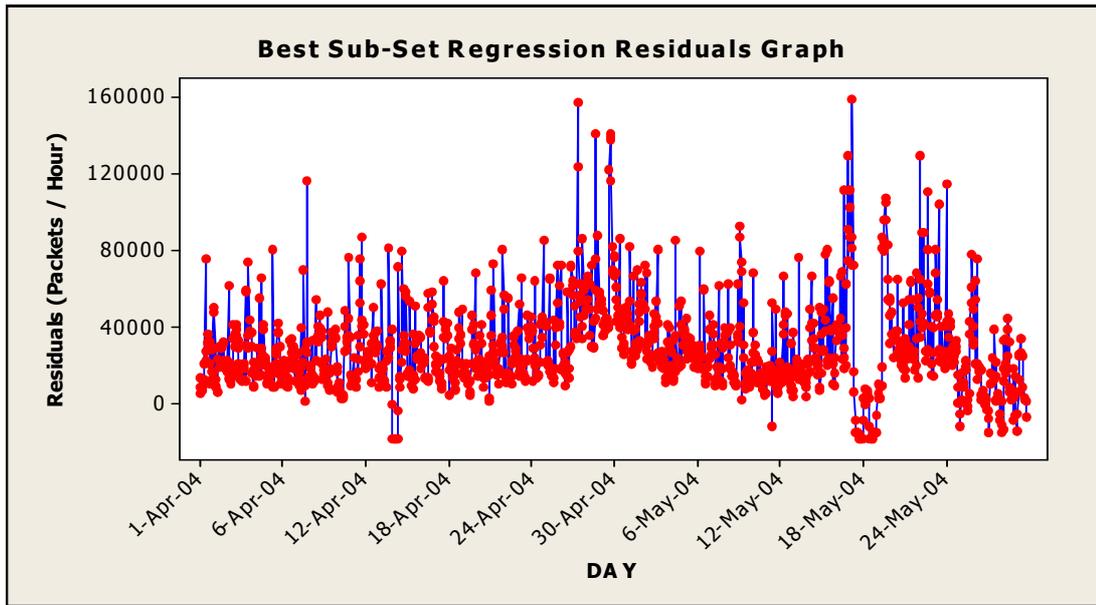


Figure 6 – 10: Sub-Set Regression Equation Residuals Plot without Seasonality for CSDS1

Comparing Figure 6 – 8 to Figure 6 – 10 shows a drop in the residuals error levels produced by Regression method (160000 packets per hour) compared to Holt-Winter method (300000 packets per hour) for the same period. This indicates that Regression forecasting is performing better than Holt-Winter forecasting.

The next section will show how neural network MLP dynamic forecasting with DoSTDM can produce better results than both Holt-Winter and Regression statistical methods.

### 6.3.3 DoSTDM Analysis & Findings from 1<sup>st</sup> Data Set (CSDS1)

Applying DoSTDM Holt-winter, Regression and MLP prediction methods to the first data set (CSDS1) has shown that dynamic MLP errors within the residuals are far better using neural network techniques. Table 6 – 21 shows a comparison of residuals errors obtained from all three methods applied to the first data set. Using Table 6 – 21, a comparison RMSE graph had been created as shown in Figure 6 – 11.

DoSTDM Training History (Weeks)	DoSTDM Residuals Error (RMSE)		
	MLP	Holt-Winter	Regression
1	6220.384276	34056.42277	22757.43136
2	3922.36462	31690.27827	24315.03489
3	3317.169665	29589.96289	26830.94604
4	4093.375772	30226.61075	26623.18885
5	4888.325075	29086.97859	41846.36266
6	3922.519306	74305.05185	45510.31731
7	4754.707233	791923.464	48481.65496
8	3048.626658	809157.5999	28920.22972
9	2352.820718	66180.52369	28376.54316
10	2629.695824	105786.0748	21428.17718
11	1836.969629	538334.7146	19644.2303
12	1530.781214	618431.7455	16899.32889
13	1255.852383	457940.8604	15891.47545
14	1091.5025	319937.2661	14915.21311
15	1270.514758	1046418.69	13982.02724
16	1426.580819	2510823.179	13490.3101
17	883.1140215	3002237.998	13161.75044
18	800.0546336	315915.2277	13352.55427
19	970.0241835	487277.6614	13087.60074
20	748.1741943	603208.6181	13062.3154
21	813.51429	518791.9035	12961.62775
<b>Full History</b>	<b>888.7749996</b>	<b>4768768.407</b>	<b>12856.06767</b>

Table 6 – 21: RMSE Comparison of All Methods by DoSTDM for CSDS1

Figure 6 – 11 shows the neural network MLP rejected packets prediction improvements with longer training history. The figure also shows that DoSTDM dynamic regression prediction factors are also doing well with longer training history. However, Holt-Winter cannot match both regression and MLP prediction as it produces high levels of RMSE errors when using longer historical training data.

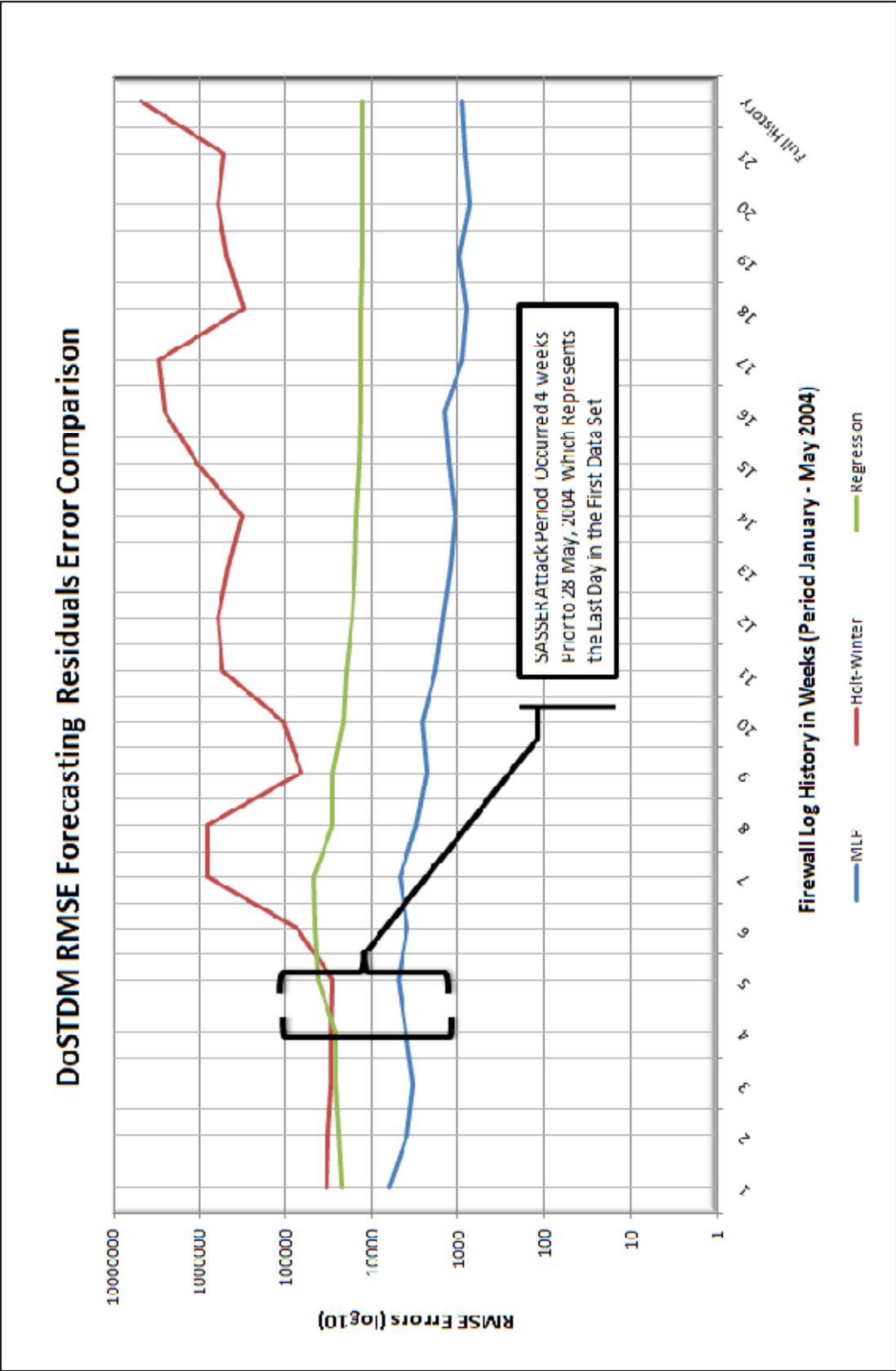


Figure 6 – 11: DoSTDM MLP vs. Holt-Winter vs. Regression for CSDS1

### 6.3.4 DoSTDM Analysis & Findings from 2<sup>nd</sup> Data Set (CSDS2)

The second data set (CSDS2) obtained from the case study network represents a full year of firewall logs. These logs were collected and analysed with DoSTDM statistical and neural network methods. During this 12 months period (January – December 2005), the case study network did not experience any attack. However, this longer base line history helped in studying the continuity of results accuracy obtained from the first dataset. Table 6 – 22 shows the RMSE analysis for that period. Using Table 6 – 21, a comparison RMSE graph had been created as shown in Figure 6 – 12.

DoSTDM Training History (Weeks)	DoSTDM Residuals Error (RMSE)		
	MLP	Holt-Winter	Regression
1	4671.72053	19258.70037	11180.35908
2	1978.436988	29920.00466	10229.19577
3	4565.80024	29330.08666	10494.52698
4	4524.41976	31854.78819	9310.299474
5	5045.231829	41081.19228	15323.97753
6	427.39604	72190.25004	14938.73722
7	809.6155466	119200.7272	14626.29411
8	201.7436492	176480.5403	13712.15351
9	416.1925135	971474.6666	13284.81777
10	828.1398332	152560.2471	12986.16279
11	345.2917245	82765.61624	12528.81416
12	1175.346066	88285.72239	12292.37855
13	599.0212155	49613.96835	12094.48093
14	1861.587897	43761.92482	11898.25684
15	248.5119045	35155.1024	11821.06577
16	245.9783188	30165.52123	11809.37841
17	1041.138159	28914.88682	11765.88096
18	379.7729036	21629.4272	11891.66694
19	1351.299128	17703.16521	12061.26577
20	877.7415622	14840.73709	12153.45465
21	857.5400136	12777.21109	12201.03834
22	1446.096052	11928.92446	12210.59238
23	1933.911322	9848.374272	12202.92065
24	1822.054747	8985.005692	12190.89192
25	715.1001387	7265.224632	12167.83754
26	140.6370921	7491.556753	12307.79287
27	577.7176358	6645.771967	12373.03278
28	655.2664852	6521.408555	12345.92486
29	1089.910489	5192.395626	12253.96661

30	1913.048832	5185.862577	12265.41964
31	726.159762	5289.646826	12213.62373
32	1147.327801	6975.884251	12128.01378
33	378.4953765	5271.007213	12186.01944
34	120.9746874	4701.56442	12166.89058
35	346.45797	4424.848604	11933.65528
36	651.3269788	4173.359708	11732.08417
37	967.4275597	236550.3405	12008.85211
38	990.2308233	12633.24381	13456.15054
39	303.6837719	11949.16504	13334.13353
40	2516.331316	10980.01157	13366.05822
41	464.162283	11417.61876	13395.076
42	3425.269795	11010.80131	13425.59194
43	195.078937	8626.580478	13421.01285
44	143.4044397	7783.55399	13439.77204
45	609.9416639	7730.855464	13464.90429
46	1126.21144	7267.581452	13513.00598
47	1240.806441	6993.220145	13509.85655
48	136.3816642	6847.981506	13542.88585
49	702.678625	7059.524358	13566.53584
50	2057.064849	6232.029919	13702.46753
51	932.0775406	7098.215594	13723.59019
52	2627.662697	5794.741165	13554.89713
<b>Full History</b>	<b>423.9247673</b>	<b>5794.741165</b>	<b>13554.89713</b>

Table 6 – 22: RMSE Comparison of All Methods by DoSTDM for CSDS2

The observations from the above table and from Figure 6 – 12 are:

- The MLP RMSE errors are dropping to the lowest level on intervals between 8 to 10 weeks. However, with 48 weeks of training data the last dropping interval was 4 weeks. The importance of this observation is that it defines the training history that can be used with MLP forecasting to be between 4 to 8 weeks at the minimum.
- The Regression dynamic statistical method of DoSTDM is shown a stable RMSE error all over the 51 weeks while the Holt-Winter dynamic statistical method of DoSTDM is shown two high spikes of RMSE errors within the residuals with 9 and 37 weeks of training history that required further investigation to find out if these are random RMSE errors or based on an unknown attack toward the case study network.

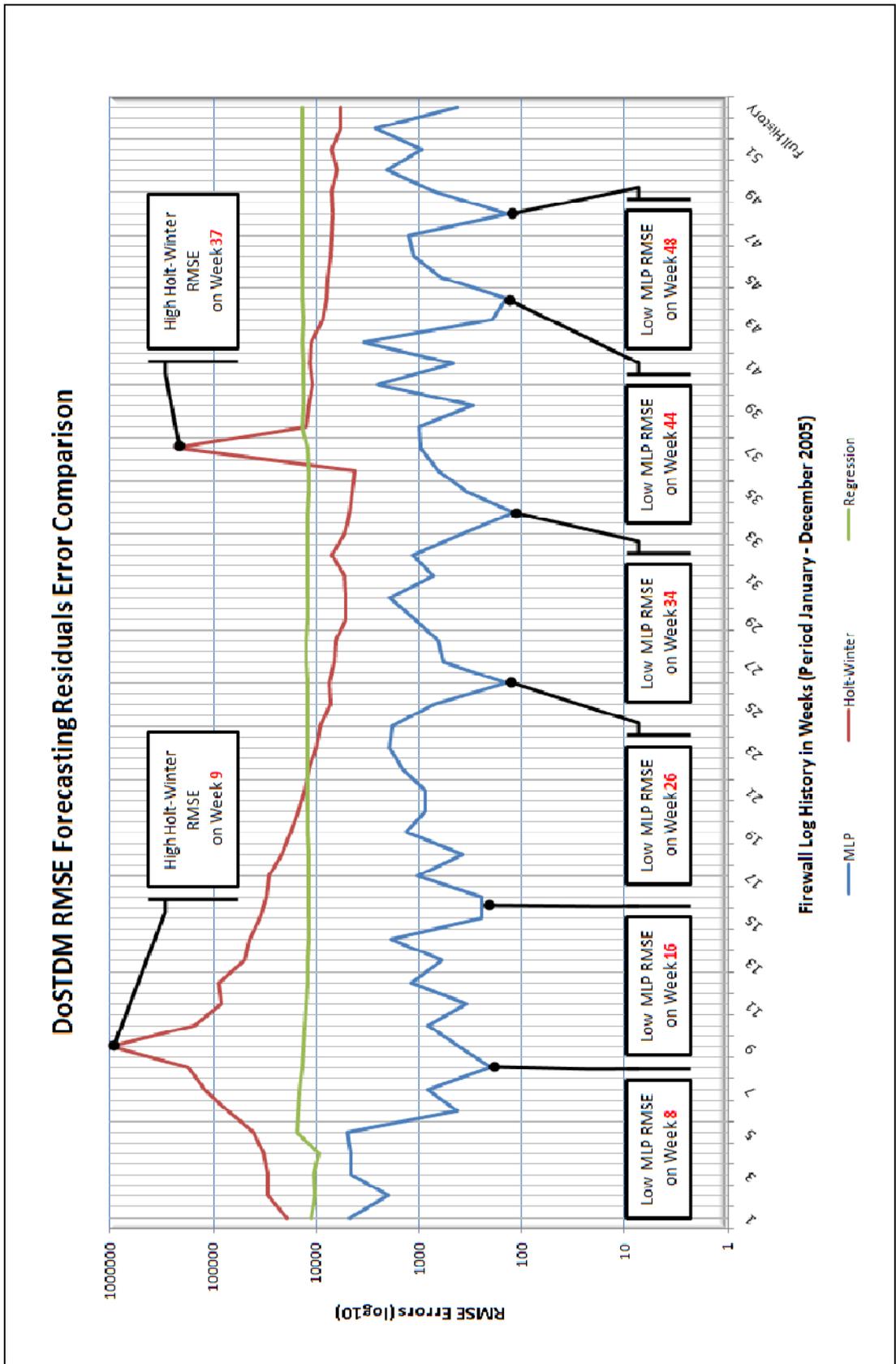


Figure 6 – 12: DoSTDM MLP vs. Holt-Winter vs. Regression for CSDS2

The significance of the previous observations from Figure 6 – 12 is that the RMSE errors are very low with MLP predictions which indicates that the MLP could detect the unknown attack with a performance nearly 10 times better than the regression method (the average RMSE errors for Regression is around the 10000 RMSE error mark while MLP RMSE errors fluctuate around the 1000 RMSE errors and below during the same period). In addition, both MLP and regression had outperformed the Holt-Winter forecasting method. Further investigation for the Holt-Winter results observed with 9 weeks of training history in Figure 6 – 12 had shown an attack that occurs during the weekend of 17<sup>th</sup> and 18<sup>th</sup> September, 2005. This attack was driven by a large amount of inbound TCP traffic and the rejected packet levels correspond to this large number of TCP packets during that period as shown in Figure 6 – 13 and Figure 6 – 14.

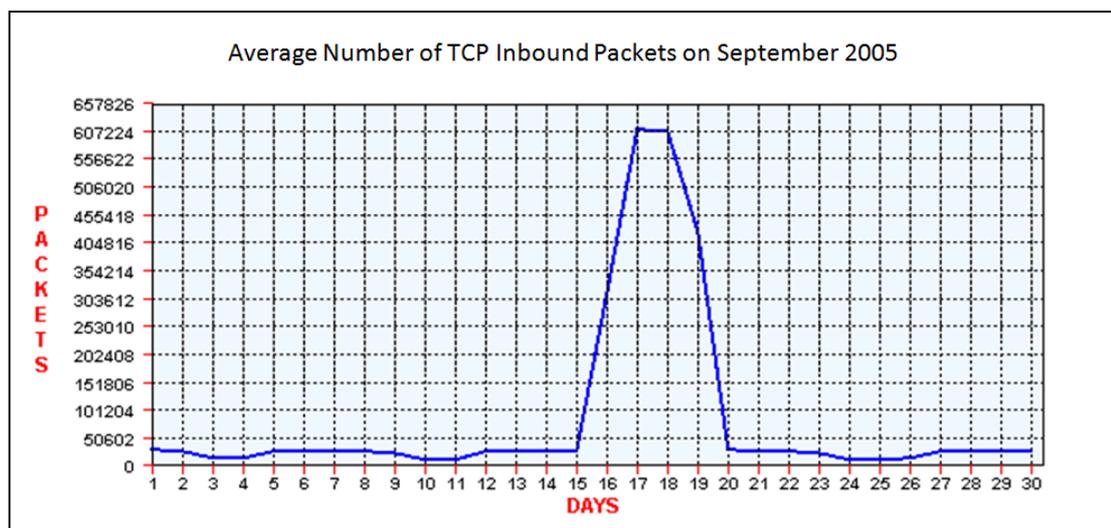


Figure 6 – 13: CSDS2 Inbound TCP Packets for September 2005

Figure 6 – 13 shows an unknown attack pattern between 15<sup>th</sup> and 20<sup>th</sup> of September 2005 using TCP protocol inbound packets with the peak of the attack occurring on the 17<sup>th</sup> and the 18<sup>th</sup> of that month. Figure 6 – 14 shows the amount of TCP rejected packets during the attack between 15<sup>th</sup> and 20<sup>th</sup> of September 2005. This indicates again the high correlation between inbound TCP packets and the level of rejected packets. Additional investigation with the Internet Storm Center (ISC) archives (source: <http://isc.sans.edu>) did not show any widespread DoS / DDoS attacks in that period. Therefore, this unknown source attack seems to be targeting the case study network only and was not a malware flooding attack like the SASSER attack.

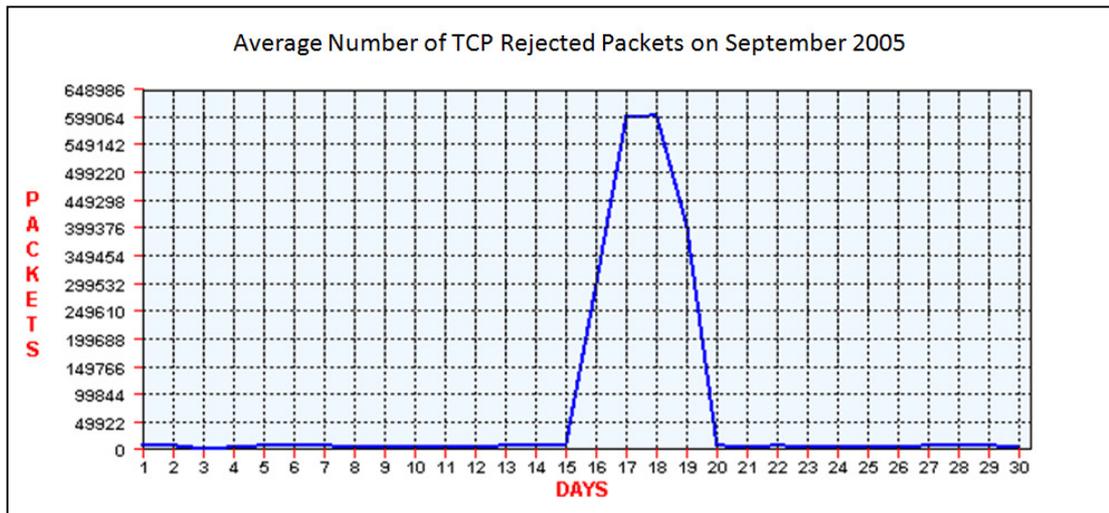


Figure 6 – 14: CSDS2 Rejected TCP Packets for September 2005

The second spike of Holt-Winter setting at 37 weeks of training history corresponds to a change in the firewall and network configuration in March – April 2005 whereby the network starts blocking certain packets and therefore, the number of rejected TCP packets has dropped in that period as shown in Figure 6 – 15.

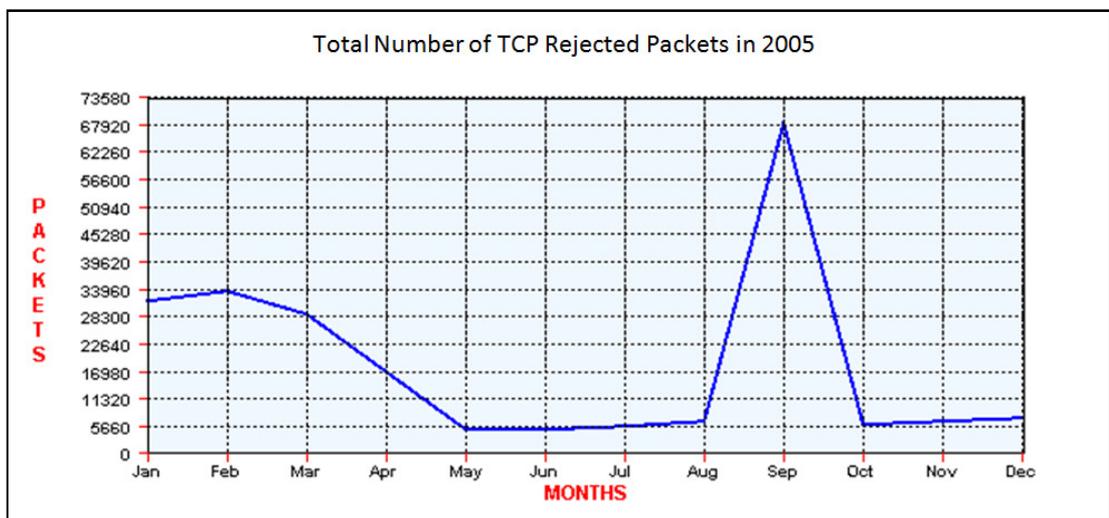


Figure 6 – 15: CSDS2 Rejected TCP Packets Over 12 Months

The modifications made to the firewall affected the results obtained with Holt-Winter; however, it did not affect both regression and MLP methods when using a long history of 37 weeks. This shows that both Regression and MLP methods had managed to adapt to changes made in the network firewall.

### 6.3.5 DoSTDM Analysis & Findings from Simulation Data Set (SNDS)

The third data set (SNDS) was obtained from the simulation network presented in Section 6.2.1 over a period of seven months between June 2010 and December 2010 and analysed with DoSTDM dynamic forecasting methods. Table 6 – 23 and the graph of RMSE in Figure 6 – 16 shows the RMSE results for all three methods using dynamic forecasting of rejected packets from 1<sup>st</sup> June to 31<sup>st</sup> December using SNDS.

DoSTDM Training History (Weeks)	DoSTDM Residuals Error (RMSE)		
	MLP	Holt-Winter	Regression
1	237.7428583	13808.03207	136.5479159
2	259.9162525	14180.57926	178.7958566
3	300.4067382	20228.37274	12591.71817
4	1672.525261	16825.0804	17554.32225
5	811.4847246	11857.17174	15231.14279
6	3238.99369	11814.25883	14177.78723
7	3897.187784	12337.60961	13992.56081
8	3535.892085	11732.49697	13859.13015
9	3059.840614	18966.35788	13556.55467
10	1827.639233	16078.24715	13265.65527
11	1356.989714	14751.24875	12941.01199
12	3392.242836	10627.59404	12423.56763
13	2055.110784	16521.18279	11937.63103
14	4548.854096	10564.57941	11342.20679
15	283.6032293	10616.88319	10911.83999
16	2483.597877	10604.98601	10618.11497
17	1938.22251	10480.33793	10282.03705
18	1761.008979	10464.74323	9602.639067
19	157.6882473	10514.31157	9871.070446
20	159.9412653	10509.20912	9724.831935
21	1558.152097	10526.35682	9573.278231
22	2571.011134	10512.40293	8926.395138
23	1349.815774	10671.89986	8778.133652
24	181.4221596	10636.74342	8604.417119
25	426.4397671	10713.01291	8461.964759
26	508.6591852	10746.87592	8364.229592
27	4378.600951	11032.71304	8154.397418
28	172.9014652	10699.27195	7427.395725
29	2413.333431	11291.15447	6845.311431
30	510.9805525	11406.81287	6630.097564
<b>Full History</b>	<b>1028.279955</b>	<b>11356.17777</b>	<b>6568.497536</b>

Table 6 – 23: RMSE Comparison of All Methods by DoSTDM for SNDS.

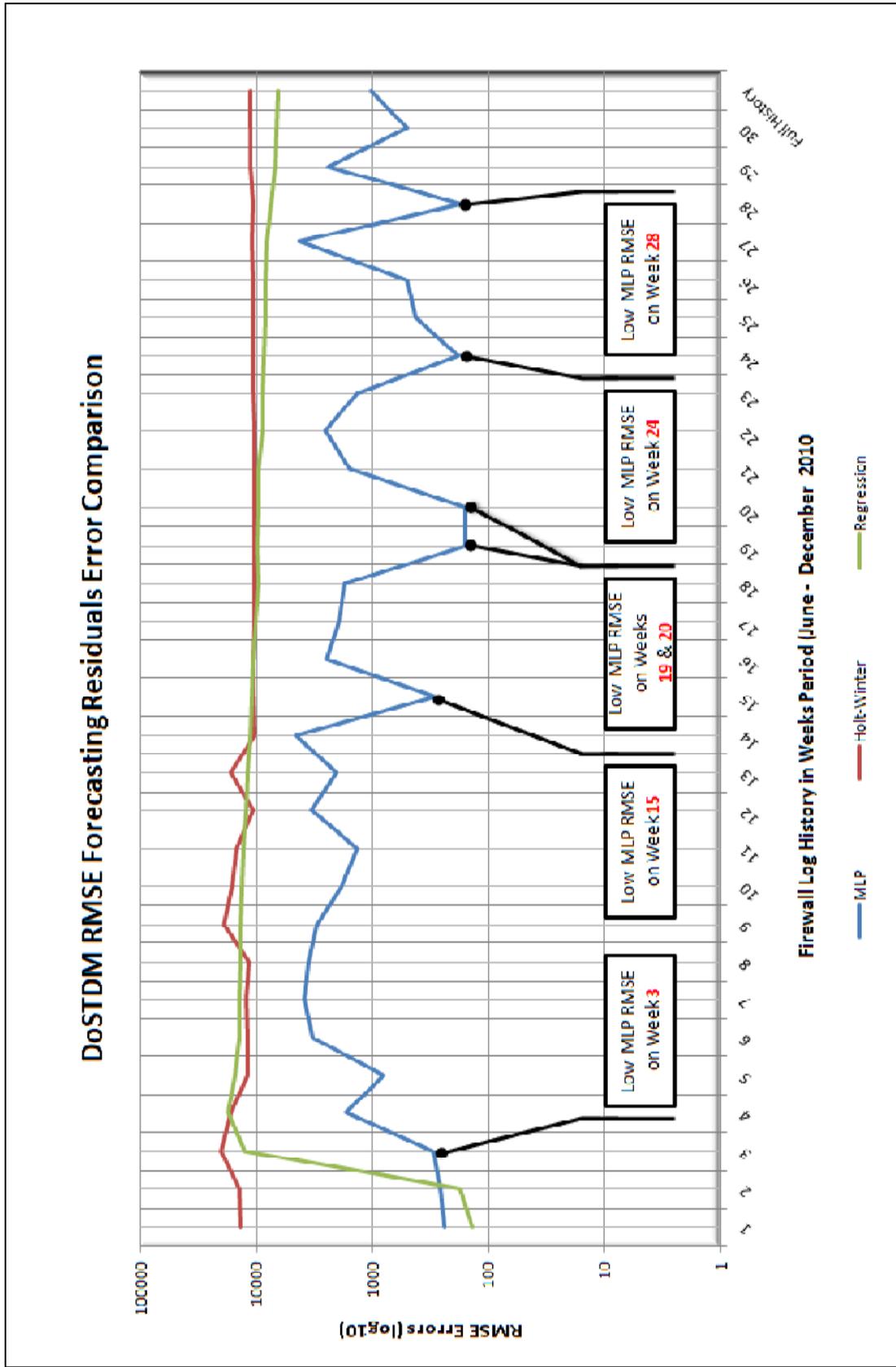


Figure 6 – 16: DoSTDM MLP vs. Holt-Winter vs. Regression for SNDS

The graphing of RMSE error residuals of SNDS data in Figure 6 – 16 shows a similarity with Figure 6 – 12 in the way MLP RMSE are low in intervals ranging 4 to 10 weeks. This will be discussed in the following section 6.4 to explain DSTPM error handling process when using MLP predictions. In addition the period using the first four weeks of history in Figure 6 – 16 has been affected with a set of simulated DoS traffic flooding attacks applied to the SNDS data set. These simulated traffic floods were created to see how the model can handle unknown attacks. The simulated attacks were made in different intervals and durations within the last 4 weeks in the SNDS data as shown in Table 6 – 24.

<b>Simulated Attack Day</b>	<b>Start Time</b>	<b>End Time</b>	<b>Attack Duration (Hours)</b>
December 4, 2010	11:56:00	12:56:00	01:00:00
December 11, 2010	11:17:00	12:30:00	01:13:00
December 17, 2010	04:25:00	09:25:00	05:00:00
December 18, 2010	01:08:00	17:55:00	16:47:00
December 27, 2010	02:25:00	22:20:00	19:55:00

Table 6 – 24: Simulated Network DoS Attacks Traffic Periods within SNDS

The simulated DoS random attacks in Table 6 – 24 affected the prediction of DoSTDM by producing higher RMSE errors of Holt-Winter and Regression forecasting residuals while the MLP RMSE error values remain at a low level as showed in Figure 6 – 16. The significance of this is that it indicates that MLP has adapt to changes in the baseline of the affected data firewall environment and produced good predictions within the window of 4 weeks of data history. The following figures show a drill down within the simulated attacks and the effect they produced on the predicted SNDS rejected packets by MLP prediction for each attacking period:

- Figure 6 – 17: SNDS 1<sup>st</sup> simulated attack on December 4, 2010
- Figure 6 – 18: SNDS 2<sup>nd</sup> simulated attack on December 11, 2010
- Figure 6 – 19: SNDS 3<sup>rd</sup> simulated attack on December 17, 2010
- Figure 6 – 20: SNDS 4<sup>th</sup> simulated attack on December 18, 2010
- Figure 6 – 21: SNDS 5<sup>th</sup> simulated attack on December 27, 2010

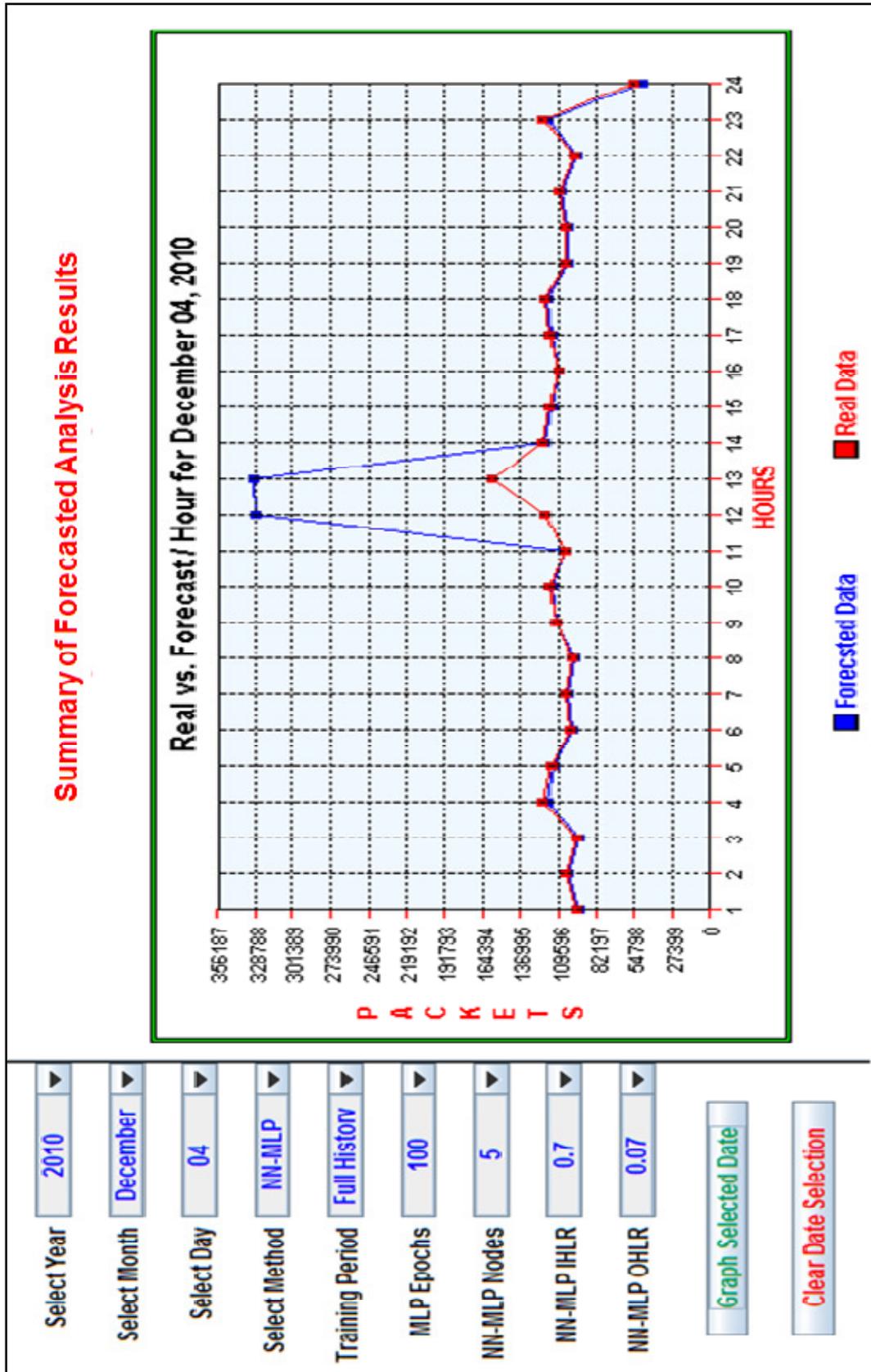


Figure 6 – 17: DoSTDM MLP Rejected Packets Prediction of the 1<sup>st</sup> Simulated DoS / DDoS Attack on December 4, 2010 for SNDS

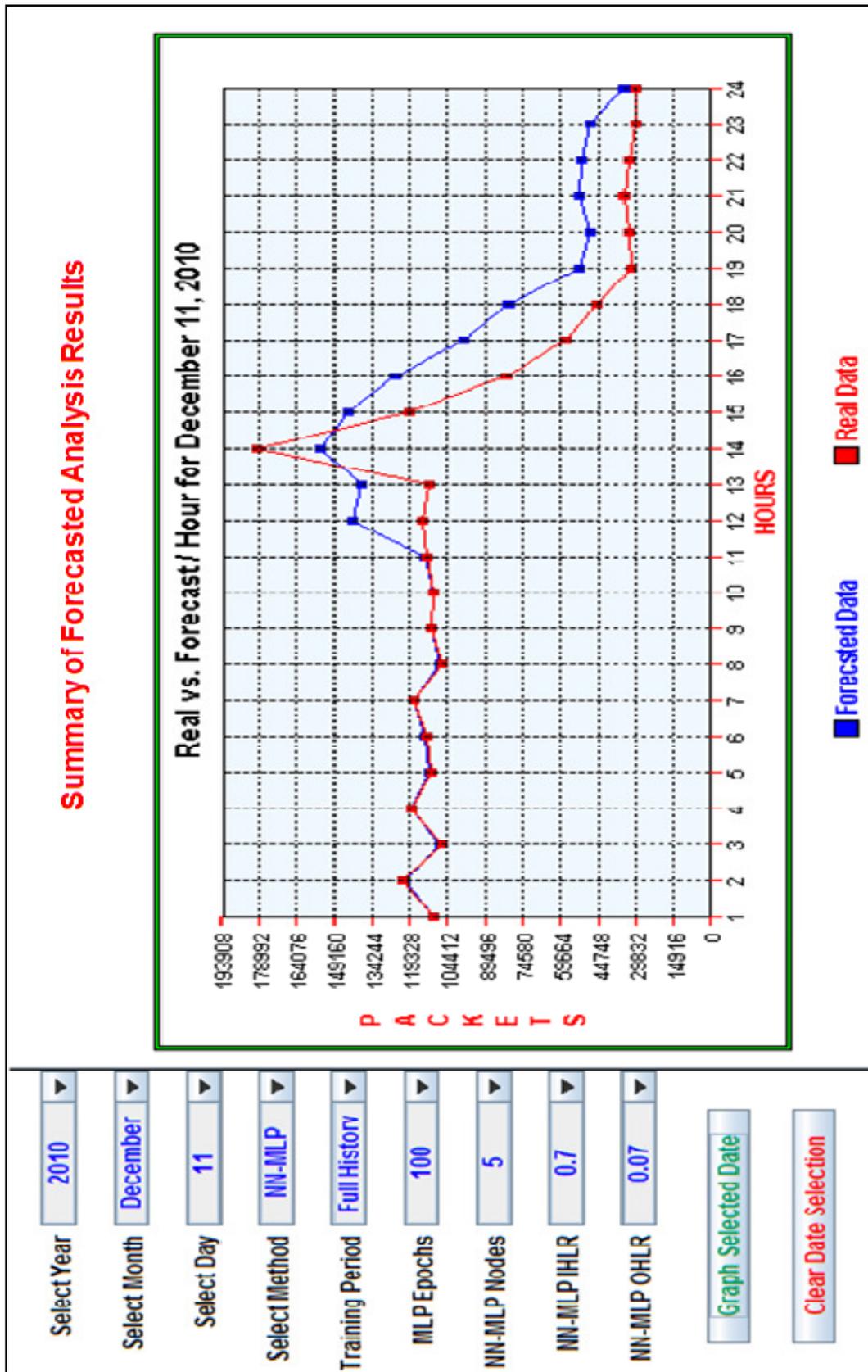


Figure 6 – 18: DoSTDM MLP Rejected Packets Prediction of the 2<sup>nd</sup> Simulated DoS / DDoS Attack on December 11, 2010 for SNDS

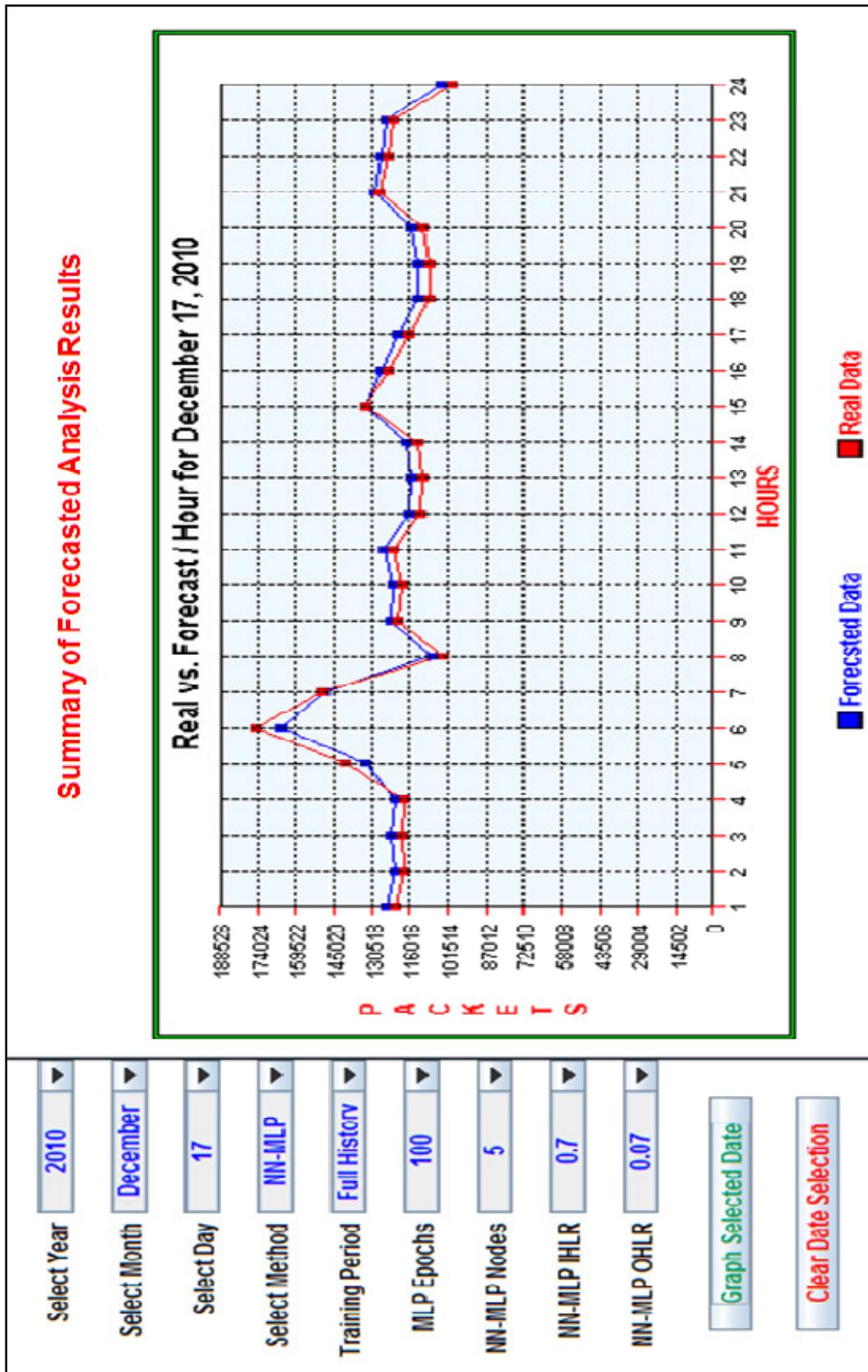


Figure 6 – 19: DoSTDM MLP Rejected Packets Prediction of the 3<sup>rd</sup> Simulated DoS / DDoS Attack on December 17, 2010 for SNDS

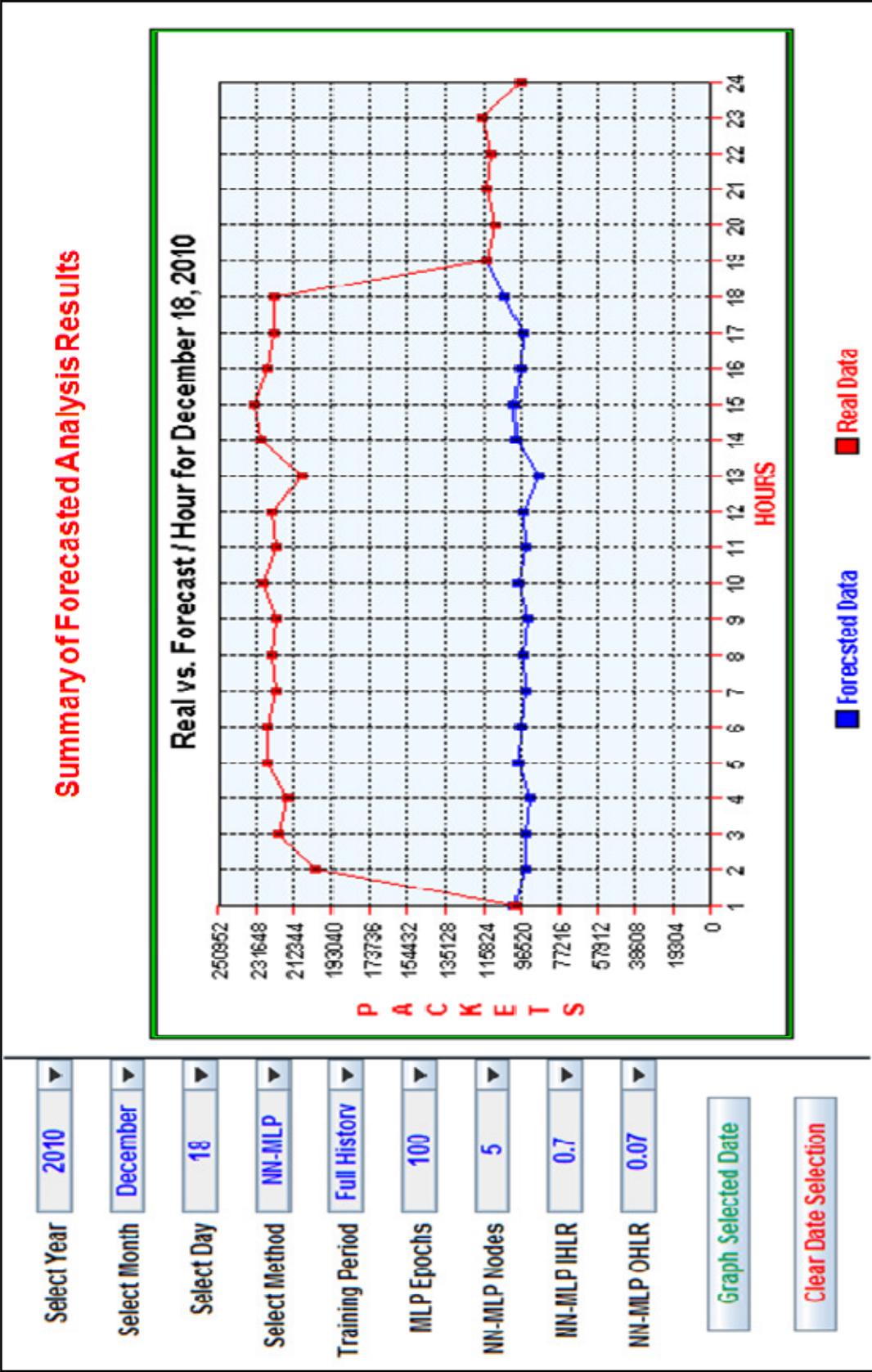


Figure 6 – 20: DoSTDM MLP Rejected Packets Prediction of the 4<sup>th</sup> Simulated DoS / DDoS Attack on December 18, 2010 for SNDS

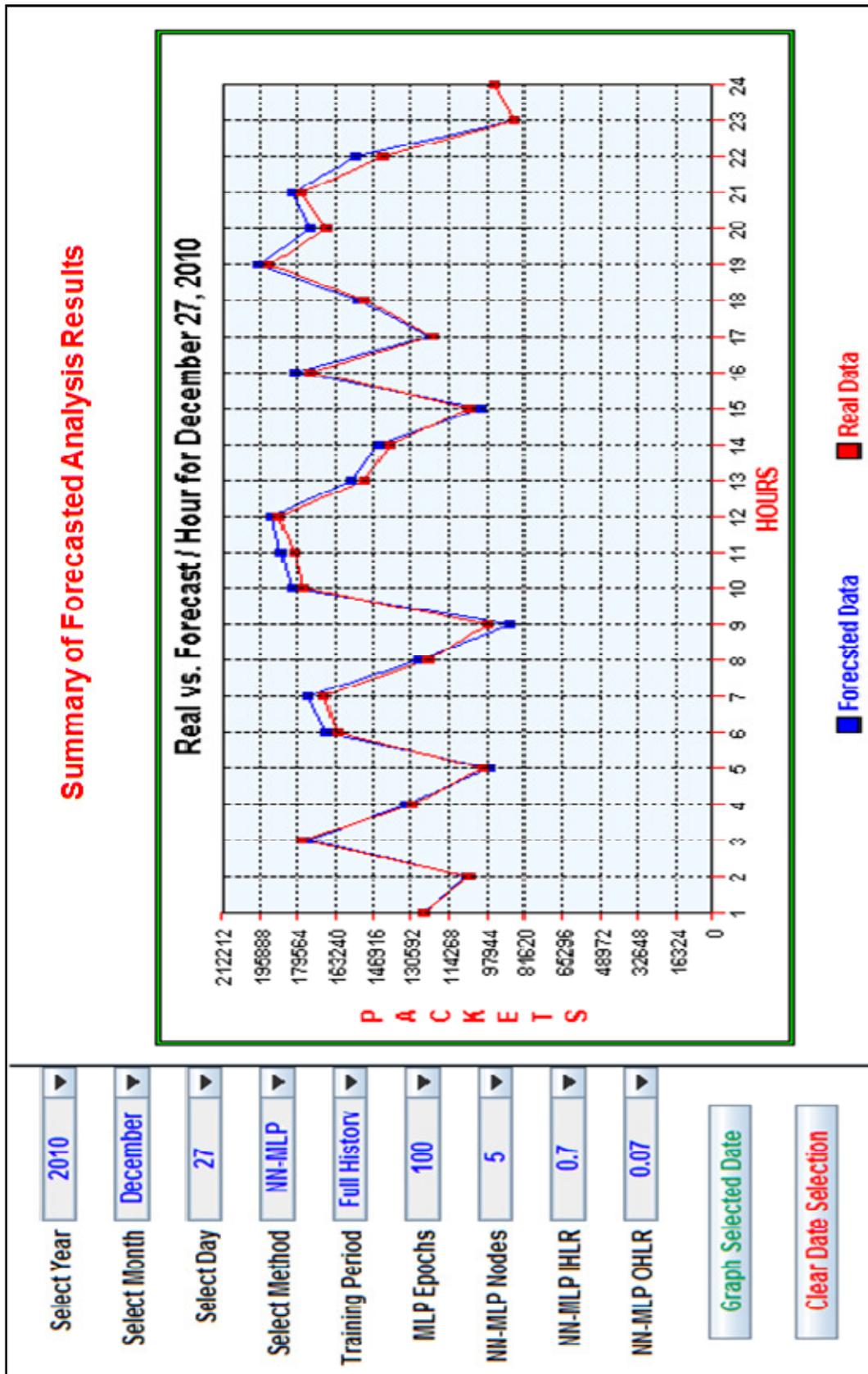


Figure 6 – 21: DoSTDM MLP Rejected Packets Prediction of the 5<sup>th</sup> Simulated DoS / DDoS Attack on December 27, 2010 for SNDS

#### 6.4 DoSTDM THRESHOLD MANAGEMENT

DoSTDM model has illustrated that MLP can produce good predictions as shown from the CSDS1, CSDS2 and SNDS data. However, in some cases like Figure 6 – 20 (SNDS December 18, 2010), it shows that the predictions were well below the real values of rejected packets. This specific case can be explained by the seasonality within the data that is represented by 24 hours intervals since an attack was launched toward the simulation network in less than 24 hours before December 18, 2010 (i.e. on December 17, 2010 from 4:25AM to 9:25AM for 5 hours as shown in Figure 6 – 19). Therefore, DoSTDM MLP retained the rejected packets threshold set to the same value of December 17, 2010 since MLP had considered that the attack is a continuation of the previous day attack.

In addition the RMSE results obtained in the previous section of this chapter has shown that the lowest RMSE values are obtained in intervals ranging between 4 to 10 weeks. Therefore, the following is recommended to reduce RMSE errors in predictions:

- DoSTDM MLP baseline data history should be 4 weeks at the minimum.
- DoSTDM MLP using full history of training data can identify unknown DoS / DDoS attacks, however, it might over-smooth the predicted level of rejected packets.

In order to verify the 4 – 10 weeks interval impact on the prediction, the prediction training period of DoSTDM MLP has been changed the on December 18, 2010 from a full history of 30 weeks (June – December 2010) to 10 weeks only which is showing that the MLP predictions are still doing better in detecting the DoS / DDoS 4<sup>th</sup> simulated attack as shown in Figure 6 – 22 compared to prediction Figure 6 – 20 from the previous section. Therefore, with the rejected packets prediction thresholds been dynamically re-assigned by MLP it is critical to observe the model behavior when consecutive attacks occur to avoid threshold overlapping between these attacks and consider lowering the training history period during these consecutive attacks.

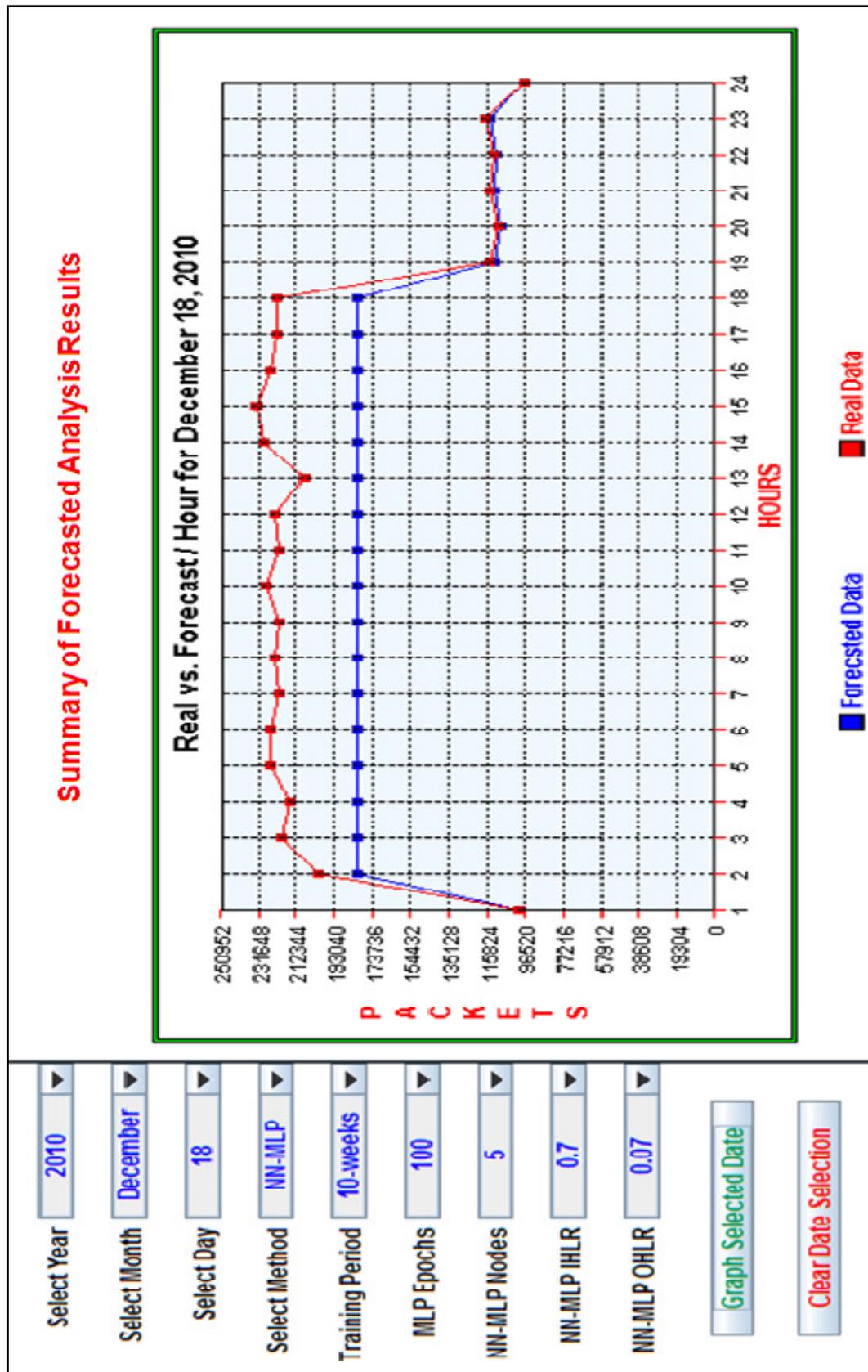


Figure 6 – 22: DoSTDM MLP Rejected Packets Prediction of the 4<sup>th</sup> Simulated DoS / DDoS Attack on December 18, 2010 for SNDS with 10 Weeks Training History Only

## 6.5 CHAPTER CONCLUSION

The chapter implemented the DoSTDM model using the design presented in Chapter 5 to provide an accurate DoS / DDoS attack detection model using different firewall logs from three data sets. The MLP proposed engine in Chapter 5 has been implemented and tested overall data sets.

This chapter also triangulated output results from the first and second data sets between Holt-Winter, Regression and MLP error analysis using RMSE to provide solid findings about the use of Neural Network models. These findings have shown a consistent ability for MLP to detect the initiation of known and unknown attacks. However, it also highlights the importance of baseline and training history values to avoid rejected packets prediction over-smoothing.

Therefore, the major finding of this chapter is that early tracking of the DoS with MLP can be achieved as long as a minimum of 4 weeks of history is available to train the MLP network. However, it has no problem detecting with larger historical data samples. The type of the firewall computer network large or small doesn't affect this 4 week minimum history condition and having more data can further enhance the detection of MLP. This finding helps in generalising the model implementation and detection findings to any firewall network irrelevant to the size of the network.

The DoSTDM MLP performance is highly coupled with the way the neural network calculate the hidden nodes weight within the implemented MLP algorithm (please see MLP pseudo code function `Hidden_Nodes_Weight_Generator` in Table 6 – 7).

The algorithm uses a random number seeding to initialise the MLP neural network hidden nodes weight, this changes with every MLP re-initialisation of these nodes. The effect of random seeding within MLP can be observed if the forecasted had been repeated for the same day, the results would be a fluctuated MLP RMSE level in comparison to Holt-Winter and Regression methods. These fluctuations in MLP RMSE are minimal but are mainly affected by the random seeding of the MLP nodes weigh as shown in Table 6 – 25.

<b>Forecasting Method</b>	<b>RMSE Readings Using Full History of the Data to Forecast the Same Day</b>								
	<b>1<sup>st</sup></b>	<b>2<sup>nd</sup></b>	<b>3<sup>rd</sup></b>	<b>4<sup>th</sup></b>	<b>5<sup>th</sup></b>	<b>6<sup>th</sup></b>	<b>7<sup>th</sup></b>	<b>8<sup>th</sup></b>	<b>10<sup>th</sup></b>
<b>MLP</b>	5112	4244	4949	4026	5732	4132	4118	4121	4528
<b>Holt-Winter</b>	16023	16023	16023	16023	16023	16023	16023	16023	16023
<b>Regression</b>	15136	15136	15136	15136	15136	15136	15136	15136	15136

Table 6 – 25: MLP Seeding Effect on RMSE with DoS Attacks  
(Readings for December 17<sup>th</sup> 2010 Forecasts from SNDS)

Table 6 – 25 shows the seeding effect of hidden node random weights initialization on MLP RMSE levels, however, the results are consistent in the way that it's always lower than the statistical methods. The MLP forecasting RMSE errors are still within the same range and did not represent any major concerns to the overall performance of the model. The table also shows that statistical method forecasts RMSE are consistently showing Regression performing better than Holt-Winter but less accurate than MLP predictions.

The next chapter will discuss the research contribution and the DoSTDM model implementation limitations and future work to address these limitations.

# CHAPTER 7

## RESEARCH DISCUSSION & CONCLUSIONS

This chapter discusses the research findings and contribution of the DoSTDM model and how it can be extended with future research initiatives. It also covers any limitations observed during the implementation and analysis of the data. In addition, this chapter discusses future research direction for DoS / DDoS problem detection and protection.

Chapter seven is divided into four main sections:

- Section one discusses DoSTDM research findings observed from Chapter 6 and relate these findings to research questions.
- Section two examines the research contribution to network security and how it can be extended to protect from DoS / DDoS and attacks.
- Section three examines DoSTDM model design methodology, data collection, analysis and limitations.
- Section four discusses future research directions within the field of DoS / DDoS to enhance detection techniques.

## 7.1 RESEARCH FINDINGS DISCUSSION

This research to develop the DoSTDM model has shown the importance of historical data and the use of this data to predict future trends of DoS / DDoS attacks. The model worked in the same way a human immune system would work when finding that an abnormality had occur (i.e. a network traffic flooding form of DoS / DDoS attacks) in comparison to previous known information (i.e. the network firewall baseline data).

This research targeted the DoS / DDoS problem using a three dimensional approach when studying the causal relationships of protocol activity patterns within the data. This three dimensional approach consists of: the protocol type, the protocol direction and the protocol status upon passing through the firewall. Previous work captured the some of these dimensions but not all of them in one approach to resolve the problem. The foundation stone of the research was the data itself with the activity patterns stored in the firewall logs being used to mitigate the problem domain of DoS / DDoS. Most of the previous research described earlier in Chapter 2, addressed different sides of the problem domain by targeting: the contents of the packets, over complicated detection models and applying rules based solutions. However, the DoSTDM targets the problem by simplifying the detection method (i.e. one rule only to track the anomalies in the packet that contribute to a high level of rejected packets by the firewall).

The final outcome of this research is a DoS / DDoS tracking and prediction model that can predict DoS / DDoS flooding attacks and triangulate results between multiple detection methods. The model has made a good use of dynamic assignment of rejected packet levels in the data and provided consistent results between the case study network and the simulation network.

Using the data structures identified by this research (as described in Chapter 4) and the spiral model designed to develop the DoSTDM model (as per Chapter 5 design) the research addressed the problem of DoS / DDoS and answered the questions as identified in Chapter 3 of this thesis.

The first research question was addressed by the dynamic model approach to discover and identify the attacks as an anomaly within the data predictions using baseline data as a training input for the neural network model. The tracking of the baseline data of the firewall logs has shown that the model can be tailored to nearly every network without having to accommodate an IDS / IPS separate system (either hardware or software implementations of IDS / IPS ).

The second research question was answered by observing the RMSE errors as the predictions accuracy measure to define the amount of training history that can be used with model (i.e. 4 to 10 weeks of baseline history). However, this specific amount of history will be different from one network to another as the sensitivity of the model relies on dynamics of the network (i.e. small to medium scale networks in comparison to large networks). Therefore, it is recommended to use the full history of the baseline as a starting point then tune the history in weeks based on the dynamical of traffic activities of the network were the model is implemented.

The findings from Chapter 6 can be summarised as follow:

- The DoSTDM can track and predict new attacks using MLP and multiple linear regression
- The MLP neural network produces low RMSE errors and can use training history data that is ranging from 4 week of history to full history.
- The statistical methods forecasting of rejected packets using multiple linear regression produces good results only with longer training periods of history but still produces more RMSE errors than MLP neural network.
- The Holt-Winter statistical forecasting produces good results only with short history and still produces higher RMSE errors than both regression and MLP.

Overall, the research has achieved its aimed objective to produce a simple, portable and independent model that can be applied to small, medium and large firewall implementations and can be easily tuned for each network as it is based on data mining within the network history.

## **7.2 RESEARCH CONTRIBUTION**

The research main contribution is the use of a quantitative approach to study empirical data collected from internet connected networks firewall logs to improve the detection of new attacks toward these networks.

The research helped in understanding the relationship between traffic patterns and how they can be used to address the problem of DoS / DDoS flooding attacks detection techniques. However, the research also contributed to other parts of academic research as shown in the following sub sections.

### **7.2.1 Contribution to Theory**

Researching in the area of firewall patterns adds to the existing theoretical research within information networks security and data protection using IDS and IPS products developed based upon the Human Immune System (HIS) and Artificial Immune Systems (AIS) theories. These theories are adapted from the human biology system and how this system can provide protection based on pattern recognition of antigens and antibodies within the human body, it also adapts the concept of self and non-self activities discrimination to define an attack.

The majority of IDS and IPS systems are based on pattern matching and decision roles that help in identifying an attack. However, the attack patterns need to be known first in order to configure the firewall, IDS and / or IPS rules and policies to stop an attack. Therefore, using the HIS analogy in the DoSTDM has provided a means to accommodate past pattern learning base lines as the decision making rule for protecting against new DoS / DDoS flooding attacks.

Research within the firewall performance domain shows that security, performance reliability and easy management tools of firewalls are the major factors behind selecting firewall products to be implemented. Therefore, the solution needs to start from the firewall without any interaction from neither IDS nor IPS in order to see the DoS / DDoS attack pattern without any partial filtration that might be applied from the existence of either IDS or IPS.

Knowledge about firewalls and how they can be improved needs to be extended further to compare different abilities of firewalls products to protect a new networked infrastructure at the design phase.

Studying attack patterns can be beneficial to any firewall implementation due to its great dependency on TCP / IP protocol suite including User Datagram Protocol (UDP) and Internet Control Message Protocol (ICMP) packets. Using these protocols to perform an attack might not be successful in hacking the network data, however, it can be used to generate a DoS / DDoS flood of packets.

In addition research within security vulnerabilities shows that new dimensions of security measures need to be explored to maintain network security far beyond traditional known techniques of finding vulnerabilities on a one by one basis. As network administrators find it too hard to predict every undefined attack, firewall patterns can provide a reliable measure for early prevention against DoS attack or even a mix of network attacks types at the same time.

By understanding how firewall attack patterns are structured and analysing such patterns, the firewall response to external and internal attacks can be enhanced. Therefore, gathering more knowledge about these attacks will provide better protection to existing and future computer networks from new attack forms that keep evolving.

This research has explored the possibility of predicting DoS / DDoS attacks in a modular way that might be used later by a new type of IDS or IPS implementations to stop the attacking traffic before it reaches the network.

The neural network aspect of this research has shown that the MLP neural network can perform well and with the advancing computer speed the computational overhead can help improving the use of this approach to mitigate the DoS / DDoS problem. The neural networks used in this research were used as a classifier and predictor at the same time. It classifies the input protocol activities and actions and predict the rejected packets level as a function of these protocols activity patterns.

The theoretical contribution of this research is represented by extending the theory domain to cater for past data as well as new data patterns that represent DoS / DDoS attack in the context of this research.

In summary, the most important contribution to the theory delivered by this research is the ability to blend and scaffold on multiple complex theories like ANN theory, AIS theory, HIS theoretical analogy, statistical analysis, statistical forecasting methodologies and simulation approach to produce a very simple theory to solve the DoS / DDoS problem. This research present this simpler theory in the form of a single statement: *the denial of service attack can be detected and measured using a single attack indicator represented by the rejected packets traffic level for a firewalled network under that attack.*

### **7.2.2 Contribution to Practice**

In the present time every network can be a target for an attack either directly by a hacker or even by an internal relay agent program. Even if these attacks cannot reach the information and cause damage to the system, it can indirectly cause DoS / DDoS attacks due to the large amount of packets flowing into the network.

Therefore, detecting attacks as early as they start can save a lot of effort and resources. From this point of view the practical, the significance of having a logical analysis and understanding of past firewall logs can be observed. However, all these tracing activities don't show what sorts of reactions have been taken against the attacks or what has been done with the data patterns.

Data patterns can help in a practical way if and only if they have been analysed within a short amount of time. It will be useless to do such analysis after identifying the attack the hard way when all network services became unusable. Valuable information such as DoS / DDoS data patterns can be obtained by combining firewall log information with statistical analyses and neural networks predictions using data mining techniques to defend networked resources.

Within the practical domain, a set of goals can be achieved if past firewall attack patterns are used effectively to protect network hosts and any associated network applications. These goals can be summarised in three major points, the first is to provide a high level of network availability in the long term to reduce the risk of DoS / DDoS attacks. The second is establish a more reliable firewall policies and configuration rules based on daily baseline pattern comparison to provide better network protection. The third is creating a proactive firewall management practice at low cost. Therefore, practically this research enhances the way the information systems industry can use this valuable data as a proactive security measure rather than just store it within a database for auditing reasons (as most information systems organisations do without any additional benefits).

Finally, the most significant contribution to the practical side of detecting DoS / DDoS is use of the KISS principle to simplify the detection problem whereby DoSTDM uses only one rule to trace the rejected packets as an indicator to these attacks to be traced (in comparison to existing commercial products within the IDS / IPS software and hardware market)

### **7.3 RESEARCH LIMITATIONS**

The models performance was reliable and consistent; however, some limitations were identified. These limitations are driven from the work carried in both Chapter 4 and Chapter 6. The first of these limitations is that the research data sets were only limited to one real world case study and one simulation network. This did not show any major issues, however, more case studies will help in providing a more generalised picture of the DoSTDM behaviour. This limitation has been mitigated by the fact that neural network algorithm results can be generalised in comparison to statistical computing methods. Another mitigation to this limitation was the data nature itself, although the research used three data sets (CSDS1, CSDS2 and SNDS), the data within each data set used to train and test the DoSTDM model contains millions of packet records and each packet represents a micro case study of it is own representing the packet path through the network firewall.

The second observed limitation is when two consecutive attacks occur within the seasonality period (i.e. within 24 hours), the model kept the previous rejected value of the threshold as explained earlier in section 6.4 when using full training history from the baseline data. The solution was found by reducing the training period under these circumstances. This limitation can be addressed by triangulating between multiple neural networks and multiple training data lengths and use averaging of the predicted level of rejected packets to detect the DoS / DDoS flooding attacks as explained in the context of this research.

The third limitation is related to the neural network MLP algorithm whereby the nature of the algorithm initiates the weights of the hidden neurons connection to a random value affecting the prediction. This, as shown in Chapter 6, affects the RMSE errors. Such a limitation (like the second limitation) can be addressed by another neural detection engine results triangulation.

Finally the simulation network implementation as shown in Chapter 6 was very basic and it could be argued that a larger scale simulation network should be used to address the network size and therefore the activities generated by such network under simulation.

#### **7.4 FUTURE RESEARCH DIRECTION**

The proposed model can be extended using new hardware processing capabilities to triangulate the predictions between multiple neural network methods like existing MLP method and other neural methods such as Radial Basis Function (RBF) neural networks and Artificial Intelligent (AI) systems.

In addition, the use of the Artificial Immune Systems (AIS) intuitive approach to solve the self and non-self behaviour for systems security is still emerging and additional research to gain more knowledge is still needed to provide computer immunological protection.

Future work on DoS / DDoS is still needed as no matter what method the attack is formed with, the result is always the same.

Using additional prediction methods and romantic decision support techniques (Leigh et al 2002) can provide a more powerful detection system especially with the introduction of the new Internet Protocol version 6 (IP6) that incorporate security measures at the packet levels. Therefore, future research is aiming toward achieving the following goals:

1. Enhance the detection capabilities by using a hybrid neural network engine with AIS approach with unsupervised learning capabilities.
2. Reduce the detection engine training history and therefore, reduce processing and memory overheads.
3. Incorporate more real world data sets for training and testing the detection model.
4. Implement the DoSTDM design principle as a pluggable model within the network firewall for online detection.
5. Leverage the detection capabilities from detecting flooding attacks to other forms of intrusion attacks.

Finally, I can conclude this research by emphasising the fact that it is still a valuable option for academic researchers to continue the investigation in the field of DoS / DDoS detection to advance the computerised network security and availability.

## REFERENCES

- Al-Haj, S., and E. Al-Shaer. 2011. *Configuration Analytics and Automation (SAFECONFIG), 2011 4th Symposium on, Oct. 31 2011-Nov. 1 2011: Measuring firewall security.*
- Alexander, P. M. 2002. Towards reconstructing meaning when text is communicated electronically. PhD, Informatics, University of Pretoria, Pretoria. University of Pretoria Electronic Theses and Dissertations (accessed March 3, 2008).
- Allan, A. 2003. Intrusion Detection Systems: Perspective. Technology Overview. Gartner Research Technology Overview No. DPRO-95367.
- Alsaleh, O. I., and A. A. Alfantookh. 2007. Improved Detection System of Denial of Service Attack. In *Proceedings of the 3rd IEEE Gulf Conference (IEEE-GCC 2007)*. IEEE Computer Society.
- Becker, J., B. Niehaves, and K. Klose. 2005. A Framework for Epistemological Perspectives on Simulation. *Journal of Artificial Societies and Social Simulation* 8 (4).
- Boehm, B. 1986. A spiral model of software development and enhancement. *SIGSOFT Softw. Eng. Notes* 11 (4): 14-24.
- Bolzoni, D., E. Zambon, S. Etalle, and P. Hartel. 2006. Poseidon: a 2-tier Anomaly-based Network Intrusion Detection System. In *Fourth IEEE International Workshop on Information Assurance, IWIA 2006*. London, UK. IEEE Computer Society.
- Cardoso, L. S. 2007. Intrusion Detection Versus Intrusion Protection. In *Network Security: Current Status and Future Directions*, ed. C. Douligeris and D. N. Serpanos: John Wiley & Sons.
- Carl, G., R. R. Brooks, and S. Rai. 2006a. Wavelet based Denial-of-Service detection. *Computers & Security* 25 (8): 600-615.
- Carl, G., G. Kesidis, R. R. Brooks, and R. Suresh. 2006b. Denial-of-service attack-detection techniques. *Internet Computing, IEEE* 10 (1): 82-89.
- Cavusoglu, H., B. Mishra, and S. Raghunathan. 2004. A model for evaluating IT security investments. *Commun. ACM* 47 (7): 87-92.
- CERT Australia. 2012, Cyber Crime and Security Survey Report 2012. <http://www.canberra.edu.au/cis/storage/Cyber%20Crime%20and%20Security%20Survey%20Report%202012.pdf>
- Chang, R. K. C. 2002. Defending against flooding-based distributed denial-of-service attacks. *Communications Magazine, IEEE* 40 (10): 42- 51.

- Chen, L.-C., T. A. Longstaff, and K. M. Carley. 2004. Characterization of defense mechanisms against distributed denial of service attacks. *Computers & Security* 23 (8): 665-678.
- Chen, Y., and K. Hwang. 2006. *Collaborative Technologies and Systems, 2006. CTS 2006. International Symposium on, Collaborative Change Detection of DDoS Attacks on Community and ISP Networks*.
- Cheng, T.-H., Y.-D. Lin, Y.-C. Lai, and P.-C. Lin. 2012. Evasion Techniques: Sneaking through Your Intrusion Detection/Prevention Systems. *Communications Surveys & Tutorials, IEEE* 14 (4): 1011-1020.
- Cho, S.-B., and H.-J. Park. 2003. Efficient anomaly detection by modeling privilege flows using hidden Markov model. *Computers & Security* 22 (1): 45-55.
- CSI 2011. 15<sup>th</sup> Annual 2010/2011 Computer Crime and Security Survey. <http://reports.informationweek.com/abstract/21/7377/Security/Research:-2010/2011-CSI-Survey.html?download=true&fwp=true#error>
- De Castro, L. N., and J. Timmis. 2002. *University of Paisley, Artificial Immune Systems: A Novel Paradigm to Pattern Recognition*. citeulike-article-id:3757143 (accessed February 13, 2011)
- Dooley, K. 2002. Simulation Research Methods. In *Companion to Organizations*, ed. J. Baum, 829-848. London: Blackwell.
- Durst, R., T. Champion, B. Witten, E. Miller, and L. Spagnuolo. 1999. Testing and evaluating computer intrusion detection systems. *Commun. ACM* 42 (7): 53-61.
- Eisenhardt, K. M. 1989. Building Theories From Case Study Research. *The Academy of Management Review* 14 (4): 532-550. (JSTOR February 29, 2008).
- Escamila, T. 1998. *Intrusion detection: network security beyond the firewall*. New York: John Wiley & Sons Inc.
- Estevez-Tapiador, J. M., P. Garcia-Teodoro, and J. E. Diaz-Verdejo. 2004. Anomaly detection methods in wired networks: a survey and taxonomy. *Computer Communications* 27 (16): 1569-1584.
- Estevez-Tapiador, J. M., P. Garcia-Teodoro, and J. E. Diaz-Verdejo. 2005. *Computers and Communications, 2005. ISCC 2005. Proceedings. 10th IEEE Symposium on, 27-30 June 2005: Detection of Web-based attacks through Markovian protocol parsing*.
- Feinstein, L., D. Schnackenberg, R. Balupari, and D. Kindred. 2003. *DARPA Information Survivability Conference and Exposition, 2003. Proceedings, Statistical approaches to DDoS attack detection and response*. vol. 1, 2003, IEEE CS Press, pp. 303-314.

- Fitzgerald, B., and D. Howcroft. 1998. *Proceedings of the international conference on Information systems, Competing dichotomies in IS research and possible strategies for resolution*. Helsinki, Finland: Association for Information Systems (accessed January 10, 2008).
- Fitzgerald, G., R. A. Hirschheim, E. Mumford, and A. T. Wood-Harpor. 1985. Information Systems Research Methodology: An Introduction to the Debate. In *Research Methods in Information Systems*, ed. G. Fitzgerald, R. A. Hirschheim, E. Mumford and A. T. Wood-Harpor, 1-7. Amsterdam: North-Holland.
- FitzGerald, J., and A. Dennis. 2007. *Business Data Communications and Networking*. 9th ed: John Wiley & Sons.
- Galliers, R. D. 1991. Choosing appropriate information systems research approaches: a revised taxonomy. *Information Systems Research: Contemporary Approaches and Emergent Traditions*: 327-345. (Curtin University Library & Information Service E-Reserve February 29, 2008).
- Ghosh, A. K., and A. Schwartzbard. 1999. A study in using neural networks for anomaly and misuse detection. In *Proceedings of the 8th conference on USENIX Security Symposium - Volume 8*. Washington, D.C. USENIX Association.
- Globa, L. S., Y. A. Demidova, and M. Y. Ternovoy. 2006. *Microwave and Telecommunication Technology, 2006. CriMiCO '06. 16th International Crimean Conference, Sept. 2006: Network Anomaly Detection using Neural Networks*.
- Goh, V. T., J. Zimmermann, and M. Looi. 2010. Detecting attacks in encrypted networks using secret-sharing schemes. *International Journal of Cryptology Research* 2 (1): 89-99.
- Golubev, V. 2005. DoS attacks: crime without penalty, Computer Crime Research Center: <http://www.crime-research.org/articles/1049/2> (accessed December 15, 2012).
- Guimarães, L. R., and P. R. S. Vilela. 2005. Comparing software development models using CDM. In *Proceedings of the 6th conference on Information technology education*. Newark, NJ, USA. ACM.
- Hand, D., H. Mannila, and P. Smyth. 2001. *Principles of Data Mining*. Cambridge, Massachusetts: The MIT Press.
- Hamed, H., and E. Al-Shaer. 2006. Dynamic rule-ordering optimization for high-speed firewall filtering. In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*. Taipei, Taiwan. ACM.

- Hashim, F., K. S. Munasinghe, and A. Jamalipour. 2010. Biologically Inspired Anomaly Detection and Security Control Frameworks for Complex Heterogeneous Networks. *Network and Service Management, IEEE Transactions on* 7 (4): 268-281.
- Hauser, C., F. Tronel, C. Fidge, and L. Mé. 2013. *IEEE ICC 2013-IEEE International Conference on Communications, Intrusion detection in distributed systems, an approach based on taint marking*.
- Helmer, G., J. S. K. Wong, V. Honavar, L. Miller, and Y. Wang. 2003. Lightweight agents for intrusion detection. *Systems and Software Journal* 67 (2): 109-122.
- Holden, G. 2004. *Guide to Firewalls and Network Security Intrusion Detection and VPNs*. Canada: Course Technology, Thomson Learning Inc.
- Iheagwara, C., and A. Blyth. 2002. Evaluation of the performance of ID systems in a switched and distributed environment: The RealSecure case study. *Computer Networks Journal* 39 (2): 93-112.
- Innella, P. 2001. *The Evolution of Intrusion Detection Systems*. <http://www.securityfocus.com/infocus/1514> (accessed 31/01/2009).
- Jenkins, A. M. 1985. Research Methodologies and MIS Research. In *Research Methods in Information Systems*, ed. E. Mumford, R. Hirschheim, G. Fitzgerald and A. T. Wood-Harper, 103-117. Amsterdam, Holland: Elsevier Science Publishers B.V.
- Kennedy, R. L., Y. Lee, B. V. Roy, C. D. Reed, and R. P. Lippman. 1997. *Solving Data Mining Problems Through Pattern Recognition*. Upper Saddle River, N.J. : Prentice Hall.
- Keromytis, A. D., and V. Prevelakis. 2007. Designing Firewalls: A survey. In *Network Security: Current Status and Future Directions*, ed. C. Douligieris and D. N. Serpanos: John Wiley & Sons.
- Kizza, J. 2013. Firewalls. In *Guide to Computer Network Security, Computer Communications and Networks*, 247-269. Springer-Verlag London
- KrishnaKumar, B., P. K. Kumar, and R. Sukanesh. 2010. *Recent Trends in Information, Telecommunication and Computing (ITC), 2010 International Conference on, 12-13 March 2010: Hop Count Based Packet Processing Approach to Counter DDoS Attacks*.
- Kruegel, C., and G. Vigna. 2003. Anomaly detection of web-based attacks. In *Proceedings of the 10th ACM conference on Computer and communications security*. Washington D.C., USA. ACM.
- Kruegel, C., D. Mutz, F. Valeur, and G. Vigna. 2003. On the Detection of Anomalous System Call Arguments. In *Computer Security – ESORICS 2003*, ed. E. Sneekenes and D. Gollmann, 326-343. Springer Berlin / Heidelberg.

- Lamp, J. 2001. The Code Red Epidemic: a Case Study. *Journal of Research and Practice in Information Technology* 33 (3): 263-266.
- Lee, H.-W., T. Kwon, and H.-J. Kim. 2005. NS-2 Based IP Traceback Simulation Against Reflector Based DDoS Attack. In *Artificial Intelligence and Simulation*: 90-99.
- Lee, K., J. Kim, K. H. Kwon, Y. Han, and S. Kim. 2008. DDoS attack detection method using cluster analysis. *Expert Systems with Applications* 34 (3): 1659-1665.
- Lee, S. C., and D. V. Heinbuch. 2001. Training a neural-network based intrusion detector to recognize novel attacks. *IEEE Transactions on Systems, Man and Cybernetics, Part A* 31 (4): 294-299.
- Lee, W., and S. J. Stolfo. 2000. A framework for constructing features and models for intrusion detection systems. *ACM Transactions on Information and System Security (TISSEC) Journal* 3 (4): 227 - 261.
- Leigh, W., R. Purvis, and J. M. Ragusa. 2002. Forecasting the NYSE composite index with technical analysis, pattern recognizer, neural network, and genetic algorithm: a case study in romantic decision support. *Decision Support Systems* 32 (4): 361-377.
- Li, J., Y. Liu, and L. Gu. 2010. *Aware Computing (ISAC), 2010 2nd International Symposium on, 1-4 Nov. 2010: DDoS attack detection based on neural network*.
- Li, M. 2004. An approach to reliably identifying signs of DDOS flood attacks based on LRD traffic pattern recognition. *Computers & Security* 23 (7): 549-558.
- Li, M. 2006. Change trend of averaged Hurst parameter of traffic under DDOS flood attacks. *Computers & Security* 25 (3): 213-220.
- Lin, S.-C., and S.-S. Tseng. 2004. Constructing detection knowledge for DDoS intrusion tolerance. *Expert Systems with Applications* 27 (3): 379-390.
- Lipson, H. F. 2002. *Tracking and Tracing Cyber-Attacks: Technical Challenges and Global Policy Issues*. Technical Report CMU/SEI-2002-SR-009, Carnegie Mellon University, Software Engineering Institute, November 2002.
- Lo, C.-C., C.-C. Huang, and L. Ku. 2010. *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on, 13-16 Sept. 2010: A Cooperative Intrusion Detection System Framework for Cloud Computing Networks*.
- Makridakis, S., S. C. Wheelwright, and R. J. Hyndman. 1998. *Forecasting Methods And Applications, 3Rd Ed*: John Wiley & Sons Inc.
- Mall, R. 2000. *Fundamentals of Software Engineering*. New Delhi: Prentice-Hall of India.

- Malliga, S., and A. Tamilarasi. 2007. *Conference on Computational Intelligence and Multimedia Applications, 2007. International Conference on, A Defensive Mechanism to Defend against DoS / DDoS Attacks by IP Traceback with DPM.*
- Malliga, S., A. Tamilarasi, and M. Janani. 2008. *Computing, Communication and Networking, 2008. ICCCN 2008. International Conference on, Filtering spoofed traffic at source end for defending against DoS / DDoS attacks.*
- McEachen, J. C., and a. J. M. Zachary. 2007. IDS for Networks. In *Network Security: Current Status and Future Directions*, ed. C. Douligieris and D. N. Serpanos: John Wiley & Sons.
- Michael, C. C., and A. Ghosh. 2000. Two state-based approaches to program-based anomaly detection. In *Proceedings of the 16th Annual Computer Security Applications Conference*. IEEE Computer Society.
- Mirkovic, J., and P. Reiher. 2004. A taxonomy of DDoS attack and DDoS defense mechanisms. *SIGCOMM Comput. Commun. Rev.* 34 (2): 39-53.
- Mitrokotsa, A., and C. Douligieris. 2005. *Signal Processing and Information Technology, 2005. Proceedings of the Fifth IEEE International Symposium on, Detecting denial of service attacks using emergent self-organizing maps.*
- Mitrokotsa, A., and C. Douligieris. 2007. Denial-of-Service Attacks. In *Network Security: Current Status and Future Directions*, ed. C. Douligieris and D. N. Serpanos: John Wiley & Sons.
- Mukkamala, S., G. Janoski, and A. Sung. 2002. *Neural Networks, 2002. IJCNN '02. Proceedings of the 2002 International Joint Conference on, 2002: Intrusion detection using neural networks and support vector machines.*
- Nazario, J. 2008. DDoS attack evolution. *Network Security* 2008 (7): 7-10.
- Newman, R. C. 2006. Cybercrime, identity theft, and fraud: practicing safe internet - network security threats and vulnerabilities. In *Proceedings of the 3rd annual conference on Information security curriculum development*. Kennesaw, Georgia. ACM.
- Nisbet, R., J. Elder, and G. Miner. 2009. *Handbook of Statistical Analysis & Data Mining Applications*. Burlington, MA: Academic Press.
- Noakes-Fry, K. 2004. Enterpris Firewalls: Technolgy Overview. Gartner Research ID No. DPRO-90318.
- Noureldien, N. A., and I. M. Osman. 2000. *Proceedings of the TENCON 2000 Conference, On firewalls evaluation criteria*. Kuala Lumpur Malaysia.
- Ollmann, G. 2003. Intrusion Prevention Systems (IPS) destined to replace legacy routers. *Network Security Journal* 2003 (11): 18-19.

- Onut, I.-V., and A. A. Ghorbani. 2007. SVision: A novel visual network-anomaly identification technique. *Computers & Security* 26 (3): 201-212.
- Oppliger, R. 1997. Internet security: firewalls and beyond. *Communications of the ACM Journal* 40 (5): 92 - 102.
- Ordenez-Cardenas, E., and R. d. J. Romero-Troncoso. 2008. MLP Neural Network And On-Line Backpropagation Learning Implementation In A Low-Cost FPGA. In *Proceedings of the 18th ACM Great Lakes symposium on VLSI*. Orlando, Florida, USA. ACM.
- Oshima, S., A. Hirakawa, T. Nakashima, and T. Sueyoshi. 2009. *Intelligent Information Hiding and Multimedia Signal Processing, 2009. IIH-MSP '09. Fifth International Conference on, 12-14 Sept. 2009: DoS / DDoS Detection Scheme Using Statistical Method Based on the Destination Port Number*.
- Otey, M., S. Parthasarathy, A. Ghoting, G. Li, S. Narravula, and D. Panda. 2003. Towards NIC-based intrusion detection. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. Washington, D.C. ACM.
- Paré, G., and J. J. Elam. 1997. *Proceedings of the IFIP TC8 WG 8.2 international conference on Information systems and qualitative research, Using case study research to build theories of IT implementation*. Philadelphia, Pennsylvania, United States: Chapman & Hall, Ltd. (accessed March 3, 2008).
- Patrikakis, C., M. Masikos, and O. Zouraraki. 2004. Distributed Denial of Service Attacks. *Internet Protocol Journal, Cisco Systems vol. 7, no. 4*. (accessed December 15, 2012).
- Pelechrinis, K., M. Iliofotou, and S. V. Krishnamurthy. 2011. Denial of service attacks in wireless networks: The case of jammers. *Communications Surveys & Tutorials, IEEE* 13 (2): 245-257.
- Pescatore, J., and R. Stiennon. 2003. Defining intrusion prevention. Research Note. Gartner Research Note No. TU-20-0149.
- Pethia, R. D. 2003. *Viruses and Worms: What Can We Do About Them?* [http://www.cert.org/congressional\\_testimony/Pethia-Testimony-9-10-2003/#2](http://www.cert.org/congressional_testimony/Pethia-Testimony-9-10-2003/#2) . (accessed January 31, 2009).
- Prolexic. 2012. *Prolexic Attack Report Q1 2012*. Prolexic Technologies, Inc. [http://www.institutionalinvestorchina.com/arfy/uploads/soft/120419/32320\\_1814556861.pdf](http://www.institutionalinvestorchina.com/arfy/uploads/soft/120419/32320_1814556861.pdf) (accessed December 15, 2012).
- Robertson, W., G. Vigna, C. Kruegel, and R. A. Kemmerer. 2006. In: *NDSS 2006: Proc. 13th ISOC Symposium on Network and Distributed Systems Security (2006), Using generalization and characterization techniques in the anomaly-based detection of web attacks*.

- Rozenshine-Kemelmakher, E., R. Puzis, A. Felner, and Y. Elovici. 2010. *Communications (ICC), 2010 IEEE International Conference on, 23-27 May 2010: Cost Benefit Deployment of DNIPS*.
- Sarker, S., and A. S. Lee. 1998. *Proceedings of the international conference on Information systems, Using a positivist case research methodology to test a theory about IT-enabled business process redesign*. Helsinki, Finland: Association for Information Systems.
- Sequeira, K., and M. Zaki. 2002. ADMIT: anomaly-based data mining for intrusions. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. Edmonton, Alberta, Canada. ACM.
- Schadwinkel, S., and W. Dilger. 2006. A dynamic approach to artificial immune systems utilizing neural networks. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. Seattle, Washington, USA. ACM.
- Shadowserver Foundation. 2012. Statistics - *DDoS Historical*.  
<http://www.shadowserver.org/wiki/pmwiki.php/Stats/DDoSHistorical>  
(accessed December 15, 2012).
- Shanks, G. 2002. Guidelines for Conducting Positivist Case Study Research in Information Systems. *Australasian Journal of Information Systems* 10 (1): 78-85.
- Shanks, G., and A. Parr. 2003. *Proceedings of the 11th European Conference on Information Systems 2003, Positivist, Single Case Study Research in Information Systems: a Critical Analysis*.
- Sheth, C., and R. Thakker 2011. *Devices and Communications (ICDeCom), 2011 International Conference on, 24-25 Feb. 2011: Performance Evaluation and Comparative Analysis of Network Firewalls*.
- SIA 2011, Survivability and Information Assurance Curriculum. *Principle 01 – Survivability, maintained by Duane Dunston and dated 08/21/2011*.  
<http://www.learnsia.org/curriculum.php> (accessed October 12, 2013).
- Singh, S., and S. Silakari. 2009. A survey of cyber attack detection systems. *International Journal of Computer Science and Network Security* 9 (5): 1-10.
- Siris, V. A., and F. Papagalou. 2006. Application of anomaly detection algorithms for detecting SYN flooding attacks. *Computer Communications* 29 (9): 1433-1442.
- Smith, R. N., and S. Bhattacharya. 1999. *Proceedings of the Performance, Computing and Communications Conference, 1999. IPCCC '99. IEEE International, Operating firewalls outside the LAN perimeter*. Scottsdale, AZ USA.

- Snyder, J. 2006. *Six Integral Steps to Selecting the Right IPS for Your Network*.  
[http://www.baker.ie/baker\\_products/downloads/Juniper\\_IPS\\_network\\_protection.pdf](http://www.baker.ie/baker_products/downloads/Juniper_IPS_network_protection.pdf) (accessed February 13, 2011).
- Specht, S. M., and R. B. Lee. 2004. Distributed Denial of Service: Taxonomies of Attacks, Tools, and Countermeasures. *Proceedings of the 17th International Conference on Parallel and Distributed Computing Systems, 2004 International Workshop on Security in Parallel and Distributed Systems, September 2004*: 543-550.
- Stallings, W. 2006. *Cryptography and network security: principles and practices*. 4ed. New Jersey: Upper Saddle River, N.J.: Prentice Hall.
- Stiennon, R. 2003. Four paths to true network security. Commentary. Gartner Research Commentary No. COM-20-0571.
- Stiennon, R., and M. Easley. 2002. Intrusion prevention will replace intrusion detection. Research Note. Gartner Research Note No. T-17-0115.
- Sudhir S., J., J. Sunil M., and R. R. Agrawal. 2013. Fast and Scalable Method to Resolve Anomalies in Firewall Policies. *International Journal of Advanced And Innovative Research (IJAIR)* 2 (3).
- Suh, S. C. 2012. *Practical Application of Data Mining*. Sudbury, MA: Jones & Bartlett Learning, LLC.
- Swingler, K. 1996. *Applying Neural Networks, A Practical Guide* San Diego, CA: Academic Press Inc.
- Takei, Y., K. Ohta, N. Kato, and Y. Nemoto. 2004. Detecting and tracing illegal access by using traffic pattern matching technique. *Electronics and Communications in Japan (Part I: Communications)* 87 (1): 61-71.
- Talihärm, A.-M. 2010. Cyberterrorism: in Theory or in Practice? *Defence Against Terrorism Review* 3 (2): 59-74.
- Tzur-David, S., H. Avissar, D. Dolev, and T. Anker. 2010. *High Performance Switching and Routing (HPSR), 2010 International Conference on, 13-16 June 2010: SPADE: Statistical Packet Acceptance Defense Engine*.
- Venter, H. S., and J. H. P. Eloff. 2003. A taxonomy for information security technologies. *Computers & Security* 22 (4): 299-307.
- Verwoerd, T., and R. Hunt. 2002. Intrusion detection techniques and approaches. *Computer Communications Journal* 25 (15): 1356-1365.
- Vigna, G., F. Valeur, and R. A. Kemmerer. 2003. *Proceedings of the 9th European software engineering conference held jointly with 10th ACM SIGSOFT international symposium on Foundations of software engineering, Designing and implementing a family of intrusion detection systems*. Helsinki, Finland: ACM Press.

- Viinikka, J., H. Debar, L. Mé, A. Lehtikainen, and M. Tarvainen. 2009. Processing intrusion detection alert aggregates with time series modeling. *Information Fusion* 10 (4): 312-324.
- Vijayarathy, R., S. V. Raghavan, and B. Ravindran. 2011. *Communication Systems and Networks (COMSNETS), 2011 Third International Conference on, 4-8 Jan. 2011: A system approach to network modeling for DDoS detection using a Na'ive Bayesian classifier.*
- Wang, B.-T., and H. Schulzrinne. 2004. *Proceedings of the IEEE Electrical and Computer Engineering Conference, An IP Traceback Mechanism For Reflective Dos Attacks, May 2004:* 901-904.
- Weiss, A. 2012. How to Prevent DoS Attacks. eSecurityPlanet.com <http://www.esecurityplanet.com/network-security/how-to-prevent-dos-attacks.html> (accessed December 16, 2012)
- Wikipedia. 2012a. Internet Traffic. [http://en.wikipedia.org/wiki/Internet\\_traffic](http://en.wikipedia.org/wiki/Internet_traffic) (accessed December 15, 2012)
- Wikipedia. 2012b. Mafiaboy. <http://en.wikipedia.org/wiki/MafiaBoy> (accessed December 15, 2012)
- Wool, A. 2004. A quantitative study of firewall configuration errors. *IEEE Computer Journal* 37 (6): 62-67.
- Wu, S. X., and W. Banzhaf. 2010. The use of computational intelligence in intrusion detection systems: A review. *Applied Soft Computing* 10 (1): 1-35.
- Xiaobo, H., W. Xiaoyan, and Z. Shisong. 2010. *Intelligent System Design and Engineering Application (ISDEA), 2010 International Conference on, 13-14 Oct. 2010: Study on Intelligent Firewall System Combining Intrusion Detection and Egress Access Control.* (accessed December 16, 2012)
- Xu, J., and M. Singhal. 1999. Design of a high-performance ATM firewall. *ACM Transactions on Information and System Security (TISSEC) Journal* 2 (3): 269 - 294.
- Xu, X.-M., and J. Zhan. 2008. Dynamic Evolution Systems and Applications in Intrusion Detection Systems. In *Proceedings of the 2008 International Conference on Information Security and Assurance (isa 2008) - Volume 00.* IEEE Computer Society.
- Ye, N., and Q. Chen. 2001. An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems. *Quality and Reliability Engineering International* 17 (2): 105-112.
- Yin, R. K. 1994. Case Study Research Design and Methods. In *Applied Social Research Methods Series.* Thousands Oaks. Sage Publications.

- Young, G., and J. Pescatore. 2008. Magic Quadrant for Network Intrusion Prevention System Appliances, 1H08 Research Note. Gartner RAS Core Research Note G00154849, 14 February 2008.
- Young, G. 2013. Magic Quadrant for Enterprise Network Firewalls Research Note. Gartner Research Note ID: G00229302, 7 February 2013.
- Yu, J., H. Lee, M.-S. Kim, and D. Park. 2008. Traffic flooding attack detection with SNMP MIB using SVM. *Computer Communications* 31 (17): 4212-4219.
- Zanero, S. 2008. ULISSE, a network intrusion detection system. In *Proceedings of the 4th annual workshop on Cyber security and information intelligence research: developing strategies to meet the cyber security and information intelligence challenges ahead*. Oak Ridge, Tennessee. ACM.
- Zhang, C., J. Jiang, and M. Kamel. 2005. Intrusion detection using hierarchical neural networks. *Pattern Recognition Letters* 26 (6): 779-791.
- Zhang, Y., W. Lee, and Y.-A. Huang. 2003. Intrusion detection techniques for mobile wireless networks. *Wireless Networks Journal* 9 (5): 545 - 556.
- Znati, T., J. Amadei, D. R. Pazehoski, and S. Sweeny. 2006. Design and Analysis of an Adaptive, Global Strategy for Detecting and Mitigating Distributed DoS Attacks in GRID Environments. In *Proceedings of the 39th annual Symposium on Simulation*. IEEE Computer Society.

Every reasonable effort has been made to acknowledge the owners of copyright material. I would be pleased to hear from any copyright owner who has been omitted or incorrectly acknowledged.

## APPENDIX – A: DoSTDM Programming Code

This appendix covers the programming code used to implement DoSTDM and its data collection tools as described in Chapter 4 and Chapter 6.

### A.1 DATA PREPREPARATION PROGRAMMING CODE

Chapter 4, Section 4.5 described the algorithm and pseudo code to complete the data re-construction from raw data to a new processed format based on direction and activity of each protocol. This can be achieved via many programming languages using Unix Shell, Perl and Java.

#### A.1.1 Data Collection Shell Scripts

The following script had been used to implement the pseudo code in Chapter 4 Section 4.5 using a five step approach to transform the data. The Unix/Perl script in Listing A – 1 reads the raw data logs and feeds the data to two Java classes (Base Data Tokenizer and Hourly Data Classifier) to reconstruct and classify the data as shown in Chapter 4 Table 4 – 9. The detailed Java programming code for these is provided in the next section.

```
#!/bin/sh

#####
# Function: Script to re-format Firewall Logs #
# Input: Raw Firewall Logs #
# Output: Single month data in 24 hour format #
# Software: Shell and Perl calling Java classes #
#####

# List firewall log files and store file names in filesID.dat text file
# the log files are in the format of dd-mmm-yyyy.log.gz
ls *.log > filesID.dat

# Set the month file name in the format mmm_year.csv
month_file=`tail -n 1 filesID.dat | awk -F'.' '{ print $1 }' | awk -F'-' '{
print $2_"$3".csv" }'`

# Echo statistics file header to the monthly file in CSV format
echo DAY, TIME, INBOUND, TCP_IN, IP_IN, ICMP_IN, UDP_IN, OTHER_IN, OUTBOUND,
TCP_OUT, IP_OUT, ICMP_OUT, UDP_OUT, OTHER_OUT, ACCEPTED, TCP_ACPT, IP_ACPT,
ICMP_ACPT, UDP_ACPT, OTHER_ACPT, REJECTED, TCP_RJCT, IP_RJCT, ICMP_RJCT,
UDP_RJCT, OTHER_RJCT > $month_file

# Calling Java Class "BaseDataTokenizer" to parse date and month start and
# end to identify days before and after the current month data and store the
# results in file called "tokens.dat"
java BaseDataTokenizer filesID.dat > tokens.dat
```

```

# Remove temporary files.dat file
rm -f filesID.dat

# Create a loop to read tokens.dat file
for x in `cat tokens.dat`
do

# Breakdown the tokens within tokens.dat file as follow:
# v1: File Name (e.g. 01-Sep-2010.log)
# v2: Day Before (e.g. 31Aug2010)
# v3: Day After (e.g. 2Sep2010)
# v4: Log Date (e.g. 01-Sep-2010)

v1=`echo $x|awk -F',' '{print $1}'`
v2=`echo $x|awk -F',' '{print $2}'`
v3=`echo $x|awk -F',' '{print $3}'`
v4=`echo $v1|awk -F'.' '{print $1}'`

# Remove dos carriage return and replace it with line feed using perl (this
# step is required for some firewall logs whereby the logs do not use line
# line feed to start a new line in the log
perl -pi -e 's/\r\n/\n/;' $v1

# Collect data and pipe it to a temporary file
cat $v1 | awk -F" " '{ print $1;"$2";"$4";"$7;"$6 }' | awk -F"-" '{ print
$1$2$3 }' > $v1.tmp

# Replace In with inbound in the file using perl
perl -pi -e 's/In/inbound/g' $v1.tmp

# Replace Out with outbound in the file using perl
perl -pi -e 's/Out/outbound/g' $v1.tmp

# Replace Pass with accept in the file using perl
perl -pi -e 's/Pass/accept/g' $v1.tmp

# Replace Deny with reject in the file using perl
perl -pi -e 's/Deny/reject/g' $v1.tmp

# Remove original log file
rm -f $v1

# Rename temporary file to the original log file name
mv $v1.tmp $v1

# Capture required data for a specific day only and omit data for day before
# and day after then write it to a staging file
cat $v1 | grep -v "$v2" | grep -v "$v3" > $v1.stg1

# Calling Java Class "HourlyDataClassifier" to separate hourly values and
# pip results in 24 hours format per day to a monthly database file

grep ";00:" $v1.stg1 > $v4.hr01
java -Xmx1024m -Xms512m HourlyDataClassifier $v4.hr01 >> $month_file
grep ";01:" $v1.stg1 > $v4.hr02
java -Xmx1024m -Xms512m HourlyDataClassifier $v4.hr02 >> $month_file
grep ";02:" $v1.stg1 > $v4.hr03
java -Xmx1024m -Xms512m HourlyDataClassifier $v4.hr03 >> $month_file
grep ";03:" $v1.stg1 > $v4.hr04
java -Xmx1024m -Xms512m HourlyDataClassifier $v4.hr04 >> $month_file
grep ";04:" $v1.stg1 > $v4.hr05
java -Xmx1024m -Xms512m HourlyDataClassifier $v4.hr05 >> $month_file
grep ";05:" $v1.stg1 > $v4.hr06
java -Xmx1024m -Xms512m HourlyDataClassifier $v4.hr06 >> $month_file
grep ";06:" $v1.stg1 > $v4.hr07
java -Xmx1024m -Xms512m HourlyDataClassifier $v4.hr07 >> $month_file

```

```

grep ";07:" $v1.stg1 > $v4.hr08
java -Xmx1024m -Xms512m HourlyDataClassifier $v4.hr08 >> $month_file
grep ";08:" $v1.stg1 > $v4.hr09
java -Xmx1024m -Xms512m HourlyDataClassifier $v4.hr09 >> $month_file
grep ";09:" $v1.stg1 > $v4.hr10
java -Xmx1024m -Xms512m HourlyDataClassifier $v4.hr10 >> $month_file
grep ";10:" $v1.stg1 > $v4.hr11
java -Xmx1024m -Xms512m HourlyDataClassifier $v4.hr11 >> $month_file
grep ";11:" $v1.stg1 > $v4.hr12
java -Xmx1024m -Xms512m HourlyDataClassifier $v4.hr12 >> $month_file
grep ";12:" $v1.stg1 > $v4.hr13
java -Xmx1024m -Xms512m HourlyDataClassifier $v4.hr13 >> $month_file
grep ";13:" $v1.stg1 > $v4.hr14
java -Xmx1024m -Xms512m HourlyDataClassifier $v4.hr14 >> $month_file
grep ";14:" $v1.stg1 > $v4.hr15
java -Xmx1024m -Xms512m HourlyDataClassifier $v4.hr15 >> $month_file
grep ";15:" $v1.stg1 > $v4.hr16
java -Xmx1024m -Xms512m HourlyDataClassifier $v4.hr16 >> $month_file
grep ";16:" $v1.stg1 > $v4.hr17
java -Xmx1024m -Xms512m HourlyDataClassifier $v4.hr17 >> $month_file
grep ";17:" $v1.stg1 > $v4.hr18
java -Xmx1024m -Xms512m HourlyDataClassifier $v4.hr18 >> $month_file
grep ";18:" $v1.stg1 > $v4.hr19
java -Xmx1024m -Xms512m HourlyDataClassifier $v4.hr19 >> $month_file
grep ";19:" $v1.stg1 > $v4.hr20
java -Xmx1024m -Xms512m HourlyDataClassifier $v4.hr20 >> $month_file
grep ";20:" $v1.stg1 > $v4.hr21
java -Xmx1024m -Xms512m HourlyDataClassifier $v4.hr21 >> $month_file
grep ";21:" $v1.stg1 > $v4.hr22
java -Xmx1024m -Xms512m HourlyDataClassifier $v4.hr22 >> $month_file
grep ";22:" $v1.stg1 > $v4.hr23
java -Xmx1024m -Xms512m HourlyDataClassifier $v4.hr23 >> $month_file
grep ";23:" $v1.stg1 > $v4.hr24
java -Xmx1024m -Xms512m HourlyDataClassifier $v4.hr24 >> $month_file

# Rename stage file to the original log file name
mv $v1.stg1 $v1

# close the log files reader loop
done

```

Listing A – 1: UNIX Shell & Perl Script to Collect Raw Data from Firewall Logs

## A.1.2 Data Processing Code in Java

The following sections show the Java classes code used with the shell code presented in Listing A – 1. These classes contain multiple functions and internal classes to produce the final data format as shown in Chapter 4 Table 4 – 9.

### A.1.2.1 Java Class Base Data Tokenizer ( `BaseDataTokenizer.java` ):

The Base Data Tokenizer class is the main class to call the Base Data Reader internal class. It reads the input file name as an argument given with the Java application name when running the application from the command line as follow:

*Class command line usage: java BaseDataTokenizer <file name >*

The log file name will be forwarded to Base Data Reader class to process the data structure within that file, if no file name is given, and then the program will generate an error message with the correct command line usage.

The following Listing A – 2 shows this class complete Java code.

```
import java.io.*;
import java.util.*;

public class BaseDataTokenizer
{
    public static void main (String args[])
    {
        int i;
        String fileName="";

        // check input argument
        if (args.length == 0)
        {
            usage();
        }
        else
        {
            // constructing the input file name from arguments input
            for( i = 0 ; i < args.length ; i++)
            {
                fileName+=args[i];
            }

            // create a file reader from BaseDataReader class
            BaseDataReader bdr = new BaseDataReader();

            // send the file name to BaseDataReader to start reading file
            bdr.readingFileLines(fileName);

            System.exit(0);
        }

        // Print a usage message for this application
        public static void usage ()
        {
            System.err.println("\n"+"t"+"t"+"Usage: java BaseDataTokenizer <file
            name>"+ "\n");
        }
    } // end of BaseDataTokenizer() class

public class BaseDataReader
{
    void readingFileLines(String inputFileNames)
    {
        try // start try to catch file reading errors
        {

            // create a file object using the input file name
            File inputFile = new File(inputFileNames);
```

```

// create a buffered reader and attach it the file reader
BufferedReader fileInputStream =
new BufferedReader(new FileReader(inputFile));

// read file lines
String inputLine = fileInputStream.readLine();
String year = "";
String month = "";
String date = "";
String dayBefore = "";
String dayAfter = "";
String monthsList[] = { "Jan", "Feb", "Mar", "Apr", "May", "Jun",
                        "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" };

int intYear = 0;
int intMonth = 0;
int intDate = 0;

while ( inputLine != null )
{
Year = inputLine.substring(7,11);
Month = inputLine.substring(3,6);
Date = inputLine.substring(0,2);
intYear = Integer.parseInt(year);

if (month.equalsIgnoreCase("Jan")) { intMonth=0; }
if (month.equalsIgnoreCase("Feb")) { intMonth=1; }
if (month.equalsIgnoreCase("Mar")) { intMonth=2; }
if (month.equalsIgnoreCase("Apr")) { intMonth=3; }
if (month.equalsIgnoreCase("May")) { intMonth=4; }
if (month.equalsIgnoreCase("Jun")) { intMonth=5; }
if (month.equalsIgnoreCase("Jul")) { intMonth=6; }
if (month.equalsIgnoreCase("Aug")) { intMonth=7; }
if (month.equalsIgnoreCase("Sep")) { intMonth=8; }
if (month.equalsIgnoreCase("Oct")) { intMonth=9; }
if (month.equalsIgnoreCase("Nov")) { intMonth=10; }
if (month.equalsIgnoreCase("Dec")) { intMonth=11; }

intDate = Integer.parseInt(date);
month = monthsList[intMonth];
dayAfter = String.valueOf(intDate+1) + month+year;
dayBefore = String.valueOf(intDate-1) + month+year;

// special cases dayAfter values
if (intDate==31 && intMonth==11) { dayAfter = "1" +
monthsList[intMonth-11] + String.valueOf(intYear+1); }

if (intDate==31 && intMonth!=11) { dayAfter = "1" +
monthsList[intMonth+1]+year; }

if (intDate==30 && (intMonth==3 || intMonth==5 || intMonth==8 ||
intMonth==10)) { dayAfter = "1" + monthsList[intMonth+1] + year; }

if (intDate==28 && intMonth==1) { dayAfter = "1" +
monthsList[intMonth+1]+year; }

// special cases dayBefore values
if (intDate==1 && intMonth==2) { dayBefore = "28" +
monthsList[intMonth-1]+year; }

if (intDate==1 && intMonth==0) { dayBefore = "31" +
monthsList[intMonth+11] + String.valueOf(intYear-1); }

if (intDate==1 && (intMonth==1 || intMonth==3 || intMonth==5 ||
intMonth==7 || intMonth==8 || intMonth==10)) { dayBefore = "31" +
monthsList[intMonth-1] + year; }

```

```

        if (intDate==1 && (intMonth==4 || intMonth==6 || intMonth==9 ||
        intMonth==11)) { dayBefore = "30" + monthsList[intMonth-1] + year; }

        System.out.println(inputLine + "," + dayBefore + "," + dayAfter);
        inputLine = fileInputStream.readLine();

        } //end of while loop

        fileInputStream.close(); // close the input stream

        } // close try statment

        catch(IOException e) // capture the file I/O exception
        {
            // print error message if file does not exist or
            // if file is unreadable.
            System.out.println("\t"+" !!!Error Reading File or File Does Not
            Exist!!!"); }

        } // end of method readingFileLines()
} // end of class BaseDataReader()

```

Listing A – 2: Java Class BaseDataTokenizer.java

#### A.1.2.2 Java Class Base Hourly Data Classifier ( HourlyDataClassiifer.java ):

This class reads the input file name as an argument given with the java application name when running the application from the command line as follows:

*Class command line: java HourlyDataClassifier <file name >*

The file name will be forwarded to internal class File Data Reader to process the data structure within that file using the Get Token internal class, if no file name is given, then the program will generate an error message with the correct command line usage. The following Listing A – 3 provides the complete Java code of this class.

```

import java.io.*;
import java.util.*

public class HourlyDataClassifier
{
    public static void main (String args[])
    {
        int i;
        String fileName="";
        // check input argument
        if (args.length == 0)

        {
            usage();
        }
    }
}

```

```

else
{
    // constructing the input file name from arguments input
    for( i = 0 ; i < args.length ; i++ )
    {
        filename+ = args[i];
    }
}
// create a file reader from FileDataReader class
FileDataReader fdr= new FileDataReader();

// send the file name to FileDataReader to start reading file
fdr.readingFileLines(fileName);
System.exit(0);
}

// Print a usage message for this application
public static void usage ()
{
    System.err.println("\n" + "\t" + "\t" +
        "Usage: java HourlyDataClassifier <file name>" + "\n");
}
} // end of HourlyDataClassifier() class

public class FileDataReader
{
    void readingFileLines(String inputFileName)
    {
        // DATE & TIME RECORDS
        String [] datelist = new String [2000000];
        String [] timelist = new String [2000000];

        // TRAFIC IN-OUT STATS
        int [] inbound = new int [2000000];
        int [] outbound = new int [2000000];

        // TRAFIC IN-OUT STATS
        int [] accepted = new int [2000000];
        int [] rejected = new int [2000000];

        // TCP PACKETS STATS
        int [] tcp_accpt = new int [2000000];
        int [] tcp_rejct = new int [2000000];
        int [] tcp_in = new int [2000000];
        int [] tcp_out = new int [2000000];

        // IP PACKETS STATS
        int [] ip_accpt = new int [2000000];
        int [] ip_rejct = new int [2000000];
        int [] ip_in = new int [2000000];
        int [] ip_out = new int [2000000];

        // ICMP PACKETS STATS
        int [] icmp_accpt = new int [2000000];
        int [] icmp_rejct = new int [2000000];
        int [] icmp_in = new int [2000000];
        int [] icmp_out = new int [2000000];

        // UDP PACKETS STATS
        int [] udp_accpt = new int [2000000];
        int [] udp_rejct = new int [2000000];
        int [] udp_in = new int [2000000];
        int [] udp_out = new int [2000000];
    }
}

```

```

// Other PACKETS STATS
int [] other_accpt = new int [2000000];
int [] other_rejct = new int [2000000];
int [] other_in = new int [2000000];
int [] other_out = new int [2000000];

// start try to catch file reading errors
try
    {

        // create a file object using the input file name
        File inputFile = new File(inputFileName);

        // create a buffered reader and attach it the file reader
        BufferedReader fileInputStream =
            new BufferedReader(new FileReader(inputFile));

        // read file lines
        String inputLine = fileInputStream.readLine();

        // reading initial data and tokenize it
        GetToken gt= new GetToken();

        String date = gt.dateToken(inputLine);
        String time = gt.timeToken(inputLine);
        String action = gt.actionToken(inputLine);
        String if_dir = gt.if_dirToken(inputLine);
        String proto = gt.protoToken(inputLine);

        int j=0;
        int hour=0;

        while (inputLine != null)
        {
            datelist[j] = date;

            if (time.substring(1,2).equalsIgnoreCase(":"))
            {
                hour=(Integer.parseInt(time.substring(0,1))) + 1;
            }
            else
            hour=(Integer.parseInt(time.substring(0,2))) + 1;
            timelist[j] = Integer.toString(hour) + ":00";

            if (if_dir.equalsIgnoreCase("inbound"))
            {
                inbound[j] = 1;
            }
            else
            {
                inbound[j]=0;
            }

            if (if_dir.equalsIgnoreCase("outbound"))
            {
                outbound[j] = 1;
            }
            else
            {
                outbound[j]=0;
            }
            if (action.equalsIgnoreCase("accept"))
            {
                accepted[j] = 1;
            }
        }
    }

```

```

else
    {
        accepted[j] = 0;
    }

if (action.equalsIgnoreCase("reject") ||
action.equalsIgnoreCase("drop"))
{
    rejected[j] = 1;
}
else
    {
        rejected[j] = 0;
    }

// initialize data entry for accepted packets breakdown
tcp_accpt[j] = 0;
ip_accpt[j] = 0;
icmp_accpt[j] = 0;
udp_accpt[j] = 0;
other_accpt[j] = 0;

// initialize data entry for rejected packets breakdown
tcp_rejct[j] = 0;
ip_rejct[j] = 0;
icmp_rejct[j] = 0;
udp_rejct[j] = 0;
other_rejct[j] = 0;

// initialize data entry for inbound packets breakdown
tcp_in[j] = 0;
ip_in[j] = 0;
icmp_in[j] = 0;
udp_in[j] = 0;
other_in[j] = 0;

// initialize data entry for outbound packets breakdown
tcp_out[j] = 0;
ip_out[j] = 0;
icmp_out[j] = 0;
udp_out[j] = 0;
other_out[j] = 0;

// tcp action stats
if (proto.equalsIgnoreCase("tcp") &&
action.equalsIgnoreCase("accept"))
{
    tcp_accpt[j] = 1;
}

if (proto.equalsIgnoreCase("tcp") &&
(action.equalsIgnoreCase("reject") ||
action.equalsIgnoreCase("drop")))
{
    tcp_rejct[j] = 1;
}

// tcp traffic stats
if (proto.equalsIgnoreCase("tcp") &&
if_dir.equalsIgnoreCase("inbound"))
{
    tcp_in[j] = 1;
}

```

```

if (proto.equalsIgnoreCase("tcp") &&
if_dir.equalsIgnoreCase("outbound"))
{
tcp_out[j] = 1;
}

// ip action stats
if (proto.equalsIgnoreCase("ip") &&
action.equalsIgnoreCase("accept"))
{
ip_accpt[j] = 1;
}
if (proto.equalsIgnoreCase("ip") &&
(action.equalsIgnoreCase("reject") ||
action.equalsIgnoreCase("drop")))
{
ip_rejct[j] = 1;
}

// ip traffic stats
if (proto.equalsIgnoreCase("ip") &&
if_dir.equalsIgnoreCase("inbound"))
{
ip_in[j] = 1;
}
if (proto.equalsIgnoreCase("ip") &&
if_dir.equalsIgnoreCase("outbound"))
{
ip_out[j] = 1;
}

// icmp action stats
if (proto.equalsIgnoreCase("icmp") &&
action.equalsIgnoreCase("accept"))
{
icmp_accpt[j] = 1;
}
if (proto.equalsIgnoreCase("icmp") &&
(action.equalsIgnoreCase("reject") ||
action.equalsIgnoreCase("drop")))
{
icmp_rejct[j] = 1;
}

// icmp traffic stats
if (proto.equalsIgnoreCase("icmp") &&
if_dir.equalsIgnoreCase("inbound"))
{
icmp_in[j] = 1;
}
if (proto.equalsIgnoreCase("icmp") &&
if_dir.equalsIgnoreCase("outbound"))
{
icmp_out[j] = 1;
}

// udp action stats
if (proto.equalsIgnoreCase("udp") &&
action.equalsIgnoreCase("accept"))
{
udp_accpt[j] = 1;
}
if (proto.equalsIgnoreCase("udp") &&
(action.equalsIgnoreCase("reject") ||
action.equalsIgnoreCase("drop")))
{
udp_rejct[j] = 1; }

```

```

// udp traffic stats
if (proto.equalsIgnoreCase("udp") &&
    if_dir.equalsIgnoreCase("inbound"))
{
    udp_in[j] = 1;
}
if (proto.equalsIgnoreCase("udp") &&
    if_dir.equalsIgnoreCase("outbound"))
{
    udp_out[j] = 1;
}

// other protocols action stats
if (!(proto.equalsIgnoreCase("tcp")) &&
    !(proto.equalsIgnoreCase("ip")) &&
    !(proto.equalsIgnoreCase("icmp")) &&
    !(proto.equalsIgnoreCase("udp")) &&
    action.equalsIgnoreCase("accept"))
{
    other_accpt[j] = 1;
}
if (!(proto.equalsIgnoreCase("tcp")) &&
    !(proto.equalsIgnoreCase("ip")) &&
    !(proto.equalsIgnoreCase("icmp")) &&
    !(proto.equalsIgnoreCase("udp")) &&
    (action.equalsIgnoreCase("reject") ||
    action.equalsIgnoreCase("drop")))
{
    other_rejct[j] = 1;
}

// other protocols action stats
if (!(proto.equalsIgnoreCase("tcp")) &&
    !(proto.equalsIgnoreCase("ip")) &&
    !(proto.equalsIgnoreCase("icmp")) &&
    !(proto.equalsIgnoreCase("udp")) &&
    if_dir.equalsIgnoreCase("inbound"))
{
    other_in[j] = 1;
}
if (!(proto.equalsIgnoreCase("tcp")) &&
    !(proto.equalsIgnoreCase("ip")) &&
    !(proto.equalsIgnoreCase("icmp")) &&
    !(proto.equalsIgnoreCase("udp")) &&
    if_dir.equalsIgnoreCase("outbound"))
{
    other_out[j]=1;
}

// increment the array index
j++;

// read the next input line from the firewall log file
inputLine = fileInputStream.readLine();

if (inputLine != null)
{
    Date = gt.dateToken(inputLine);
    Time = gt.timeToken(inputLine);
    Action = gt.actionToken(inputLine);
    if_dir = gt.if_dirToken(inputLine);
    proto = gt.protoToken(inputLine);
}
} //end of while loop

```

```

// close the input stream
fileInputStream.close();

// close try statment
}
// capture the file I/O exception
catch(IOException e)
{
    // print error message if file does not exist or
    // if file is unreadable.
    System.out.println("\t" + " !!!Error Reading File or File
Does Not Exist!!!");
}

// initializing statistics per protocol per hour for 24 hours
int h = 0;
int sum_inbound = 0;
int sum_outbound = 0;
int sum_accepted = 0;
int sum_rejected = 0;
int sum_tcp_accpt = 0;
int sum_tcp_rejct = 0;
int sum_tcp_in = 0;
int sum_tcp_out = 0;
int sum_ip_accpt = 0;
int sum_ip_rejct = 0;
int sum_ip_in = 0;
int sum_ip_out = 0;
int sum_icmp_accpt = 0;
int sum_icmp_rejct = 0;
int sum_icmp_in = 0;
int sum_icmp_out = 0;
int sum_udp_accpt = 0;
int sum_udp_rejct = 0;
int sum_udp_in = 0;
int sum_udp_out = 0;
int sum_other_accpt = 0;
int sum_other_rejct = 0;
int sum_other_in = 0;
int sum_other_out = 0;

try{
while (timelist[h] != null)
{
    sum_inbound = sum_inbound + inbound[h];
    sum_outbound = sum_outbound + outbound[h];
    sum_accepted = sum_accepted + accepted[h];
    sum_rejected = sum_rejected + rejected[h];
    sum_tcp_accpt = sum_tcp_accpt + tcp_accpt[h];
    sum_tcp_rejct = sum_tcp_rejct + tcp_rejct[h];
    sum_tcp_in = sum_tcp_in + tcp_in[h];
    sum_tcp_out = sum_tcp_out + tcp_out[h];
    sum_ip_accpt = sum_ip_accpt + ip_accpt[h];
    sum_ip_rejct = sum_ip_rejct + ip_rejct[h];
    sum_ip_in = sum_ip_in + ip_in[h];
    sum_ip_out = sum_ip_out + ip_out[h];
    sum_icmp_accpt = sum_icmp_accpt + icmp_accpt[h];
    sum_icmp_rejct = sum_icmp_rejct + icmp_rejct[h];
    sum_icmp_in = sum_icmp_in + icmp_in[h];
    sum_icmp_out = sum_icmp_out + icmp_out[h];
    sum_udp_accpt = sum_udp_accpt + udp_accpt[h];
    sum_udp_rejct = sum_udp_rejct + udp_rejct[h];
    sum_udp_in = sum_udp_in + udp_in[h];
    sum_udp_out = sum_udp_out + udp_out[h]; sum_other_accpt
sum_other_accpt + other_accpt[h];
}
}

```

```

sum_other_rejct = sum_other_rejct + other_rejct[h];
sum_other_in = sum_other_in + other_in[h];
sum_other_out = sum_other_out + other_out[h];

// increment the while loop counter
h++;

// end of while inner loop
}

// end of try
}
catch(NullPointerException nerr)
{
} // end of catch

// write the hourly data to the output file in CSV format
System.out.println(datelist[0] + "," + timelist[0] + ","
+ Integer.toString(sum_inbound) + ","
+ Integer.toString(sum_tcp_in) + ","
+ Integer.toString(sum_ip_in) + ","
+ Integer.toString(sum_icmp_in) + ","
+ Integer.toString(sum_udp_in) + ","
+ Integer.toString(sum_other_in) + ","
+ Integer.toString(sum_outbound) + ","
+ Integer.toString(sum_tcp_out) + ","
+ Integer.toString(sum_ip_out) + ","
+ Integer.toString(sum_icmp_out) + ","
+ Integer.toString(sum_udp_out) + ","
+ Integer.toString(sum_other_out) + ","
+ Integer.toString(sum_accepted) + ","
+ Integer.toString(sum_tcp_accpt) + ","
+ Integer.toString(sum_ip_accpt) + ","
+ Integer.toString(sum_icmp_accpt) + ","
+ Integer.toString(sum_udp_accpt) + ","
+ Integer.toString(sum_other_accpt) + ","
+ Integer.toString(sum_rejected) + ","
+ Integer.toString(sum_tcp_rejct) + ","
+ Integer.toString(sum_ip_rejct) + ","
+ Integer.toString(sum_icmp_rejct) + ","
+ Integer.toString(sum_udp_rejct) + ","
+ Integer.toString(sum_other_rejct));

} // end of method readingFileLines()
} // end of class FileDataReader()

public class GetToken
{
    public static String dateToken(String line)
    {
        StringTokenizer st = new StringTokenizer(line, ";");
        String date = st.nextToken();
        return date;

    } // end of dateToken() method

    public static String timeToken(String line)
    {
        StringTokenizer st = new StringTokenizer(line, ";");
        String date = st.nextToken();
        String time = st.nextToken();
        return time;
    } // end of timeToken() method
}

```

```

public static String actionToken(String line)
{
StringTokenizer st = new StringTokenizer(line, ";");
String date = st.nextToken();
String time = st.nextToken();
String action = st.nextToken();
return action;
} // end of actionToken() method

public static String if_dirToken(String line)
{
StringTokenizer st = new StringTokenizer(line, ";");
String date = st.nextToken();
String time = st.nextToken();
String action = st.nextToken();
String if_dir = st.nextToken();
return if_dir;
} // end of if_dirToken() method

public static String protoToken(String line)
{
StringTokenizer st = new StringTokenizer(line, ";");
String date = st.nextToken();
String time = st.nextToken();
String action = st.nextToken();
String if_dir = st.nextToken();
String proto = st.nextToken();
return proto;
} // end of protoToken() method

} // end of GetToken() class

```

Listing A – 3: Java Class HourlyDataClassifier.java

## A.2 PROGRAMMING CODE LISTING

This section lists all the scripts and Java code used to complete the build the proposed DoSTDM model in order to analyse the data prepared in Chapter 4.

### A.2.1 Simulation Network Packets Generator Scripts

This section lists all the scripts used in the simulation network to generate random traffic patterns from the Linux server (using nemesis packet generator) toward the firewall server and vice versa as per Chapter 4 Figure 4 – 2. The scripts are controlled using a standard Linux scheduling program (cron) and scheduling table (crontab) setup within the simulation infrastructure using a time gap between the startup of every script to simulate network traffic as shown in Listing A – 4:

```
# m h dom mon dow    command
0,7,14,21,28,35,42,49,56 * * * * /scripts/tcp.sh >> /dev/null
1,8,15,22,29,36,43,50,57 * * * * /scripts/ip.sh >> /dev/null
2,9,16,23,30,37,44,51,58 * * * * /scripts/icmp.sh >> /dev/null
3,10,17,24,31,38,45,52,59 * * * * /scripts/udp.sh >> /dev/null
4,11,18,25,32,39,46,53,59 * * * * /scripts/arp.sh >> /dev/null
5,12,19,26,33,40,47,54,59 * * * * /scripts/ethernet.sh >> /dev/null
6,13,20,27,34,41,48,55,59 * * * * /scripts/dns.sh >> /dev/null
7,14,21,28,35,42,49,56,59 * * * * /scripts/rip.sh >> /dev/null
```

Listing A – 4: Scripts Scheduler for Random Traffic Pattern Generator in The Simulation Network

Listing A – 5 to A – 12 shows the scripts called to generate TCP, IP, UDP, ICMP, ARP, RIP, DNS and Ethernet packets within the simulation network. The scheduling services (cron UNIX / Linux scheduler tool) take care of repeating executing every script with one minute of an hour during the 24 hours of the day every day of the week every month of the year. The following scripts listings were used to produce the random traffic patterns.

#### A.2.1.1 ARP Packets Generator Script

```
#!/bin/bash
count=0
ENDNUM=$RANDOM
let "ENDNUM=$ENDNUM/2"
while [ $count -le $ENDNUM ]
do
    let "count=$count + 1"
    /usr/local/bin/nemesis arp -S 192.168.3.1 -D 192.168.3.2
done
```

Listing A – 5: ARP Traffic Random Generator Script

### A.2.1.2 DNS Packets Generator Script

```
#!/bin/bash
count=0
ENDNUM=$RANDOM
let "ENDNUM=$ENDNUM/2"
while [ $count -le $ENDNUM ]
do
    let "count=$count + 1"
    /usr/local/bin/nemesis dns -D 192.168.3.2
done
```

Listing A – 6: DNS Traffic Random Generator Script

### A.2.1.3 Ethernet Packets Generator Script

```
#!/bin/bash
count=0
ENDNUM=$RANDOM
let "ENDNUM=$ENDNUM/2"
while [ $count -le $ENDNUM ]
do
    let "count=$count + 1"
    /usr/local/bin/nemesis ethernet -d eth2 -M 00-0C-76-0A-C1-0E
done
```

Listing A – 7: Ethernet Traffic Random Generator Script

### A.2.1.4 ICMP Packets Generator Script

```
#!/bin/bash
count=0
ENDNUM=$RANDOM
let "ENDNUM=$ENDNUM/2"
while [ $count -le $ENDNUM ]
do
    let "count=$count + 1"
    /usr/local/bin/nemesis icmp -D 192.168.3.2
done
```

Listing A – 8: ICMP Traffic Random Generator Script

### A.2.1.5 IP Packets Generator Script

```
#!/bin/bash
count=0
ENDNUM=$RANDOM
let "ENDNUM=$ENDNUM/2"
while [ $count -le $ENDNUM ]
do
    let "count=$count + 1"
    /usr/local/bin/nemesis ip -D 192.168.3.2
done
```

Listing A – 9: IP Traffic Random Generator Script

### A.2.1.6 RIP Packets Generator Script

```
#!/bin/bash
count=0
ENDNUM=$RANDOM
let "ENDNUM=$ENDNUM/2"
while [ $count -le $ENDNUM ]
do
    let "count=$count + 1"
    /usr/local/bin/nemesis rip -S 192.168.3.1 -h 192.168.3.2
done
```

Listing A – 10: RIP Traffic Random Generator Script

### A.2.1.7 TCP Packets Generator Script

```
#!/bin/bash
count=0
ENDNUM=$RANDOM
let "ENDNUM=$ENDNUM/2"
while [ $count -le $ENDNUM ]
do
    let "count=$count + 1"
    /usr/local/bin/nemesis tcp -D 192.168.3.2
done
```

Listing A – 11: TCP Traffic Random Generator Script

### A.2.1.8 UDP Packets Generator Script

```
#!/bin/bash
count=0
ENDNUM=$RANDOM
let "ENDNUM=$ENDNUM/2"
while [ $count -le $ENDNUM ]
do
    let "count=$count + 1"
    /usr/local/bin/nemesis udp -D 192.168.3.2
done
```

Listing A – 12: UDP Traffic Random Generator Script

## A.2.2 DoSTDM Java Applet & Associated Classes

This section lists all the Java classes used with DoSTDM as the applet call these classes to read the pre-processed data from the CSV database file and produces a 24 hours prediction. The main parts among other internal and external classes are:

- The main applet class (DoSTDM.java) called by the web browser HTML code (DoSTDM.html)
- The forecasting engine classes (HWF.java, REG.java and MLP.java).
- The formatting and presentation classes

These coding components are linked together as shown in Figure A – 1.

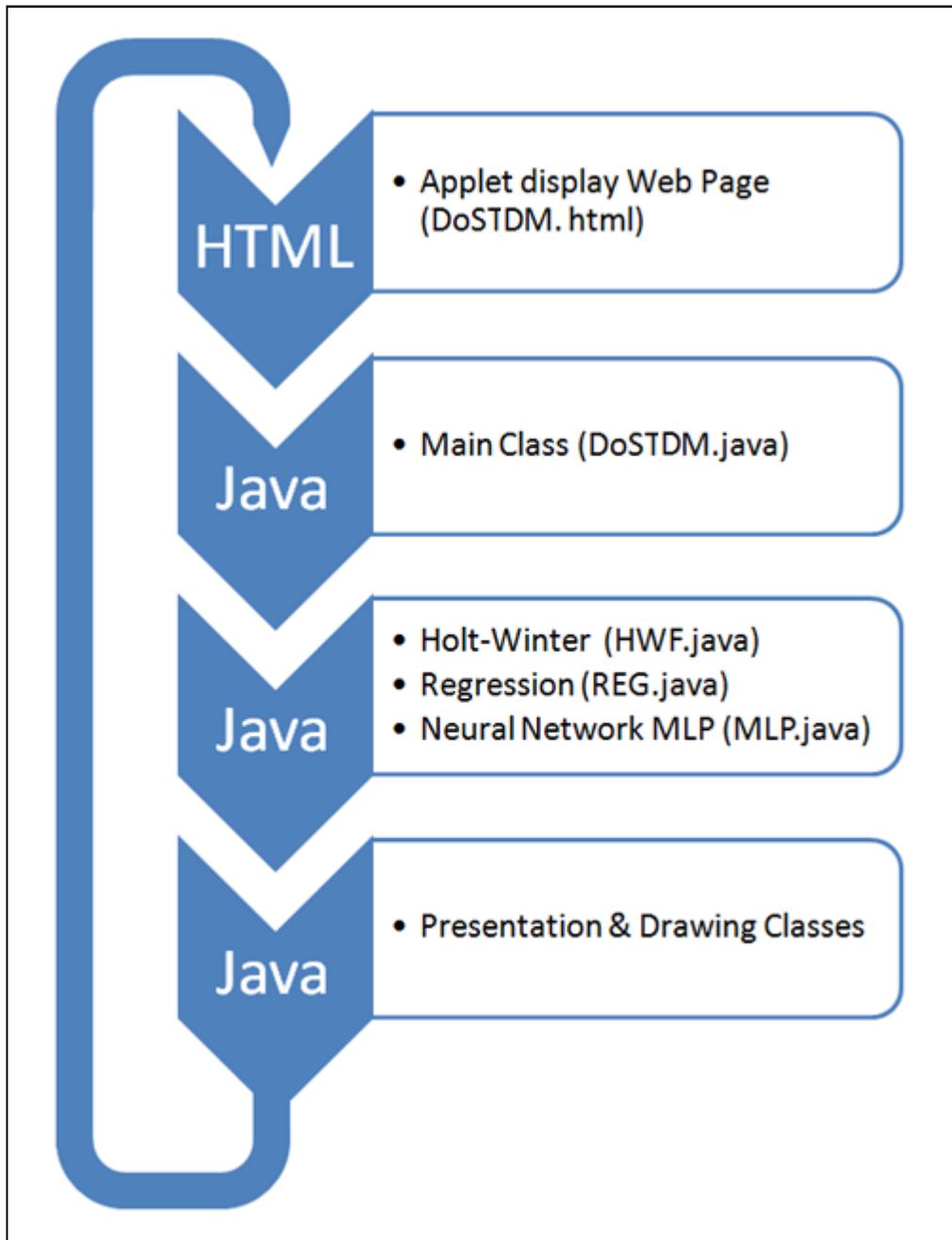


Figure A – 1: DoSTDM Java Applet Coding Design

The following sections provide the code listing of these Java classes used to test DoSTDM design, the compiled classes can be found in Appendix – C.

### A.2.2.1 DoSTDM Applet HTML Code

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<meta name="GENERATOR" content="Microsoft FrontPage 4.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<title>DoSTDM Network Analyser v1.0</title>
</head>

<body bgcolor="#979bca">
<p align="center"><b><font size="6"> DoSTDM <i>Network
Analyser</i></font></b></p>
<table border="0" width="100%">
  <tr>
    <td width="100%">
      <p align="center">
<applet align="center" width="900" height="500" code="DoSTDM.class">
</applet>
      <p align="center" style="line-height: 100%; word-spacing: 0; margin-
top: 0; margin-bottom: 0"><font face="Times New Roman" size="2"><font
color="red">Copyright
      © 2013 Mohammed Ahmad Salem, All rights reserved.</font><br>
      <b>Designed and best viewed @ 1024x768 resolution</b></font>
    </td>
  </tr>
</table>
<p>
&nbsp;
</p>
</body>
</html>
```

Listing A – 13: DoSTDM Web Applet Code (DoSTDM.html)

### A.2.2.2 DoSTDM Java Applet Code

```
// import required packages to run the class
import java.lang.String;
import java.io.*;
import java.io.File;
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import java.net.URL;
import java.applet.*;
import javax.swing.*;
import javax.swing.border.*;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.text.ParseException;
import java.util.Date;

public class DoSTDM extends JApplet
{
  // define class variables and arrays
  public static int counter;
  public static String protocolsList[] =
  { "", "TCP", "IP", "ICMP", "UDP", "Other" };
  public static String statusList[] =
  { "", "Inbound", "Outbound", "Accepted", "Rejected" };
}
```

```

public static String forecastingList[] = { "", "NN-MLP", "Compare MLP", "Holt-
Winter", "Regression", "Compare All", "Report" };
public static String epochsList[] = { "", "100", "200", "300", "400", "500",
"600", "700", "800", "900", "1000" };
public static String hiddenUnitsList[] =
{ "", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15" };
public static String mlpHILRLList[] =
{ "", "0.1", "0.2", "0.3", "0.4", "0.5", "0.6", "0.7", "0.8", "0.9", "1.0" };
public static String mlpHOLRLList[] =
{ "", "0.01", "0.02", "0.03", "0.04", "0.05", "0.06", "0.07", "0.08", "0.09", "0.1" };
public static String trainingPeriodList[] = { "" };
public static String [] daysList;
public static String [] yearsList;
public static String [] monthsList;
public static String [] t2daysList;
public static String [] t2yearsList;
public static String [] t2monthsList;
public static int startTrainIndex;
public static int lastTrainIndex;
public static int trainingPeriodLength;
public static String anyList[] = { "" };
public static String chartDataList[] = { "", "Actual", "Forecaste", "Actual vs.
Forecaste" };
public static String year;
public static String month;
public static String day;
public static String t2year;
public static String t2month;
public static String t2day;
private JButton startSelectedButton;
private JButton clearSelectedButton;
private ButtonGroup group;
private JLabel label;
private JLabel yearlabel;
private JLabel monthlabel;
private JLabel daylabel;
private JScrollPane slideImagePanel;
private JPanel textPanel;
private JPanel control1Panel;
private JPanel control2Panel;
private JPanel explorerControl1Panel;
private JPanel controlPanel;
private JPanel graphPanel;
private JPanel lowerPanel;
private JPanel appletPanel;
private JTabbedPane tabbedPane;
public JTextArea chartedDataTextArea;
private static ImageIcon returnedImage;
private URL dataPath;
public static int record;
public static int passedMinOfDay, passedMaxOfDay;
public static String [] dateArray;
public static String [] timeArray;
public static int [] inBoundArray;
public static int [] outBboundArray;
public static int [] acceptedArray;
public static int [] rejectedArray;
public static int [] tcp_in;
public static int [] tcp_out;
public static int [] tcp_accpt;
public static int [] tcp_rejct;
public static int [] ip_in;
public static int [] ip_out;
public static int [] ip_accpt;
public static int [] ip_rejct;

```

```

public static int [] icmp_in;
public static int [] icmp_out;
public static int [] icmp_accpt;
public static int [] icmp_rejct;
public static int [] udp_in;
public static int [] udp_out;
public static int [] udp_accpt;
public static int [] udp_rejct;
public static int [] oth_in;
public static int [] oth_out;
public static int [] oth_accpt;
public static int [] oth_rejct;
public static String [] trainDateArray;
public static String [] trainTimeArray;
public static int [] trainInBoundArray;
public static int [] trainOutBboundArray;
public static int [] trainAcceptedArray;
public static int [] trainRejectedArray;
public static int [] trainTcp_in;
public static int [] trainTcp_out;
public static int [] trainTcp_accpt;
public static int [] trainTcp_rejct;
public static int [] trainIp_in;
public static int [] trainIp_out;
public static int [] trainIp_accpt;
public static int [] trainIp_rejct;
public static int [] trainIcmp_in;
public static int [] trainIcmp_out;
public static int [] trainIcmp_accpt;
public static int [] trainIcmp_rejct;
public static int [] trainUdp_in;
public static int [] trainUdp_out;
public static int [] trainUdp_accpt;
public static int [] trainUdp_rejct;
public static int [] trainOth_in;
public static int [] trainOth_out;
public static int [] trainOth_accpt;
public static int [] trainOth_rejct;
public static String [] trainWeeklyDateArray;
public static String [] trainWeeklyTimeArray;
public static int [] trainWeeklyInBoundArray;
public static int [] trainWeeklyOutBboundArray;
public static int [] trainWeeklyAcceptedArray;
public static int [] trainWeeklyRejectedArray;
public static int [] trainWeeklyTcp_in;
public static int [] trainWeeklyTcp_out;
public static int [] trainWeeklyTcp_accpt;
public static int [] trainWeeklyTcp_rejct;
public static int [] trainWeeklyIp_in;
public static int [] trainWeeklyIp_out;
public static int [] trainWeeklyIp_accpt;
public static int [] trainWeeklyIp_rejct;
public static int [] trainWeeklyIcmp_in;
public static int [] trainWeeklyIcmp_out;
public static int [] trainWeeklyIcmp_accpt;
public static int [] trainWeeklyIcmp_rejct;
public static int [] trainWeeklyUdp_in;
public static int [] trainWeeklyUdp_out;
public static int [] trainWeeklyUdp_accpt;
public static int [] trainWeeklyUdp_rejct;
public static int [] trainWeeklyOth_in;
public static int [] trainWeeklyOth_out;
public static int [] trainWeeklyOth_accpt;
public static int [] trainWeeklyOth_rejct;
public static String [] wTempDateArray;

```

```

public static String [] wTempTimeArray;
public static int [] wTempInBoundArray;
public static int [] wTempOutBboundArray;
public static int [] wTempAcceptedArray;
public static int [] wTempRejectedArray;
public static int [] wTempTcp_in;
public static int [] wTempTcp_out;
public static int [] wTempTcp_accpt;
public static int [] wTempTcp_rejct;
public static int [] wTempIp_in;
public static int [] wTempIp_out;
public static int [] wTempIp_accpt;
public static int [] wTempIp_rejct;
public static int [] wTempIcmp_in;
public static int [] wTempIcmp_out;
public static int [] wTempIcmp_accpt;
public static int [] wTempIcmp_rejct;
public static int [] wTempUdp_in;
public static int [] wTempUdp_out;
public static int [] wTempUdp_accpt;
public static int [] wTempUdp_rejct;
public static int [] wTempOth_in;
public static int [] wTempOth_out;
public static int [] wTempOth_accpt;
public static int [] wTempOth_rejct;
public static int trainRecord;
public static String [] testDateArray;
public static String [] testTimeArray;
public static int [] testInBoundArray;
public static int [] testOutBboundArray;
public static int [] testAcceptedArray;
public static int [] testRejectedArray;
public static int [] testTcp_in;
public static int [] testTcp_out;
public static int [] testTcp_accpt;
public static int [] testTcp_rejct;
public static int [] testIp_in;
public static int [] testIp_out;
public static int [] testIp_accpt;
public static int [] testIp_rejct;
public static int [] testIcmp_in;
public static int [] testIcmp_out;
public static int [] testIcmp_accpt;
public static int [] testIcmp_rejct;
public static int [] testUdp_in;
public static int [] testUdp_out;
public static int [] testUdp_accpt;
public static int [] testUdp_rejct;
public static int [] testOth_in;
public static int [] testOth_out;
public static int [] testOth_accpt;
public static int [] testOth_rejct;
public static int [] testForecastTcp_in;
public static int [] testForecastTcp_out;
public static int [] testForecastTcp_accpt;
public static int [] testForecastIp_in;
public static int [] testForecastIp_out;
public static int [] testForecastIp_accpt;
public static int [] testForecastIcmp_in;
public static int [] testForecastIcmp_out;
public static int [] testForecastIcmp_accpt;
public static int [] testForecastUdp_in;
public static int [] testForecastUdp_out;
public static int [] testForecastUdp_accpt;
public static int [] testForecastOth_in;
public static int [] testForecastOth_out;

```

```

public static int [] testForecastOth_accpt;
public static int [] HWResiduals;
public static int [] MlpResiduals;
public static int [] MlpCompareResiduals;
public static int [] RegResiduals;
public static double HWME;
public static double HWMAD;
public static double HWRMSE;
public static double MlpME;
public static double MlpMAD;
public static double MlpRMSE;
public static double MlpWME;
public static double MlpWMAD;
public static double MlpWRMSE;
public static double RegME;
public static double RegMAD;
public static double RegRMSE;
public static String selDate;
public static String selProtocol;
public static String selDirection;
public static String selStatus;
public static String selChartType;
public static double [] cData = new double [16];
public static double [] calcMeError; // Calculated Reject array ME error
public static double [] calcMadError; // Calculated Reject array MAD error
public static double [] calcMseError; // Calculated Reject array MSE error

public static JTextArea t1;

/**
 * The DoSTDM() constructor initialize applet main page by
 * calling initMain() method.
 */

public DoSTDM()
{
// calling the initialization method to load applet main page
initMain();
}

/**
 * This method construct the initial applet layout and the selection
 * panel. Only SWING componenets are included.
 */
protected void initMain()
{
try {
// create a file object using the input file name
java.net.URL dataPath = DoSTDM.class.getResource("data / " +
"Database.CSV");

// create a buffered reader and attach it the file reader
BufferedReader fileInputStream = new BufferedReader( new
InputStreamReader (dataPath.openStream()));

// read file lines
String inputLine = fileInputStream.readLine();
inputLine = fileInputStream.readLine();
ArrayList<String> arl = new ArrayList<String>();
arl.add("");
record = 0;
while (inputLine != null)
{
StringTokenizer st = new StringTokenizer(inputLine, ",");
String rawData = st.nextToken();
int textLength = rawData.length();

```

```

        if (textLength == 8) { year = (String)rawData.substring(4,8); }
        else year = rawData.substring(5,9);
        if (!(ar1.contains(year)) { ar1.add(year); }
        inputLine = fileInputStream.readLine();
        record + + ;
    } // end of while loop

    yearsList = (String []) ar1.toArray (new String [0]);
    fileInputStream.close(); // close the input stream
} // close try statment

catch(IOException e) { }

// clear all the contnents from the current contnentPane, this is good
// when the user choose to return back to the main page of the applet

JPanel panel2 = new JPanel();
panel2.setLayout( new BorderLayout() );
panel2.add( new JButton( "North" ), BorderLayout.NORTH );
panel2.add( new JButton( "South" ), BorderLayout.SOUTH );
panel2.add( new JButton( "East" ), BorderLayout.EAST );
panel2.add( new JButton( "West" ), BorderLayout.WEST );
panel2.add( new JButton( "Center" ), BorderLayout.CENTER );
JTabbedPane tabbedPane = new JTabbedPane();
tabbedPane.setFont(new Font("Times", Font.BOLD, 12));
tabbedPane.addTab( "Baseline Data Explorer", initBaseLineTab());
tabbedPane.addTab( "Forecasting Analyser of Rejected Packets",
initForecastingTab());
getContentPane().removeAll();
getContentPane().setBackground(new Color(151,155,202));\
getContentPane().setLayout(new BorderLayout());
getContentPane().add("Center",tabbedPane);

// validate and repaint applet
tabbedPane.revalidate();
tabbedPane.repaint();
dataCollector();
} // end of method initMain()

protected static JPanel initBaseLineTab()
{
// construct combobox for day selection options
final JComboBox selectDayCombobox = new JComboBox(anyList);
selectDayCombobox.setForeground(Color.blue);
selectDayCombobox.setRenderer(new
AlignedListCellRenderer(SwingConstants.CENTER));
selectDayCombobox.setFont(new Font("Arial", Font.BOLD, 12));
selectDayCombobox.setToolTipText("Click on the drop-down arrow to select a
day of the week");

// construct combobox for month selection options
final JComboBox selectMonthCombobox = new JComboBox(anyList);
selectMonthCombobox.setForeground(Color.blue);
selectMonthCombobox.setRenderer(new
AlignedListCellRenderer(SwingConstants.CENTER));
selectMonthCombobox.setFont(new Font("Arial", Font.BOLD, 12));
selectMonthCombobox.setToolTipText("Click on the drop-down arrow to select a
month of the year");
selectMonthCombobox.addItemListener(new ItemListener()
{
public void itemStateChanged(ItemEvent e)
{
if (e.getStateChange() == ItemEvent.SELECTED)
{
String selectedMonth = (String)selectMonthCombobox.getSelectedItem();

```

```

selectedMonth = selectedMonth.substring(0,3);
try
{
// create a file object using the input file name
java.net.URL daysPath = DoSTDM.class.getResource("data / " +
"Database.CSV");

// create a buffered reader and attach it the file reader
BufferedReader daysFileInputStream = new BufferedReader( new
InputStreamReader (daysPath.openStream()));

// read file lines
String daysInputLine = daysFileInputStream.readLine();
daysInputLine = daysFileInputStream.readLine();
ArrayList<String> daysArrayList = new ArrayList<String>();
while (daysInputLine != null)
{
StringTokenizer daysSt = new StringTokenizer(daysInputLine, ",");
String daysData = daysSt.nextToken();
String yearValue = "";
String monthValue = "";
int daysLength = daysData.length();
if (daysLength == 8)
{
day = daysData.substring(0,1);
monthValue = daysData.substring(1,4);
yearValue = daysData.substring(4);
}
if (daysLength == 9)
{
day = daysData.substring(0,2);
monthValue = daysData.substring(2,5);
yearValue = daysData.substring(5);
}
if (!(daysArrayList.contains(day)) &&
(yearValue.equalsIgnoreCase(year)) &&
(monthValue.equalsIgnoreCase(selectedMonth)))
{
daysArrayList.add(day);
}
daysInputLine = daysFileInputStream.readLine();
} // end of while loop

daysList = (String []) daysArrayList.toArray (new String [0]);
selectDayCombobox.removeAllItems();
selectDayCombobox.addItem("");
for (int i = 0;i<daysList.length; i + + )
{
selectDayCombobox.addItem(daysList[i]);
}
daysFileInputStream.close(); // close the input stream
} // close try statement
catch(IOException edays) { }
} // end of inner class of year selection
} // end of item selection validation
});

// construct combobox for year selection options
final JComboBox selectYearCombobox = new JComboBox(yearsList);
selectYearCombobox.setForeground(Color.blue);
selectYearCombobox.setRenderer(new
AlignedListCellRenderer(SwingConstants.CENTER));
selectYearCombobox.setFont(new Font("Arial", Font.BOLD, 12));
selectYearCombobox.setToolTipText("Click on the drop-down arrow to select a
year");

```

```

selectYearCombobox.addActionListener(new ActionListener()
{
// inner class to detect the year selection action command
public void actionPerformed(ActionEvent e)
{
year = (String)selectYearCombobox.getSelectedItem();
try
{
// create a file object using the input file name
java.net.URL monthsPath = DoSTDM.class.getResource("data / " +
"Database.CSV");

// create a buffered reader and attach it the file reader
BufferedReader monthsFileInputStream = new BufferedReader( new
InputStreamReader (monthsPath.openStream()));

// read file lines
String monthsInputLine = monthsFileInputStream.readLine();
monthsInputLine = monthsFileInputStream.readLine();
ArrayList<String> monthsArrayList = new ArrayList<String>();
while (monthsInputLine != null)
{
StringTokenizer monthsSt = new StringTokenizer(monthsInputLine, ",");
String monthsData = monthsSt.nextToken();
String yearValue = "";
int monthsLength = monthsData.length();
if (monthsLength == 8)
{
month = monthsData.substring(1,4);
yearValue = monthsData.substring(4);
}
if (monthsLength == 9)
{
month = monthsData.substring(2,5);
yearValue = monthsData.substring(5);
}
if (!(monthsArrayList.contains(month)) &&
(yearValue.equalsIgnoreCase(year)))
{
monthsArrayList.add(month);
}
monthsInputLine = monthsFileInputStream.readLine();
} // end of while loop
monthsList = (String []) monthsArrayList.toArray (new String [0]);
selectMonthCombobox.removeAllItems();
for (int i = 0;i<monthsList.length; i++ )
{
if(monthsList[i].equalsIgnoreCase("jan")) { monthsList[i] = "January"; }
if(monthsList[i].equalsIgnoreCase("feb")) { monthsList[i] = "February"; }
if(monthsList[i].equalsIgnoreCase("mar")) { monthsList[i] = "March"; }
if(monthsList[i].equalsIgnoreCase("apr")) { monthsList[i] = "April"; }
if(monthsList[i].equalsIgnoreCase("may")) { monthsList[i] = "May"; }
if(monthsList[i].equalsIgnoreCase("jun")) { monthsList[i] = "June"; }
if(monthsList[i].equalsIgnoreCase("jul")) { monthsList[i] = "July"; }
if(monthsList[i].equalsIgnoreCase("aug")) { monthsList[i] = "August"; }
if(monthsList[i].equalsIgnoreCase("sep")) { monthsList[i] = "September"; }
if(monthsList[i].equalsIgnoreCase("oct")) { monthsList[i] = "October"; }
if(monthsList[i].equalsIgnoreCase("nov")) { monthsList[i] = "November"; }
if(monthsList[i].equalsIgnoreCase("dec")) { monthsList[i] = "December"; }
selectMonthCombobox.addItem(monthsList[i]);
}
monthsFileInputStream.close(); // close the input stream
} // close try statment
catch(IOException emonths) { }
} // end of inner class of year selection
} );

```

```

final JComboBox selectProtocolCombobox = new JComboBox(protocolsList);
selectProtocolCombobox.setForeground(Color.blue);
selectProtocolCombobox.setRenderer(new
AlignedListCellRenderer(SwingConstants.CENTER));
selectProtocolCombobox.setFont(new Font("Arial", Font.BOLD, 12));
selectProtocolCombobox.setToolTipText("Click on the drop-down arrow to
select a protocol");

// construct combobox for status selection optionsfinal JComboBox
selectStatusCombobox = new JComboBox(statusList);
selectStatusCombobox.setForeground(Color.blue);
selectStatusCombobox.setRenderer(new
AlignedListCellRenderer(SwingConstants.CENTER));
selectStatusCombobox.setFont(new Font("Arial", Font.BOLD, 12));
selectStatusCombobox.setToolTipText("Click on the drop-down arrow to select
a protocol status");

// set up the year JLabel and format it
JLabel yearLabel = new JLabel("Select Year ");
yearLabel.setBackground(Color.white);
yearLabel.setForeground(Color.black);
yearLabel.setHorizontalAlignment(SwingConstants.RIGHT);
yearLabel.setVerticalAlignment(SwingConstants.CENTER);
yearLabel.setOpaque(true);
yearLabel.setFont(new Font("Arial", Font.BOLD, 12));

// set up the month JLabel and format it
JLabel monthLabel = new JLabel("Select Month ");
monthLabel.setBackground(Color.white);
monthLabel.setForeground(Color.black);
monthLabel.setHorizontalAlignment(SwingConstants.RIGHT);
monthLabel.setVerticalAlignment(SwingConstants.CENTER);
monthLabel.setOpaque(true);
monthLabel.setFont(new Font("Arial", Font.BOLD, 12));

// set up the day JLabel and format it
JLabel dayLabel = new JLabel("Select Day ");
dayLabel.setBackground(Color.white);
dayLabel.setForeground(Color.black);
dayLabel.setHorizontalAlignment(SwingConstants.RIGHT);
dayLabel.setVerticalAlignment(SwingConstants.CENTER);
dayLabel.setOpaque(true);
dayLabel.setFont(new Font("Arial", Font.BOLD, 12));

// set up the protocol JLabel and format it
JLabel protocolLabel = new JLabel("Select Protocol ");
protocolLabel.setBackground(Color.white);
protocolLabel.setForeground(Color.black);
protocolLabel.setHorizontalAlignment(SwingConstants.RIGHT);
protocolLabel.setVerticalAlignment(SwingConstants.CENTER);
protocolLabel.setOpaque(true);
protocolLabel.setFont(new Font("Arial", Font.BOLD, 12));

// set up the status JLabel and format it
JLabel statusLabel = new JLabel("Select Status ");
statusLabel.setBackground(Color.white);
statusLabel.setForeground(Color.black);
statusLabel.setHorizontalAlignment(SwingConstants.RIGHT);
statusLabel.setVerticalAlignment(SwingConstants.CENTER);
statusLabel.setOpaque(true);
statusLabel.setFont(new Font("Arial", Font.BOLD, 12));
JPanel yearSelectionPanel = new JPanel();
yearSelectionPanel.setLayout(new GridLayout(1,2));
yearSelectionPanel.setBackground(Color.yellow);
yearSelectionPanel.add(yearLabel);

```

```

yearSelectionPanel.add(selectYearCombobox);
JPanel monthSelectionPanel = new JPanel();
monthSelectionPanel.setLayout(new GridLayout(1,2));
monthSelectionPanel.setBackground(Color.yellow);
monthSelectionPanel.add(monthLabel);
monthSelectionPanel.add(selectMonthCombobox);
JPanel daySelectionPanel = new JPanel();
daySelectionPanel.setLayout(new GridLayout(1,2));
daySelectionPanel.setBackground(Color.yellow);
daySelectionPanel.add(dayLabel);
daySelectionPanel.add(selectDayCombobox);
JPanel protocolSelectionPanel = new JPanel();
protocolSelectionPanel.setLayout(new GridLayout(1,2));
protocolSelectionPanel.setBackground(Color.yellow);
protocolSelectionPanel.add(protocolLabel);
protocolSelectionPanel.add(selectProtocolCombobox);
JPanel statusSelectionPanel = new JPanel();
statusSelectionPanel.setLayout(new GridLayout(1,2));
statusSelectionPanel.setBackground(Color.yellow);
statusSelectionPanel.add(statusLabel);
statusSelectionPanel.add(selectStatusCombobox);
final JPanel daily = new JPanel();
daily.setLayout(new BorderLayout());
daily.setPreferredSize(new Dimension(675,300));
final JPanel monthly = new JPanel();
monthly.setLayout(new BorderLayout());
monthly.setPreferredSize(new Dimension(675,300));
final JPanel yearly = new JPanel();
yearly.setLayout(new BorderLayout());
yearly.setPreferredSize(new Dimension(675,300));
final JLabel chartGuide = new JLabel(createImageIcon("images /
plot_guide.gif"));
// construct a panel to view graphs
final JPanel graphPanel = new JPanel();
graphPanel.setLayout(new BorderLayout());
graphPanel.setBackground(Color.white);
graphPanel.setBorder(new LineBorder(Color.white, 1));
graphPanel.setPreferredSize(new Dimension(675,445));
graphPanel.add("Center", chartGuide);
// construct a panel to view graphs
final JPanel graphTitle = new JPanel();
graphTitle.setLayout(new BorderLayout());
graphTitle.setBackground(Color.white);
graphTitle.setMaximumSize(new Dimension(675,45));
// construct a panel to view graphs
final JPanel graphics = new JPanel();
graphics.setLayout(new BorderLayout());
graphics.setBackground(Color.white);
graphics.setPreferredSize(new Dimension(675,945));
final JScrollPane chartScrolPanel = new JScrollPane();
chartScrolPanel.add(graphPanel);
chartScrolPanel.setViewportView(graphPanel);
final JTextArea chartedDataTextArea = new JTextArea();
chartedDataTextArea.setEditable(false);
chartedDataTextArea.setBackground(Color.white);
chartedDataTextArea.setForeground(Color.blue);
chartedDataTextArea.setBorder(new LineBorder(Color.white, 1));
chartedDataTextArea.setFont(new Font("Courier", Font.BOLD, 12));
chartedDataTextArea.setLineWrap(true);
// construct the starting charting action button
JButton startSelectedButton = new JButton("Explore Data");
startSelectedButton.setHorizontalAlignment(SwingConstants.CENTER);
startSelectedButton.setVerticalAlignment(SwingConstants.CENTER);
startSelectedButton.setForeground(new Color(0,151,82));
startSelectedButton.setFont(new Font("Arial", Font.BOLD, 12));
startSelectedButton.addActionListener(new ActionListener()

```

```

{
// inner class to detect the action command
public void actionPerformed(ActionEvent eStart)
{
// check if user selection is valid for exploring data charts
if(((String)selectYearCombobox.getSelectedItem() != "") &&
((String)selectMonthCombobox.getSelectedItem() != "") &&
((String)selectDayCombobox.getSelectedItem() != "") &&
((String)selectProtocolCombobox.getSelectedItem() != "") &&
((String)selectStatusCombobox.getSelectedItem() != ""))
{
graphPanel.removeAll();
graphTitle.removeAll();
String selYear = (String)selectYearCombobox.getSelectedItem();
String selMonth =
((String)selectMonthCombobox.getSelectedItem()).substring(0,3);
String selDay = (String)selectDayCombobox.getSelectedItem();
String selProtocol = (String)selectProtocolCombobox.getSelectedItem();
String selStatus = (String)selectStatusCombobox.getSelectedItem();
String selDate = selDay + selMonth + selYear;
String graphYear = selYear;
String graphMonth = (String)selectMonthCombobox.getSelectedItem();
String graphDay = selDay;
JLabel graphLabel = new JLabel("Summary of " + selStatus + " " +
selProtocol + " Packet Activities");
graphLabel.setHorizontalAlignment(SwingConstants.CENTER);
graphLabel.setVerticalAlignment(SwingConstants.CENTER);
graphLabel.setBackground(Color.white);
graphLabel.setForeground(Color.RED);
graphLabel.setOpaque(true);
graphLabel.setFont(new Font("Arial", Font.BOLD, 16));
graphTitle.add("Center", graphLabel);
if(selStatus.equalsIgnoreCase("Inbound")) { selStatus = "in"; }
if(selStatus.equalsIgnoreCase("Outbound")) { selStatus = "out"; }
if(selStatus.equalsIgnoreCase("Accepted")) { selStatus = "accpt"; }
if(selStatus.equalsIgnoreCase("Rejected")) { selStatus = "rejct"; }
if(selProtocol.equalsIgnoreCase("Other")) { selProtocol = "oth"; }
daily.add("Center", new DailyGraph(selDate, selProtocol, selStatus,
record, graphYear, graphMonth, graphDay));
monthly.add("Center", new MonthlyGraph(selDate, selProtocol,
selStatus, record, graphYear, graphMonth, daysList.length));
yearly.add("Center", new YearlyGraph(selProtocol, selStatus, record,
graphYear, monthsList.length));
graphPanel.setLayout(new GridLayout(3,1));
graphPanel.setPreferredSize(new Dimension(675,900));
graphPanel.setBorder(new LineBorder(Color.white, 1));
graphPanel.add(daily);
graphPanel.add(monthly);
graphPanel.add(yearly);
graphics.add("Center", graphTitle);
graphics.add("South", graphPanel);
chartScrolPanel.setViewportView(graphics);
graphics.revalidate();
graphics.repaint();
chartedDataTextArea.setBorder(new LineBorder(Color.red, 1));
chartedDataTextArea.setText(null);
chartedDataTextArea.append(" Date: " + selDate + "\n");
chartedDataTextArea.append(" =====" + "\n");

chartedDataTextArea.append(" Day Minimum: " + passedMinOfDay + "\n");
chartedDataTextArea.append(" Day Maximum: " + passedMaxOfDay + "\n");
chartedDataTextArea.append(" -----" + "\n");
}
else
{

```

```

if(((String)selectYearCombobox.getSelectedItemAt() == "") ||
((String)selectMonthCombobox.getSelectedItemAt() == "") ||
((String)selectDayCombobox.getSelectedItemAt() == "") ||
((String)selectProtocolCombobox.getSelectedItemAt() == "") ||
((String)selectStatusCombobox.getSelectedItemAt() == ""))
{
    // if user selection is invalid, send error message and load
    // guideline steps image to show user how to select to chart
    graphPanel.removeAll();
    graphPanel.setLayout(new BorderLayout());
    graphPanel.setPreferredSize(new Dimension(675,445));
    graphPanel.setBorder(new LineBorder(Color.white, 1));
    graphPanel.add("Center", chartGuide);
    chartScrolPanel.setViewportView(graphPanel);
    graphPanel.revalidate();
    graphPanel.repaint();
    JOptionPane.showMessageDialog(null, "<html><center>Your
explorer parameters are incomplete.<br>Please read the guide
line steps to graph selected data.< / center>< / html>",
"Missing Parameters", JOptionPane.INFORMATION_MESSAGE);
}
} // end of inner class
} );

// construct the clearing selection action button
JButton clearSelectedButton = new JButton("Clear Selection");
clearSelectedButton.setHorizontalAlignment(SwingConstants.CENTER);
clearSelectedButton.setVerticalAlignment(SwingConstants.CENTER);
clearSelectedButton.setForeground(Color.RED);
clearSelectedButton.setFont(new Font("Arial", Font.BOLD, 12));
clearSelectedButton.addActionListener(new ActionListener()
{
    // inner class to detect the action command to reset selections
    public void actionPerformed(ActionEvent eClear)
    {
        selectYearCombobox.setSelectedItem("");
        selectMonthCombobox.setSelectedItem("");
        selectDayCombobox.setSelectedItem("");
        selectProtocolCombobox.setSelectedItem("");
        selectStatusCombobox.setSelectedItem("");
        chartedDataTextArea.setText(null);
        chartedDataTextArea.setBorder(new LineBorder(Color.white, 1));
        graphPanel.removeAll();
        graphPanel.setLayout(new BorderLayout());
        graphPanel.setPreferredSize(new Dimension(675,445));
        graphPanel.setBorder(new LineBorder(Color.white, 1));
        graphPanel.add("Center", chartGuide);
        chartScrolPanel.setViewportView(graphPanel);
        graphPanel.revalidate();
        graphPanel.repaint();
    } // end of inner class
} );

// construct a panel to hold control data explorer buttons
JPanel startPanel = new JPanel();
startPanel.setLayout(new BorderLayout());
startPanel.setBackground(Color.white);
startPanel.setPreferredSize(new Dimension(200,35));
startPanel.add(new JLabel("      "), BorderLayout.SOUTH);
startPanel.add(new JLabel("      "), BorderLayout.EAST);
startPanel.add(new JLabel("      "), BorderLayout.WEST);
startPanel.add(startSelectedButton, BorderLayout.CENTER);
JPanel clearPanel = new JPanel();
clearPanel.setLayout(new BorderLayout());
clearPanel.setBackground(Color.white);

```

```

clearPanel.setPreferredSize(new Dimension(200,35));
clearPanel.add(new JLabel("      "), BorderLayout.SOUTH);
clearPanel.add(new JLabel("      "), BorderLayout.EAST);
clearPanel.add(new JLabel("      "), BorderLayout.WEST);
clearPanel.add(clearSelectedButton, BorderLayout.CENTER);
final JPanel explorerControllPanel = new JPanel();
explorerControllPanel.setLayout(new BorderLayout());
explorerControllPanel.setPreferredSize(new Dimension(200,70));
explorerControllPanel.add("Center",startPanel);
explorerControllPanel.add("South",clearPanel);

// construct a panel to hold combobox and action buttons
JPanel controllPanel = new JPanel();
controllPanel.setLayout(new GridLayout(11,1));
controllPanel.setBackground(Color.white);
controllPanel.setPreferredSize(new Dimension(200,200));
controllPanel.add(new JLabel("      "));
controllPanel.add(yearSelectionPanel);
controllPanel.add(new JLabel("      "));
controllPanel.add(monthSelectionPanel);
controllPanel.add(new JLabel("      "));
controllPanel.add(daySelectionPanel);
controllPanel.add(new JLabel("      "));
controllPanel.add(protocolSelectionPanel);
controllPanel.add(new JLabel("      "));
controllPanel.add(statusSelectionPanel);
controllPanel.add(new JLabel("      "));

// charting additional control with text area
JPanel control2Panel = new JPanel();
control2Panel.setLayout(new BorderLayout());
control2Panel.setBackground(Color.white);
control2Panel.setBorder(new LineBorder(Color.white, 1));
control2Panel.setPreferredSize(new Dimension(200,250));
control2Panel.add("North",explorerControllPanel);
control2Panel.add("Center",chartedDataTextArea);
JPanel controlPanel = new JPanel();
controlPanel.setLayout(new BorderLayout());
controlPanel.setBackground(Color.black);
controlPanel.setBorder(new LineBorder(Color.black, 1));
controlPanel.setPreferredSize(new Dimension(200,470));
controlPanel.add("Center", controllPanel);
controlPanel.add("South", control2Panel);

// construct a applet control & display panel
JPanel lowerPanel = new JPanel();
lowerPanel.setLayout(new BorderLayout());
lowerPanel.setBackground(Color.white);
lowerPanel.setBorder(new LineBorder(Color.black, 1));
lowerPanel.setPreferredSize(new Dimension(900,470));
lowerPanel.add("West", controlPanel);
lowerPanel.add("Center", chartScrolPanel);
return lowerPanel;

} // end method initBaseLineTab()

protected static JPanel initForecastingTab()
{
// construct a panel for forecasting graphs
final JPanel t2graphPanel = new JPanel();
t2graphPanel.setLayout(new BorderLayout());
t2graphPanel.setBackground(Color.white);
t2graphPanel.setBorder(new LineBorder(Color.white, 1));
t2graphPanel.setPreferredSize(new Dimension(675,445));
t2graphPanel.add("Center",new JLabel(createImageIcon("images /
forecasting_guide.gif")));
final JScrollPane t2chartScrolPanel = new JScrollPane();

```

```

t2chartScrolPanel.add(t2graphPanel);
t2chartScrolPanel.setViewportView(t2graphPanel);
final JPanel t2graphics = new JPanel();
t2graphics.setLayout(new BorderLayout());
t2graphics.setBackground(Color.white);
t2graphics.setPreferredSize(new Dimension(675,765));
final JPanel t2graphTitle = new JPanel();
t2graphTitle.setLayout(new BorderLayout());
t2graphTitle.setBackground(Color.white);
t2graphTitle.setMaximumSize(new Dimension(675,65));
final JPanel t2ForecastingGraph = new JPanel();
t2ForecastingGraph.setLayout(new BorderLayout());
t2ForecastingGraph.setPreferredSize(new Dimension(675,350));
final JPanel t2ResidualsGraph = new JPanel();
t2ResidualsGraph.setLayout(new BorderLayout());
t2ResidualsGraph.setPreferredSize(new Dimension(675,350));

// set up the year JLabel and format it
JLabel t2yearLabel = new JLabel("Select Year ");
t2yearLabel.setBackground(Color.white);
t2yearLabel.setForeground(Color.black);
t2yearLabel.setHorizontalAlignment(SwingConstants.RIGHT);
t2yearLabel.setVerticalAlignment(SwingConstants.CENTER);
t2yearLabel.setOpaque(true);
t2yearLabel.setFont(new Font("Arial", Font.BOLD, 12));

// set up the month JLabel and format it
JLabel t2monthLabel = new JLabel("Select Month ");
t2monthLabel.setBackground(Color.white);
t2monthLabel.setForeground(Color.black);
t2monthLabel.setHorizontalAlignment(SwingConstants.RIGHT);
t2monthLabel.setVerticalAlignment(SwingConstants.CENTER);
t2monthLabel.setOpaque(true);
t2monthLabel.setFont(new Font("Arial", Font.BOLD, 12));

// set up the day JLabel and format it
JLabel t2dayLabel = new JLabel("Select Day ");
t2dayLabel.setBackground(Color.white);
t2dayLabel.setForeground(Color.black);
t2dayLabel.setHorizontalAlignment(SwingConstants.RIGHT);
t2dayLabel.setVerticalAlignment(SwingConstants.CENTER);
t2dayLabel.setOpaque(true);
t2dayLabel.setFont(new Font("Arial", Font.BOLD, 12));

// set up the forecasting method JLabel and format it
JLabel t2methodLabel = new JLabel("Select Method ");
t2methodLabel.setBackground(Color.white);
t2methodLabel.setForeground(Color.black);
t2methodLabel.setHorizontalAlignment(SwingConstants.RIGHT);
t2methodLabel.setVerticalAlignment(SwingConstants.CENTER);
t2methodLabel.setOpaque(true);
t2methodLabel.setFont(new Font("Arial", Font.BOLD, 12));

// set up the forecasting method Training period JLabel and format it
JLabel t2trainingPeriodLabel = new JLabel("Training Period ");
t2trainingPeriodLabel.setBackground(Color.white);
t2trainingPeriodLabel.setForeground(Color.black);
t2trainingPeriodLabel.setHorizontalAlignment(SwingConstants.RIGHT);
t2trainingPeriodLabel.setVerticalAlignment(SwingConstants.CENTER);
t2trainingPeriodLabel.setOpaque(true);
t2trainingPeriodLabel.setFont(new Font("Arial", Font.BOLD, 12));

// set up the forecasting Training Epochs JLabel and format it
JLabel t2trainingEpochsLabel = new JLabel("MLP Epochs ");
t2trainingEpochsLabel.setBackground(Color.white);
t2trainingEpochsLabel.setForeground(Color.black);

```

```

        t2trainingEpochsLabel.setHorizontalAlignment(SwingConstants.RIGHT);
        t2trainingEpochsLabel.setVerticalAlignment(SwingConstants.CENTER);
        t2trainingEpochsLabel.setOpaque(true);
        t2trainingEpochsLabel.setFont(new Font("Arial", Font.BOLD, 12));

// set up the forecasting Training Hidden Units JLabel and format it
JLabel t2trainingHiddenUnitsLabel = new JLabel("NN-MLP Nodes ");
t2trainingHiddenUnitsLabel.setBackground(Color.white);
t2trainingHiddenUnitsLabel.setForeground(Color.black);
t2trainingHiddenUnitsLabel.setHorizontalAlignment(SwingConstants.RIGH
T);
t2trainingHiddenUnitsLabel.setVerticalAlignment(SwingConstants.CENTER
);
t2trainingHiddenUnitsLabel.setOpaque(true);
t2trainingHiddenUnitsLabel.setFont(new Font("Arial", Font.BOLD, 12));

// set up the forecasting Training Input Hidden Units Learning Rate JLabel
// and format it
JLabel t2trainingHILRLLabel = new JLabel("NN-MLP IHLR ");
t2trainingHILRLLabel.setBackground(Color.white);
t2trainingHILRLLabel.setForeground(Color.black);
t2trainingHILRLLabel.setHorizontalAlignment(SwingConstants.RIGHT);
t2trainingHILRLLabel.setVerticalAlignment(SwingConstants.CENTER);
t2trainingHILRLLabel.setOpaque(true);
t2trainingHILRLLabel.setFont(new Font("Arial", Font.BOLD, 12));

// set up the forecasting Training Output Hidden Units Learning Rate JLabel
// and format it
JLabel t2trainingHOLRLLabel = new JLabel("NN-MLP OHLR ");
t2trainingHOLRLLabel.setBackground(Color.white);
t2trainingHOLRLLabel.setForeground(Color.black);
t2trainingHOLRLLabel.setHorizontalAlignment(SwingConstants.RIGHT);
t2trainingHOLRLLabel.setVerticalAlignment(SwingConstants.CENTER);
t2trainingHOLRLLabel.setOpaque(true);
t2trainingHOLRLLabel.setFont(new Font("Arial", Font.BOLD, 12));

final JComboBox t2selectTrainingPeriodCombobox = new JComboBox(anyList);
t2selectTrainingPeriodCombobox.setForeground(Color.blue);
t2selectTrainingPeriodCombobox.setRenderer(new
AlignedListCellRenderer(SwingConstants.CENTER));
t2selectTrainingPeriodCombobox.setFont(new Font("Arial", Font.BOLD, 12));
t2selectTrainingPeriodCombobox.setToolTipText("Click on the drop-down arrow
to set training period");
final JComboBox t2selectTrainingEpochsCombobox = new JComboBox(epochsList);
t2selectTrainingEpochsCombobox.setForeground(Color.blue);
t2selectTrainingEpochsCombobox.setRenderer(new AlignedListCellRenderer(
SwingConstants.CENTER));
t2selectTrainingEpochsCombobox.setFont(new Font("Arial", Font.BOLD, 12));
t2selectTrainingEpochsCombobox.setToolTipText("Click on the drop-down arrow
to set training epochs");

t2selectTrainingEpochsCombobox.setSelectedIndex(1);
final JComboBox t2selectTrainingHiddenUnitsCombobox = new
JComboBox(hiddenUnitsList);
t2selectTrainingHiddenUnitsCombobox.setForeground(Color.blue);
t2selectTrainingHiddenUnitsCombobox.setRenderer(new
AlignedListCellRenderer(SwingConstants.CENTER));
t2selectTrainingHiddenUnitsCombobox.setFont(new Font("Arial", Font.BOLD,
12));
t2selectTrainingHiddenUnitsCombobox.setToolTipText("Click on the drop-down
arrow to set training hidden units");
t2selectTrainingHiddenUnitsCombobox.setSelectedIndex(3);
final JComboBox t2selectTrainingHILRCombobox = new JComboBox(mlpHILRList);
t2selectTrainingHILRCombobox.setForeground(Color.blue);
t2selectTrainingHILRCombobox.setRenderer(new
AlignedListCellRenderer(SwingConstants.CENTER));
t2selectTrainingHILRCombobox.setFont(new Font("Arial", Font.BOLD, 12));

```

```

t2selectTrainingHILRCombobox.setToolTipText("Click on the drop-down arrow to
set hidden input units learning rate");
t2selectTrainingHILRCombobox.setSelectedIndex(7);
final JComboBox t2selectTrainingHOLRCombobox = new JComboBox(mlpHOLRList);
t2selectTrainingHOLRCombobox.setForeground(Color.blue);
t2selectTrainingHOLRCombobox.setRenderer(new
AlignedListCellRenderer(SwingConstants.CENTER));
t2selectTrainingHOLRCombobox.setFont(new Font("Arial", Font.BOLD, 12));
t2selectTrainingHOLRCombobox.setToolTipText("Click on the drop-down arrow to
set hidden output units learning rate");
t2selectTrainingHOLRCombobox.setSelectedIndex(7);
final JPanel t2trainingPeriodSelectionPanel = new JPanel();
t2trainingPeriodSelectionPanel.setLayout(new GridLayout(1,2));
t2trainingPeriodSelectionPanel.setBackground(Color.yellow);
t2trainingPeriodSelectionPanel.add(t2trainingPeriodLabel);
t2trainingPeriodSelectionPanel.add(t2selectTrainingPeriodCombobox);
t2trainingPeriodSelectionPanel.setVisible( false );
final JPanel t2trainingEpochsSelectionPanel = new JPanel();
t2trainingEpochsSelectionPanel.setLayout(new GridLayout(1,2));
t2trainingEpochsSelectionPanel.setBackground(Color.yellow);
t2trainingEpochsSelectionPanel.add(t2trainingEpochsLabel);
t2trainingEpochsSelectionPanel.add(t2selectTrainingEpochsCombobox);
t2trainingEpochsSelectionPanel.setVisible( false );
final JPanel t2trainingHiddenUnitsSelectionPanel = new JPanel();
t2trainingHiddenUnitsSelectionPanel.setLayout(new GridLayout(1,2));
t2trainingHiddenUnitsSelectionPanel.setBackground(Color.yellow);
t2trainingHiddenUnitsSelectionPanel.add(t2trainingHiddenUnitsLabel);
t2trainingHiddenUnitsSelectionPanel.add(t2selectTrainingHiddenUnitsCombobox)
;
t2trainingHiddenUnitsSelectionPanel.setVisible( false );
final JPanel t2trainingHILRSelectionPanel = new JPanel();
t2trainingHILRSelectionPanel.setLayout(new GridLayout(1,2));
t2trainingHILRSelectionPanel.setBackground(Color.yellow);
t2trainingHILRSelectionPanel.add(t2trainingHILRLabel);
t2trainingHILRSelectionPanel.add(t2selectTrainingHILRCombobox);
t2trainingHILRSelectionPanel.setVisible( false );
final JPanel t2trainingHOLRSelectionPanel = new JPanel();
t2trainingHOLRSelectionPanel.setLayout(new GridLayout(1,2));
t2trainingHOLRSelectionPanel.setBackground(Color.yellow);
t2trainingHOLRSelectionPanel.add(t2trainingHOLRLabel);
t2trainingHOLRSelectionPanel.add(t2selectTrainingHOLRCombobox);
t2trainingHOLRSelectionPanel.setVisible( false );
final JComboBox t2selectDayCombobox = new JComboBox(anyList);
t2selectDayCombobox.setForeground(Color.blue);
t2selectDayCombobox.setRenderer(new AlignedListCellRenderer(
SwingConstants.CENTER));
t2selectDayCombobox.setFont(new Font("Arial", Font.BOLD, 12));
t2selectDayCombobox.setToolTipText("Click on the drop-down arrow to select a
day of the week");

// construct combobox for month selection options
final JComboBox t2selectMonthCombobox = new JComboBox(anyList);
t2selectMonthCombobox.setForeground(Color.blue);
t2selectMonthCombobox.setRenderer(new
AlignedListCellRenderer(SwingConstants.CENTER));
t2selectMonthCombobox.setFont(new Font("Arial", Font.BOLD, 12));
t2selectMonthCombobox.setToolTipText("Click on the drop-down arrow to select
a month of the year");
t2selectMonthCombobox.setSelectedIndex(0);
t2selectMonthCombobox.addItemListener(new ItemListener()
{ // inner class to detect status change in the selection
public void itemStateChanged(ItemEvent e)
{
if (e.getStateChange() == ItemEvent.SELECTED)

```

```

{
String t2selectedMonth =
(String)t2selectMonthCombobox.getSelectedItem();
t2selectedMonth = t2selectedMonth.substring(0,3);
try
{
// create a file object using the input file name
java.net.URL t2daysPath = DoSTDM.class.getResource("data / " +
"Database.CSV");

// create a buffered reader and attach it the file reader
BufferedReader t2daysFileInputStream = new BufferedReader( new
InputStreamReader(t2daysPath.openStream()));

// read file lines
String t2daysInputLine = t2daysFileInputStream.readLine();
t2daysInputLine = t2daysFileInputStream.readLine();
ArrayList<String> t2daysArrayList = new ArrayList<String>();
while (t2daysInputLine != null)
{
StringTokenizer t2daysSt = new
StringTokenizer(t2daysInputLine, ",");
String t2daysData = t2daysSt.nextToken();
String t2yearValue = "";
String t2monthValue = "";
int t2daysLength = t2daysData.length();
if (t2daysLength == 8)
{
t2day = t2daysData.substring(0,1);
t2monthValue = t2daysData.substring(1,4);
t2yearValue = t2daysData.substring(4);
}
if (t2daysLength == 9)
{
t2day = t2daysData.substring(0,2);
t2monthValue = t2daysData.substring(2,5);
t2yearValue = t2daysData.substring(5);
}
if (!(t2daysArrayList.contains(t2day)) &&
(t2yearValue.equalsIgnoreCase(t2year)) &&
(t2monthValue.equalsIgnoreCase(t2selectedMonth)))
{ t2daysArrayList.add(t2day); }
t2daysInputLine = t2daysFileInputStream.readLine();
} // end of while loop
t2daysList = (String []) t2daysArrayList.toArray (new String [0]);
t2selectDayCombobox.removeAllItems();
t2selectMethodCombobox.removeAllItems();
t2selectTrainingPeriodCombobox.removeAllItems();
t2selectDayCombobox.addItem("");
// modified for 1 week history of data considerations
if ((t2monthsList[0].substring(0,3)).equalsIgnoreCase(
t2selectedMonth))

{
for (int i = 7;i<t2daysList.length; i++)
{
t2selectDayCombobox.addItem(t2daysList[i]);
}
}
else
{
for (int i = 0;i<t2daysList.length; i++)
{
t2selectDayCombobox.addItem(t2daysList[i]);
}
}
}
}

```

```

        t2daysFileInputStream.close(); // close the input stream
    } // close try statement
    catch(IOException t2edays) { }
    } // end of inner class of year selection
} // end of item selection validation
} );

// construct combobox for year selection options
final JComboBox t2selectYearCombobox = new JComboBox(yearsList);
t2selectYearCombobox.setForeground(Color.blue);
t2selectYearCombobox.setRenderer(new
AlignedListCellRenderer(SwingConstants.CENTER));
t2selectYearCombobox.setFont(new Font("Arial", Font.BOLD, 12));
t2selectYearCombobox.setToolTipText("Click on the drop-down arrow to select
a year");
t2selectYearCombobox.addActionListener(new ActionListener()
{
    // inner class to detect the year selection action command
    public void actionPerformed(ActionEvent e)
    {
        t2year = (String)t2selectYearCombobox.getSelectedItem();
        try
        {
            // create a file object using the input file name
            java.net.URL t2monthsPath = DoSTDM.class.getResource("data / " +
"Database.CSV");

            // create a buffered reader and attach it the file reader
            BufferedReader t2monthsFileInputStream = new BufferedReader( new
InputStreamReader(t2monthsPath.openStream()));

            // read file lines
            String t2monthsInputLine = t2monthsFileInputStream.readLine();
            t2monthsInputLine = t2monthsFileInputStream.readLine();
            ArrayList<String> t2monthsArrayList = new ArrayList<String>();
            while (t2monthsInputLine != null)
            {
                StringTokenizer t2monthsSt = new
StringTokenizer(t2monthsInputLine, ",");
                String t2monthsData = t2monthsSt.nextToken();
                String t2yearValue = "";
                int t2monthsLength = t2monthsData.length();
                if (t2monthsLength == 8)
                {
                    t2month = t2monthsData.substring(1,4);
                    t2yearValue = t2monthsData.substring(4);
                }
                if (t2monthsLength == 9)
                {
                    t2month = t2monthsData.substring(2,5);
                    t2yearValue = t2monthsData.substring(5);
                }

                if (!(t2monthsArrayList.contains(t2month)) &&
(t2yearValue.equalsIgnoreCase(t2year)))
                {
                    t2monthsArrayList.add(t2month);
                }
            }
            t2monthsInputLine = t2monthsFileInputStream.readLine();
        } // end of while loop
        t2monthsList = (String []) t2monthsArrayList.toArray (new String
[0]);
        t2selectMonthCombobox.removeAllItems();
        t2selectDayCombobox.removeAllItems();
        t2selectMethodCombobox.removeAllItems();
        t2selectTrainingPeriodCombobox.removeAllItems();
    }
});

```

```

for (int i = 0; i < t2monthsList.length; i++)
{
    if (t2monthsList[i].equalsIgnoreCase("jan"))
    { t2monthsList[i] = "January"; }
    if (t2monthsList[i].equalsIgnoreCase("feb"))
    { t2monthsList[i] = "February"; }
    if (t2monthsList[i].equalsIgnoreCase("mar"))
    { t2monthsList[i] = "March"; }
    if (t2monthsList[i].equalsIgnoreCase("apr"))
    { t2monthsList[i] = "April"; }
    if (t2monthsList[i].equalsIgnoreCase("may"))
    { t2monthsList[i] = "May"; }
    if (t2monthsList[i].equalsIgnoreCase("jun"))
    { t2monthsList[i] = "June"; }
    if (t2monthsList[i].equalsIgnoreCase("jul"))
    { t2monthsList[i] = "July"; }
    if (t2monthsList[i].equalsIgnoreCase("aug"))
    { t2monthsList[i] = "August"; }
    if (t2monthsList[i].equalsIgnoreCase("sep"))
    { t2monthsList[i] = "September"; }
    if (t2monthsList[i].equalsIgnoreCase("oct"))
    { t2monthsList[i] = "October"; }
    if (t2monthsList[i].equalsIgnoreCase("nov"))
    { t2monthsList[i] = "November"; }
    if (t2monthsList[i].equalsIgnoreCase("dec"))
    { t2monthsList[i] = "December"; }
    t2selectMonthCombobox.addItem(t2monthsList[i]);
}
t2selectTrainingPeriodCombobox.addItem("");
t2monthsFileInputStream.close(); // close the input stream
} // close try statement
catch (IOException t2emonths) { }
} // end of inner class of year selection
});

final JComboBox t2selectMethodCombobox = new JComboBox(forecastingList);
t2selectMethodCombobox.setForeground(Color.blue);
t2selectMethodCombobox.setRenderer(new AlignedListCellRenderer(
    SwingConstants.CENTER));
t2selectMethodCombobox.setFont(new Font("Arial", Font.BOLD, 12));
t2selectMethodCombobox.setToolTipText("Click on the drop-down arrow to
select a forecasting method");
t2selectMethodCombobox.addItemListener(new ItemListener()
{
    public void itemStateChanged(ItemEvent e)
    {
        if (e.getStateChange() == ItemEvent.SELECTED)
        {
            String methodSelected =
                (String)t2selectMethodCombobox.getSelectedItem();
            System.out.println("selection is: " + methodSelected);
            if (methodSelected.equalsIgnoreCase("NN-MLP") ||
                methodSelected.equalsIgnoreCase("Compare All") ||
                methodSelected.equalsIgnoreCase("Compare MLP"))
            {
                t2trainingPeriodSelectionPanel.setVisible( true );
                t2trainingEpochsSelectionPanel.setVisible( true );
                t2trainingHiddenUnitsSelectionPanel.setVisible( true );
                t2trainingHILRSelectionPanel.setVisible( true );
                t2trainingHOLRSelectionPanel.setVisible( true );
            }
            if (methodSelected.equalsIgnoreCase("Holt-Winter"))
            {
                t2trainingPeriodSelectionPanel.setVisible( true );
                t2trainingEpochsSelectionPanel.setVisible( false );
                t2trainingHiddenUnitsSelectionPanel.setVisible( false );
            }
        }
    }
});

```

```

t2trainingHILRSelectionPanel.setVisible( false );
t2trainingHOLRSelectionPanel.setVisible( false );
}
if (methodSelected.equalsIgnoreCase("Regression"))
{
t2trainingPeriodSelectionPanel.setVisible( true );
t2trainingEpochsSelectionPanel.setVisible( false );
t2trainingHiddenUnitsSelectionPanel.setVisible( false );
t2trainingHILRSelectionPanel.setVisible( false );
t2trainingHOLRSelectionPanel.setVisible( false );
}
if (methodSelected.equalsIgnoreCase("Report"))
{
t2trainingPeriodSelectionPanel.setVisible( false );
t2trainingEpochsSelectionPanel.setVisible( false );
t2trainingHiddenUnitsSelectionPanel.setVisible( false );
t2trainingHILRSelectionPanel.setVisible( false );
t2trainingHOLRSelectionPanel.setVisible( false );
}
try
{
String t2selectedTrainingDay =
(String)t2selectDayCombobox.getSelectedItem();
String t2selectedTrainingMonth =
(String)t2selectMonthCombobox.getSelectedItem();
t2selectedTrainingMonth = t2selectedTrainingMonth.substring(0,3);
String t2selectedTrainingYear =
(String)t2selectYearCombobox.getSelectedItem();
String t2selectedTrainingDate = t2selectedTrainingDay +
t2selectedTrainingMonth + t2selectedTrainingYear;
startTrainIndex = 0;
lastTrainIndex = 0;
for(int s = 0;s<record;s + + )
{
String t2TrainYear = dateArray[s];
if (t2TrainYear.length() == 8)
{ t2TrainYear = t2TrainYear.substring(4); }
if (t2TrainYear.length() == 9)
{ t2TrainYear = t2TrainYear.substring(5); }
if (t2TrainYear.equalsIgnoreCase(t2selectedTrainingYear))
{
startTrainIndex = s;
break;
}
}
}
for(int i = 0;i<record;i++)
{
String t2TrainYear = dateArray[i];
if (t2TrainYear.length() == 8)
{ t2TrainYear = t2TrainYear.substring(4); }
if (t2TrainYear.length() == 9)
{ t2TrainYear = t2TrainYear.substring(5); }
if (dateArray[i].equalsIgnoreCase(t2selectedTrainingDate))
{ lastTrainIndex = i;
break; }}

System.out.println("Training Start Point = " + startTrainIndex + " &
End Point = " + lastTrainIndex);
trainRecord = lastTrainIndex - startTrainIndex;
trainDateArray = new String [trainRecord];
trainTimeArray = new String [trainRecord];
trainInBoundArray = new int [trainRecord];
trainOutBboundArray = new int [trainRecord];
trainAcceptedArray = new int [trainRecord];
trainRejectedArray = new int [trainRecord];
trainTcp_in = new int [trainRecord];
trainTcp_out = new int [trainRecord];

```

```

trainTcp_accpt = new int [trainRecord];
trainTcp_rejct = new int [trainRecord];
trainIp_in = new int [trainRecord];
trainIp_out = new int [trainRecord];
trainIp_accpt = new int [trainRecord];
trainIp_rejct = new int [trainRecord];
trainIcmp_in = new int [trainRecord];
trainIcmp_out = new int [trainRecord];
trainIcmp_accpt = new int [trainRecord];
trainIcmp_rejct = new int [trainRecord];
trainUdp_in = new int [trainRecord];
trainUdp_out = new int [trainRecord];
trainUdp_accpt = new int [trainRecord];
trainUdp_rejct = new int [trainRecord];
trainOth_in = new int [trainRecord];
trainOth_out = new int [trainRecord];
trainOth_accpt = new int [trainRecord];
trainOth_rejct = new int [trainRecord];
int trainCounter = 0;
for (int j = startTrainIndex;j<lastTrainIndex;j++)
{
    trainDateArray[trainCounter] = dateArray[j];
    trainTimeArray[trainCounter] = timeArray[j];
    trainInBoundArray[trainCounter] = inBoundArray[j];
    trainOutBboundArray[trainCounter] = outBboundArray[j];
    trainAcceptedArray[trainCounter] = acceptedArray[j];
    trainRejectedArray[trainCounter] = rejectedArray[j];
    trainTcp_in[trainCounter] = tcp_in[j];
    trainTcp_out[trainCounter] = tcp_out[j];
    trainTcp_accpt[trainCounter] = tcp_accpt[j];
    trainTcp_rejct[trainCounter] = tcp_rejct[j];
    trainIp_in[trainCounter] = ip_in[j];
    trainIp_out[trainCounter] = ip_out[j];
    trainIp_accpt[trainCounter] = ip_accpt[j];
    trainIp_rejct[trainCounter] = ip_rejct[j];
    trainIcmp_in[trainCounter] = icmp_in[j];
    trainIcmp_out[trainCounter] = icmp_out[j];
    trainIcmp_accpt[trainCounter] = icmp_accpt[j];
    trainIcmp_rejct[trainCounter] = icmp_rejct[j];
    trainUdp_in[trainCounter] = udp_in[j];
    trainUdp_out[trainCounter] = udp_out[j];
    trainUdp_accpt[trainCounter] = udp_accpt[j];
    trainUdp_rejct[trainCounter] = udp_rejct[j];
    trainOth_in[trainCounter] = oth_in[j];
    trainOth_out[trainCounter] = oth_out[j];
    trainOth_accpt[trainCounter] = oth_accpt[j];
    trainOth_rejct[trainCounter] = oth_rejct[j];
    trainCounter++;
}

testDateArray = new String [24];
testTimeArray = new String [24];
testInBoundArray = new int [24];
testOutBboundArray = new int [24];
testAcceptedArray = new int [24];
testRejectedArray = new int [24];
testTcp_in = new int [24];
testTcp_out = new int [24];
testTcp_accpt = new int [24];
testTcp_rejct = new int [24];
testIp_in = new int [24];
testIp_out = new int [24];
testIp_accpt = new int [24];
testIp_rejct = new int [24];
testIcmp_in = new int [24];
testIcmp_out = new int [24];

```

```

testIcmp_accpt = new int [24];
testIcmp_rejct = new int [24];
testUdp_in = new int [24];
testUdp_out = new int [24];
testUdp_accpt = new int [24];
testUdp_rejct = new int [24];
testOth_in = new int [24];
testOth_out = new int [24];
testOth_accpt = new int [24];
testOth_rejct = new int [24];
int testCounter = 0;
for (int k = lastTrainIndex;k<lastTrainIndex + 24;k++)
    {
        testDateArray[testCounter] = dateArray[k];
        testTimeArray[testCounter] = timeArray[k];
        testInBoundArray[testCounter] = inBoundArray[k];
        testOutBboundArray[testCounter] = outBboundArray[k];
        testAcceptedArray[testCounter] = acceptedArray[k];
        testRejectedArray[testCounter] = rejectedArray[k];
        testTcp_in[testCounter] = tcp_in[k];
        testTcp_out[testCounter] = tcp_out[k];
        testTcp_accpt[testCounter] = tcp_accpt[k];
        testTcp_rejct[testCounter] = tcp_rejct[k];
        testIp_in[testCounter] = ip_in[k];
        testIp_out[testCounter] = ip_out[k];
        testIp_accpt[testCounter] = ip_accpt[k];
        testIp_rejct[testCounter] = ip_rejct[k];
        testIcmp_in[testCounter] = icmp_in[k];
        testIcmp_out[testCounter] = icmp_out[k];
        testIcmp_accpt[testCounter] = icmp_accpt[k];
        testIcmp_rejct[testCounter] = icmp_rejct[k];
        testUdp_in[testCounter] = udp_in[k];
        testUdp_out[testCounter] = udp_out[k];
        testUdp_accpt[testCounter] = udp_accpt[k];
        testUdp_rejct[testCounter] = udp_rejct[k];
        testOth_in[testCounter] = oth_in[k];
        testOth_out[testCounter] = oth_out[k];
        testOth_accpt[testCounter] = oth_accpt[k];
        testOth_rejct[testCounter] = oth_rejct[k];
        testCounter++;
    }
trainingPeriodLength = trainRecord / 24;
ArrayList<String> t2dayStrArrayList = new ArrayList<String>();
t2dayStrArrayList.add("");
System.out.println("Training Period Length = " +
trainingPeriodLength);
for (int d = 1;d <= trainingPeriodLength / 7;d++)
    {
        if (d == 1)
        {
            String dayStr = (Integer.toString(d) + "-week");
            t2dayStrArrayList.add(dayStr);
        }
        else
        {
            String dayStr = (Integer.toString(d) + "-weeks");

            t2dayStrArrayList.add(dayStr);} }
trainingPeriodList = (String []) t2dayStrArrayList.toArray(
new String [0]);
t2selectTrainingPeriodCombobox.removeAllItems();
for (int i = 0;i<trainingPeriodList.length;i++)
    {
        t2selectTrainingPeriodCombobox.addItem(
trainingPeriodList[i]); }

```

```

        t2selectTrainingPeriodCombobox.addItem("Full History");
        t2selectTrainingPeriodCombobox.setSelectedIndex(
            trainingPeriodList.length);
    } // end try
    catch(NullPointerException eNull)
    { t2selectMethodCombobox.setSelectedItem(""); }
    } // end of item selection validation
    if (e.getStateChange() == ItemEvent.DESELECTED)
    {
        t2selectTrainingEpochsCombobox.setSelectedIndex(1);
        t2selectTrainingHiddenUnitsCombobox.setSelectedIndex(5);
        t2selectTrainingHILRCCombobox.setSelectedIndex(7);
        t2selectTrainingHOLRCCombobox.setSelectedIndex(7);
    } } } );

t2selectMonthCombobox.addItemListener(new ItemListener()
{
public void itemStateChanged(ItemEvent e)
    {
        if (e.getStateChange() == ItemEvent.DESELECTED)
        {
            t2selectMethodCombobox.setSelectedItem(""); } } } );

t2selectDayCombobox.addItemListener(new ItemListener()
{
public void itemStateChanged(ItemEvent e)
    {
        if (e.getStateChange() == ItemEvent.DESELECTED)
        {
            t2selectMethodCombobox.setSelectedItem(""); } } } );

JPanel t2yearSelectionPanel = new JPanel();
t2yearSelectionPanel.setLayout(new GridLayout(1,2));
t2yearSelectionPanel.setBackground(Color.yellow);
t2yearSelectionPanel.add(t2yearLabel);
t2yearSelectionPanel.add(t2selectYearCombobox);
JPanel t2monthSelectionPanel = new JPanel();
t2monthSelectionPanel.setLayout(new GridLayout(1,2));
t2monthSelectionPanel.setBackground(Color.yellow);
t2monthSelectionPanel.add(t2monthLabel);
t2monthSelectionPanel.add(t2selectMonthCombobox);
JPanel t2daySelectionPanel = new JPanel();
t2daySelectionPanel.setLayout(new GridLayout(1,2));
t2daySelectionPanel.setBackground(Color.yellow);
t2daySelectionPanel.add(t2dayLabel);
t2daySelectionPanel.add(t2selectDayCombobox);
final JPanel t2methodSelectionPanel = new JPanel();
t2methodSelectionPanel.setLayout(new GridLayout(1,2));
t2methodSelectionPanel.setBackground(Color.yellow);
t2methodSelectionPanel.add(t2methodLabel);
t2methodSelectionPanel.add(t2selectMethodCombobox);
final JTextArea t2chartedDataTextArea = new JTextArea();
t2chartedDataTextArea.setEditable(false);
t2chartedDataTextArea.setBackground(Color.white);
t2chartedDataTextArea.setForeground(Color.blue);
t2chartedDataTextArea.setBorder(new LineBorder(Color.white, 1));
t2chartedDataTextArea.setPreferredSize(new Dimension(200,85));
t2chartedDataTextArea.setFont(new Font("Courier", Font.BOLD, 12));
t2chartedDataTextArea.setLineWrap(true);

//construct the starting charting action button
JButton t2startSelectedButton = new JButton("Graph Selected Date");
t2startSelectedButton.setHorizontalAlignment(SwingConstants.CENTER);
t2startSelectedButton.setVerticalAlignment(SwingConstants.CENTER);
t2startSelectedButton.setForeground(new Color(0,151,82));
t2startSelectedButton.setFont(new Font("Arial", Font.BOLD, 12));

```

```

t2startSelectedButton.addActionListener(new ActionListener()
{
public void actionPerformed(ActionEvent eStart)
{
t2chartedDataTextArea.setText(null);
t2chartedDataTextArea.setBorder(new LineBorder(Color.white, 1));
t2graphTitle.removeAll();
t2ForecastingGraph.removeAll();
t2ResidualsGraph.removeAll();
t2graphPanel.removeAll();
MlpResiduals = new int [24];
MlpCompareResiduals = new int [24];
HWResiduals = new int [24];
RegResiduals = new int [24];
// check if user selection is valid for exploring data charts
if(((String)t2selectYearCombobox.getSelectedItem() != "") &&
((String)t2selectMonthCombobox.getSelectedItem() != "") &&
((String)t2selectDayCombobox.getSelectedItem() != "") &&
((String)t2selectMethodCombobox.getSelectedItem() != ""))
{
int startIndx = 0;
int endIndx =
trainingHistory((String)t2selectTrainingPeriodCombobox.getSelectedItem());
System.out.println("Start Training Index = " + startIndx);
System.out.println("End Training Index = " + endIndx);
System.out.println("Full History Length = " + trainRecord);
System.out.println(" ===== ");
String t2selYear = (String)t2selectYearCombobox.getSelectedItem();
String t2selMonth =
((String)t2selectMonthCombobox.getSelectedItem()).substring(0,3);
String t2selDay = (String)t2selectDayCombobox.getSelectedItem();
String t2selMethod = (String)t2selectMethodCombobox.getSelectedItem();
String t2selDate = t2selDay + t2selMonth + t2selYear;
DayOfTheWeek dotw = new DayOfTheWeek();
String t2selDateStringValue = dotw.DayOfTheWeek(Integer.parseInt(t2selDay),
t2selMonth, Integer.parseInt(t2selYear));
System.out.println("Curent Selected Day is: " + t2selDateStringValue + ", "
+ t2selDate);
wTempDateArray = new String [endIndx];
wTempTimeArray = new String [endIndx];
wTempInBoundArray = new int [endIndx];
wTempOutBboundArray = new int [endIndx];
wTempAcceptedArray = new int [endIndx];
wTempRejectedArray = new int [endIndx];
wTempTcp_in = new int [endIndx];
wTempTcp_out = new int [endIndx];
wTempTcp_accpt = new int [endIndx];
wTempTcp_rejct = new int [endIndx];
wTempIp_in = new int [endIndx];
wTempIp_out = new int [endIndx];
wTempIp_accpt = new int [endIndx];
wTempIp_rejct = new int [endIndx];
wTempIcmp_in = new int [endIndx];
wTempIcmp_out = new int [endIndx];
wTempIcmp_accpt = new int [endIndx];
wTempIcmp_rejct = new int [endIndx];
wTempUdp_in = new int [endIndx];
wTempUdp_out = new int [endIndx];
wTempUdp_accpt = new int [endIndx];
wTempUdp_rejct = new int [endIndx];
wTempOth_in = new int [endIndx];
wTempOth_out = new int [endIndx];
wTempOth_accpt = new int [endIndx];
wTempOth_rejct = new int [endIndx];
int trainWeeklyCounter = 0;
for (int j = 0;j<endIndx;j++)

```

```

{
String Date = trainDateArray[j];
int DayToken = 0;
String MonthToken = "";
int YearToken = 0;
if (Date.length() == 8)
{
DayToken = Integer.parseInt(Date.substring(0,1));
MonthToken = Date.substring(1,4);
YearToken = Integer.parseInt(Date.substring(4));
}
if (Date.length() == 9)
{
DayToken = Integer.parseInt(Date.substring(0,2));
MonthToken = Date.substring(2,5);
YearToken = Integer.parseInt(Date.substring(5));
}

String DateValue = dotw.DayOfTheWeek(DayToken, MonthToken,
YearToken);

System.out.println("Current d date string is ---> " + DateValue + " "
+ Date);

if(DateValue.equalsIgnoreCase(t2selDateStringValue))
{
wTempDateArray[trainWeeklyCounter] = trainDateArray[j];
wTempTimeArray[trainWeeklyCounter] = trainTimeArray[j];
wTempInBoundArray[trainWeeklyCounter] = trainInBoundArray[j];
wTempOutBboundArray[trainWeeklyCounter] = trainOutBboundArray[j];
wTempAcceptedArray[trainWeeklyCounter] = trainAcceptedArray[j];
wTempRejectedArray[trainWeeklyCounter] = trainRejectedArray[j];
wTempTcp_in[trainWeeklyCounter] = trainTcp_in[j];
wTempTcp_out[trainWeeklyCounter] = trainTcp_out[j];
wTempTcp_accpt[trainWeeklyCounter] = trainTcp_accpt[j];
wTempTcp_rejct[trainWeeklyCounter] = trainTcp_rejct[j];
wTempIp_in[trainWeeklyCounter] = trainIp_in[j];
wTempIp_out[trainWeeklyCounter] = trainIp_out[j];
wTempIp_accpt[trainWeeklyCounter] = trainIp_accpt[j];
wTempIp_rejct[trainWeeklyCounter] = trainIp_rejct[j];
wTempIcmp_in[trainWeeklyCounter] = trainIcmp_in[j];
wTempIcmp_out[trainWeeklyCounter] = trainIcmp_out[j];
wTempIcmp_accpt[trainWeeklyCounter] = trainIcmp_accpt[j];
wTempIcmp_rejct[trainWeeklyCounter] = trainIcmp_rejct[j];
wTempUdp_in[trainWeeklyCounter] = trainUdp_in[j];
wTempUdp_out[trainWeeklyCounter] = trainUdp_out[j];
wTempUdp_accpt[trainWeeklyCounter] = trainUdp_accpt[j];
wTempUdp_rejct[trainWeeklyCounter] = trainUdp_rejct[j];
wTempOth_in[trainWeeklyCounter] = trainOth_in[j];
wTempOth_out[trainWeeklyCounter] = trainOth_out[j];
wTempOth_accpt[trainWeeklyCounter] = trainOth_accpt[j];
wTempOth_rejct[trainWeeklyCounter] = trainOth_rejct[j];
trainWeeklyCounter++;
System.out.println("Training weekly counter for selected training
date " + DateValue + " " + Date + " is -->" + trainWeeklyCounter);
} }

// initializing training weekly arrays
trainWeeklyDateArray = new String [trainWeeklyCounter];
trainWeeklyTimeArray = new String [trainWeeklyCounter];
trainWeeklyInBoundArray = new int [trainWeeklyCounter];
trainWeeklyOutBboundArray = new int [trainWeeklyCounter];
trainWeeklyAcceptedArray = new int [trainWeeklyCounter];
trainWeeklyRejectedArray = new int [trainWeeklyCounter];
trainWeeklyTcp_in = new int [trainWeeklyCounter];
trainWeeklyTcp_out = new int [trainWeeklyCounter];

```

```

trainWeeklyTcp_accpt = new int [trainWeeklyCounter];
trainWeeklyTcp_rejct = new int [trainWeeklyCounter];
trainWeeklyIp_in = new int [trainWeeklyCounter];
trainWeeklyIp_out = new int [trainWeeklyCounter];
trainWeeklyIp_accpt = new int [trainWeeklyCounter];
trainWeeklyIp_rejct = new int [trainWeeklyCounter];
trainWeeklyIcmp_in = new int [trainWeeklyCounter];
trainWeeklyIcmp_out = new int [trainWeeklyCounter];
trainWeeklyIcmp_accpt = new int [trainWeeklyCounter];
trainWeeklyIcmp_rejct = new int [trainWeeklyCounter];
trainWeeklyUdp_in = new int [trainWeeklyCounter];
trainWeeklyUdp_out = new int [trainWeeklyCounter];
trainWeeklyUdp_accpt = new int [trainWeeklyCounter];
trainWeeklyUdp_rejct = new int [trainWeeklyCounter];
trainWeeklyOth_in = new int [trainWeeklyCounter];
trainWeeklyOth_out = new int [trainWeeklyCounter];
trainWeeklyOth_accpt = new int [trainWeeklyCounter];
trainWeeklyOth_rejct = new int [trainWeeklyCounter];

// populating training arrays with data
for(int j = 0;j<trainWeeklyCounter;j++)
{
    trainWeeklyDateArray[j] = wTempDateArray[j];
    trainWeeklyTimeArray[j] = wTempTimeArray[j];
    trainWeeklyInBoundArray[j] = wTempInBoundArray[j];
    trainWeeklyOutBboundArray[j] = wTempOutBboundArray[j];
    trainWeeklyAcceptedArray[j] = wTempAcceptedArray[j];
    trainWeeklyRejectedArray[j] = wTempRejectedArray[j];
    trainWeeklyTcp_in[j] = wTempTcp_in[j];
    trainWeeklyTcp_out[j] = wTempTcp_out[j];
    trainWeeklyTcp_accpt[j] = wTempTcp_accpt[j];
    trainWeeklyTcp_rejct[j] = wTempTcp_rejct[j];
    trainWeeklyIp_in[j] = wTempIp_in[j];
    trainWeeklyIp_out[j] = wTempIp_out[j];
    trainWeeklyIp_accpt[j] = wTempIp_accpt[j];
    trainWeeklyIp_rejct[j] = wTempIp_rejct[j];
    trainWeeklyIcmp_in[j] = wTempIcmp_in[j];
    trainWeeklyIcmp_out[j] = wTempIcmp_out[j];
    trainWeeklyIcmp_accpt[j] = wTempIcmp_accpt[j];
    trainWeeklyIcmp_rejct[j] = wTempIcmp_rejct[j];
    trainWeeklyUdp_in[j] = wTempUdp_in[j];
    trainWeeklyUdp_out[j] = wTempUdp_out[j];
    trainWeeklyUdp_accpt[j] = wTempUdp_accpt[j];
    trainWeeklyUdp_rejct[j] = wTempUdp_rejct[j];
    trainWeeklyOth_in[j] = wTempOth_in[j];
    trainWeeklyOth_out[j] = wTempOth_out[j];
    trainWeeklyOth_accpt[j] = wTempOth_accpt[j];
    trainWeeklyOth_rejct[j] = wTempOth_rejct[j];
}

System.out.println("Training weekly array length ---> " +
trainWeeklyDateArray.length);

String t2graphYear = t2selYear;
String t2graphMonth =
(String)t2selectMonthComboBox.getSelectedItem();
String t2graphDay = t2selDay;
JLabel t2ForecastingLabel = new JLabel("Summary of Forecasted
Analysis Results");
t2ForecastingLabel.setBackground(Color.white);
t2ForecastingLabel.setForeground(Color.red);
t2ForecastingLabel.setHorizontalAlignment(SwingConstants.CENTER);
t2ForecastingLabel.setVerticalAlignment(SwingConstants.CENTER);
t2ForecastingLabel.setOpaque(true);
t2ForecastingLabel.setFont(new Font("Arial", Font.BOLD, 16));
t2graphTitle.add("Center", t2ForecastingLabel);

```

```

// check to see if MLP is selected with MLP forecasting parameters
// then produce MLP forecasted data with RMSE error calculations
if(t2selMethod.equalsIgnoreCase("NN-MLP"))
{
String t2selPeriod =
(String)t2selectTrainingPeriodCombobox.getSelectedItemAt();
String t2selEpochs =
(String)t2selectTrainingEpochsCombobox.getSelectedItemAt();
String t2selHidden =
(String)t2selectTrainingHiddenUnitsCombobox.getSelectedItemAt();
String t2selHILR =
(String)t2selectTrainingHILRCombobox.getSelectedItemAt();
String t2selHOLR =
(String)t2selectTrainingHOLRCombobox.getSelectedItemAt();
System.out.println("Selected Date: " + t2selDate);
System.out.println("Selected Method: " + t2selMethod);
System.out.println("Selected Period: " + t2selPeriod);
System.out.println("Selected Epochs: " + t2selEpochs);
System.out.println("Selected Hidden: " + t2selHidden);
System.out.println("Selected HILR: " + t2selHILR);
System.out.println("Selected HOLR: " + t2selHOLR);
System.out.println(" ++++++ ");
int noOfEpochs = Integer.valueOf(t2selEpochs);
int noOfHidden = Integer.valueOf(t2selHidden);
double hiddenInputLR = Double.valueOf(t2selHILR);
double hiddenOutputLR = Double.valueOf(t2selHOLR);
t2ForecastingGraph.add("Center", new MLPGraph(startIndx, endIndx,
noOfEpochs, noOfHidden, hiddenInputLR, hiddenOutputLR,t2graphYear,
t2graphMonth, t2graphDay));
t2ResidualsGraph.add("Center", new ResidualsGraph(MlpResiduals,
t2graphYear, t2graphMonth, t2graphDay));
t2chartedDataTextArea.setBorder(new LineBorder(Color.red, 1));
t2chartedDataTextArea.setText(null);
t2chartedDataTextArea.append(" " + t2selMethod + " Method
Errors\n");
t2chartedDataTextArea.append(" -----\n");
t2chartedDataTextArea.append(" ME = " + (float)MlpME + "\n");
t2chartedDataTextArea.append(" MAD = " + (float)MlpMAD + "\n");
t2chartedDataTextArea.append(" RMSE = " + (float)MlpRMSE + "\n");
}

// check to see if Holt-Winter is selected & set forecasting
// parameters then produce forecasted data with RMSE error
// calculations

if(t2selMethod.equalsIgnoreCase("Holt-Winter"))
{
String t2selPeriod =
(String)t2selectTrainingPeriodCombobox.getSelectedItemAt();
System.out.println("Selected Date: " + t2selDate);
System.out.println("Selected Method: " + t2selMethod);
System.out.println("Selected Period: " + t2selPeriod);
System.out.println(" ++++++ ");
t2ForecastingGraph.add("Center", new HWGraph(startIndx, endIndx,
t2graphYear, t2graphMonth, t2graphDay));
t2ResidualsGraph.add("Center", new ResidualsGraph(HWRResiduals,
t2graphYear, t2graphMonth, t2graphDay));
t2chartedDataTextArea.setBorder(new LineBorder(Color.red, 1));
t2chartedDataTextArea.setText(null);
t2chartedDataTextArea.append(" " + t2selMethod + " Method
Errors\n");
t2chartedDataTextArea.append(" -----\n");
t2chartedDataTextArea.append(" ME = " + (float)HWME + "\n");
t2chartedDataTextArea.append(" MAD = " + (float)HWMAD + "\n");
t2chartedDataTextArea.append(" RMSE = " + (float)HWRMSE + "\n");
}

```

```

// check to see if Regression is selected & set forecasting
// parameters then produce forecasted data with RMSE error
// calculations
if(t2selMethod.equalsIgnoreCase("Regression"))
{
String t2selPeriod =
(String)t2selectTrainingPeriodCombobox.getSelectedItemAt();
System.out.println("Selected Date:   " + t2selDate);
System.out.println("Selected Method: " + t2selMethod);
System.out.println("Selected Period: " + t2selPeriod);
System.out.println(" ++++++ ");
t2ForecastingGraph.add("Center", new RegGraph(startIndx, endIndx,
t2graphYear, t2graphMonth, t2graphDay));
t2ResidualsGraph.add("Center", new ResidualsGraph(RegResiduals,
t2graphYear, t2graphMonth, t2graphDay));
t2chartedDataTextArea.setBorder(new LineBorder(Color.red, 1));
t2chartedDataTextArea.setText(null);
t2chartedDataTextArea.append("   " + t2selMethod + " Method
Errors\n");
t2chartedDataTextArea.append(" ----- \n");
t2chartedDataTextArea.append("  ME   =   " + (float)RegME + "\n");
t2chartedDataTextArea.append("  MAD  =   " + (float)RegMAD + "\n");
t2chartedDataTextArea.append("  RMSE =   " + (float)RegRMSE + "\n");
}

// check to see if All methods had been selected (for comparison)
// and set forecasting parameters then produce forecasted data with
// RMSE error calculations
if(t2selMethod.equalsIgnoreCase("Compare All"))
{
String t2selPeriod =
(String)t2selectTrainingPeriodCombobox.getSelectedItemAt();
String t2selEpochs =
(String)t2selectTrainingEpochsCombobox.getSelectedItemAt();
String t2selHidden =
(String)t2selectTrainingHiddenUnitsCombobox.getSelectedItemAt();
String t2selHILR =
(String)t2selectTrainingHILRCombobox.getSelectedItemAt();
String t2selHOLR =
(String)t2selectTrainingHOLRCombobox.getSelectedItemAt();
int noOfEpochs = Integer.valueOf(t2selEpochs);
int noOfHidden = Integer.valueOf(t2selHidden);
double hiddenInputLR = Double.valueOf(t2selHILR);
double hiddenOutputLR = Double.valueOf(t2selHOLR);
System.out.println("Selected Date:   " + t2selDate);
System.out.println("Selected Method: " + t2selMethod);
System.out.println("Selected Period: " + t2selPeriod);
System.out.println("Selected Epochs: " + t2selEpochs);
System.out.println("Selected Hidden: " + t2selHidden);
System.out.println("Selected HILR:   " + t2selHILR);
System.out.println("Selected HOLR:   " + t2selHOLR);
System.out.println(" ++++++ ");
t2ForecastingGraph.add("Center", new CompareGraph(startIndx, endIndx,
noOfEpochs, noOfHidden, hiddenInputLR, hiddenOutputLR, t2graphYear,
t2graphMonth, t2graphDay));
t2ResidualsGraph.add("Center", new AnalysisTable(MlpME, HWME, RegME,
MlpMAD, HWMAD, RegMAD, MlpRMSE, HWRMSE, RegRMSE, t2graphYear,
t2graphMonth, t2graphDay)); }

// generate a forecasting report to file with RMSE error calculations
if(t2selMethod.equalsIgnoreCase("Report"))
{
try
{
Date currentDate = new Date ();
SimpleDateFormat dateFormat = new SimpleDateFormat("yyyyMMdd");

```

```

StringBuilder reportDate = new StringBuilder( dateFormat.format(
currentDate ) );
int strLn = t2selDate.length();
String repYear = "";
String repMonth = "";
if (strLn == 8)
{
repYear = t2selDate.substring(4);
repMonth = t2selDate.substring(1,4);
}
else {
repYear = t2selDate.substring(5);
repMonth = t2selDate.substring(2,5);
}
System.out.println("String length for year: " + strLn );
System.out.println("Selected year: " + repYear );
System.out.println("Selected month: " + repMonth );
String reportFileName = (". /Reports// " + repYear + "// " +
repMonth + " // " + t2selDate + "_Day_Report_" +
reportDate.toString() + ".csv");
BufferedWriter reporter = new BufferedWriter(new F
ileWriter(reportFileName));
int weeksRef = (trainRecord / 7) / 24;
int noOfEpochs = 100;
int noOfHidden = 15;
double hiddenInputLR = 0.7;
double hiddenOutputLR = 0.07;
int [] mlpData = new int [24];
int [] hwfData = new int [24];
int [] regData = new int [24];
System.out.println("Selected Date: " + t2selDate);
System.out.println("Total Training Records ---> " + trainRecord);
System.out.println(" ===== ");
System.out.println("Training History (Weeks)\t" + "MLP_RMSE\t\t" +
"HW_RMSE\t\t" + "Regression_RMSE");
reporter.write("Training History
(Weeks),MLP_RMSE,HW_RMSE,Regression_RMSE" + "\n");
System.out.println(" ===== ");
for(int i = 1;i <= weeksRef;i++)
{

// calling mlp
MLP myMlp = new MLP((i*7*24), noOfEpochs, noOfHidden,
hiddenInputLR, hiddenOutputLR);
mlpData = myMlp.returnForecastedResults();

//calling holt-winter
double [] HWTrainingData = new double [i*7*24];
double [] HWTestData = new double [24];
for (int k = 0;k<(i*7*24);k++)
{
HWTrainingData[k] = (double)trainRejectedArray[k];
}
for (int l = 0;l<24;l++)
{
HWTestData[l] = (double)testRejectedArray[l];
}
HWF myHwf = new HWF(HWTrainingData,HWTestData,(i*7*24));
hwfData = myHwf.returnForecastedResults();

// calling regression with dummy variables D8 to D18
REG myREG = new REG(0, (i*7*24));
for (int m = 0; m < 24; m++)
{

```

```

int D8 = 0;
int D9 = 0;
int D10 = 0;
int D11 = 0;
int D12 = 0;
int D13 = 0;
int D14 = 0;
int D15 = 0;
int D16 = 0;
int D17 = 0;
int D18 = 0;

if(testTimeArray[m].equals("8:00")) { D8 = 1; }
if(testTimeArray[m].equals("9:00")) { D9 = 1; }
if(testTimeArray[m].equals("10:00")) { D10 = 1; }
if(testTimeArray[m].equals("11:00")) { D11 = 1; }
if(testTimeArray[m].equals("12:00")) { D12 = 1; }
if(testTimeArray[m].equals("13:00")) { D13 = 1; }
if(testTimeArray[m].equals("14:00")) { D14 = 1; }
if(testTimeArray[m].equals("15:00")) { D15 = 1; }
if(testTimeArray[m].equals("16:00")) { D16 = 1; }
if(testTimeArray[m].equals("17:00")) { D17 = 1; }
if(testTimeArray[m].equals("18:00")) { D18 = 1; }

regData[m] = (int)(cData[0] +
(cData[1]*(DoSTDM.testTcp_in[m])) +
(cData[2]*(DoSTDM.testIcmp_in[m])) +
(cData[3]*(DoSTDM.testUdp_in[m])) +
(cData[4]*(DoSTDM.testOth_in[m])) +
(cData[5]*D8) + (cData[6]*D9) + (cData[7]*D10) +
(cData[8]*D11) + (cData[9]*D12) + (cData[10]*D13) +
(cData[11]*D14) + cData[12]*D15) + (cData[13]*D16) +
(cData[14]*D17) + (cData[15]*D18));
}

// weeks history counter
int weeksHistoryCounter = i;

// mlp error calculations parameters
double meMlpRepSum = 0.0;
double madMlpRepSum = 0.0;
double mseMlpRepSum = 0.0;

// HW error calculations parameters
double meHwRepSum = 0.0;
double madHwRepSum = 0.0;
double mseHwRepSum = 0.0;

// regression error calculations parameters
double meRegRepSum = 0.0;
double madRegRepSum = 0.0;
double mseRegRepSum = 0.0;

// final values for MLP, HW & Regression errors
MlpME = 0.0;
MlpMAD = 0.0;
MlpRMSE = 0.0;
HWME = 0.0;
HWMAD = 0.0;
HWRMSE = 0.0;
RegME = 0.0;
RegMAD = 0.0;
RegRMSE = 0.0;
calcMeError = new double[24];
calcMadError = new double[24];
calcMseError = new double[24];

```

```

for (int x = 0; x < 24; x + + )
{
    calcMeError[x] = testRejectedArray[x] - hwfData[x];
    calcMadError[x] = java.lang.Math.abs(calcMeError[x]);
    calcMseError[x] = calcMadError[x] * calcMadError[x];
    meHwRepSum = meHwRepSum + calcMeError[x];
    madHwRepSum = madHwRepSum + calcMadError[x];
    mseHwRepSum = mseHwRepSum + calcMseError[x];
}

HWME = meHwRepSum / 24;
HWMAD = madHwRepSum / 24;
HWRMSE = java.lang.Math.sqrt(mseHwRepSum / 24);
for (int x = 0; x < 24; x++)
{
    calcMeError[x] = testRejectedArray[x] - mlpData[x];
    calcMadError[x] = java.lang.Math.abs(calcMeError[x]);
    calcMseError[x] = calcMadError[x] * calcMadError[x];
    meMlpRepSum = meMlpRepSum + calcMeError[x];
    madMlpRepSum = madMlpRepSum + calcMadError[x];
    mseMlpRepSum = mseMlpRepSum + calcMseError[x];
}
MlpME = meMlpRepSum / 24;
MlpMAD = madMlpRepSum / 24;
MlpRMSE = java.lang.Math.sqrt(mseMlpRepSum / 24);
for (int x = 0; x < 24; x++)
{
    calcMeError[x] = testRejectedArray[x]-regData[x];
    calcMadError[x] = java.lang.Math.abs(calcMeError[x]);
    calcMseError[x] = calcMadError[x] * calcMadError[x];
    meRegRepSum = meRegRepSum + calcMeError[x];
    madRegRepSum = madRegRepSum + calcMadError[x];
    mseRegRepSum = mseRegRepSum + calcMseError[x];
}

RegME = meRegRepSum / 24;
RegMAD = madRegRepSum / 24;
RegRMSE = java.lang.Math.sqrt(mseRegRepSum / 24);
System.out.println(weeksHistoryCounter + "\t\t" + MlpRMSE +
"\t" + HWRMSE + "\t" + RegRMSE);
reporter.write(weeksHistoryCounter + "," + MlpRMSE + "," +
HWRMSE + "," + RegRMSE + "\n");
} // end of for loop using weeks breakdown

// calling mlp with full history
MLP myMlp = new MLP(trainRecord, noOfEpochs, noOfHidden,
hiddenInputLR, hiddenOutputLR);
mlpData = myMlp.returnForecastedResults();

// calling holt-winter with full history
double [] HWTrainingData = new double [trainRecord];
double [] HWTestData = new double [24];
for (int k = 0;k<trainRecord;k++)
{
    HWTrainingData[k] = (double)trainRejectedArray[k];
}
for (int l = 0;l<24;l++)
{
    HWTestData[l] = (double)testRejectedArray[l];
}
HWF myHwf = new HWF(HWTrainingData,HWTestData,trainRecord);
hwfData = myHwf.returnForecastedResults();

// calling regression with full history
REG myREG = new REG(0,trainRecord);

```

```

for (int m = 0; m < 24; m++)
{
    int D8 = 0;
    int D9 = 0;
    int D10 = 0;
    int D11 = 0;
    int D12 = 0;
    int D13 = 0;
    int D14 = 0;
    int D15 = 0;
    int D16 = 0;
    int D17 = 0;
    int D18 = 0;
    if(testTimeArray[m].equals("8:00")) { D8 = 1; }
    if(testTimeArray[m].equals("9:00")) { D9 = 1; }
    if(testTimeArray[m].equals("10:00")) { D10 = 1; }
    if(testTimeArray[m].equals("11:00")) { D11 = 1; }
    if(testTimeArray[m].equals("12:00")) { D12 = 1; }
    if(testTimeArray[m].equals("13:00")) { D13 = 1; }
    if(testTimeArray[m].equals("14:00")) { D14 = 1; }
    if(testTimeArray[m].equals("15:00")) { D15 = 1; }
    if(testTimeArray[m].equals("16:00")) { D16 = 1; }
    if(testTimeArray[m].equals("17:00")) { D17 = 1; }
    if(testTimeArray[m].equals("18:00")) { D18 = 1; }
    regData[m] = (int)(cData[0] + cData[1]*(DoSTDM.testTcp_in[m]))
    + (cData[2]*(DoSTDM.testIcmp_in[m])) +
    (cData[3]*(DoSTDM.testUdp_in[m])) +
    (cData[4]*(DoSTDM.testOth_in[m])) + (cData[5]*D8) +
    (cData[6]*D9) + (cData[7]*D10) + (cData[8]*D11) +
    (cData[9]*D12) + (cData[10]*D13) + (cData[11]*D14) +
    (cData[12]*D15) + (cData[13]*D16) + (cData[14]*D17) +
    (cData[15]*D18));
}

// mlp error calculations parameters
double meMlpRepSum = 0.0;
double madMlpRepSum = 0.0;
double mseMlpRepSum = 0.0;

// HW error calculations parameters
double meHwRepSum = 0.0;
double madHwRepSum = 0.0;
double mseHwRepSum = 0.0;

// regression error calculations parameters
double meRegRepSum = 0.0;
double madRegRepSum = 0.0;
double mseRegRepSum = 0.0;

// final values for MLP, HW & Regression errors
MlpME = 0.0;
MlpMAD = 0.0;
MlpRMSE = 0.0;
HWME = 0.0;
HWMAD = 0.0;
HWRMSE = 0.0;
RegME = 0.0;
RegMAD = 0.0;
RegRMSE = 0.0;
calcMeError = new double[24];
calcMadError = new double[24];

calcMseError = new double[24];

for (int x = 0; x < 24; x++)
{

```

```

        calcMeError[x] = testRejectedArray[x] - hwfData[x];
        calcMadError[x] = java.lang.Math.abs(calcMeError[x]);
        calcMseError[x] = calcMadError[x] * calcMadError[x];
        meHwRepSum = meHwRepSum + calcMeError[x];
        madHwRepSum = madHwRepSum + calcMadError[x];
        mseHwRepSum = mseHwRepSum + calcMseError[x];
    }
    HWME = meHwRepSum / 24;
    HWMAD = madHwRepSum / 24;
    HWRMSE = java.lang.Math.sqrt(mseHwRepSum / 24);

    for (int x = 0; x < 24; x++)
    {
        calcMeError[x] = testRejectedArray[x] - mlpData[x];
        calcMadError[x] = java.lang.Math.abs(calcMeError[x]);
        calcMseError[x] = calcMadError[x] * calcMadError[x];
        meMlpRepSum = meMlpRepSum + calcMeError[x];
        madMlpRepSum = madMlpRepSum + calcMadError[x];
        mseMlpRepSum = mseMlpRepSum + calcMseError[x];
    }
    MlpME = meMlpRepSum / 24;
    MlpMAD = madMlpRepSum / 24;
    MlpRMSE = java.lang.Math.sqrt(mseMlpRepSum / 24);

    for (int x = 0; x < 24; x++)
    {
        calcMeError[x] = testRejectedArray[x] - regData[x];
        calcMadError[x] = java.lang.Math.abs(calcMeError[x]);
        calcMseError[x] = calcMadError[x] * calcMadError[x];
        meRegRepSum = meRegRepSum + calcMeError[x];
        madRegRepSum = madRegRepSum + calcMadError[x];
        mseRegRepSum = mseRegRepSum + calcMseError[x];
    }
    RegME = meRegRepSum / 24;
    RegMAD = madRegRepSum / 24;
    RegRMSE = java.lang.Math.sqrt(mseRegRepSum / 24);

    System.out.println("Full History\t\t" + MlpRMSE + "\t" + HWRMSE +
"\t" + RegRMSE);
    System.out.println(" ===== ");
    reporter.write("Full History," + MlpRMSE + "," + HWRMSE + "," +
RegRMSE + "\n");
    reporter.close();
} // end of try statement
catch (IOException e)
{ System.out.println("Error writing to the report file"); }
} // end of option report

// compare MLP forecasted data
if(t2selMethod.equalsIgnoreCase("Compare MLP"))
{
String t2selPeriod =
(String)t2selectTrainingPeriodCombobox.getSelectedItemAt();
String t2selEpochs =
(String)t2selectTrainingEpochsCombobox.getSelectedItemAt();
String t2selHidden =
(String)t2selectTrainingHiddenUnitsCombobox.getSelectedItemAt();
String t2selHILR =
(String)t2selectTrainingHILRCombobox.getSelectedItemAt();
String t2selHOLR =
(String)t2selectTrainingHOLRCombobox.getSelectedItemAt();
System.out.println("Selected Date: " + t2selDate);
System.out.println("Selected Method: " + t2selMethod);
System.out.println("Selected Period: " + t2selPeriod);
System.out.println("Selected Epochs: " + t2selEpochs);
System.out.println("Selected Hidden: " + t2selHidden);
}

```

```

System.out.println("Selected HILR:  " + t2selHILR);
System.out.println("Selected HOLR:  " + t2selHOLR);
System.out.println(" ++++++ ");
int noOfEpochs = Integer.valueOf(t2selEpochs);
int noOfHidden = Integer.valueOf(t2selHidden);
double hiddenInputLR = Double.valueOf(t2selHILR);
double hiddenOutputLR = Double.valueOf(t2selHOLR);
int MLPStartIndx = startIndx;
int MLPEndIndx = endIndx;
int MLPWStartIndx = 0;
int MLPWEndIndx = trainWeeklyCounter;
t2ForecastingGraph.add("Center", new CompareMLPGraph(MLPStartIndx,
MLPEndIndx,MLPWStartIndx, MLPWEndIndx, noOfEpochs, noOfHidden,
hiddenInputLR, hiddenOutputLR,t2graphYear, t2graphMonth,
t2graphDay));
t2ResidualsGraph.add("Center", new MLPAnalysisTable(MlpME, MlpWME,
MlpMAD, MlpWMAD, MlpRMSE, MlpWRMSE, t2graphYear, t2graphMonth,
t2graphDay));
}
t2graphPanel.setLayout(new GridLayout(2,1));
t2graphPanel.setPreferredSize(new Dimension(675,700));
t2graphPanel.setBorder(new LineBorder(Color.white, 1));
t2graphPanel.add(t2ForecastingGraph);
t2graphPanel.add(t2ResidualsGraph);
t2graphics.add("Center",t2graphTitle);
t2graphics.add("South",t2graphPanel);
t2chartScrolPanel.setViewportView(t2graphics);
t2graphics.revalidate();
t2graphics.repaint();
}
else
{
if(((String)t2selectYearCombobox.getSelectedItem() == "") ||
((String)t2selectMonthCombobox.getSelectedItem() == "") ||
((String)t2selectDayCombobox.getSelectedItem() == "") ||
((String)t2selectMethodCombobox.getSelectedItem() == ""))
{
// if user selection is invalid, send error message and load
// guideline steps image to show user how to select to chart
t2graphPanel.removeAll();
t2graphPanel.setLayout(new BorderLayout());
t2graphPanel.setPreferredSize(new Dimension(675,445));
t2graphPanel.setBorder(new LineBorder(Color.white, 1));
t2graphPanel.add("Center",new JLabel(createImageIcon("images /
forecasting_guide.gif")));
t2chartScrolPanel.setViewportView(t2graphPanel);
t2graphPanel.revalidate();
t2graphPanel.repaint();
JOptionPane.showMessageDialog(null, "<html><center> Your data
selection parameters are incomplete.<br>Please reselect Year,
Month, Day & Method of Analysis.< / center>< / html>",
"Missing Parameters", JOptionPane.INFORMATION_MESSAGE);
}
}} // end of inner class
} );

// construct the clearing selection action button
JButton t2clearSelectedButton = new JButton("Clear Date Selection");
t2clearSelectedButton.setHorizontalAlignment(SwingConstants.CENTER);
t2clearSelectedButton.setVerticalAlignment(SwingConstants.CENTER);
t2clearSelectedButton.setForeground(Color.RED);
t2clearSelectedButton.setFont(new Font("Arial", Font.BOLD, 12));

t2clearSelectedButton.addActionListener(new ActionListener()
{

```

```

// inner class to detect the action command to reset the applet
public void actionPerformed(ActionEvent eClear)
{
    t2selectYearCombobox.setSelectedItem("");
    t2selectMonthCombobox.setSelectedItem("");
    t2selectDayCombobox.setSelectedItem("");
    t2selectMethodCombobox.setSelectedItem("");
    t2selectTrainingPeriodCombobox.setSelectedItem("");
    t2selectTrainingEpochsCombobox.setSelectedItem("");
    t2selectTrainingHiddenUnitsCombobox.setSelectedItem("");
    t2selectTrainingHILRCombobox.setSelectedItem("");
    t2selectTrainingHOLRCombobox.setSelectedItem("");
    t2selectTrainingPeriodCombobox.setSelectedItem("");
    t2trainingPeriodSelectionPanel.setVisible( false );
    t2trainingEpochsSelectionPanel.setVisible( false );
    t2trainingHiddenUnitsSelectionPanel.setVisible( false );
    t2trainingHILRSelectionPanel.setVisible( false );
    t2trainingHOLRSelectionPanel.setVisible( false );
    t2chartedDataTextArea.setText( null );
    t2chartedDataTextArea.setBorder( new LineBorder( Color.white, 1 ) );
    t2graphPanel.removeAll();
    t2graphPanel.setLayout( new BorderLayout() );
    t2graphPanel.setPreferredSize( new Dimension( 675, 445 ) );
    t2graphPanel.setBorder( new LineBorder( Color.white, 1 ) );
    t2graphPanel.add( "Center", new JLabel( createImageIcon( "images /
forecasting_guide.gif" ) ) );
    t2chartScrolPanel.setViewportView( t2graphPanel );
    t2graphPanel.revalidate();
    t2graphPanel.repaint();
} //
end of inner class
} );

final JPanel t2controllPanel = new JPanel();
t2controllPanel.setLayout( new GridLayout( 19, 1 ) );
t2controllPanel.setBackground( Color.white );
t2controllPanel.setPreferredSize( new Dimension( 200, 310 ) );
t2controllPanel.add( new JLabel( " " ) );
t2controllPanel.add( t2yearSelectionPanel );
t2controllPanel.add( new JLabel( " " ) );
t2controllPanel.add( t2monthSelectionPanel );
t2controllPanel.add( new JLabel( " " ) );
t2controllPanel.add( t2daySelectionPanel );
t2controllPanel.add( new JLabel( " " ) );
t2controllPanel.add( t2methodSelectionPanel );
t2controllPanel.add( new JLabel( " " ) );
t2controllPanel.add( t2trainingPeriodSelectionPanel );
t2controllPanel.add( new JLabel( " " ) );
t2controllPanel.add( t2trainingEpochsSelectionPanel );
t2controllPanel.add( new JLabel( " " ) );
t2controllPanel.add( t2trainingHiddenUnitsSelectionPanel );
t2controllPanel.add( new JLabel( " " ) );
t2controllPanel.add( t2trainingHILRSelectionPanel );
t2controllPanel.add( new JLabel( " " ) );
t2controllPanel.add( t2trainingHOLRSelectionPanel );
t2controllPanel.add( new JLabel( " " ) );
JPanel t2startPanel = new JPanel();
t2startPanel.setLayout( new BorderLayout() );
t2startPanel.setBackground( Color.white );
t2startPanel.setPreferredSize( new Dimension( 200, 35 ) );
t2startPanel.add( new JLabel( " " ), BorderLayout.SOUTH );
t2startPanel.add( new JLabel( " " ), BorderLayout.EAST );
t2startPanel.add( new JLabel( " " ), BorderLayout.WEST );
t2startPanel.add( t2startSelectedButton, BorderLayout.CENTER );
JPanel t2clearPanel = new JPanel();
t2clearPanel.setLayout( new BorderLayout() );

```

```

t2clearPanel.setBackground(Color.white);
t2clearPanel.setPreferredSize(new Dimension(200,35));
t2clearPanel.add(new JLabel("          "), BorderLayout.SOUTH);
t2clearPanel.add(new JLabel("          "), BorderLayout.EAST);
t2clearPanel.add(new JLabel("          "), BorderLayout.WEST);
t2clearPanel.add(t2clearSelectedButton, BorderLayout.CENTER);
JPanel t2control2Panel = new JPanel();
t2control2Panel.setLayout(new BorderLayout());
t2control2Panel.setBackground(Color.white);
t2control2Panel.setPreferredSize(new Dimension(200,70));
t2control2Panel.add("Center",new JLabel("          "));
t2control2Panel.add("North",t2startPanel);
t2control2Panel.add("South",t2clearPanel);
JPanel t2controlSelectionPanel = new JPanel();
t2controlSelectionPanel.setLayout(new BorderLayout());
t2controlSelectionPanel.setBackground(Color.white);
t2controlSelectionPanel.setBorder(new LineBorder(Color.black, 1));
t2controlSelectionPanel.setPreferredSize(new Dimension(200,470));
t2controlSelectionPanel.add("North",t2control1Panel);
t2controlSelectionPanel.add("Center",t2control2Panel);
t2controlSelectionPanel.add("South", t2chartedDataTextArea);
JPanel t2finalPanel = new JPanel();
t2finalPanel.setLayout(new BorderLayout());
t2finalPanel.setBackground(Color.black);
t2finalPanel.setBorder(new LineBorder(Color.black, 1));
t2finalPanel.setPreferredSize(new Dimension(900,470));
t2finalPanel.add("West", t2controlSelectionPanel);
t2finalPanel.add("Center", t2chartScrolPanel);
return t2finalPanel;
} // end of method initForecastingTab()

protected static ImageIcon createImageIcon(String path)
{
// construct a URL to the given image file string path from the root of the
// applet code directory
java.net.URL imgURL = DoSTDM.class.getResource(path);

// check if the URL is not null
If (imgURL != null)
{
// define the image icon to be returned as the file been passed with the
//constructed URL
returnedImage = new ImageIcon(imgURL);
}

// if the file path or name does not exist
else
{
//show the message for the supplied path for main page images or any
//other required image
JOptionPane.showMessageDialog(null, "<html><center>Error Loading " +
(path.substring(7)) + " Image<p>Please Click OK to Continue Loading<
/ center>< / html>", "Image Loading Exception",
JOptionPane.INFORMATION_MESSAGE);
}

// return the value of the requested image or if image is not available this
// would be the error image
return returnedImage;

} // end of method createImageIcon()

public static void dataCollector()
{
// create a file object using the input file name

```

```

java.net.URL dataColletcorPath = DoSTDM.class.getResource("data /
Database.CSV");

try
{
// create a buffered reader and attach it the file reader
BufferedReader dataInputStream = new BufferedReader( new InputStreamReader
(dataColletcorPath.openStream()));

// read file lines
String dataInputLine = dataInputStream.readLine();

// initialize data variables
int i = 0;
dateArray = new String [record];
timeArray = new String [record];
inBoundArray = new int [record];
outBboundArray = new int [record];
acceptedArray = new int [record];
rejectedArray = new int [record];
tcp_in = new int [record];
tcp_out = new int [record];
tcp_accpt = new int [record];
tcp_rejct = new int [record];
ip_in = new int [record];
ip_out = new int [record];
ip_accpt = new int [record];
ip_rejct = new int [record];
icmp_in = new int [record];
icmp_out = new int [record];
icmp_accpt = new int [record];
icmp_rejct = new int [record];
udp_in = new int [record];
udp_out = new int [record];
udp_accpt = new int [record];
udp_rejct = new int [record];
oth_in = new int [record];
oth_out = new int [record];
oth_accpt = new int [record];
oth_rejct = new int [record];

    try
    {
        while (dataInputLine != null)
        {
            dataInputLine = dataInputStream.readLine();
            StringTokenizer st = new StringTokenizer(dataInputLine, ",");
            dateArray[i] = st.nextToken();
            timeArray[i] = st.nextToken();
            inBoundArray[i] = Integer.valueOf(st.nextToken());
            tcp_in[i] = Integer.valueOf(st.nextToken());
            ip_in[i] = Integer.valueOf(st.nextToken());
            icmp_in[i] = Integer.valueOf(st.nextToken());
            udp_in[i] = Integer.valueOf(st.nextToken());
            oth_in[i] = Integer.valueOf(st.nextToken());
            outBboundArray[i] = Integer.valueOf(st.nextToken());
            tcp_out[i] = Integer.valueOf(st.nextToken());
            ip_out[i] = Integer.valueOf(st.nextToken());
            icmp_out[i] = Integer.valueOf(st.nextToken());
            udp_out[i] = Integer.valueOf(st.nextToken());
            oth_out[i] = Integer.valueOf(st.nextToken());
            acceptedArray[i] = Integer.valueOf(st.nextToken());
            tcp_accpt[i] = Integer.valueOf(st.nextToken());
            ip_accpt[i] = Integer.valueOf(st.nextToken());
            icmp_accpt[i] = Integer.valueOf(st.nextToken());
            udp_accpt[i] = Integer.valueOf(st.nextToken());
        }
    }
}

```

```

        oth_accpt[i] = Integer.valueOf(st.nextToken());
        rejectedArray[i] = Integer.valueOf(st.nextToken());
        tcp_rejct[i] = Integer.valueOf(st.nextToken());
        ip_rejct[i] = Integer.valueOf(st.nextToken());
        icmp_rejct[i] = Integer.valueOf(st.nextToken());
        udp_rejct[i] = Integer.valueOf(st.nextToken());
        oth_rejct[i] = Integer.valueOf(st.nextToken());
        i++;
    } // end of while loop

    dataInputStream.close(); // close the input stream
} // close inner try
catch(NullPointerException eNull) { }
} // close outer try statment
catch(IOException eCollectValues) { }

} // end of method dataCollector()

public static int trainingHistory(String trainPeriodString)
{
int startPointIndx = 0;
int parsedNo = 0;
String parsedStr = "";
if(trainPeriodString.equalsIgnoreCase("Full History"))
{ startPointIndx = trainRecord; }
else
{
StringTokenizer t2st = new StringTokenizer(trainPeriodString, "-");
parsedNo = Integer.valueOf(t2st.nextToken());
parsedStr = t2st.nextToken();

if(parsedStr.equalsIgnoreCase("day") ||
parsedStr.equalsIgnoreCase("days"))
{ startPointIndx = parsedNo * 24; }

if(parsedStr.equalsIgnoreCase("week") ||
parsedStr.equalsIgnoreCase("weeks"))
{ startPointIndx = parsedNo * 24 * 7; }

if(parsedStr.equalsIgnoreCase("month") ||
parsedStr.equalsIgnoreCase("months"))
{ startPointIndx = parsedNo * 24 * 30; }
}

return startPointIndx;
} // end of method trainingHistory

} // end of class DoSTDM

```

Listing A – 14: DoSTDM Java Class (DoSTDM.java)

### A.2.2.3 Holt-Winter Forecasting Engine Java Code

```
import java.io.*;
import java.util.*;
import java.lang.Math;

public class HWF
{
//HWF defineable variables
public static int numTrainingPatterns;
public static int arrayIndex;

//Trend process variables
public static double aSum=0.0;           // a coff (sum) component
public static double bSum=0.0;           // b coff (sum) component
public static double tSqrSum=0.0; // time coff (sum) component
public static double sSum=0.0;           // Seasonal indexes (sum) component
public static double a=0.0;              // Trend coff (a)
public static double b=0.0;              // Trend coff (b)
public static double ME=0.0;              // ME error rate
public static double MAD=0.0;            // MAD error rate
public static double MSE=0.0;            // MSE error rate
public static double RMSE=0.0;           // RMSE error rate
public static double meSum=0.0;          // ME error (sum) component
public static double madSum=0.0;         // MAD error (sum) component
public static double mseSum=0.0;         // MSE error (sum) component
public static double alpha=0;            // HW forecasting alpha factor
public static double beta=0;             // HW forecasting delta factor
public static double gamma;              // HW forecasting gamma factor
public static double hwA=0.0; // HW forecasting trend a level factor
public static double hwB=0.0; // HW forecasting trend b trend factor
public static double hwI=0.0; // HW forecasting trend i seasonality factor
public static double [] timePeriod;
public static double [] timeSqured;
public static double [] realRejct;       //Training Reject array
public static double [] calcTrend;
public static double [] multiSeasIndex;
public static double [] seasIndexAverage;
public static double [] calcRejct;       //Calculated Reject array
public static double [] calcMeError;     //Calculated Reject array ME error
public static double [] calcMadError;    //Calculated Reject array MAD error
public static double [] calcMseError;    //Calculated Reject array MSE error
public static double [] realTestRejct;   //Forecasted Reject array
public static double [] forecastTestRejct; //Forecasted Reject array
public static double [] calcForecastMeError; //Calculated Reject ME error
public static double [] calcForecastMadError; //Calculated Reject MAD error
public static double [] calcForecastMseError; //Calculated Reject MSE error
public static double forecastME;        // ME error rate
public static double forecastMAD;       // MAD error rate
public static double forecastMSE;       // MSE error rate
public static double forecastRMSE;      // RMSE error rate
public static double forecastMeSum;     // ME error (sum) component
public static double forecastMadSum;    // MAD error (sum) component
public static double forecastMseSum;    // MSE error (sum) component

//=====
//***** THIS IS THE MAIN CONSTRUCTOR *****
//=====

public HWF(double [] passedTrainingArray, double [] passedTestArray, int
noOfTrainPat)
{
numTrainingPatterns = noOfTrainPat;
System.out.println("no of traing pat= "+numTrainingPatterns );
realRejct = passedTrainingArray;
timePeriod = new double[numTrainingPatterns];
}
```

```

timeSqured = new double[numTrainingPatterns];
calcTrend = new double[numTrainingPatterns];
multiSeasIndex = new double[numTrainingPatterns];
seasIndexAverage = new double[24];
calcRejct = new double[numTrainingPatterns];
calcMeError = new double[numTrainingPatterns];
calcMadError = new double[numTrainingPatterns];
calcMseError = new double[numTrainingPatterns];
calcForecastMeError = new double[24];
calcForecastMadError = new double[24];
calcForecastMseError = new double[24];
realTestRejct = passedTestArray;
forecastTestRejct = new double[24];
dataCollector();

a = aSum / numTrainingPatterns;
b = bSum / tSqrSum;

//Trend calculations
for(int j = 0;j <numTrainingPatterns;j++)
{
    calcTrend[j] = a + (b * (timePeriod[j] +
((numTrainingPatterns/2)- numTrainingPatterns)));
    multiSeasIndex[j] = realRejct[j] / calcTrend[j];
}

for(int i = 0;i<24;i++)
{
    sSum=0.0;
    int counter = 1;
    for (int k=0;k<numTrainingPatterns-24;k=k+24)
    {
        sSum = sSum + multiSeasIndex[k+i];
        counter++;
    }
    seasIndexAverage[i] = sSum / counter;
}

for(int m = 0;m <numTrainingPatterns;m++)
{
    for (int r = 0;r < 24;r++)
    {
        try{calcRejct[m+r] = calcTrend[m+r] * seasIndexAverage[r];}
        catch(ArrayIndexOutOfBoundsException rNull)
            { continue; }
    }
    m = m+23;
}

double [] alphaArray = new double[1250000];
double [] betaArray = new double[1250000];
double [] rmseArray = new double[1250000];
int counter=0;
for (double i=.001D; i<=.3D;i=i+.001D)
{
    for (double j=.0001D;j<=0.01D;j=j+.0001D)
    {
        alphaArray[counter]=i;
        betaArray[counter]=j;
        calculateForecast(alphaArray[counter], betaArray[counter], 0.0);
        rmseArray[counter]= calcForecastingErrors();
        counter++;
    }
}

```

```

arrayIndex = min(rmseArray);
double minRmse=rmseArray[arrayIndex];
alpha = alphaArray[arrayIndex];
beta = betaArray[arrayIndex];
calcForecastingErrors();
returnForecastedResults();
}
//=====
//***** END OF THE MAIN CONSTRUCTOR *****
//=====

public static int[] returnForecastedResults()
{
int [] returnedArray = new int [24];
for(int i=0;i<24;i++)
    {
        returnedArray[i]=(int)Math.round(forecastTestRejct[i]);
    }
return returnedArray;
} // end of method returnForecastedResults
//*****

public static int min(double[] t)
{
double minimum = t[0]; // start with the first value
for (int i=1; i<t.length; i++)
{
    if (t[i] < minimum && t[i]>0.0)
    {
        arrayIndex = i;
    }
}
return arrayIndex;
} //end method min
//*****

public static void calcErrors()
{
meSum=0;
madSum=0;
mseSum=0;
for(int i = 0; i<numTrainingPatterns; i++)
    {
        calcMeError[i] = realRejct[i] - calcRejct[i];
        calcMadError[i] = java.lang.Math.abs(calcMeError[i]);
        calcMseError[i] = calcMeError[i] * calcMeError[i];
        meSum = meSum + calcMeError[i];
        madSum = madSum + calcMadError[i];
        mseSum = mseSum + calcMseError[i];
    }
ME = meSum / numTrainingPatterns;
MAD = madSum / numTrainingPatterns;
MSE = mseSum / numTrainingPatterns;
RMSE = java.lang.Math.sqrt(MSE);
} //end method calcErrors
//*****

public static double calcForecastingErrors()
{
forecastMeSum=0.0;
forecastMadSum=0.0;
forecastMseSum=0.0;
forecastME=0.0;
forecastMAD=0.0;
forecastRMSE=0.0;
}

```

```

for(int i = 0;i<24;i++)
{
    calcForecastMeError[i] = realTestRejct[i] - forecastTestRejct[i];
    calcForecastMadError[i] =
    java.lang.Math.abs(calcForecastMeError[i]);
    calcForecastMseError[i] = calcForecastMeError[i] *
    calcForecastMeError[i];
    forecastMeSum = forecastMeSum + calcForecastMeError[i];
    forecastMadSum = forecastMadSum + calcForecastMadError[i];
    forecastMseSum = forecastMseSum + calcForecastMseError[i];
}
forecastME = forecastMeSum / 24;
forecastMAD = forecastMadSum / 24;
forecastMSE = forecastMseSum / 24;
forecastRMSE = java.lang.Math.sqrt(forecastMSE);
return forecastRMSE;
}

//*****

public static void dataCollector()
{
    aSum=0;
    bSum=0;
    tSqrSum=0;
    for (int z=0;z<numTrainingPatterns;z++)
    {
        timePeriod[z]=z-(numTrainingPatterns/2);
        timeSqured[z] = timePeriod[z] * timePeriod[z];
        aSum = aSum + realRejct[z];
        bSum = bSum + (timePeriod[z] * realRejct[z]);
        tSqrSum = tSqrSum + timeSqured[z];
    }//end of loop
}

// end of method dataCollector()

//*****

public static void calculateForecast(double testAlpha, double testBeta,
double testGamma)
{
    Alpha = testAlpha;
    beta = testBeta;
    gamma = testGamma;
    hwA = (alpha*(realRejct[numTrainingPatterns -1] / seasIndexAverage[23])) +
    ((1-alpha)*(calcTrend[numTrainingPatterns - 1]+b));
    hwB = (beta*(hwA-calcTrend[numTrainingPatterns-1])) + ((1-beta)*b);
    hwI = (gamma*(realRejct[numTrainingPatterns-1]-hwA)) +
    ((1-gamma)*seasIndexAverage[23]);

    for(int z=0;z<24;z++)
    {
        // Forecasting Formula;
        // Ft+m = {hwA(t) + hwB(t) * m}*I(t-L+m)
        // L is set to 24 as it is the length of the seasion
        forecastTestRejct[z] = ((hwA + (hwB*(z+1)))*seasIndexAverage[z]);
    }
} // end of method calculateForecast()

} // end of class HWF

```

Listing A – 15: Java Class Holt-Winter (HWF.java)

#### A.2.2.4 Regression Forecasting Interface Java Code

The following Java class is the regression equation interface that uses Flanagan Regression Class (designed by Dr Michael Thomas Flanagan and obtained from the WWW at <http://www.ee.ucl.ac.uk/~mflanaga/java/Regression.html>).

The Flanagan Regression Class author gave permission to use, copy and modify the original code for NON-COMMERCIAL purposes without fee as long it is acknowledged. However, the DoSTDM regression class uses the Flanagan regression class as a Java library without modifications.

```
// This class implement the Regression class by Dr. Michael Thomas Flanagan

import flanagan.analysis.Regression;
import flanagan.analysis.RegressionFunction;

// define regression function
class Funct implements RegressionFunction
{
public double a = 0.0D;
public double function(double[ ] p, double[ ] x)
    {
        double y = a + p[0]*x[0] + p[1]*x[1] + p[2]*x[2] + p [3]*x[3] +
        p[4]*x[4] + p[5]*x[5] + p[6]*x[6] +p [7]*x[7] + p[8]*x[8] +
        p[9]*x[9] + p[10]*x[10] + p[11]*x[11] + p[12]*x[12] +
        p[13]*x[13] + p[14]*x[14];
        return y;
    }
} // end of method function

public class REG
{
public REG(int startingPoint, int endingPoint)
    {
        int passedArrayLength = endingPoint - startingPoint;
        System.out.println("Start @ "+startingPoint+" and End @
        "+endingPoint);

        // x data array(s) - regression variables
        double[] x1 = new double[passedArrayLength];
        double[] x2 = new double[passedArrayLength];
        double[] x3 = new double[passedArrayLength];
        double[] x4 = new double[passedArrayLength];
        double[] x5 = new double[passedArrayLength];
        double[] x6 = new double[passedArrayLength];
        double[] x7 = new double[passedArrayLength];
        double[] x8 = new double[passedArrayLength];
        double[] x9 = new double[passedArrayLength];
        double[] x10 = new double[passedArrayLength];
        double[] x11 = new double[passedArrayLength];
        double[] x12 = new double[passedArrayLength];
        double[] x13 = new double[passedArrayLength];
        double[] x14 = new double[passedArrayLength];
        double[] x15 = new double[passedArrayLength];
        double[] [] xArray = new double[15][passedArrayLength];
    }
}
```

```

// y data array - rejected regression value
double[] yArray = new double[passedArrayLength];
for (int j = 0; j < passedArrayLength; j++)
{
// intilize regression dummy variables
double D8 = 0.0;
double D9 = 0.0;
double D10 = 0.0;
double D11 = 0.0;
double D12 = 0.0;
double D13 = 0.0;
double D14 = 0.0;
double D15 = 0.0;
double D16 = 0.0;
double D17 = 0.0;
double D18 = 0.0;
String compareTimeString = DoSTDM.trainTimeArray[j];
if(compareTimeString.equals("8:00")){D8 = 1.0;}
if(compareTimeString.equals("9:00")){D9 = 1.0;}
if(compareTimeString.equals("10:00")){D10 = 1.0;}
if(compareTimeString.equals("11:00")){D11 = 1.0;}
if(compareTimeString.equals("12:00")){D12 = 1.0;}
if(compareTimeString.equals("13:00")){D13=1.0;}
if(compareTimeString.equals("14:00")){D14=1.0;}
if(compareTimeString.equals("15:00")){D15=1.0;}
if(compareTimeString.equals("16:00")){D16=1.0;}
if(compareTimeString.equals("17:00")){D17=1.0;}
if(compareTimeString.equals("18:00")){D18=1.0;}

// read training data from DoSTDM master class arrays
x1[j] = DoSTDM.trainTcp_in[j];
x2[j] = DoSTDM.trainIcmp_in[j];
x3[j] = DoSTDM.trainUdp_in[j];
x4[j] = DoSTDM.trainOth_in[j];
x5[j] = D8;
x6[j] = D9;
x7[j] = D10;
x8[j] = D11;
x9[j] = D12;
x10[j] = D13;
x11[j] = D14;
x12[j] = D15;
x13[j] = D16;
x14[j] = D17;
x15[j] = D18;
yArray[j] = DoSTDM.trainRejectedArray[j];

} // end of for loop

// combine x1, x2, x3, ..., x38 data into a common x array
for (int i = 0; i < passedArrayLength; i++)
{
xArray[0][i]=x1[i];
xArray[1][i]=x2[i];
xArray[2][i]=x3[i];
xArray[3][i]=x4[i];
xArray[4][i]=x5[i];
xArray[5][i]=x6[i];
xArray[6][i]=x7[i];
xArray[7][i]=x8[i];
xArray[8][i]=x9[i];
xArray[9][i]=x10[i];
xArray[10][i]=x11[i];
xArray[11][i]=x12[i];
xArray[12][i]=x13[i];

```

```

        xArray[13][i]=x14[i];
        xArray[14][i]=x15[i];

    } // end of for loop

// Create instances of the class holding the function,  $y = a + c1*x1 + c2*x2$ 
// + .... +  $c16*x16$ , regression method

Funct f1 = new Funct();

// create an instance of Regression
Regression reg = new Regression(xArray, yArray);

// call regression using default tolerance and maximum iterations and plot
// display option

reg.linear();
reg.print("regression.txt");
double[] currentCoeff = new double[16];
currentCoeff = reg.coeffPrint();
for(int i=0; i<16; i++)
    {
        RegGraph.cData[i]=currentCoeff[i];
        CompareGraph.cData[i]=currentCoeff[i];
        DoSTDM.cData[i]=currentCoeff[i];
        System.out.println("C["+i+"]\t"+currentCoeff[i]);
    }
} // end of method REG
} // end of class

```

Listing A – 16: Java Class Regression Interface (REG.java)

### A.2.2.5 MLP Neural Network Forecasting Engine Java Code

This code had been written based on the original Java MLP Class from. The author had given authorization to use & modify this code and made it available from his web site ( <http://www.philbrierley.com/code.html> ) for free. However, the code had been completely re-organised for the use of forecasting rejected packets and additional functions beyond the original code listing were added to meet the needs of this research.

```

//*****
/* Original arthur code comments below.          *
//*****
/*
MLP neural network in Java
by Phil Brierley
www.philbrierley.com

This code may be freely used and modified at will

Tanh hidden neurons
Linear output neuron

To include an input bias create an
extra input in the training data
and set to 1

Routines included:

calcNet()
WeightChangesHO()
WeightChangesIH()
initWeights()
initData()
tanh(double x)
displayResults()
calcOverallError()

compiled and tested on
Symantec Cafe Lite
*/
//*****

import java.io.*;
import java.util.*;
import java.lang.Math;

public class MLP
{
    // MLP variables

    // number of training cycles
    public static int numEpochs;

    // number of inputs - this includes the input bias
    public static int numInputs = 16;

    // number of hidden units
    public static int numHidden;

```

```

// number of training patterns
public static int numTrainingPatterns;

// number of testing patterns
public static int numTestPatterns = 24;

// hidden inputs learning rate
public static double LR_IH;

// hidden output learning rate
public static double LR_HO;

// process variables
public static int patNum;          // pattern id
public static double errThisPat; // error rate for a specific pattern
public static double outPred;     // predicted output of a training pattern
public static double testPred;    // predicted output of a testing pattern
public static double RMSerror;    // RMS error rate for the training patterns

// Training data arrays
// 2D array to hold training pattern inputs
public static double[][] trainInputs;
// 1D array to hold training pattern output
public static double[] trainOutput;

// Testing data arrays
// 2D array to hold testing pattern inputs
public static double[][] testInputs;
// 1D array to hold testing pattern output
public static double[] testOutput;
// 1D array to hold testing pattern output
public static double[] testPredOutput;

// The outputs of the hidden neurons
// 1D array to hold hidden neuron cumulative values
public static double[] hiddenVal;

// The weights
// 2D array to hold training pattern inputs hidden weights
public static double[][] weightsIH;
// 1D array to hold training pattern hidden output weights
public static double[] weightsHO;

//Data reader variables & arrays (for Training & Test)
public static int record;          // Train & Test records counter
public static double [] input1;   // Train & Test input1 array
public static double [] input2;   // Train & Test input2 array
public static double [] input3;   // Train & Test input3 array
public static double [] input4;   // Train & Test input4 array
public static double [] input5;   // Train & Test input5 array
public static double [] input6;   // Train & Test input6 array
public static double [] input7;   // Train & Test input7 array
public static double [] input8;   // Train & Test input8 array
public static double [] input9;   // Train & Test input9 array
public static double [] input10;  // Train & Test input10 array
public static double [] input11;  // Train & Test input11 array
public static double [] input12;  // Train & Test input12 array
public static double [] input13;  // Train & Test input13 array
public static double [] input14;  // Train & Test input14 array
public static double [] input15;  // Train & Test input15 array
public static double [] output;   // Train & Test output array
public static double [] normalisedOutput; // Train normalised output array
public static double [] maximumArray; // Train maximum inputs array
public static double [] minimumArray; // Train minimum inputs array

```

```

//=====
//***** THIS IS THE MAIN CONSTRUCTOR *****
//=====

public MLP(int noOfTrainingPaterns, int noOfEpochs, int noOfHidden, double
hiddenInputLR, double hiddenOutputLR)
{
    numEpochs = noOfEpochs;
    numTrainingPatterns = noOfTrainingPaterns;
    numHidden = noOfHidden;
    LR_IH = hiddenInputLR;
    LR_HO = hiddenOutputLR;
    trainInputs = new double[numTrainingPatterns][numInputs];
    trainOutput = new double[numTrainingPatterns];
    testInputs = new double[numTestPatterns][numInputs];
    testOutput = new double[numTestPatterns];
    testPredOutput = new double[numTestPatterns];
    hiddenVal = new double[numHidden];
    weightsIH = new double[numInputs][numHidden];
    weightsHO = new double[numHidden];

    //load in the data
    record = numTrainingPatterns;
    trainDataCollector();

    //initiate the weights
    initWeights();
    miniMaxArrays();
    setNormArrays();
    initTrainData();

    //train the network
    for(int j = 0;j <= numEpochs;j++)
    {
        for(int i = 0;i<numTrainingPatterns;i++)
        {
            //select a pattern at random
            patNum = (int)((Math.random()*numTrainingPatterns)-0.001);

            //calculate the current network output
            //and error for this pattern
            calcNet();

            //change network weights
            WeightChangesHO();
            WeightChangesIH();
        }
        //display the overall network error
        //after each epoch
        calcOverallTrainingError();
        //System.out.println("epoch = " + j + " RMS Error = " + RMSerror);
    }
    //training has finished
    //display the results
    //displayTrainingResults();
    record = numTestPatterns;
    testDataCollector();
    initTestData();

    for (int l=0;l<record;l++)
        { testPredOutput[l] = displayTestResults(l); } //end of loop
    returnForecastedResults();
}
//=====
//***** END OF THE MAIN CONSTRUCTOR *****
//=====

```

```

public static int[] returnForecastedResults()
{
    int [] returnedArray = new int [24];
    for(int i=0;i<24;i++)
    {
        returnedArray[i] = (int)Math.round(testPredOutput[i]);
    } //end of loop
    return returnedArray;
} // end of method returnForecastedResults()

//*****
public static void calcNet()
{
    // calculate the outputs of the hidden neurons
    // the hidden neurons are tanh
    for(int i = 0;i<numHidden;i++)
    {
        hiddenVal[i] = 0.0;
        for(int j = 0;j<numInputs;j++)
            hiddenVal[i] = hiddenVal[i] + (trainInputs[patNum][j]* weightsIH[j][i]);
        hiddenVal[i] = tanh(hiddenVal[i]);
    }

    // calculate the output of the network
    // the output neuron is linear
    outPred = 0.0;
    for(int i = 0;i<numHidden;i++)
        outPred = outPred + hiddenVal[i] * weightsHO[i];

    // calculate the error
    errThisPat = outPred - trainOutput[patNum];
} // end of method calcNet()

//*****
public static void WeightChangesHO()
// adjust the weights hidden-output
{
    for(int k = 0;k<numHidden;k++)
    {
        double weightChange = LR_HO * errThisPat * hiddenVal[k];
        weightsHO[k] = weightsHO[k] - weightChange;
        // regularisation on the output weights
        if (weightsHO[k] < -5)
            weightsHO[k] = -5;
        else if (weightsHO[k] > 5)
            weightsHO[k] = 5;
    } //end of loop
} // end of method WeightChangesHO()

//*****
public static void WeightChangesIH()
// adjust the weights input-hidden
{
    for(int i = 0;i<numHidden;i++)
    {
        for(int k = 0;k<numInputs;k++)
        {
            double x = 1 - (hiddenVal[i] * hiddenVal[i]);
            x = x * weightsHO[i] * errThisPat * LR_IH;
            x = x * trainInputs[patNum][k];
            double weightChange = x;
            weightsIH[k][i] = weightsIH[k][i] - weightChange;
        } // end of loop
    } // end of loop
} // end of method WeightChangesIH()

```

```

//*****
public static void initWeights()
{
    for(int j = 0;j<numHidden;j++)
    {
        weightsHO[j] = (Math.random() - 0.5)/2;
        for(int i = 0;i<numInputs;i++)
        {
            weightsIH[i][j] = (Math.random() - 0.5)/5;
        } // end of loop
    } // end of loop
} // end of method initWeights()

//*****
public static void initTrainData()
{
    // the data at this stage has been rescaled the range [-1][1]
    // an extra input valued 1 is also added to act as the bias

    for(int i = 0;i<record;i++)
    {
        trainInputs[i][0] = input1[i];
        trainInputs[i][1] = input2[i];
        trainInputs[i][2] = input3[i];
        trainInputs[i][3] = input4[i];
        trainInputs[i][4] = input5[i];
        trainInputs[i][5] = input6[i];
        trainInputs[i][6] = input7[i];
        trainInputs[i][7] = input8[i];
        trainInputs[i][8] = input9[i];
        trainInputs[i][9] = input10[i];
        trainInputs[i][10] = input11[i];
        trainInputs[i][11] = input12[i];
        trainInputs[i][12] = input13[i];
        trainInputs[i][13] = input14[i];
        trainInputs[i][14] = input15[i];
        trainInputs[i][15] = 1;//bias
        trainOutput[i] = normalisedOutput[i];
    } // end of loop
} //end of method initTrainData()

//*****
public static void initTestData()
{
    testInputs = new double [record][numInputs];
    testOutput = new double [record];

    // the data here is the normalised test data and it has been rescaled
    // to the range [-1][1] with an extra input valued 1 is also added
    // to act as the bias

    for(int i = 0;i<record;i++)
    {
        testInputs[i][0] = normalise(input1[i],0);
        testInputs[i][1] = normalise(input2[i],1);
        testInputs[i][2] = normalise(input3[i],2);
        testInputs[i][3] = normalise(input4[i],3);
        testInputs[i][4] = normalise(input5[i],4);
        testInputs[i][5] = normalise(input6[i],5);
        testInputs[i][6] = normalise(input7[i],6);
        testInputs[i][7] = normalise(input8[i],7);
        testInputs[i][8] = normalise(input9[i],8);
        testInputs[i][9] = normalise(input10[i],9);
        testInputs[i][10] = normalise(input11[i],10);
        testInputs[i][11] = normalise(input12[i],11);
    }
}

```

```

        testInputs[i][12] = normalise(input13[i],12);
        testInputs[i][13] = normalise(input14[i],13);
        testInputs[i][14] = normalise(input15[i],14);
        testInputs[i][15] = 1;
        testOutput[i] = output[i];
    } // end of loop
} // end of method initTestData()

//*****
public static double tanh(double x)
{
    if (x > 20)
        return 1;
    else if (x < -20)
        return -1;
    else
    {
        double a = Math.exp(x);
        double b = Math.exp(-x);
        return ( a - b ) / ( a + b );
    }
} // end of method tanh

//*****
public static void displayTrainingResults()
{
    for(int i = 0;i<numTrainingPatterns;i++)
    {
        patNum = i;
        calcNet();
    } // end of loop
} // end of method displayTrainingResults()

//*****
public static double displayTestResults(int k)
{
    // calculate the outputs of the hidden neurons the hidden neurons are tanh
    for(int i = 0;i<numHidden;i++)
    {
        hiddenVal[i] = 0.0;

        for(int j = 0;j<numInputs;j++)
            hiddenVal[i] = hiddenVal[i] + (testInputs[k][j] * weightsIH[j][i]);

        hiddenVal[i] = tanh(hiddenVal[i]);
    }

    // calculate the output of the network the output neuron is linear
    double testPred = 0.0;

    for(int i = 0;i<numHidden;i++)
    {
        testPred = testPred + hiddenVal[i] * weightsHO[i];
    }

    double shift=testOutput[k]-deNormalise(testPred,15);
    System.out.println("Test Pattern #"+(k+1)+" Rejected = " + testOutput[k]
    + " & MLP model Output = " + deNormalise(testPred,15) + " Results Shift
    = " + shift);
    return deNormalise(testPred,15);
} // end of method displayTestResults

```

```

//*****
public static void calcOverallTrainingError()
{
    RMSerror = 0.0;
    for(int i = 0;i<numTrainingPatterns;i++)
    {
        patNum = i;
        calcNet();
        RMSerror = RMSerror + (errThisPat * errThisPat);
    }
    RMSerror = RMSerror / numTrainingPatterns;
    RMSerror = java.lang.Math.sqrt(RMSerror);
} // end of method calcOverallTrainingError()

//*****
public static void trainDataCollector()
{
    input1 = new double [record];
    input2 = new double [record];
    input3 = new double [record];
    input4 = new double [record];
    input5 = new double [record];
    input6 = new double [record];
    input7 = new double [record];
    input8 = new double [record];
    input9 = new double [record];
    input10 = new double [record];
    input11 = new double [record];
    input12 = new double [record];
    input13 = new double [record];
    input14 = new double [record];
    input15 = new double [record];
    output = new double [record];
    normalisedOutput = new double [record];

    for(int z=0;z<record;z++)
    {
        // TCP training input
        input1[z] = (double) DoSTDM.trainTcp_in[z];
        input2[z] = (double) DoSTDM.trainTcp_out[z];
        input3[z] = (double) DoSTDM.trainTcp_accpt[z];

        // IP training input
        input4[z] = (double) DoSTDM.trainIp_in[z];
        input5[z] = (double) DoSTDM.trainIp_out[z];
        input6[z] = (double) DoSTDM.trainIp_accpt[z];

        // ICMP training input
        input7[z] = (double) DoSTDM.trainIcmp_in[z];
        input8[z] = (double) DoSTDM.trainIcmp_out[z];
        input9[z] = (double) DoSTDM.trainIcmp_accpt[z];

        // UDP training input
        input10[z] = (double) DoSTDM.trainUdp_in[z];
        input11[z] = (double) DoSTDM.trainUdp_out[z];
        input12[z] = (double) DoSTDM.trainUdp_accpt[z];

        // Other protocols (if any) training input
        input13[z] = (double) DoSTDM.trainOth_in[z];
        input14[z] = (double) DoSTDM.trainOth_out[z];
        input15[z] = (double) DoSTDM.trainOth_accpt[z];

        // Rejected training output
        output[z] = (double) DoSTDM.trainRejectedArray[z];
    } //end of loop
} // end of method trainDataCollector()

```

```

//*****
public static void testDataCollector()
{
    input1 = new double [record];
    input2 = new double [record];
    input3 = new double [record];
    input4 = new double [record];
    input5 = new double [record];
    input6 = new double [record];
    input7 = new double [record];
    input8 = new double [record];
    input9 = new double [record];
    input10 = new double [record];
    input11 = new double [record];
    input12 = new double [record];
    input13 = new double [record];
    input14 = new double [record];
    input15 = new double [record];
    output = new double [record];

    for(int z=0;z<record;z++)
    {
        // TCP testing input
        input1[z] = (double) DoSTDM.testTcp_in[z];
        input2[z] = (double) DoSTDM.testTcp_out[z];
        input3[z] = (double) DoSTDM.testTcp_accpt[z];

        // IP testing input
        input4[z] = (double) DoSTDM.testIp_in[z];
        input5[z] = (double) DoSTDM.testIp_out[z];
        input6[z] = (double) DoSTDM.testIp_accpt[z];

        // ICMP testing input
        input7[z] = (double) DoSTDM.testIcmp_in[z];
        input8[z] = (double) DoSTDM.testIcmp_out[z];
        input9[z] = (double) DoSTDM.testIcmp_accpt[z];

        // UDPP testing input
        input10[z] = (double) DoSTDM.testUdp_in[z];
        input11[z] = (double) DoSTDM.testUdp_out[z];
        input12[z] = (double) DoSTDM.testUdp_accpt[z];

        // Other protocols (if any) testing input
        input13[z] = (double) DoSTDM.testOth_in[z];
        input14[z] = (double) DoSTDM.testOth_out[z];
        input15[z] = (double) DoSTDM.testOth_accpt[z];

        // Rejected testing output
        output[z] = (double) DoSTDM.testRejectedArray[z];
    } //end of loop
} // end of method testDataCollector()

//*****
public static void setNormArrays()
{
    for (int u=0;u<record;u++)
    {
        // sorting arrays of normalised values
        input1[u] = normalise(input1[u],0);
        input2[u] = normalise(input2[u],1);
        input3[u] = normalise(input3[u],2);
        input4[u] = normalise(input4[u],3);
        input5[u] = normalise(input5[u],4);
        input6[u] = normalise(input6[u],5);
        input7[u] = normalise(input7[u],6);
    }
}

```

```

        input8[u] = normalise(input8[u],7);
        input9[u] = normalise(input9[u],8);
        input10[u] = normalise(input10[u],9);
        input11[u] = normalise(input11[u],10);
        input12[u] = normalise(input12[u],11);
        input13[u] = normalise(input13[u],12);
        input14[u] = normalise(input14[u],13);
        input15[u] = normalise(input15[u],14);
        normalisedOutput[u] = normalise(output[u],15);
    } // end of loop
} // end of method setNormArrays()

//*****
public static void miniMaxArrays()
{
    // initialize maximum and minimum arrays
    maximumArray = new double [16];
    minimumArray = new double [16];

    // adding inputs maximum to maximum arrays
    maximumArray[0] = max(input1);
    maximumArray[1] = max(input2);
    maximumArray[2] = max(input3);
    maximumArray[3] = max(input4);
    maximumArray[4] = max(input5);
    maximumArray[5] = max(input6);
    maximumArray[6] = max(input7);
    maximumArray[7] = max(input8);
    maximumArray[8] = max(input9);
    maximumArray[9] = max(input10);
    maximumArray[10] = max(input11);
    maximumArray[11] = max(input12);
    maximumArray[12] = max(input13);
    maximumArray[13] = max(input14);
    maximumArray[14] = max(input15);
    maximumArray[15] = max(output);

    // adding inputs minimum to minimum arrays
    minimumArray[0] = min(input1);
    minimumArray[1] = min(input2);
    minimumArray[2] = min(input3);
    minimumArray[3] = min(input4);
    minimumArray[4] = min(input5);
    minimumArray[5] = min(input6);
    minimumArray[6] = min(input7);
    minimumArray[7] = min(input8);
    minimumArray[8] = min(input9);
    minimumArray[9] = min(input10);
    minimumArray[10] = min(input11);
    minimumArray[11] = min(input12);
    minimumArray[12] = min(input13);
    minimumArray[13] = min(input14);
    minimumArray[14] = min(input15);
    minimumArray[15] = min(output);
} //end of method miniMaxArrays()

//*****
public static double normalise(double input, int id)
{
    //this function normalises the inputs to lie in the range -1 to +1
    double v1 = 0.0;
    double v2 = 0.0;
    double v3 = 0.0;
    double v4 = 0.0;
    double v5 = 0.0;

```

```

        if(input==0.0 && maximumArray[id]==0.0 && minimumArray[id]==0.0)
        { v5 = 0.0; }
        else
        {
            v1 = input - minimumArray[id];
            v2 = maximumArray[id] - minimumArray[id];
            v3 = v1 / v2;
            v4 = (v3 - ((maximumArray[id] - minimumArray[id]) /
                (maximumArray[id] + minimumArray[id])));
            v5 = (v4 * ((maximumArray[id] + minimumArray[id]) /
                (maximumArray[id] - minimumArray[id])));
        }
        return v5; //return normalised input value
    } //end of method normalise

//*****
public static double deNormalise(double modeledOutput, int id)
{
//this function normalises the modelled outputs to lie in the range -1 to +1
double v0 = 0.0;
double v1 = 0.0;
double v2 = 0.0;
double v3 = 0.0;
double v4 = 0.0;

v4 = (modeledOutput / ((maximumArray[id] + minimumArray[id]) /
    (maximumArray[id] - minimumArray[id])));
v3 = (v4 + ((maximumArray[id] - minimumArray[id]) / (maximumArray[id]
    + minimumArray[id])));
v2 = maximumArray[id] - minimumArray[id];
v1 = v3 * v2;
v0 = v1 + minimumArray[id];
return v0; //return normalised modelled output value
}

//*****
public static double max(double[] t)
{
    double maximum = t[0]; // start with the first value
    for (int i=1; i<t.length; i++)
    {
        if (t[i] > maximum)
        {
            maximum = t[i]; // new maximum
        }
    } //end of loop
    return maximum; //return maximum value within the array
} //end method max

//*****
public static double min(double[] t)
{
    double minimum = t[0]; // start with the first value
    for (int i=1; i<t.length; i++) {
        if (t[i] < minimum)
        {
            minimum = t[i]; // new maximum
        }
    } //end of loop
    return minimum; //return minimum value within the array
} //end method min
} //end of class MLP

```

**Listing A – 17: Java Class MLP (MLP.java, Based on Phil Brierley MLP Code)**

### A.2.2.6 Formatting & Presentation Java Classes

The following Java classes code is used by DoSTDM applet to format the applet graphical analysis output. These classes had been written in Java and the compiled code can be found in Appendix – C. The code internal documentation explains the function of each class and the methods used by these classes.

#### A.2.2.6.1 Class AnalysisTable.Java

```
/**
 * This class extends JPanel and override the paintComponent() method to
 * be able to draw graphics within a JPanel using Graphics and
 * Graphics2D classes. It can draw data from arrays that been collected
 * passed on used selection of the data to be draw.
 */

public class AnalysisTable extends JPanel
{
// defining string identification variables for user choices
    public static String passedYear;
    public static String passedMonth;
    public static String passedDay;
    public static String passedHWME;
    public static String passedHWMAD;
    public static String passedHWRMSE;
    public static String passedRegME;
    public static String passedRegMAD;
    public static String passedRegRMSE;
    public static String passedMlpME;
    public static String passedMlpMAD;
    public static String passedMlpRMSE;

// defining drawing panel x-axis and y-axis virtual minimum and maximum
// dimentions variables
    public static int vxmin;
    public static int vxmax;
    public static int vymin;
    public static int vymax;

public AnalysisTable(double MlpME, double HWME, double RegME, double MlpMAD,
double HWMAD, double RegMAD, double MlpRMSE, double HWRMSE, double RegRMSE,
String graphYear, String graphMonth, String graphDay)
{
    // user choices for the drawing
    passedYear=graphYear;
    passedMonth=graphMonth;
    passedDay=graphDay;
    passedMlpME=""+(float)MlpME;
    passedHWME=""+(float)HWME;
    passedRegME=""+(float)RegME;
    passedMlpMAD=""+(float)MlpMAD;
    passedHWMAD=""+(float)HWMAD;
    passedRegMAD=""+(float)RegMAD;
    passedMlpRMSE=""+(float)MlpRMSE;
    passedHWRMSE=""+(float)HWRMSE;
    passedRegRMSE=""+(float)RegRMSE;
    // screen coordinates
    vxmin=85;
    vxmax=610;
    vymin=110;
    vymax=250;}
}
```

```

/**
 * The method graph the given data using Graphics and Graphics2D
 * classes to override the paintComponent() method provided by standard java
 * classes. It can graph bars, pies and line charts based on user selection
 * from DoSTDM Applet.
 */

public void paintComponent(Graphics g)
{
    g.setColor(Color.black);
    g.drawLine(10,0,658,0);//black
    g.drawLine(12,2,656,2);//black
    g.drawLine(10,116,658,116);//black
    g.drawLine(12,114,656,114);//black
    g.drawLine(10,0,10,116);//black
    g.drawLine(12,0,12,114);//black
    g.drawLine(658,0,658,116);//black
    g.drawLine(656,2,656,114);//black
    g.setColor(Color.green);
    g.drawLine(11,115,657,115);//green
    g.drawLine(11,1,657,1);//green
    g.drawLine(11,1,11,115);//green
    g.drawLine(657,1,657,115);//green

    //draw graph title
    g.setColor (Color.black);
    g.setFont(new Font("Arial", Font.BOLD, 15));
    String strTitle=" Model Type ME MAD RMSE";
    String spacer = "-----";
    g.drawString(strTitle, vxmin, vymin-80);
    g.drawString(spacer, vxmin, vymin-70);
    g.drawString("NN-MLP", vxmin, vymin-50);
    g.drawString("Dynamic", vxmin+130, vymin-50);
    g.drawString(passedMlpME, vxmin+230, vymin-50);
    g.drawString(passedMlpMAD, vxmin+330, vymin-50);
    g.drawString(passedMlpRMSE, vxmin+430, vymin-50);
    g.drawString("Holt-Winter", vxmin, vymin-30);
    g.drawString("Dynamic", vxmin+130, vymin-30);
    g.drawString(passedHWME, vxmin+230, vymin-30);
    g.drawString(passedHWMAD, vxmin+330, vymin-30);
    g.drawString(passedHWRMSE, vxmin+430, vymin-30);
    g.drawString("Regression", vxmin, vymin-10);
    g.drawString("Dynamic", vxmin+130, vymin-10);
    g.drawString(passedRegME, vxmin+230, vymin-10);
    g.drawString(passedRegMAD, vxmin+330, vymin-10);
    g.drawString(passedRegRMSE, vxmin+430, vymin-10);
} // end of method paintComponent
} // end of class AnalysisTable

```

Listing A – 18: Java Class Analysis Table (Analysis.java)

#### A.2.2.6.2 Class AlignedListCellRenderer.Java

```

/*
 * Class to render and align combobox selection, this class customize the
 * DefaultListCellRenderer Java Class for combobox selection.
 * It is partially based on Java code written by Harald Barsnes found on:
 * http://code.google.com/p/store-bioinfo-gui/source/browse/trunk/src/main/
 * java/no/uib/storbioinfoGUI/util/AlignedListCellRenderer.java?r=10
 * retrieved from the web on April 2012.
 */

```

```

class AlignedListCellRenderer extends DefaultListCellRenderer
{
private int align;

public AlignedListCellRenderer(int align)
{
this.align = align;
}

public Component getListCellRendererComponent(JList list, Object value,
int index, boolean isSelected, boolean cellHasFocus)
{
// DefaultListCellRenderer uses a JLabel as the rendering component:
JLabel myLabel = (JLabel)super.getListCellRendererComponent(
list, value, index, isSelected, cellHasFocus);
myLabel.setHorizontalAlignment(align);
return myLabel;
} // end of method getListCellRendererComponent

} // end of class AlignedListCellRenderer

```

**Listing A – 19: Java Class Aligned List Cell Renderer  
(AlignedListCellRenderer.java)**

### **A.2.2.6.3 Class Sort.Java**

```

/**
 * This class sort array of numbers to find the maximum and the minimum of
 * values within the array and then sort the array from smallest value to
 * largest value using selection sort algorithm. The original idea of this
 * sort algorithm was obtained from the book:
 * Java An Introduction To Computer Science and Programming)
 * by Walter Savitch, 1999 - Printce Hall-ISBN 013-287426-1.
 */

public class Sort
{
/**
 * The Sort() constructor initialize class variables for the array to
 * be sorted. It then finds the next smallest element and change it is
 * position within the array.
 */

public Sort(int[] elementsArray)
{
int elementIdx;
int nextSmallElementIdx;
for (elementIdx=0;elementIdx<elementsArray.length-1;elementIdx++)
{
nextSmallElementIdx = smallestElementIdx(elementIdx, elementsArray);
swapElement(elementIdx, nextSmallElementIdx, elementsArray);
}
} //end of sort constructor

/**
 * The smallestElementIdx() method will find the next smallest element
 * index and return it back to the constructor to insert it in a lower
 * index position within the array using swapelement() method.
 * to swap current smallest element to the top
 */

```

```

private static int smallestElementIdx(int startIdx, int [] elementsArray)
{
    int min = elementsArray[startIdx];
    int minIdx = startIdx;
    int idx;
    for (idx=startIdx+1;idx<elementsArray.length;idx++)
    {
        if (elementsArray[idx]<min)
        {
            min = elementsArray[idx];
            minIdx = idx;
        }
    }

    return minIdx;
}

} // end of method smallestElementIdx()

/**
 * The swapElement() method will swap the position within the array for any
 * two given elements using temporary element value (int tempElement)..
 */

private static void swapElement(int elementI , int elementJ, int[]
elementsArray)
{
    int tempElement;
    tempElement = elementsArray[elementI];
    elementsArray[elementI]=elementsArray[elementJ];
    elementsArray[elementJ]=tempElement;
} // end of method swapElement()
} // end of class Sort()

```

Listing A – 20: Java Class Sort (Sort.java)

#### A.2.2.6.4 Class DayOfTheWeek.Java

```

/**
 * This class Determining the Day-of-Week for a Particular Date.
 * The day-of-week is an integer value where 1 is Sunday, 2 is
 * Monday, ..., and 7 is Saturday
 */

public class DayOfTheWeek
{
    public static String daysList[] =
    {"Sunday", "Monday", "Tuesday", "Wendsday", "Thuerday", "Friday", "Saturday"};

    /**
     * The DayOfTheWeek() constructor initialize class variables for the day
     * of the week to be generated. It then use a lookup array to convert the
     * returned integer value to string representing that day.
     */

    public static String DayOfTheWeek(int passedDay, String passedMonth, int
passedYear)
    {
        // initialize day string value to null
        String dayString = "";
        int passedMonthValue = 0;
    }
}

```

```

if(passedMonth.equalsIgnoreCase("jan")) {passedMonthValue=0;}
if(passedMonth.equalsIgnoreCase("feb")) {passedMonthValue=1;}
if(passedMonth.equalsIgnoreCase("mar")) {passedMonthValue=2;}
if(passedMonth.equalsIgnoreCase("apr")) {passedMonthValue=3;}
if(passedMonth.equalsIgnoreCase("may")) {passedMonthValue=4;}
if(passedMonth.equalsIgnoreCase("jun")) {passedMonthValue=5;}
if(passedMonth.equalsIgnoreCase("jul")) {passedMonthValue=6;}
if(passedMonth.equalsIgnoreCase("aug")) {passedMonthValue=7;}
if(passedMonth.equalsIgnoreCase("sep")) {passedMonthValue=8;}
if(passedMonth.equalsIgnoreCase("oct")) {passedMonthValue=9;}
if(passedMonth.equalsIgnoreCase("nov")) {passedMonthValue=10;}
if(passedMonth.equalsIgnoreCase("dec")) {passedMonthValue=11;}

// create a Gregorian Calendar instant with passed day, month & year
GregorianCalendar gCal = new GregorianCalendar(passedYear, passedMonthValue,
passedDay);
int dayOfWeek = gCal.get(Calendar.DAY_OF_WEEK); // 6=Friday
dayString = daysList[dayOfWeek-1];

// return day string value
return dayString;
}

} // end of class DayOfTheWeek()

```

Listing A – 21: Java Class Day of The Week (DayOfTheWeek.java)

#### A.2.2.6.5 Class DailyGraph.Java

```

/**
 * This class extends JPanel and override the default paintComponent()
 * method to draw daily selected data within a JPanel using Graphics and
 * Graphics2D classes.
 */

public class DailyGraph extends JPanel
{
// defining string identification variables for user choices
    public static String passedDate;
    public static String passedYear;
    public static String passedMonth;
    public static String passedDay;
    public static String passedProtocol;
    public static String passedStatus;
    public static int passedArraySize;
    public static String passedChartType;

// defining drawing panel x-axis and y-axis virtual minimum and maximum
// dimention variables
    public static int vxmin;
    public static int vxmax;
    public static int vymin;
    public static int vymax;

// define data arrays imported from DoSTDM.class
    public static String [] passedDateArray;
    public static String [] passedTimeArray;
    public static int [] passedProtocolArray;

```

```

// defining arrays to hold drawing data within
    public static String [] xData = new String [24];
    public static int [] yData = new int [24];
    public static int [] sortedYData = new int [24];

// defining drawing data minimum and maximum
    public static int max,min;
    public static int maximumValueOfTheDay;
    public static int minimumValueOfTheDay;

/**
 * The DailyGraph() constructor initialize drawing graphics panel
 * components based on passed choices from the DoSTDM applet.
 */

public DailyGraph(String date, String protocol, String status, int size,
String graphYear, String graphMonth, String graphDay)
{
// user choices for the drawing
passedDate=date;
passedYear=graphYear;
passedMonth=graphMonth;
passedDay=graphDay;
passedProtocol=protocol;
passedStatus=status;
passedArraySize=size;
passedDateArray=new String [passedArraySize];
passedTimeArray=new String [passedArraySize];
passedProtocolArray=new int [passedArraySize];
minimumValueOfTheDay=0;
maximumValueOfTheDay=0;
for (int k=0;k<passedArraySize;k++)
    {passedDateArray[k]=DoSTDM.dateArray[k];
    passedTimeArray[k]=DoSTDM.timeArray[k];}

// calling tcp_in data
if((passedProtocol.equalsIgnoreCase("tcp")) &&
(passedStatus.equalsIgnoreCase("in")))
    {for (int k=0;k<passedArraySize;k++)
        {passedProtocolArray[k]=DoSTDM.tcp_in[k];}}

// calling tcp_out data
if((passedProtocol.equalsIgnoreCase("tcp")) &&
(passedStatus.equalsIgnoreCase("out")))
    {for (int k=0;k<passedArraySize;k++)
        {passedProtocolArray[k]=DoSTDM.tcp_out[k];}}

// calling tcp_accpt data
if((passedProtocol.equalsIgnoreCase("tcp")) &&
(passedStatus.equalsIgnoreCase("acctp")))
    {for (int k=0;k<passedArraySize;k++)
        {passedProtocolArray[k]=DoSTDM.tcp_accpt[k];}}

// calling tcp_rejct data
if((passedProtocol.equalsIgnoreCase("tcp")) &&
(passedStatus.equalsIgnoreCase("rejct")))
    {for (int k=0;k<passedArraySize;k++)
        {passedProtocolArray[k]=DoSTDM.tcp_rejct[k];}}

// calling ip_in data
if((passedProtocol.equalsIgnoreCase("ip")) &&
(passedStatus.equalsIgnoreCase("in")))
    {for (int k=0;k<passedArraySize;k++)
        {passedProtocolArray[k]=DoSTDM.ip_in[k];}}

```

```

// calling ip_out data
if((passedProtocol.equalsIgnoreCase("ip")) &&
(passedStatus.equalsIgnoreCase("out")))
    {for (int k=0;k<passedArraySize;k++)
        {passedProtocolArray[k]=DoSTDM.ip_out[k];}}

// calling ip_acpt data
if((passedProtocol.equalsIgnoreCase("ip")) &&
(passedStatus.equalsIgnoreCase("acpt")))
    {for (int k=0;k<passedArraySize;k++)
        {passedProtocolArray[k]=DoSTDM.ip_acpt[k];}}

//calling ip_rejct data
if((passedProtocol.equalsIgnoreCase("ip")) &&
(passedStatus.equalsIgnoreCase("rejct")))
    {for (int k=0;k<passedArraySize;k++)
        {passedProtocolArray[k]=DoSTDM.ip_rejct[k];}}

// calling icmp_in data
if((passedProtocol.equalsIgnoreCase("icmp")) &&
(passedStatus.equalsIgnoreCase("in")))
    {for (int k=0;k<passedArraySize;k++)
        {passedProtocolArray[k]=DoSTDM.icmp_in[k];}}

// calling icmp_out data
if((passedProtocol.equalsIgnoreCase("icmp")) &&
(passedStatus.equalsIgnoreCase("out")))
    {for (int k=0;k<passedArraySize;k++)
        {passedProtocolArray[k]=DoSTDM.icmp_out[k];}}

// calling icmp_acpt data
if((passedProtocol.equalsIgnoreCase("icmp")) &&
(passedStatus.equalsIgnoreCase("acpt")))
    {for (int k=0;k<passedArraySize;k++)
        {passedProtocolArray[k]=DoSTDM.icmp_acpt[k];}}

// calling icmp_rejct data
if((passedProtocol.equalsIgnoreCase("icmp")) &&
(passedStatus.equalsIgnoreCase("rejct")))
    {for (int k=0;k<passedArraySize;k++)
        {passedProtocolArray[k]=DoSTDM.icmp_rejct[k];}}

// calling udp_in data
if((passedProtocol.equalsIgnoreCase("udp")) &&
(passedStatus.equalsIgnoreCase("in")))
    {for (int k=0;k<passedArraySize;k++)
        {passedProtocolArray[k]=DoSTDM.udp_in[k];}}

// calling udp_out data
if((passedProtocol.equalsIgnoreCase("udp")) &&
(passedStatus.equalsIgnoreCase("out")))
    {for (int k=0;k<passedArraySize;k++)
        {passedProtocolArray[k]=DoSTDM.udp_out[k];}}

// calling udp_acpt data
if((passedProtocol.equalsIgnoreCase("udp")) &&
(passedStatus.equalsIgnoreCase("acpt")))
    {for (int k=0;k<passedArraySize;k++)
        {passedProtocolArray[k]=DoSTDM.udp_acpt[k];}}

// calling udp_rejct data
if((passedProtocol.equalsIgnoreCase("udp")) &&
(passedStatus.equalsIgnoreCase("rejct")))

    {for (int k=0;k<passedArraySize;k++)
        {passedProtocolArray[k]=DoSTDM.udp_rejct[k];}}

```

```

// calling oth_in data
if((passedProtocol.equalsIgnoreCase("oth")) &&
(passedStatus.equalsIgnoreCase("in")))
    {for (int k=0;k<passedArraySize;k++)
        {passedProtocolArray[k]=DoSTDM.oth_in[k];}}

// calling oth_out data
if((passedProtocol.equalsIgnoreCase("oth")) &&
(passedStatus.equalsIgnoreCase("out")))
    {for (int k=0;k<passedArraySize;k++)
        {passedProtocolArray[k]=DoSTDM.oth_out[k];}}

// calling oth_accpt data
if((passedProtocol.equalsIgnoreCase("oth")) &&
(passedStatus.equalsIgnoreCase("acctp")))
    {for (int k=0;k<passedArraySize;k++)
        {passedProtocolArray[k]=DoSTDM.oth_accpt[k];}}

// calling oth_rejct data
if((passedProtocol.equalsIgnoreCase("oth")) &&
(passedStatus.equalsIgnoreCase("rejct")))
    {for (int k=0;k<passedArraySize;k++)
        {passedProtocolArray[k]=DoSTDM.oth_rejct[k];}}

// screen coordinates
vxmin=85;
vxmax=610;
vymin=110;
vymax=250;

int b=0;
for (int j = 0; j < passedArraySize; j++)
{
    if((passedDateArray[j]).equals(passedDate))
    {
        xData[b]=passedTimeArray[j];
        yData[b]=passedProtocolArray[j];
        b++;}
}
int q=0;
while (q<yData.length)
{
    // assign data to sorted array to hold values that need to be
    // sorted for a given array
    sortedYData[q]=yData[q];
    q++;
}

// sort selected source data
new Sort(sortedYData);
/ identify minimum and maximum values
min = sortedYData[0];
max = sortedYData[(sortedYData.length)-1];
DoSTDM.passedMinOfDay=min;
DoSTDM.passedMaxOfDay=max;

} //end of DailyGraph constructor method
// call Graphics Paint Component method to graph the data
public void paintComponent(Graphics g) { }

} // end of class DailyGraph

```

Listing A – 22: Java Class Daily Graph (ResidualsGraph.java)

### A.2.2.6.6 Class HWGraph.Java

```
/**
 * This class extends JPanel and override the paintComponent() method to
 * graph Holt-Winter forecasted data.
 */

public class HWGraph extends JPanel
{
//defining string identification variables for user choices
public static String passedYear;
public static String passedMonth;
public static String passedDay;
public static int passedArraySize;
public static double ME; // ME error rate
public static double MAD; // MAD error rate
public static double MSE; // MSE error rate
public static double RMSE; // RMSE error rate
public static double meSum; // ME error (sum) component
public static double madSum; // MAD error (sum) component
public static double mseSum; // MSE error (sum) component
public static double [] calcMeError; //Calculated Reject array ME error
public static double [] calcMadError; //Calculated Reject array MAD error
public static double [] calcMseError; //Calculated Reject array MSE error

// defining drawing panel x-axis and y-axis virtual minimum and maximum
//dimentions variables
public static int vxmin;
public static int vxmax;
public static int vymin;
public static int vymax;

// define data arrays imported from DoSTDM.class
public static double [] passedTrainingDataArray;

// defining arrays to hold drawing data within
public static String [] xData = new String [24];
public static int [] yData = new int [24];
public static int [] sortedYData = new int [24];

public static int [] zData = new int [24];
public static int [] sortedZData = new int [24];
public static double [] testArray = new double [24];

// defining drawing data minimum and maximum
public static int min,max,minY,maxY,minZ,maxZ;
public static int maximumValueOfTheDay;
public static int minimumValueOfTheDay;

/**
 * The HWGraph() constructor initialize drawing graphics panel
 * components based on passed choices from the DoSTDM applet.
 */

public HWGraph(int sIndx, int eIndx, String graphYear, String graphMonth,
String graphDay)
{
ME=0.0; // ME error rate
MAD=0.0; // MAD error rate
MSE=0.0; // MSE error rate
RMSE=0.0; // RMSE error rate
meSum=0.0; // ME error (sum) component
madSum=0.0; // MAD error (sum) component
mseSum=0.0; // MSE error (sum) component
```

```

// user choices for the drawing
passedYear=graphYear;
passedMonth=graphMonth;
passedDay=graphDay;
passedArraySize= eIndx - sIndx;
passedTrainingDataArray=new double [passedArraySize];
testArray=new double [24];
minimumValueOfTheDay=0;
maximumValueOfTheDay=0;

for (int k=0;k<passedArraySize;k++)
    {
        passedTrainingDataArray[k]=(double)DoSTDM.trainRejectedArray[k];
    }

// screen coordinates
vxmin=85;
vxmax=610;
vymin=110;
vymax=250;

int b=1;
for (int j = 0; j < 24; j++)
    {
        xData[j]=Integer.toString(b);
        testArray[j]=(double)DoSTDM.testRejectedArray[j];
        zData[j]=DoSTDM.testRejectedArray[j];
        b++;
    }

// create a new instance of Holt-Winter forecasting class
HWF myHwf = new HWF(passedTrainingDataArray,testArray,passedArraySize);
yData = myHwf.returnForecastedResults();

calcMeError = new double[24];
calcMadError = new double[24];
calcMseError = new double[24];

for (int j = 0; j < 24; j++)
    {
        DoSTDM.HWResiduals[j]=zData[j]-yData[j];
        calcMeError[j] = zData[j]-yData[j];;
        calcMadError[j] = java.lang.Math.abs(calcMeError[j]);
        calcMseError[j] = calcMadError[j] * calcMadError[j];
        meSum = meSum + calcMeError[j];
        madSum = madSum + calcMadError[j];
        mseSum = mseSum + calcMseError[j];
    }

ME = meSum/24;
MAD = madSum/24;
MSE = mseSum/24;
RMSE = java.lang.Math.sqrt(MSE);
DoSTDM.HWME=ME;
DoSTDM.HWMAD=MAD;
DoSTDM.HWRMSE=RMSE;

int q=0;
while (q<yData.length)
    {
        sortedYData[q]=yData[q];
        sortedZData[q]=zData[q];
        q++;
    }

```

```

// find the minimum and the maximum values of the data
new Sort(sortedYData);
new Sort(sortedZData);
minY = sortedYData[0];
maxY = sortedYData[(sortedYData.length)-1];
minZ = sortedZData[0];
maxZ = sortedZData[(sortedZData.length)-1];

if(maxZ>maxY){max=maxZ;}
else {if(maxY>maxZ){max=maxY;}
else {if(maxY==maxZ){max=maxY;}}}
if(minZ<minY){min=minZ;}
else {if(minY<minZ){min=minY;}
else {if(minY==minZ){min=minY;}}}
max=max-min;

} // end of HWGraph constructor method

// call Graphics Paint Component method to graph the data
// public void paintComponent(Graphics g) { }

} // end of class HWGraph

```

Listing A – 23: Java Class Holt-Winter Graph (HWGraph.java)

#### A.2.2.6.7 Class RegGraph.Java

```

/**
 * This class extends JPanel and override the paintComponent() method to
 * graph Regression forecasted data.
 */

public class RegGraph extends JPanel
{

// defining string identification variables for user choices
public static String passedDate;
public static String passedYear;
public static String passedMonth;
public static String passedDay;
public static int passedArraySize;
public static String passedChartType;
public static double ME; // ME error rate
public static double MAD; // MAD error rate
public static double MSE; // MSE error rate
public static double RMSE; // RMSE error rate
public static double meSum; // ME error (sum) component
public static double madSum; // MAD error (sum) component
public static double mseSum; // MSE error (sum) component
public static double [] calcMeError; //Calculated Reject array ME error
public static double [] calcMadError; //Calculated Reject array MAD error
public static double [] calcMseError; //Calculated Reject array MSE error

// defining drawing panel x-axis and y-axis virtual minimum and maximum
// dimentions variables
public static int vxmin;
public static int vxmax;
public static int vymin;
public static int vymax;

//define data arrays imported from DoSTDM.class
public static String [] passedDateArray;
public static String [] passedTimeArray;

```

```

// defining arrays to hold drawing data within
public static String [] xData = new String [24];
public static int [] yData = new int [24];
public static int [] sortedYData = new int [24];
public static int [] zData = new int [24];
public static int [] sortedZData = new int [24];
public static double [] testArray = new double [24];

// array to hold regression equation coefficients
public static double [] cData = new double [16];

// defining drawing data minimum and maximum
public static int min,max,minY,maxY,minZ,maxZ;
public static int maximumValueOfTheDay;
public static int minimumValueOfTheDay;

/**
 * The RegressionGraph() constructor initialize drawing graphics panel
 * components based on passed choices from the DoSTDM applet.
 */

public RegGraph(int startIdx, int endIdx, String graphYear, String
graphMonth, String graphDay)
{
ME=0.0; // ME error rate
MAD=0.0; // MAD error rate
MSE=0.0; // MSE error rate
RMSE=0.0; // RMSE error rate
meSum=0.0; // ME error (sum) component
madSum=0.0; // MAD error (sum) component
mseSum=0.0; // MSE error (sum) component

// creat an instance of the Regression class to obtain forecasted data
REG myREG = new REG(startIdx, endIdx);

// user choices for the drawing
passedYear=graphYear;
passedMonth=graphMonth;
passedDay=graphDay;
passedDate=passedDay+(passedMonth.substring(0,3))+passedYear;
passedDateArray=new String [24];
passedTimeArray=new String [24];
minimumValueOfTheDay=0;
maximumValueOfTheDay=0;

for (int k=0;k<24;k++)
{
passedDateArray[k]=DoSTDM.testDateArray[k];
passedTimeArray[k]=DoSTDM.testTimeArray[k];
}

// screen coordinates
vxmin=85;
vxmax=610;
vymin=110;
vymax=250;

// setting regression dummy variables data
for (int j = 0; j < 24; j++)
{
int D8=0;
int D9=0;
int D10=0;
int D11=0;
int D12=0;
}

```

```

int D13=0;
int D14=0;
int D15=0;
int D16=0;
int D17=0;
int D18=0;

if((passedDateArray[j]).equals(passedDate))
{
if(passedTimeArray[j].equals("8:00")){D8=1;}
if(passedTimeArray[j].equals("9:00")){D9=1;}
if(passedTimeArray[j].equals("10:00")){D10=1;}
if(passedTimeArray[j].equals("11:00")){D11=1;}
if(passedTimeArray[j].equals("12:00")){D12=1;}
if(passedTimeArray[j].equals("13:00")){D13=1;}
if(passedTimeArray[j].equals("14:00")){D14=1;}
if(passedTimeArray[j].equals("15:00")){D15=1;}
if(passedTimeArray[j].equals("16:00")){D16=1;}
if(passedTimeArray[j].equals("17:00")){D17=1;}
if(passedTimeArray[j].equals("18:00")){D18=1;}
xData[j]=passedTimeArray[j];

// create regression equation test data of rejected packets
yData[j]=(int)(cData[0]+
(cData[1]*(DoSTDM.testTcp_in[j]))+
(cData[2]*(DoSTDM.testIcmp_in[j]))+
(cData[3]*(DoSTDM.testUdp_in[j]))+
(cData[4]*(DoSTDM.testOth_in[j]))+
(cData[5]*D8)+(cData[6]*D9)+(cData[7]*D10)+(cData[8]*D11)+
(cData[9]*D12)+(cData[10]*D13)+(cData[11]*D14)+(cData[12]*D15)+
(cData[13]*D16)+(cData[14]*D17)+(cData[15]*D18));
zData[j]=DoSTDM.testRejectedArray[j];
}
} // end of for loop

calcMeError = new double[24];
calcMadError = new double[24];
calcMseError = new double[24];

for (int j = 0; j < 24; j++)
{
// collect residuals and calculate errors
DoSTDM.RegResiduals[j]=zData[j]-yData[j];
calcMeError[j] = zData[j]-yData[j];
calcMadError[j] = java.lang.Math.abs(calcMeError[j]);
calcMseError[j] = calcMadError[j] * calcMadError[j];
meSum = meSum + calcMeError[j];
madSum = madSum + calcMadError[j];
mseSum = mseSum + calcMseError[j];
} // end of for loop

ME = meSum/24;
MAD = madSum/24;
MSE = mseSum/24;
RMSE = java.lang.Math.sqrt(MSE);
DoSTDM.RegME=ME;
DoSTDM.RegMAD=MAD;
DoSTDM.RegRMSE=RMSE;

int q=0;
while (q<yData.length
{
sortedYData[q]=yData[q];
sortedZData[q]=zData[q];
q++;
}

```

```

// sort given data array
new Sort(sortedYData);
new Sort(sortedZData);
minY = sortedYData[0];
maxY = sortedYData[(sortedYData.length)-1];
minZ = sortedZData[0];
maxZ = sortedZData[(sortedZData.length)-1];
if(maxZ>maxY){max=maxZ;}
else {if(maxY>maxZ){max=maxY;}
else {if(maxY==maxZ){max=maxY;}}}
if(minZ>minY){min=minZ;}
else {if(minY>minZ){min=minY;}
else {if(minY==minZ){min=minY;}}}

// end of RegGraph constructor method

// call Graphics Paint Component method to graph the data
public void paintComponent(Graphics g) { }

}// end of class RegGraph

```

Listing A – 24: Java Class Regression Graph (RegGraph.java)

### A.2.2.6.8 Class MLPGraph.Java

```

/**
 * This class extends JPanel and override the paintComponent() method to
 * graph MLP forecasted data.
 */

public class MLPGraph extends JPanel
{

// defining string identification variables for user choices
public static String passedYear;
public static String passedMonth;
public static String passedDay;
public static int passedArraySize;
public static double ME; // ME error rate
public static double MAD; // MAD error rate
public static double MSE; // MSE error rate
public static double RMSE; // RMSE error rate
public static double meSum; // ME error (sum) component
public static double madSum; // MAD error (sum) component
public static double mseSum; // MSE error (sum) component
public static double [] calcMeError; //Calculated Reject array ME error
public static double [] calcMadError; //Calculated Reject array MAD error
public static double [] calcMseError; //Calculated Reject array MSE error

// defining drawing panel x-axis and y-axis virtual minimum and maximum
// dimentions variables
public static int vxmin;
public static int vxmax;
public static int vymin;
public static int vymax;

// defining arrays to hold drawing data within
public static String [] xData = new String [24];
public static int [] yData = new int [24];
public static int [] sortedYData = new int [24];
public static int [] zData = new int [24];
public static int [] sortedZData = new int [24];
public static double [] testArray = new double [24];

```

```

// defining drawing data minimum and maximum
public static int min,max,minY,maxY,minZ,maxZ;
public static int maximumValueOfTheDay;
public static int minimumValueOfTheDay;

/**
 * The MLPGraph() constructor initialize drawing graphics panel
 * components based on passed choices from the IPS applet.
 */

public MLPGraph(int sIndx, int eIndx, int noOfEpochs, int noOfHidden, double
hiddenInputLR, double hiddenOutputLR, String graphYear, String graphMonth,
String graphDay)
{
ME=0.0; // ME error rate
MAD=0.0; // MAD error rate
MSE=0.0; // MSE error rate
RMSE=0.0; // RMSE error rate
meSum=0.0; // ME error (sum) component
madSum=0.0; // MAD error (sum) component
mseSum=0.0; // MSE error (sum) component

// user choices for the drawing
passedYear=graphYear;

passedMonth=graphMonth;
passedDay=graphDay;
passedArraySize= eIndx - sIndx;
testArray=new double [24];
minimumValueOfTheDay=0;
maximumValueOfTheDay=0;

// screen coordinates
vxmin=85;
vxmax=610;
vymin=110;
vymax=250;

//copying data for the day selected as test data to compare results
int b=1;
for (int j = 0; j < 24; j++)
{
xData[j]=Integer.toString(b);
testArray[j]=(double)DoSTDM.testRejectedArray[j];
zData[j]=DoSTDM.testRejectedArray[j];
b++;
}

// create a new instance of MLP forecasting class
MLP myMlp = new MLP(passedArraySize, noOfEpochs, noOfHidden, hiddenInputLR,
hiddenOutputLR);
yData = myMlp.returnForecastedResults();
calcMeError = new double[24];
calcMadError = new double[24];
calcMseError = new double[24];
for (int j = 0; j < 24; j++)
{
DoSTDM.MlpResiduals[j]=zData[j]-yData[j];
calcMeError[j] = zData[j]-yData[j];;
calcMadError[j] = java.lang.Math.abs(calcMeError[j]);
calcMseError[j] = calcMadError[j] * calcMadError[j];
meSum = meSum + calcMeError[j];
madSum = madSum + calcMadError[j];
mseSum = mseSum + calcMseError[j];
}
}

```

```

ME = meSum/24;
MAD = madSum/24;
MSE = mseSum/24;
RMSE = java.lang.Math.sqrt (MSE);
DoSTDM.MlpME=ME;
DoSTDM.MlpMAD=MAD;
DoSTDM.MlpRMSE=RMSE;

int q=0;
while (q<yData.length)
    {
        sortedYData[q]=yData[q];
        sortedZData[q]=zData[q];
        q++;
    }

// selection source applied to every resulted sortedArray[] from
// previous operations to find the minimum and the maximum values
// of the data
new Sort(sortedYData);
new Sort(sortedZData);

minY = sortedYData[0];
maxY = sortedYData[(sortedYData.length)-1];
minZ = sortedZData[0];
maxZ = sortedZData[(sortedZData.length)-1];

if (maxZ>maxY) {max=maxZ;}
else {if (maxY>maxZ) {max=maxY;}
else {if (maxY==maxZ) {max=maxY;}}}
if (minZ>minY) {min=minZ;}
else {if (minY>minZ) {min=minY;}
else {if (minY==minZ) {min=minY;}}}

} // end of MLPGraph constructor method

// call Graphics Paint Component method to graph the data
// public void paintComponent(Graphics g) { }

} // end of class MLPGraph

```

Listing A – 25: Java Class MLP Graph (MLPGraph.java)

### A.2.2.6.9 Class ResidualsGraph.Java

```

/**
 * This class extends JPanel and override the paintComponent() method to
 * be able to draw graphics within a JPanel using Graphics and Graphics2D
 * classes. It can draw the residuals of forecasted data from arrays that
 * been collected and passed on from main class to be draw.
 */

public class ResidualsGraph extends JPanel
{
// defining string identification variables for user choices
public static String passedYear;
public static String passedMonth;
public static String passedDay;
public static int passedArraySize;

```

```

// defining drawing panel x-axis and y-axis virtual minimum and maximum
// dimentions variables
public static int vxmin;
public static int vxmax;
public static int vymin;
public static int vymax;

// defining arrays to hold drawing data within
public static String [] xData = new String [24];
public static int [] rData = new int [24];
public static int [] sortedRData = new int [24];

// defining drawing data minimum and maximum
public static int min,max,minY,maxY,minZ,maxZ,minR,maxR;
public static int maximumValueOfTheDay;
public static int minimumValueOfTheDay;

public ResidualsGraph(int[] residualsArray, String graphYear, String
graphMonth, String graphDay)
{
// user choices for the drawing
passedYear=graphYear;
passedMonth=graphMonth;
passedDay=graphDay;
minimumValueOfTheDay=0;
maximumValueOfTheDay=0;
for(int i=0;i<24;i++) {rData[i]=residualsArray[i];}

// screen coordinates
vxmin=85;
vxmax=610;
vymin=110;
vymax=250;
int q=0;
while (q<rData.length)
{
// assign data to sorted array
xData[q]=Integer.toString(q+1);
sortedRData[q]=rData[q];
q++;
}
// find the minimum and the maximum values of the data using sort class
new Sort(sortedRData);

// assigne minimum and maximum values of the passed data
minR = sortedRData[0];
maxR = sortedRData[(sortedRData.length)-1];
max = maxR-minR;
min = minR;
} // end of main constructor of ResidualsGraph()

// call Graphics Paint Component method to graph the data
public void paintComponent(Graphics g) { }

} // end of class ResidualsGraph

```

Listing A – 26: Java Class Residuals Graph (ResidualsGraph.java)

### A.2.2.6.10 Class CompareGraph.java

```
/**
 * This class extends JPanel and override the paintComponent() method to
 * be able to draw graphics within a JPanel using Graphics and Graphics2D
 * classes. It can draw data from arrays that been collected passed on used
 * selection of the data to be draw.
 */

public class CompareGraph extends JPanel
{
    // defining string identification variables for user choices
    public static String passedDate;
    public static String passedYear;
    public static String passedMonth;
    public static String passedDay;
    public static int passedArraySize;
    public static double ME1; // ME error rate
    public static double MAD1; // MAD error rate
    public static double MSE1; // MSE error rate
    public static double RMSE1; // RMSE error rate
    public static double me1Sum; // ME error (sum) component
    public static double mad1Sum; // MAD error (sum) component
    public static double mse1Sum; // MSE error (sum) component
    public static double ME2; // ME error rate
    public static double MAD2; // MAD error rate
    public static double MSE2; // MSE error rate
    public static double RMSE2; // RMSE error rate
    public static double me2Sum; // ME error (sum) component
    public static double mad2Sum; // MAD error (sum) component
    public static double mse2Sum; // MSE error (sum) component
    public static double ME3; // ME error rate
    public static double MAD3; // MAD error rate
    public static double MSE3; // MSE error rate
    public static double RMSE3; // RMSE error rate
    public static double me3Sum; // ME error (sum) component
    public static double mad3Sum; // MAD error (sum) component
    public static double mse3Sum; // MSE error (sum) component
    public static double [] calc1MeError; // Calculated Reject array ME error
    public static double [] calc1MadError; // Calculated Reject array MAD error
    public static double [] calc1MseError; // Calculated Reject array MSE error
    public static double [] calc2MeError; // Calculated Reject array ME error
    public static double [] calc2MadError; // Calculated Reject array MAD error
    public static double [] calc2MseError; // Calculated Reject array MSE error
    public static double [] calc3MeError; // Calculated Reject array ME error
    public static double [] calc3MadError; // Calculated Reject array MAD error
    public static double [] calc3MseError; // Calculated Reject array MSE error

    // defining drawing panel x-axis and y-axis virtual minimum and maximum
    // dimentions variables
    public static int vxmin;
    public static int vxmax;
    public static int vymin;
    public static int vymax;

    // define data arrays imported from DoSTDM.class
    public static double [] passedTrainingDataArray;
    public static String [] passedDateArray;
    public static String [] passedTimeArray;

    // defining arrays to hold drawing data within
    public static String [] xData = new String [24];
    public static int [] y1Data = new int [24];
    public static int [] sortedY1Data = new int [24];
    public static int [] y2Data = new int [24];
    public static int [] sortedY2Data = new int [24];
}
```

```

public static int [] y3Data = new int [24];
public static int [] sortedY3Data = new int [24];
public static int [] zData = new int [24];
public static int [] sortedZData = new int [24];
public static double [] testArray = new double [24];
public static double [] cData = new double [16];

// defining drawing data minimum and maximum
public static int min, max, minY, maxY, minY1, maxY1, minY2, maxY2, minY3,
maxY3, minZ, maxZ;
public static int maximumValueOfTheDay;
public static int minimumValueOfTheDay;

/**
 * The CompareGraph() constructor initialize drawing graphics panel
 * components based on passed choices from the IPS applet.
 */

public CompareGraph(int sIndx, int eIndx, int noOfEpochs, int noOfHidden,
double hiddenInputLR, double hiddenOutputLR, String graphYear, String
graphMonth, String graphDay)
{
ME1 = 0.0; // ME error rate
MAD1 = 0.0; // MAD error rate
MSE1 = 0.0; // MSE error rate
RMSE1 = 0.0; // RMSE error rate
me1Sum = 0.0; // ME error (sum) component
mad1Sum = 0.0; // MAD error (sum) component
mse1Sum = 0.0; // MSE error (sum) component
ME2 = 0.0; // ME error rate
MAD2 = 0.0; // MAD error rate
MSE2 = 0.0; // MSE error rate
RMSE2 = 0.0; // RMSE error rate
me2Sum = 0.0; // ME error (sum) component
mad2Sum = 0.0; // MAD error (sum) component
mse2Sum = 0.0; // MSE error (sum) component
ME3 = 0.0; // ME error rate
MAD3 = 0.0; // MAD error rate
MSE3 = 0.0; // MSE error rate
RMSE3 = 0.0; // RMSE error rate
me3Sum = 0.0; // ME error (sum) component
mad3Sum = 0.0; // MAD error (sum) component
mse3Sum = 0.0; // MSE error (sum) component

// user choices for the drawing
passedYear = graphYear;
passedMonth = graphMonth;
passedDay = graphDay;
passedDate = passedDay+(passedMonth.substring(0,3))+passedYear;
passedDateArray = new String [24];
passedTimeArray = new String [24];
passedArraySize = eIndx - sIndx;
passedTrainingDataArray = new double [passedArraySize];
testArray = new double [24];
minimumValueOfTheDay = 0;
maximumValueOfTheDay = 0;

for (int k = 0;k<passedArraySize;k++)
    {passedTrainingDataArray[k] = (double)DoSTDM.trainRejectedArray[k];}

for (int k = 0;k<24;k++)
    {passedDateArray[k] = DoSTDM.testDateArray[k];
    passedTimeArray[k] = DoSTDM.testTimeArray[k];}

```

```

// screen coordinates
vxmin = 85;
vxmax = 610;
vymin = 110;
vymax = 250;
int b = 1;
for (int j = 0; j < 24; j++)
{
    xData[j] = Integer.toString(b);
    testArray[j] = (double)DoSTDM.testRejectedArray[j];
    zData[j] = DoSTDM.testRejectedArray[j];
    b++;
}

// create an instance of MLP forecasting class
MLP myMlp = new MLP(passedArraySize, noOfEpochs, noOfHidden, hiddenInputLR,
hiddenOutputLR);
y1Data = myMlp.returnForecastedResults();

// create an instance of Holt-Winter forecasting class
HWF myHwf = new HWF(passedTrainingDataArray, testArray, passedArraySize);
y2Data = myHwf.returnForecastedResults();

// create an instance of Regression forecasting class
REG myREG = new REG(sIndx, eIndx);

// setip rregression dummy variables values
for (int j = 0; j < 24; j++)
{
    int D8 = 0;
    int D9 = 0;
    int D10 = 0;
    int D11 = 0;
    int D12 = 0;
    int D13 = 0;
    int D14 = 0;
    int D15 = 0;
    int D16 = 0;
    int D17 = 0;
    int D18 = 0;
    if((passedDateArray[j]).equals(passedDate))
    {
        if(passedTimeArray[j].equals("8:00")){D8 = 1;}
        if(passedTimeArray[j].equals("9:00")){D9 = 1;}
        if(passedTimeArray[j].equals("10:00")){D10 = 1;}
        if(passedTimeArray[j].equals("11:00")){D11 = 1;}
        if(passedTimeArray[j].equals("12:00")){D12 = 1;}
        if(passedTimeArray[j].equals("13:00")){D13 = 1;}
        if(passedTimeArray[j].equals("14:00")){D14 = 1;}
        if(passedTimeArray[j].equals("15:00")){D15 = 1;}
        if(passedTimeArray[j].equals("16:00")){D16 = 1;}
        if(passedTimeArray[j].equals("17:00")){D17 = 1;}
        if(passedTimeArray[j].equals("18:00")){D18 = 1;}
        y3Data[j] = (int)(cData[0] + (cData[1]*(DoSTDM.testTcp_in[j])) +
(cData[2]*(DoSTDM.testIcmp_in[j])) +
(cData[3]*(DoSTDM.testUdp_in[j])) +
(cData[4]*(DoSTDM.testOth_in[j])) + (cData[5]*D8) + (cData[6]*D9) +
(cData[7]*D10) + (cData[8]*D11) + (cData[9]*D12) + (cData[10]*D13) +
(cData[11]*D14) + (cData[12]*D15) + (cData[13]*D16) +
(cData[14]*D17) + (cData[15]*D18));}

    calc1MeError = new double[24];
    calc1MadError = new double[24];
    calc1MseError = new double[24];
    calc2MeError = new double[24];
    calc2MadError = new double[24];
}

```

```

calc2MseError = new double[24];
calc3MeError = new double[24];
calc3MadError = new double[24];
calc3MseError = new double[24];
for (int j = 0; j < 24; j++)
    {
    DoSTDM.MlpResiduals[j]=zData[j]-y1Data[j];
    calc1MeError[j] = zData[j]-y1Data[j];
    calc1MadError[j] = java.lang.Math.abs(calc1MeError[j]);
    calc1MseError[j] = calc1MadError[j] * calc1MadError[j];
    me1Sum = me1Sum + calc1MeError[j];
    mad1Sum = mad1Sum + calc1MadError[j];
    mse1Sum = mse1Sum + calc1MseError[j];
    }

ME1 = me1Sum/24;
MAD1 = mad1Sum/24;
MSE1 = mse1Sum/24;
RMSE1 = java.lang.Math.sqrt(MSE1);
DoSTDM.MlpME=ME1;
DoSTDM.MlpMAD=MAD1;
DoSTDM.MlpRMSE=RMSE1;

for (int j = 0; j < 24; j++)
    {
    DoSTDM.HWRResiduals[j]=zData[j] - y2Data[j];
    calc2MeError[j] = zData[j] - y2Data[j];
    calc2MadError[j] = java.lang.Math.abs(calc2MeError[j]);
    calc2MseError[j] = calc2MadError[j] * calc2MadError[j];
    me2Sum = me2Sum + calc2MeError[j];
    mad2Sum = mad2Sum + calc2MadError[j];
    mse2Sum = mse2Sum + calc2MseError[j];
    }

ME2 = me2Sum / 24;
MAD2 = mad2Sum / 24;
MSE2 = mse2Sum / 24;
RMSE2 = java.lang.Math.sqrt(MSE2);
DoSTDM.HWME=ME2;
DoSTDM.HWMAD=MAD2;
DoSTDM.HWRMSE=RMSE2;

for (int j = 0; j < 24; j++)
    {DoSTDM.RegResiduals[j] = zData[j] - y3Data[j];
    calc3MeError[j] = zData[j] - y3Data[j];
    calc3MadError[j] = java.lang.Math.abs(calc3MeError[j]);
    calc3MseError[j] = calc3MadError[j] * calc3MadError[j];
    me3Sum = me3Sum + calc3MeError[j];
    mad3Sum = mad3Sum + calc3MadError[j];
    mse3Sum = mse3Sum + calc3MseError[j];}

ME3 = me3Sum / 24;
MAD3 = mad3Sum / 24;
MSE3 = mse3Sum / 24;
RMSE3 = java.lang.Math.sqrt(MSE3);
DoSTDM.RegME=ME3;
DoSTDM.RegMAD=MAD3;
DoSTDM.RegRMSE=RMSE3;
int q = 0;
while (q<zData.length)
    {
    // assign data to sorted array to hold data values to be sorted
    sortedY1Data[q] = y1Data[q];
    sortedY2Data[q] = y2Data[q];
    sortedY3Data[q] = y3Data[q];
    sortedZData[q] = zData[q];
    q++;}

```

```

// selection source applied to every resulted sortedArray[] from
// previous operations to find the minimum and the maximum values
// of the data
new Sort(sortedY1Data);
new Sort(sortedY2Data);
new Sort(sortedY3Data);
new Sort(sortedZData);

// assigne minimum and maximum values of the passed data and use this
// at a later stage to graph the selected data using the java
// paintcomponenet() method.
minY1 = sortedY1Data[0];
minY2 = sortedY2Data[0];
minY3 = sortedY3Data[0];
maxY1 = sortedY1Data[(sortedY1Data.length) - 1];
maxY2 = sortedY2Data[(sortedY2Data.length) - 1];
maxY3 = sortedY3Data[(sortedY3Data.length) - 1];
int [] minis = new int [3];
int [] maxis = new int [3];
minis[0] = minY1;
maxis[0] = maxY1;
minis[1] = minY2;
maxis[1] = maxY2;
minis[2] = minY3;
maxis[2] = maxY3;
new Sort(minis);
new Sort(maxis);
minY = minis[0];
maxY = maxis[(maxis.length)-1];
minZ = sortedZData[0];
maxZ = sortedZData[(sortedZData.length)-1];
if(maxZ>maxY){ max = maxZ; }
else { if(maxY > maxZ){max = maxY; }
else { if(maxY == maxZ){max = maxY; }}}
if(minZ < minY){ min = minZ; }
else {if(minY < minZ){ min = minY;}
else {if(minY == minZ){ min = minY;}}}
max=max-min;
} // end of CompareGraph()

/**
 * The paintComponent method graph the given data using Graphics and
 * Graphics2D classes to override the painComponent() method provided by
 * standard java classes. It can graph bars, pies and line charts based on
 * user selection from DoSTDM Applet.
 */
public void paintComponent(Graphics g)
{ // set graph background
g.setColor(new Color(240,248,255));
g.fillRect(vxmin+10, vymin-80, vxmax-82, vymax-35);
g.setColor(Color.black);
g.drawLine(10,0,658,0);
g.drawLine(12,2,656,2);
g.drawLine(10,281,658,281);
g.drawLine(12,279,656,279);
g.drawLine(10,0,10,281);
g.drawLine(12,0,12,279);
g.drawLine(658,0,658,281);
g.drawLine(656,2,656,279);
g.setColor(Color.green);
g.drawLine(11,280,657,280);
g.drawLine(11,1,657,1);
g.drawLine(11,1,11,280);
g.drawLine(657,1,657,280);
g.setColor(Color.black);

```

```

// x & y axis
g.drawRect(vxmin+10, vymin-80, vxmax-81, vymax-35);

// x & y axis labels
g.setColor(Color.RED);
g.setFont(new Font("Arial", Font.BOLD, 12));
g.drawString("HOURS", vxmin+255, vymax+24);
g.setFont(new Font("Arial", Font.BOLD, 10));
Graphics2D g2d = (Graphics2D)g;
int xscDisplay=0;

// x-scale
for (int xsc=23;xsc<=552;xsc+=23)
{
    // draw data x-axis line
    g.setColor(Color.RED);
    g.drawLine(vxmin+xsc-13, vymax-5, vxmin+xsc-13, vymax);

    // setup x-axis labels gap
    xscDisplay=(int)xsc/23;
    String hrPoint=xData[xscDisplay-1];
    int textLength = hrPoint.length();

    // draw hours labels
    g.setColor(Color.BLACK);
    if(textLength==1)
    {
        g.drawString(hrPoint, vxmin+xsc-16, vymax+10);
    }

    if(textLength==2)
    {
        g.drawString(hrPoint, vxmin+xsc-19, vymax+10);
    }

    if(xscDisplay<=23)
    {
        // mlp data
        g.setColor(Color.BLUE);
        g.drawLine(vxmin+xsc-13, vymax-((y1Data[xscDisplay-1]-min)*200/max)-5, vxmin+xsc+10, vymax-((y1Data[xscDisplay]-min)*200/max)-5);
        g.fill3DRect(vxmin+xsc-15, vymax-((y1Data[xscDisplay-1]-min)*200/max)-7, 5, 5, true);

        // hw data
        g.setColor(new Color(52,128,23));
        g.drawLine(vxmin+xsc-13, vymax-((y2Data[xscDisplay-1]-min)*200/max)-5, vxmin+xsc+10, vymax-((y2Data[xscDisplay]-min)*200/max)-5);
        g.fill3DRect(vxmin+xsc-15, vymax-((y2Data[xscDisplay-1]-min)*200/max)-7, 5, 5, true);

        // regression data
        g.setColor(Color.BLACK);
        g.drawLine(vxmin+xsc-13, vymax-((y3Data[xscDisplay-1]-min)*200/max)-5, vxmin+xsc+10, vymax-((y3Data[xscDisplay]-min)*200/max)-5);
        g.fill3DRect(vxmin+xsc-15, vymax-((y3Data[xscDisplay-1]-min)*200/max)-7, 5, 5, true);

        // real data
        g.setColor(Color.RED);
        g.drawLine(vxmin+xsc-13, vymax-((zData[xscDisplay-1]-min)*200/max)-5, vxmin+xsc+10, vymax-((zData[xscDisplay]-min)*200/max)-5);
        g.fill3DRect(vxmin+xsc-15, vymax-((zData[xscDisplay-1]-min)*200/max)-7, 5, 5, true);}
}

```

```

    if(xscDisplay==23)
    {

        // mlp data
        g.setColor(Color.BLUE);
        g.fill3DRect(vxmin+xsc+8, vymax-((y1Data[xscDisplay]-min)*200/max)-
        7,5,5, true);

        // hw data
        g.setColor(new Color(52,128,23));
        g.fill3DRect(vxmin+xsc+8, vymax-((y2Data[xscDisplay]-min)*200/max)-
        7,5,5, true);

        // regression data
        g.setColor(Color.BLACK);
        g.fill3DRect(vxmin+xsc+8, vymax-((y3Data[xscDisplay]-min)*200/max)-
        7,5,5, true);

        // real data
        g.setColor(Color.RED);
        g.fill3DRect(vxmin+xsc+8, vymax-((zData[xscDisplay]-min)*200/max)-
        7,5,5, true);
    }

g2d.setColor (Color.black);
g2d.setStroke (new BasicStroke(1f, BasicStroke.CAP_ROUND,
BasicStroke.JOIN_ROUND, 1f, new float[] {2f}, 0f));
g2d.drawLine(vxmin+xsc-13, vymax-5,vxmin+xsc-13,vymin-80);
g2d.setStroke (new BasicStroke());
}

int yscDisplay=0;
if(max>10)
{
    yscDisplay=Math.round(max/12);
    for (int ysc=min;(ysc-min)<=max;ysc+=yscDisplay)
    {
        g.setColor(Color.red);
        g.drawLine(vxmin+5, vymax-((ysc-min)*200/max)-5,vxmin+10,vymax-((ysc-
        min)*200/max)-5);
        g.setColor(Color.black);

        int absYsc=Math.abs(ysc);
        if(absYsc>=0 && absYsc<10)
        {
            g.drawString(Integer.toString(ysc), vxmin-6, vymax-((ysc-
            min)*200/max));
        }
        else
        {
            if(absYsc>=10 && absYsc<100)
            {
                g.drawString(Integer.toString(ysc), vxmin-12, vymax-((ysc-
                min)*200/max));
            }
            else
            {
                if(absYsc>=100 && absYsc<1000)
                {
                    g.drawString(Integer.toString(ysc), vxmin-18, vymax-
                    ((ysc-min)*200/max));
                }
                else
                {
                    if(absYsc>=1000 && absYsc<10000)

```

```

        {
            g.drawString(Integer.toString(yisc), vxmin-24,
                vymax-((yisc-min)*200/max));
        }
        else
        {
            if(absYisc>=10000 && absYisc<100000)
            {
                g.drawString(Integer.toString(yisc), vxmin-
                    30, vymax-((yisc-min)*200/max));
            }
            else
            {
                if(absYisc>=100000 && absYisc<1000000)
                {
                    g.drawString(Integer.toString(yisc),
                        vxmin-36, vymax-((yisc-min)*200/max));
                }
                else
                {
                    g.drawString(Integer.toString(yisc), vxmin-
                        42, vymax-((yisc-min)*200/max));
                }
            }
        }
    }

if((yisc-min)<=max)
{
    g2d.setColor (Color.black);
    g2d.setStroke (new BasicStroke(1f, BasicStroke.CAP_ROUND,
        BasicStroke.JOIN_ROUND, 1f, new float[] {2f}, 0f));
    g2d.drawLine(vxmin+10, vymax-((yisc-min)*200/max)-5, vxmax+15, vymax-((yisc-
        min)*200/max)-5);
    g2d.setStroke (new BasicStroke());
}
else
{
    if(max==0 || max<=10)
    {
        for (int yisc=min;yisc<=10;yisc++)
        {
            g.setColor(Color.red);
            g.drawLine(vxmin+5, vymax-((yisc-min)*200/10)-5, vxmin+10, vymax-
                ((yisc-min)*200/10)-5);
            g.setColor(Color.black);
            if(yisc>=0 && yisc<10)
            {
                g.drawString(Integer.toString(yisc), vxmin-6, vymax-((yisc-
                    min)*200/10));
            }
            else {g.drawString(Integer.toString(yisc), vxmin-12, vymax-
                ((yisc-min)*200/10));}
            g2d.setColor (Color.black);
            g2d.setStroke (new BasicStroke(1f, BasicStroke.CAP_ROUND,
                BasicStroke.JOIN_ROUND, 1f, new float[] {2f}, 0f));
            g2d.drawLine(vxmin+10, vymax-(yisc*200/10)-5, vxmax+15, vymax-
                ((yisc-min)*200/10)-5);
            g2d.setStroke (new BasicStroke());
            if(max == 0){
                g2d.setColor (Color.RED);
                g2d.setFont(new Font("Arial", Font.BOLD, 45));
                g2d.drawString("Zero Packets", Math.round((vxmin+vxmax)/3.2),
                    Math.round((vymin+vymax)/2.4));
                g2d.setFont(new Font("Arial", Font.BOLD, 10));
            }
        }
    }
    g.setColor (Color.red);
    g.setFont(new Font("Arial", Font.BOLD, 12));
    g.drawString("P", vxmin-59, vymin-10);
    g.drawString("A", vxmin-59, vymin+5);
    g.drawString("C", vxmin-59, vymin+20);
}

```

```

g.drawString("K", vxmin-59, vymin+35);
g.drawString("E", vxmin-59, vymin+50);
g.drawString("T", vxmin-59, vymin+65);
g.drawString("S", vxmin-59, vymin+80);
g.setColor(Color.BLUE);
g.fill3DRect(vxmin, vymax+50, 10, 10, true);
g.drawString("NN-MLP", vxmin+15, vymax+60);
g.setColor(new Color(52, 128, 23));
g.fill3DRect(vxmin+150, vymax+50, 10, 10, true);
g.drawString("Holt-Winter", vxmin+165, vymax+60);
g.setColor(Color.BLACK);
g.fill3DRect(vxmin+300, vymax+50, 10, 10, true);
g.drawString("Regression", vxmin+315, vymax+60);
g.setColor(Color.red);
g.fill3DRect(vxmin+450, vymax+50, 10, 10, true);
g.drawString("Real Data", vxmin+465, vymax+60);
g.setColor(Color.BLACK);
g.drawRect(vxmin, vymax+50, 10, 10);
g.drawRect(vxmin+150, vymax+50, 10, 10);
g.drawRect(vxmin+300, vymax+50, 10, 10);
g.drawRect(vxmin+450, vymax+50, 10, 10);

//draw graph title
g.setColor(Color.black);
g.setFont(new Font("Arial", Font.BOLD, 15));

String strTitle="Real vs. Forecast / Hour for "+passedMonth+" "+passedDay+",
"+passedYear;
int strLength=strTitle.length();
int strPosition=vxmin+10+Math.round(((vxmax-82)-(vxmin+10))/2)-
((strLength*5)/2));
g.drawString(strTitle, strPosition, vymin-90);
} // end of method paintComponent

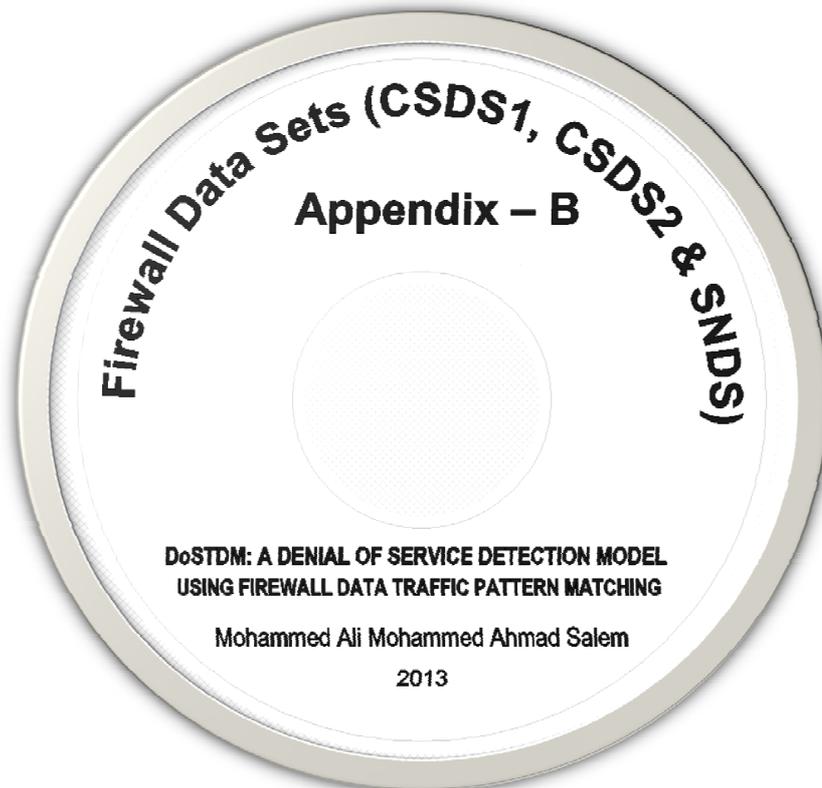
} //end of class CompareGraph

```

Listing A – 27: Java Class Compare Graph (CompareGraph.java)

## **APPENDIX – B: Processed Firewall Logs**

The attached CD contains the raw data, pre-processed data and all the data pre-processing scripts used to extract network protocol statistics from raw data of CSDS1, CSDS2 and SNDS to complete DoSTDM model design and testing (as listed in Chapter 4).



## **APPENDIX – C: Compiled DoSTDM Java Applet Classes**

The attached CD contains DoSTDM applet program interface implemented in a compiled Java class format. The applet uses the processed data of CSDS1, CSDS2 and SNDS to examine Holt-Winter, Regression and MLP neural network methods as used in this research thesis (as listed in Chapter 6).

Note: To start the DoSTDM Java applet run the file called DoSTDM.html

