

This item was submitted to [Loughborough's Research Repository](#) by the author.
Items in Figshare are protected by copyright, with all rights reserved, unless otherwise indicated.

Detecting TCP-based applications using packet size distributions

PLEASE CITE THE PUBLISHED VERSION

PUBLISHER

© Bo Li

LICENCE

CC BY-NC-ND 4.0

REPOSITORY RECORD

Li, Bo. 2019. "Detecting Tcp-based Applications Using Packet Size Distributions". figshare.
<https://hdl.handle.net/2134/15329>.

This item was submitted to Loughborough University as a PhD thesis by the author and is made available in the Institutional Repository (<https://dspace.lboro.ac.uk/>) under the following Creative Commons Licence conditions.



For the full text of this licence, please go to:
<http://creativecommons.org/licenses/by-nc-nd/2.5/>

University Library

Author/Filing Title *LI, B.*

Class Mark *T*

Please note that fines are charged on ALL
overdue items.

--	--	--


0403694787



Bo Li

Detecting TCP-based applications using packet size distributions

2008

 Loughborough University Pilkington Library
Date 25/2/09
Class T
Acc No. 0408694787

Abstract

To know what applications are currently in operation across modern packet based communication networks such as the Internet is always attractive to network administrators, network service providers and security systems. The availability of this information can contribute to preventing improper network use, which may include illegal activities, consume a large amount of bandwidth, or may cause security problems or break policies in network usage. In addition, using this information, the network may be able to establish enhanced environments for the applications, which are in use.

Various techniques exist to perform network application detection. However difficulty is encountered where the traditional techniques will fail in their task. For example, if the application uses non-registered port numbers, the capture of certain specific packets is impossible or the data portion of at least some of the packets is unavailable due to encryption or processing overload.

In this Thesis an alternative approach to application detection, using packet size distributions, is applied to TCP applications. This statistical property of the traffic stream is found to be unique to certain kinds of network applications. The detection can be achieved by simply comparing this “fingerprint” with pre-evaluated samples stored in a database. Previous work has shown that packet size distributions can successfully identify many types of UDP application.

This Thesis suggests that for those TCP-based network applications that do not use the Nagle Algorithm, the detection mechanism, which had been proved to be successful for UDP-based applications, could be also adopted without any modification. For Nagle-based applications, the situation becomes more complicated, however, with some pre-computation, successful detection can be achieved as well. A prototype detector implementing the suggested approaches has been designed in order to test the feasibility and performance of the approach proposed. The tests carried out upon this prototype platform indicate that the method is universally suitable for several of distributions and give out satisfied detection success ratios.

Acknowledgments

First of all I would like to thank my supervisor, Prof. David Parish, for his continuous support during my PhD study. David always gave good advices and encouragement during my least inspired times. I have leant a lot from him, ranging from general research skills to specific knowledge.

I also gratefully appreciate help provided by researchers in the High Speed Network group within the department, namely Dr. Mark Sandford, Mr. Peter Sandford, Dr. Mark Withal and Dr. Shiru De Silva. I had many discussions with Mark S and Peter at the beginning of my research, which gave invaluable input into my research. Thank you to Mark W and Shiru for the ideas and supports in using the test platform.

Special thanks go to my friend and colleague Konstantinos Kyriakopoulos. Although we were working on completely different research topics, he was always there to listen and give his opinions. Thank you for being with me over the last two years.

Furthermore, I would like to thank my parents, for giving me life, educating me, and encouraging me to pursue my interests, and supported me unconditionally. I would like to dedicate this Thesis to them.

Publication

Bo, L., Parish, D.J., Sandford, J.M. and Sandford, P., "Using TCP Packet Size Distributions for Application Detection", PGNet 2006 Proceedings, Merabiti, Pereira and Abuelma'atti, Liverpool John Moores University, PGNet, Liverpool John Moores University, June 2006, pp 184-189, ISSN 1 9025 6013 9

Table of Contents

1	1
1.1	Introduction 1
1.2	Network Management 2
1.2.1	The Importance of Network Management..... 2
1.2.2	Quality of Service..... 3
1.3	Network Security 4
1.3.1	The Importance of Network Security 4
1.3.2	Network Threats 5
1.4	Contribution to Knowledge 7
1.5	Outline of Chapters 7
2	9
2.1	Introduction 9
2.2	TCP/IP Reference Model 10
2.2.1	Seven Layer OSI Model 10
2.2.2	Transport Layer Protocols 11
2.3	Current Application Detection Methods 13
2.3.1	Port Number 13
2.3.2	Packet Classification 15
2.3.3	Stateful Inspection 16
2.3.4	Deep Packet Inspection 17
2.4	Statistical Methods 18
2.4.1	The Fragility of Current Detection Techniques..... 18
2.4.2	Rationale of the New Statistical Approach..... 19
2.4.3	Potential Effects Caused by TCP on Packet Size Distribution 20
2.4.4	Statistical Chi-square Test..... 20
2.5	Summary 23
3	24
3.1	Introduction 24
3.2	The Experiment System Architecture 25
3.2.1	Loaded Network Emulator (LNE)..... 25
3.2.2	The Experimental Architecture 25
3.3	The Consistency of the Packet Size Distribution as an Application Detecting Signature 26

3.3.1	The Effect of Application Settings/Scenarios on the Packet Size Distributions	27
3.3.2	The Effect of Network Load on The Packet Size Distributions	39
3.4	The Uniqueness Tests on the TCP Packet Size Distributions	51
3.5	Summary	54
4	55
4.1	Introduction	55
4.2	Methods of Aggregation Application Detection	56
4.2.1	Methodology	56
4.2.2	Selection of Referring Bins	65
4.2.3	The Selection of Aggregation Method	66
4.3	Verification	67
4.3.1	The Generation of Sample Original Distribution for Virtual Applications	67
4.3.2	Nagle-Based Application Emulator	68
4.3.3	Tests of Real Nagle-Based Applications	69
4.3.4	Tests of Virtual Nagle-Based Applications	85
4.3.5	Tests on Other Virtual Applications	100
4.4	Summary	102
5	103
5.1	Introduction	103
5.2	Operation Overview	104
5.3	Thread LoadPackets()	107
5.4	Thread Detect()	110
5.5	Thread SanpUI()	113
5.6	Thread BuildStoredProfiles()	116
5.7	Summary	119
6	120
6.1	Introduction	120
6.2	The Settings of the Detector	121
6.2.1	Non-Nagle Applications	121
6.2.2	Nagle-Based Applications	127
6.2.3	Virtual Nagle-Based Applications	131
6.3	Application Detector Performance	133
6.3.1	Non-Nagle Applications	134
6.3.2	Nagle-Based Applications	136
6.3.3	Virtual Nagle-Based Applications	138
6.3.4	Virtual Applications with Large Packet Sizes	141
6.4	Simultaneously Running Application	141

6.5 **Summary** 142

7 145

7.1 **Introduction** 145

7.2 **Conclusions** 145

7.3 **Future Works**..... 148

7.4 **Contribution Remarks** 149

List of Figures

<i>Figure 2.1 OSI and TCP/IP reference mode and associated protocols</i>	10
<i>Figure 2.2 Typical χ^2 distribution profile</i>	22
<i>Figure 3.1 Experimental System Architecture</i>	26
<i>Figure 3.2 Packet size distribution for Crim-Sky (svr-cli) 2 players</i>	27
<i>Figure 3.3 Packet size distribution for Crim-Sky (cli-svr) 2 players</i>	28
<i>Figure 3.4 Packet size distribution for Crim-Sky (svr-cli) Free-Fly, 4 players</i>	28
<i>Figure 3.5 Packet size distribution for Crim-Sky (svr-cli) Group-combat, 4 players</i>	28
<i>Figure 3.6 Packet size distribution for WarCraft III (svr-cli) overall</i>	30
<i>Figure 3.7 Packet size distribution for WarCraft III (cli-svr) overall</i>	30
<i>Figure 3.8 Packet size distribution for WarCraft III (svr-cli) high intensity fight</i>	31
<i>Figure 3.9 Packet size distribution for WarCraft III (cli-svr) high intensity fight</i>	31
<i>Figure 3.10 Packet size distributions for RealPlayer while playing files at different bit rates</i>	32
<i>Figure 3.11 Packet size distributions for HTTP</i>	34
<i>Figure 3.12 Dumped file details for HTTP</i>	34
<i>Figure 3.13 Packet size distribution for FTP control-command sessions</i>	36
<i>Figure 3.14 Packet size distribution for SMTP control-command sessions</i>	36
<i>Figure 3.15 Packet size distributions for FTP data transmission session</i>	37
<i>Figure 3.16 Packet size distributions for SMTP data transmission session</i>	37
<i>Figure 3.17 Packet size distribution for SSH</i>	38
<i>Figure 3.18 Packet size distribution for SSH</i>	39
<i>Figure 3.19 Packet size distribution for Crim-Sky under low-load network condition</i>	40
<i>Figure 3.20 Packet size distribution for Crim-Sky under moderate-load network condition</i>	41
<i>Figure 3.21 Packet size distributions for Crim-Sky under heavy-load network condition</i>	42
<i>Figure 3.22 Packet size distributions for WarCraft III under network condition of 100 ms delay</i>	43
<i>Figure 3.23 Dumped file details of the WarCraft III under ideal network condition</i>	44
<i>Figure 3.24 Packet size distributions for WarCraft III under network condition of 200 ms delay</i>	45
<i>Figure 3.25 Packet size distributions for WarCraft III under network condition of 3 percent loss</i>	46
<i>Figure 3.26 Packet size distributions for RealPlayer under low-load network condition (600 kbps bit rate)</i>	47
<i>Figure 3.27 Packet size distributions for RealPlayer under heavy-load network</i>	48
<i>Figure 3.28 Packet size distributions for HTTP under network condition of 200 ms delay</i>	48
<i>Figure 3.29 Packet size distributions under network condition of 100 ms delay</i>	49
<i>Figure 3.30 Packet size distributions for SSH under worsening network condition</i>	50
<i>Figure 3.31 Chi-square Values of Crim-Sky against database (Cli-Srv)</i>	52
<i>Figure 3.32 Chi-square Values of WarCraft III against database (Cli-Srv)</i>	53

Figure 4.1 Packet size distribution for WarCraft III under heavy-load network condition.....	60
Figure 4.2 A sample original distribution, which needs to use the second method.....	66
Figure 4.3 Parameters to describe a distribution.....	68
Figure 4.4 Original packet size distribution of WarCraft III (cli to srv)	70
Figure 4.5 Original packet size distribution of WarCraft III (srv to cli)	70
Figure 4.6 Captured aggregated distribution for WarCraft III under the network condition of 150ms delay (srv to cli)	71

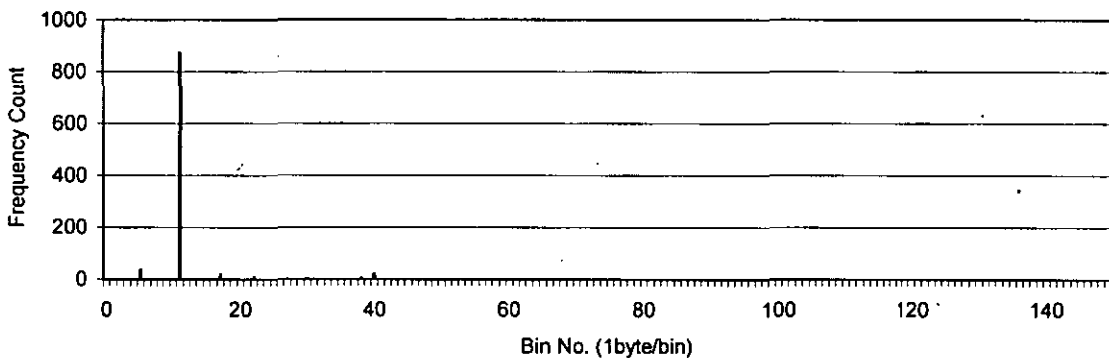


Figure 4.7 Theoretical aggregated distribution calculated from the original distribution of WarCraft III (cli to srv)	71
Figure 4.8 Theoretical aggregated distribution calculated from the original distribution of another virtual application (cli to srv)	71
Figure 4.9 Captured aggregated distribution for WarCraft III under the network condition of 150ms delay (srv to cli)	72
Figure 4.10 Calculated theoretical aggregated distribution from the original distribution of WarCraft III (cli to srv)	72
Figure 4.11 Theoretical aggregated distribution calculated from the original distribution of another virtual application (cli to srv)	72
Figure 4.12 Summarization of Chi-square tests for identification of WarCraft III	74
Figure 4.13 Captured aggregated distribution for WarCraft III under the network condition of 300ms fixed delay (srv to cli)	75
Figure 4.14 Theoretical aggregated distribution calculated from the original distribution of WarCraft III (srv to cli)	75

Figure 4.15 Theoretical aggregated distribution calculated from the original distribution of another virtual application (srv to cli)	75
Figure 4.16 Captured aggregated distribution for WarCraft III under the network condition of 150ms delay (srv to cli)	76
Figure 4.17 Theoretical aggregated distribution calculated from the original distribution of WarCraft III (srv to cli)	76
Figure 4.18 Theoretical aggregated distribution calculated from the original distribution of another virtual application (srv to cli)	76
Figure 4.19 Summarization of Chi-square tests for identification of WarCraft III	77
Figure 4.20 Captured aggregated distribution of WarCraft III under the network condition of 300ms delay (srv to cli)	78
Figure 4.21 Theoretical aggregated distribution calculated from the original distribution of WarCraft III (cli to srv)	78
Figure 4.22 Theoretical aggregated distribution calculated from the original distribution of another virtual application (cli to srv)	79
Figure 4.23 Captured aggregated distribution of WarCraft III under the network condition of jittered delay (srv to cli)	79
Figure 4.24 Theoretical aggregated distribution calculated from the original distribution of WarCraft III (cli to srv)	79
Figure 4.25 Theoretical aggregated distribution calculated from the original distribution of another virtual application (cli to srv)	80
Figure 4.26 Summarization of Chi-square tests for identification of WarCraft III under network condition of jittered delay	81
Figure 4.27 Original packet size distribution of SSH-Client III	82
Figure 4.28 Captured aggregated distribution for SSH-Client under the network condition of 300 ms fixed delay	82
Figure 4.29 Theoretical original distribution calculated from the original distribution of SSH-Client	82
Figure 4.30 Summarization of Chi-square tests for identification of SSH-Client under network condition of 300ms fixed delay	83
Figure 4.31 Captured aggregated distribution for SSH-Client under the network condition of jittered delay	83
Figure 4.32 Theoretical original distribution calculated from the original distribution of SSH-Client	84
Figure 4.33 Summarization of Chi-square tests for identification of SSH-Client under network condition of jittered delay	84
Figure 4.34 Parameters and profile of original distribution of virtual application 1	86
Figure 4.35 Captured aggregated distribution for virtual application 1 under the network condition of 300 ms fixed delay	86
Figure 4.36 $Dist^{temp}$ after $Dist^{4th-theory}$ and $Dist^{3rd-theory}$ were removed	87
Figure 4.37 2 nd order distribution of virtual application 1	87
Figure 4.38 Original distribution of virtual application 18	88

Figure 4.39 Theoretical aggregated distribution extracted using the original distribution of virtual application 18.....	88
Figure 4.40 Theoretical aggregated distribution calculated from the original distribution for virtual application 25.....	89
Figure 4.41 Captured aggregated distribution for virtual application 1 under the network condition of jittered delay.....	89
Figure 4.42 Theoretical original distribution extracted using the original distribution for virtual application 1.....	90
Figure 4.43 Summarizations of Chi-square tests of identifying Virtual Application 1 under network condition of jittered delay.....	90
Figure 4.44 Original packet size distribution of virtual application 2	91
Figure 4.45 Distribution profiles for different orders of virtual application 2.....	92
Figure 4.46 Captured aggregated distribution for virtual application 2 under the network condition of 300 ms fixed delay.....	92
Figure 4.47. $Dist^{temp}$ after $Dist^{4th-theory}$ and $Dist^{3rd-theory}$ were removed.....	93
Figure 4.48 2 nd order distribution of virtual application 2.....	93
Figure 4.49 Captured aggregated distribution for Virtual Application 2 under the network condition of jittered delay.....	94
Figure 4.50 Theoretical original distribution extracted using the original distribution of virtual application 2.....	94
Figure 4.51 Original packet size distribution of virtual application 3	96
Figure 4.52 Distribution profiles of different orders for virtual application 3.....	96
Figure 4.53 Captured aggregated distribution for virtual application 3 under the network condition of 100 ms fixed delay.....	97
Figure 4.54 Theoretical aggregated distribution calculated from the original distribution for virtual application 3.....	97
Figure 4.55 Captured aggregated distribution for virtual application 3 under the network condition of jittered delay.....	98
Figure 4.56 Theoretical aggregated distribution calculated from the original distribution for virtual application 3.....	98
Figure 4.57 Theoretical original distribution extracted using the original distribution of virtual application 3.....	99
Figure 4.58 Captured aggregated distribution for virtual application 3 under a network condition of 300 ms fixed delay.....	100
Figure 4.59 Chi-square test summary for all tested virtual applications	101
Figure 5.1 Flow diagram of the prototype detector.....	106
Figure 5.2 Relationship diagram of classes in thread LoadPackets()	108
Figure 5.3 Flow diagram for thread LoadPackets().....	109
Figure 5.4 Relationship diagram of classes in thread Detect()	111
Figure 5.5 Flow diagram for thread Detect().....	113

Figure 5.6 Flow diagram for thread SnapUi()	114
Figure 5.7 Flow diagram for thread BuildStoredProfiles()	118
Figure 6.1 Chi-square results of Nadar's distribution profiles over 15 seconds sampling periods	122
Figure 6.2 Chi-square results of the Nadar distribution profiles over 30 seconds sampling periods	122
Figure 6.3 Chi-square results of Nadar's distribution profiles over 45 seconds sampling periods	123
Figure 6.4 Chi-square results of Diablos' distribution profiles over 15 seconds sampling periods	124
Figure 6.5 Chi-square results of Diablo's distribution profiles over 30 seconds sampling periods	124
Figure 6.6 Chi-square results of Diablo's distribution profiles over 45 seconds sampling periods	125
Figure 6.7 Acceptance ratios for all applications over 15 second sampling periods	125
Figure 6.8 Acceptance ratios for all applications over 30 second sampling periods	126
Figure 6.9 Acceptance ratios for all applications over 45 second sampling periods	126
Figure 6.10 Chi-square results for WarCraft III's distribution profiles over 400 packets	128
Figure 6.11 Chi-square results for WarCraft III's distribution profiles over 600 packets	128
Figure 6.12 Chi-square results for WarCraft III's distribution profiles over 800 packets	129
Figure 6.13 Chi-square results of WarCraft III's distribution profiles over 400 packets	130
Figure 6.14 Chi-square results of WarCraft III's distribution profiles over 600 packets	130
Figure 6.15 Chi-square results of WarCraft III's distribution profiles over 800 packets	130
Figure 6.16 Chi-square results for Virtual Nagle-based applications for different numbers of sample packets	131
Figure 6.17 Acceptance ratios for all virtual Nagle-based applications under a jittered delay network condition over 600 packets	132
Figure 6.18 Acceptance ratios for all virtual Nagle-based applications under a 300ms fixed delay network condition over 600 packets	132
Figure 6.19 Acceptance ratios for all virtual Nagle-based applications under a jittered delay network condition over 800 packets	133
Figure 6.20 Acceptance ratios for all virtual Nagle-based applications under a 300 ms fixed delay network condition over 800 packets	133
Figure 6.21 Success ratios for all non-Nagle applications under ideal network condition	135
Figure 6.22 Success ratios for all Non-Nagle applications under loaded network condition	136
Figure 6.23 Success ratios for real Nagle-based applications under ideal network condition	137
Figure 6.24 Success ratios for real Nagle-based applications under loaded network condition	137
Figure 6.25 Success ratios for all virtual Nagle-based applications under loaded network condition	138
Figure 6.26 A simple aggregated distribution profile	139
Figure 6.27 Two simple original distribution profiles	139
Figure 6.28 Success ratios for all virtual Nagle-based applications under loaded network condition	140
Figure 6.29 Success ratios for Nagle-based applications with large packets under low-load network condition	141

List of Tables

<i>Table 1.1 Rationales for Network Management</i>	3
<i>Table 1.2 Security Threats and Access Techniques</i>	6
<i>Table 3.1 Chi-square analysis results for Crim-Sky under different gaming conditions</i>	29
<i>Table 3.2 Chi-square analysis results for WarCraft III under different gaming conditions</i>	31
<i>Table 3.3 Chi-square analysis results for RealPlayer for different bit-rates</i>	33
<i>Table 3.4 Chi-square analysis results for Crim-Sky under different network conditions</i>	42
<i>Table 3.5 Chi-square analysis results for WarCraft III under different network conditions</i>	46
<i>Table 3.6 The stored profile numbers and corresponding application names</i>	52
<i>Table 3.7 Results of Chi-squared tests between different application traces</i>	53
<i>Table 4.1 Original packet size distribution values of first 28 bins for WarCraft III</i>	60
<i>Table 4.2 Chosen parameters for virtual application distributions</i>	68
<i>Table 4.3 A sample of ordered packet series table</i>	69
<i>Table 5.1 Detector process summary</i>	105
<i>Table 5.2 User Interface Information</i>	115
<i>Table 6.1 Success ratios for simultaneously running applications</i>	142
<i>Table 6.2 Parameters selected for the prototype detector</i>	143
<i>Table 6.3 Success ratios for strictly defined "success"</i>	144
<i>Table 6.4 Success ratios for less strictly defined "success"</i>	144

List of Symbols

ACK	Acknowledgement
API	Application Programming Interface
ATM	Asynchronous Transfer Mode
DNS	Domain Name Service
DoS	Denial of Service
DPI	Deep Packet Inspection
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
IANA	Internet Assigned Numbers Authority
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISP	Internet Service Provider
JDBC	Java DataBase Connectivity
JSDK	Java Servlet Development Kits
JVM	Java Virtual Machine
LNE	Loaded Network Emulator
MIB	Management Information Base
MSS	maximum segment size
MTU	Maximum Transmission Unit
NIC	Network Interface Card
OSI	Open Systems Interconnection
PC	Personal Computer
PLR	Packet Loss Rate
PKR	Packet Error Rate
QoS	Quality of Service
RFC	Request For Comments
RPC	Remote Procedure Call
RTP	Real Time Transport Protocol
RTSP	Real Time Streaming Protocol

SMTP	Simple Mail Transfer Protocol
SSH	Secure Shell
STD	Standard
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UI	User Interface
XML	Extensible Markup Language

CHAPTER

1

Introduction

1.1 Introduction

In recent years, following the revolution of the computer age, mankind has stepped into the age of Information which means that networks have come to be part of people's life, not just a mystic word only used by a few researchers any more. Also, advances in computing power and the acceptance of the personal computer in the home and office have led to a plethora of new applications, which tap the enormous computational resource now available even in mainstream desktop personal computers.

These have triggered explosive growth of some new bandwidth-hungry applications, such as networking games, video/audio on demand, e-learning and so on, at the same time traditional network applications do not diminish their speed of increment. In the information age, the network user population has been increasing by almost 100% per-year, the increment of network traffic even reaching a 300% growth in annual traffic [FomKMC04]. As a result, Internet traffic has reached an unprecedented level [Odl03]. A suggestion has been made that technological advances in networking will make bandwidth cheap and consequently, always under-utilized. The reality is quite different – the adoption of faster networking technologies such as ATM did alleviate some of the problems but once again, however, the combination of the expensive cost of implementation, need for standards, infrastructure and widely needed applications slowed the deployment of ATM [Acts03]. The conflict between increasing user demand and existent network resources thus becomes more and more serious.

1.2 Network Management

One solution to the problem is to fundamentally rebuild the existing network system. This is a perfect solution since it would solve all known defects, and furthermore, provide a nice foundation platform for further exploitation [Cle06]. Nevertheless, the cost will be huge and the deployment will take a long time.

Maximizing existing infrastructure performance would be much more feasible. This does not alter the structure of existing equipment, instead, by optimizing this, it would supply decent performance with a cost as low as possible. As such, network management is raising its value.

1.2.1 The Importance of Network Management

Performance issues are very important in computer networks. When hundreds or thousands of computers are interconnected, complex interactions, with unforeseen consequences, are common. Some problems, such as congestion, are caused by temporary resource overloads, some by structural resource imbalance. Solving these problems is the primary task for network management [Dan01].

“Network management is the process of using hardware and software by trained personnel to monitor the status of network components and line facilities, question end-users and carrier personnel, and implement or recommend actions to alleviate outages and/or improve communications performance as well as conduct administrative tasks associated with the operation of network” [Com99]. This is the definition of network management. In my words, it is “the process of making the most of the network”. As indicated in this chapter, we are in the midst of an explosive growth of network traffic. Network management is entering a new phase in its development. For any network manager, it is quite easy to list several reasons to demonstrate how important the task of network management is. A fast, advanced network must have all of its components optimally working together to most effectively deliver the applications and services for which it was designed. This requires solid capabilities of controlling, planning, allocating, deploying, coordinating, and monitoring the resources of a network [Pilm04]. It is also

essential to maintain an alert mechanism and troubleshooting capabilities to discover problems, and react quickly to fix them. **Table 1.1** summarizes the major reasons why the network must be managed, providing the rationale for network management.

Dependence upon network availability

Effect of network failure

Network size and complexity

Coping with network device sophistication

Network performance and capacity planning balance

Operating cost containment

Table 1.1 *Rationales for Network Management*

1.2.2 Quality of Service

QoS (Quality of Service) refers to both the performance of a network relative to application needs and the set of technologies that enable a network to make performance assurances. It uses a set of quantities, which are so-called QoS parameters that network elements would control in order for networks to make QoS guarantees to applications in terms of providing a certain contracted level of service throughout the application session. QoS parameters include the following:

- Packet Rate
- Latency
- Jitter
- Packet Loss Rate (PLR)
- Packet Error Rate (PKR)
- Throughput

Much effort and research related to QoS has been made to improve the performances of networking applications [KarKL03] [PlaBH99] [GelL05] [WanYFZF00]. QoS does not create bandwidth, but manages it so that it is used more effectively and efficiently to meet the wide range of application requirements. In the application layer, a variety of QoS protocols such as RSVP [BraZBH97] were developed to enhance the performance of applications crossing network, these include some compression techniques and a host of coding techniques to adapt applications to the network conditions on which they are running [NirB95], not just assigning high priority to time sensitive applications, but involving programmable network infrastructures which can be dynamically configured to service individual applications.

Application detection provides the ability to identify what applications are currently in operation and the knowledge of the requirements from the network. If this information has been known, a QoS activity can then carry out on the next action, which is probably adapting the network configuration to satisfy the needs of that application and maximizing performance [TenSSW97]. This is an area where the ability to automatically detect what applications are running in a given part of the network will be a valuable aid.

1.3 Network Security

The growth of communications in the age of Information has made both individuals and organizations highly dependent upon the use of networks to perform their normal day-to-day tasks. Unfortunately, there has been a corresponding increase in unauthorized exploitation and misuse of these computer systems.

1.3.1 The Importance of Network Security

The network is a bad neighbour [CheB03]. For the information security aspect, no individual or organization would like to suffer from the damage or loss of their information, regardless of how sensitive or critical the information. Problems are not limited to information security breaches. Many organizations are losing a lot of money through uncontrolled Internet access or in a word, abuse. A survey of nearly 200

international companies carried out by Infosec, NetPartners and Secure Computing Magazine estimated that a typical large company (1,000 employees) could be losing £2.5 million per year through employees' use of Internet for non-business purposes – an average loss of £2,500 per employee [Cla00]. Therefore, the need for security has become a vital issue for networks, not only because of the extreme importance of the data residing on the networks but also the important performance of communication lines over which the data are transmitted.

1.3.2 Network Threats

Network threats can be classified in two high level terms—Destruction and Unauthorized Access [AshM99].

Denial of service (DoS) [CheB03] is the most typical of destructive attacks. By sending more requests to a machine than it can handle, it makes the host unable to service the requests coming from legitimate users. DoS attacks are probably the nastiest, and most difficult to address [HusHP03]. These are the nastiest, because they're very easy to launch, difficult (sometimes impossible) to track, and it isn't easy to refuse the requests of the attacker, without also refusing legitimate requests for service.

Unauthorized Access refers to a number of different attacks, such as Executing Commands Illicitly, Confidentiality Breaches, Data Diddling (or Data Changing) and Data Destruction [HusHP03]. The goal of these attacks is to access some resource that your machine should not provide to the attacker.

Unauthorized Access can be achieved in both passive and active ways. **Table 1.2** (reproduced from [AshM99]) summarizes ways in which access can be gained.

Passive accesses are in the nature of eavesdropping on, or monitoring (sniffing) of, transmissions [Kum05]. The goal of the opponent is to obtain information that is being transmitted. This kind of attack is very difficult to detect because it does not involve any alteration of the data. However, it is feasible to prevent the success of these attacks. Thus, the emphasis in dealing with passive attacks is on prevention rather than detection.

Another category of unauthorized access is active accesses. These attacks involve some modification of data stream or creation of an attacking stream. They present the opposite characteristics of passive attacks, it is quite difficult to prevent active attacks absolutely, instead, the goal is to detect them and to recover from any disruption or delays caused by them. Because the detection has a deterrent effect, it may also contribute to prevention.

Intrusion Activity	Type of Attack Tool	Method of Access
Access violation (Actively)	Password cracker or guessing IP spoofing, Trojan horse Java applet	Telnet, remote control program,
User/host discovery (Actively, Passively)	Ping, Sniffer	ICMP tools on corrupted system, Sniffer Tools
Exposure of Network Vulnerabilities (Actively)	Scanner	Tools on corrupted system
Corruption of information (Actively)	Virus, worm	E-mail, file transfer corrupted system
Theft of information (Actively, Passively)	Password cracker or guessing, IP spoofing, Sniffer	Tools on corrupted system, Sniffer Tools

Table 1.2 Security Threats and Access Techniques

The availability of a robust application detection mechanism helps the network manager to discover unwanted networking sessions by positively examining the malicious traffic or negatively identify abnormal traffic on well-known ports such as a virus operating on port 80.

1.4 Contribution to Knowledge

Traditional application detection techniques heavily rely upon the content inside the traffic. This Thesis attempts to find a new application detection mechanism for some classes of TCP-based applications. The ideas discussed in this Thesis can provide an alternative approach in order to achieve application level classification for some areas where traditional or current popular application detection mechanisms show poor ability.

In summary, the work in this Thesis has found a new identifiable signature and a statistical method in detecting network applications based on this signature for some classes of TCP-based applications.

For some TCP-based applications, variations on the distribution profiles response to varied network conditions are seen to follow a predictable pattern. It appears that the impact of delay is to cause an aggregation of packet payloads. This is basically caused by the use of the Nagle algorithm, which states that any data sent subsequently should be held until the outstanding data is acknowledged (ACKed) or until there is a full packet's worth of data to send. However, as the variations are predictable, an approach has been suggested and experimentally verified its feasibility for various distributions.

The detection technique discussed in this Thesis provides the ability for detecting some classes of TCP-based applications on which conventional techniques would show poor detecting ability. As this technique is a statistical method, it shows advantage over those traditional content techniques especially when the application data are encrypted or under bad network conditions such that some specific packets could not be captured.

1.5 Outline of Chapters

This Thesis is organised as below:

Chapter 2 introduces the main TCP/IP concepts related to the work in this Thesis, the traditional and current popular application detection techniques and their characteristics are briefly discussed. The statistical comparison method Chi-square test is also introduced.

Chapter 3 describes preliminary experimental work which was for the purpose of examining the feasibility of making the packet size distribution a detectable signature for TCP-based networking applications. The distribution profiles have been tested under different running conditions and loaded network conditions in order to discover if they are unique and consistent.

Chapter 4 concentrates on solving the problems emerging from Chapter 3, i.e. some Nagle-based applications show aggregation in their packet size distribution. An aggregated detection mechanism is presented and some experiments are described to verify the feasibility of the methods proposed.

Chapter 5 describes the design and architecture of a prototype application detector employing the ideas discussed in the last two chapters.

Chapter 6 describes the procedure of selecting suitable parameters for the detector. A number of tests performed and the results obtained with the prototype detector under a variety of conditions are also presented.

Chapter 7 provides summary conclusions of the results achieved by this work.

CHAPTER

2

Application Detection Techniques

2.1 Introduction

In this chapter, the traditional and current popular application detection techniques and their limitations are briefly discussed. The idea of detecting TCP-based applications using an alternative application “signature” packet size distribution, which is the core of this work, is also presented.

2.2 TCP/IP Reference Model

2.2.1 Seven Layer OSI Model

7	Application				
6	Presentation				
5	Session				
4	Transport		Transport		TCP, UDP
3	Network		Internet		IP
2	Data Link		Network Interface		Network driver software, NIC
1	Physical		Hardware		
<i>OSI Reference</i>		<i>TCP/IP Reference</i>		<i>Protocols</i>	

Figure 2.1 OSI and TCP/IP reference mode and associated protocols

Figure 2.1 (reproduced from [Mar94]) is a seven-layer model. Each layer performs specific functions and communicates with the layers directly above and below it. The higher three layers deal more with user services, applications, and activities whereas the lower four are concerned more with the actual transmission of information.

The TCP/IP reference Model is used in the worldwide Internet, which is the most popular communication network today. The second column shows the correspondence between the TCP/IP Reference Model and OSI Model. The TCP/IP model does not have session or presentation layers and experience with the OSI model has proven this view correct: they are of little use to most applications [KurR03]. TCP and UDP protocols have been defined at the Transport Layer, while the Internet Layer defines an official packet format and protocol called the Internet Protocol. The TCP/IP model does not care about what happens below the Internet Layer, but points out that the host has to connect to the network using some protocols over which IP packets can be sent/received.

2.2.2 Transport Layer Protocols

TCP/UDP Protocols

Two different protocols are defined in this layer—TCP and UDP. The User Datagram Protocol (UDP), defined by IETF RFC768, provides a simple, but unreliable message service for connectionless services. Each UDP header carries both a source port identifier and destination port identifier, which allow high-level protocols to target specific applications and services among hosts. A datagram can be sent at any moment without prior advertising, negotiation or preparation [Com99]. The datagram is just sent and it is hoped that the receiver is able to handle it. There is no guarantee that the datagram will be delivered to the destination host. Not only could the datagram be undelivered, but it could be also delivered in an incorrect order. It means a packet can be received before another one, even if the second has been sent before the first one received. [KurR03]

All applications included in the work of this Thesis are based on TCP (Transmission Control Protocol), which is described in STD-7/RFC-793. TCP is, unlike UDP, a connection-oriented protocol that is responsible for reliable communication between two end processes [Ric94]. Before actually transmitting data, a connection must be established between the two end points. The recipient need only be completely identified at the time the connection is established. The data can be then transferred in full duplex (send and receive on a single session). When transferring data, only information sufficient to identify the connection is required. TCP guarantees that all data sent will be received without any error and in the correct order. Should any error occur, it will automatically be corrected (retransmitted as needed) or the error will be notified if it cannot be corrected [WanC91].

Interactive and Bulk Data Flow

TCP applications basically use either interactive or bulk mode to exchange communication traffics. In the interactive mode, the TCP initially uses a stop-and-wait protocol. The sender of a data packet requires an acknowledgment for that packet before the next one is sent. An optional *delayed acknowledgement* strategy has been defined in [Bra89]. Delayed ACKs allow a receiver to refrain from transmitting an ACK for every

incoming packet immediately [Koz05]. However, the receiver must send a piggyback ACK with the next packet it sends. In addition, an ACK should not be delayed for more than 500m sec while waiting for payload from applications to arrive [Mar94]. In the bulk mode, on the other hand, the sender is allowed to transmit multiple packets before it stops and waits for an acknowledgment. TCP uses a variety of mechanisms in order to achieve the congestion control in this mode. These mechanisms control the rate of data entering the network, keeping the data flow below a rate that would trigger collapse. Modern implementations of TCP contain four major algorithms: Slow-start, congestion avoidance, fast retransmit, and fast recovery [AllP99]. Each of these algorithms controls the sending rate by manipulating a congestion window that limits the number of outstanding unacknowledged bytes that are allowed at any time. When a connection starts, the slow start algorithm is used to quickly increase congestion window to reach the network capacity [SteCWA99]. When the sender detects that a packet has been lost, it deems that the network is overload and decreases the congestion window quickly. After estimating the network capacity, TCP switches to the congestion avoidance algorithm [WanC91], which slowly increases the congestion window in order to make the most use of the bandwidth.

The Nagle Algorithm

In addition, some applications in employ the Nagle Algorithm, which is named after its creator – John Nagle. The Nagle algorithm is used to automatically aggregate a number of small-buffered messages. This process increases the efficiency of a network application system by decreasing the number of packets that must be sent.

Nagle's document [Nag84] specified a means of dealing with what he called *the small packet problem*, created when an application generates data one byte at a time, causing the network to be overloaded with small packets (a situation often referred to as *send-side silly window syndrome*). A single character – one byte of data – originating from a keyboard could result in the transmission of a 41-byte packet consisting of one byte of useful information and 40 bytes of header data. This situation translates into 4000% overhead, which was considered to be acceptable for a lightly loaded network, but not so for a heavily loaded network, where it could necessitate retransmissions, cause lost

packets, and hamper transmission speed through excessive congestion in switching nodes and gateways [Ric94].

The Nagle algorithm works by aggregating a number of small outgoing messages, and sending them all at once. Specifically, as long as there is an outstanding packet for which the sender has received no acknowledgement, the sender should keep buffering its output until it has a full packet's worth of output, so that output can be sent all at once.

In the work of this Thesis, information contained in the Internet layer is used in the application identification process, and some effects on this information caused by the Transport Layer are considered.

2.3 Current Application Detection Methods

A number of methods may be applied to identify an application. The ideal network application detection approach would require two communicating network nodes to announce the identity of the application they are running as part of the session start process. The identity information could be included within the session traffic. This requires that the location of this information in the packets should be known in advance. However, this is still an "ideal", many applications in use today operate without such announcement.

2.3.1 Port Number

A port number is a way to identify a specific process to which an Internet or other network message is to be forwarded when it arrives at a host. For the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP), a port number is a 16-bit integer that is put in the header appended to a message unit. This port number is passed logically between client and server transport layers and physically between the transport layer and the Internet Protocol layer and forwarded on.

The port numbers are divided into three ranges [Iana07]:

Well Known Ports: from 0 through 1023 (inclusive).

Registered Ports: from 1024 through 49151 (inclusive).

Dynamic/Private Ports: from 49152 through 65535 (inclusive).

Some services or processes have conventionally assigned permanent port numbers. These are known as well-known port numbers. The Well Known Ports are controlled and assigned by IANA [Iana07] and on most systems can only be used by system (or root) processes or by programs executed by privileged users. The well-known ports use a small portion of the possible port numbers. For many years the assigned ports were in the range 0-255. In 1992, the range for assigned ports managed by the IANA was expanded to the range 0-1023 [PosR94]. Many traditional network services were assigned associated port numbers on which they could bind. For these processes, the identifications would be relatively reliably detected by looking at the port number they are using. However, this would not always be the case as almost no service specifies that it must listen on a fixed port. For example, the HTTP protocol indicates that a Web server should basically listen on port 80, but it is possible for a particular Web server to use a different port number [Koz05]. An administrator could manually change the configuration to make the server listen on port 8080. As such, identifying a well-known service using port number will not guarantee the result is always right.

The Registered Ports are listed by IANA and on most systems can be used by ordinary user processes or programs executed by ordinary users [Pos94]. The registered port numbers are in the range from 1024 through 49151. Companies and other users register port numbers in this range with IANA for use by the applications. Some applications use fixed registered port numbers when sending and receiving data over the Internet by default. Application detection at its simplest level can make use of this fact. An example of this is WarCraft III. It uses port 6112 at the server side. Nevertheless, in fact, one could only say that WarCraft III will use port 6112 instead of saying that an application using port 6112 must be WarCraft III. This is because although many applications have been registered to IANA so that multiple uses of the same ports can be avoided, there are still many that have not yet as registration is not forced. IANA only registers uses of these ports as a convenience to the community [Pos94]. Hence, the possibility of two or more applications using the same registered port does exist. In addition, in a similar way to the

situation for well-known ports, most applications using registered ports allow users to change the default port usage. However many users never use this function.

With so many applications developed for use over the Internet, registration of port usage for every one becomes impractical. For this reason, there is a move toward using dynamically allocated ports [Bha01]. For these applications, a dynamic port number also named an ephemeral port is assigned temporarily (for the duration of the request and its completion). An ephemeral port can be chosen from either the range of Registered Port Numbers or Dynamic/Private port numbers. It will last for only a brief time. The port is then abandoned and can be used by other applications. One could argue that ephemeral port is typically used in server-client direction – registered port is still used in client-server direction. However, there are many applications that are using ephemeral ports in dual directions and rely upon the centralized servers in order to allow the servers and clients can be found by each other. Third-Party network gaming platform is a notable example. People are establishing many servers to distribute the ephemeral port pairs and clearly, port numbers cannot be utilized to identify these applications.

2.3.2 Packet Classification

There are a number of network services that may require packet classification, such as application-aware routing, access-control in firewalls, policy based routing, provision of differentiated qualities of service, and traffic billing [GupM00]. The categorization function is performed by a *flow classifier* (also called a packet classifier), which maintains a set of rules, where each flow obeys at least one rule. The rules classify which flow a packet belongs to based on the contents of the packet header(s), for example the set of packets whose source address starts with prefix bits S , whose destination address is D , and which are sent to the server port for web traffic. Associated with each flow is an action which defines the additional processing — example actions include sending to a specific queue, dropping the packet, making a copy, etc [MacF02].

Packet classification does supply some information about the types of services by extracting some fields from the packets. However, actually, packet classification is not an application detection technique. Its goal is to determine which flow an arriving packet belongs to so as to determine — for example — whether to forward or filter it, where to

forward it to, and what class of service it should receive. These services require that packets be classified based on a set of rules that are applied to the destination address, flow identifiers such as source address, layer-4 protocol type, and port numbers. As such, no application level information is required for this technique. And also, no application level detection ability can be supplied. Therefore, for an application detection scheme to work, deeper analysis of the traffic stream must be performed.

2.3.3 Stateful Inspection

As well as examining header information, Stateful Inspection can examine the contents of a packet (up to the application layer) to determine more contexts about the packet beyond its source and destination information accommodated in the header. In addition, Stateful Inspection monitors the state of a connection and compiles historic information in a state table. Some services, which are operated over the application level of the TCP/IP reference model, are standards-based. With these services, generally, application level headers were defined by RFC or some other well-known document, and one could simply detect these services through digging into the service header field available inside every single packet [CheP04]. This approach can for example be applied to those traditional network applications such as email server (or client) that use SMTP over TCP/IP and web browser clients that use HTTP.

Always-on Stateful Inspection introduces a further ability of application detection by incorporating full-time session state awareness and extra context information, which is stored and updated dynamically. This form of Stateful Inspection provides cumulative data against which subsequent communication attempts can be evaluated and acted upon in real time [Raj05]. It also delivers the ability to create virtual session information for tracking connectionless protocols (e.g. Microsoft RPC and UDP-based applications). This approach is often applied on services such as RTP or the set of standards defined in H.323, the session start-up procedure could be tracked in order to achieve identification as for these services, the establishment patterns of connections are often known in advance and source application information could be extracted from a series of packets [Bha01].

Stateful inspection uses information that utilizes layers 3-7 of the OSI model (network

layer and upwards) in order to obtain information such as the type of service/application [Raj05]. By combining information from various layers (transport, session, and network), a detector is able to better understand the protocol that it is inspecting and make more intelligent decisions on the identification of application.

2.3.4 Deep Packet Inspection

The term “Deep Packet Inspection” describes a variety of features that enable a detector to scour individual data packets or streams of packets to spot specific code or other context that might be part of a signature of certain application [SchML03]. Deep Packet Inspection directs, persists, filters and logs IP-based applications, including Web traffic, based on content encapsulated in a packet’s header or payload. As already discussed, always-on Stateful Inspection features enable detection devices to move beyond just inspecting traffic based on the information contained in data packet headers to monitor active connections. Deep packet inspection supplies the ability to dig even deeper into traffic flows to spot hidden signatures like Web, e-mail, and DNS servers. By way of example, Deep Packet Inspection lets the detection device look deep into the content of a TCP or UDP flow for a complete view. This is accomplished by reassembling IP datagrams, TCP datastreams and UDP packets as they flow through the device to view the entire application content. Viruses and many recent applications have fallen into the area where the application detection can be performed by this approach. To perform effective Deep Packet Inspection (DPI), it is vital that the protocol traffic be reordered in the form it was originally transmitted. To bridge the gap between receiving packets out of order and/or in fragments and the requirement to perform Deep Packet Inspection, the detection device requires a “reorder engine” to take all of the packets and put them in the original transmission order [KruVVK02]. Once put in the proper sequence, by searching for the key strings, which included in the payload portions, some non-standard application patterns can be matched.

Current application detection mechanisms provide accurate detection for some classes of applications, especially well-known services that are standards-based messaging.

However, due to the nature of these techniques, they would show poor detecting ability on some applications on which the work of this Thesis has concentrated.

2.4 Statistical Methods

2.4.1 The Fragility of Current Detection Techniques

Current detection techniques clearly have their advantages. However, there are cases where they will fail in their task. The operation of content based classification techniques such as deep packet inspection require that considerable processing and buffer memory is used as a number of complete datagrams often in sequence, including their headers and data portions must be captured and stored for processing for every network session. In addition, depending upon the exact technique adopted, the capture of certain specific packets may be necessary (i.e. the packets that are sent to establish the session or a sequence of packets potentially containing a key string) which may be quite difficult to implement, especially in a high load network environment. The fundamental shortcoming of content-based detection techniques is that the packet formats, the content signature and session negotiation mechanisms must be known in advance, and the content of packets captured should be obtained. This limits their ability to operate on encrypted datagrams in which the content will present in an unintelligible form [ParBLPO03].

The work to be described presents a different approach to the problem. The detection techniques proposed would compliment conventional methods since they target detection of particular classes of applications. If a statistical property of the traffic stream can be found that is largely unique to a given application then detection may be possible with very little packet decoding necessary. One could simply compare this characteristic (or measured for an unknown application) with pre-evaluated samples stored in a database and select the appropriate match. We have examined one such characteristic, namely the packet size distribution.

2.4.2 Rationale of the New Statistical Approach

The theory of the approach is quite reasonable. In a packet switched network, messages that some networked applications exchange may be of fixed size. However, when a networked application is designed, to ensure efficiency, the messages that are exchanged should be as brief as possible. As a result, an exchanged message may represent just a few elements of application information as is necessary. In a packet switched network, the messages would be encapsulated in a packet when ready for transportation. The length of the packet header is known in advance and is a standard size. The length of the payload (the message) can be known by checking the transmitted packet size or alternatively, checking the packet header field. While network applications are running, the information that they exchange would generally conform to a certain pattern, which means the messages or the length of the packets would also conform to this pattern. For example, in some real-time games, the information regarding a player such as position and status are the most frequent messages that are transmitted, this information can be encapsulated in a defined structure *Struct A*, which has a fixed length for transportation. Contrarily, some building information using a structure *Struct B* that is a different length may be used occasionally, thus the packet sizes will present a distribution with a major peak at the size of *Struct A* and a minor peak at the size of *Struct B*. In some networked video/audio players, almost all the messages transmitted will contain the frames of video/audio files. One video/audio file is basically coded at a single bit rate, which decides the frame size and the definition of the playback. As a result the packet sizes will present a distribution with a peak at the frame size, since the size of packets that contain playing frames will be fixed or just vary a little.

The two examples above may be too idealised to represent the practical reality, as the actual cases will be much more complex. However, the packet size distributions observed so far are unique and it is expected that many networked applications will have their own particular packet size distributions [ClaM00]. The distribution of packet sizes observed over some time interval could be determined and if sufficiently consistent for a given application but different across applications, one could use this as an application signature. The approach has been successfully applied for some UDP real-time applications [ParBLPO03]. The work of this Thesis has concentrated on extending this work to TCP applications.

2.4.3 Potential Effects Caused by TCP on Packet Size Distribution

As mentioned in the previous section, TCP and UDP are two different Transport Layer Protocols. For UDP-based applications, the sending of packets is controlled completely by the application itself, the packet size distribution thus can only depend on the application by which it is generated. However, it is very likely that the packet size distribution from a TCP-based application will be affected by the transportation mechanism of TCP. This is the area where most effort has been focused in the work of this Thesis.

Some applications studied in this work utilize the TCP bulk mode or interactive mode [Ric94] without the Nagle Algorithm. These applications behave in a similar way to UDP-based applications. Their original packet size distributions may show consistency and no variation under different network conditions. The Nagle-based applications however, could potentially suffer from the effects caused by this algorithm. Due to the nature of the Nagle Algorithm, a packet could be aggregated when the last outstanding packet is lost or its acknowledgement is delayed. As TCP is a stream-oriented protocol without apparent boundary [ForB05], it is impossible to know the original packet size of the application. The packet size distribution therefore could vary from case to case. Hence, some further analysis needs to be performed in order to detect Nagle-based applications. Another significant effect caused by TCP could be the MSS (maximum segment size), which is the largest amount of data, specified in bytes, that a computer or communications device could handle in a single, unfragmented packet.

2.4.4 Statistical Chi-square Test

Chi-square Test

The essential requirement of the detection technique described in this Thesis is to achieve detection by comparing the packet size distribution of an unknown application against stored distributions from known applications. As such, it is necessary to choose a proper statistical method to test the fitness between two distributions. The Chi-square test is thus adopted in order to test the goodness-of-fit.

The reason for choosing the Chi-square test over other statistical test methods is that the packet size distribution is a frequency statistical measurement, and the Chi-square test is particularly appropriate with variables expressed as frequencies [Sch88]. In addition, when used for frequency comparisons, the Chi-square test is a non-parametric test, since it compares entire distributions rather than parameters (means, variances) of distributions. Except for the need to avoid very small hypothetical frequencies, the test is relatively free of constraining assumptions [Hay88].

Goodness of Fit Test

The goodness-of-fit test consists of determining whether the frequency counts in the categories of the variable agree with a specified distribution. To carry out a Chi-square goodness-of-fit test, a null hypothesis must be defined in advance:

H_0 = the two distributions come from the same random discrete variable population

and the alternative hypothesis is stated as:

H_1 = the two distributions are not from the same random discrete variable population

In general, one may not expect the observed sample frequencies to be equal to the expected frequencies, even when H_0 is the correct model. However, if there are large differences, a poor “fit” is evidence that H_0 is not the correct model.

The Chi-square value (χ^2) is an overall measure of discrepancy between the observed frequencies and the expected frequencies under H_0 . The Chi-square value χ^2 of two sets of data is given by

$$\chi^2 = \sum \frac{(o_j - e_j)^2}{e_j} \quad (2.1)$$

where the sum is over all categories, with o_j the observed frequency count and e_j the expected frequency count in bin number j .

This weighted sum of squared differences is equal to 0 if and only if every observed frequency is equal to the corresponding expected frequency under H_0 , that is, if the fit is perfect. If there are large differences between o_j and e_j , then χ^2 will be large, which in turn suggests that the null hypothesis should be rejected [Kit96].

When the sum of frequency counts in all categories is large and H_0 is true, the sampling distribution of χ^2 is known to be approximately Chi-squared with $k-1$ degrees of freedom. **Figure 2.2** illustrates a typical Chi-square density curve.

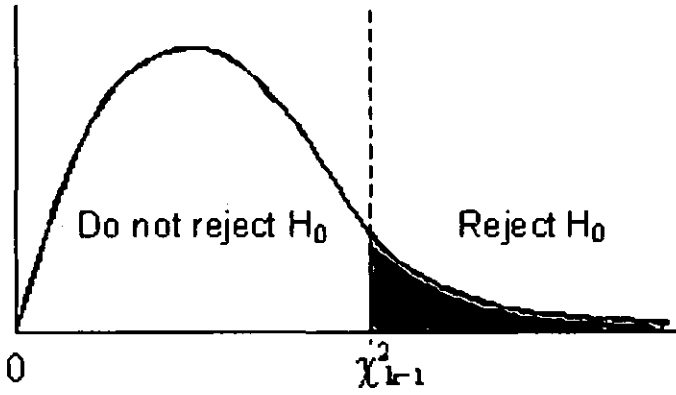


Figure 2.2 Typical χ^2 distribution profile

As shown in **Figure 2.2**, the square of the shadowed area represents the probability that a test Chi-square value χ^2 is greater than a critical value χ^2_{k-1} where $k-1$ is the degree of freedom. In such a case, we have, with a given probability, confidence to believe that H_0 should be rejected. The critical value $X_{v[k]}^2$ for a certain percentage confidence v can be calculated approximately using

$$X_{v[k]}^2 = \frac{1}{2} (t_{2v[\infty]} + \sqrt{2v-1}) \quad (2.2)$$

where $t_{2\nu[\infty]}$ can be looked up in the table of critical values for t -Distribution when the degree of freedom is greater than 50. For those cases lower than 50, table of Critical Values of the Chi-square Distribution could be referenced [HerCRA63].

The Chi-square test requires that frequency counts in all categories should be no less than 5 unless the number of categories is very large and only very few categories have frequency counts less than 5 [Ham75]. Those categories with small-expected frequencies can be combined with neighbour categories to meet this requirement in order to improve the approximation.

2.5 Summary

Current application detection mechanisms provide accurate detection for some classes of applications, especially well-known services that are standards-based messaging. However, due to the nature of these techniques, they would show poor detecting ability on some applications on which the work of this Thesis has concentrated. A new statistical approach was proposed and demonstrated by [ParBLPO03] in order to detect some UDP-based applications. This Thesis considers applying the method on the TCP-based applications. Due to the difference between TCP and UDP, it is expected that some TCP-based applications, which are using the Bulk Mode of the TCP, may behave similarly as the UDP-based application, the packet size distributions will be independent of the network conditions. In another hand, for some TCP-based applications that the Nagle-Algorithm is used, the packet size distributions are very likely to vary under different network conditions.

TCP-Based Application Data Analysis

3.1 Introduction

In this chapter, the preliminary experimental work for the purpose of examining the feasibility of making packet size distribution a detectable signature for TCP-based networking applications is presented.

As described previously in Chapter 2, the packet size distribution has been experimentally shown to be a detectable signature for real-time UDP-based applications [ParBLP03]. Now, we come to the TCP-based situation. In the first place, tests were carried out on a number of TCP-based applications in order to examine the packet size distribution consistencies of these applications. The Chi-square detections among these applications were subsequently introduced so that the visual results can be mathematically confirmed. The effects of network load and application settings on these characteristics were examined to determine if sufficient tolerance existed such that the distribution could be used as an application signature.

3.2 The Experiment System Architecture

3.2.1 Loaded Network Emulator (LNE)

In order to simulate different network conditions, a Loaded Network Emulator (LNE) [OliBPD99] was adopted for this work. Its purpose is simply to drop or delay packets passing through it according to user entered parameters. Loss and delay variations are applied according to a Gaussian distribution and one is also able to specify means around which the variations occur. Similar work has been done by [ArmS04].

This tool works in combination with the *Linux Iptables* [And05], which is a framework that enables packet filtering. *Iptables* is responsible for filtering all packets with an attribute of “FORWARD” and pushes them onto a stack named `ip_queue`. Afterwards, the LNE carries out actions on those packets inside the `ip_queue` in order to decide their behaviour according to the parameters that the user entered---delayed to be forwarded or dropped. Hence, a packet passing through a node on which the LNE is running will suffer delay or loss. A loaded network condition is so emulated. It is important to emphasise that the LNE does not actually generate any new traffic on the network. It simply emulates load by recreating the symptoms experienced by the application traffic (i.e. loss and delay) in a loaded network.

3.2.2 The Experimental Architecture

In order to obtain the packet size distributions of different applications, an experimental architecture was established as shown in **Figure 3.1**.

Two subnets 192.168.1.0 and 192.168.2.0 were deployed. In each subnet, there were PCs on which the applications under test were run. The Linux Box was configured as a network bridge and set to forward so that the two subnets can be connected. *Iptables* was also employed using command

```
iptables -A FORWARD -j QUEUE
```

which forced all packets passing through the Linux Box with an attribute of FORWARD into the *Iptables* packet queue for further action [Ipt06]. The LNE then was used in order to process those packets inside the *Iptables* packet queue so as that different network conditions could be emulated.

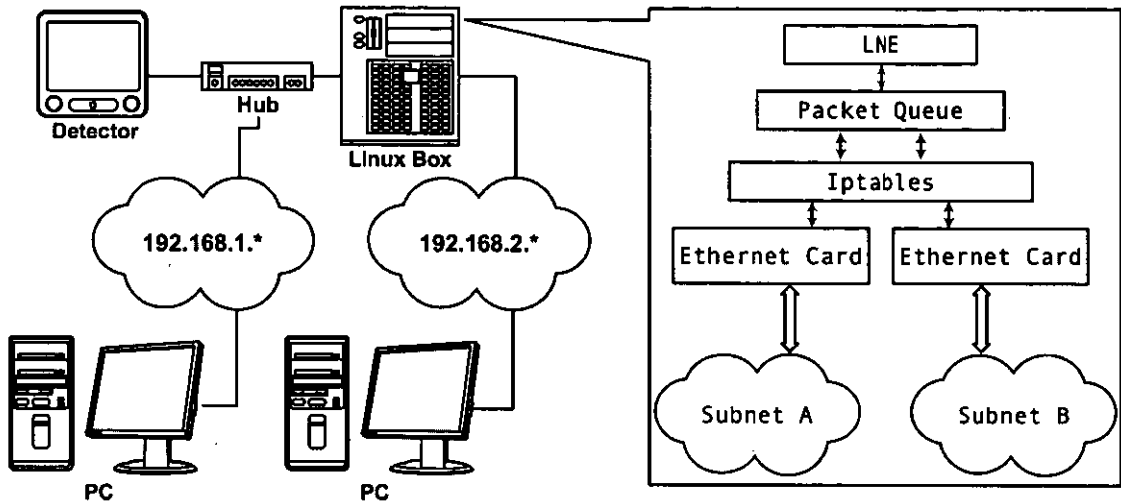


Figure 3.1 Experimental System Architecture

A packet capture process was carried out via the hub inside subnet 192.168.1.0. Since all packets coming from subnet 192.168.2.0 need to be passed through the Linux Box on which a network condition emulating mechanism functioned, all arriving packets on the hub were perturbed by the LNE --- delays or losses were caused, so that, depending on the configuration of the LNE, the packet size distribution profiles under various network conditions were obtained in repeatable and understood conditions.

3.3 The Consistency of the Packet Size Distribution as an Application Detecting Signature

In order to test the consistency of the packet size distribution profiles, a number of applications in different general classes were tested; games, video streaming servers, email-clients and web browsers and so on. The traffic was monitored and analyzed, unique source/destination IP addresses and port numbers identified each connection and the streams in two directions (client to server, server to client) were separately recorded.

3.3.1 The Effect of Application Settings/Scenarios on the Packet Size Distributions

Packet size distributions may potentially vary under different running conditions (such as the intensities of the battles for some networked games or playback bit-rates for the stream media players) [BorS99] [SheN03]. This is vitally important as it affects the use of packet size distributions as a detection signature. Applications must be profiled and tested with differing settings to ensure the accuracy of the approach. In each test performed here, the traffic was captured for a 60s interval, the distribution granularity used was 1 byte per bin for a network MTU of 1500 bytes.

Networked Game Crimson-Sky

Crimson-sky [Mic00] is a typical local area networked game of air-combat simulation originally developed by Microsoft for the platform of the XBOX, and was transplanted to the PC platform recently. It functions by allowing a number of network clients running the game to connect to a server host, which is also running the game. The game is operated in a semi-duplex like mode: it produces two connections, each of which is in charge of the message transportations of one direction (client to server or reverse), the other direction of each connection only transmits the acknowledgements of the packets in which gaming messages are carried, and ports used are assigned dynamically.

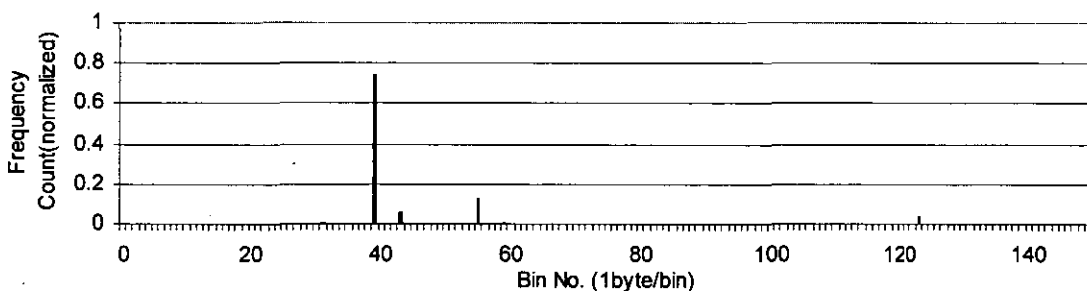


Figure 3.2 Packet size distribution for Crim-Sky (svr-cli) 2 players

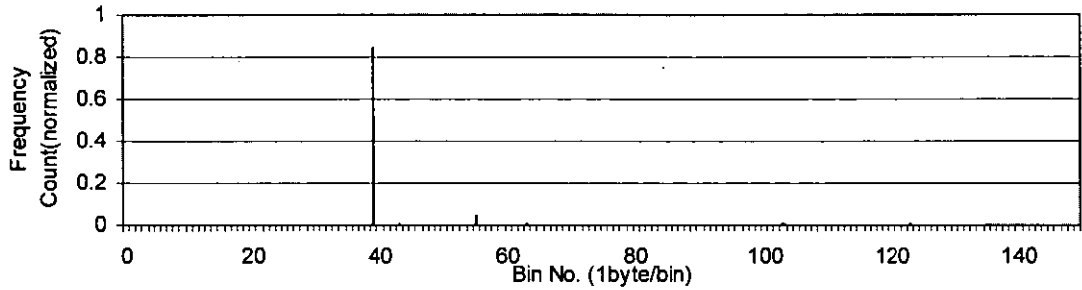


Figure 3.3 Packet size distribution for Crim-Sky (cli-svr) 2 players

Figure 3.2 and **Figure 3.3** show the original profiles under ideal network conditions and in two-player mode. Within the interval period, no more intensive action than flying was performed. It can be clearly seen that for the server to client traffic most packets are 40 bytes in size excluding IP and TCP headers, with a minor peak at 56 bytes, and with a similar profile in the reverse connection.

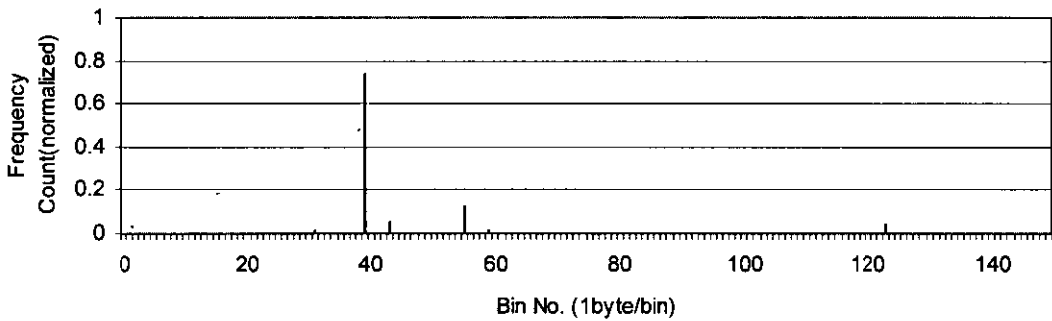


Figure 3.4 Packet size distribution for Crim-Sky (svr-cli) Free-Fly, 4 players

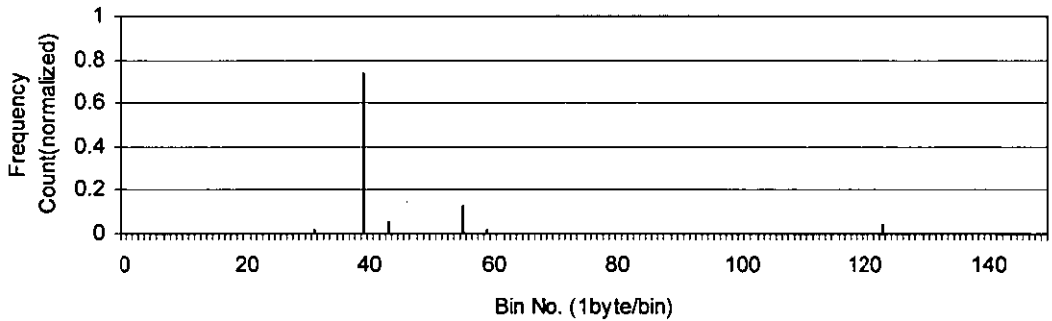


Figure 3.5 Packet size distribution for Crim-Sky (svr-cli) Group-combat, 4 players

With an increasing number of players, or increasing intensity of combat, the packet size distribution appears to remain stable as shown in **Figure 3.4** and **Figure 3.5**, although a few packets of other sizes are also observed.

Gaming Conditions	Chi-Value	Critical value 95%	Freedom
Free-fly 2 players	0	3.940	10
Free-fly 4 players	0.9416	3.940	10
One-on-One combat 2players	1.4237	5.226	12
One-on-One combat 4 players	1.2250	4.574	11
Group combat 4 players	1.7428	5.226	12

Table 3.1 Chi-square analysis results for Crim-Sky under different gaming conditions

The Chi-square values for different gaming conditions tested against the distribution for the condition of Free-fly 2 player (server to client direction) has been calculated and summarized in **Table 3.1**. All packet size distribution profiles have been accepted as coming from the same sampling population with a confidence of 95%. Thus, consistency is proven. The client to server direction presented a similar result, although no details are given here.

Networked Game WarCraft III

In contrast to Crimson-Sky, the popular Internet game, WarCraft III [Bli02], uses only one session for the purpose of communication. The profiles in both directions are shown in **Figure 3.6** and **Figure 3.7**. These are distributions that were captured from a whole battle process. The predominant packet size was 9 bytes with smaller counts for larger packets in the server to client direction and 6 bytes in the reverse direction.

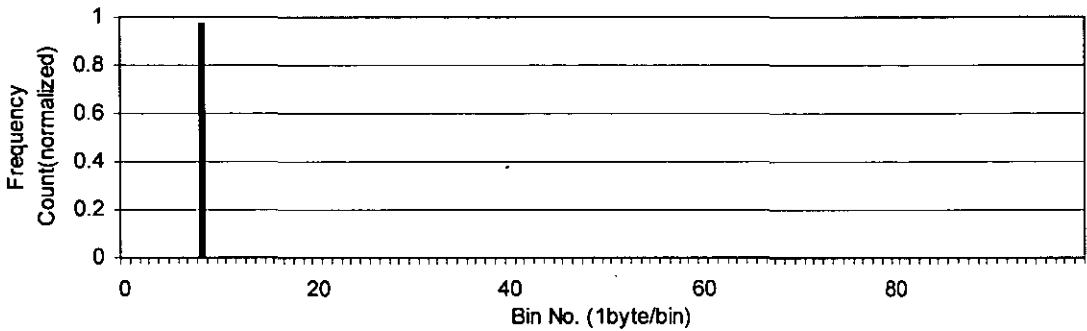


Figure 3.6 Packet size distribution for WarCraft III (svr-cli) overall

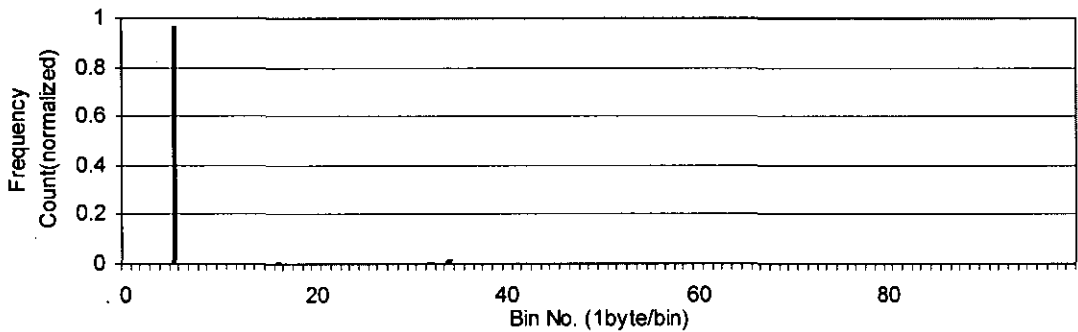


Figure 3.7 Packet size distribution for WarCraft III (cli-svr) overall

As for Crimson Sky, the profiles in both directions remained stable with increasing number of players, while under different intensities of battle. The inbound stream under serious intensity was seen to have slightly lower peaks compared to the mild case. The counts obtained for larger packets were seen to be slightly larger compared with those for smaller packets. The profile shapes were in general, similar, however (Figure 3.8 and Figure 3.9).

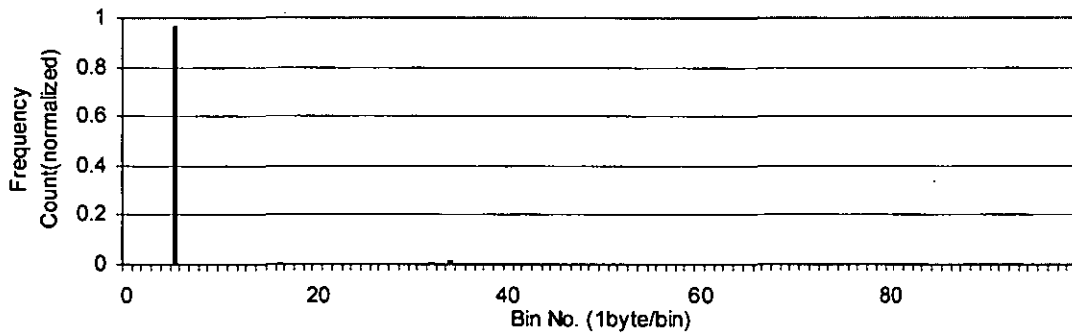


Figure 3.8 Packet size distribution for WarCraft III (svr-cli) high intensity fight

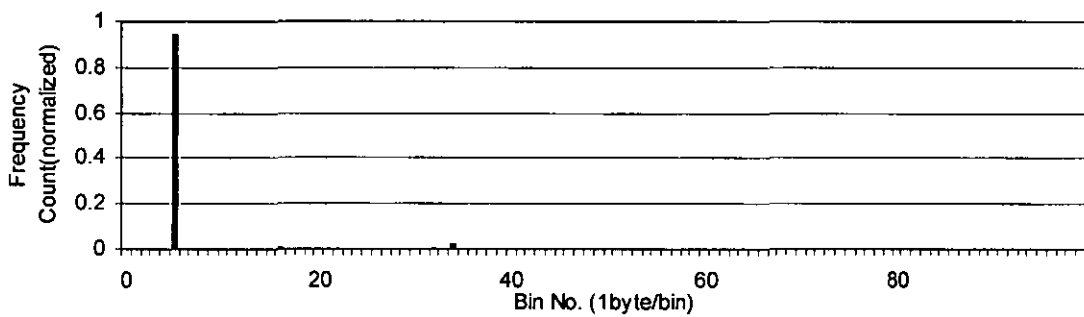


Figure 3.9 Packet size distribution for WarCraft III (cli-svr) high intensity fight

Gaming Conditions	Chi-Value	Critical value 95%	Freedom
Integrated Battle 2 players	0	40.645	57
Low Intensity Battle 2 players	2.9999	45.741	63
Low Intensity Battle 4 players	4.4129	54.325	73
High Intensity Battle 4 players	3.5053	49.162	67

Table 3.2 Chi-square analysis results for WarCraft III under different gaming conditions

The Chi-square values between the profiles under different running conditions are given in **Table 3.2**, with a confidence value of 95%. As seen, the distributions are similar.

Networked Multimedia Software RealPlayer

RealPlayer [Real00] is a widely used media player. It utilizes the RTSP protocol [BaiSWC00] developed by Real-Network Ltd. to provide a multimedia service across the Internet. The profiles for playing a number of files with different bit rates have been obtained and shown in **Figure 3.10**. Traffic is only available in the server to client direction, no traffic in reverse direction was observed (except the ACK packets without any payload) as one would expect because of the nature of the video/audio service.

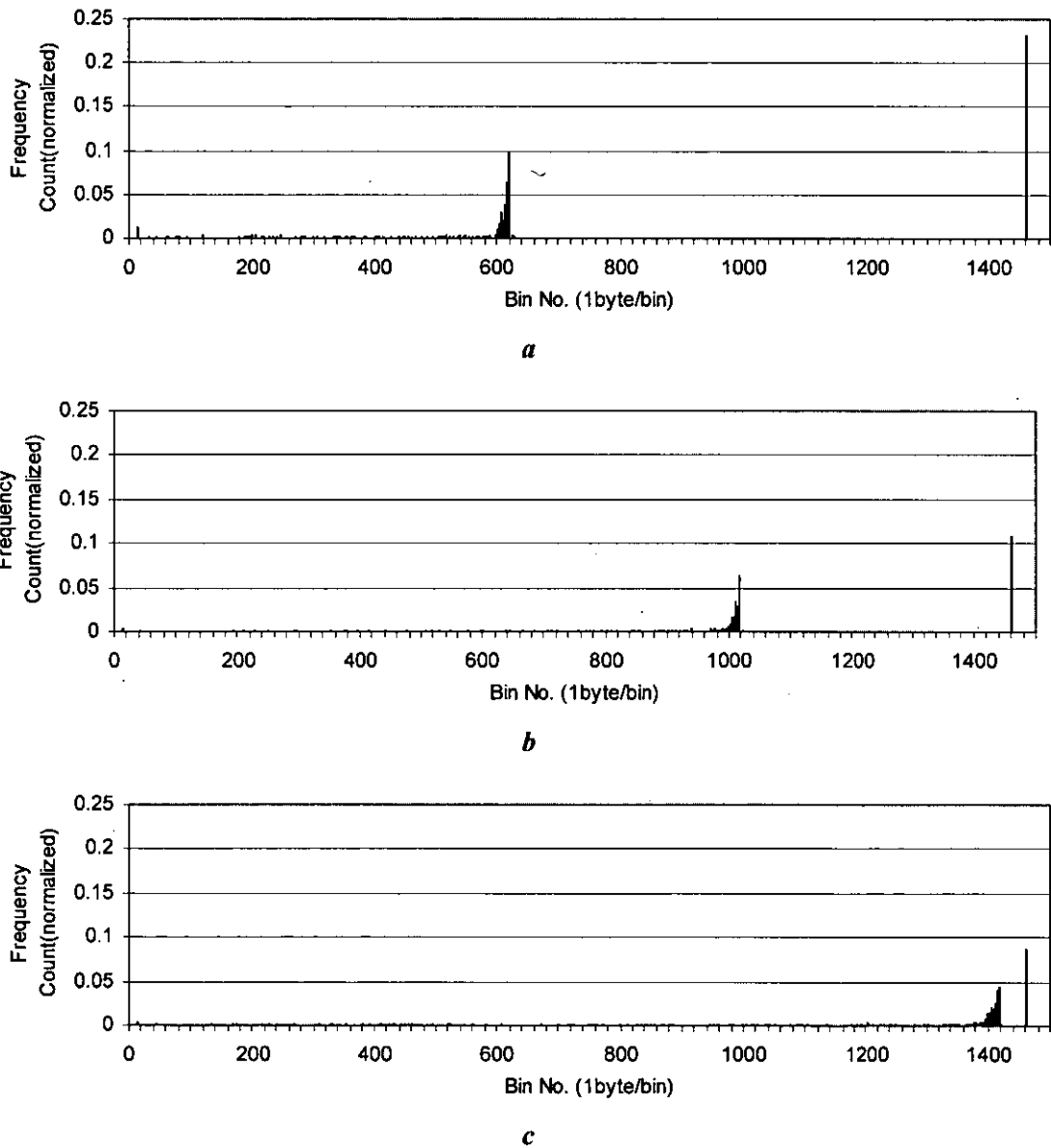


Figure 3.10 Packet size distributions for RealPlayer while playing files at different bit rates

- a. 225 kbps
- b. 450 kbps
- c. 600 kbps

The profiles (**Figure 3.10**) indicate that the packet size distributions shifted rightward with increasing bit-rate of the media files, although the shapes of the profiles appear to remain similar. As such, poor packet size distribution consistency was expected. Another problem is that most of the traffic had been transmitted in the packets that are at the size of the MTU. Different clips with various coding bit-rates were subsequently tested and Chi-square values were summarized as follow (**Table 3.3**).

	225 kbps(Clip 1)	450 kbps(clip 1)	600 kbps(clip1)
225 kbps(clip 2)	11.573(accepted)	194.999(rejected)	198.132(rejected)
450 kbps(clip 2)	193.899(rejected)	16.655(accepted)	199.999(rejected)
600 kbps(clip 2)	199.999(rejected)	189.234(rejected)	21.333(accepted)

Table 3.3 Chi-square analysis results for RealPlayer for different bit-rates

Other media players show a similar trait as RealPlayer. In addition, media files coded using the same scheme encoding at the same bit rate would show the same profile regardless of the format of them or on which application they are played back. As a result, it may be quite difficult to identify the operating application by matching the packet size distributions at the application level. However, the packet size distribution profile provides the ability to identify the general class of the application, i.e. media streaming in this case.

Traditional TCP Applications

Some traditional TCP applications were also investigated. These traditional applications are not real-time and the traffic they generate is often based on manual input or the content transferred. The packet size distributions of these applications hence could be seriously subject to factitious factors.

HTTP Browser

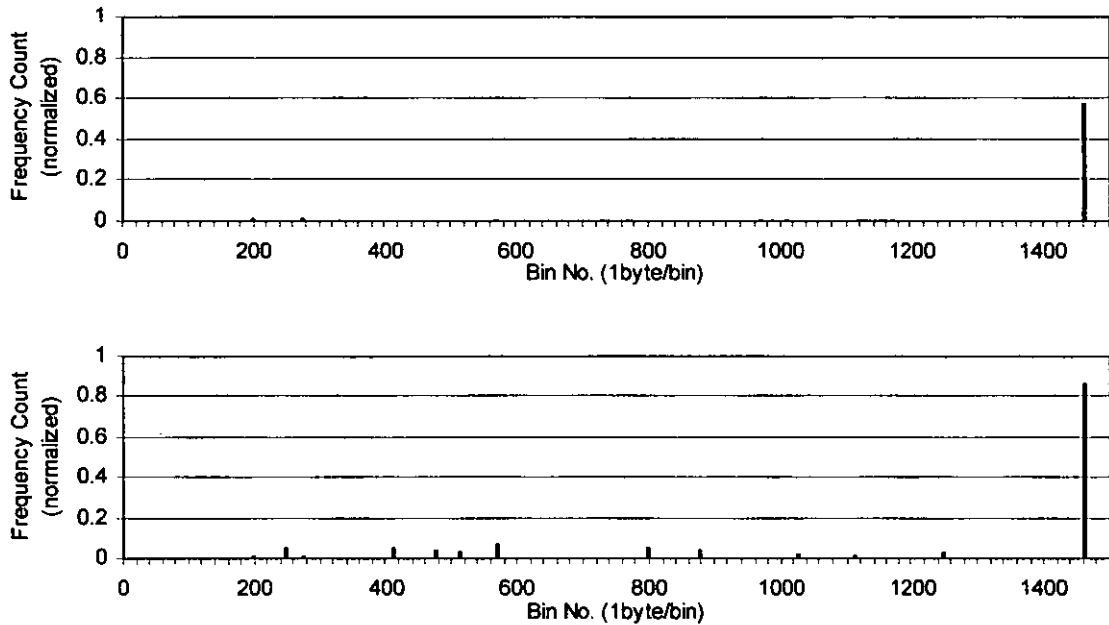


Figure 3.11 Packet size distributions for HTTP

Figure 3.11 shows the profiles for the HTTP [FieGMFB97] while opening two different website pages. Most of the packets were transmitted at the size of the MTU, a few of them are not. Even discounting all the MTU size packets, the distributions still represent no common ground at all as the Chi-square value between them is 196.3433, which was too high to accept the hypothesis that the two profiles came from same random distribution population.

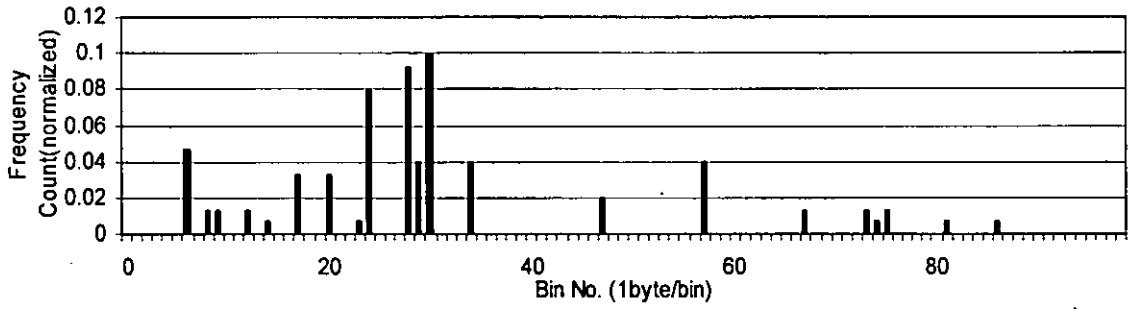
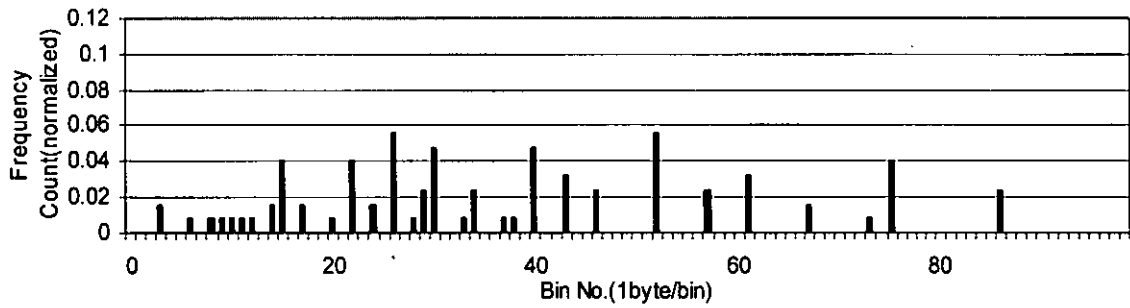
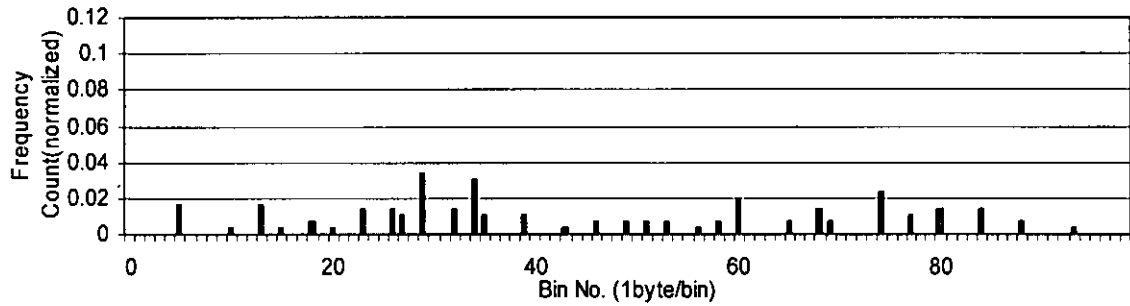
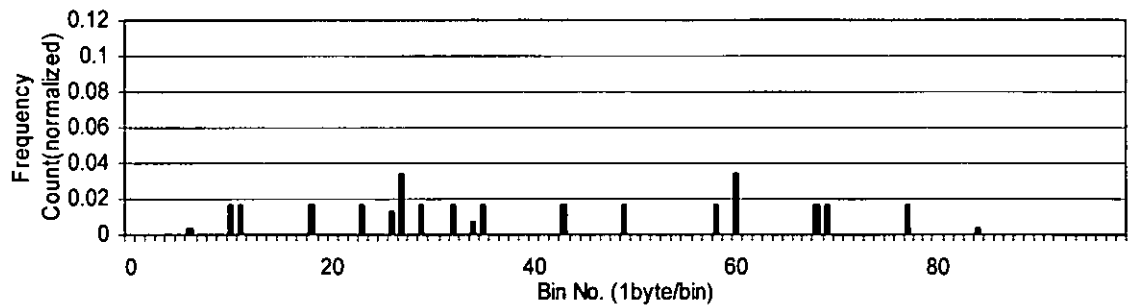
```
uk.www > bo.mshome.net.supfileby: . 197642:199102(1460) ack 2669 win 16483
uk.www > bo.mshome.net.supfileby: P 199102:199223(121) ack 2669 win 16483
uk.www > bo.mshome.net.supfileby: . 199223:200683(1460) ack 2669 win 16483
uk.www > bo.mshome.net.supfileby: P 200683:201137(454) ack 2669 win 16483
uk.www > bo.mshome.net.supfileby: . 201137:202597(1460) ack 2669 win 16483
uk.www > bo.mshome.net.supfileby: . 202597:204057(1460) ack 2669 win 16483
uk.www > bo.mshome.net.supfileby: . 204057:205517(1460) ack 2669 win 16483
uk.www > bo.mshome.net.supfileby: . 205517:206977(1460) ack 2669 win 16483
uk.www > bo.mshome.net.supfileby: . 206977:208437(1460) ack 2669 win 16483
uk.www > bo.mshome.net.supfileby: P 208437:208597(160) ack 2669 win 16483
uk.www > bo.mshome.net.supfileby: . 209897:211357(1460) ack 2669 win 16483
uk.www > bo.mshome.net.supfileby: . 211357:212817(1460) ack 2669 win 16483
```

Figure 3.12 Dumped file details for HTTP

Looking more detailed at the dumped files (**Figure 3.12**), all the packets that are not at the size of the MTU come with the flag PUSH set, following several MTU packets without this flag set. This may result from the fact that a webpage cites a number of images, Flashes or other sort of files which would be sent in a few continuous packets and when the single file has been transmitted, the last packet would be flagged as PUSH (highlighted in Pink) so that TCP can submit it to the browser at once, thus, the last packet should be the tail of such a file transportation. Since different websites will contain different files, the packet size distribution will show diversity. Hence, they cannot be considered as a signature to identify the HTTP traffic.

FTP and SMTP

The FTP and SMTP share a common feature, which is that they establish two sessions for one transportation process—a command/control session and a data transmission session. Generally, the packet sizes of the command/control sessions are always the length of the commands they receive while that of data transmission session are decided by the actual data that FTP or SMTP transmits [PosR85] [Pos82]. Obviously, the command lengths will depend upon the users' input and thus will show no consistency at all, which has been experimentally demonstrated and shown below (**Figure 3.13**, **Figure 3.14**).

*a**b***Figure 3.13** Packet size distribution for FTP control-command sessions*a**b***Figure 3.14** Packet size distribution for SMTP control-command sessions

The packet size distributions of the data transmission sessions are similar to the HTTP transmission. One file (may be one e-mail in SMTP) will be transmitted in a few continuous packets, which are always the MTU size except the last one, which is a tail packet (**Figure 3.15**, **Figure 3.16**). This can be useful as by watching which bin size represents as a peak except the MTU, it may be possible to detect certain abnormal behaviour such as spam or virus transmission, but this is not what this work concentrates on and will not be discussed further.

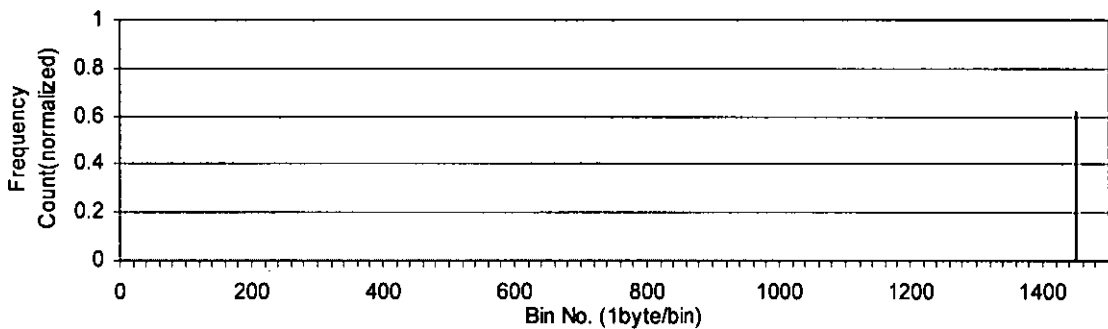


Figure 3.15 Packet size distributions for FTP data transmission session

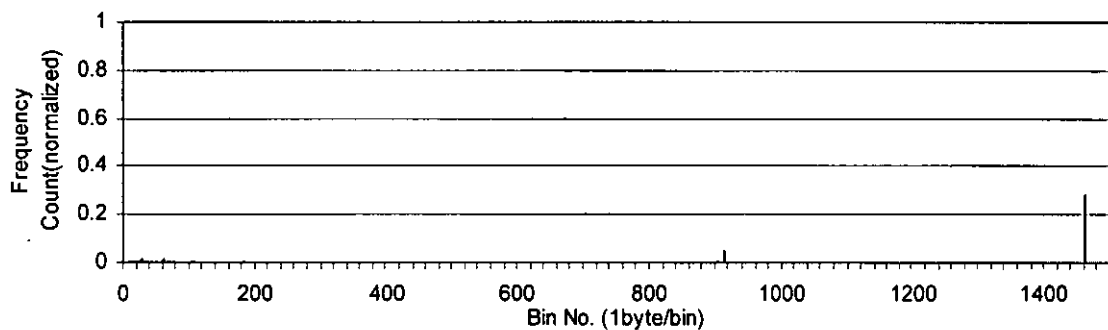


Figure 3.16 Packet size distributions for SMTP data transmission session

SSH

The profiles of SSH (**Figure 3.17**) are quite interesting. SSH provides cryptography and authentication technologies for secure remote login, which means all the packets are encrypted [Ssh05]. For the traditional remote login protocol TELNET [PosR83] which works with the Nagle Algorithm of the TCP protocol, command characters will be buffered and not be sent out until the next sending instant is due and the acknowledgement of the last packet sent has arrived. As such, packet sizes are both command-sensitive and affected by the state of the network connection. The profiles of

SSH were seen to be significantly different from those of traditional remote login protocols. Under ideal network conditions which means no delay and no loss, every keyboard stroke would force the SSH client to generate a single 44 bytes packet without any buffering by TCP, thus, the packets that contain command information (direction Client to Server) were all 44 bytes and the profile show a robust consistency during the tests in which a number of different commands had been operated, while some packets other than 44 bytes were observed among those containing feedback information (direction Server to Client).

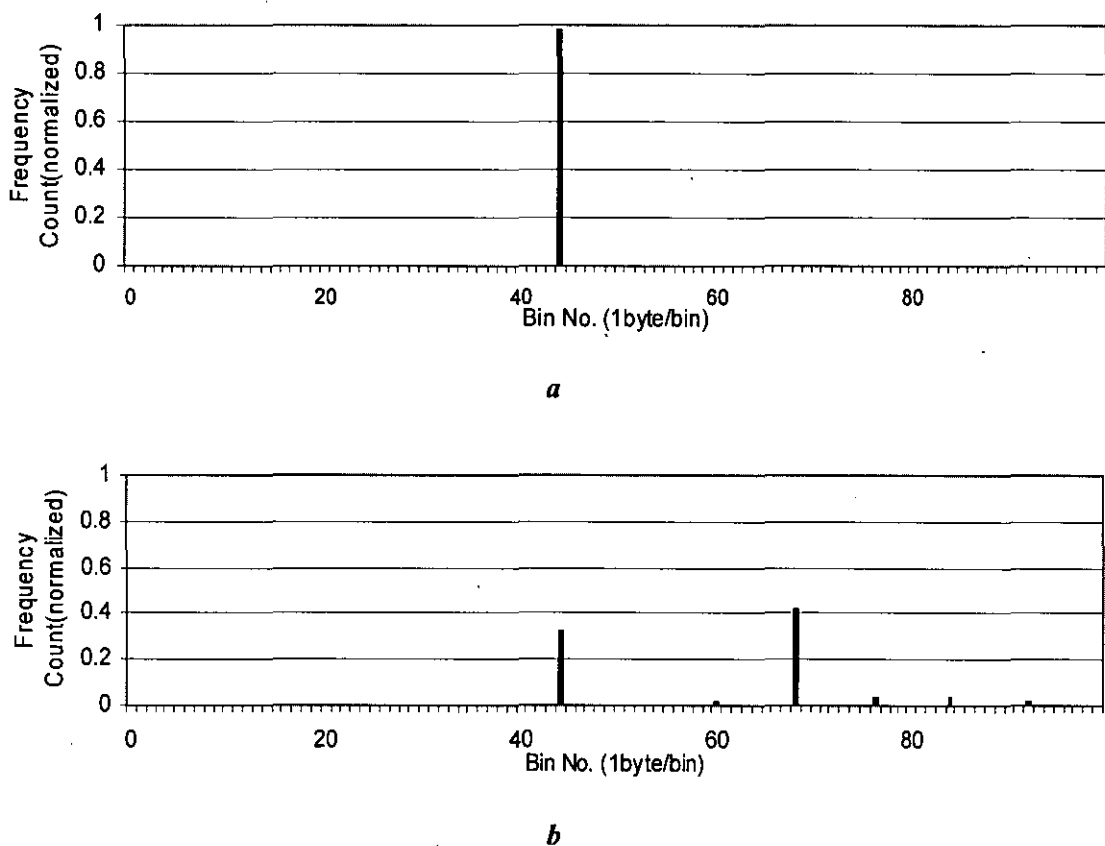


Figure 3.17 Packet size distribution for SSH

a. Client to Server

b. Server to Client

In several later rounds of tests, some commands different from the first round were executed so as to obtain more distribution profiles. Profile plots of a certain round are given in **Figure 3.18**. The Chi-square analysis mathematically indicates that only the profiles of client to server direction support the hypothesis that those profiles were from

the same random distribution population (Chi-square value was 0, critical value with 95% confidence 0.270) but not in the client to server direction (the Chi-square value was 83.033, critical value with 95% confidence 8.343).

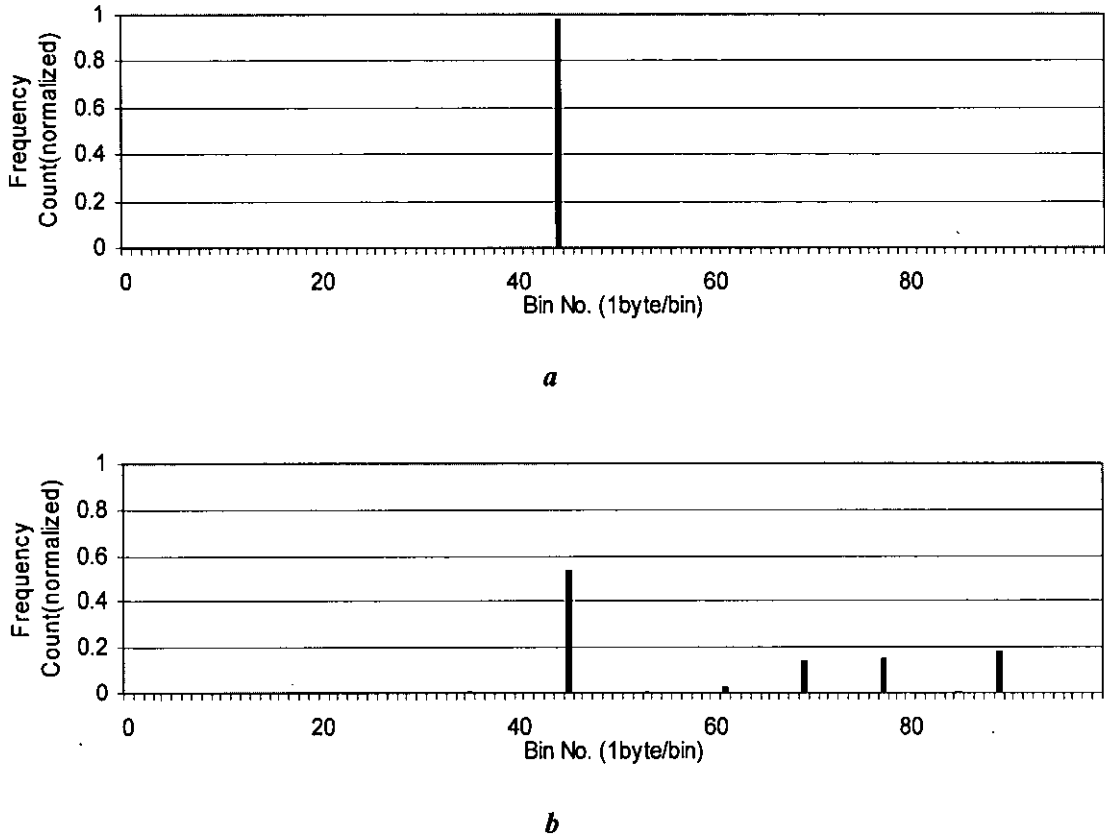


Figure 3.18 Packet size distribution for SSH

a. Client to Server

b. Server to Client

SSH has shown its robust packet size distribution consistency, at least, in one direction, in this case, further analysis would be valuable.

3.3.2 The Effect of Network Load on The Packet Size Distributions

In addition to the effect of running conditions, one must also consider the effects of network load, which could potentially affect the application signature [FenFW02] [Hen01]. The analysis below describes the results obtained with a number of samples of real-time applications operating under varying conditions of emulated load on the

Ethernet test network. For the tests conducted with network load, sessions of each application were run and the traffic stream perturbed by the introduction of packet loss and/or delay by the Loaded Network Emulator.

Networked Game Crimson-Skies

Network Condition Emulated

Packet Loss Ratio (percentage)	0
Packet Delay (ms)	100

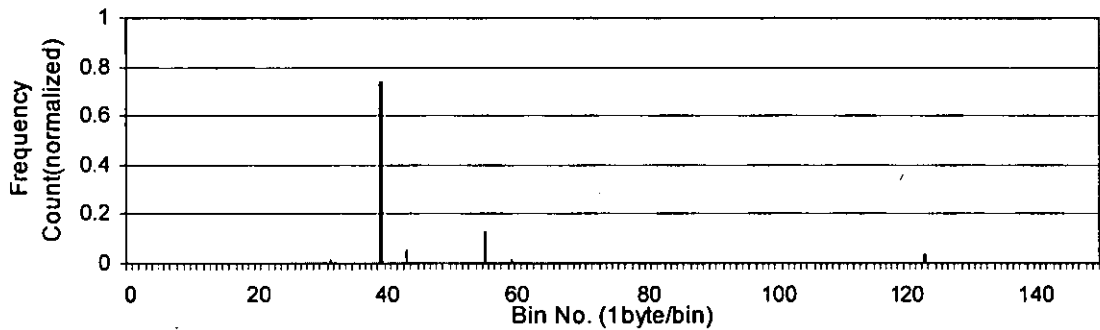


Figure 3.19 Packet size distribution for Crim-Sky under low-load network condition

Figure 3.19 shows the low-load packet size distribution profile of the server to client traffic generated. The profile in the reverse direction observed was similar to the server to client direction. The predominant packet size was 40 bytes with a minor peak at 56 bytes. The difference between it and that under ideal network conditions (**Figure 3.2**) cannot be visually seen.

Network Condition Emulated

Packet Loss Ratio (percentage)	3
Packet Delay (ms)	150

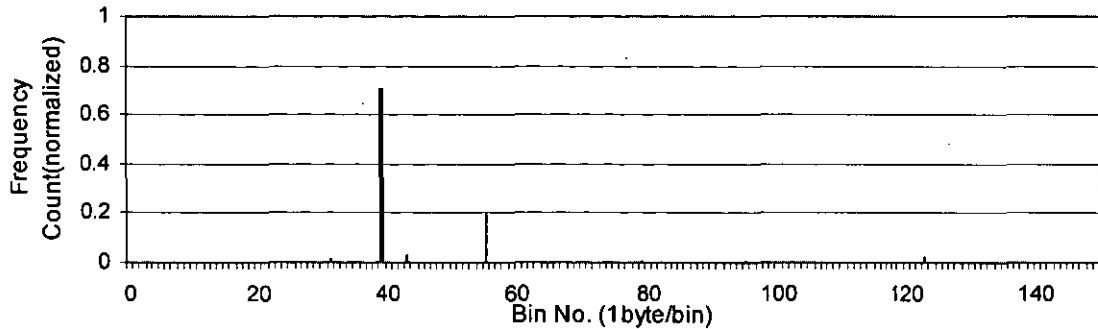


Figure 3.20 Packet size distribution for Crim-Sky under moderate-load network condition

Under the network condition of 3 percent loss ratio in conjunction of 150ms delay (**Figure 3.20**), the counts obtained for 56 bytes packets were seen to be slightly smaller comparing with those for 40 bytes packets. The profile shape was in general, similar, however.

Network Condition Emulated

Packet Loss Ratio (percentage)	5
Packet Delay (ms)	300

A similar result was obtained under the condition of 5 percent packet loss ratio in conjunction of 300ms delay (**Figure 3.21**), packets of 40 bytes still dominate the distribution with slightly higher or lower counts for the packets with size of 44 bytes, 56 bytes or 124 bytes. Even for the condition of 400ms delay (Figure not given), the

distributions remain stable. The profile was therefore very slightly different at this level of load but perhaps still within the bounds of variation that one may expect to see in all applications.

The consistency of the packet size distribution profiles under different network conditions was then proven by a Chi-square test over these results. The results are given in **Table 3.4**. All Chi-square values are lower than the associated critical values of 95% confidence.

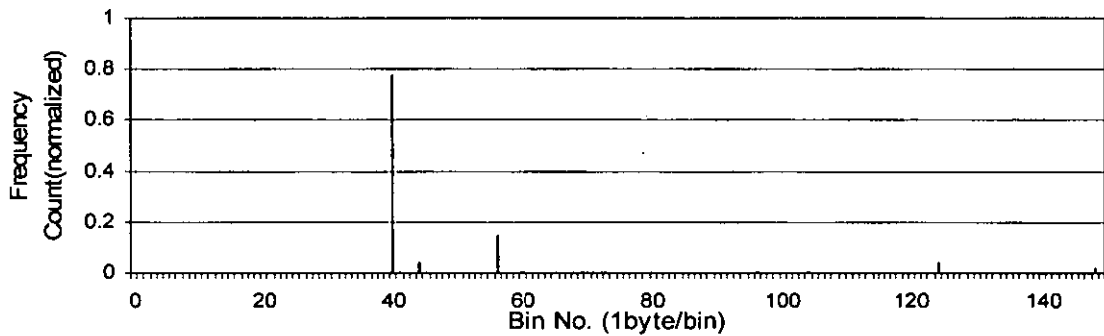


Figure 3.21 Packet size distributions for Crim-Sky under heavy-load network condition

Network Conditions	Chi-Value	Critical value 95%	Freedom
No-load	0	3.247	10
Low-load	1.3343	3.816	11
Moderate-load	1.1275	3.816	11
Heavy-load	1.9344	4.404	12

Table 3.4 Chi-square analysis results for Crim-Sky under different network conditions

Networked Game WarCraft III

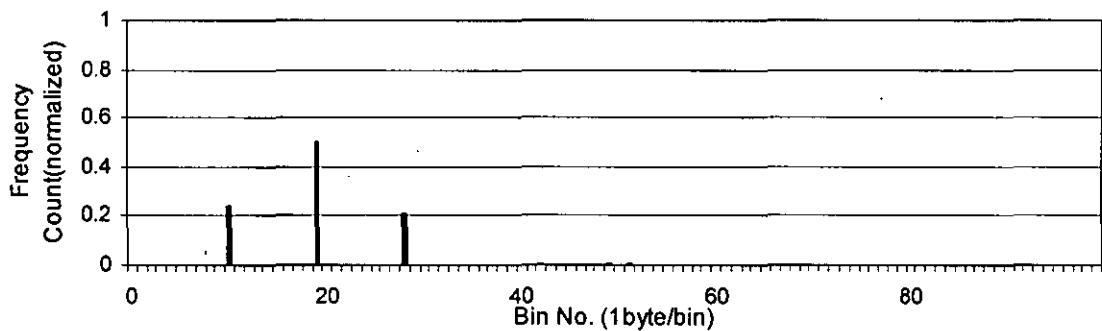
This game is attractive because it reflects the effects on the packet size distributions, which are introduced by the congestion control mechanism of the TCP. The result is corroborated by [She03].

As described previously, WarCraft III established only one session connection during gaming. The two packet transportation directions show different profiles, one can easily tell which end is the server and which is the client.

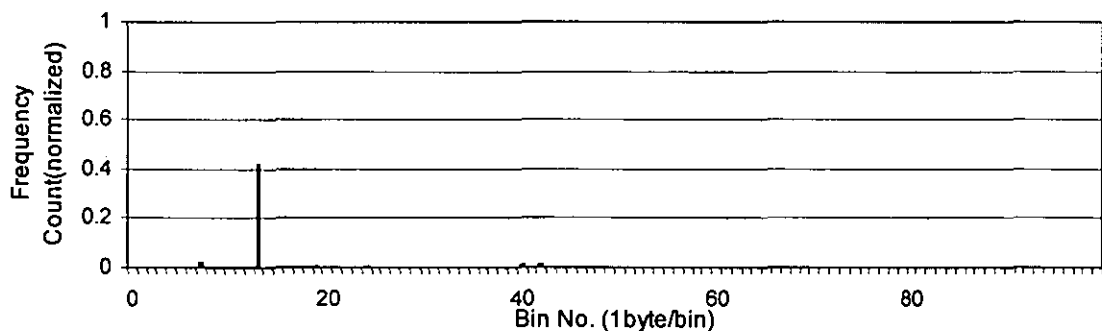
Network Condition Emulated

Packet Loss Ratio (percentage)	3
Packet Delay (ms)	100

Firstly, consider the network condition of 100ms delay and 3% loss ratio (**Figure 3.22**). In both directions, packet counts of 6 bytes and 9 bytes had retreated to secondary peaks, other peaks at 12 bytes (client to server) and 18 bytes represented the first peaks in the plots, packets at 18 bytes, 27bytes even 36 bytes also appeared.



a



b

Figure 3.22 Packet size distributions for WarCraft III under network condition of 100 ms delay

a. Server – Client

b. Client – Server

```

15:22:15.900514 IP 158.125.50.57.1078 > 158.125.51.148.6112: P 10:19(9) ack 12 win 635
15:22:15.999984 IP 158.125.51.148.6112 > 158.125.50.57.1078: P 12:18(6) ack 19 win 167
15:22:16.000658 IP 158.125.50.57.1078 > 158.125.51.148.6112: P 19:28(9) ack 18 win 635
15:22:16.100127 IP 158.125.51.148.6112 > 158.125.50.57.1078: P 18:24(6) ack 28 win 167
15:22:16.100881 IP 158.125.50.57.1078 > 158.125.51.148.6112: P 28:37(9) ack 24 win 635
15:22:16.200216 IP 158.125.51.148.6112 > 158.125.50.57.1078: P 24:30(6) ack 37 win 167
15:22:16.200915 IP 158.125.50.57.1078 > 158.125.51.148.6112: P 37:46(9) ack 30 win 635
15:22:16.300285 IP 158.125.51.148.6112 > 158.125.50.57.1078: P 30:36(6) ack 46 win 167
15:22:16.300952 IP 158.125.50.57.1078 > 158.125.51.148.6112: P 46:55(9) ack 36 win 635
15:22:16.400543 IP 158.125.51.148.6112 > 158.125.50.57.1078: P 36:42(6) ack 55 win 167
15:22:16.401217 IP 158.125.50.57.1078 > 158.125.51.148.6112: P 55:64(9) ack 42 win 634
15:22:16.500670 IP 158.125.51.148.6112 > 158.125.50.57.1078: P 42:48(6) ack 64 win 167
15:22:16.501358 IP 158.125.50.57.1078 > 158.125.51.148.6112: P 64:73(9) ack 48 win 634
15:22:16.600785 IP 158.125.51.148.6112 > 158.125.50.57.1078: P 48:54(6) ack 73 win 166

```

Figure 3.23 Dumped file details of the WarCraft III under ideal network condition

By examining the *Tcpdump* file (**Figure 3.23**), the original packets were sent with a fixed interval of 100ms (highlighted in Red), whilst in the case of loaded network conditions, these intervals were prolonged and no more packets were sent until the acknowledgement of the outstanding packet had been arrived. This phenomenon occurring in WarCraft III could have been caused by utilization of the Nagle Algorithm [Tan96]. It can also be seen that the packet size varies according to network condition, and that the size mostly increased by an integer multiple of the peak size of original profile, which was considered to result from packet aggregation. If the network delays the returning acknowledgements, the Nagle transmission mechanism would buffer any due packets and send a larger packet containing multiple Application Layer messages at the next sending instant. This results in an addition of the original packet size distribution, potentially a number of times when due packets are increasing. One would expect that more packets would be aggregated with worsening of the network condition in the following tests.

Network Condition Emulated

Packet Loss Ratio (percentage)	5
Packet Delay (ms)	200

With the worsening of network conditions, the aggregation phenomenon becomes even more distinct (**Figure 3.24**) as expected. Further analysis could be used to identify the Nagle-based application packet size distributions.

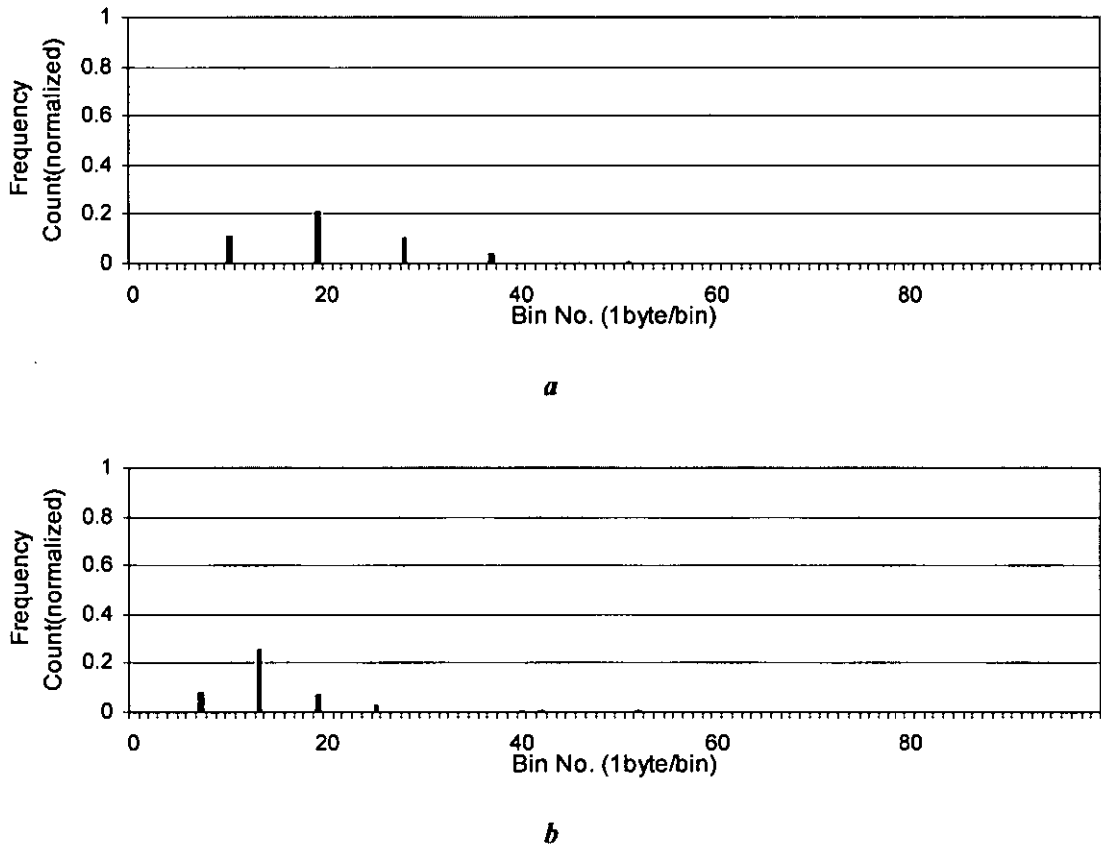


Figure 3.24 Packet size distributions for WarCraft III under network condition of 200 ms delay

- a. Server – Client*
- b. Client – Server*

Network Condition Emulated

Packet Loss Ratio (percentage)	3
Packet Delay (ms)	.0

Under the network condition of 3 percent loss ratio (**Figure 3.25**), the profile shows no distinct difference from that under ideal condition although the losses may have resulted in the resending of a few packets. This suggests that the simple loss of some packets would not cause a variation of packet size distribution profile for Nagle-based applications.

Table 3.5 gives the Chi-square test results, which show that the network delay condition would significantly impact upon the packet size distributions for this Nagle-based application.

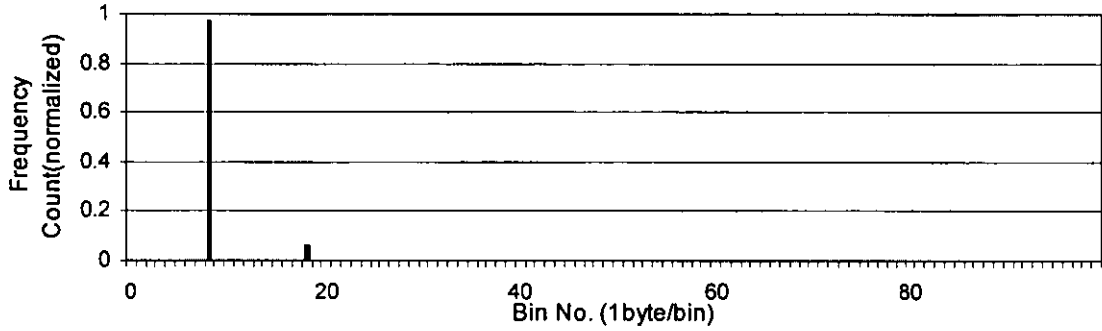
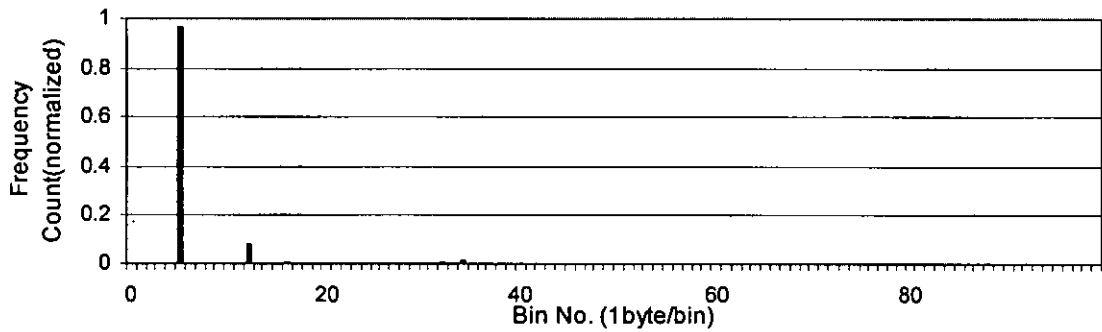
*a**b*

Figure 3.25 Packet size distributions for WarCraft III under network condition of 3 percent loss

a. Server – Client

b. Client – Server

Network Conditions	Chi-Value	Critical value 95%	Freedom
No-load	0	49.162	67
Low-load	86.133	79.697	102
Moderate-load	196.355	105.560	131
Heavy-load	196.934	110.956	137
Loss Only	2.333	50.879	69

Table 3.5 Chi-square analysis results for WarCraft III under different network conditions

Networked Media Player RealPlayer

Network Condition Emulated

Packet Loss Ratio (percentage)	3
Packet Delay (ms)	100

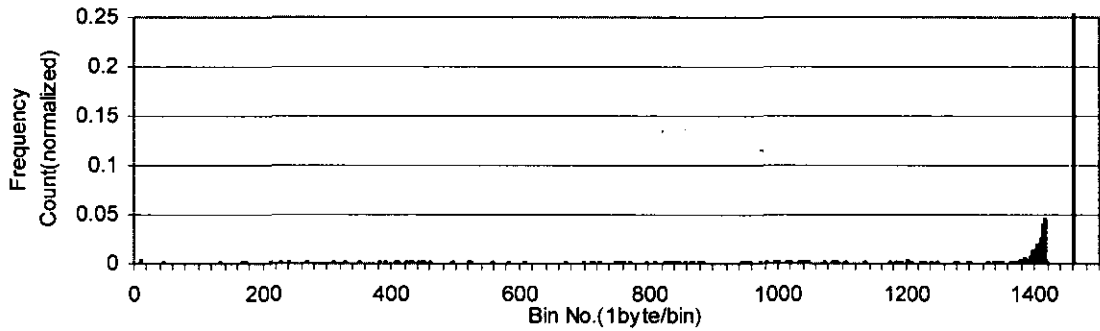


Figure 3.26 Packet size distributions for RealPlayer under low-load network condition (600 kbps bit rate)

As shown in **Figure 3.26**, in accordance with the result expected previously, while a 600 kbps clip was played, more traffic had been transmitted in the packets that are at the size of the MTU while the network condition is worsening, this is also extended by packet aggregation. Being different from networked games, the original packets released by media players are much larger than those of games. Hence, the aggregated packets would frequently exceed the MTU and are thus sent as an MTU packet with the remaining bits pushed into next packet. The playbacks of other bit-rate clips followed similar profiles (Figures not given).

Network Condition Emulated

Packet Loss Ratio (percentage)	5
Packet Delay (ms)	250

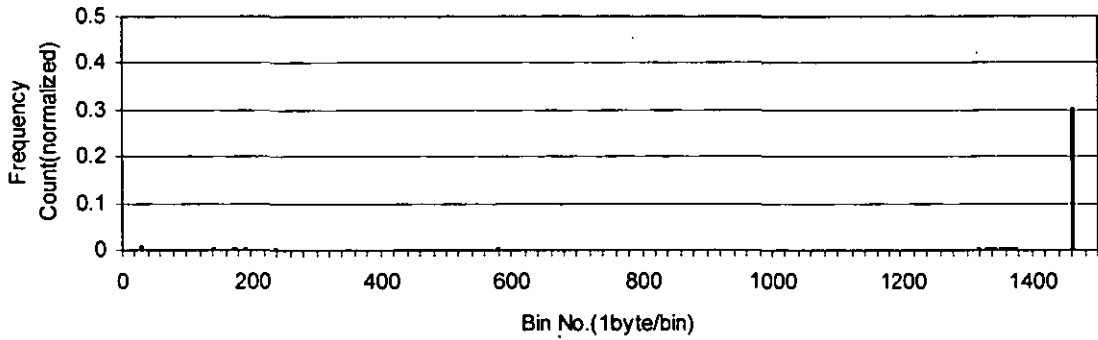


Figure 3.27 Packet size distributions for RealPlayer under heavy-load network

With the network condition worsening, all the packets were at the size of the MTU (**Figure 3.27**). This demonstrates the poor consistency of the packet size distribution of networked media players.

Traditional TCP Applications

The profiles of HTTP under loaded network conditions show no difference from those under an ideal network condition (**Figure 3.28**). This is explainable, when the server received the request from client, the full responding information will be submitted to the Transport Layer at the same time, and then divided into packets that can be transmitted. Thus all the packet sizes have been determined at the same time, the delay or loss of packets will have no effect on them.

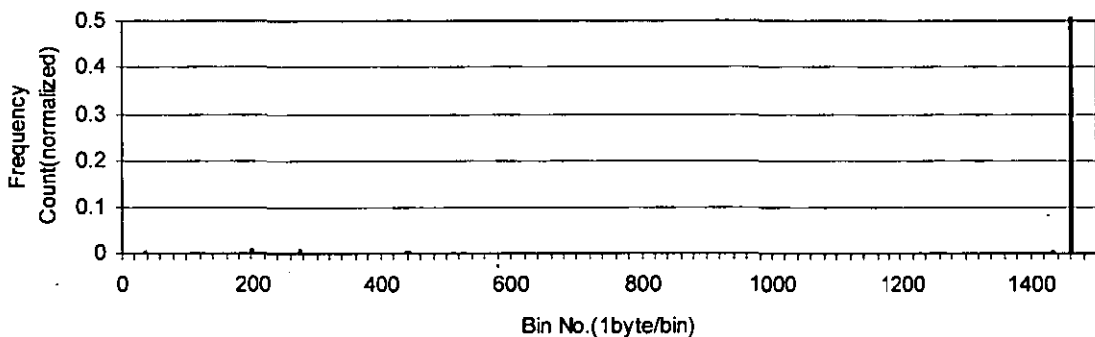


Figure 3.28 Packet size distributions for HTTP under network condition of 200 ms delay

It can be seen that SMTP and FTP were not affected by varied network conditions. **Figure 3.29** gives the profiles of the control sessions of the two protocols under loaded network conditions. This should result from the Bulk Transportation mechanism they deploy.

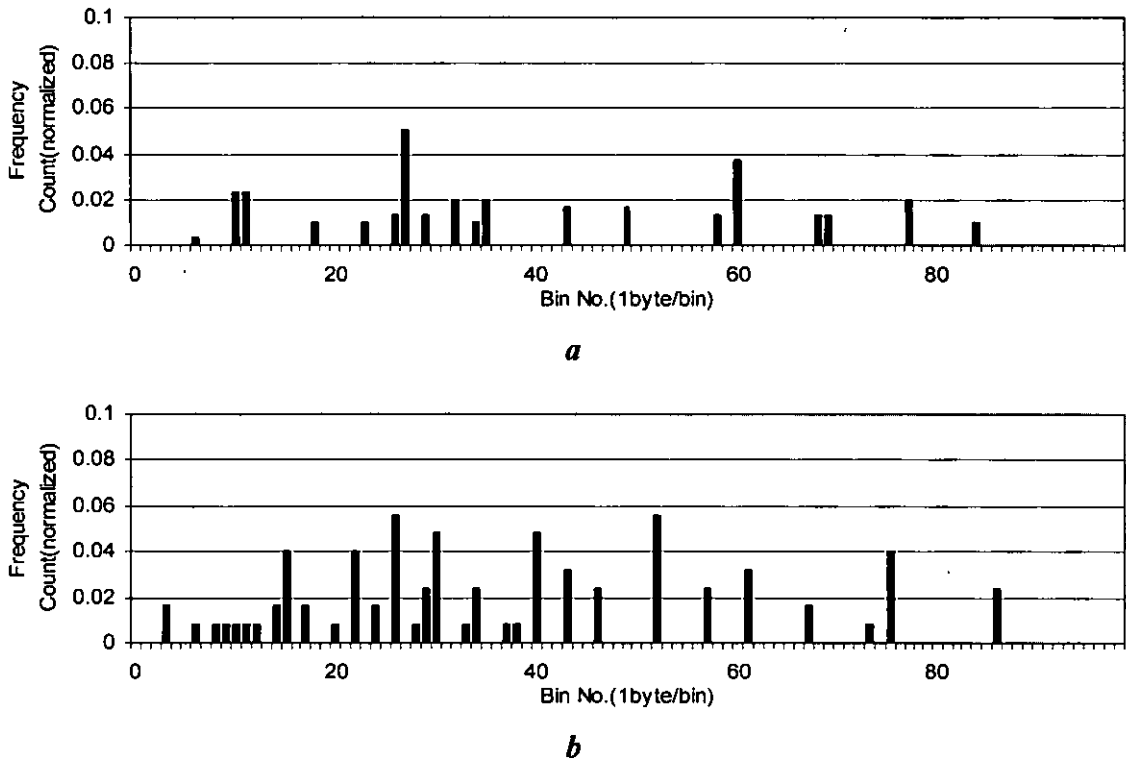
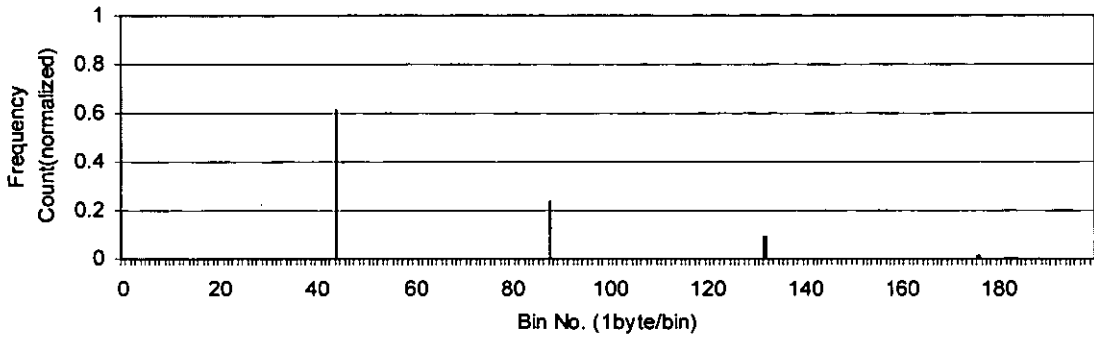


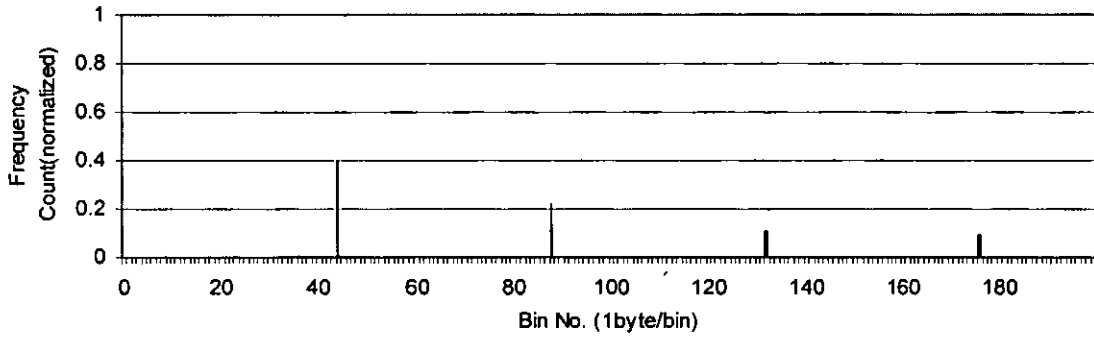
Figure 3.29 Packet size distributions under network condition of 100 ms delay

- a. FTP control session*
- b. SMTP control session*

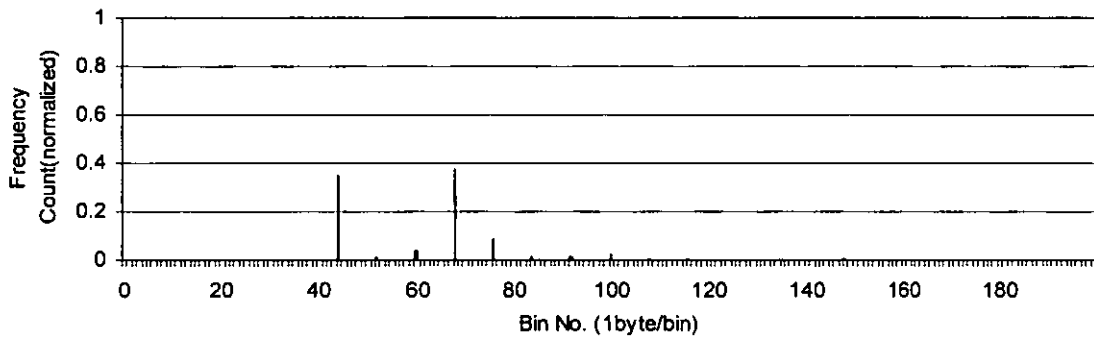
SSH once again showed an interesting packet size distribution (**Figure 3.30**). In the direction of client to server, the same aggregation phenomenon as that occurring on WarCraft III was observed. The packet sizes, which are always 44 bytes under ideal conditions, have been multiplied to 88 bytes or 132 bytes with worsening network conditions, as shown in **Figure 3.30**. On the other hand, in the reverse direction, the situation became more complicated. There seems to be some mechanism controlling packet delay within SSH, the worse the network condition, the more packets of size 76 bytes being observed. As the packets of SSH are all encrypted, it is impossible to dig into the data portion of the packet to find out what exactly happened. Therefore, the packet size distribution profile from client to server is unique from all other applications seen.



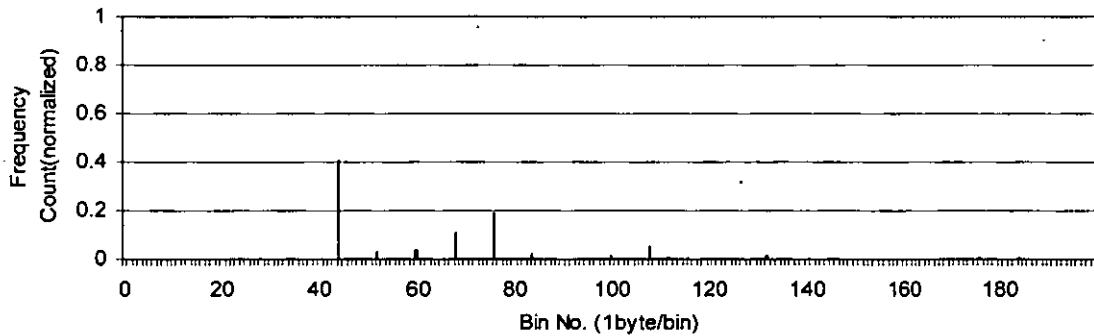
a



b



c



d

Figure 3.30 Packet size distributions for SSH under worsening network condition

- a.* Server – Client under 100 ms delay
- b.* Client – Server under 100 ms delay
- c.* Server – Client under 300 ms delay
- d.* Client – Server under 300 ms delay

Other traditional TCP-based applications (protocols) such as HTTP, FTP, SMTP and TELNET, have shown very poor packet size distribution consistencies. These TCP based applications, however, are standards-based. Web browsers and email clients must be so in order to allow inter-operability with similar products from different vendors. The detection of these applications, for example, in the case of web browsers, may require knowledge of exactly what HTTP transaction is contained in the stream [FieGMFB97] and this can only be done by extensive packet decode through knowledge of the HTTP packet format and transmission mechanism. Commercial products allowing detection and analysis of this type of non real-time application are already available and are not therefore discussed in great depth in this work.

3.4 The Uniqueness Tests on the TCP Packet Size Distributions

It is now necessary to investigate the uniqueness of the packet size distributions for TCP-based applications, in order to determine if a TCP-based application can be differentiated from the other applications by comparing its packet size distribution profile with those of others.

The quantitative uniqueness between relevant distributions was investigated using a database of packet size distributions against which new traces of applications could be compared. The Chi-square test is designed to convert the differences (or deviations) between the two into the probability of their occurring by chance, taking into account both the size of the sample and the number of variables (degrees of freedom) [Sch88].

The selected in-database applications are five TCP-based games and the SSH application. For those applications using the Nagle Algorithm, only the original (i.e. under ideal network condition) profile was considered; the solution of the detailed detection method for these applications will be discussed in Chapter 4. **Table 3.6** gives the stored profile numbers with corresponding application names.

StoredProfile Number	Application Name
1-2	Nardar
3-4	Need For Speed III
5-6	Crimson Sky
7-8	Diablo II
9-10	WarCraft III
11	SSH-Client

Table 3.6 The stored profile numbers and corresponding application names

Crimson-Sky and WarCraft III

The plots below (**Figure 3.31**, **Figure 3.32**) show the Chi-square values of the tests on Crimson-Sky and WarCraft III distribution profiles compared against those of six other applications in the database. The horizontal axis represents the stored profile number. The performance of the uniqueness was good with the lowest Chi-square values occurring at the profile number for Crimson-Sky (No.5, No.6) and WarCraft III (No.9) respectively. Very importantly, in these two cases, the correct applications were very well differentiated from the other applications.

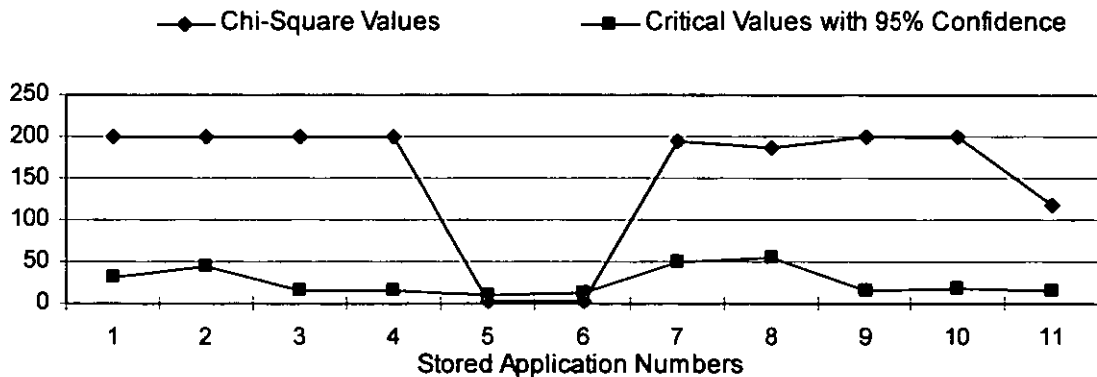


Figure 3.31 Chi-square Values of Crim-Sky against database (Cli-Srv)

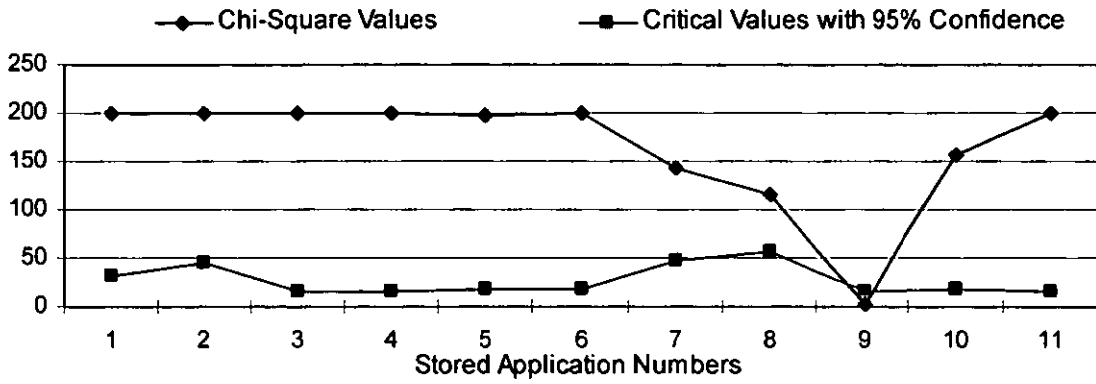


Figure 3.32 Chi-square Values of WarCraft III against database (Cli-Srv)

APP-TEST	Chi-Value	Critical value 95%	Critical value 50%	Next closest app	Chi-Value	Critical value 50%
Crim	10.213	113.090	22.337	WarCraft	196.226	37.335
Nadar	4.281	68.249	88.334	Diablo	191.131	88.334
WarCraft	3.505	47.449	64.335	Diablo	115.892	72.334
DiabloII	49.647	77.929	82.358	WarCraft	189.897	99.334
NFS III	0	0.351	0.584	SSH	200.000	1.064
SSH	0	0.351	0.584	NFS II	200.000	1.064

Table 3.7 Results of Chi-squared tests between different application traces

Table 3.7 shows a general Chi-square summary for all application that had been tested. For each application, the Chi-squared value resulting from the computation with the pre-stored trace for that application is shown, along with the corresponding 95% and 50% confidence value (which varied according to number of degrees of freedom). The lowest Chi-squared value resulting from the computation with a different application trace is also given, again with the corresponding 50% confidence value. As the computation ignores packet size probabilities of zero in both data sets, the confidence value varies from application to application.

It is seen that, for all the applications shown, the lowest Chi-squared value always occurred when the computation was performed with a trace from the same application. In all cases, this Chi-square value is below the critical value for 95% confidence value. In all cases, the next lowest Chi-squared value from a different application is seen to be significantly greater than the first, and much greater than the associated 50% confidence value for this second choice application. Hence the second choice application must be rejected in favour of the first. All the application traces are uniquely identifiable, and the majority can be considered to be statistically unique for the database.

3.5 Summary

In this chapter, the consistency of the packet size distributions of the TCP-based applications was tested. It experimentally shows that, for some applications, the packet size distributions do not vary as the running conditions are changed. Networked media players and traditional TCP applications however show different characteristics – the packet size distributions vary significantly when different contents are transferred. In addition, as discussed previously, some applications using bulk mode behave in the same way as the UDP-based applications, the packet size distributions are consistent under different network delay/loss levels, whereas some the Nagle-based applications aggregate their packet size distributions as the network condition is worsening. For these applications, analysis in advance is required in order to achieve the detection.

CHAPTER 4

TCP Applications with the Nagle Algorithm

4.1 Introduction

The tests described in the previous chapter have illuminated that some TCP-based real-time applications show robust packet size distribution consistencies whereas others do not. Those that do not have in common that the Nagle Algorithm has been adopted based on the analysis of the *Tcpdump* output files (see Chapter 3).

In these applications, the packet sizes sent could be varied by loaded network conditions, under which the data would be buffered by the TCP layer implementation and will not be sent out until the acknowledgement of last outstanding packet has been received. In this case, the packet size distribution profiles would therefore show poor consistencies, however, the packet aggregations have some certain patterns to adhere to. We suggest an approach and test this to perform application detection on the Nagle-based applications.

In this chapter, the approach is explained and its feasibility experimentally is proven.

4.2 Methods of Aggregation Application Detection

Firstly, the methods discussed in this chapter are based on the results experimentally obtained in Chapter 3, i.e. that the original (under ideal network condition) packet size distributions of Nagle-based applications are consistent and detectable.

4.2.1 Methodology

During the operation of a Nagle-based application, the packet generation and the network condition variation are two independent events as information about one of them does not tell anything about the other one [HogMC06].

Let the original packet size be an independent discrete random variable X , and the network condition be the second independent discrete random variable C . Due to the nature of the Nagle Algorithm, in every time interval, the possible values of C can be described as {condition that makes no aggregation, condition that the delay is enough for 2 packets being aggregated, condition that the delay is enough for 3 packets being aggregated but not for 4 packets, condition that the delay is enough for 4 packets being aggregated but not for packets,}, or simply defined as $\{C1, C2, C3, C4, \dots\}$. Therefore, when aggregation occurs, we define a packet generated as an original packet as a 1st order packet, whilst we define a packet aggregated from 2 original packets as a 2nd order packet (from 3 original packets as 3rd order, etc). A packet size distribution can certainly consist of different order packets, however, the presence of each packet will not affect the generation of the others. As such, let us simply consider the probability of a single packet generation.

Under a loaded network condition, in every time interval, the probability of the generation of a 1st order packet in an aggregated packet series, $P_n^{1st-agg}$ is:

$$P_n^{1st-agg} = P^{orig}(n) \cdot P(C_1) \quad (4.1)$$

where $P_n^{orig}(n)$ is the probability of the generation of a n byte original packet. If the generation of every single packet is an independent event which means that the generated

packet size has no relation to the last packet generated, then the probability of a 2nd order packet in an aggregated distribution with a given packet size n $P_n^{2nd-agg}$ being generated can be written:

$$P_n^{2nd-agg} = \sum_{k=0}^{k \leq n} P_k^{orig} \cdot P_{n-k}^{orig} \cdot P(C_2) \quad (4.2)$$

while that for the 3rd order is:

$$P_n^{3rd-agg} = \sum_{k=0}^{k \leq n} P_k^{orig} \cdot P_{n-k}^{2nd} \cdot P(C_3) \quad (4.3)$$

the equations for other orders can be derived in the same way. As a result, the probability of a given packet size n under a certain network condition P_n^{agg} is

$$P_n^{agg} = \sum_{i=0}^j P_n^{i^{th}-agg} \quad (4.4)$$

where $P_n^{i^{th}-agg}$ is the probability for the i^{th} order, and j is the maximum order of aggregation.

Let the probability of an aggregated packet without consideration of the aggregating probability $P(C)$ be $P_n^{i^{th}}$, then

$$P_n^{1st} = P_n^{orig}$$

and P_n^{2nd} is:

$$P_n^{2nd} = \sum_{k=0}^{k \leq n} P_k^{1st} \cdot P_{n-k}^{1st}$$

P_n^{3rd} is:

$$P_n^{3rd} = \sum_{k=0}^{k \leq n} P_k^{1st} \cdot \left(\sum_{l=0}^{l \leq n-k} P_l^{1st} \cdot P_{n-k-l}^{1st} \right)$$

$P_n^{i^{th}}$ can thus be iteratively written as:

$$p_n^{ith} = \sum_{k=0}^{k \leq n} p_k^{1st} \cdot p_{n-k}^{(i-1)th}$$

or

$$p_n^{ith} = \sum_{k=0}^{k \leq n} p_k^{orig} \cdot p_{n-k}^{(i-1)th} \quad (i > 1) \quad (4.5)$$

and rewrite (4.1), (4.2), (4.3):

$$p_n^{1st-agg} = p_n^{1st} \cdot p(C_1) \quad (4.6)$$

$$p_n^{2nd-agg} = p_n^{2nd} \cdot p(C_2) \quad (4.7)$$

$$p_n^{3rd-agg} = p_n^{3rd} \cdot p(C_3) \quad (4.8)$$

Precedent research suggests that, for a real-time application, 300ms is a threshold delay that a user can tolerate [Far02]. Therefore a maximum order of 4 was observed as the greatest aggregation order number under this network condition and was adopted for this work. The delay emulated in all tests was limited to 300ms. However, consideration for higher order packets and greater delay could be made.

The packets generated by applications represent a time series [Kan076]. They are sometimes generated in an orderly manner [Far02]. For instance, a packet of 31 bytes may always follow a packet of 30 bytes, or packets of 10, 35 and 50 bytes may always appear continuously in the order of “35bytes, 10 bytes and 50 bytes”. This is explainable. For example, in an application, on events, e.g. the event1, which sends out a 30-byte packet, will trigger an event2, which generates a 31-byte packet. As such, the generation of every single packet would become a conditional probability event [How04]. This circumstance must be considered as it may occur in practice. In this case, let the original

probability of an n byte packet be $p_n^{orig-con}$, then (4.6), (4.7), and (4.8) can be rewritten:

$$p_n^{1st-agg-con} = p_n^{orig-con} \cdot p(C_1) \quad (4.9)$$

$$p_n^{2nd-agg-con} = \sum_{k=0}^{k \leq n} p_k^{orig-con} \cdot p_{n-k}^{orig-con} \cdot p(C_2) \quad (4.10)$$

$$p_n^{3rd-agg-con} = \sum_{k=0}^{k \leq n} p_k^{orig-con} \cdot p_{n-k}^{2nd-agg-con} \cdot p(C_3) \quad (4.11)$$

and (4.4) here is

$$p_n^{agg-con} = \sum_{i=0}^i p_n^{ith-agg-con} \quad (4.12)$$

For an aggregated packet size distribution $Dist^{agg}\{Count_1^{agg}, Count_2^{agg}, Count_3^{agg}, \dots, Count_{1460}^{agg}\}$

where $Count_n^{agg}$ is the number of packets of n bytes, we have:

$$Count_n^{agg} = Sum^{agg} \cdot (p_n^{1st-agg} + p_n^{2nd-agg} + p_n^{3rd-agg} + p_n^{4th-agg}) \quad (4.13)$$

where Sum^{agg} is the total number of packets in an aggregation distribution. Since the total number of packets of the i th order is:

$$Sum^{ith} = Sum^{agg} \cdot P(C_i) \quad (4.14)$$

(4.13) hence is:

$$Count_n^{agg} = Sum^{1st} \cdot p_n^{1st} + Sum^{2nd} \cdot p_n^{2nd} + Sum^{3rd} \cdot p_n^{3rd} + Sum^{4th} \cdot p_n^{4th} \quad (4.15)$$

As such, with the availability of Sum^{ith} , using (4.15), the aggregated packet size distribution of a known application under a given network condition can be theoretically calculated. Alternatively, the original probability of a given packet size existing in an aggregated packet size series can be extracted according to this equation.

Now, it is necessary to investigate the aggregated packet size distribution profiles to seek a way to obtain Sum^{ith} .

The plot in **Figure 4.1** gives the profile of WarCraft III under a heavy-load network condition. Let it be an unknown aggregated packet size distribution, which is defined as

$$Dist^{agg}\{Count_1^{agg}, Count_2^{agg}, Count_3^{agg}, \dots\}$$

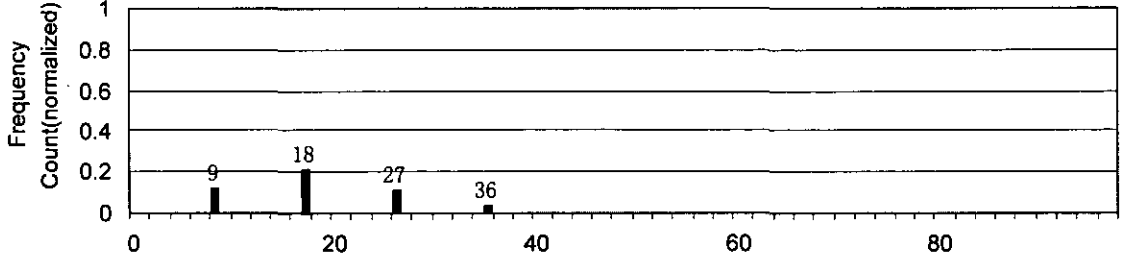


Figure 4.1 Packet size distribution for WarCraft III under heavy-load network condition

Packet Size	1	2	3	4	5	6	7
Probability	0	0	0	0	0	0	0
Packet Size	8	9	10	11	12	13	14
Probability	0	0.9770	0.0002	0	0	0	0.0012
Packet Size	15	16	17	18	19	20	21
Probability	0	0	0	0	0.0003	0.0003	0
Packet Size	22	23	24	25	26	27	28
Probability	0.0003	0.0028	0	0	0	0.0003	0

Table 4.1 Original packet size distribution values of first 28 bins for WarCraft III

Now, assume that $Dist^{agg}$ is aggregated from the original distribution of WarCraft III. The normalized distributions for different order of WarCraft III are

$$Dist^{1st} \{p_1^{1st}, p_2^{1st}, p_3^{1st}, \dots\}$$

$$Dist^{2nd} \{p_1^{2nd}, p_2^{2nd}, p_3^{2nd}, \dots\}$$

$$Dist^{3rd} \{p_1^{3rd}, p_2^{3rd}, p_3^{3rd}, \dots\}$$

$$Dist^{4th} \{p_1^{4th}, p_2^{4th}, p_3^{4th}, \dots\}$$

Obviously, the bin number N whose value is non-zero for the (i) th order packet size distribution is

$$N \in \{i \cdot Min\{n\}, i \cdot Max\{n\}\} \quad (4.16)$$

where n is a non-zero bin in the original distribution.

Table 4.1 gives the probabilities of the first 28 bins of WarcraftIII (Cli-Srv) under ideal network condition. More than 97% of the packets are generated as 9 bytes and 9 bytes is the smallest packet size observed. According to **Table 4.1**, the $\text{Min}\{n\}$ of WarCraft III is 9, so that the 2nd order packet size would all be greater or equal to 18 bytes, and all 9 bytes packets should have been generated as original packets which we refer to as a bin of 1st order. Thus, (4.14) here is:

$$\text{Count}_9^{\text{agg}} = \text{Sum}^{1\text{st}} \cdot p_9^{1\text{st}} \quad (4.17)$$

and we have

$$\text{Sum}^{1\text{st}} = \text{Count}_9^{\text{agg}} / p_9^{1\text{st}} \quad (4.18)$$

Thus, a theoretical packet size distribution for original packets

$$\text{Dist}^{1\text{st-theory}} \{ \text{Count}_1^{1\text{st-theory}}, \text{Count}_2^{1\text{st-theory}}, \text{Count}_3^{1\text{st-theory}}, \dots \}^*$$

can be calculated using

$$\text{Count}_n^{1\text{st-theory}} = p_n^{1\text{st}} \cdot \text{Count}_9^{\text{agg}} / p_9^{1\text{st}} \quad (4.19)$$

where $\text{Count}_n^{1\text{st-theory}}$ is the number of original(or named as 1st order) packets of n bytes, $\text{Count}_9^{\text{agg}}$ is the number of packets of 9 bytes in the unknown aggregated distribution.

Afterwards, remove all original packets using

$$\text{Count}_n^{\text{temp}} = \text{Count}_n^{\text{agg}} - \text{Count}_n^{1\text{st-theory}}$$

where $\text{Count}_n^{\text{agg}}$ is the number of the aggregated packets of n bytes. So far, an intermediate distribution

$$\text{Dist}^{\text{temp}} \{ \text{Count}_1^{\text{temp}}, \text{Count}_2^{\text{temp}}, \text{Count}_3^{\text{temp}}, \dots \}$$

theoretically without any original packet has been obtained. Again, $Min\{n\}$ of WarCraft III is 9, according to (4.16), the 3rd order packet size would be all equal to or greater than 27 bytes, so all 18 bytes packets in $Dist^{temp}$ should have been generated as 2nd order packets. Thus, we refer to the bin of 2nd order equal to 18, and the packet size distribution for those 2nd packets

$$Dist^{2nd-theory} \{Count_1^{2nd-theory}, Count_2^{2nd-theory}, Count_3^{2nd-theory}, \dots\}$$

can be calculated using

$$Count_n^{2nd-theory} = P_n^{2nd} \cdot Count_{18}^{temp} / P_{18}^{2nd} \quad (4.20)$$

a $Dist^{temp}$ without original or 2nd order packet is therefore obtained using

$$Count_n^{temp} = Count_n^{agg} - Count_n^{2nd-theory}$$

We repeat the steps above until the 4th order has been reached. Now, we have gained four theoretical packet size distributions

$$Dist^{1st-theory} \{Count_1^{1st-theory}, Count_2^{1st-theory}, Count_3^{1st-theory}, \dots\}$$

$$Dist^{2nd-theory} \{Count_1^{2nd-theory}, Count_2^{2nd-theory}, Count_3^{2nd-theory}, \dots\}$$

$$Dist^{3rd-theory} \{Count_1^{3rd-theory}, Count_2^{3rd-theory}, Count_3^{3rd-theory}, \dots\}$$

$$Dist^{4th-theory} \{Count_1^{4th-theory}, Count_2^{4th-theory}, Count_3^{4th-theory}, \dots\}$$

Using

$$Count_n^{theory} = Count_n^{1st-theory} + Count_n^{2nd-theory} + Count_n^{3rd-theory} + Count_n^{4th-theory} \quad (4.21)$$

a theoretical aggregated distribution

$$Dist^{theory} \{Count_1^{theory}, Count_2^{theory}, Count_3^{theory}, \dots\}$$

can be obtained and it should be theoretically equal to the unknown aggregated distribution $Dist^{agg}$.

When an unknown aggregated distribution has been obtained, one could assume that it is aggregated from a certain known original distribution $Dist^{orig}$, and the same method as above applied to it to calculate $Dist^{theory}$. $Dist^{theory}$ can be described as “how should the assumed distribution theoretically aggregate under the network condition which the captured aggregated distribution had suffered”. Afterwards, a Chi-square test is carried out on $Dist^{theory}$ against the captured aggregated distribution with the null hypothesis that “ $Dist^{theory}$ is aggregated from the assumed original distribution $Dist^{orig}$ ”. Detections thus are achieved.

An alternative method to identify the application is to reverse the previous approach. Instead of obtaining $Dist^{theory}$, we try to extract the original packet size distribution from an aggregated one $Dist^{agg}$. Assume that an unknown aggregated distribution $Dist^{agg}$ is aggregated from a known original distribution $Dist^{orig}$. We have assumed that the maximum aggregation order is 4, thus, according to (4.16), all packets whose sizes are greater than $3 \cdot Max\{n\}$ should have been generated as 4th order packets. A referring bin r_4 can be then chosen from $\{3 \cdot Max\{n\}, 4 \cdot Max\{n\}\}$ to calculate

$$Dist^{4th-theory} \{Count_1^{4th-theory}, Count_2^{4th-theory}, Count_3^{4th-theory}, \dots\}$$

using:

$$Count_n^{4th-theory} = p_n^{4th} \cdot Count_{r_4}^{agg} / p_{r_4}^{4th} \quad (4.22)$$

We then remove it from $Dist^{agg}$ using

$$Count_n^{temp} = Count_n^{agg} - Count_n^{4th-theory} \quad (4.23)$$

So far, $Dist^{temp} \{Count_1^{temp}, Count_2^{temp}, Count_3^{temp}, \dots\}$ should theoretically contain no 4th order packets, hence the 3rd order referring bin r_3 can be chosen from $\{2 \cdot Max\{n\}, 3 \cdot Max\{n\}\}$ to calculate

$$Dist^{3rd-theory} \{Count_1^{3rd-theory}, Count_2^{3rd-theory}, Count_3^{3rd-theory}, \dots\}$$

using

$$Count_n^{3rd-theory} = P_n^{3rd} \cdot Count_{r_3}^{temp} / P_{r_3}^{3rd} \quad (4.24)$$

and remove it from $Dist^{temp}$:

$$Count_n^{temp} = Count_n^{temp} - Count_n^{3rd-theory} \quad (4.25)$$

We repeat these steps until the 2nd order theoretical distribution has been removed. $Dist^{temp}$ therefore can be renamed as $Dist^{1st-theory}$ since it should theoretically consist of only 1st order packets, and a Chi-square test can be undertaken on $Dist^{1st-theory}$ and $Dist^{orig}$ with a null hypothesis that “ $Dist^{agg}$ is aggregated from the assumed original distribution $Dist^{orig}$ ”.

Here, we have a problem. When a captured aggregated packet size distribution mostly consists of high order packets, the extracted $Dist^{1st-theory}$ would be null or almost null and therefore difficult to identify. The solution to this problem is that when $Dist^{(i+1)th-theory}$ has been removed from $Dist^{temp}$, if there are no (or almost no) lower order packets in the distribution, $Dist^{temp}$ would theoretically contain only (or mostly) i th order packets, as such, $Dist^{temp}$ should comply with the i th order distribution $Dist^{th}$ of the assumed application. Hence, detection can be achieved by comparing $Dist^{temp}$ against $Dist^{th}$ every time $Dist^{(i+1)th-theory}$ has just been removed from $Dist^{temp}$.

4.2.2 Selection of Referring Bins

The referring bin for the i th order can be selected from

$$\{(i-1) \cdot \text{Min}\{n\}, i \cdot \text{Min}\{n\}\}$$

for the first method or from

$$\{\text{Max}\{n\}, (i+1) \cdot \text{Max}\{n\}\}$$

for the second method. However, since the generation of packet is a random event, the packet size distribution may vary from case to case, therefore not every bin in the range is suitable for use as a referring bin.

As the probability of each bin is different from each other, it is possible that, if the probability of a certain bin is very low, a packet with a size of this bin will not appear in a sampling interval. Also, if a low probability bin is chosen, the calculated theoretical distribution may vary considerably with a tiny variation of the number of packets with sizes of that bin from one sampling interval to another. For instance, for WarCraft III, if 10 bytes is chosen, in the first captured distribution, 2 packets of this sizes and 9770 9-byte packets are observed, in the second one, 4 10-byte and 9500 9-byte packets are generated. In both captured distributions, sample population are 10000. The variation of the number of the 10-byte packets can be considered as “very tiny (from 2 to 4)” compared to that of the 9-byte (from 9770 to 9500), the number of the 9-byte packets in the two calculated theoretical distributions however will vary 2 times ($4 / 2$). On the other hand, if 9-byte is chosen, the number of the 10-byte packets in the calculated theoretical distribution will only vary 1.028 times ($9770 / 9500$). As such, basically, a bin with the relatively high probability in the usable range should be adopted.

In some circumstances, all bins in the useable range are very small. In this case, several bins could be adopted and theoretical distribution calculated for every non-zero bin respectively, making the mean of values of all obtained theoretical distributions the value of the final theoretical distribution in order to improve the accuracy of the methods.

4.2.3 The Selection of Aggregation Method

To understand the reason that two methods have been proposed, it is necessary to explain a simple example.

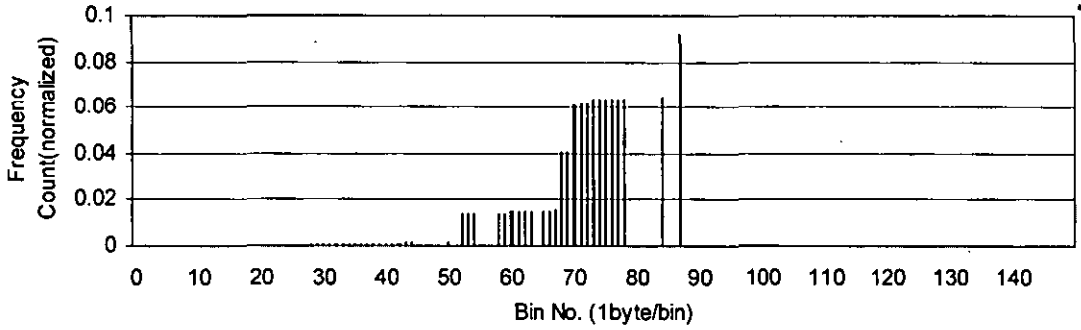


Figure 4.2 A sample original distribution, which needs to use the second method

In the plot given by **Figure 4.2**, the general shape of the original (or 1st order) packet size distribution is quite different from that of WarCraft III, which shows an increasing trend. In this sample, the original distribution spreads from 25 to 90, according to (4.16), the usable range of referring bin for the 1st order is $\{0, 50\}$, however, all bins in this range represent too small values to utilise the first method. On the other hand, as figure 4.3 shows, in the usable range of referring bins for the 4th order $\{270, 360\}$, 3rd order $\{180, 270\}$, 2nd order $\{90, 180\}$, the values of bins are much more reasonable, hence, it is easy to chose referring bins for every order. As such, the second method is suitable for this original distribution profile.

In practice, suitable methods may vary for any given packet size distribution. It is necessary to decide which one should be adopted in respect of the actual distribution profiles of known applications.

4.3 Verification

To verify the feasibility of the proposed methods, a number of tests have been carried out, and in this section, these tests are described.

The tests can be divided into two catalogues: real applications and emulated applications. During the experiments, we had found that there were very few real Nagle-based applications in practice. However, one would expect that there will be more Nagle-based applications in the future [Pax94]. So, it is still necessary to verify the potential feasibility of the methods mentioned above. A Nagle-based application emulator had been developed in order to investigate the performance of the methods to identify a variety of aggregated distributions.

4.3.1 The Generation of Sample Original Distribution for Virtual Applications

Of course, we cannot test all possible distributions. In order to verify that the methods are generally feasible to all kinds of distributions, a hundred typical sample distributions were generated. Three parameters were used to describe a distribution: distribution trend which give the general shape; median position and spread, which quantitatively define the profile position and range of non-zero bins of the distribution as shown in **Figure 4.3**. The chosen values for these three parameters are summarized below in **Table 4.2**

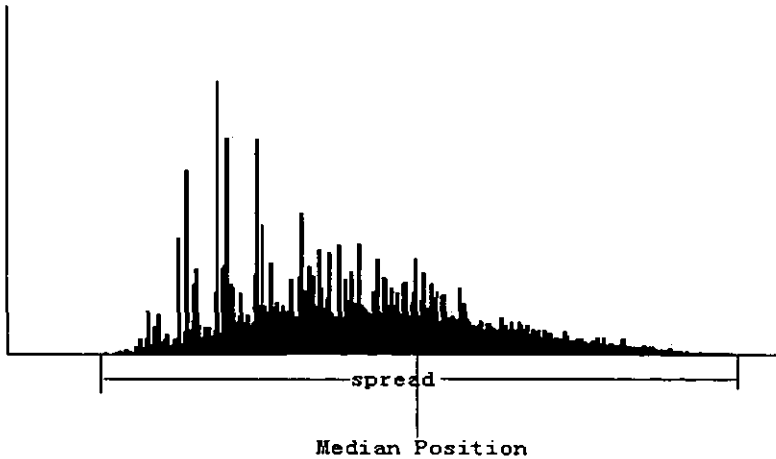


Figure 4.3 Parameters to describe a distribution

Trends	Increasing, Decreasing, Concave, Protruding, Random
Median Position	10, 20, 50, 100, 200, 500
Spread	10, 20, 50, 100, 200

Table 4.2 Chosen parameters for virtual application distributions

The distances and value differences between two neighbour bins were generated randomly. Using the above, one hundred sample application distributions have been obtained.

4.3.2 Nagle-Based Application Emulator

The purpose of this tool is to emulate the packet generations of virtual Nagle-based applications. It uses Client/Server architecture. Since the Nagle Algorithm is generally performed by default by all the TCP implementations, the Nagle Algorithm is in operation when the communication socket between the Client and the Server is established as long as we do not set the *no_delay* option while programming the socket.

Both ordered and non-ordered packet series can be generated based on the user's input. When the pattern option is set as non-ordered, the emulator will simply generate a series of packets using a random number generator, which complies with a user-input

distribution. The packet size distribution so generated may show variation from the input if a finite number of packets were generated, but, if the generated packet number is infinite, the distribution would be no different from the input. If the ordered option was set, the emulator would firstly seek a user-input table (Table 4.3 gives one line of an example table) for the probability of the next packet size according to the size of the last packet generated. If no ordered information for the last packet size is available or the random generator return value shows that no ordered packet should be generated, the emulator will simply generate a packet which complies with the user-input distribution, otherwise, an ordered packet will be sent with the probability from the given table.

30 bytes (Last Packet Size)	Next Packet Size	15 bytes	22 bytes	45 bytes	60 bytes	Others
	probability	15%	10%	5%	40%	30%

Table 4.3 A sample of ordered packet series table

4.3.3 Tests of Real Nagle-Based Applications

Two real applications in this category are currently available----WarCraft III and SSH-Client. These tests used the same experimental system as that described in Chapter 3. For WarCraft III, both traffic in the client to server and server to client directions were captured while only client to server direction traffic for SSH-client is considered. The packet size distributions were based on 1000 packet samples. All original distributions of real applications and virtual applications had been stored in the database in advance so that a captured distribution of the test application can be calculated and compared against all the others.

Network Game WarCraft III**Network Condition Emulated**

Packet Delay (ms)	150
-------------------	-----

The original packet size distribution profiles of WarCraft III in both directions are represented in **Figure 4.4** and **Figure 4.5**.

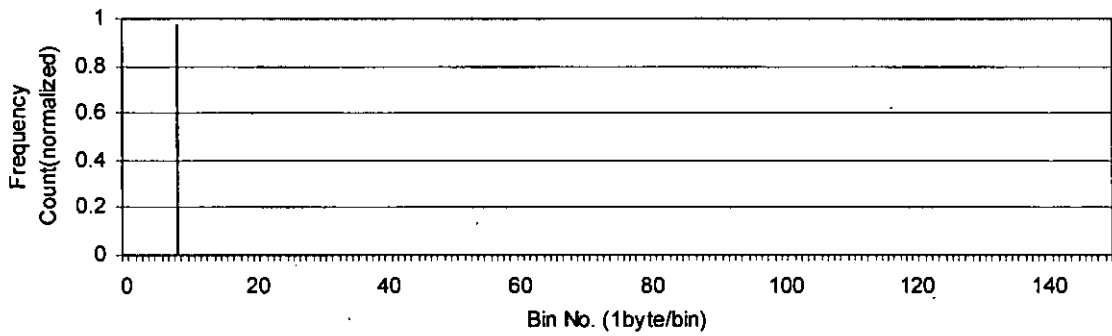


Figure 4.4 Original packet size distribution of WarCraft III (cli to srv)

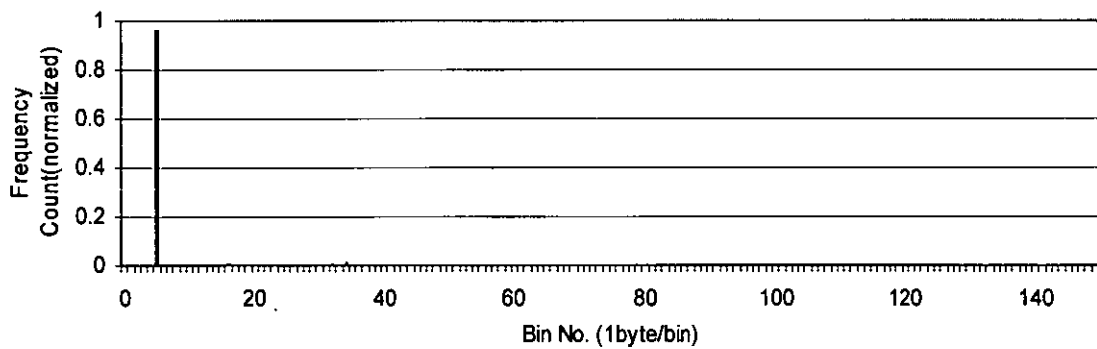


Figure 4.5 Original packet size distribution of WarCraft III (srv to cli)

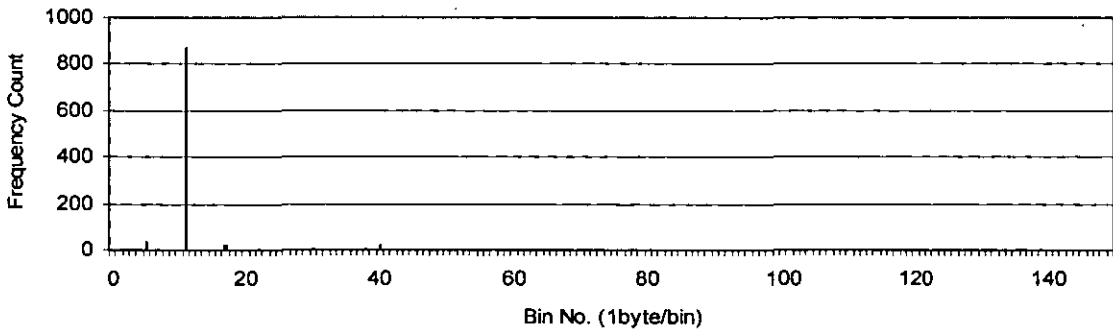


Figure 4.6 Captured aggregated distribution for WarCraft III under the network condition of 150ms delay (srv to cli)

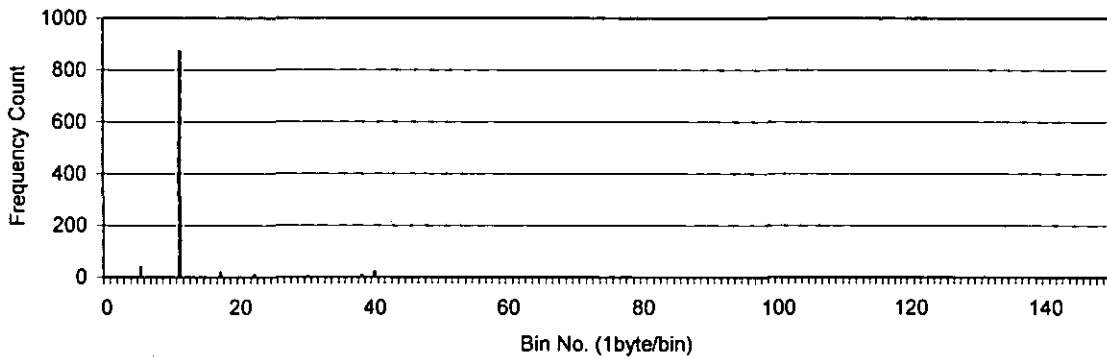


Figure 4.7 Theoretical aggregated distribution calculated from the original distribution of WarCraft III (cli to srv)

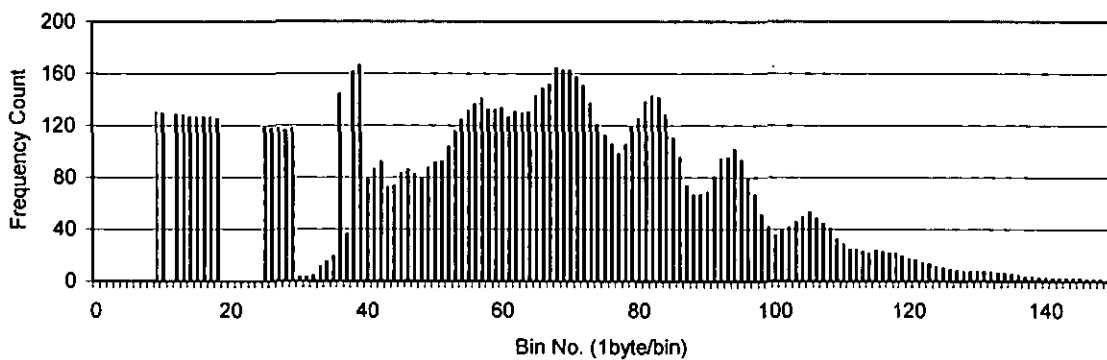


Figure 4.8 Theoretical aggregated distribution calculated from the original distribution of another virtual application (cli to srv)

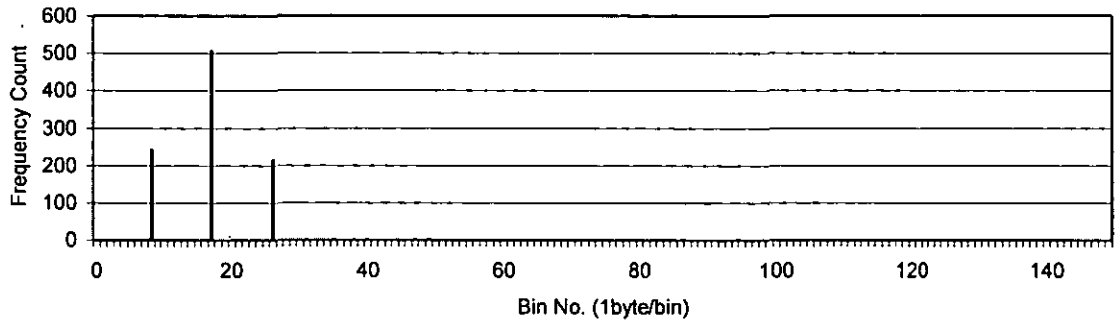


Figure 4.9 Captured aggregated distribution for WarCraft III under the network condition of 150ms delay (srv to cli)

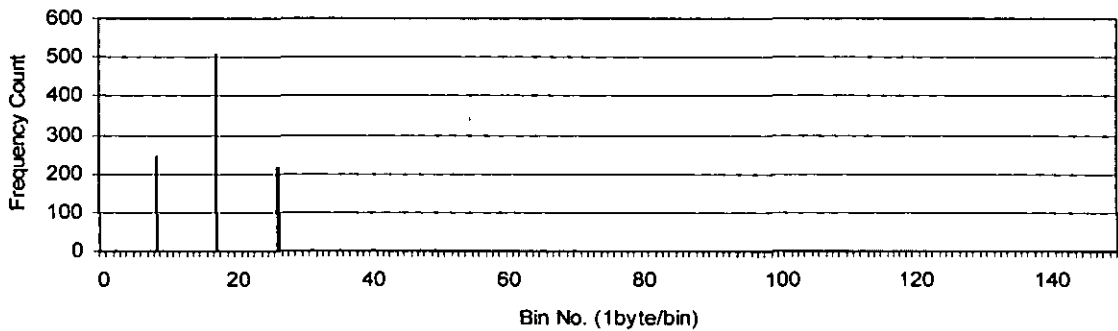


Figure 4.10 Calculated theoretical aggregated distribution from the original distribution of WarCraft III (cli to srv)

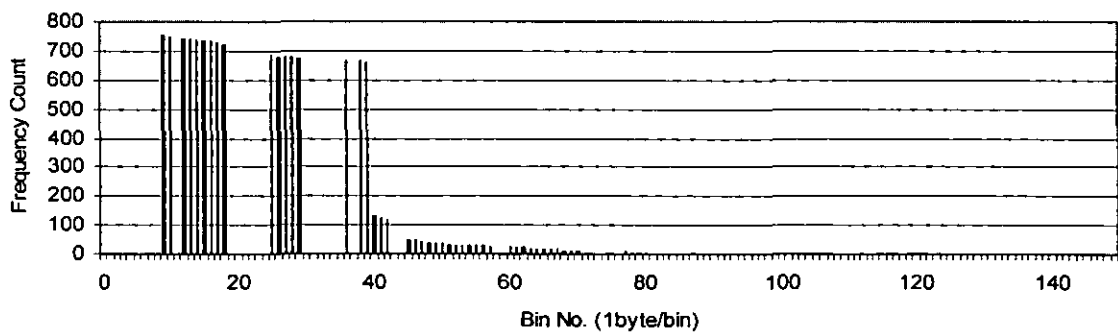
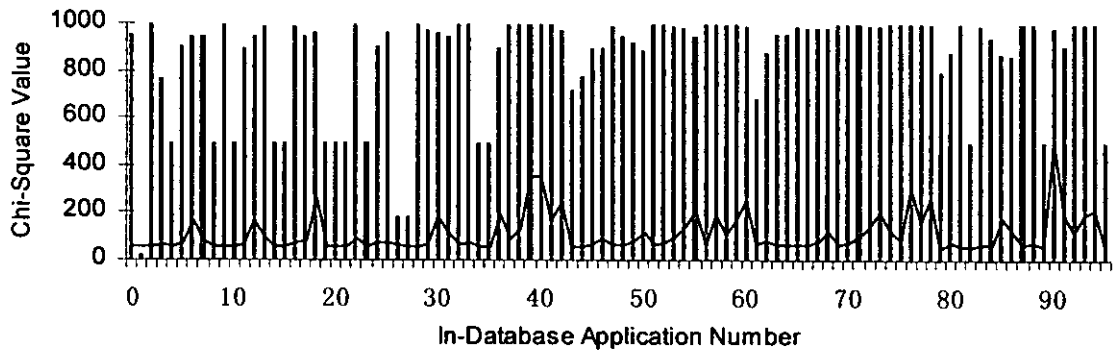
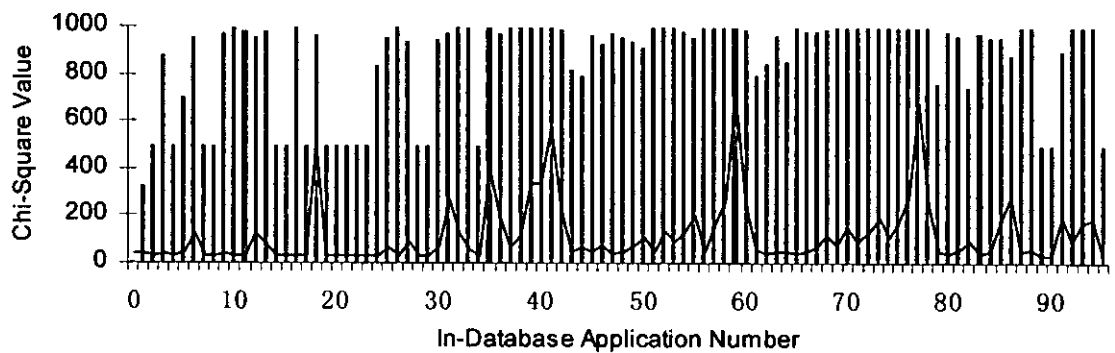


Figure 4.11 Theoretical aggregated distribution calculated from the original distribution of another virtual application (cli to srv)

For WarCraft III, the theoretical aggregation method was employed. In **Figure 4.6**, the plot gives the captured packet size distribution using *Tcpdump*. A peak appeared at 12 bytes with a minor peak at 6 bytes, and some 18 byte packets were also observed. It appears that packets from the 1st, 2nd and 3rd order were all present in the sampled distribution. **Figure 4.7** then gives the calculated theoretical distribution profile from the original packet size distribution of WarCraft III. Another plot (**Figure 4.8**) which shows the calculated theoretical distribution profile with a virtual application (Decreasing, Median Position 50, and Spread 50) is also given here. It can be visually seen that the theoretical profile calculated from the original distribution of WarCraft III was very similar with the sampled one whilst the other distribution from the virtual application is not. The reverse direction presented the same result as indicated by **Figure 4.9** to **4.11**. The following plot (**Figure 4.12**) gives the result of the Chi-square tests on the captured distributions of traffic from both directions against all the calculated theoretical distributions from the in-database distributions generated by the Nagle application emulator.



a



b

Figure 4.12 Summarization of Chi-square tests for identification of WarCraft III

a. Client to Server

b. Server to Client

The red curves represent the Critical Value with 95% confidence. It can be seen that in both directions, only at the In-Database Application Numbers for WarCraft III-Server-to-Client (1) or WarCraft III-Client-to-Server (2) have the Chi-square values lower than the critical value.

Network Condition Emulated

Packet Delay (ms)	300
-------------------	-----

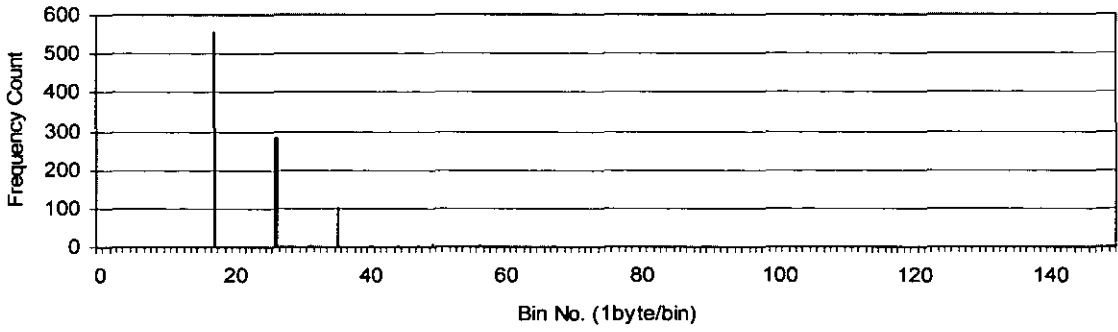


Figure 4.13 Captured aggregated distribution for WarCraft III under the network condition of 300ms fixed delay (srv to cli)

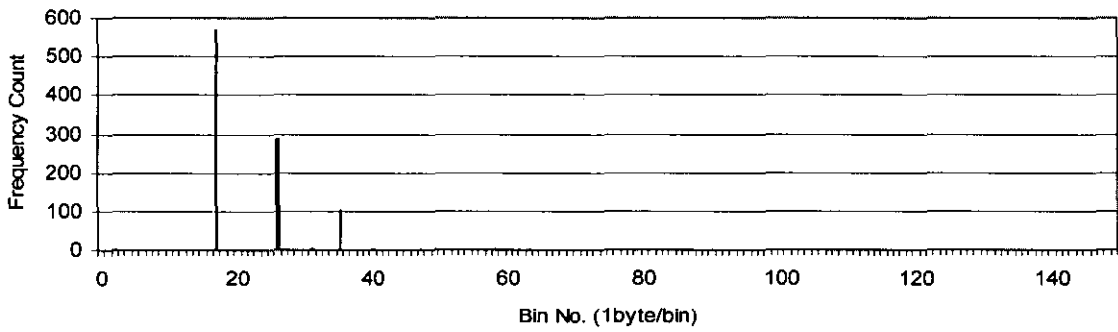


Figure 4.14 Theoretical aggregated distribution calculated from the original distribution of WarCraft III (srv to cli)

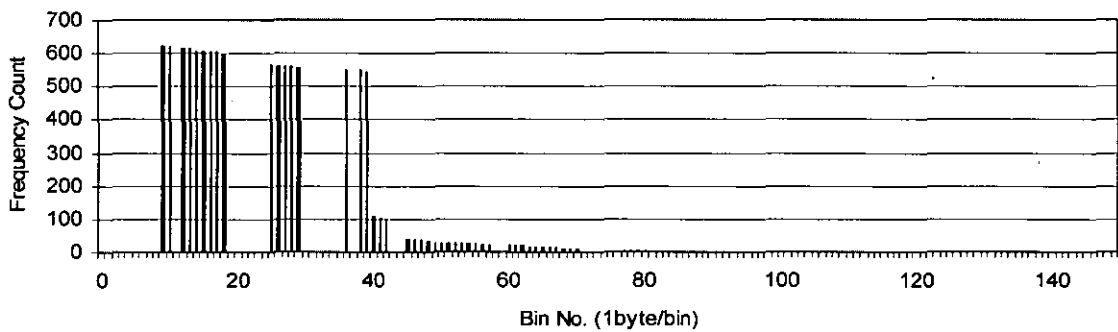


Figure 4.15 Theoretical aggregated distribution calculated from the original distribution of another virtual application (srv to cli)

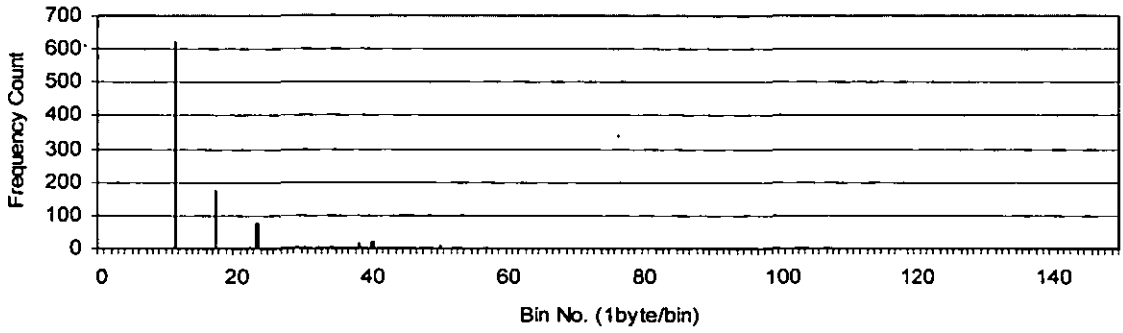


Figure 4.16 Captured aggregated distribution for WarCraft III under the network condition of 150ms delay (srv to cli)

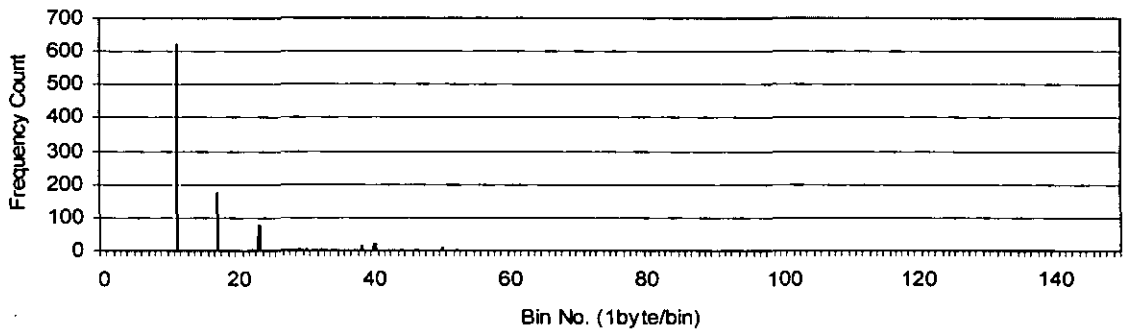


Figure 4.17 Theoretical aggregated distribution calculated from the original distribution of WarCraft III (srv to cli)

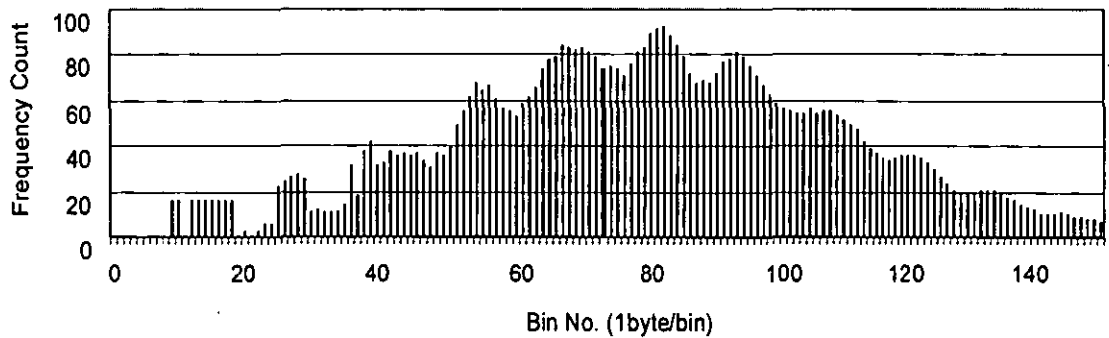


Figure 4.18 Theoretical aggregated distribution calculated from the original distribution of another virtual application (srv to cli)

Under this network condition, from **Figure 4.13**, very few 9 bytes packets had been observed, most packets were aggregated as 2nd or 3rd order packets. A similar circumstance was seen in the reversed direction (**Figure 4.16**) The method had successfully calculated the theoretical aggregated distribution from the original distribution of WarCraft III (as indicated in **Figure 4.14** and **Figure 4.17**) and mathematically differentiated from those calculated from other In-Database distributions as shown by the Chi-square analysis (**Figure 4.19**). Calculated theoretical aggregated distributions with another distribution are also given (**Figure 4.15** and **Figure 4.18**) here for visually comparison.

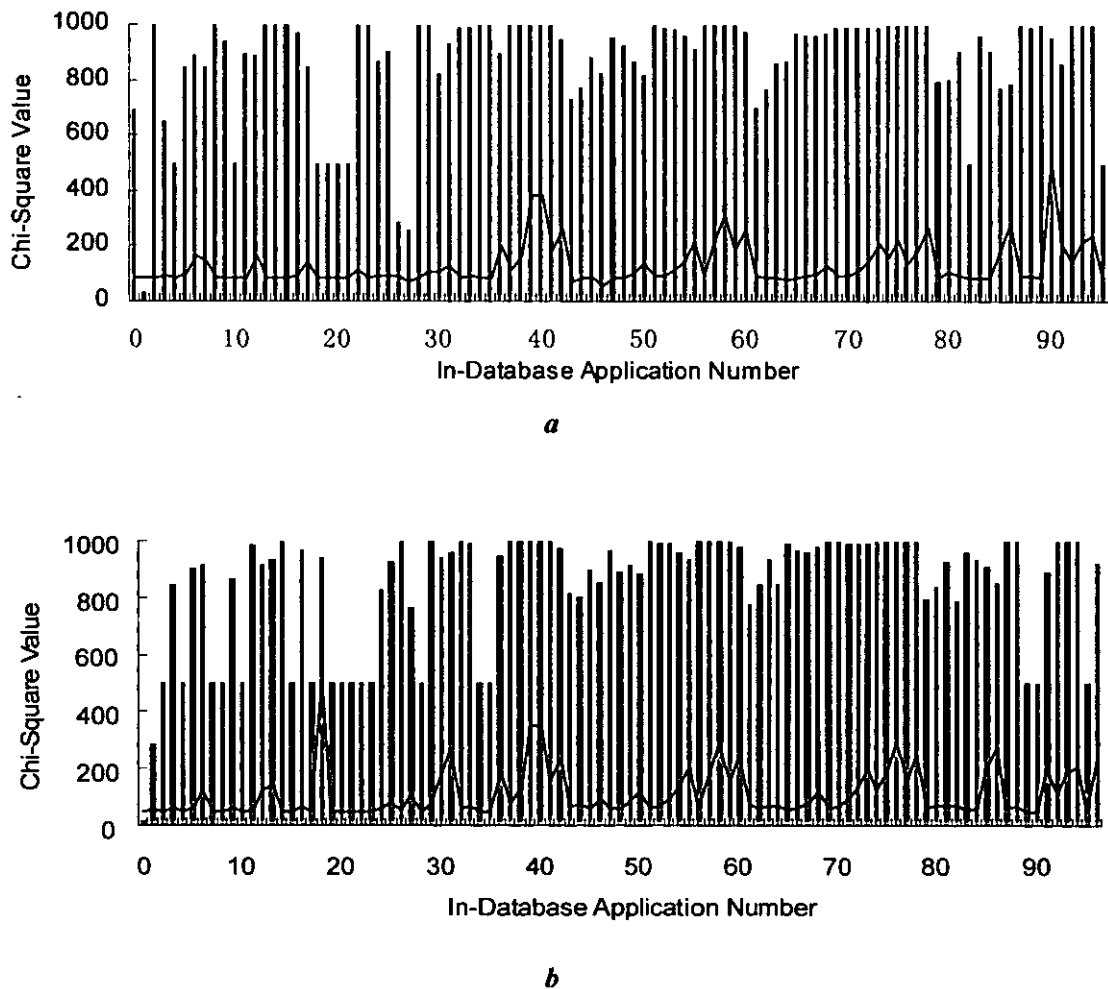


Figure 4.19 Summarization of Chi-square tests for identification of WarCraft III

- a.** Client to Server
- b.** Server to Client

Network Condition Emulated

Packet Loss Ratio (percentage)	5
Packet Delay (ms)	Jitter (50-300)

This network condition is closer to a WAN environment in practice. The delay was jittered from 50ms to 300ms randomly for every packet that passed through LNE.

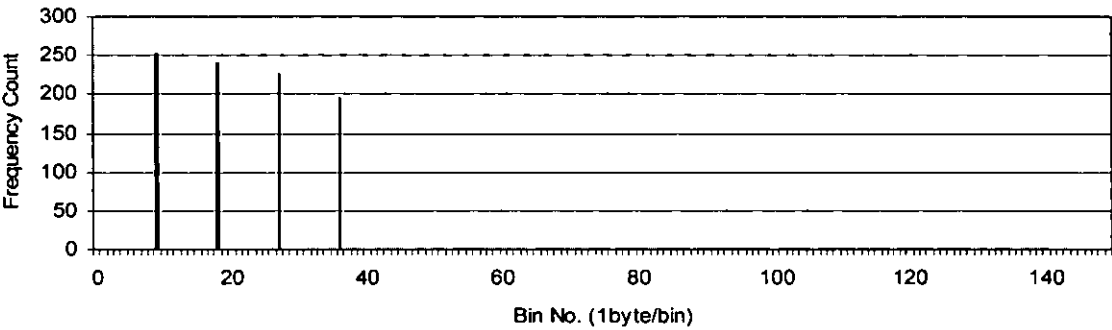


Figure 4.20 Captured aggregated distribution of WarCraft III under the network condition of 300ms delay (srv to cli)

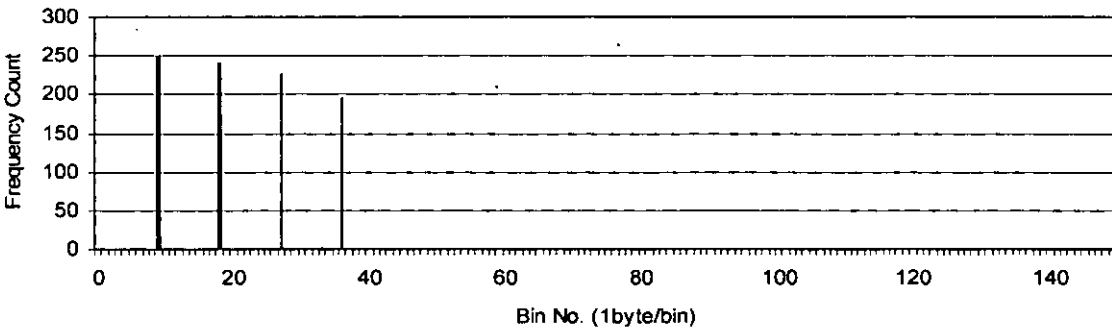


Figure 4.21 Theoretical aggregated distribution calculated from the original distribution of WarCraft III (cli to srv)

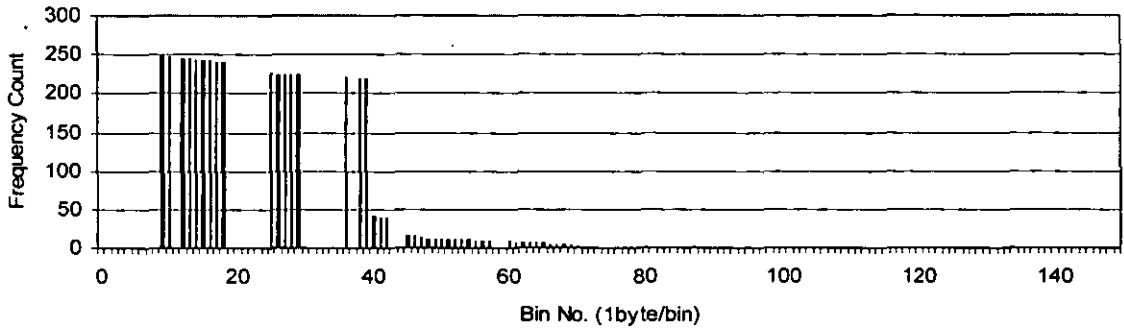


Figure 4.22 Theoretical aggregated distribution calculated from the original distribution of another virtual application (*cli to srv*)

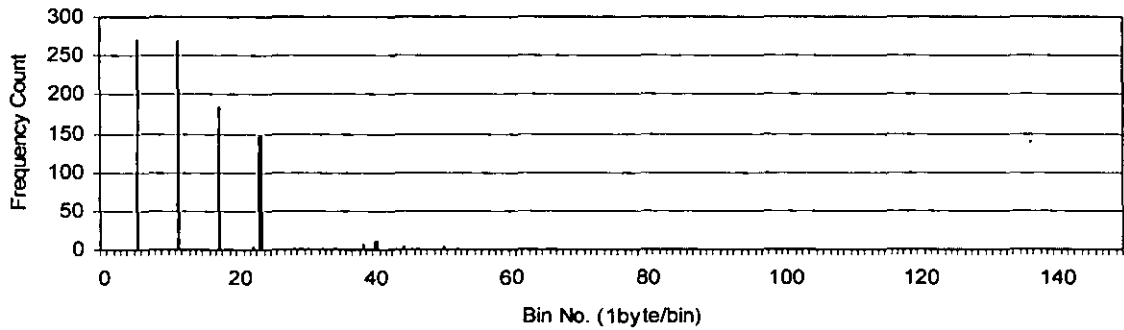


Figure 4.23 Captured aggregated distribution of WarCraft III under the network condition of jittered delay (*srv to cli*)

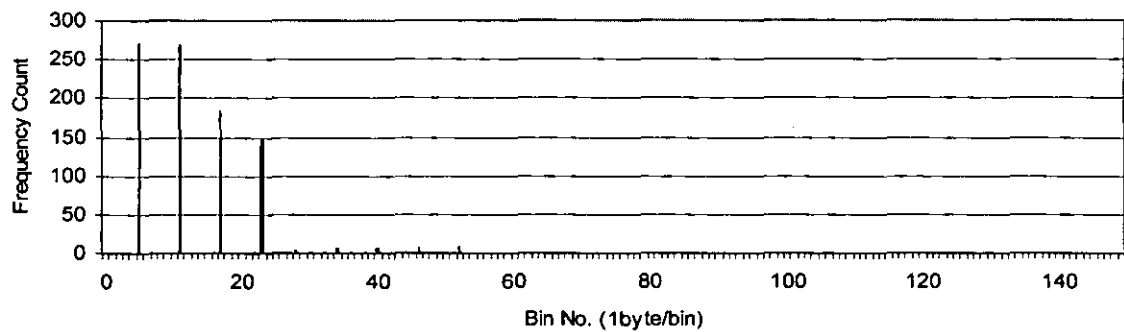


Figure 4.24 Theoretical aggregated distribution calculated from the original distribution of WarCraft III (*cli to srv*)

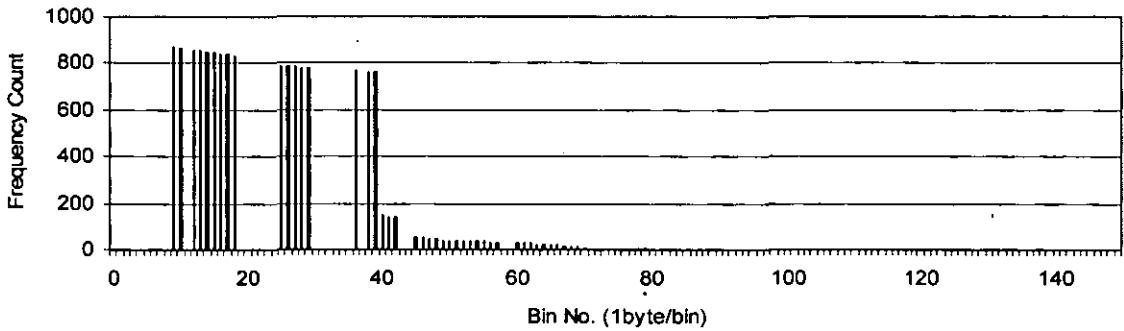
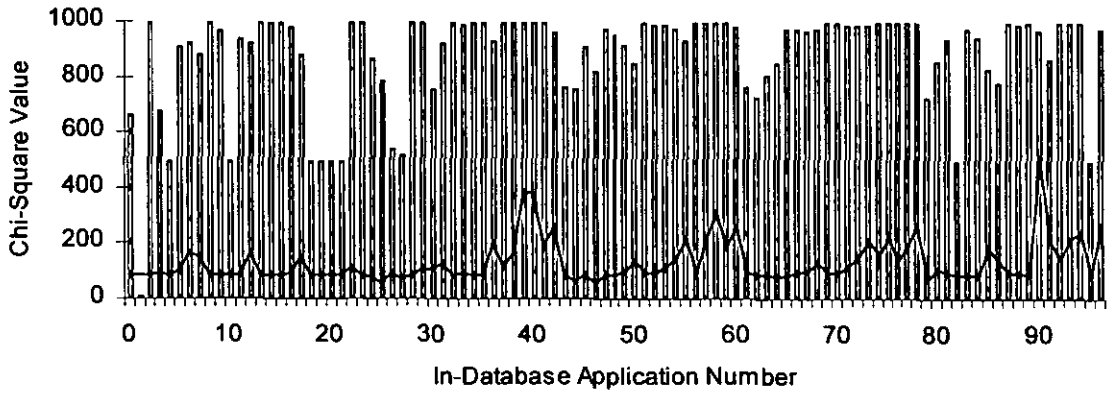
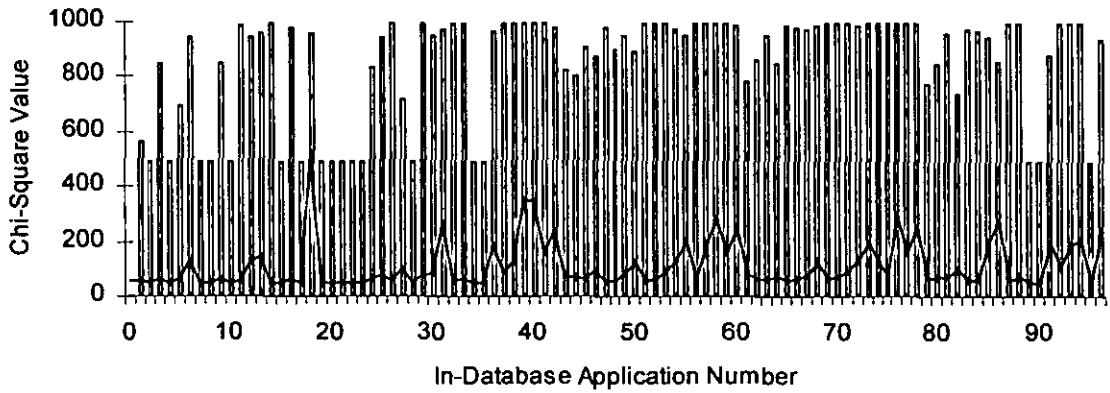


Figure 4.25 Theoretical aggregated distribution calculated from the original distribution of another virtual application (cli to srv)

Under this kind of reality-similar network condition, much more the high-order packets were observed. The numbers of 1st and 2nd order packets are very similar, with little decrease for the numbers of 3rd and 4th order packets (**Figure 4.20** and **Figure 4.23**). The analysis method had again generated theoretical aggregated packet size distributions, which were similar to the captured ones with the original packet size distributions of WarCraft III in both directions (**Figure 4.21** and **Figure 4.24**). Those generated using the incorrect original distribution did not (**Figure 4.23** and **Figure Figure 4.25**). The results of Chi-square tests are also summarized below (**Figure 4.26**). Again, the Chi-square values were only accepted at the application numbers of WarCraft III itself.



a



b

Figure 4.26 Summarization of Chi-square tests for identification of WarCraft III under network condition of jittered delay

a. Client to Server

b. Server to Client

SSH-Client

As indicated in Chapter 3, SSH-client generates a very simple original packet size distribution under ideal network condition, only 44 bytes packets were observed in the traffic captured (**Figure 4.27**). As such, only this bin and multiple of this bin can be chosen as the referring bins for the purpose of calculation. In fact, either the first or the second method is suitable for SSH-client. The second one was adopted here to test for real applications.

Network Condition Emulated

Packet Delay (ms)	300
-------------------	-----

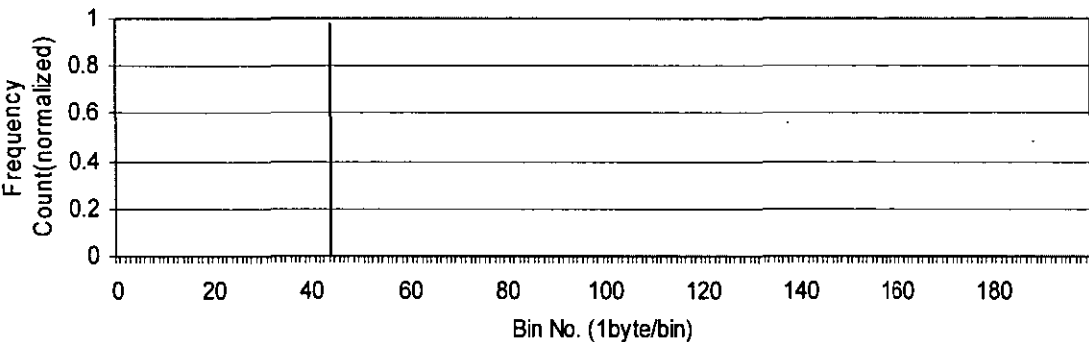


Figure 4.27 Original packet size distribution of SSH-Client III

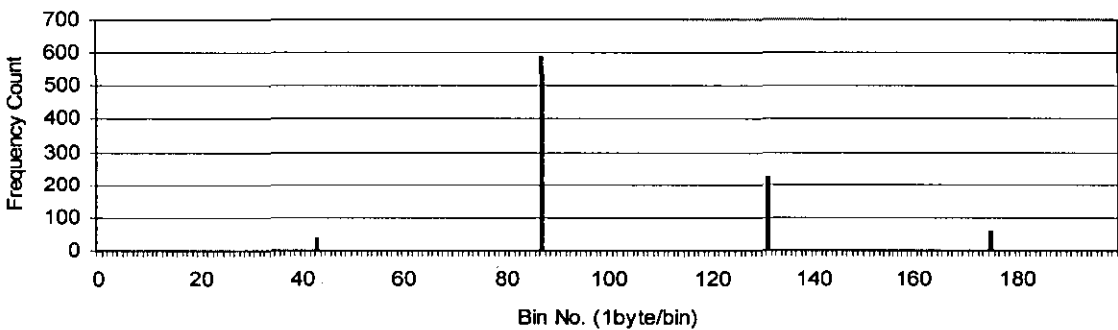


Figure 4.28 Captured aggregated distribution for SSH-Client under the network condition of 300 ms fixed delay

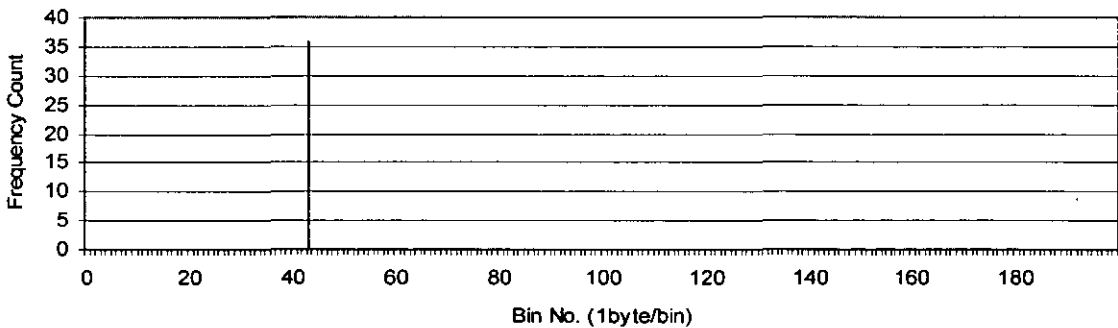


Figure 4.29 Theoretical original distribution calculated from the original distribution of SSH-Client

Under this network condition, similar to WarCraft III, very few packets were generated as original packets (**Figure 4.28**). The original packet size distribution of the captured distribution has been easily extracted (**Figure 4.29**) and identified using a Chi-square test with a Chi-square value 0, which means “completely match”. The results of the Chi-square tests are summarized below (**Figure 4.30**). Again, only at the SSH-Client, the lowest Chi-square value was observed and was lower than 95% confidence critical value.

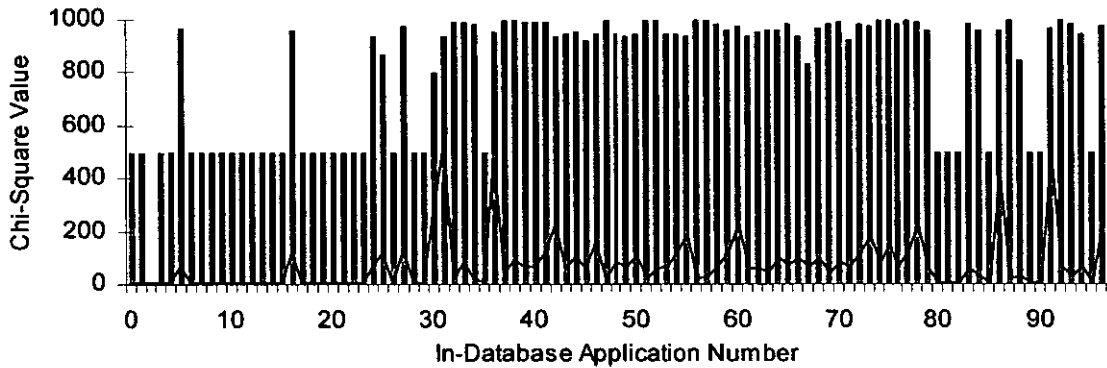


Figure 4.30 Summarization of Chi-square tests for identification of SSH-Client under network condition of 300ms fixed delay

Network Condition Emulated

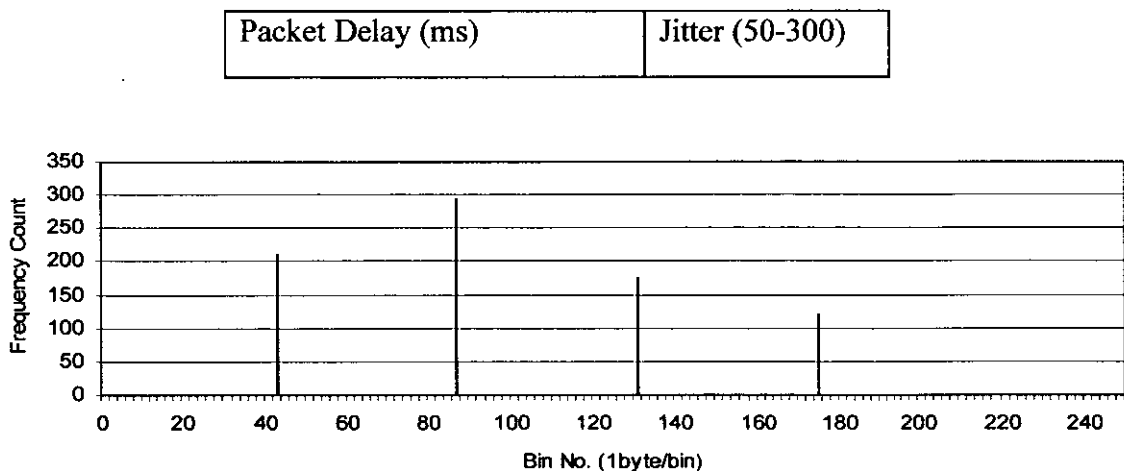


Figure 4.31 Captured aggregated distribution for SSH-Client under the network condition of jittered delay

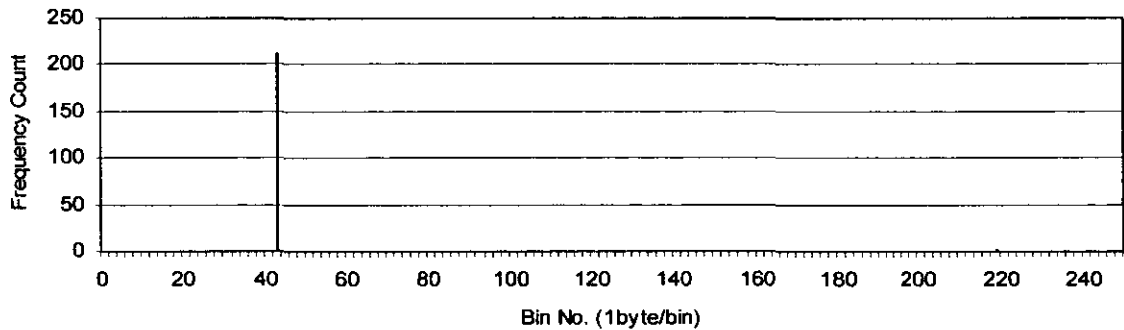


Figure 4.32 Theoretical original distribution calculated from the original distribution of SSH-Client

Under this network condition, packets are produced in all orders. Also one 5th order packet was observed as seen in **Figure 4.31**. The analysis extracted the theoretical original packet size distribution. However, the fifth order packet has not been removed as we had chosen 4 as the maximum order number (**Figure 4.32**). The Chi-square value therefore was not “completely matched”, but still acceptable with a critical value of 95%. In practice, more orders could be included in the SSH-client detection as its packet size distribution is very simple, the increment of orders would not significantly increase quantity of computation. The plot below shows the Chi-square test results obtained (**Figure 4.33**). The acceptance occurred at the application number of SSH-Client.

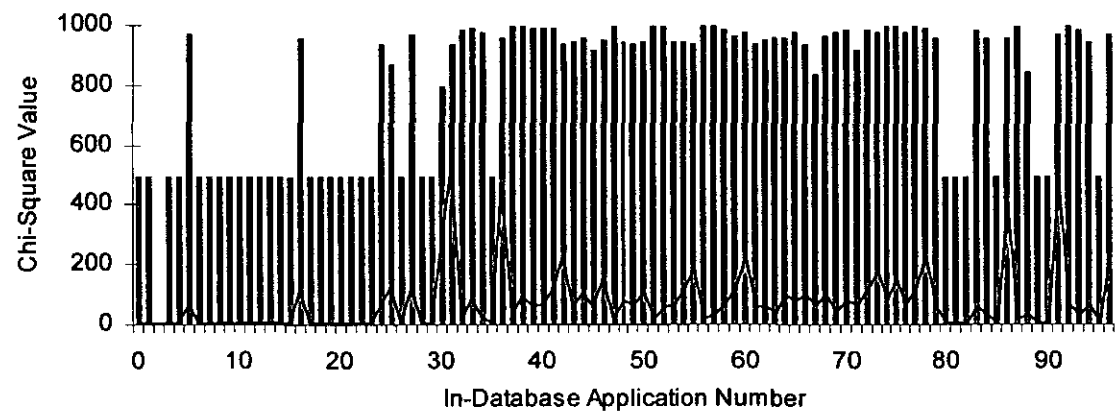


Figure 4.33 Summarization of Chi-square tests for identification of SSH-Client under network condition of jittered delay

4.3.4 Tests of Virtual Nagle-Based Applications

The testing objects in the following tests were virtual Nagle-based applications. We have selected about a hundred typical distributions and entered them into the virtual Nagle-based application emulator described in the last section. For every distribution, either random or ordered packet series were generated and captured based on 1000 packet samples. Three of them have been chosen for discussion here. The others showed similar results.

Virtual Application 1

The three parameters for the packet size distribution of this application and the original distribution profile are shown in **Figure 4.34**. The packet order attribute of this application was set as non-ordered with a packet rate of 5/sec. The reason for choosing this configuration is because it has a large spread which is thought relatively difficult to detect, in addition, for this application, the second method is preferred and the feasibility of this was verified. The original packet size distribution of this virtual application represents a general increasing shape of with a non-zero bin range of 1-122 as shown in **Figure 4.34**.

Trend	Increasing
Median Position	50
Spread	100
Packet Series	Non-Ordered
Id in Database	69

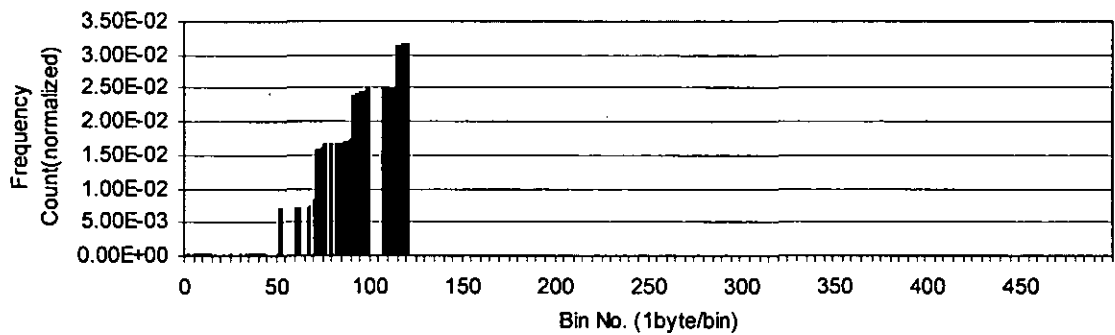


Figure 4.34 Parameters and profile of original distribution of virtual application 1

Network condition emulated

Packet Delay (ms)	300
-------------------	-----

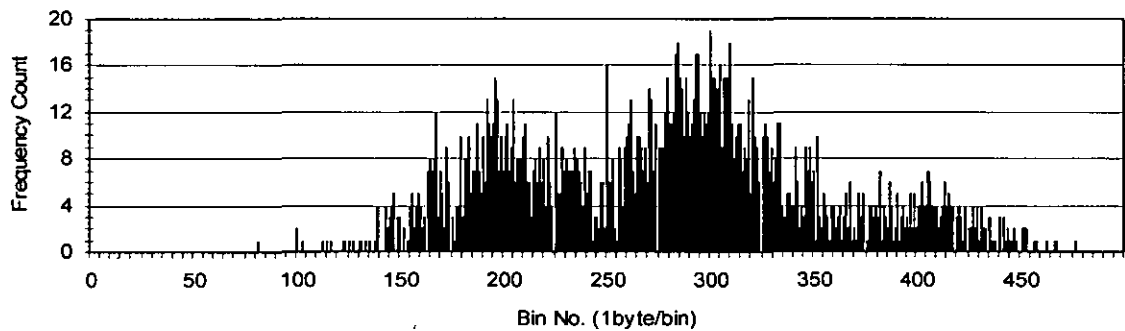


Figure 4.35 Captured aggregated distribution for virtual application 1 under the network condition of 300 ms fixed delay

Figure 4.35 shows the aggregated packet size distribution profile for virtual application 1 under the network condition of 300 ms fixed delay. The profile looks very complicated. In the range of 366-488 bytes between which packets could be from the 4th order, many packets had been observed. Quite a few low order (0-122) packets also appeared.

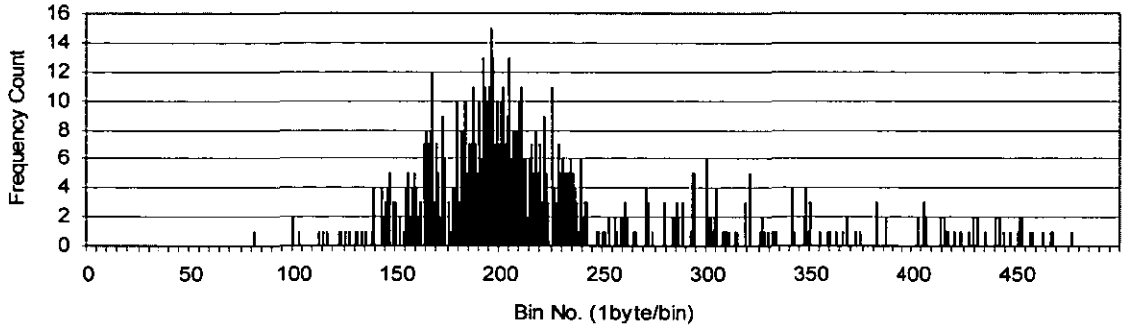


Figure 4.36 $Dist^{temp}$ after $Dist^{4th-theory}$ and $Dist^{3rd-theory}$ were removed

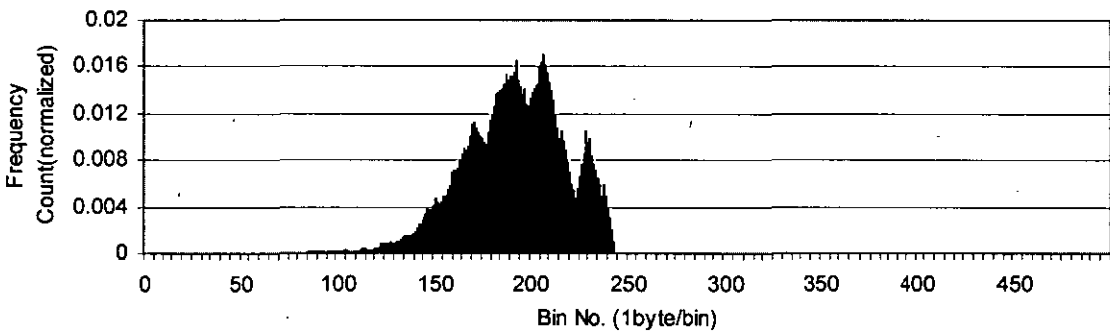


Figure 4.37 2nd order distribution of virtual application 1

Figure 4.36 gives the profile of $Dist^{temp}$ after $Dist^{4th-theory}$ and $Dist^{3rd-theory}$ had been removed from the captured packet size distribution $Dist^{agg}$. **Figure 4.37** shows the profile of $Dist^{2nd}$ for virtual application 1. It was mathematically accepted that $Dist^{agg}$ was aggregated from the original packet size distribution of virtual application 1 because it had a Chi-square value of 122.222 and the 95% confidence critical value was 176.556.

A virtual application (No.18 in database) original packet size distribution is give by **Figure 4.38**, which has the same median position (50 bytes) and spread (100) but different shape (concave). **Figure 4.39** gives the theoretical original distribution extracted using the original distribution of virtual application 18. Even visually, one can immediately argue that the two profiles are from the different random distribution population and were afterwards rejected mathematically with a Chi-square value 794.978.

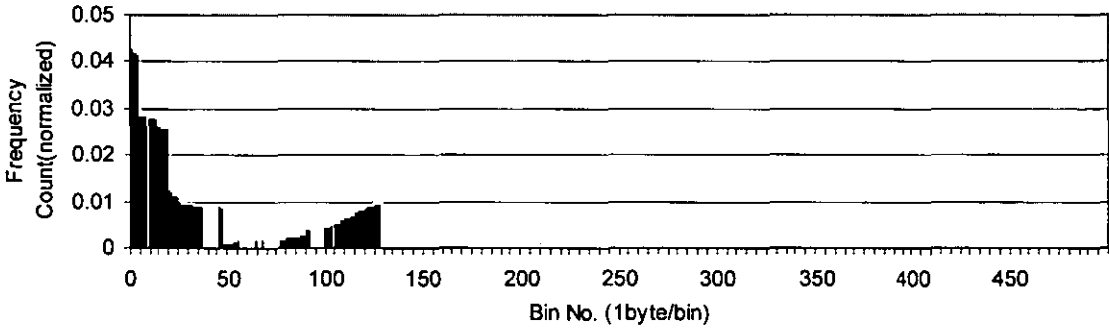


Figure 4.38 Original distribution of virtual application 18

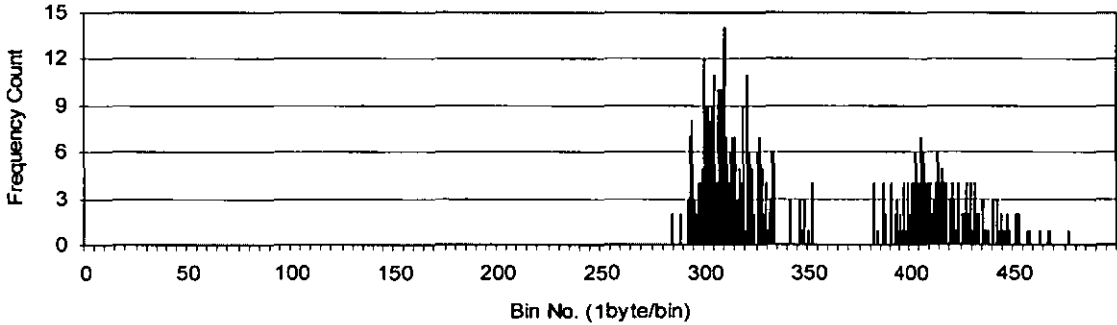


Figure 4.39 Theoretical aggregated distribution extracted using the original distribution of virtual application 18

The plot (**Figure 4.40**) below shows the profile of the theoretical aggregated distribution calculated with another In-Database distribution (concave, median position 200 bytes, and spread 200, No.25 in database) using the first method. It is definitely far away from the captured one (**Figure 4.35**) and was also rejected with a Chi-square value of 653.256.

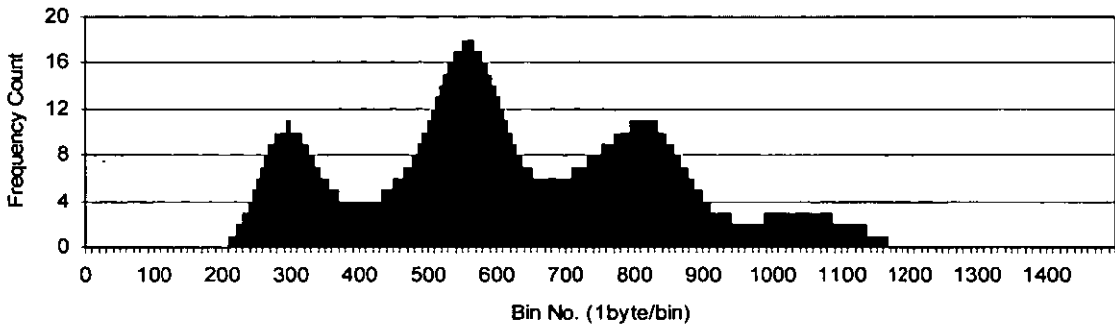


Figure 4.40 Theoretical aggregated distribution calculated from the original distribution for virtual application 25

Network Condition Emulated

Packet Delay (ms)	Jitter (50–300)
-------------------	-----------------

The packet size distribution profile of virtual application 1 under a jittered delay network condition seen in **Figure 4.41** looks quite different from that under the 300ms fixed delay network condition. It is difficult to determine how many 1st order packets in the range 1 byte to 122 bytes were captured, as the 2nd order distribution ranges from 2 bytes to 244 bytes. In addition, many high order packets were also seen in this aggregated distribution.

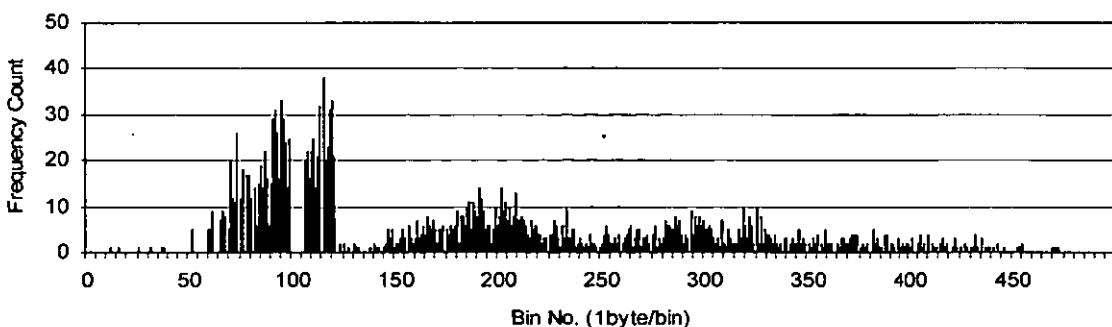


Figure 4.41 Captured aggregated distribution for virtual application 1 under the network condition of jittered delay

Again, the Nagle-detection mechanism successfully extracted the original packet size distribution using the original distribution of virtual application 1. The extracted distribution is given in **Figure 4.42**. There were still some higher order packets that have not been removed, and slight value variations at some bins, nevertheless, the Chi-square test has accepted that this aggregated distribution was aggregated from the original packet size distribution of virtual application 1. Mathematical results against other In-Database applications are summarized in **Figure 4.43**. The red column represents the acceptance of Chi-square test which occurred at the number of this virtual application.

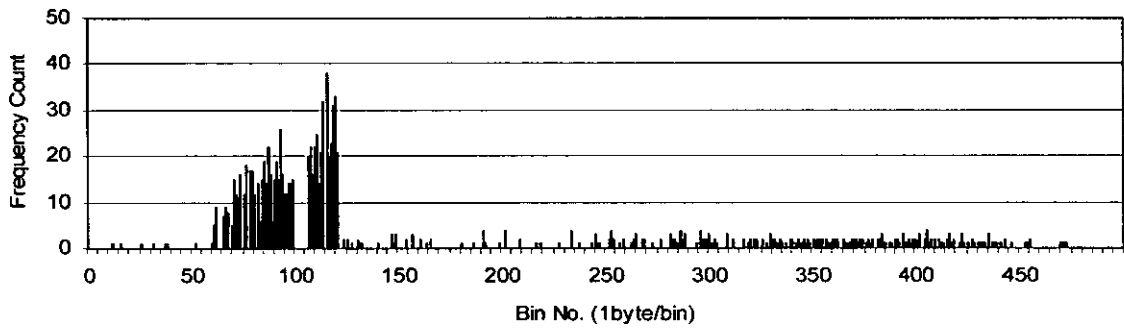


Figure 4.42 Theoretical original distribution extracted using the original distribution for virtual application 1

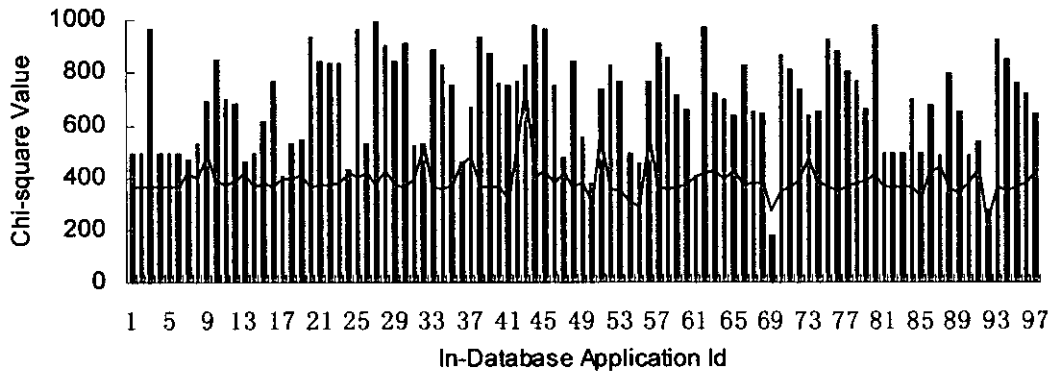


Figure 4.43 Summarizations of Chi-square tests of identifying Virtual Application 1 under network condition of jittered delay

Virtual Application 2

Trend	Random
Median Position	100
Spread	200
Packet Series	Ordered
Id in Database	37

Figure 4.44 shows the profile of original packet size distribution for virtual application 2. This distribution profile looks quite similar to a real application. Three major peaks at 34 bytes, 41 bytes, and 64 bytes can be seen. Some other size packets are also seen with relatively fewer numbers. The distributions for different order packets overlap each other as given in **Figure 4.45**. Ordered packet series have been generated by the Nagle-based Application Emulator with a packet rate of 7/sec which is also similar with those of some real-time applications [ArmS04].

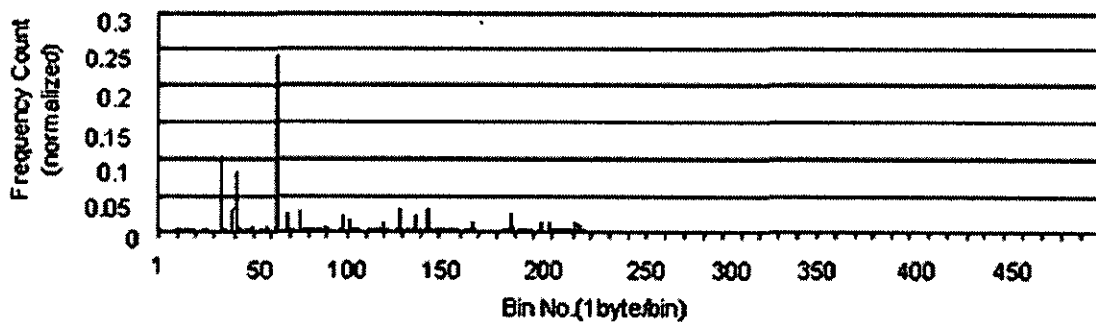


Figure 4.44 Original packet size distribution of virtual application 2

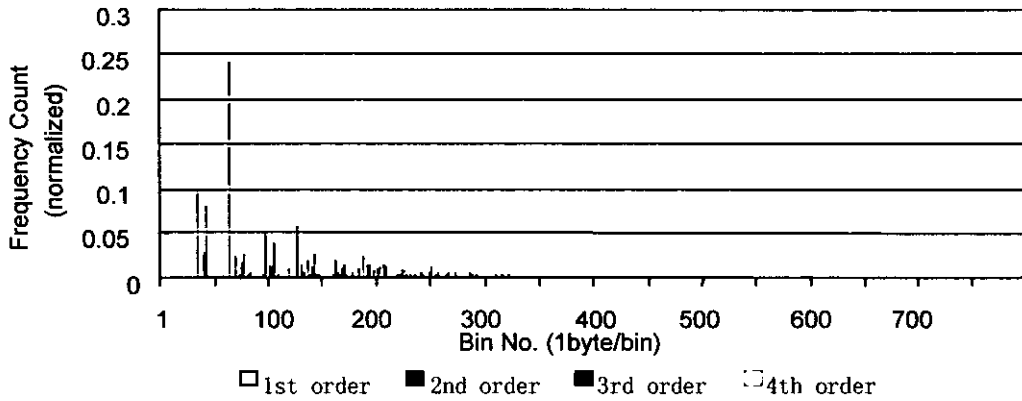


Figure 4.45 Distribution profiles for different orders of virtual application 2

Network Condition Emulated

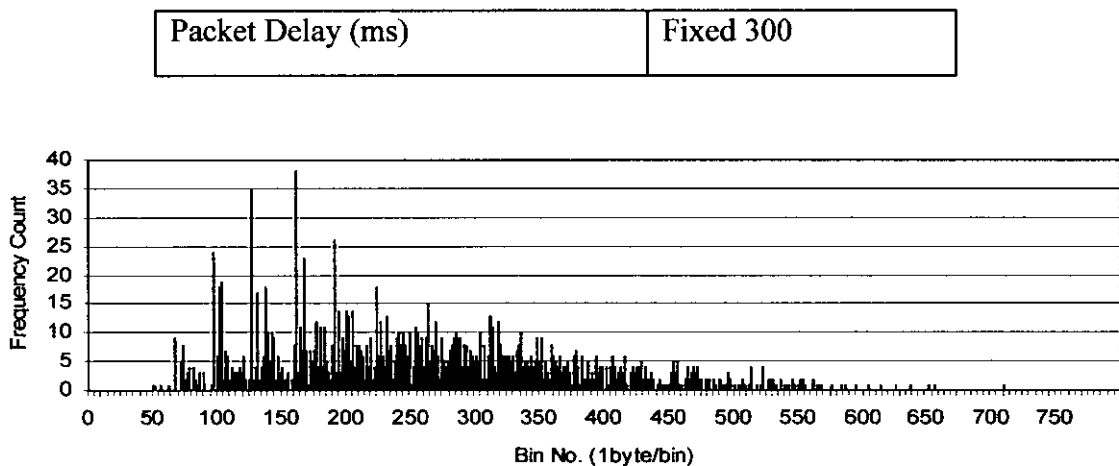


Figure 4.46 Captured aggregated distribution for virtual application 2 under the network condition of 300 ms fixed delay

The second method was adopted for detection of this virtual application. As shown in **Figure 4.46**, under network condition of 300 ms fixed delay, 1st order packets are few, four major peaks occurred at 98 bytes, 128 bytes which are two primary peaks in the 2nd order distribution, 162 bytes, and 192 bytes which are two primary peaks in the 3rd order distribution. It can be deduced that this aggregated distribution consisted mainly of 2nd and the 3rd order packets.

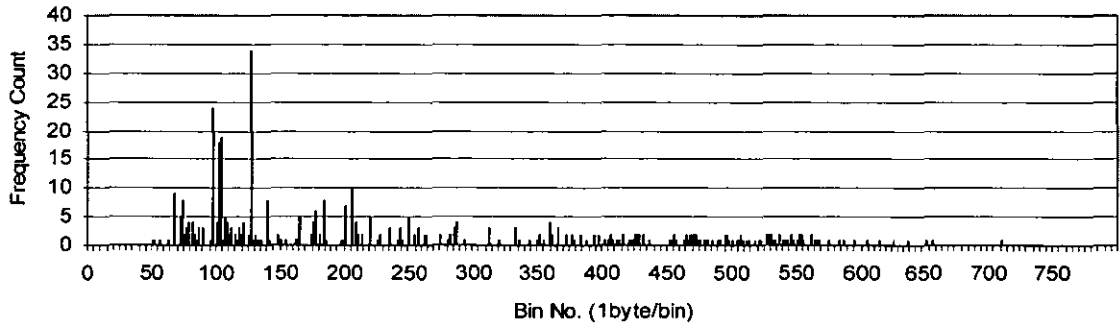


Figure 4.47 $Dist^{temp}$ after $Dist^{4th-theory}$ and $Dist^{3rd-theory}$ were removed

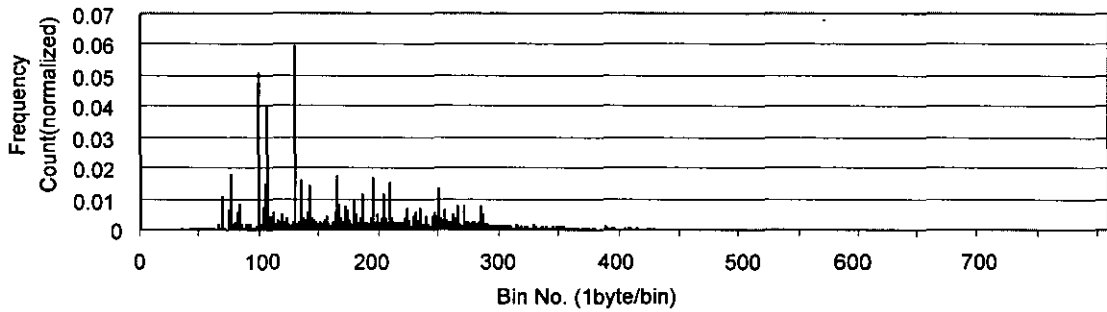


Figure 4.48 2nd order distribution of virtual application 2

The detection was achieved after $Dist^{4th-theory}$ and $Dist^{3rd-theory}$ had been removed from captured packet size distribution $Dist^{agg}$. At this time, the profile of $Dist^{temp}$ (Figure 4.47) looks quite similar to that of the 2nd order distribution of virtual application 2 (Figure 4.48), except for a very few 4th order packets that had not been removed. It was accepted by the Chi-square test with a value of 137.234 against the 95% confidence critical value 164.457.

Network Condition Emulated

Packet Delay (ms)	Jitter (50- 300)
-------------------	------------------

Under this network condition, a large peak at 64 bytes, which is the primary peak of the 1st order distribution of virtual application 2, has been observed in the captured

distribution in **Figure 4.49**. The overall shape of this distribution hence was largely different from that under 300ms fixed delay.

Once again, the original packet size distribution was extracted successfully using the second method (

Figure 4.50), showing a major peak at 64 bytes while two secondary peaks at 34 bytes and 41 bytes were observed, closed to the original distribution of virtual application 2. Some higher order packets seemed to remain in the extracted distribution. However, this inaccuracy has not affected the result of the detection. A Chi-square value of 212.367 was obtained with a 95% critical value of 275.817.

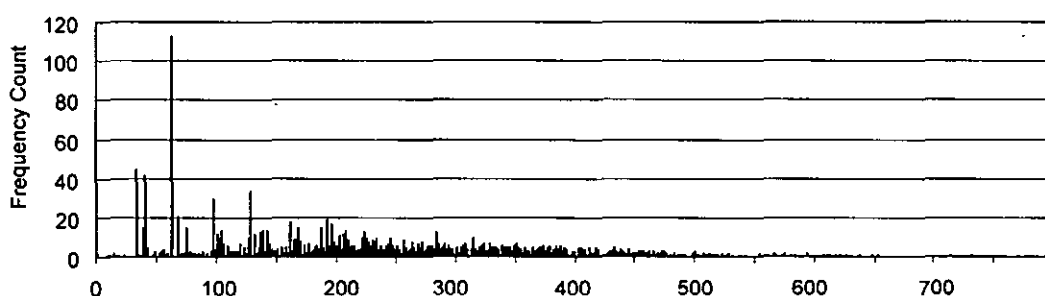


Figure 4.49 Captured aggregated distribution for Virtual Application 2 under the network condition of jittered delay

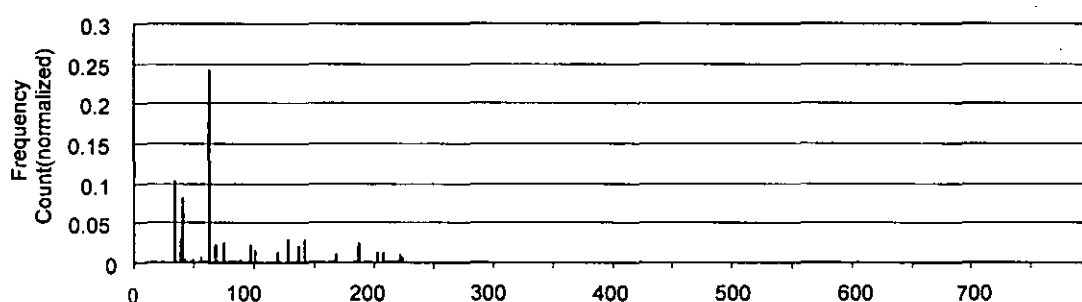


Figure 4.50 Theoretical original distribution extracted using the original distribution of virtual application 2

Some other tests on ordered packet series were carried out as well, the results represented similar theoretical original packet size distributions or theoretical aggregated ones could be extracted or calculated using the methods described in this chapter. Clearly, the

packets generated by real applications would show variations from case to case, however, the nature of the statistical technique suggests that these could be tolerated in a certain range.

Virtual Application 3

Due to the limitation of the MTU and Nagle-based application's packet size distribution aggregation, in some cases, the packet sizes actually sent would not be decided by an application itself. For instance, when a payload buffered by TCP has exceeded the MTU of the Ethernet on which the application is operating, only the first 1460 bytes would be sent out in a packet and TCP would be hung up waiting for the acknowledgement of that outstanding packet, even worse, during this phase the application does not stop submitting messages to the transport layer. As such, more and more 1460 bytes packets would be generated. Hence, if an application's original packet size distribution contains many large packets, with a given packet rate, it could have a high probability of suffering from MTU size packets. In such case, the methods described would be helpless. Virtual application 3 is a typical one with such characteristics, and the following experiments verified this limitation of the methods.

Trend	Random
Median Position	500
Spread	200
Packet Series	Ordered
Packet Rate	5/sec

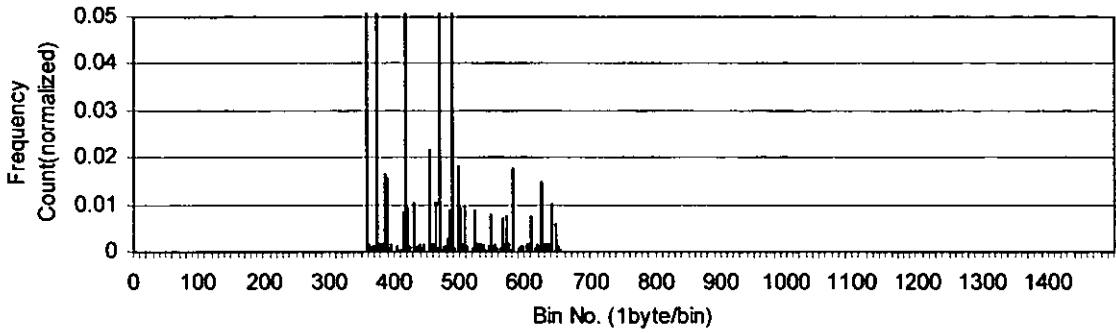


Figure 4.51 Original packet size distribution of virtual application 3

The original packet size distribution profile is given by **Figure 4.51**. It can be seen that five huge peaks appear at 359 bytes, 375 bytes, 416 bytes, 469 bytes, and 489 bytes respectively. These packet sizes are extremely large for real-time applications and are seldom seen, however, they do have the possibility to occur. For this virtual application, only 1st and 2nd order distributions could be completely calculated in advance (**Figure 4.52**) as those distributions higher than 2nd order would all exceed the MTU and all bins exceeding the MTU should be ignored.

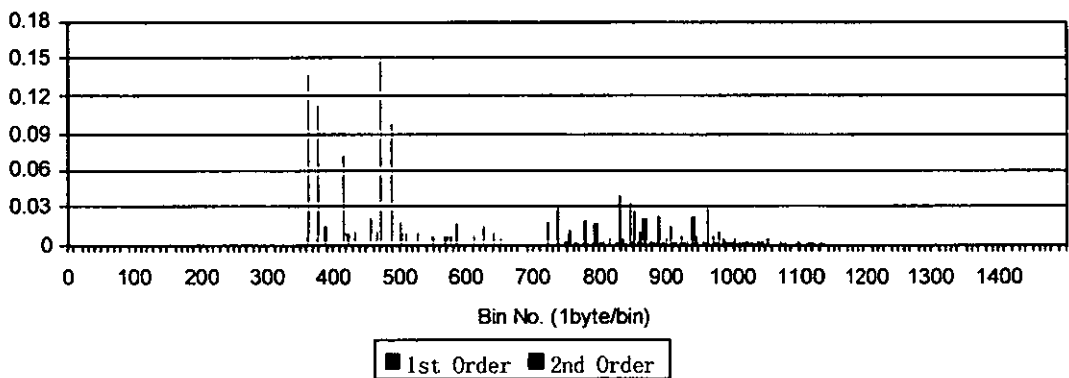


Figure 4.52 Distribution profiles of different orders for virtual application 3

Network Condition Emulated

Packet Delay (ms)	Fixed 100
-------------------	-----------

Under this low-load network condition, MTU size packets have not been largely observed as shown in **Figure 4.53**. It seems that the aggregated distribution mostly consisted of 1st and 2nd order packets. Thus, detection was successful using the first method. The Chi-square value was 23.141, lower than 95% confidence critical value of 193.126. The theoretical aggregated distribution calculated with the original distribution of virtual application 3 is shown in **Figure 4.54**.

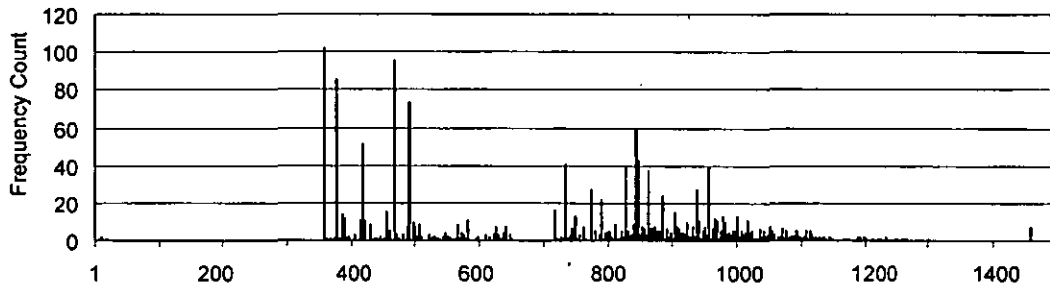


Figure 4.53 Captured aggregated distribution for virtual application 3 under the network condition of 100 ms fixed delay

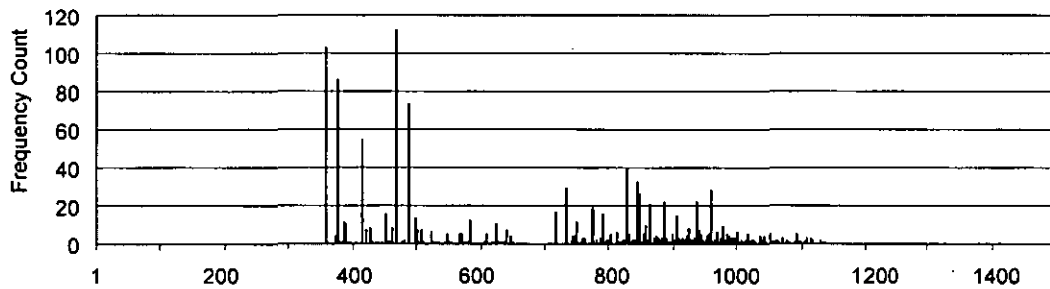


Figure 4.54 Theoretical aggregated distribution calculated from the original distribution for virtual application 3

Network Condition Emulated

Packet Delay (ms)	Jitter (50-300)
-------------------	-----------------

With the increment of delay, more MTU size packets were observed as expected. Some small size packets which may be the tails coming from the splitting of payload exceeding the MTU were also seen as shown in

Figure 4.55.

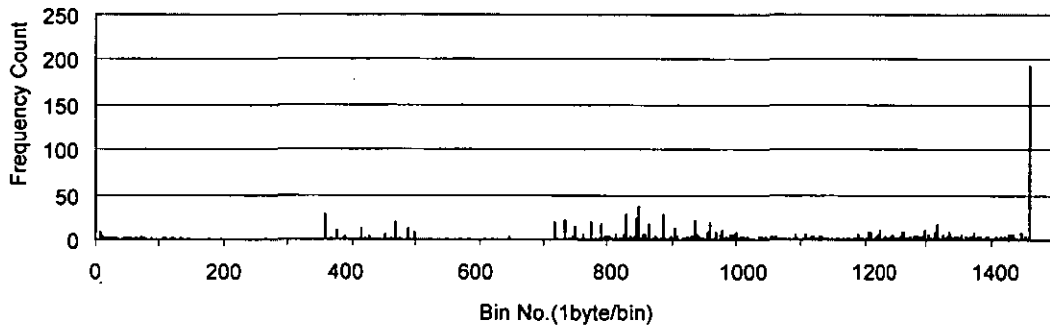


Figure 4.55 Captured aggregated distribution for virtual application 3 under the network condition of jittered delay

Both the first and the second methods then were applied to try to achieve the detection. As shown in **Figure 4.56**, the so calculated theoretical aggregated distribution has little in common with the captured one, and was rejected with a Chi-square value of 568.743 which was much greater than even the 50% confidence critical value.

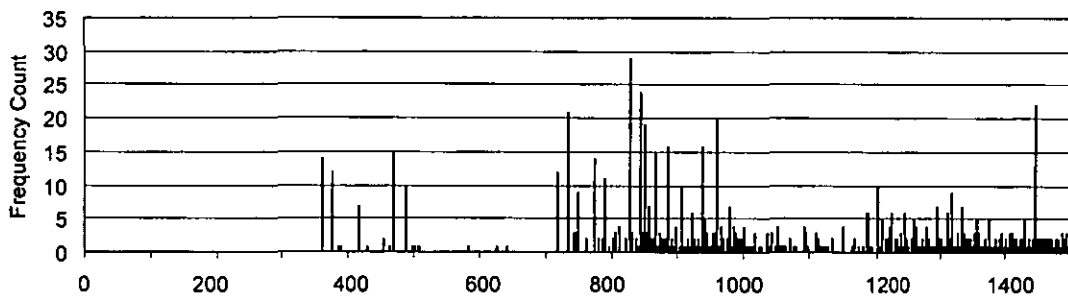


Figure 4.56 Theoretical aggregated distribution calculated from the original distribution for virtual application 3

The results of the second method look better than the first method (

Figure 4.57). The extracted theoretical original distribution removed some high order packets, however, a huge peak at 1460 bytes remained. In addition, the method was unable to remove the “tails”. In this case, a Chi-square value 272.445 was obtained and was also rejected with a critical value 172.870.

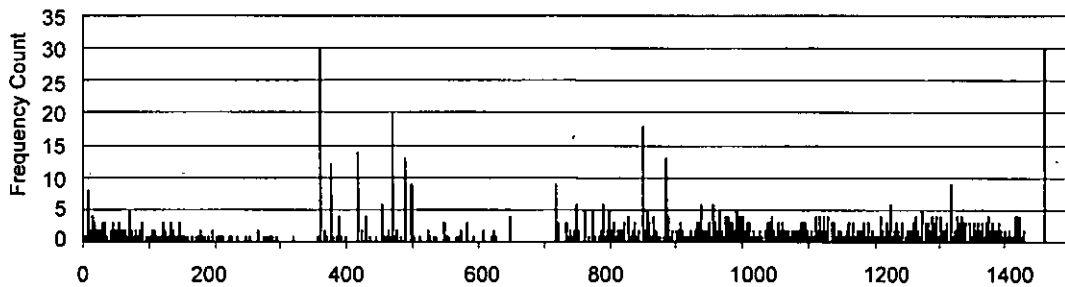


Figure 4.57 Theoretical original distribution extracted using the original distribution of virtual application 3

Network Condition Emulated

Packet Loss Ratio (percentage)	3
Packet Delay (ms)	Fixed 300

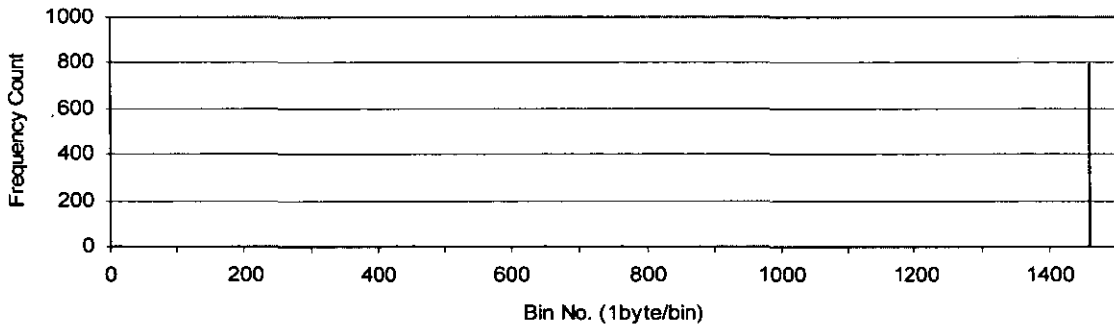


Figure 4.58 Captured aggregated distribution for virtual application 3 under a network condition of 300 ms fixed delay

When the network condition was worsened to 300 ms fixed delay, the profile of the aggregated distribution contained the MTU size packets as shown in **Figure 4.58**. In such case, detection becomes impossible. As shown in the tests above, in the circumstances that the original packet size distribution consists of large packets and the network is highly loaded, the aggregated distribution could potentially exceed the MTU and lead to many or even most payloads being sent as MTU packets. In these cases, it would be difficult to achieve detection using the methods described in this work.

4.3.5 Tests on Other Virtual Applications

All In-Database virtual applications with various original distributions were tested in the same way as above. **Figure 4.59** shows a general Chi-square summary for all virtual applications that were tested under a 50ms to 300ms jittered delay network condition.

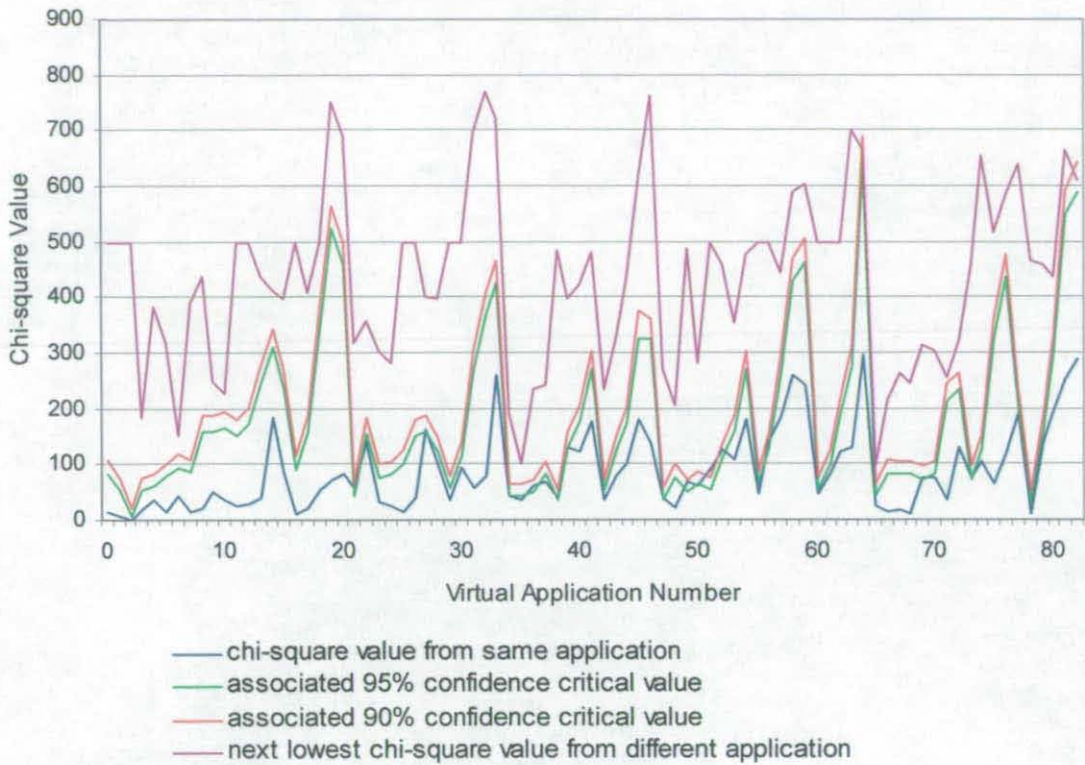
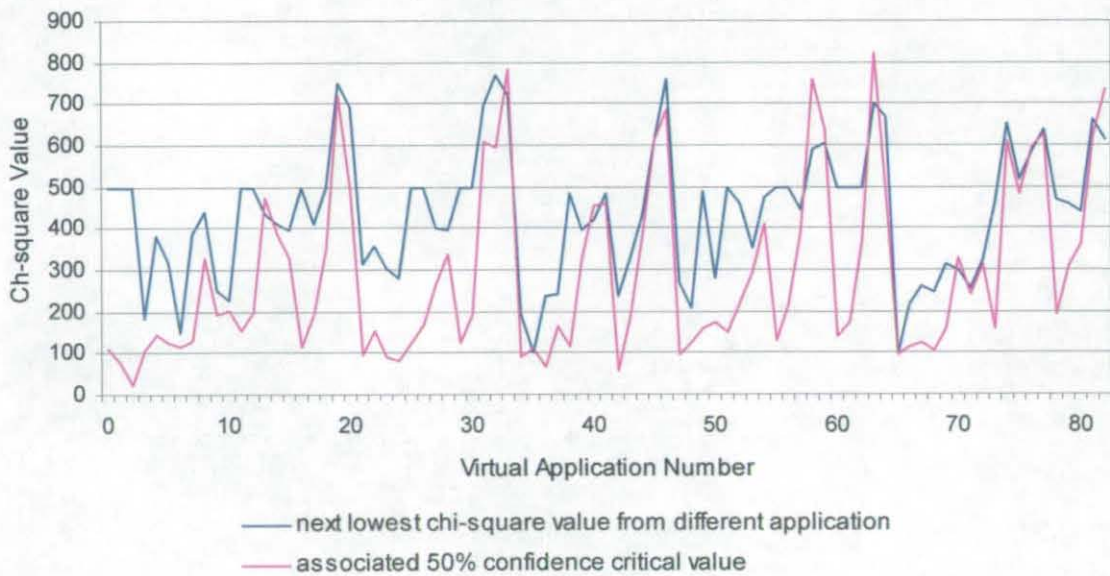
*a**b*

Figure 4.59 Chi-square test summary for all tested virtual applications

In **Figure 4.59(a)**, the blue curve represents the Chi-square values resulting from the computation with the theoretical distributions calculated with their own original packet size distributions. The red and green curves represent corresponding critical values with

confidence 95% and 90% respectively. The second lowest Chi-squared values resulting from the computation with the theoretical distributions calculated with a different application trace are also given as a purple curve.

It is seen that, for all the applications shown, the lowest Chi-squared value always occurred when the detection mechanism was performed with the original packet size distribution from the same application. In all but a few cases, the Chi-squared value is below the critical value of 95% confidence value and, in all cases, below the critical value for 90% confidence value. In all cases, the next lowest Chi-squared value from a different application is seen to be significantly greater than the lowest ones.

In **Figure 4.59(b)**, the second lowest Chi-square values are given in the blue curve, accompanied with corresponding critical values with confidence 50% in pink. In most cases, the Chi-square value is much greater than the associated 50% confidence value for that second choice application for a few cases it is not. Nevertheless, as the second lowest value must be rejected in favour of the lowest one, those second lowest values could be excluded from consideration. The circumstance under 300ms fixed delay network condition gives a similar result. Hence, the aggregated distribution detection mechanism discussed in this chapter can be considered as successful and feasible.

4.4 Summary

The aggregated distribution detection mechanism is proposed and tested in this chapter. Two methods were developed in order to suit the needs of a variety of distribution profiles. Because of the shortage of the Nagle-based application in reality, around one hundred of virtual applications were generated and run across the network. The results show that, in spite of a few virtual applications with large size packets often send out packets at the MTU size under the loaded network conditions and thus are difficult to detect, most applications can be successfully detected regardless the packet size distribution profiles' shape, median and spread are varying. Hence, the aggregated distribution detection mechanism discussed in this chapter can be considered as successful and feasible.

CHAPTER

5

Design and Architecture of A Prototype Application Detector

5.1 Introduction

A TCP-based application detector had been designed employing the ideas described in the last two chapters. The aim of establishing this prototype detector is to verify the feasibilities of those ideas and discover those parameters that could potentially make the idea performs better. In this chapter, the design and architecture of this prototype detector is discussed.

5.2 Operation Overview

This prototype TCP-based application detector implemented the ideas that packet size distribution could be a fingerprint of TCP-based applications and with which identification of TCP-based applications can be achieved. The prototype detector was written using object—oriented techniques and designed to be as efficient as possible. The development language adopted was object-oriented Java. As Java is both a programming language and a platform, with the installation of JVM (Java Virtual Machine), the ability of operating on a variety of platforms could be obtained [Sun05].

In addition, as Java supplies the ability of simultaneous multi-threading [Sun98], this detector has been designed as a whole but could be broken down into a number of threads that work in parallel and relatively independently of each other. The advantage of this idea is that a thread could continue working on its own jobs without being interrupted by other jobs. Among these threads, one is in charge of dumping packets from the network and generating packet size distribution profiles, another performs the actual detection procedure, while the user interface is operated as the third thread. In fact, the whole detector is made up of five threads. **Table 5.1** below gives all threads with their functions.

The first stage of the application detection is the collection of the raw packets by *tcpdump*(or *windump* under windows environment [Win04]). This utility is called by thread *loadPackets()*, which in turn receives the output of *tcpdump* and generates summaries in the form of packet size distribution profiles. The next thread in the flow, *Detect()* reads these profiles and analyses the streams and connection data and performs the identification via the statistical detection methods described in the previous chapter. The identified information is displayed by the user interface *snapui()*.

Thread Name	Description
<i>tcpdump</i>	Runs in promiscuous mode in order to collect all raw packets from the traffic stream seen by the host network interface.
<i>LoadPackets()</i>	Reads the output stream, decodes the hex data of IP and TCP headers to load raw packets into the detector. Creates packet size distribution profiles with the received packets.
<i>Detect()</i>	Performs statistical detection on the TCP packet size distribution profiles, generates detection result information.
<i>Snapui</i>	Receives and displays the results of detection
<i>build</i>	This process is used to populate the statistical store with samples of distribution profiles of applications – it runs off line from the other detector processes.

Table 5.1 Detector process summary

Synchronizing problems arose during the programming, because for the Java language, the variable transfer from one method to another is based on address, not on variable [Sch01]. This mechanism is similar to the circumstance of transfer a pointer to a function in the C language. Hence, it could happen that, if a variable is operated by more than one method in different threads, conflicts could potentially exist when multiple methods visit the variable at the same time. These problems were resolved by transferring cloned objects and using the keyword *synchronized* for every method which will visit a certain object.

The database used was Mysql [Mys06], which is a medium size database software. Data in the database were organised using a relational database technique and connected to the detector via JDBC which is a standard database-connection package supplied by JSDK. **Figure 5.1** gives the overall flow diagram of this prototype detector.

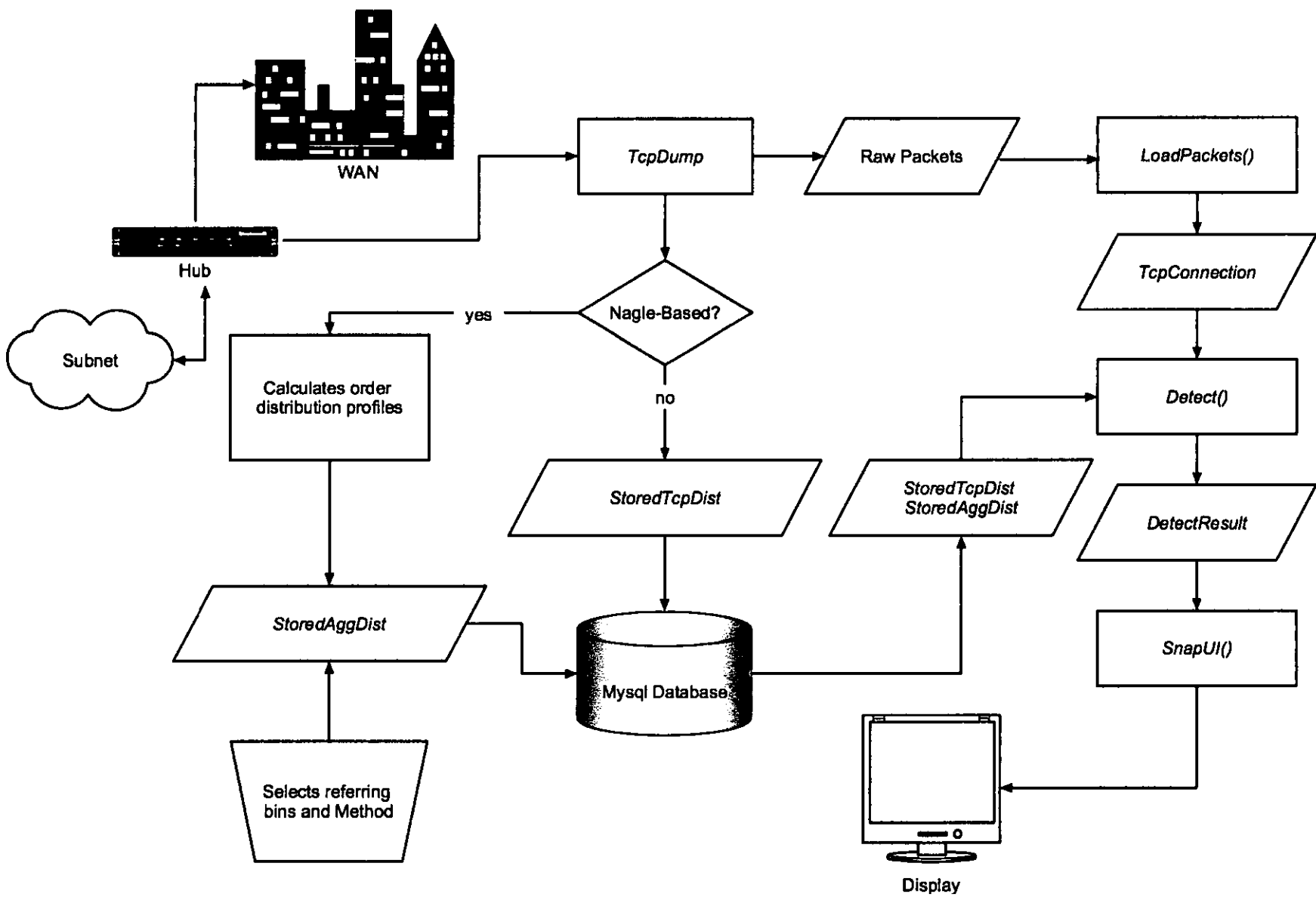


Figure 5.1 Flow diagram of the prototype detector

5.3 Thread LoadPackets()

The thread *LoadPackets()* is responsible for collecting data from the network and generating packet size distribution profiles readied for detection. The mission of dumping raw packets is done by the *tcpdump*. At the start of this thread, *tcpdump* is called and run as a system process. The format of the *tcpdump* command is:

```
tcpdump -nx -tt -s56 tcp
```

The 'x' switch forces the output to be in hexadecimal format and 'n' suppresses translation of IP address to host name (in order to save compute cycles – the detector does not use host names). The 'tt' switch causes the timestamp on each packet to be in calendar time (seconds passed since 1st January 1970). The 's56' switch causes it to capture just the first 56 bytes of the IP frame (the link level header preceding the IP header is omitted in any case). This also helps to reduce the processing load on the host machine; no use is made of the remainder of the packet in any case [Bha01]. The last protocol switch 'tcp' forces all packets with protocol other than tcp to be filtered [Tcw06]. There is no host switch specified so that *tcpdump* will run in promiscuous mode and capture all packets it has seen.

The output of *tcpdump* is streamed in to the thread *LoadPackets()* for further action. The function of loading the stream from the other applications' standard output is offered by JSDK, which could allow the detector to operate in an operating system without a pipe output. As such, the data received by *LoadPackets()* is therefore at the individual packet level and it is processed at full line rate.

The classes' relationship is shown in **Figure 5.2**. For each TCP packet received, an object of type *Packet* that contains a number of relevant fields will be created by decoding the hexadecimal output stream fed by *tcpdump*. Then a connection level object of type *tcpConnection* will be created which is identified by the hosts and ports information, if an object matching this information already exists, the proper *tcpConnection* object will be updated according to the new arriving *packet*. The packet size distribution profile is established as an object of type *tcpDist* which is a member of *tcpConnection*, the distribution is actually built into an integer array *tcpDist[]*, besides that another float array *tcpDistNor[]* which is used to accommodate normalized distribution is also a member of

the object *tcpDist*. After the update is accomplished with the size of new arriving packet, this object *Packet* will be no longer useful and is released by the auto-rubbish-cleaning mechanism offered by Java Runtime.

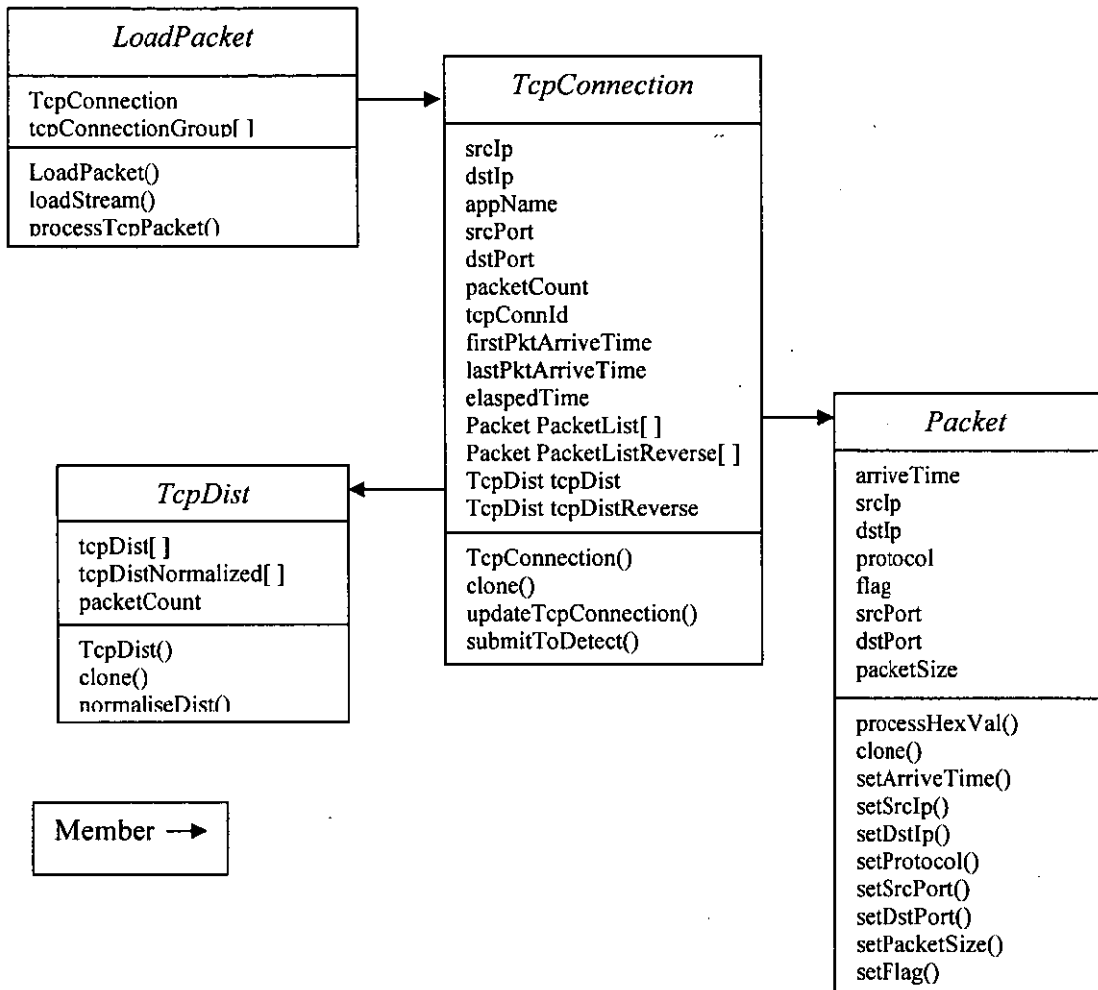


Figure 5.2 Relationship diagram of classes in thread *LoadPackets()*

The object *tcpDist* is updated with the new incoming packets during an amount of time set by the variable *tcpDetectInterval* that is parsed from an XML file [Xcwg05], which contains the configuration information for the detector. When a time of the *tcpDetectInterval* elapses from the arrival time of the first packet of this connection, the *TcpConnection* object will be submitted to thread *Detect()* in order to perform actual detection. **Figure 5.3** gives the flow diagram of this thread.

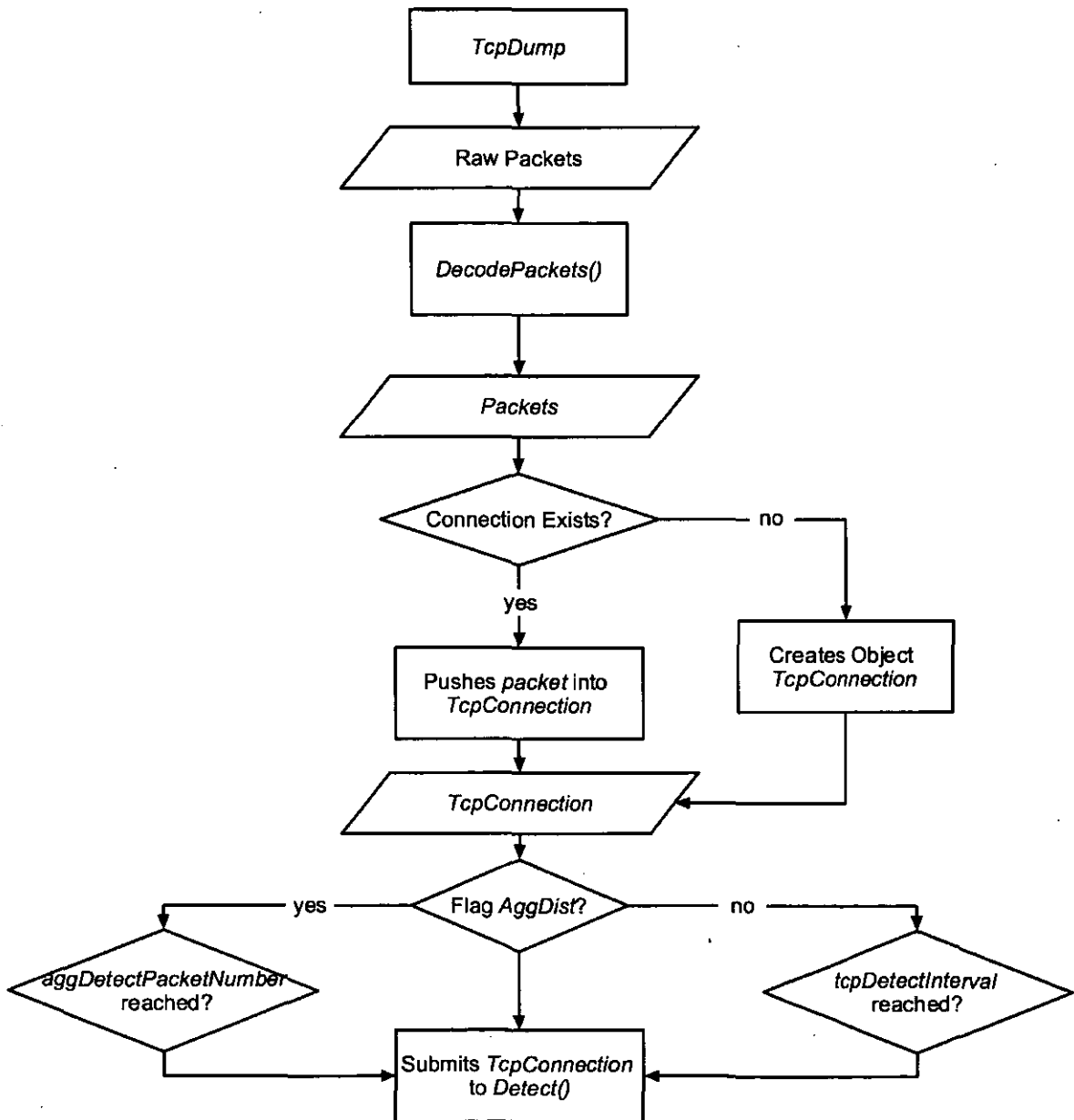


Figure 5.3 Flow diagram for thread *LoadPackets()*

For the Nagle-based applications, as more sampling packets would be needed so as to build a reliable aggregated packet size distribution. The object will not be destroyed until detection is completed. If the normal detect method fails in identifying the connection (a flag *AggDist* will be set), more packets captured belonging to this connection will be added until the variable *aggDetectPacketNumber*, which is also parsed from the configuration file, is reached in order to allow aggregation distribution detection methods to be functioned.

On most networks, one can expect a large number of connections to be simultaneously active. This would soon overload the array bounds of this thread causing it to fail. Some connections, however, don't exist for long enough to reach the *tcpDetectInterval*. These connections will be pending and Java Runtime cannot release the memories automatically. As such, a sub-thread, which is responsible for cleaning these inactive connections, is included in the detector. It periodically checks the existing time of each connection, if this parameter is greater than the threshold *inactiveConnectionExistingTime*, which is set by the configuration file, then the application is deemed to have been terminated and the object will be destroyed.

5.4 Thread Detect()

This thread is at the heart of the application detector system. It performs the actual detection, and maintains and archives the *detectResult* objects, as connections and streams on the network are established and terminated. The output of this process feeds the user interfaces *snapui()*.

When this thread is started, a number of initial jobs are done at the same time. The stored packet size distributions of both the Non-Nagle and Nagle-based applications are loaded. Each stored distribution is created as an object of type either *StoreTcpDist* or *StoredTcpAggDist* and pushed into the static object *StoredTcpDistGroup* or *StoredAggDistGroup* respectively. Some detector settings are also initialized here, such as the detecting interval and the required packet number of the Nagle-based application detection. **Figure 5.4** gives the classes' relationship.

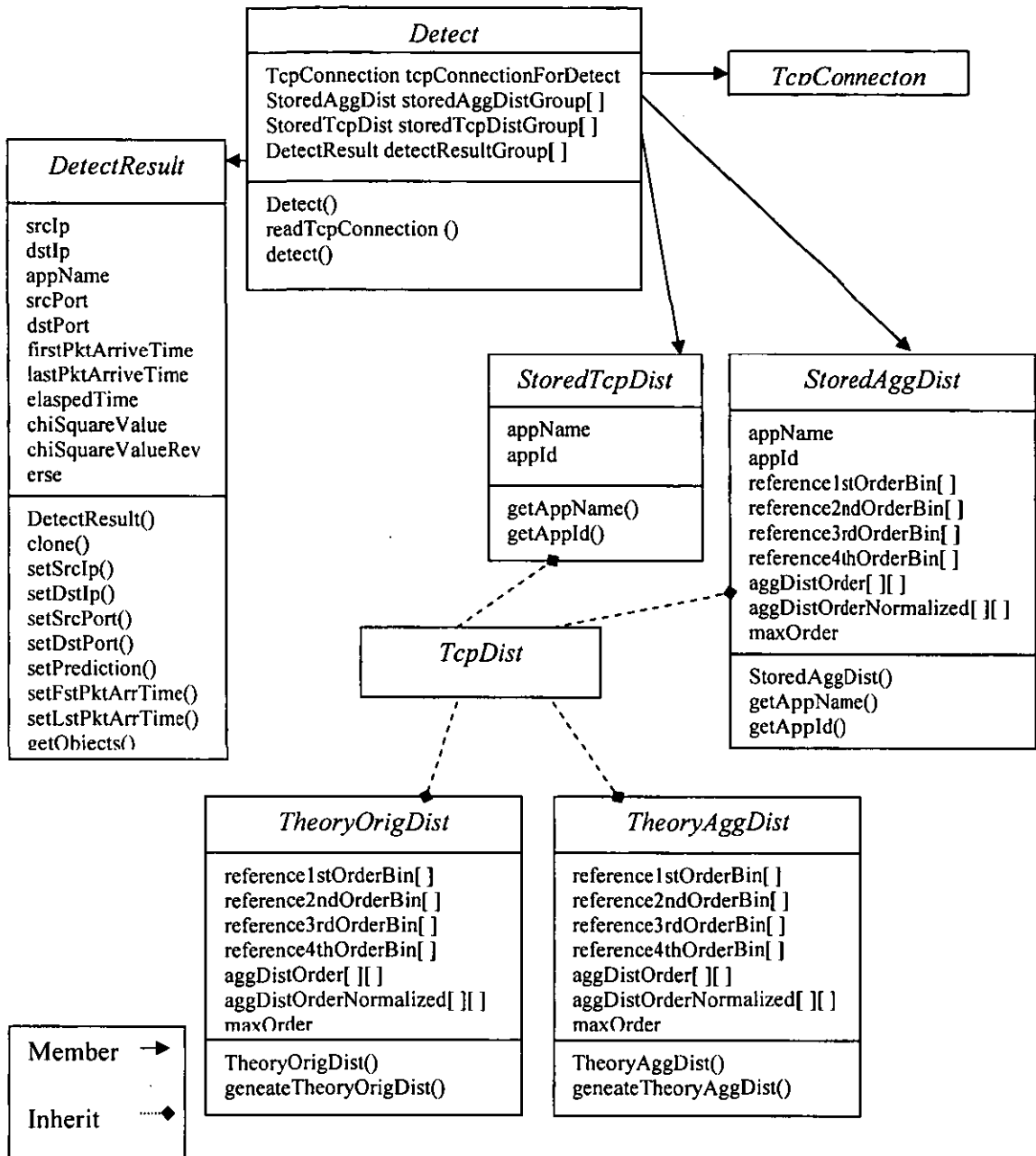


Figure 5.4 Relationship diagram of classes in thread Detect()

When a *tcpConnection* object arrives and readies for detection, the main method *detect()* in this thread will be triggered, and the statistical detection technique will be applied on the incoming *tcpConnection*. First of all, some *tcpConnection* objects that contain very few packets will be ignored. This is aimed at reducing the chance of misdetection. Then, the thread applies the normal detection method on the *tcpDist* object. During this procedure, all the *StoredTcpDist* objects contained within the *StoredTcpDistGroup* will be enumerated, and compared against the test distribution using the Chi-square test described

in the previous chapters. If a successful detection attempt is achieved, a *DetectResult* object will be created in order to feed the next stage thread *snapi()* for the purpose of display. However, if the normal detection method returns a result "unknown", the *tcpConnection* object will set the flag "AggDist" and return to the thread *LoadPackets()* for complementary packet capture. When the number of sampling packets for this *tcpConnection* object reaches the setting *aggDetectPacketNumber*, it will be submitted to the *Detect()* thread once again. The thread will iterate all *StoredAggDist* objects contained within the *StoredAggDistGroup* object and at this time, aggregated distribution detection methods will be used. Then a *DetectResult* object is created and submitted to the *snapi()*.

As the adopted aggregated detection methods and the referring bins vary from application to application, if information is loaded from the Mysql database associated with the packet size distribution profiles at the initialization stage. Two different methods dealing with the aggregated distribution are implemented as described in the Chapter 4. When the first method is adopted, the comparison using the Chi-square test is performed once after the theoretical aggregated distribution object *TheoryAggDist* is created. While the second method is in operation, a theoretical original distribution object *TheoryOrigDist* object will be calculated. In addition, as described in Chapter 4, detection attempts are also applied each time certain order packets are just removed.

Each time a detection attempt is performed, the critical value of the threshold confidence (set in XML file) is calculated and compared with the Chi-square value to see if the result should be accepted with the given confidence. If more than one application returns acceptance, all the Chi-square values and the associated critical values will be recorded. In the case of that no acceptance is observed, the *appName* field in *DetectResult* object will be set as "Unknown". The flow diagram of this thread is given in **Figure 5.5**.

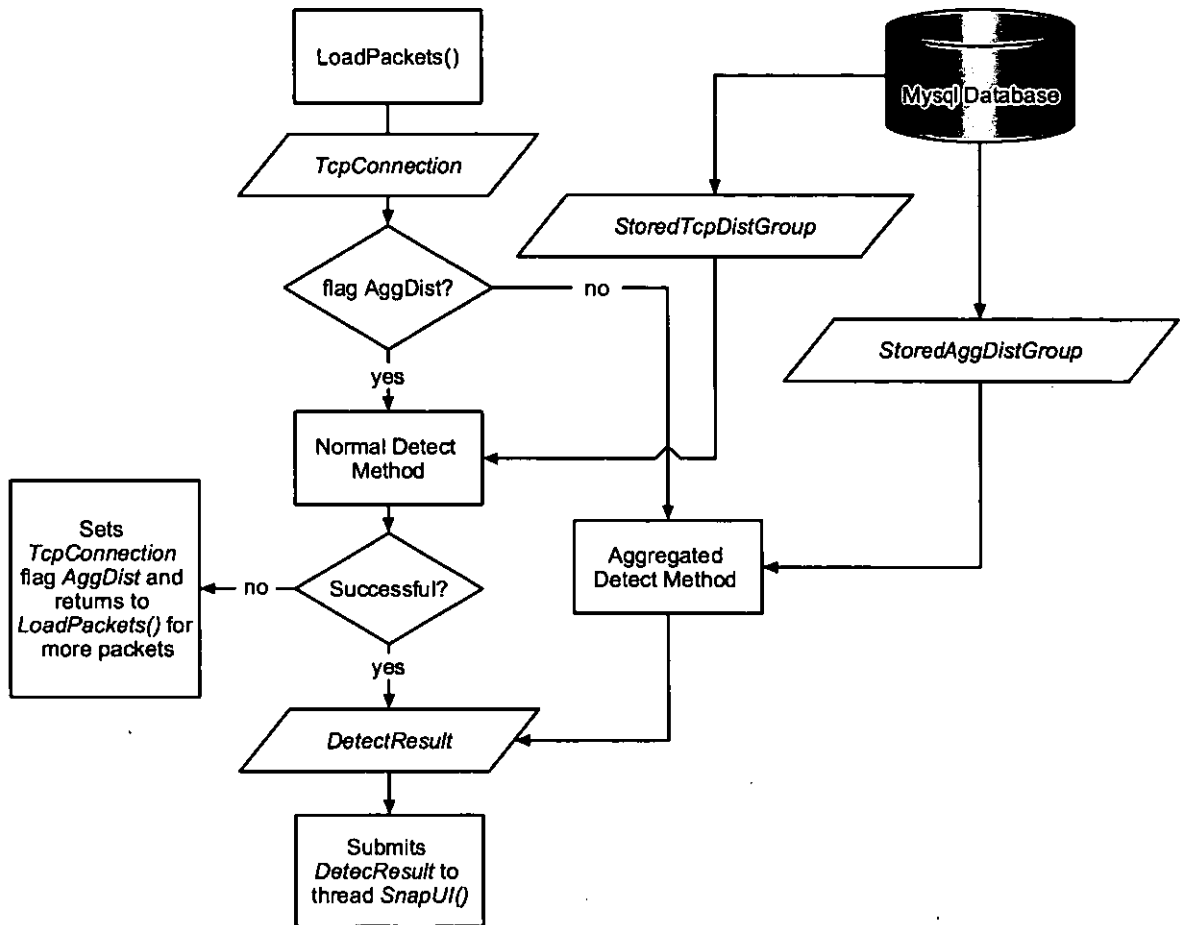


Figure 5.5 Flow diagram for thread `Detect()`

5.5 Thread `SanpUI()`

The thread `snapui()` is in charge of providing final detection results information to the user. As this detector is only designed as a prototype and mainly for the purpose of experiments, the user interface is quite simple and some useful information from the performance tests is also displayed. Detailed discussion of the structure of this thread will not be entered into here.

Except the `java.awt` and `java.swing`, which are two standard packages in JSDK [SunM06], no other operating system related UI API is employed. This is also aimed at trans-platform operation. A static object of the type `connTableData` is maintained by this thread in order to allow the thread `Detect()` to push the `DetectResult` objects into it. This thread simply iterates all `DetectResult` objects accommodated in `connTableData` and

parses them so as that detailed detection result information can be displayed. The update interval of the user interface is set to 5 seconds in the configuration file. The flow diagram of this thread is given in **Figure 5.6**.

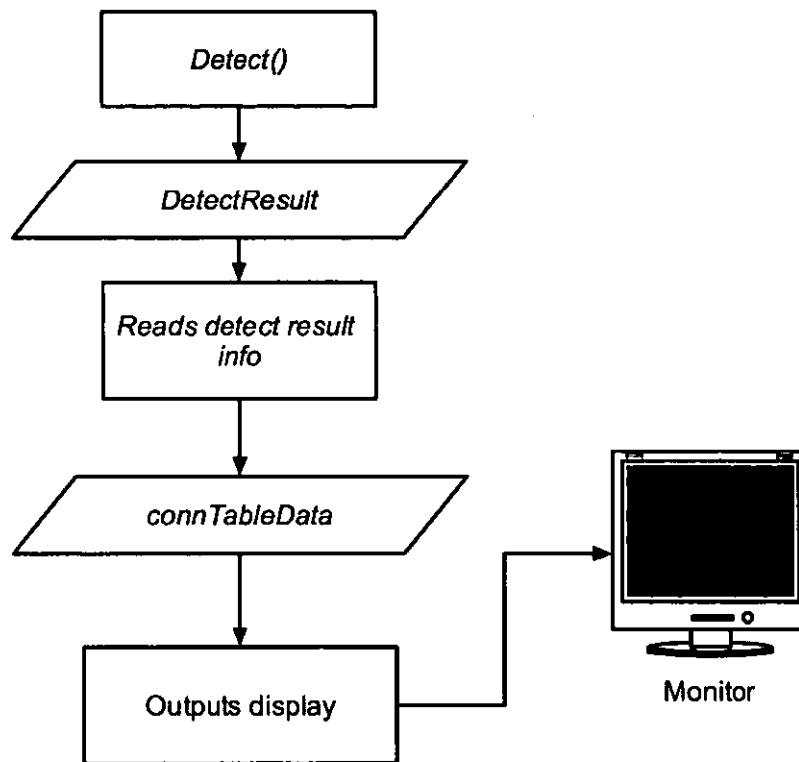


Figure 5.6 Flow diagram for thread `SnapUi()`

The whole user interface could be divided into two major parts. The Detector Information Display shows the working status of the detector, such as current threshold confidence, capture interval, and number of stored distributions. This information must be supplied as it could potentially affect the performance of this detector (this will be discussed in Chapter 6). The Detect Result Display shows all detections achieved whatever they result in identification are “unknown”. Each line in the basic data display represents one stream or connection. The information shown is summarised in **Table 5.2**.

Column Heading	Description
Source IP	IP address of source machine generating traffic.
Source Port	Source port number in use for this connection.
Dest IP	IP address of destination machine.
Dest Port	Destination port number in use for this connection.
Time Start	Time of first packet for this connection was captured.
Prot	Protocol of this connection.
Pkt Count	Packets captured for current build. (Reset when DETECTINTEVAL has elapsed).
Lo Chi-square Value	The lowest Chi-square value obtained for this connection from the last detection attempt
Lo Critical value with confidence 1	The critical value with confidence 1 associated with the lowest Chi-square value from the last detection attempt
Lo Critical value with confidence 2	The critical value with confidence 2 associated with the lowest Chi-square value from the last detection attempt
Lo Critical value with confidence 3	The critical value with confidence 3 associated with the lowest Chi-square value from the last detection attempt
2 nd Chi-square Value	The second lowest Chi-square value obtained for this connection from the last detection attempt
2 nd Critical value with confidence 2	The critical value with confidence 1 associated with the second lowest Chi-square value from the last detection attempt
2 nd Critical value with confidence 2	The critical value with confidence 2 associated with the second lowest Chi-square value from the last detection attempt
2 nd Critical value with confidence 3	The critical value with confidence 3 associated with the second lowest Chi-square value from the last detection attempt
Prediction	Application identity determined through port usage.
Hits	Number of times associated identity was found for each result of detection
Detect Attempts	Total number of times carried on this connection
Acceptances	Number of times the correct application had been accepted by Chi-square tests.

Table 5.2 User Interface Information

Actually, following the concept of object-oriented programming, the design of the software should follow the order from the top layer (i.e. User interfaces) to the bottom layer (the actual detection functions) [DatK97], hence, this thread was the first one to be written and acts as the entry of the whole detector. The other threads, which are working in parallel, are all initialized or triggered inside this thread.

The design of this interface is exclusively for the purpose of testing the ideas described in previous chapters and to find out any areas where potential improvement could be achieved. The display format was so designed to allow users to see information relating to the experiments.

5.6 Thread `BuildStoredProfiles()`

This thread is separated from the prototype detector and works independently for the purpose of inputting stored packet size distribution profiles into the database. The architecture of this thread shares the same code with thread `LoadPackets()`. In addition, a module, which is in charge of connecting the database, is added. We below discuss the process to establish a stored distribution profile using WarCraft III as an example.

The Unix utility *tcpdump* or its windows version *windump* [Win04], is used here to do the actual packet capture from the network. The command options used are shown below.

`Tcpdump -nx -tt -s56 host port 6112 and tcp black`

The meaning of the switches is explained in Section 5.2. The host, port, and protocol parameters enable capture of all tcp packets (as only TCP packets are of interest in this work) destined to or from the specified port 6112 on the host named black, so that other traffic is filtered out. The hardware architecture is the same as **Figure 3.1**.

When the thread is started, *tcpdump* will capture packets. The arrival time of the first packet is extracted and stored (from the timestamp inserted by *tcpdump* rather than the arrival time determined from the host computer real time clock). The distribution profiles are built in the capture array using all the subsequent packets until the first one whose time stamp is *storedProfileInterval* seconds (preset in the configuration file) after that of

the first packet was received. Then the array that accommodates the distribution profile is normalized and written into the table of TCP packet size distribution in the *Mysql* database and the user is prompted to enter the applications' names.

For the Nagle-based applications, after the establishment of the distribution array, some pre-computation will be performed using equation 4.5 in order to obtain packet size distributions for different orders. Afterwards, the referring number for each order and the aggregation detection method adopted for this specific application are requested. The distribution profiles of the Nagle-based applications and above information, which the user entered, will be written into the table of aggregated packet size distributions in the *Mysql* database. The flow diagram is given in **Figure 5.7**.

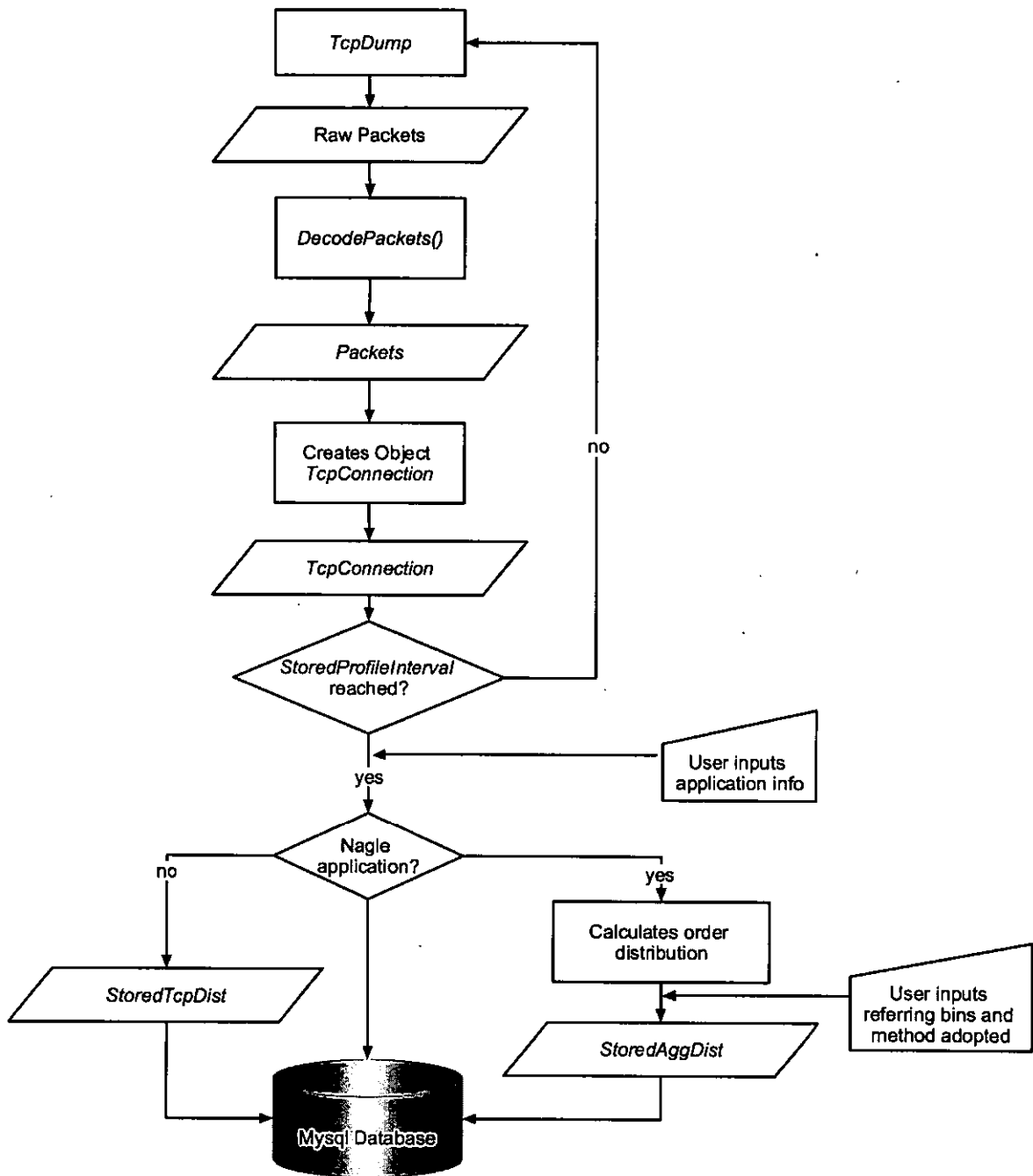


Figure 5.7 Flow diagram for thread *BuildStoredProfiles()*

In the database, all pre-stored application distribution profiles will be normalized before they can be written into the table. For the Nagle-based pre-stored applications, the packet size distribution of each order is calculated in advance and stored in the database so as to reduce the computation quantity cost when the detection is carried out in real-time. When a connection is readied for detection, the normalized distribution will be multiplied with

the total packet number for the connection to obtain an integer distribution as required by the Chi-square test.

The referring bin numbers of the Nagle-based application were chosen according to the principle described in Chapter 4. So far, this is only done manually as the selection of referring bins could vary from case to case.

5.7 Summary

This prototype TCP application detector is designed for testing the ideas described in the last two chapters. Most effort was directed at implementation of these ideas and founding a testing platform in order to find out how well the ideas work and to determine some parameters, which could optimize the detection job. As such, the architecture of this prototype is very simple but may satisfy some requirements for a complete application detector.

CHAPTER

6

Application Detector Performance

6.1 Introduction

In the last chapter, a TCP-based application detector prototype implementing the suggestion described in Chapter 3 and Chapter 4 that TCP-based applications could be identified using packet size distributions is presented. Now, it is necessary to test the performance of this prototype detector.

In this chapter, the performance of the TCP Application Detector is measured and discussed. In addition, some parameters for the detector are found experimentally in order to achieve the best balance between detection reliability and timeliness of the identity information.

6.2 The Settings of the Detector

The packet size distribution is a statistical measurement, and could show varying stabilization over different counts of samples. As such, it is necessary to determine how many packet samples it should take to build stable enough packet size distribution profiles of applications for the purpose of detection.

The Chi-square test detection approach is such a good detection technique that it has the ability to accept applications with a certain percentage confidence, which is essential for this work, rather than just give out the one that the operating application is most likely to be. Hence, another aim of these tests is to try to find the appropriate confidence threshold for the Chi-square detection, which means the degree of assurance that the result of the Chi-square detection is correct. Although the choice of confidence level is somewhat arbitrary, in practice 90%, 95%, and 99% intervals are often used [FerT89]. We chose 90% and 95% to be the candidate thresholds that may be appropriate for the detector.

6.2.1 Non-Nagle Applications

For those applications that do not use the Nagle Algorithm, the packet rates would not be affected by the network condition. The packets generated during a certain period would be fixed, the number of sampled packets depends upon the capture interval, hence, the capture interval was utilized as a parameter that can determine the reliability of packet size distributions for the non-Nagle applications and potentially affect the performance of the detector.

In the following tests, different capture intervals were used to build sample profiles for detection. All applications were monitored for fifteen minutes with the detector. When the appropriate capture intervals had been reached, the captured sample profiles were submitted to the *Detect()* thread, Chi-square values were then given to measure the reliabilities of the distribution profiles captured. For the purpose of comparison, only the results obtained up to 20th attempts are plotted.

Network Game Nadar

Three different capture intervals were tested for this application---15 seconds, 30 seconds, and 45 seconds. For this game, the packet size distribution profiles of outbound and inbound streams are very similar. Therefore, Only one direction is given here.

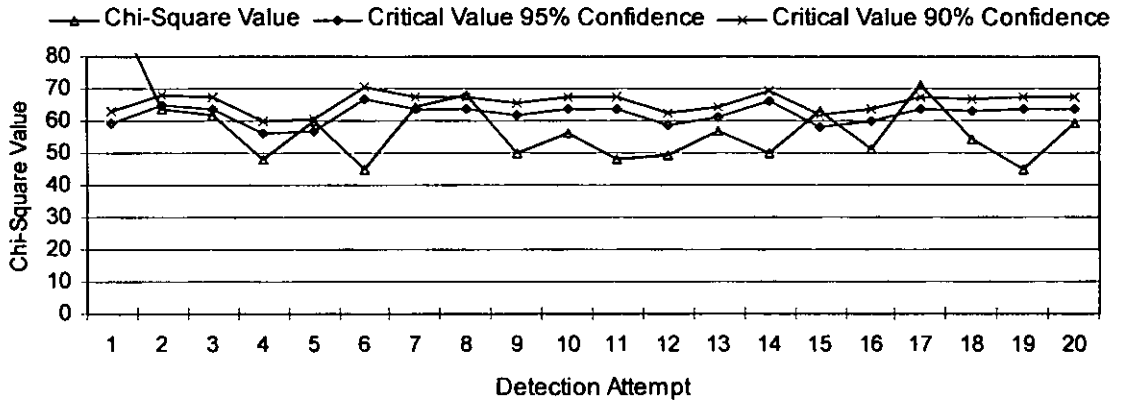


Figure 6.1 Chi-square results of Nadar's distribution profiles over 15 seconds sampling periods

In **Figure 6.1**, the test results for a capture interval of 15 seconds are given. The first detection attempt was rejected by both critical values of 95% and 90% confidence. This rejection could have been caused by some specific packets generated for exchanging control messages during the connection establishment period. Afterwards, Chi-square tests returned a series of Chi-square values with a mean of around 58. In a few cases, Chi-square values were greater than the associated 95% but lower than 90% confidence critical values. One attempt (attempt 17) rejected by both 95% and 90% confidence was observed. However, the detector had identified the application in most attempts.

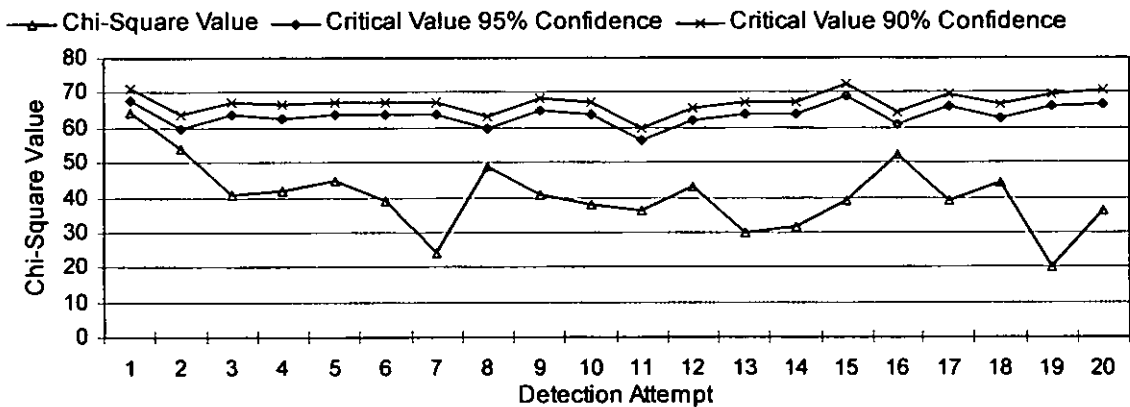


Figure 6.2 Chi-square results of the Nadar distribution profiles over 30 seconds sampling periods

Network Game Nadar

Three different capture intervals were tested for this application---15 seconds, 30 seconds, and 45 seconds. For this game, the packet size distribution profiles of outbound and inbound streams are very similar. Therefore, Only one direction is given here.

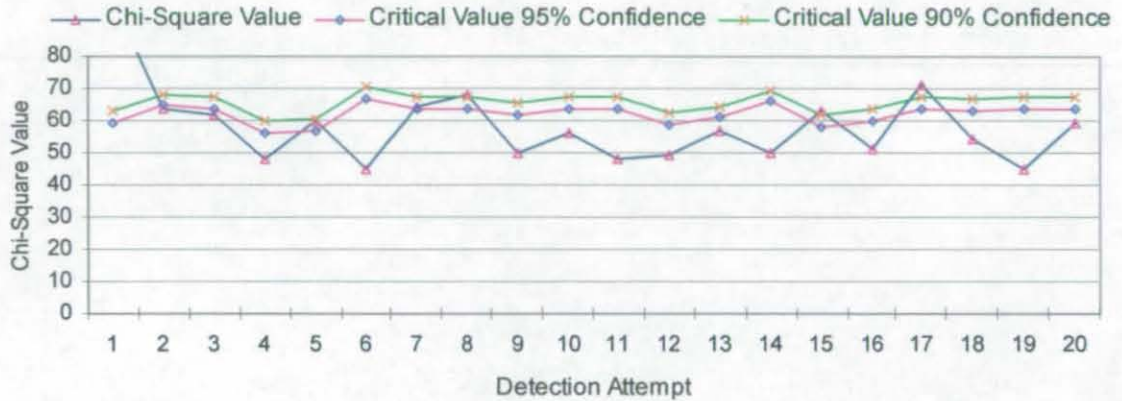


Figure 6.1 Chi-square results of Nadar's distribution profiles over 15 seconds sampling periods

In **Figure 6.1**, the test results for a capture interval of 15 seconds are given. The first detection attempt was rejected by both critical values of 95% and 90% confidence. This rejection could have been caused by some specific packets generated for exchanging control messages during the connection establishment period. Afterwards, Chi-square tests returned a series of Chi-square values with a mean of around 58. In a few cases, Chi-square values were greater than the associated 95% but lower than 90% confidence critical values. One attempt (attempt 17) rejected by both 95% and 90% confidence was observed. However, the detector had identified the application in most attempts.

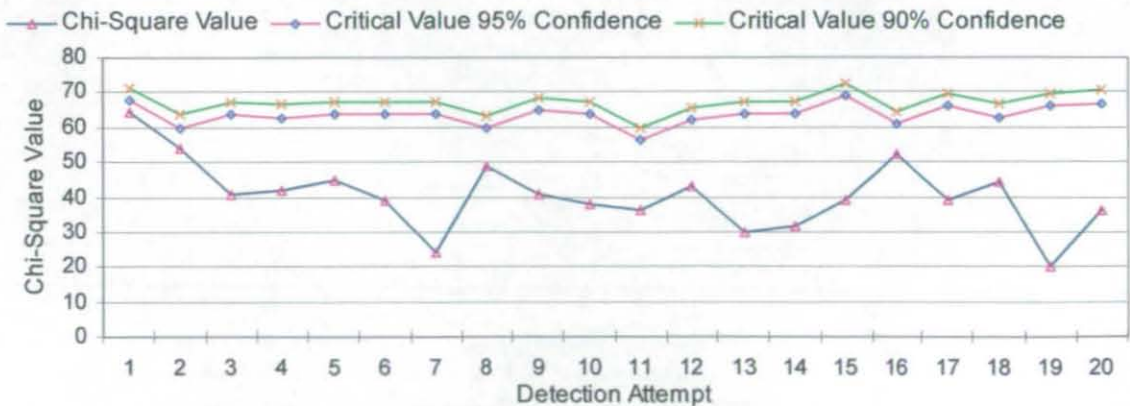


Figure 6.2 Chi-square results of the Nadar distribution profiles over 30 seconds sampling periods

Figure 6.2 shows the results for a capture interval of 30seconds. The first detection attempt still shows a high Chi-squared value; however, this could be accepted. The remaining detection attempts give a series of Chi-square values with a mean of about 40. They could all be accepted by the Chi-square tests as these values were much lower than critical values with 95% confidence. The profiles' reliabilities thus were improved considerably with this capture interval.

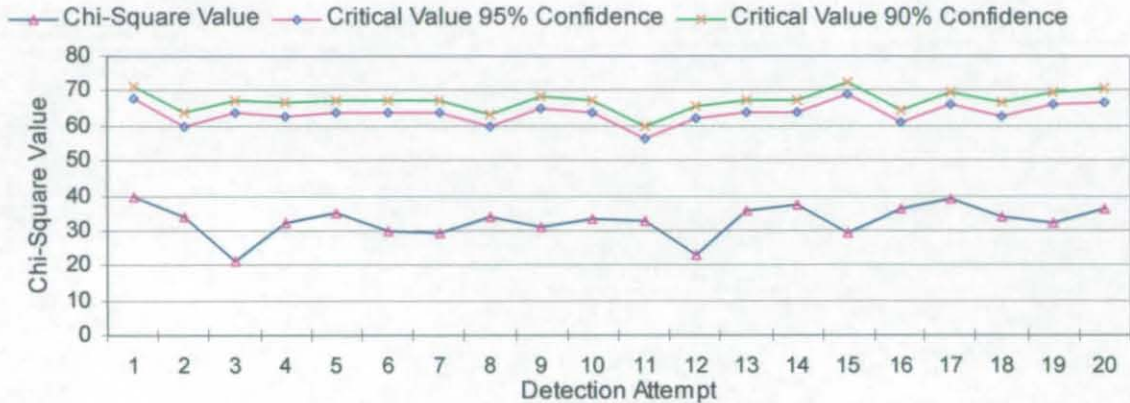


Figure 6.3 Chi-square results of Nadar's distribution profiles over 45 seconds sampling periods

As the capture interval was increased to 45 seconds, the Chi-square values returned provide a smoother profile. This indicates that the packet size distributions built over this capture interval were more stable than the previous ones. However, the mean of all the Chi-square values did not drop significantly (**Figure 6.3**).

Network Game Diablo II

The packet size distribution profiles of Diablo II [Bli01] are more complicated than those of Nadar. One would expect a longer capture interval to achieve decent profile reliability. For this application, the chosen capture test intervals are also 15 seconds, 30 seconds and 45 seconds. The results are shown in **Figure 6.4** blow.

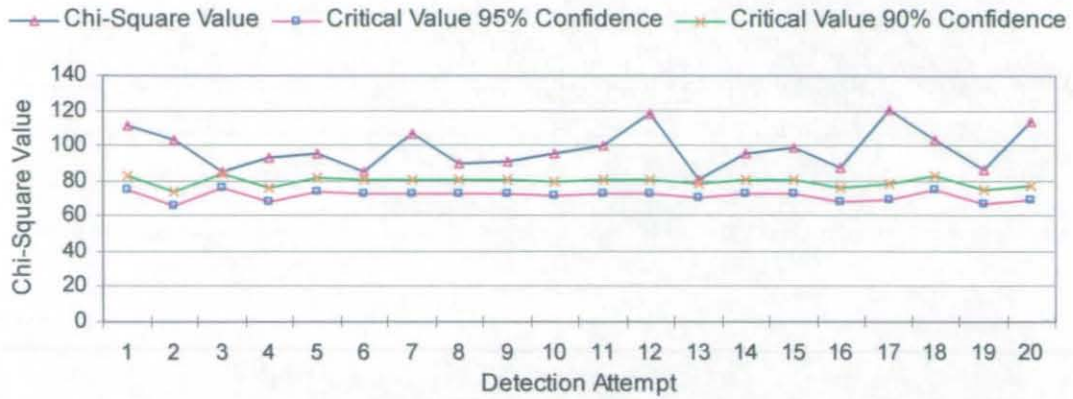


Figure 6.4 Chi-square results of Diablos' distribution profiles over 15 seconds sampling periods

In the resulting plot obtained over 15 second capture intervals, the profile reliability was not achieved as most detection attempts returned Chi-square values greater than the critical values for both 95% and 90% confidence which indicated unsuccessful detections. There were insufficient packets captured to allow a consistent distribution to be built.

In **Figure 6.5**, as the capture interval was increased to 30 seconds, the results were much better. Detection attempts were in general successful, but in four cases, the Chi-square values were still greater than 90% confidence critical values.

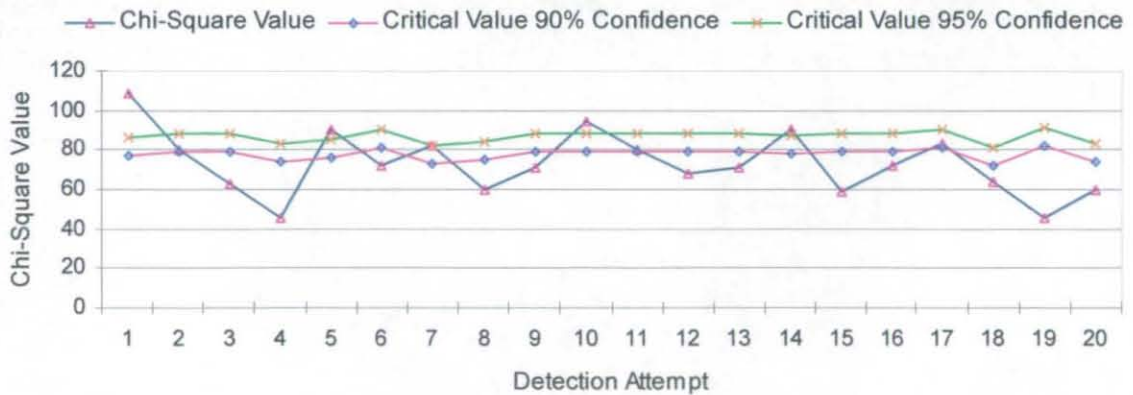


Figure 6.5 Chi-square results of Diablo's distribution profiles over 30 seconds sampling periods

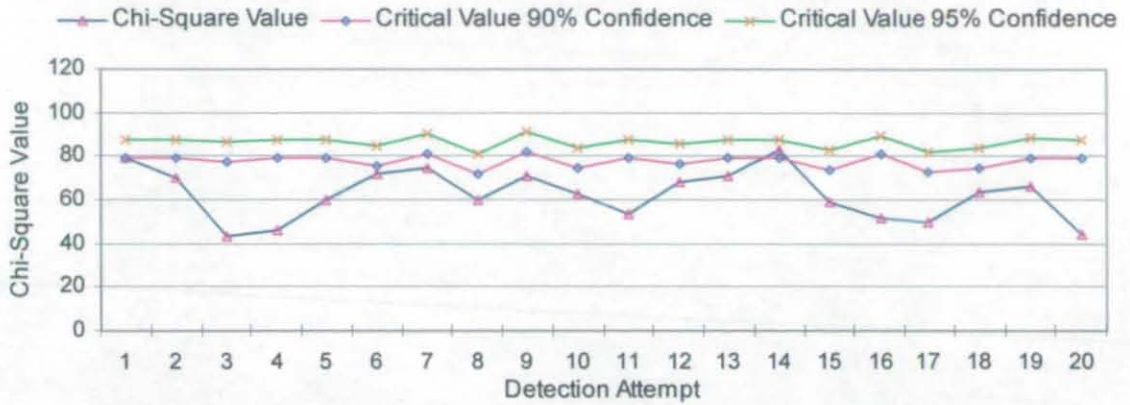


Figure 6.6 Chi-square results of Diablo's distribution profiles over 45 seconds sampling periods

When the capture interval was set to 45 seconds, considerable improvement of the distribution profiles' reliabilities was observed. From **Figure 6.6**, most Chi-square values were much lower than the 95% confidence critical values. Only one detection attempt was not (attempt 14), however, this was still very close to the 90% confidence critical value.

The three plots below (**Figure 6.7, 6.8, 6.9**) show the results of acceptance (by Chi-square tests) ratios for all available non-Nagle based applications over different capture intervals.

StoredProfile Number	Application Name
1-2	Nardar
3-4	Need For Speed III
5-6	Crimson Sky
7-8	Diablo II

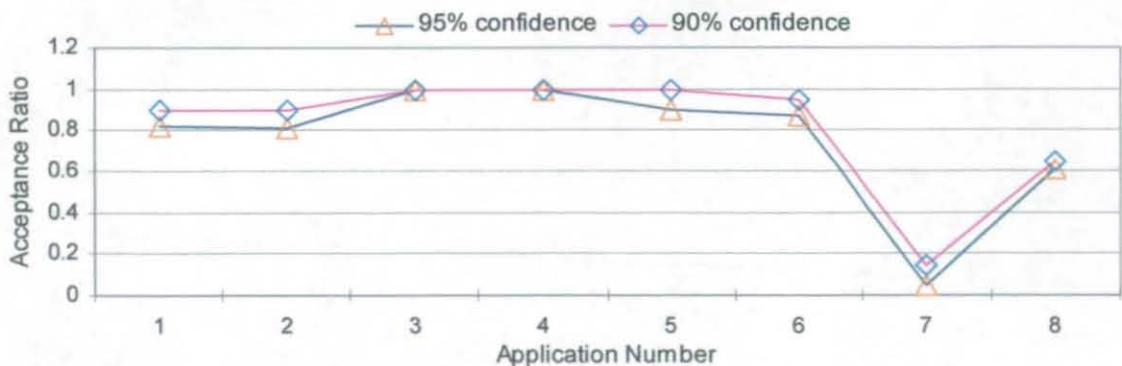


Figure 6.7 Acceptance ratios for all applications over 15 second sampling periods

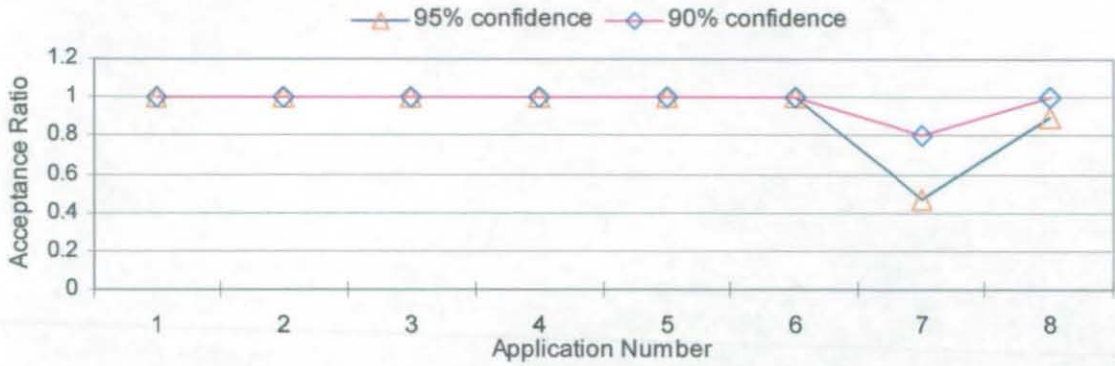


Figure 6.8 Acceptance ratios for all applications over 30 second sampling periods

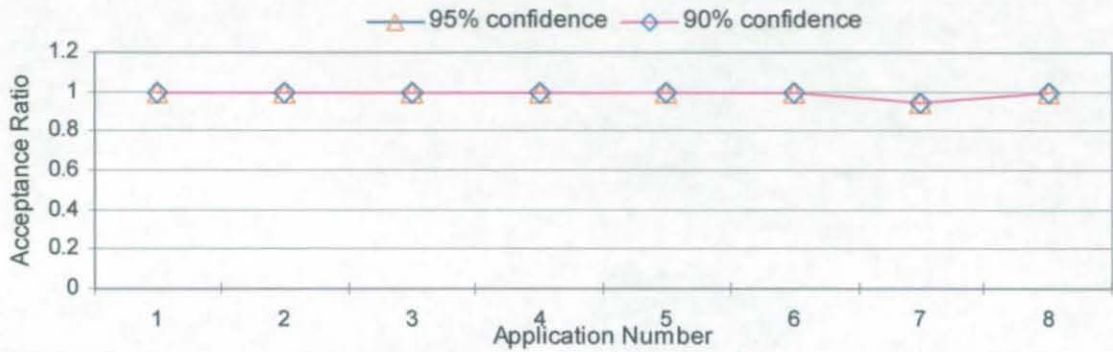


Figure 6.9 Acceptance ratios for all applications over 45 second sampling periods

In **Figure 6.7**, the acceptance ratio observed is not acceptable as the results for several applications are low. For Diablo, which has a complex packet size distribution profile, in the client to server direction, almost no successful detections were achieved with 95% confidence. Even with 90% confidence, the acceptance ratio was still quite low. It seems that over a 15 seconds interval, a consistent distribution cannot be obtained. In the second and third plots, the results are much better, most applications had achieved 100% acceptance ratios, while for Diablo, acceptable results were observed.

As the capture interval over which the profiles were built increased, the profiles' reliability did improve but the drawback was the fact that one had to wait longer between each detection attempt. In some cases, a few failures in detection could be considered to not seriously affect the performance of the detector in favour of a shorter capture interval. Hence, a capture interval of 45 seconds with a threshold confidence 95% is considered to be suitable for these non-Nagle applications.

6.2.2 Nagle-Based Applications

In contrast to the non-Nagle applications, as a packet will not be sent out until the acknowledgement of the last outstanding packet arrives, the packet rates of the Nagle-based applications could vary according to the worsening of network conditions. Thus, it may be difficult to find a fixed capture interval for a Nagle-based application. Instead, the number of sampled packets is considered to be a parameter which could potentially affect the performance of the detector.

In the following tests, which were similar to those of the non-Nagle applications, the tested applications were kept running and the *detect()* process was applied every time a given number of packets had been captured until 20 detection attempts had been achieved. Two loaded network conditions, which are jitter of 50 to 300 ms delay and a 300 ms fixed delay, were introduced by the LNE, so that the performance of the detector could be evaluated.

Real Network Application WarCraft III

Three different numbers of sample packets were tested, 400, 600, and 800. During the tests, the application was kept running, and once a test number of sample packets had been captured, detection attempts were carried out over the samples obtained.

Under 50-300 ms Jitter

Figure 6.10 shows the Chi-square test results over samples of 400 – packets.

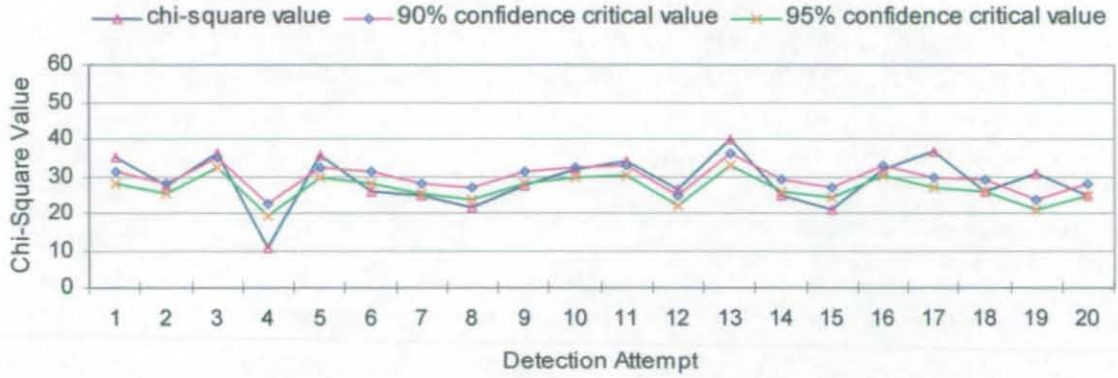


Figure 6.10 Chi-square results for WarCraft III's distribution profiles over 400 packets

In both directions (only the client to server direction plot provided), about one-third of the Chi-square values calculated were lower than the 90% confidence critical values, the others were higher. In the client to server direction, 8 detection attempts had returned Chi-square values greater even than the 90% confidence critical values, while that number is 10 for the server to client stream.

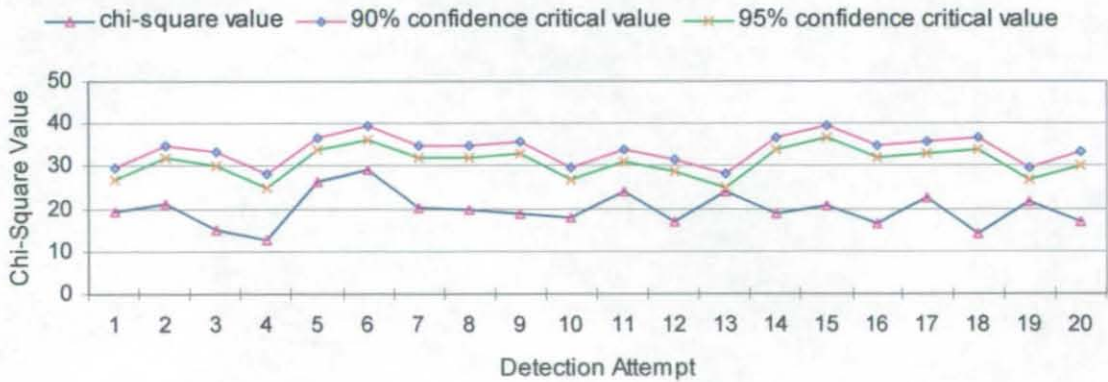


Figure 6.11 Chi-square results for WarCraft III's distribution profiles over 600 packets

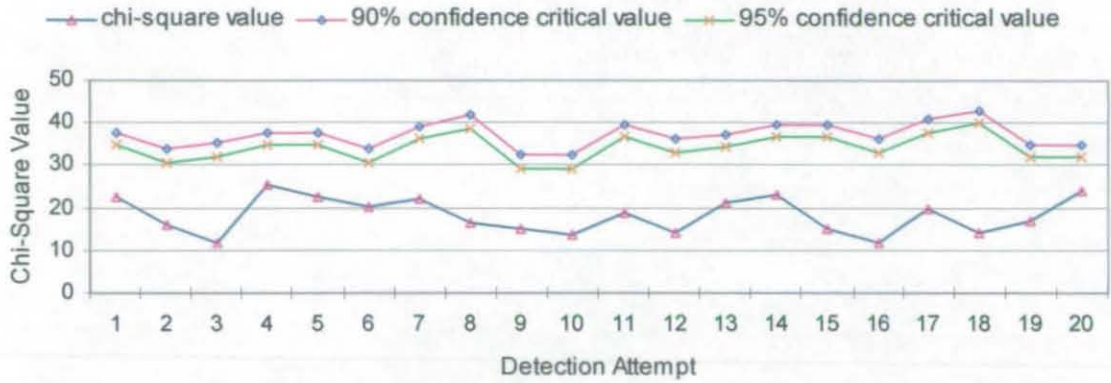


Figure 6.12 Chi-square results for WarCraft III's distribution profiles over 800 packets

Figure 6.11 and **Figure 6.12** show the Chi-square test results over samples of 600 and 800 packets. Drops in the mean of the Chi-square values were observed for these two sampling numbers compared with the results over 400 packets. In addition, all Chi-square values were much lower than both the 95% and the 90% confidence critical values. In the case where the sampling packet number increased from 600 to 800, there seems no visible performance improvement.

Under 300 ms Fixed Delay

Under 300 ms fixed delay, the results shown in **Figures 6.13, 6.14, 6.15** were obtained. The Chi-square values over 400 packets were erratic, many detection attempts returned values greater than the 95% confidence critical values, and even a few were rejected with the 90% confidence critical values. As the sampling packet number was increased to 600, the mean of Chi-square values showed a drop from 30 to 25. When the sampling packet number was increased to 800, no significant improvement was observed. On one occasion, the detector returned a very high Chi-square value, this happened as during this capture interval, the PC on which game client was running experienced a 100% usage ratio, and might have caused a temporary stop in sending/receiving packets. As such, this is considered to be a random inaccuracy and should not affect the results as a whole.

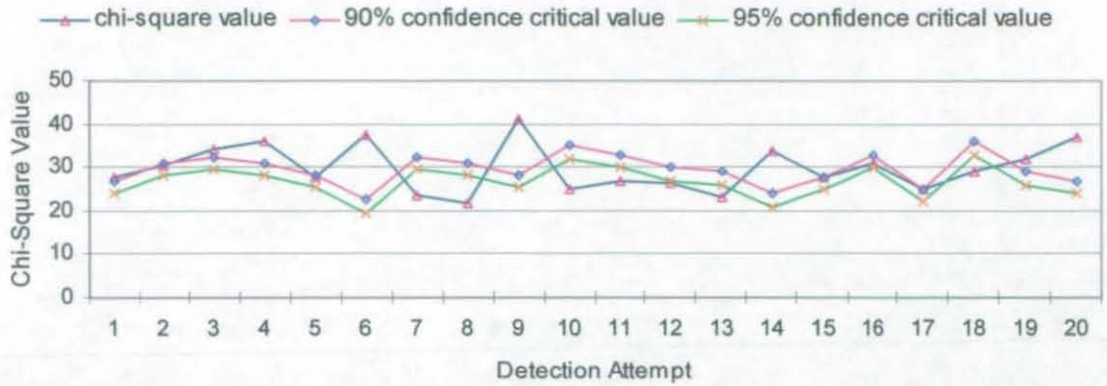


Figure 6.13 Chi-square results of WarCraft III's distribution profiles over 400 packets

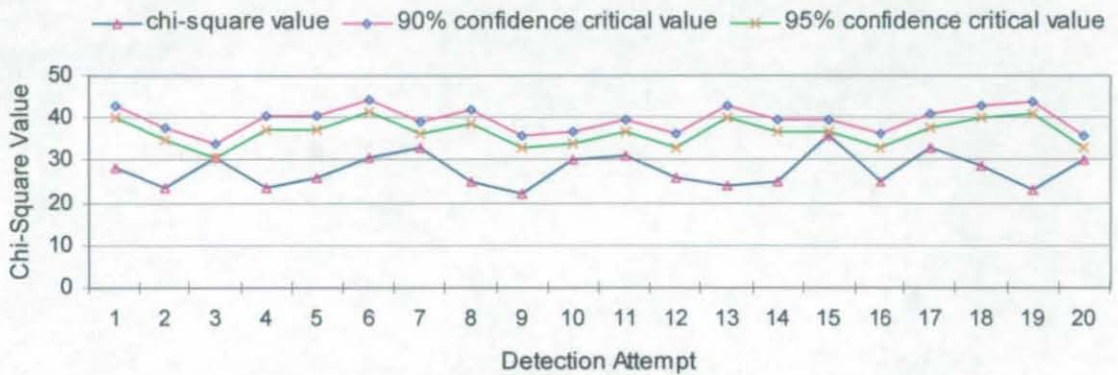


Figure 6.14 Chi-square results of WarCraft III's distribution profiles over 600 packets

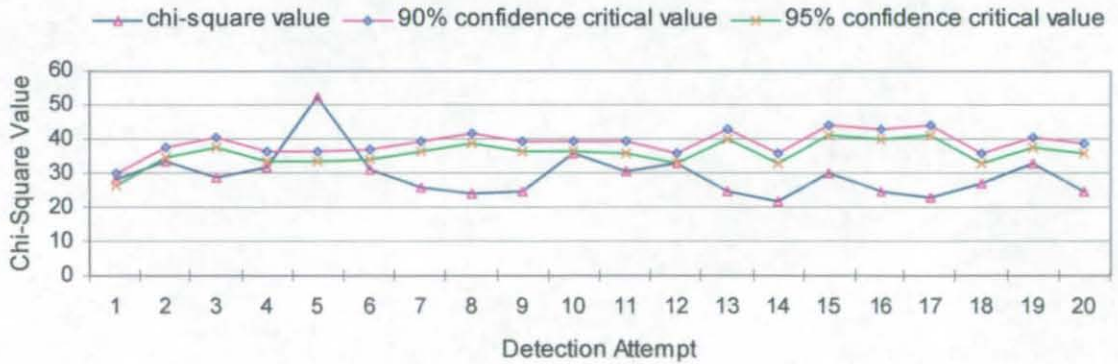


Figure 6.15 Chi-square results of WarCraft III's distribution profiles over 800 packets

6.2.3 Virtual Nagle-Based Applications

As there are very few Nagle-based applications to investigate, the analysis continued using a simulation. All virtual applications had been tested with various sampled packet numbers; 400, 600, and 800.

Figure 6.16 shows the mean of the Chi-square values calculated over different sampling packet numbers for all the virtual Nagle-based applications. Most applications show a considerable drop in value as the sample number increases from 400 to 600, and very little variation from 600 to 800. This is similar to the situation observed in WarCraft III. However, for a few applications, the mean kept dropping from 600 packets to 800 packets. Interestingly, these instances all occurred for those applications, which had large bin spreads in their packet size distributions. This is explainable as with the increase of the distribution freedom, more sample packets would be needed to establish reliable and stable distributions. In a few instances, the means of the Chi-square values showed no significant variation as the sampling number increased. These applications often had a small spread of packet size distributions, and contrary to those with large spreads, 400 packets were sufficient to establish reliable packet size distributions.

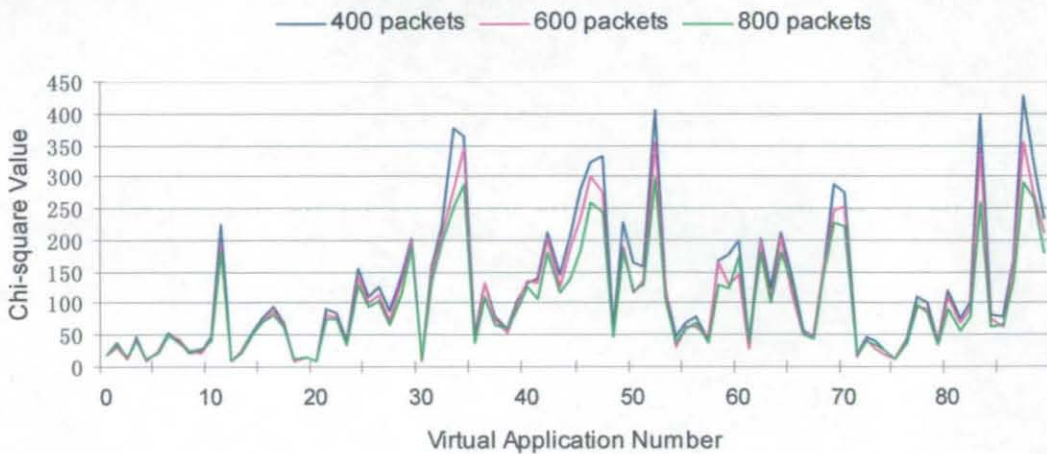


Figure 6.16 Chi-square results for Virtual Nagle-based applications for different numbers of sample packets

Figures 6.17 and **6.18** plot the acceptance ratios with 95% and 90% confidence for 600 sample packets under a jittered delay network and a 300 ms fixed delay network. Except for those applications with very small spreads, most applications have poor acceptance

rations for 95% confidence. For some applications, acceptance ratios below 70% were observed with 95% confidence. When it comes to 90% confidence, many applications show improved the acceptance ratios than 95%, the lowest acceptance ratio was greater than 80% and 100% acceptance ratios for 2/3 applications were obtained.

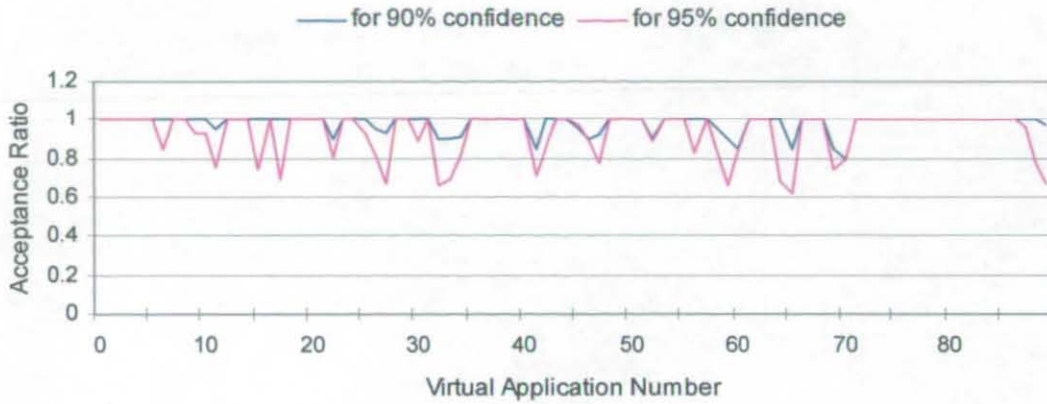


Figure 6.17 Acceptance ratios for all virtual Nagle-based applications under a jittered delay network condition over 600 packets

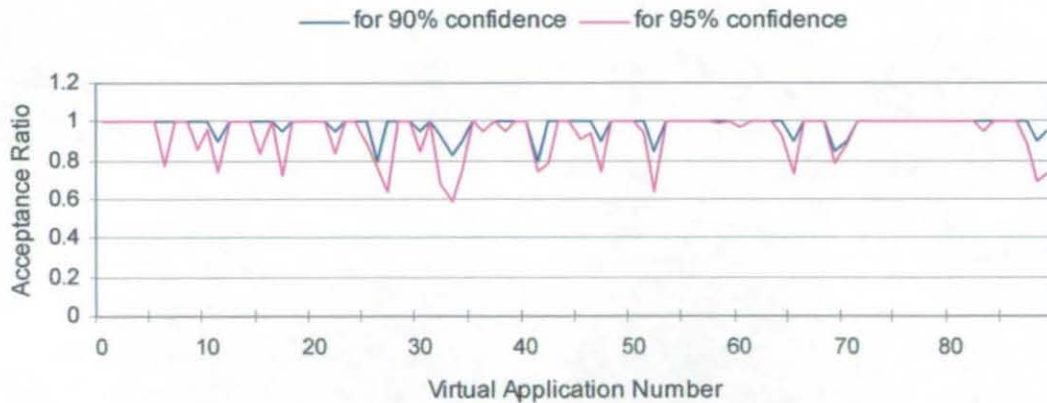


Figure 6.18 Acceptance ratios for all virtual Nagle-based applications under a 300ms fixed delay network condition over 600 packets

When the sampling packet number was set to 800, considerable performance improvements were observed for many applications with large spreads. Under both jittered delay and 300ms fixed delay, most of them achieved acceptance ratios that were greater than the 90% using the 90% confidence critical values, with three showing 83%, 85% and 88% in **Figure 6.19** and other two showing 84% and 88% in **Figure 6.20** respectively. For the 95% confidence, the results obtained still did not reach an acceptable level, one may expect improvement on acceptance ratios with the increase of the

sampling packet number. However, it will take too long a time in order to capture packets for a practical detector if sampling packet number is increased further. The settings of the detector were thus set as sampling packet number 800 and threshold confidence 90%.

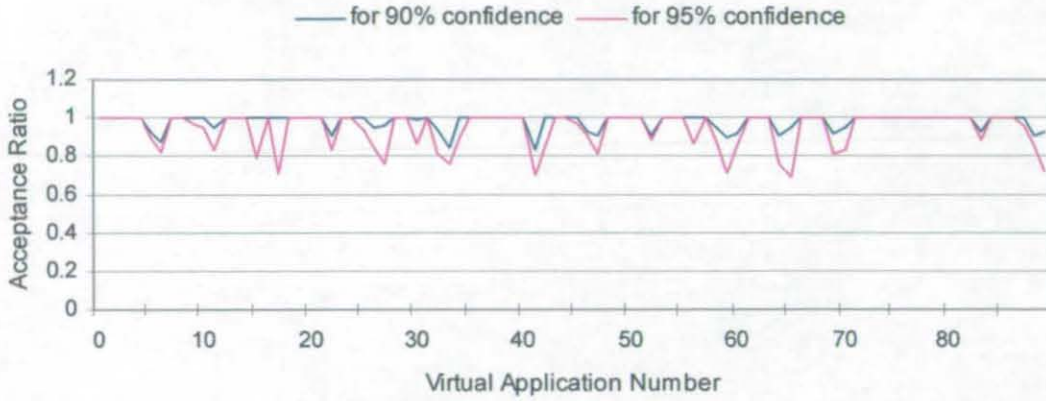


Figure 6.19 Acceptance ratios for all virtual Nagle-based applications under a jittered delay network condition over 800 packets

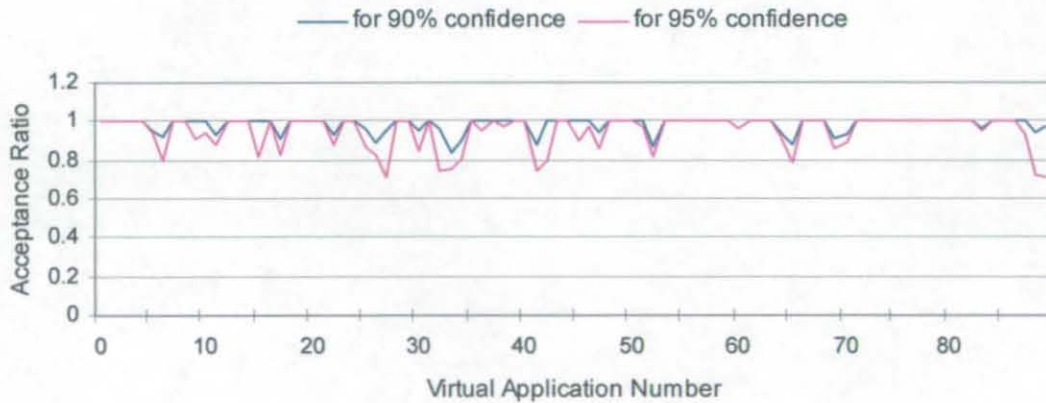


Figure 6.20 Acceptance ratios for all virtual Nagle-based applications under a 300 ms fixed delay network condition over 800 packets

6.3 Application Detector Performance

After the parameters had been decided, it was necessary to test the performance of the prototype application detector. The following tests were carried out on all stored applications. The testing system architecture was the same as that on which the previous tests had been carried out. The PCs on which tested applications were running were connected using 10MB Ethernet cards.

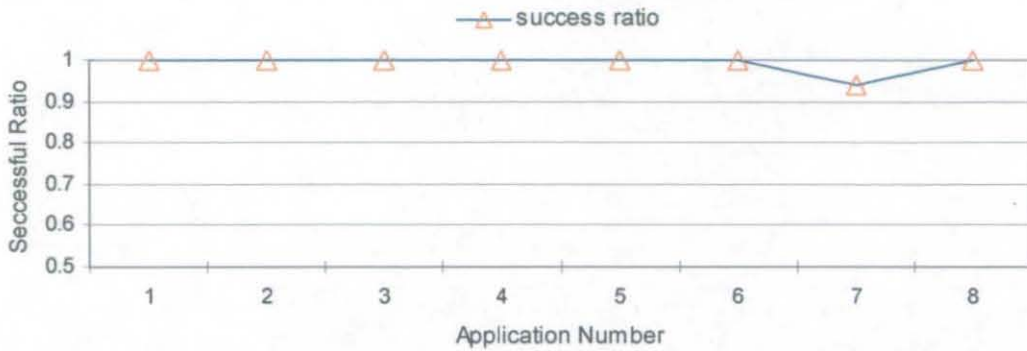


Figure 6.21 Success ratios for all non-Nagle applications under ideal network condition

Under Loaded Network Conditions

A suggestion was made and proven in Chapter 4 of this Thesis that the presence of network load would not affect the packet size distributions of the Non-Nagle applications; hence, the performance of the detector should not be impacted by loaded network conditions. This assumption was tested here with the prototype detector. The test network was artificially loaded at different levels and the performance of the detector measured with various Non-Nagle applications.

Figure 6.22 plots the success ratios for all the non-Nagle applications under different network conditions, three levels of load were emulated with the LNE which were low-load (100ms fixed delay, 3% loss), medium-load (jittered 50-300ms delay, 3%loss), and high-load (300ms fixed delay, 5% loss). Similar results to those obtained from tests under ideal network conditions were observed. The detector therefore performed well over a loaded network. Even at the highest level of load, the success ratios were acceptable as most applications retained 100% success except for Diablo II which achieved a 92% success ratio.

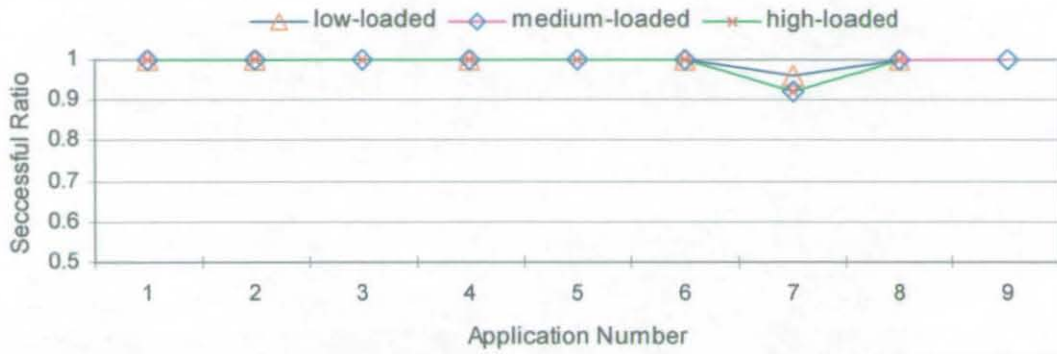


Figure 6.22 Success ratios for all Non-Nagle applications under loaded network condition

6.3.2 Nagle-Based Applications

The Nagle-based applications were then tested on the same experimental system as the Non-Nagle applications. The sampling packet number was assigned to 800 as described previously and detection attempts were carried out each time 800 sampling packets had been captured. The build time of a test profile varied according to the network conditions, basically, for an application which has an original packet rate of 7/sec, under a medium level load network, the build time would vary from 80 seconds to 200 seconds approximately.

In the following tests, each Nagle-based application in the database was run 10 times for a running time of at least 15 minutes and would stop once 20 detection attempts had been made. As such, during the operation of each application, the number of detection attempts was more than 200 in total over which success ratios were calculated.

Under Ideal Network Conditions

Under the no-load network, the detector functioned perfectly as shown in **Figure 6.23**. All detection attempts had returned the correct application name, no misdetection occurred.

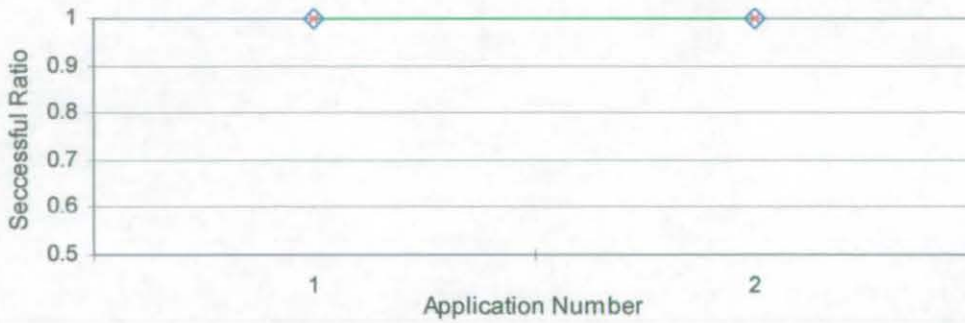


Figure 6.23 Success ratios for real Nagle-based applications under ideal network condition

Under Loaded Network Conditions

Three levels of load were adopted for the two real Nagle-based applications. Under these loaded network conditions, the aggregated packet size distribution mechanism was operational. The general results are summarized in **Figure 6.24**

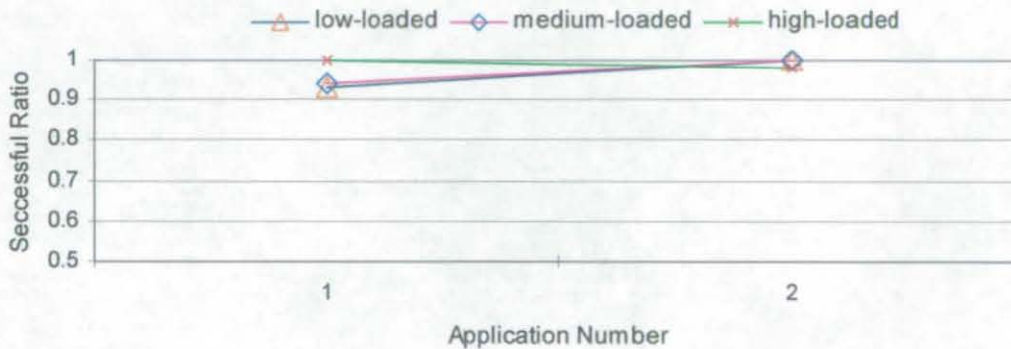


Figure 6.24 Success ratios for real Nagle-based applications under loaded network condition

Under low-load and medium-load network, the success ratios of both applications were good, for SSH-Client, 100% success was even achieved. Variation was observed under the high-load network condition. The success ratio for WarCraft III was improved, with a small drop for SSH-Client when the network condition was worse. These results could be caused by the fact that when 300 ms fixed delays were introduced by the LNE, very few first order packets were captured. This could lead to a decrease in the degree of freedom of the aggregated packet size distribution profile for WarCraft III and therefore improved the reliabilities of the Chi-square tests. However, for SSH-Client, under the high-load network condition, some more of the fifth order packets were captured. As the detector

was set to deal with the 4th order packets at most, those 5th order packets could have caused some misdetection.

6.3.3 Virtual Nagle-Based Applications

The tests on the virtual Nagle-based applications were carried out only under loaded network conditions. **Figure 6.25** shows the success ratio results of these applications.

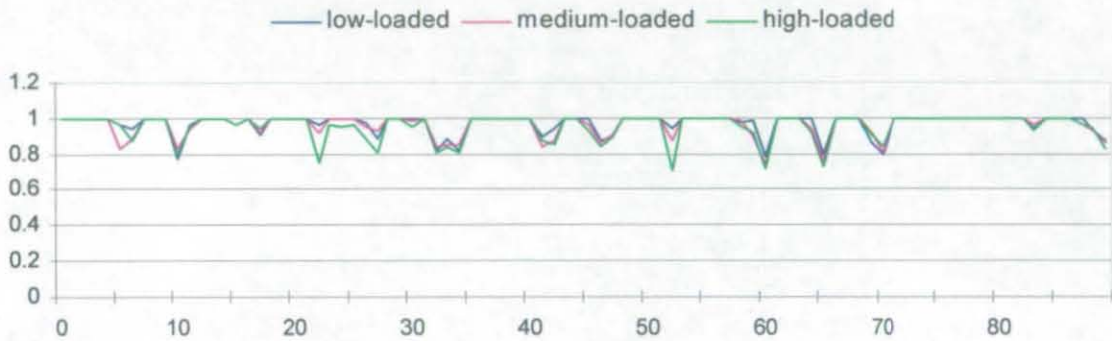


Figure 6.25 Success ratios for all virtual Nagle-based applications under loaded network condition

For the reason that these virtual Non-Nagle applications had much more complicated packet size distributions, the success ratios were generally lower than those of the real Nagle-based applications. Under the low-load and medium-load network conditions, some virtual applications with small spreads achieved 100% success, whereas, some large spread applications gained lower success ratios. The success ratios showed positive correlation to the spreads of the applications. The larger spread the application has the lower success ratio the detector achieved. When the application were operated under high-load network conditions, some applications that use the second aggregated detection method showed slight drops in success ratios, these drops could have been caused by the situations mentioned in Chapter 4. This suggested that when the aggregated packet size distributions were mostly consist of higher order packets, the detection attempts will be applied once a certain order packets are just removed from the sample distributions, however, at that moment, some lower packets are still remained in the sample distributions, as such, the detection attempts would be affected by the presence of these

lower order packets. Nevertheless, the effects were not serious and the success ratios kept in acceptable levels.

Actually, it is possible that a sample aggregated packet size distribution could be considered to be aggregated from more than one original stored distribution. A simple aggregated distribution is given in **Figure 6.26**. Another two simple original distributions are also shown in **Figure 6.27**.

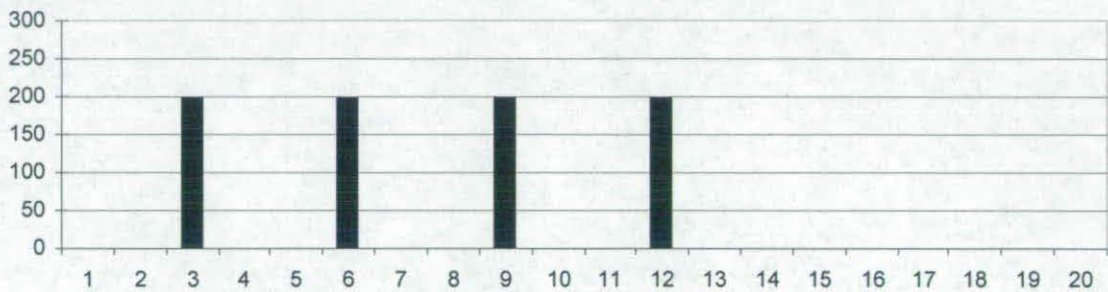
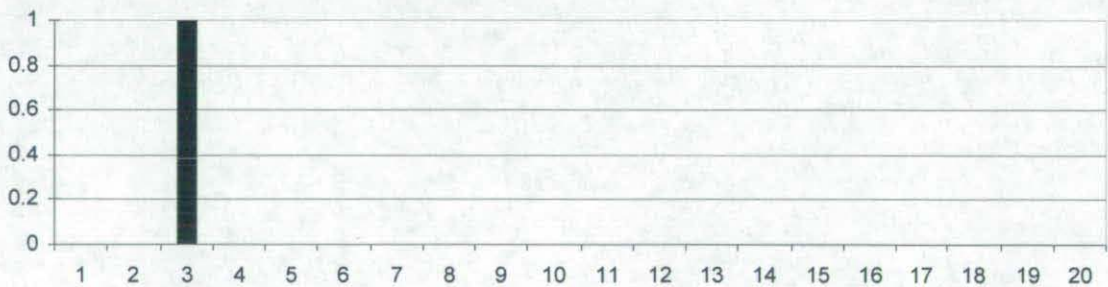
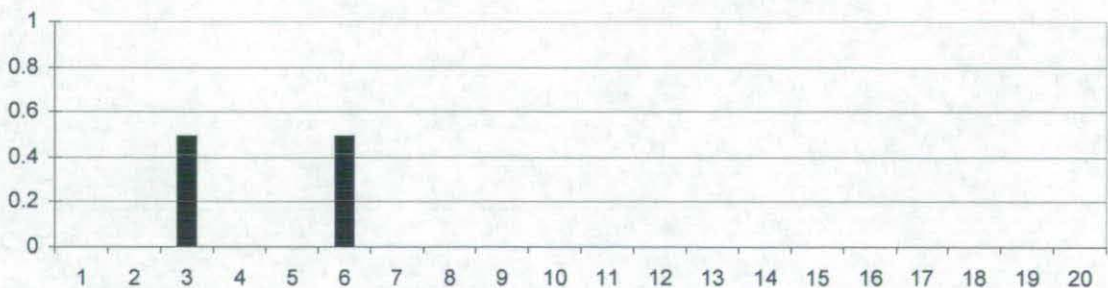


Figure 6.26 A simple aggregated distribution profile



a



b

Figure 6.27 Two simple original distribution profiles

The aggregated distribution profile given out in **Figure 6.26** consisted of 200 packets of 3 bytes, 200 packets of 6 bytes, 200 packets of 9 bytes and 200 packets of 12 bytes. This

profile could be considered to be aggregated from the original distribution shown in **Figure 6.27 (a)** with 200 1st order packets, 200 2nd order packets, 200 3rd order packets and 200 4th order packets. On the other hand, it could also be considered to be aggregated from the original distribution shown in **Figure 6.27 (b)** with 400 1st order packets and 400 2nd packets. In reality, this instance could also potentially occur. The detector thus would sometimes return more than one application that was accepted by the Chi-square test for the Nagle-based applications. An immediate example of this situation is virtual application no.61, which has a success ratio (as shown in **Figure 6.25**) that is obviously lower than the acceptance ratio obtained previously. In these cases, the detector may have no ability to achieve a unique identification. However, at least the possible applications had been reduced into a significantly smaller range.

In **Figure 6.28**, the detection success ratios of the virtual Nagle-based applications when success was redefined as “the correct application had been accepted by Chi-square test regardless of whether or not other incorrect applications were also accepted” are plotted.

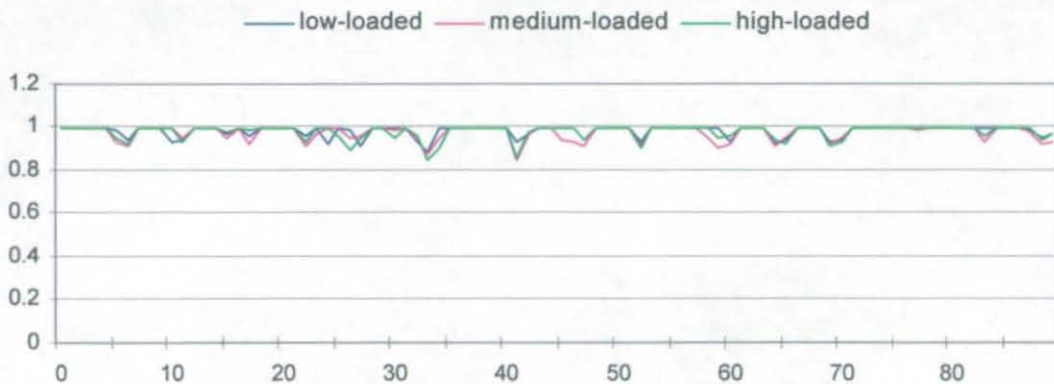


Figure 6.28 Success ratios for all virtual Nagle-based applications under loaded network condition

From the plot, it can be seen that for the virtual applications, the success ratios were improved considerably. The lowest success ratio observed was 85%, which is quite reasonable. Since the real Nagle-based applications would probably have much less complicated spreads than the virtual applications, one could expect higher success ratios when a real detector is used.

6.3.4 Virtual Applications with Large Packet Sizes

In Chapter 4, the suggestion that the aggregated detection mechanism shows poor ability to identify those applications with large size packets was discussed. Here, five virtual applications falling into this category were tested. These applications' packet size distributions have the same median position which is 500 bytes but different spreads and trends.

Figure 6.29 plots the success ratios for these virtual applications under low-load network conditions. The delay was set to 150 ms. It can be seen that the results exhibited are acceptable as the lowest one represents a 83% success ratio. This result suggests that under this low-load network condition, most aggregated packet sizes did not reach the MTU size, and the profiles captured showed in general no difference from those applications with smaller packet sizes.

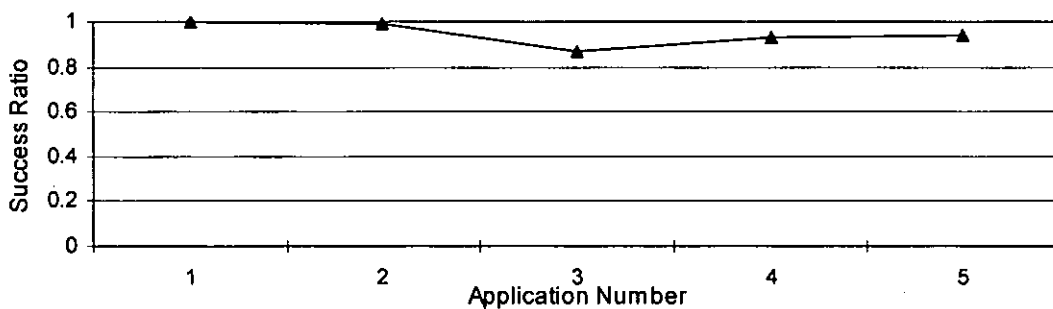


Figure 6.29 Success ratios for Nagle-based applications with large packets under low-load network condition

6.4 Simultaneously Running Application

Clearly, a practical detector will need to identify applications running simultaneously on multiple clients across the network [Bha01]. Some tests on simultaneously running applications were therefore carried out.

In the first round test, three applications were operated on a single PC. In the second round, three applications were operated on three separate PCs, which were connected to a

hub, and the detector was attached to the hub in order to analyse the traffic. The aim of these tests was to verify the ability to identify applications in the presence of other traffic.

The three applications were Crim-Sky, Need for Speed III and WarCraft III. Network load of jittered 50-300 ms delay in conjunction with 5% loss was introduced by the LNE. The three applications were operated only once for twenty minutes during which several detection attempts were made. **Table 6.1** below shows the success ratios for the three applications of this test.

Test round	Crim-Sky	Need for Speed III	WarCraft III
1	100%	100%	99%
2	100%	100%	100%

Table 6.1 Success ratios for simultaneously running applications

According to the results obtained, the detector performance showed no difference from that when the applications were running individually. This test did demonstrate that the detector has the ability to distinguish the identities of the streams of multiple applications, all present at the same time.

6.5 Summary

Some parameters of the detector prototype are found in this chapter. For the non-Nagle applications, capture interval is selected as a parameter as the sending of packets is not affected by the TCP implementation and the number of packets in one capture interval is fixed. However, for the Nagle-based applications, this number will be potentially changed. The number of sample packets is therefore chosen as the corresponding parameter.

An interval of thirty seconds is enough for most non-Nagle applications except DiabloII due to its complex distribution profile. A better performance was observed when the interval increased from 30 seconds to 45 seconds, and no significant change as the interval increased further to 60 seconds. Most Nagle-based applications exhibited a similar characteristic. Performance was improved as the sampling number was set from

400 to 600 but little variation from 600 to 800. However, for a few applications with large spreads of distribution profile, performance can be even improved further when sampling number increased from 600 to 800. Nevertheless, as discussed previously, it will take too long a time in order to capture packets for a practical detector if sampling packet number is increased further. In order to achieve the best balance between detection reliability and timeliness of the identity information, 45 seconds and 800 were selected to be the parameters so that stable captured distributions can be established.

The Chi-square tests were applied on the captured distributions. Most applications can be successfully detected using 95% as the threshold confidence. For several virtual applications with very complicated distribution profiles, 90% is considered to be more suitable. The settings of the detector are summarized in **Table 6.2**

	Capture Interval/Sampling Number	Threshold Confidence
Non-Nagle Applications	45 seconds	95%
Nagle-based Applications	800 packets	90%

Table 6.2 Parameters selected for the prototype detector

A lot of tests had been carried out in order to see how well the mechanism was working. All real applications including eight Non-Nagle applications and two Nagle-based applications can be identified without any difficulty. The success ratios are all more than 90%. For those virtual applications with complicated profiles, drops on the success ratios were seen comparing to those virtual applications with simple profiles. Nevertheless, most of them achieved success ratios more than 85%. The lowest one 77% was occurred when it tried to detect virtual application 61, which can be still considered as acceptable. The results were summarized in **Table 6.3**. The numbers in the table show the count of applications achieved the corresponding successful ratios.

	100%	99%-90%	90%-80%	<80%
Non-Nagle Applications	8	1	0	0
Nagle-based Applications	61	17	12	2

Table 6.3 Success ratios for strictly defined "success"

As the "success" is defined strictly, improvement can be seen if the definition is changed to "the correct application had been accepted by Chi-square test regardless of whether or not other incorrect applications were also accepted". In this circumstance, many applications achieved better successful ratios (see **Table 6.4**). The lowest success ratio observed was 85%, which is quite reasonable.

	100%	99%-90%	90%-80%	<80%
Non-Nagle Applications	8	1	0	0
Nagle-based Applications	59	31	2	0

Table 6.4 Success ratios for less strictly defined "success"

Some tests on simultaneously running applications were afterwards carried out. The results suggested that the prototype detector did have the ability to identify network applications with other traffic in present.

CHAPTER

7

Conclusions and Recommendations for Future Work

7.1 Introduction

In this chapter, the conclusions on the study undertaken in this Thesis are drawn. The results based upon the experiences are summarized. Finally, the recommendations of future work arising from this work are discussed.

7.2 Conclusions

To know what applications are currently in operation across modern packet based communication networks such as the Internet is always attractive to network administrators, network service providers and security systems. The availability of this information can contribute to preventing improper network use. In addition, using this information, the network may be able to establish enhanced environments for the applications which are in use. In such cases, an efficient mechanism for identification of the application generating a traffic stream is required.

Traditionally, application detection has been based on well-known port numbers. All applications using TCP/IP as their transport protocol must use a port number with which to identify the packets generated. This simple identification mechanism has however become much less accurate for identifying applications. A recently developed technique, deep packet inspection, is a process which involves searching for uniquely identifying information held within the data portion of the packets. Such mechanisms are accurate enough for application detection, yet have shortcomings as well. Although this technique

functions well for standards-based applications, other applications for which the relevant technical information is not forthcoming would not be detectable.

An approach for detecting networked applications using an alternative application “signature”, the Packet Size Distribution, has been proposed and discussed in this Thesis in order to identify those TCP-based real-time applications that are not standards-based.

For TCP-based real-time applications utilising bulk mode or interactive mode without the Nagle Algorithm, the packet size distribution profiles show robust consistency under different running conditions or under loaded network conditions. However, some traditional TCP-based applications such as FTP, SMTP, and HTTP could not be identified using this fingerprint due to the fact that the packet size distributions of these applications highly depend on their users’ input and had poor consistencies.

On the other hand, those applications using the Nagle Algorithm present variations in the packet size distribution profiles with the worsening of the supporting network condition. The Nagle Algorithm says that when a TCP connection has outstanding data that has not been acknowledged, small packets cannot be sent until the outstanding data is acknowledged. The size of packet generated by these applications would therefore be aggregated and the distribution profiles would look quite different from those under ideal network conditions. However, the response to such aggregation phenomenon is seen to follow a predictable pattern in many cases, and a solution to their detection has been arisen and discussed in Chapter 4. This was experimentally proved and assessed for its feasibility. A variety of packet size distribution profiles have been tested, and satisfactory detection results have been obtained showed that the method is suitable universally for different kinds of distributions.

Two limitations were discovered with the aggregated detection methods. The first is that when the packets generated by the applications are relatively large, the MTU of the subnet would limit the size of the outgoing packets leading to many payloads being sent out as the MTU size packets. The aggregated distribution would thus become much less predictable and make it difficult for an aggregated detection mechanism to achieve successful detection. Another drawback is that it could potentially happened that an aggregated packet size distribution could be generated that was aggregated from two or more completely different original distributions. In this case, the detection attempt would

return two or more “correct” results, and there is no way so far to tell which one is actually correct. Nevertheless, the range of possible applications would be considerably diminished by the aggregated detection mechanism, and in any case this circumstance has a very low probability of actually occurring.

It was found that a relatively long capture interval is needed by these non-Nagle applications in order to build reliable packet size distribution profiles, and for these Nagle-based applications, the packet count would vary according to different network conditions. As such, the number of packets captured was chosen as the key parameter in order to guarantee the stability of the distribution profiles generated. As the interval or number of packets captured over which the profiles were built increased, the detector performance improved but the drawback was the fact that one had to wait longer between each detection attempt. Tests were carried out in order to achieve the best balance between detection reliability and computation resource consumption or timeliness of the identity information.

Chi-square analysis was adopted to statistically compare the captured packet size distributions against a database and was considered a successful mechanism for this work. The threshold confidence could be another factor which may affect the performance of the detector.

A prototype detector implementing the ideas of this work was developed and tested. For non-Nagle applications, the performance was perfect since excellent success ratios had been obtained for all applications tested. For the two real Nagle-based applications, WarCraft III and SSH-Client, perfect performance was also observed, whereas, some drop in success ratios for the virtually generated Nagle-based applications was seen. This could be explained as the packet size distribution profiles of these applications are often much more complicated than those real applications. However, the detector successfully identified these virtual applications in most cases and the success ratios were kept at an acceptable level.

In some cases, detection was unsuccessful on the first attempt, but improved with subsequent attempts, as each new sample of the stream was captured. This was often due to the application not having reached a steady state of operation. With many applications, this was usually when connection negotiation was still occurring.

7.3 Future Works

The packet size distribution has been verified for its ability to be an identifiable signature for some classes of TCP-based applications. This section aims to briefly introduce some of the research issues related to the work discussed in this Thesis. Investigation into these issues provides knowledge to improve the application detection mechanism.

The probability of encountering a packet size distribution profile which is similar to one profile in the database but generated by a different application potentially exists. That is, for the TCP-based applications whose packet size distribution profiles show no consistency under varied conditions, sometime, the distribution profiles generated by them could exhibit similarity to a certain profile in the database which is captured from an application has consistent packet size distributions. Although this should be a low-probability event, it could potentially occur. As a result, a misdetection result may be returned, and cause a network operator to draw incorrect conclusions about the utility of his/her network and perhaps lead to incorrect optimisation. In addition, clearly, as more applications are added to the database, the likelihood of two or more applications having very similar packet size distribution profiles increases. In such cases, two suggestions can be made in order to reduce the likelihood of the said misdetection. Firstly, whilst Chi-square is a suitable analysis technique for this work as it is a distribution-independent test and has the ability to reject poorly matched profiles some of the matching techniques employed in pattern recognition may be applicable to application detection [DudH73]. The second suggestion is to include more application related information during the detection procedure. Some information contained in the headers of the packets could not be considered as unique. Nevertheless, it could be used to reduce the set of applications against which the sample distribution profile needs to be compared. Further, some additional statistical measurements such as packet rates or packet sending intervals could contribute to reduce the comparison domain.

Some applications (i.e. FTP) follow a distribution consisting of predominantly large (MTU) and small (ack) packet sizes. Whilst the unique identification of such applications is not possible, it is clear that only certain TCP applications show this characteristic. The detection mechanism could still have benefit however. The analysis could be used to

verify that the application is indeed what it claims to be. For example, the approach could not formally identify an FTP stream on Port 20, but it could be used to *reject* such a classification if, for example, Nadar traffic was masquerading on Port 20.

7.4 Contribution Remarks

The statistical application detection mechanism was first used by [Bha01] which discussed and demonstrated the feasibility of using the packet size distribution as the fingerprint in order to uniquely identify an UDP-based application running on a network. With the improvement of the quality of the Internet, one may expect more and more networked applications that adopt TCP protocol will be developed in the recent future, and it is as such necessary and practical to apply this mechanism on the TCP-based applications.

The statistical technique is capable of identifying the non-Nagle based applications with remarkable success and shows resistant to network load. However, due to the difference between the UDP and TCP transportation protocol, when this detection method was applied on the TCP-based applications, some problems were emerged and makes it is not applicable to detect some Nagle-based applications. The statistical technique was then improved by this work and experimentally shows reasonable capability in identifying those applications.

The pattern matching method adopted by this work is the Chi-square test which is a simple statistical method based on probability theory. One can certainly expect that the performance will be improved when some more sophisticated pattern recognition methods are utilized.

The idea of this work is not intend to replace the traditional content-based detection techniques but try to provide a complementary detection method in detecting some kind of networked applications on which conventional packet analysis often fails. It was proven to be suitable for the real-time applications. In addition, this work gives an option while detecting some applications that exchanging encrypted traffics such as SSH which is a typical application falling into this category. In summary, the results achieved by this work not only contributes to the research in the field of the traffic stream analysis as well

as the network monitoring, but provides a useful technique while developing a practical intelligent network.

References

- And05 O. Andreasson
 “Iptables Tutorial”
 <http://www.netfilter.org>, 2005
- Acts03 Advanced Communication Technologies and Services (ACTS)
 Programme
 “ATM in Europe”
 <http://www.dit.upm.es/infowin/atmeurope/>, 2003
- AllP99 M. Allman and V. Paxson
 “TCP Congestion Control”
 Request for Comments, RFC1122,
 Network Working Group, 1989
- ArmS04 G. Armitage and L. Stewart
 “Some Thoughts on Emulating Jitter for User Experience Trials”
 Proceedings of the 3rd ACM SIGCOMM workshop on Network and
 system support for games, Portland, USA, Aug 2004, pp.157-160
- AshM99 P. Ashley and V. Mark
 “Practical Intranet Security”
 Kluwer Academic Publisher, 1999, ISBN: 978-0-7923-8354-3
- BaiSWC00 G. Bai, Z. Shen, W. Wang and S. Cheng;
 “RSTP: a new lightweight transport protocol for VoIP”
 Communication Technology Proceedings, 2000. WCC - ICCT 2000.
 International Conference, vol. 1, Beijing, China, Aug 2000, pp. 639 -
 642

- Bas98 O. Bashir
 "Management and Processing of Network Performance Information"
 Doctoral Thesis, Loughborough University, 1998
- Bha01 K. Bharadia
 "Network Application Detection Techniques"
 Doctoral Thesis, Loughborough University, 2001
- Bli01 Blizzard Inc.
 "Diablo II Game Guide"
 <http://www.blizzard.com/diablo2>, 2001
- Bli02 Blizzard Inc.
 "WarCraft III Game Guide"
 <https://www.worldofWarCraft.com/info>, 2002
- BorS99 M. Borella and M. S
 "Source Models of Network Game Traffic"
 Computer Communications, vol. 23(4), 2000, pp. 403-410
- Bra89 R. Braden
 "Requirements for Internet Hosts: Communication Layers"
 Request for Comments, RFC1122,
 Network Working Group, 1989
- BraCS97 R. Braden, D. Clark and S. Shenker,
 "Integrated Services in the Internet Architecture: an Overview"
 Request for Comments, RFC1633,
 Network Working Group, 1997

- BraZBH97 R. Braden, L. Zhang, S. Berson and S. Herzog,
 "Resource ReSerVation Protocol (RSVP)"
 Request for Comments, RFC2205,
 Network Working Group, 1997
- CheB03 W. Cheswick and S. Bellov
 "Firewalls and Internet Security: Repelling the Wily Hacker"
 Addison-Wesley, 2003, ISBN-10: 020163466x
- Cla00 H. Clare
 "Internet and E-Mail: Use and Abuse"
 Institute of Personnel and Development, 2000, ISBN-10: 0852928815
- ClaM00 S. McCreary and K. Claffy,
 "Trends in Wide Area IP Traffic Patterns: A View from Ames Internet
 Exchange"
 Proceedings of 13th ITC Specialist Seminar on Measurement and
 Modeling of IP Traffic, Monterey, USA, September 2000, pp. 1--11
- Cle06 A. Clemm
 "Network Management Fundamentals"
 Cisco Systems, 2006, ISBN-10: 1587201372
- Com99 E. Comer
 "Computer Networks and Internets"
 Prentice Hall, 1999, ISBN-10: 0131434519
- Dan01 A. Daniel
 "Internet Future Strategies: How Pervasive Computing Services Will
 Change the World"
 Prentice Hall, 2001, ISBN-10: 013041803X

- DatK97 D. Kayshav
 “Effective Object-Oriented Software Construction: Concepts,
 Principles, Industrial Strategies, and Practices”
 Prentice Hall, 1997, ISBN-10: 0130867691
- DudH73 O. Duda and P. Hart
 “Pattern Classification and Scene Analysis”
 John Wiley & Sons, 1973, ISBN-10: 0471223611
- Far02 J. Farber,
 “Network Game Traffic Modelling”
 Proceedings of the 1st Workshop on Network and System Support for
 Games, Braunschweig, Germany, April 2002, pp. 53-57
- FenFW02 W. Feng, F. Chang, W. Feng, and J. Walpole,
 “Provisioning Online Games: A Traffic Analysis of a Busy Counter-
 Strike Server”
 SIGCOMM, Proceedings of the Internet Measurement Workshop,
 Marseille, France, November 2002, pp. 151-156
- FieGMFB97 R. Fielding, J. Gettys, J. Mogul, H. Frystyk and T. Berners-Lee
 “Hypertext Transfer Protocol – HTTP/1.1”
 Request for Comments, RFC2068,
 Network Working Group, 1997
- FomKMC04 M. Fomenkov, K. Keys, D. Moore and K. Claffy,
 “Longitudinal Study of Internet Traffic in 1998-2003”
 Proceedings of the Winter International Symposium on Information and
 Communication Technologies, Cancun, Mexico, Jan 2004, pp. 1-6

- ForB05 A. Forouzan and A. Behrouz
 “TCP/IP Protocol Suite”, 3rd edition.
 McGraw-Hill, 2005, ISBN: 0072967722
- GelL05 E. Gelenbe and P. Liu
 “QoS and Routing in the Cognitive Packet Network”
 Proceedings of the IEEE International Symposium on a World of
 Wireless, Mobile and Multimedia Networks, Taormina, ITALY, Jun
 2005, pp. 517–521
- Gil92 H. Gilbert
 “Network Management: Technique, Tools, and Systems Network
 Management Overview”
 Courier International Ltd, 1992, ISBN-10: 0471927813
- GupM00 P. Gupta and N. McKeown
 “Classifying Packets Using Hierarchical Intelligent Cuttings”
 IEEE Micro, vol. 20, no. 1, Jan.-Feb. 2000, pp. 34-41
- Ham75 M. Hammerton
 “Statistics for The Human Sciences”
 Longman Group Ltd, 1975, ISBN-10: 0582442737
- Hay88 L. Hays
 “Statistics”
 Holt, Richard and Winston Inc, 1988, ISBN-10: 0030744679
- Hen01 T. Henderson
 “Latency and User Behaviour on a Multiplayer Games Server”
 Proceedings of Third International Workshop on Networked Group
 Communication, London, UK, Nov 2001, pp. 1-13

- HerCRA63 Herbert, Colton, Raymond and R. Arkin
 "Tables for Statistics"
 Barnes & Noble Books, 1963
- HogMC06 V. Hogg, W. Mckean and T. Craig
 "Introduction to Mathematical Statistics", 6th Edition
 Pearson Education, 2006, ISBN-10: 0131867938
- How04 D. Howell
 "Fundamental Statistics", 5th Edition
 Thomson Learning Inc, 2004, ISBN: 0495099007
- HusHP03 A. Hussain, J. Heidemann, and C. Papadopoulos
 "A Framework for Classifying Denial of. Service Attacks"
 Proceedings of the ACM SIGCOMM Conference, Karlsruhe,
 Germany, August 2003, pp. 99-110
- Iana07 IANA
 "Port Assignments"
 Current List of Well-Known and Registered Port Assignments, 2003
 <http://www.isi.edu/in-notes/iana/assignments/port-numbers>
- Ipt06 The Netfilter Core Team
 "The Netfilter/Iptables HOWTO's"
 <http://www.netfilter.org>, 2006
- KanO76 M. Kendall and J. Ord
 "Time Series"
 Edward Arnold, 1976, ISBN-10: 0852642954

- KarKL03 M. Karol, J. Krishnan and J. J. Li
 "VoIP Protection and Performance Improvement"
 Computer Communications and Networks. Proceedings of the 12th
 International Conference, Dallas, USA, Oct 2003, pp. 505-510
- Kit96 J. Kitchens
 "Exploring Statistics: A Modern Introduction to Data Analysis and
 Inference", 2nd Edition
 Brooks/Cole Publishing Company, 1996, ISBN-10: 0534781403
- Koz05 M. Kozierok
 "The TCP/IP Guide, A Comprehensive, Illustrated Internet Protocols
 Reference"
 Bennington Vermont, 2005, ISBN-10: 1-59327-047-X
 <http://www.tcpipguide.com/>
- Kum05 S. Karnouskos
 "Dealing with Denial-of-Service Attacks in Agent-Enabled Active and
 Programmable Infrastructures"
 Proceedings of the 25th Annual International Computer Software and
 Applications Conference (COMPSAC'01), Chicago, USA, Oct 2001,
 pp. 445-450
- KurR03 F. Kurose and W. Rose
 "Computer Networking"
 Pearson Education, 2003, ISBN-10: 0321497708
- MacF02 C. Macian and R. Finthammer
 "An Evaluation of the Key Design Criteria to Achieve High Update
 Rates in Packet Classifiers"
 IEEE Network, Vol. 15.6, Nov. 2001, pp. 24-29

- Mar94 J. Martin
"TCP/IP Networking: Architecture, Administration and Programming"
Prentice Hall, 1994, ISBN-10: 0136422322
- Mic00 Microsoft Ltd.
"Crimson-Sky Versions"
<http://www.microsoft.com/games/crimsonskies>, 2000
- Mys06 Mysql AB
"The MySQL Knowledge Base"
<http://www.mysql.com/network/knowledgebase.html>, 2006
- Nad93 M. Nadler and E. Smith
"Pattern Recognition Engineering"
Wiley, 1993, ISBN-10: 0471622931
- Nag84 J. Nagle
"Congestion Control in IP/TCP Internetworks"
Request for Comments, RFC896,
Network Working Group, 1984
- NirB95 V. Nirkhe and M. Baugher
"Quality of Service Support for Networked Media Players"
Proceedings of IEEE COMPCON, San Francisco, USA, Mar 1995, pp.
234-238
- Od103 A. M. Odlyzko
"Internet Traffic Growth: Sources and Implications"
Proceedings of the SPIE, vol. 5247, Aug 2003, pp. 1-15

- OliBPD99 M.A.Oliver, K.R.Bharadia, I.W.Phillips and D.J.Parish
 “Grading and Predicting Networked Application Behaviour”
 IEE Computing and Control Journal, vol .11, Apr 2000, pp. 65-72
- ParBLPO03 J. Parish, K. Bharadia, A. Larkum, I. W. Phillips and M. Oliver
 “Using Packet Size Distributions to Identify Real-Time Networked
 Applications”
 Communications, IEE Proceedings, vol. 150, Aug 2003, pp. 221-227
- Pax94 V. Paxson
 “Growth Trends in Wide Area TCP Connections”
 IEEE Network, Vol. 8, No. 4, Jul-Aug 1994, pp. 8-17
- PlaBH99 A Plaat, H. E. Bal and R. F. Hofman
 “Sensitivity of Parallel Applications to Large Differences in
 Bandwidth and Latency in Two-Layer Interconnects”
 Proceedings on the Fifth International Symposium on High
 Performance Computer Architecture, Orlando, USA, Jan 1999, pp.
 244-253
- PilM04 T. Piliuoras and C. Mann.
 “Network Design: Management and Technical Perspectives”
 Auerbach Publishers, 2004, ISBN-10: 0849316081
- Pos82 J. Postel
 “Simple Mail Transfer Protocol”
 Request for Comments, RFC821,
 Network Working Group, 1982

- PosR83 J. Postel and J. Reynolds
 “Telnet protocol specification”
 Request for Comments, RFC854,
 Network Working Group, 1983
- PosR85 J. Postel and J. Reynolds
 “File Transfer Protocol”
 Request for Comments, RFC959,
 Network Working Group, 1985
- PosR94 J. Postel and J. Reynolds
 “Assigned Numbers”
 Request for Comments, RFC1700,
 Network Working Group, 1994
- Raj05 S. Raja
 “Why Always-On Stateful Inspection and Deep Packet Analysis are
 Essential to Deliver Non-Stop Protection”
 White Paper, Top Layer Networks, 2005
 <http://www.toplayer.com>
- Ric94 W. R. Stevens
 “TCP/IP Illustrated”
 Addison-Wesley, 1994, ISBN-10: 0201633469
- Real00 Real Networks
 “RealPlayer G2 User Information”, 2000
 <http://www.real.com>
- SanPL05 M. Sandford, D. Parish and B. Li
 “Using Flow Statistics to Identify Network Applications”
 White Paper, Loughborough University, 2005

- Sch88 W. Scheffler
 "Statistics: Concepts and Applications"
 The Benjamin/Cummings Publishing Company, 1988, ISBN 0-8053-
 8780-3
- Sch01 H. Schildt
 "Java 2: The Complete Reference", Fifth Edition
 McGraw-Hill, 2001, ISBN-10: 8441518653
- SchML03 D. V. Schuehler, J. Moscola and J. Lockwood, □
 "Architecture for a Hardware Based, TCP/IP Content Scanning
 System"
 High Performance Interconnects, Proceedings of the 11th Symposium,
 Stanford, USA, Aug. 2003, □pp. 89-94
- She03 N. Sheldon
 "The Effect of Latency on User Performance in WarCraft III"
 Proceedings of the 2nd Workshop on Network and System Support for
 Games (NetGames 2003), Redwood, USA, May 2003, pp. 3-14
- Ssh05 SSH Communication Security
 "Technical Solution Description"
 <http://www.ssh.com>
- SteCWA99 S. Savage, N. Cardwell, D. Wetherall and T. Anderson
 "TCP Congestion Control with a Misbehaving Receiver"
 ACM Computer Communication Review, vol. 29(5), Oct 1999, pp. 71-
 78
- Sun05 Sun Microsystem, Ltd
 "Java Overview White Paper"
 <http://java.sun.com> , 2005

- Sun98 Sun Microsystem, Ltd
 “Java Language Overview White Paper”
 <http://java.sun.com> , 1998
- SunM06 Sun Microsystem, Ltd
 “Java™ 2 Platform Standard Edition 5.0 API Specification”
 <http://java.sun.com/j2se/1.5.0/docs/api/> , 2006
- Tcw06 Tcpdump Workers
 “Tcpdump Howto”
 <http://www.tcpdump.org> , 2006
- TenSSW97 D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall,
 and G. J. Minden
 “A Survey of Active Network Research”
 IEEE Communications Magazine, vol.35, Jan 1997, pp. 80-86
- Win04 Windump Team
 “WinDump Manual”
 <http://www.winpcap.org/windump>, 2004
- WanC91 Z. Wang and J. Crowcroft
 “A New Congestion Control Scheme: Slow Start and Search”
 ACM Computer Communication Review, vol. 21, Jan 1991, pp. 32-43
- WanYFZF00 P.Y.Wang, Y.Yemini, D.Florissi, J.Zinky and P.Florissi
 “Experimental QoS Performances of Multimedia Applications”
 INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE
 Computer and Communications Societies. Proceedings. IEEE, vol.2,
 Tel Aviv, Israel, Mar 2000, pp. 970-979

Xcwg05 XML Core Working Group
 “Extensible Markup Language (XML)”
 <http://www.w3.org/XML>, 2005

