uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Vicky Laurens

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.Sc. (Systems Science)

GRADE / DEGREE

Department of Systems Science

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

DDoSniffer : An Attack Detection Tool Detecting TCP-Based Distributed Denial of Service Attack Traffic at the Agent Machines

TITRE DE LA THÈSE / TITLE OF THESIS

Professor A. El Saddik

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Professor Dominique Ferrand

Professor Amiya Nayak

Gary W. Slater

Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

# DDoSniffer:
# An Attack Detection Tool
# Detecting TCP-Based Distributed Denial of Service Attack Traffic
# at the Agent Machines

**Vicky Laurens**

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the MSc degree in Systems Science

Systems Science
Faculty of Graduate and Postdoctoral Studies
University of Ottawa

*Your file Votre référence*
*ISBN: 978-0-494-25797-5*
*Our file Notre référence*
*ISBN: 978-0-494-25797-5*

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# LIST OF ABBREVIATIONS

| | |
|---|---|
| **ACK** | Acknowledgement flag in the TCP header |
| **AS** | Autonomous System |
| **BOT** | Robot |
| **BOTNET** | Bot network |
| **CBOS** | Cisco Broadband Operating System |
| **CRv1** | Code Red version 1 |
| **CRv2** | Code Red version 2 |
| **DNS** | Domain Name System |
| **DoS** | Denial of Service |
| **DDoS** | Distributed Denial of Service |
| **DSL** | Digital Subscriber Line |
| **D-WARD** | DDoS Network Attack Recognition and Defense |
| **FDS** | Flooding Detection System |
| **FIN** | Finish flag in the TCP header |
| **FTP** | File Transfer Protocol |
| **ICMP** | Internet Control Message Protocol |
| **IP** | Internet Protocol |
| **IPSE** | IP Service Engine |
| **IRC** | Internet Relay Chat |
| **ISP** | Internet Service Provider |
| **HTTP** | Hypertext Transfer Protocol |
| **MULTOPS** | Multi-Level Tree for Online Packet Statistics |
| **NMAP** | Network Mapper |
| **RIAA** | Recording Industry Association of America |
| **RST** | Reset flag in the TCP header |
| **SYN** | Synchronize sequence numbers flag in the TCP header |
| **TCP** | Transmission Control Protocol |
| **TFN** | Tribe Flood Network |
| **UDP** | User Datagram Protocol |
| **URL** | Uniform Resource Locator |

# ABSTRACT

Distributed Denial of Service (DDoS) attacks are an important and challenging security threat. Despite of the availability of several defence mechanisms and ongoing academic research in the field, attackers handle to build a large network of agent machines. This research developed a tool, DDoSniffer, to tackle the DDoS attack by detecting ongoing attack traffic at the agent machines. Due to the diversity in DDoS attack strategies, it is not realistic to deal with all type of attacks with one single solution. DDoSniffer focuses on TCP-based attacks. Different scenarios were tested to evaluate the performance of DDoSniffer when detecting what we classified as connection attacks and bandwidth attacks. The former attacks generate connections with four packets or fewer. The latter attacks create connections with traffic ratios larger than usual. Detection is the minimum requirement of all defence mechanisms, and DDoSniffer is capable of detecting a broad range of attacks within seconds.

# ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Abdulmotaleb El Saddik for his guidance, help, and support throughout the course of my studies.

My sincere thank goes to Brian Stacey, Alok Patnaik and Pulak Dhar for the innumerable discussions of ideas as well as their clever suggestions.

A very special acknowledgement goes out to my family who has always encouraged and supported me in pursuing all my endeavours.

I also want to thank all my friends for their love, patience and support. I am deeply grateful to my friends Abdel-Aziz El-Solh and Andrew Soon; I would never have made without them.

# PUBLICATIONS AND PRESENTATIONS

V. Laurens, and A. El Saddik, "DDoSniffer: An Attack Detection Tool," presented at Cistech Limited, Ottawa, June 2006.

V. Laurens, A. El Saddik, P. Dhar, and V. Srivastava, "Detecting Distributed Denial of Service Attack Traffic at The Agent Machines", In Proceedings of the 2006 Canadian Conference on Electrical and Computer Engineering, Ottawa, ON, Canada, May 7-10, 2006

V. Laurens, and A. El Saddik, "Detecting Distributed Denial of Service attack traffic at the agent machines," Cistech Technical Report, TR-CL-DDOS-3, 2005.

V. Laurens, and A. El Saddik, "DDoS: Detecting Attack Packets at the Agent Machines," presented at Cistech Limited, Ottawa, June 2005.

# CHAPTER 1
# INTRODUCTION

Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks are aimed to thwart legitimate users from accessing to shared resources. In general, DoS/DDoS attacks prevent legitimate clients from accessing Internet services provided by web servers, FTP servers, mail servers, DNS servers, and so forth. For instance, when an attacker overloads a server with several requests, the server consumes its resources and denies its services to legitimate users. A DDoS attack differs from a DoS attack in that several machines are impersonated and used by the attacker to send a large number of requests to the targeted server (without the legitimate machines' users knowing that their computers are being employed to mount an attack).

Even though DoS/DDoS attacks have been known for a long time, the general public has just recently become aware of them not only due to the highly publicised DDoS attack to the Internet root servers (the Domain Name Server (DNS)) in October 2002 [1], but also due to DDoS attacks to Google, Yahoo, SCO, Microsoft, Lycos, and AltaVista [2]. Furthermore, in the 2004 CSI/FBI annual Computer Crime and Security Survey, responders ranked DoS type of attacks as being the second most damaging threat with financial losses of more than US$26 million [3]. Another security report, Symantec Internet Security Threat Report [4], identified a peak rate of 9,163 bot hosts recruited per day as well as an average number of 1,402 DDoS attacks per day. A bot (short for robot) host is an infected computer that is part of a botnet (bot network) and follows commands coming from the botnet controller [5].

It has been said that early DDoS attacks were seen as an Internet Relay Chat (IRC) problem [3]. However, history of successful DDoS attacks against sites such as Yahoo, Google, eBay, CNN, and Microsoft has shown that DDoS attacks are a serious threat beyond the IRC environment. IRC is an instant communication system over the Internet. Financial losses have been the main motivation for deploying defence mechanisms near the target

server; however, despite the availability of several commercial solutions and ongoing academic research work, attackers still succeed.

## 1.1 Motivations

One of the several reasons for this success is the evolution of attack tools. Presently, even people with little or no technical knowledge can use them to perpetrate DDoS attacks. This trend of tool simplicity was noted as early as in 1999 [6]. Another contributing factor is the increasing number of home users with high speed Internet access such as DSL and cable services. Currently, computers belonging to home users and universities are the main targets of the recruitment phase of DDoS attacks [5]. For instance, in April 2004, 400 computers at the University of North Texas in Denton were infected with Agobot [7]. Also in the summer of 2003, several universities reported compromised hosts [8]. Personal firewalls are one of the best solutions available to protect end-users' machines; however, 67 percent of people do not use firewalls [9]. Another Symantec report [10] indicated that during a six month period, 1,403 new vulnerabilities were found. Ninety-seven percent of these vulnerabilities can result in complete or partial compromise of a system and 70 percent of the vulnerabilities were ranked as easy to exploit [10]. The high percentage of "unfirewalled" PCs along with the vast number of security vulnerabilities provide attackers a field of hosts to form their large agent armies. An agent computer is a compromised machine which is controlled by the attacker.

As the most damage of a DDoS attack is suffered by the main victim (the targeted server), the general public might believe DDoS threats are not of their concern. Nevertheless, once a machine has been impersonated, the attacker can have access to user's private information such as passwords and financial data. Recent DDoS trends include blended threats which could be referred as "all in one": recruitment, spreading, and attack payload in one single program tool. Nine-hundred variants of Agobot, a powerful blended threat, have been identified [11]; additionally, 6,542 variants of Spybot were reported [4]. Another common trend is the use of bots programs; attackers employ a bot network which allows remote control of impersonated hosts through the use of communication channels such as IRC.

The remote control capabilities also allow attackers to collect information from the impersonated hosts. Actually, Symantec's report indicated that 54 percent of threats are capable of exposing confidential information and these capabilities are attributed to the use of bot networks and their remote control power [10]. Figure 1-1 depicts the capabilities of the tools identified as the top threats in 2005 by Sophos [12]. Due to this new trend of combining different attack techniques into a single tool, classification of attack tools has become difficult. For example, the attack tool Powerbot has been classified as both an Internet worm and a bot. An Internet worm is a self-replicating program similar to a virus, but usually a virus is part of another program whereas a worm does not need another application to propagate itself.

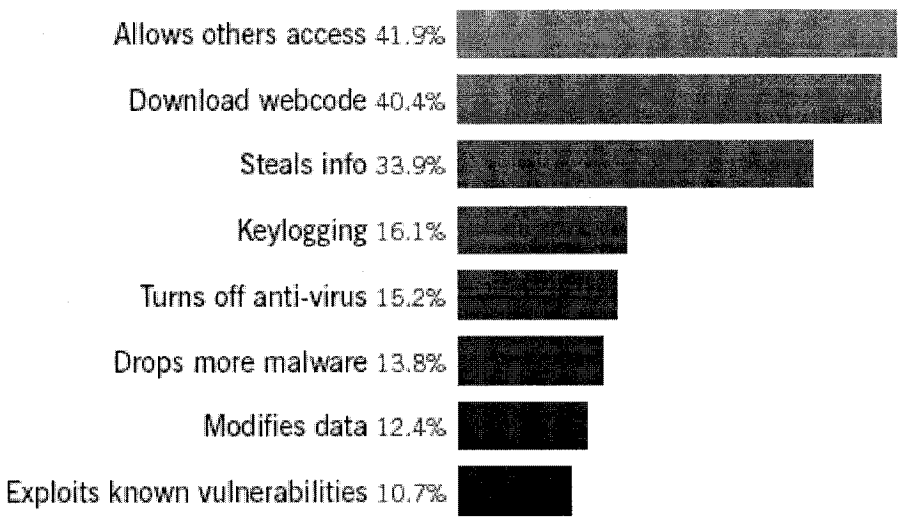


**Figure 1-1: Top threats characteristics [12]**

## *1.2 Problem statement*

Due to the fast development of DDoS techniques as well as the increasing number of new vulnerabilities (which make it almost impossible to keep up to date), there is a need to deploy a tool which ideally does not require updates in order to detect ongoing attacks. DDoS threats are a complex problem, and it is not realistic to expect one single solution to undertake the entire problem of DDoS on its own. Detection of attack traffic is the minimum requirement of all defence mechanisms. Our approach to tackle the DDoS attacks is to develop a software tool for detecting ongoing attack traffic at the agent machines. Moreover,

as most Internet traffic is two-way communications (about 90% of Internet packets are TCP [13]), this research focuses on TCP-based DDoS attacks.

This research's approach to tackle the DDoS attacks is to develop a software tool for detecting TCP-based DDoS attack traffic at the agent machines.

## 1.3 Research contributions

In this thesis, a software tool (DDoSniffer) was implemented to detect attack traffic at the agent machines. This thesis' contributions to the DDoS defence field are:

- DDoSniffer is the first detection tool for measuring traffic at each agent machine.
- DDoSniffer's performance results demonstrate that it is feasible to detect a broad range of TCP-based DDoS attacks at agent machines participating in an attack. Moreover, average detection time was 7.36 seconds, and the lowest detection rate was 73 percent. The detection rate indicates the percentage of attacks that were in fact detected.
- DDoS attacks, architectures and methodologies were deeply studied in order to narrow down the roots of the success of attackers: agent machine participation.
- TCP-based DDoS attacks were analyzed in detail. This analysis provided the decomposition of attack methodologies into two categories: connection attacks and bandwidth attacks. The former attacks create connections with four packets or fewer. The latter attacks generate connections with traffic ratios larger than usual. The traffic ratio of a connection is defined by the number of outgoing packets to the number of incoming packets.

## 1.4 Thesis outline

This thesis is organized into eight chapters. Chapter 2 covers DDoS attack trends, attack networks, and attack victims. Existing source-end defence mechanisms and previously conducted research in this area are discussed in Chapter 3. In Chapter 4, TCP-based DDoS attack methodologies are described. Chapter 5 explains DDoSniffer's design and functionalities. The experimental setup used for experiments is presented in Chapter 6.

Chapter 7 describes the performance results. In Chapter 8, the conclusions and future work are discussed.

# CHAPTER 2
# DISTRIBUTED DENIAL OF SERVICE ATTACKS

Internet distributed and non-distributed denial-of-service (DoS) attacks are launched with the intent to incapacitate one or more servers, rendering the targeted server(s) offline and inaccessible to legitimate users. For example, a Web Server experiencing a DoS type of attack would not be able to provide its services to users; in the most severe case, no user would obtain access to the services normally provided by the attacked server. A distributed DoS (DDoS) attack is a coordinated DoS attack originating from several different machines, usually called zombies or agents (other names include daemons, drones, slaves and bots; in this thesis, the name agent is adopted). Sometimes both terms DoS and DDoS are used interchangeably; however, regardless of their commonalities, DDoS attacks have evolved up to a point that referring to them as DoS attacks might be misleading. DDoS attacks succeed where their DoS counterpart fail. For example, client puzzle protocols could thwart DoS attacks but their effectiveness against DDoS is doubtful [14]. This research is concerned with DDoS threats, and DoS references will only be used for comparison to facilitate understanding.

## *2.1 DDoS Attack Strategies*

DDoS attacks are typically accomplished by overwhelming the targeted server with bogus service requests such that the server engages its resources in processing these false requests and most likely denying its services to legitimate users. This type of attack is known as a flooding attack. The difference between DoS and DDoS flooding attacks is that in the former, the attack packets come from a single source whereas in the latter, the attack packets come from multiple sources. This is why DDoS attacks are also referred to as multiple

source DoS attacks. The key factor of a flooding attack is to send enough packets in order to consume the victim's resources (CPU cycles, network bandwidth, memory, and so forth). A typical example of DDoS flooding attacks is the TCP SYN Flooding attack which is usually based on breaking up the three-way handshake that occurs when a TCP connection is opened.

The establishment of a TCP connection begins with a three-way handshake:

1. The client issues a SYN message to the server whose services it requires;
2. The server replies with a SYN-ACK message acknowledging that a SYN message was received;
3. The client then completes the TCP connection by sending an acknowledgement message.

The TCP connection is established and the data communication exchange starts after the third message is issued. When the server sends its SYN-ACK message, a slot on the server's memory is also reserved for the TCP connection being requested. An attacker, wishing to consume the server's memory, only needs to send several concurrent SYN messages. However, no final ACK for those SYN messages would be sent. The server, as usual, would reserve some memory resources for each of the connections being requested; though, as the attacker does not complete any of the connections, the server would exhaust its memory resources [15]. In a DoS attack, these requests most likely come from the attacker's machine (using a spoofed IP address); on the other hand, in a DDoS attack, the bogus requests come from the impersonated clients (using their real IP addresses).

When no spoofing is used, attackers might hack the TCP/IP stack of the impersonated computer such that final ACKs are not sent. However, if a sufficiently large number of requests is sent in a short amount of time, there is no need to modify the TCP/IP stack because the server will be overwhelmed and will not be able to send out SYN-ACKs. Figure 2-1 illustrates the TCP three-way handshake as well as the broken three-way handshake during a SYN Flooding attack (when IP spoofing is employed). Other flooding attacks include ICMP echo request flood, ICMP echo reply flood, TCP data segment flood, TCP RST flood, TCP FIN flood, TCP ACK flood, and UDP flood [16].

Another method used to perpetrate DDoS attacks is based on exploiting one or more vulnerabilities existing in the targeted system. This DDoS attack is called a vulnerability

attack. Frequently the vulnerability is a bug in an application program, a protocol, or an operating system. By sending an incomplete packet, for instance, the application could reboot, crash, or degrade in performance. An example of a DDoS vulnerability attack is in the case of the Cisco 600 family of routers. A router could be locked by sending a specific URL if the router was configured to allow Web Access (which is frequently done to ease remote configuration) [17]. Only a couple of weeks after this vulnerability was published, Cisco released another Security Advisory about the same routers' family. In this case, four different vulnerabilities in the Cisco Broadband Operating System (CBOS) allowed attackers to make these routers unresponsive, and if a router is made unresponsive, the router must be rebooted to recover normal operations [18].

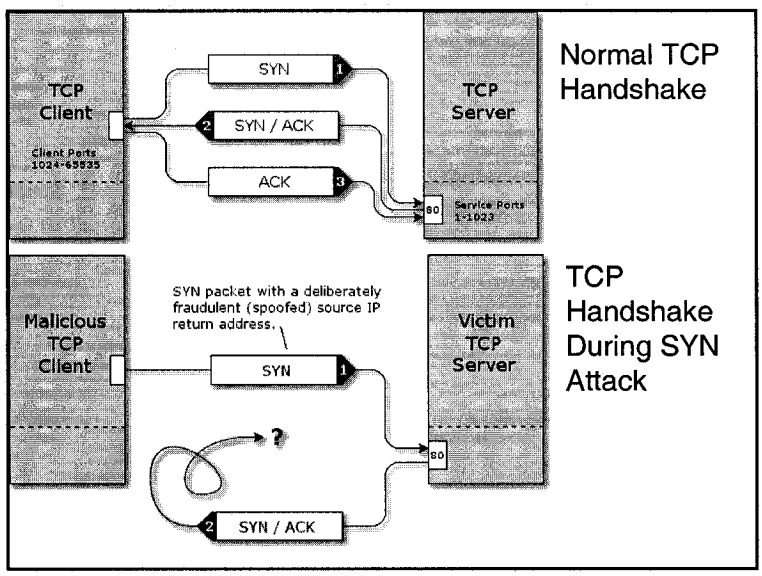


**Figure 2-1: TCP three-way handshake [19]**

The main advantage of vulnerability attacks is that only a few packets are needed to cause serious damages. Nevertheless, the two types of attacks are in fact similar, and thus, an attack can be classified as belonging to both categories. For instance, Cisco classified an attack as a vulnerability attack because the rate of SYN packets was low (one packet per second) [17]. Regardless of the type of DDoS attack, intruders usually recruit and control multiple computers in order to form an army of attack agent machines. A DDoS attack thus comprises two main phases: the recruitment phase (where agent machines are compromised), and the attack itself to the main victim (where services are denied to legitimate users). For an

attacker, success is achieved by not only denying services to legitimate clients, but also by not being caught.

A key factor contributing to achieve the attacker's goals is the ease of the pre-attack phase or recruitment phase. In early DDoS attacks, recruitment was done manually; current attack trends include the use of Internet worms, bot programs, and blended threats in order to scan for vulnerable machines, compromise them, and even install the attack tool (if applicable). Two examples of Internet worms used for DDoS purposes are the Code-Red versions 1 and 2, CRv1 and CRv2 respectively. These worms were programmed to disseminate themselves during days 1 to 19 of every month and to launch a DDoS attack on www1.whitehouse.gov during days 20 to 27 [20]. CRv2 infected more than 359,000 machines in only fourteen hours and this was accomplished by introducing a small change in the code of CRv1.

Another infamous Internet worm employed for DDoS attacks is MyDoom, also known as Doomjuice and Zindos. Its variants do not differ much between one another and they were used to attack SCO, Microsoft, Altavista, Lycos, Google, Yahoo, and Recording Industry Association of America (RIAA). The second version of MyDoom blocked access to more than 60 computer security companies' websites [21], making detection and removal of the worm very difficult for users with little technical expertise. Due to the fact that the name of the worm varies among security companies, it is very difficult to track down the exact number of versions of MyDoom that have been identified, but only Microsoft's support site enumerates 11 distinct versions [22]. Moreover, MyDoom spread so quickly that its first version contributed to 30 percent of all global e-mail traffic [23]. It is important to note that attack tool classification is becoming ambiguous because as tools are combining diverse techniques, a given tool can be classified as belonging to more than one category. For instance, Powerbot was classified as a worm by the Cert Coordination Center but as a bot for David Dittrich in his analysis [24].

## 2.2 DDoS Attack Architectures

Another important aspect of DDoS attacks' success is the DDoS networks. To mount a successful multiple source DoS attack, the source agents must somehow first be compromised. The pre-attack or recruitment phase is often composed of four steps: a)

scanning for vulnerabilities, b) compromising vulnerable machines, c) installing an attack tool, and d) employing compromised machines to keep scanning and compromising other hosts [25]. It should be noted that the vulnerabilities used to compromise and gain control of agent machines are not necessarily the same vulnerabilities used in the attack phase for shutting down the main victim. A significant aspect to point out is that not all attacks require an attack tool on the source vulnerable machines (agents); instead, the vulnerability is exploited during the attack itself. Sometimes, attackers prefer this method in order to avoid early detection through discovery of the attack tool. However, exploiting the vulnerability in the pre-attack phase guarantees the attacker control of compromised machines, especially if the vulnerability is patched by the attacker after it has been exploited. This is done not only to avoid further exploitations by other attackers, but also to retain control of the impersonated machine. If attacker fixes the vulnerability, the attacker installs a backdoor which is a program that opens a port that listens for incoming connections (usually commands from the attackers). If the vulnerability is left to be exploited during the attack phase, there is no guarantee that the given vulnerability has not been patched (either by other attackers or by legitimate users).

Both attack phases, pre-attack steps and the attack itself are carried out in a client/server architecture. Figure 2-2 depicts such an attack client/server network consisting mainly of handler and agent machines. A handler is an impersonated machine which runs an attack tool for controlling several agent computers; handlers are also in charge of scanning and compromising new machines. Between handler machines and the attacker machine, there could be several layers of clients called stepping stones [3]. In this configuration, if a handler machine is detected, the identity of the attacker is still unknown. On the other hand, the agent machines generate packets toward the targeted server (the main victim of the attack) through the handlers' commands. Examples of DDoS attacks employing this handler/agent network are Trinoo [27], Tribe Flood Network (TFN) [28], and Stacheldraht [29].

Most early DDoS attacks employed architecture like the one depicted in Figure 2-2 where the attack traffic flows from agents to the intended victim. A reflector attack on the other hand uses a slightly different mechanism where attack traffic comes from machines (frequently legitimate servers) that have not been compromised. Reflector attacks are based on IP spoofing rather than exploiting a given vulnerability where agents send a request for

services that require an acknowledgement. A typical example of a reflector attack is a DNS query; in this type of attack, agents send DNS queries to legitimate servers (the reflectors). However, the agents' requests have the source IP address spoofed with the IP address of the targeted victim such that reply packets from the reflectors flood the victim. Figure 2-3 shows the DDoS architecture used in a reflector attack. Examples of reflector attacks are "smurf" and "fraggle" attacks [26].

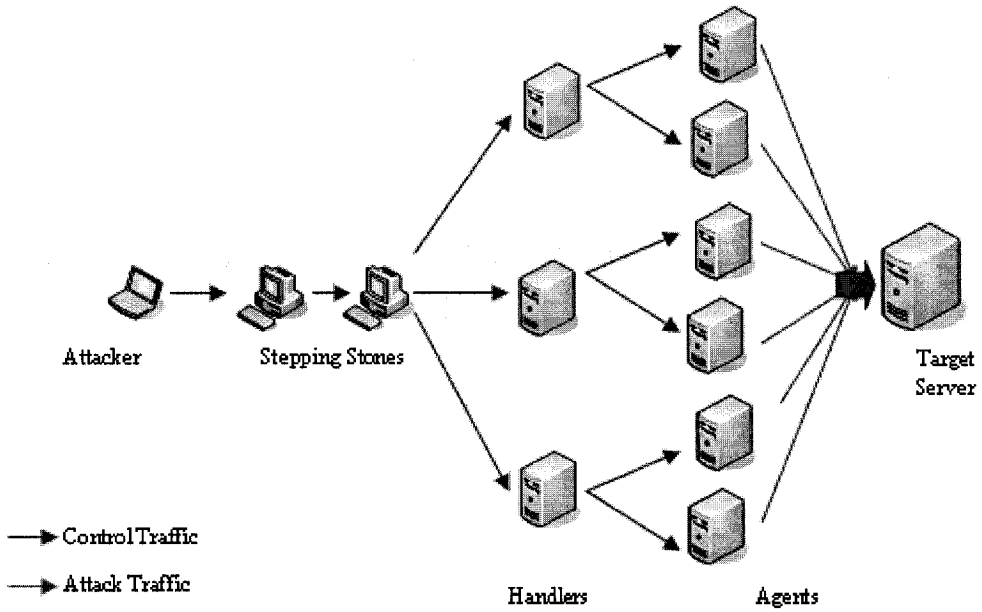


**Figure 2-2: DDoS attack architecture**

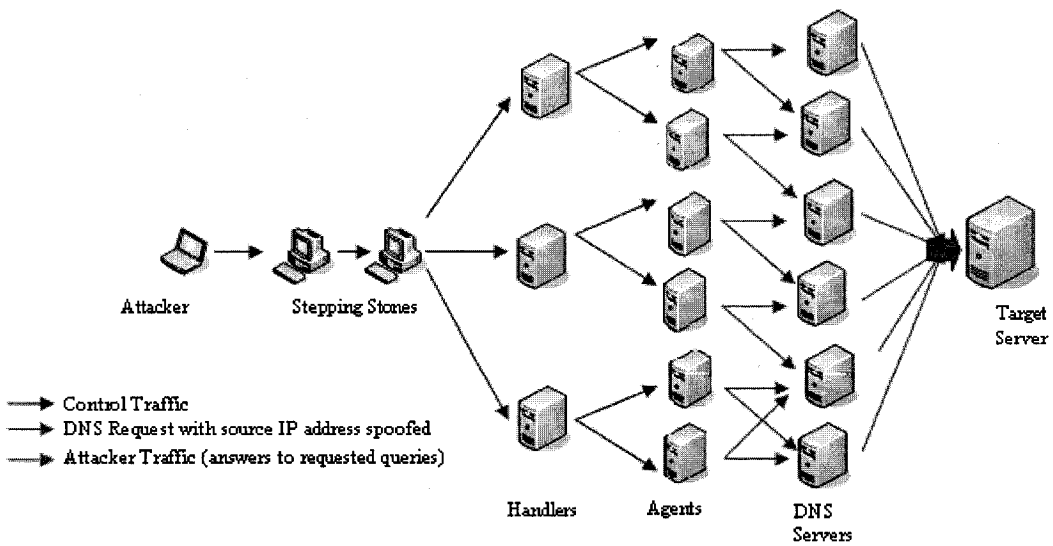


**Figure 2-3: Reflector attack architecture**

The handler/agent DDoS architectures described above have several disadvantages from the attacker's point of view. First and perhaps the most limiting factor of these types of architectures is the number of agent machines a handler host can control. In order to communicate, handlers and agents need two connections: agents listen for commands from handlers, and handlers listen for agents such that agents can register their IP addresses on a handler's list. For example, in the DDoS attack tool Trinoo, handlers listen for agents over port 31335/UDP whereas agents listen for handlers over port 27444/UDP [27]. However, the number of open connections is limited by operating systems and this is why some DDoS tools are able to control only 1024 agents per handler [3]. Second, attackers need to develop and install an attack tool on the handler machines and a different tool on the agent machines. In other words, two different applications are needed in order to establish a command/control communication channel. Third, if an agent host is detected, there is an opportunity to trace back the attacker [30]. In some early DDoS tools, the agent kept a list of handlers and vice-versa; therefore, discovering one or the other implied most likely disruption of the entire DDoS network [31].

Attackers began replacing the handler/agent architecture with DDoS IRC-based networks to avoid detection by way of discovering attack tools, monitoring obscure open ports (opened by the attack tools) and/or tracing back (or disrupting the whole attack network). This tendency was noticed as early as 2001 [31]. Generally speaking, the role of handler machines, which basically provide a communication channel between the attacker and the agents, is now provided by channels on IRC servers. Figure 2-4 illustrates a DDoS IRC-based network which is frequently referred to as a bot network (or botnet for short) and thus, agents are called bots. Attack tools are also referred to as bots in this scheme. The agent machines and the attacker connect to one or more IRC servers and communication is done by using a legitimate service (IRC) and a standard port number. There is no need to deploy a communication protocol since IRC protocols are used. Due to the popularity of IRC among end-users, DDoS IRC-based networks are difficult to identify; additionally, if the employed IRC network is public, then removing IRC servers to which agents are connected is not practically feasible [31].

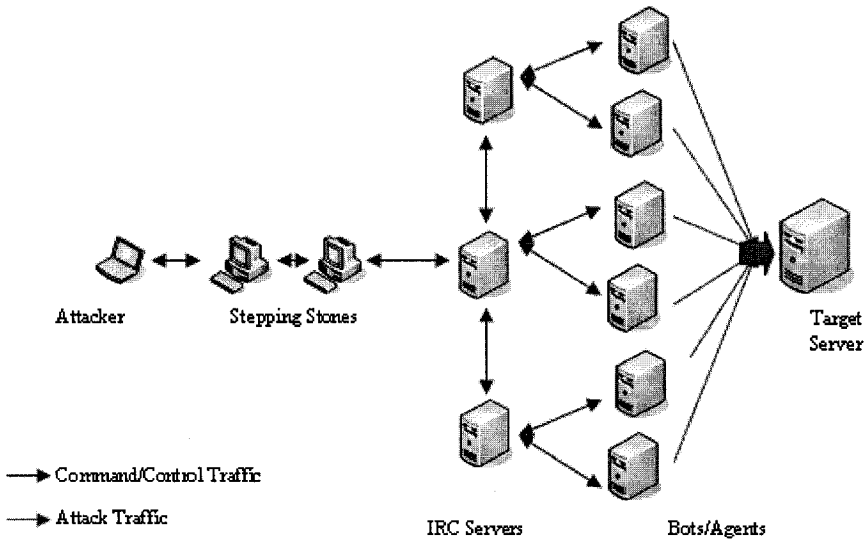


**Figure 2-4: IRC-based DDoS architecture**

Bot programs are very powerful and most of them provide downloading functions through FTP or HTTP. Once downloaded, these files can be executed immediately or at a later time [5]. Thus, attackers can update their malicious code or install new malicious programs at any given moment. Botnets are employed for more than mounting DDoS attacks; among other examples, bots can log the victim's keystrokes in order to collect sensitive information such as usernames, passwords and financial data [32]. Mytob is a bot program which records keystrokes, steals passwords and downloads files [4]. Other usages of bot networks include spamming, traffic sniffing, identity theft, and spreading new bots [33].

An important aspect of bot development is the modular fashion in which the bots are programmed. This modularity allows an attacker to extend the malicious code to exploit new vulnerabilities by adding a new module to the existing malware. The modular design might be the major reason for a decreasing number of new bot families but an increasing number of variants of existing families. For instance, Symantec reported 6,542 new variants of Spybot whereas the number of new families decreased by 39 percent in the last six months of 2005 [4]. Moreover, five to ten new variants per day of Agobot were seen in May of 2004 [34]. Other examples of bot programs are Phatbot, Netbot, and Gbot.

## 2.3 Keys to a Successful DDoS Attack

The history of denial of service attacks (both distributed and non-distributed) shows that attackers seem to be always one step ahead of most defence mechanisms. This is mainly because there are many ways (known and unknown) to mount a distributed denial of service attack (as described above) and attackers usually only need to introduce small changes in their strategies in order to defeat defence mechanisms. For instance, defence mechanisms can be defeated based on:

- Attack signatures - Attackers use encrypted or IRC channels for communications between agents and handlers such that attack signatures are not easily recognized [31], [30]. Figure 2-5 shows an example of an IRC console.
- Anomaly detection - Attackers usually alternate the group of agents participating in the attack such that traffic is not always coming from the same group of IP addresses [35], which can be detected as suspicious activity.

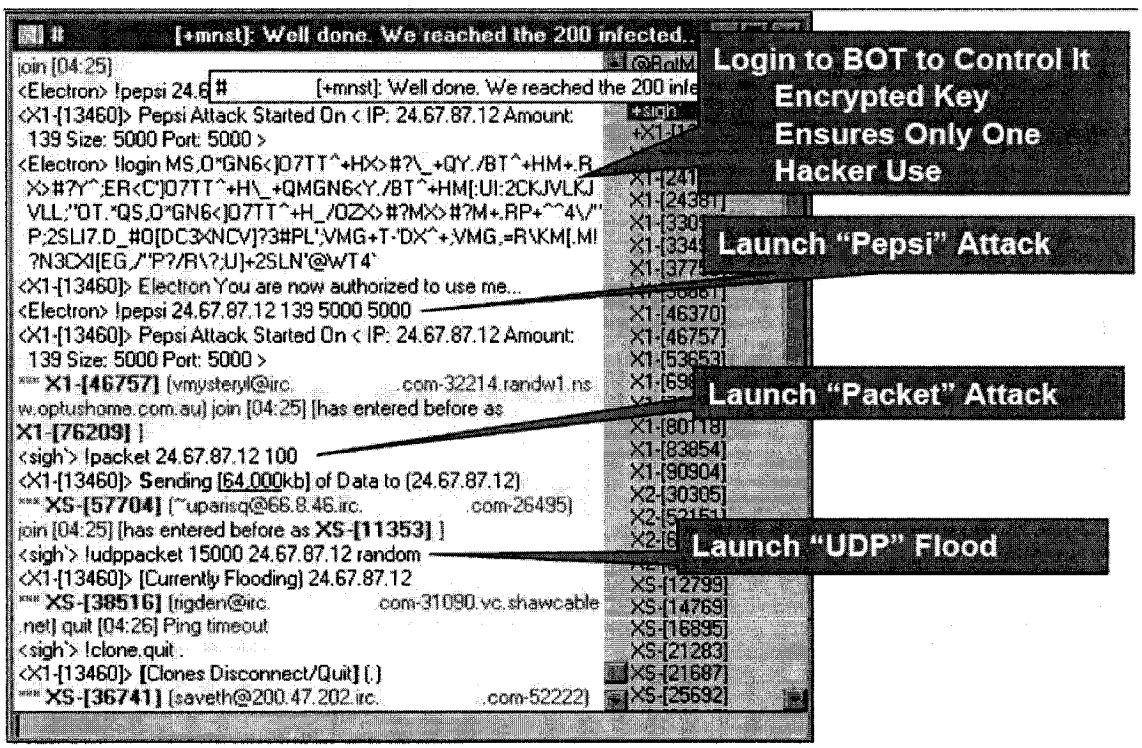


**Figure 2-5: DDoS master console [30]**

Another example of a key to a successful attack is related to raw socket access. In order to mount a TCP connection depletion attack, attackers often use raw sockets to avoid depleting their own resources. Raw sockets were eliminated with the release of Service Pack 2 for Windows XP and only a few days later, a version of Nmap was able to build raw IP packets [36]. The list of defence measures and attack counter measures is as vast as the number of current DDoS solutions and academic research. Many distinct DDoS attacks exist, and the only common factor present in all of them is the agent army that is recruited before the attack actually starts.

Attacker success is greatly dependent on the ease of the recruitment phase or pre-attack phase. Several factors contribute to this success:

1. The availability of a large number of known and unknown vulnerabilities. Symantec reported the number of discovered vulnerabilities has been increasing for the past three consecutively reported periods. A total of 1,416 new vulnerabilities were discovered between July 1 and December 31 in 2004, 1,871 vulnerabilities in the first six months of 2005, and 1,896 vulnerabilities in the last six months of 2005 (79 percent of which were categorised as easily exploitable) [4]. In addition, Sophos reported 15,907 new malware threats in 2005 [12].
2. The increasing number of barely protected computers with high speed Internet access. Sixty-seven percent of people do not use firewalls [9]. For example, the U.K.'s ranking as the country with the highest number of bot infections was attributed to the fast growth of broadband access [10]. Another important source of poorly secured computers is educational institutions (.edu address space) which usually provide fast network connections and large storage space [5]. An unprotected computer has a 40 percent risk of being compromised within the first ten minutes and the risk of infection grows to 94 percent after the first hour [12].
3. The turnaround time between vulnerability discovery and patch release is typically long. For example, in the second half of 2005, there was a 42-day window between vulnerability exploitation and patch dissemination [4]. Additionally, patching is not of much help when the machine is already

compromised since cleaning up a bot infection is a complex and long process even for experts. Frequently, procedures include booting the compromised computer in safe mode and editing the registry, which are unknown matters for general users with little technical expertise (see [30] for De-Botting procedures). For the case of worms, some cleanup procedures require a reinstallation of the operating system [8]. Patching is also ineffective if attackers patch the vulnerability or vulnerabilities after intrusion to prevent other attackers from taking over the machine. Netsky, a worm written in 2004, went beyond preventing others from taking over the computer. Netsky removed previous infections of other worms/viruses as part of its installation process [22].

4. Detecting infections is also a contributing factor since several bot programs block access to important security web sites as well as disable anti-virus software [37], [12]. Detection of bot infection is not a straightforward task also due to the fact that bot programs, unlike worms, do not consume large amounts of resources nor do they reboot systems. Therefore, for the general public, infection detection is quite difficult. An example of user obliviousness is in the case of the Sasser Worm which caused notorious damage to the end users. It was only after detection of the Sasser Worm that other malicious codes were also found on the compromised machines [7].
5. The lifespan of vulnerabilities could be unlimited; regardless of patch availability, old vulnerabilities are still being exploited, which extend the persistence of attack tools over time. For example, it was reported that Symantec accounted for 20,000 instances of the Code Red worm in 2004 [34]. This was more than three years after the second version of this worm infected 359,104 hosts in less than 14 hours [20].
6. The evolution and availability of attack tools is an extremely important factor in the prevalence of DDoS attacks. Not only are attack tools freely available online, but they are also typically easy to use. Figure 2-6 shows the evolution of attack tools over time. It can be seen that intruder knowledge has been less demanding compared to attack sophistication which has been more complex.

Attackers are not necessarily the developers of the tools and this is probably why DDoS attacks are so prevalent. Figure 2-7 charts the daily occurrences of DDoS attacks over a two-year period beginning January 2004.

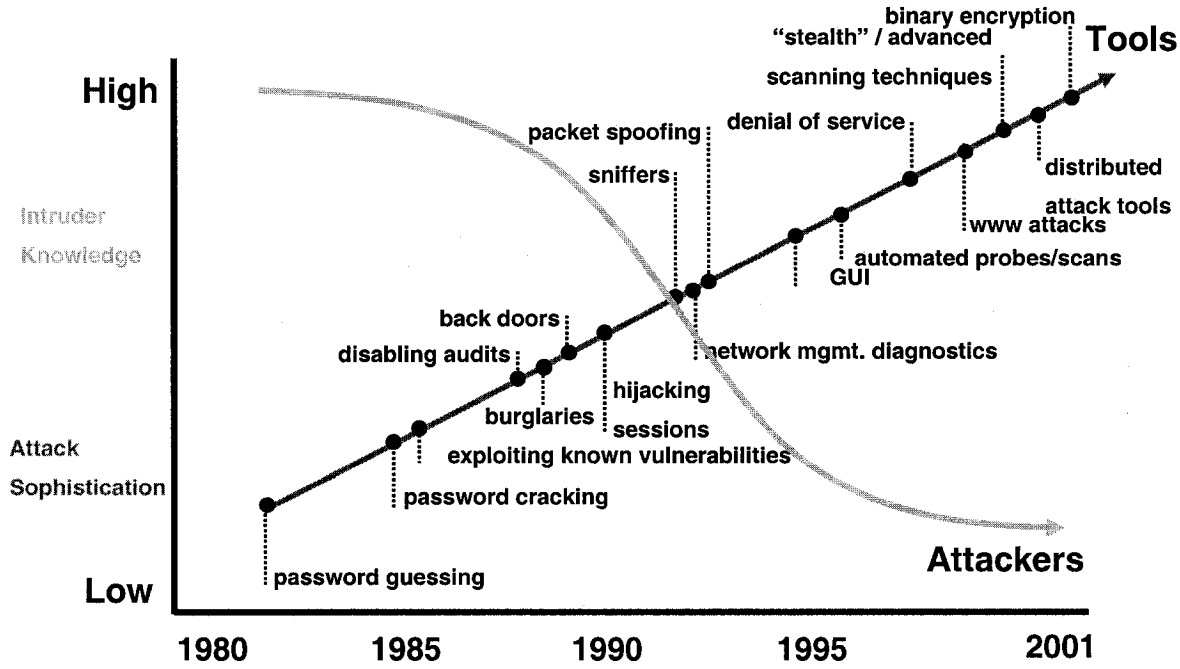


**Figure 2-6: Attack tool evolution over time [38]**

The lack of an easy-to-use tool for detecting ongoing attacks at the agent machines also plays an important contributing role in the success of the pre-attack phase. Due to the fast development of DDoS techniques as well as the increasing number of new vulnerabilities (which makes it almost impossible to be up to date), there is a need to deploy a tool which ideally does not require updates in order to detect ongoing attacks. Our approach to tackle the DDoS attacks is to develop a software tool for detecting ongoing attack traffic at the agent machines.

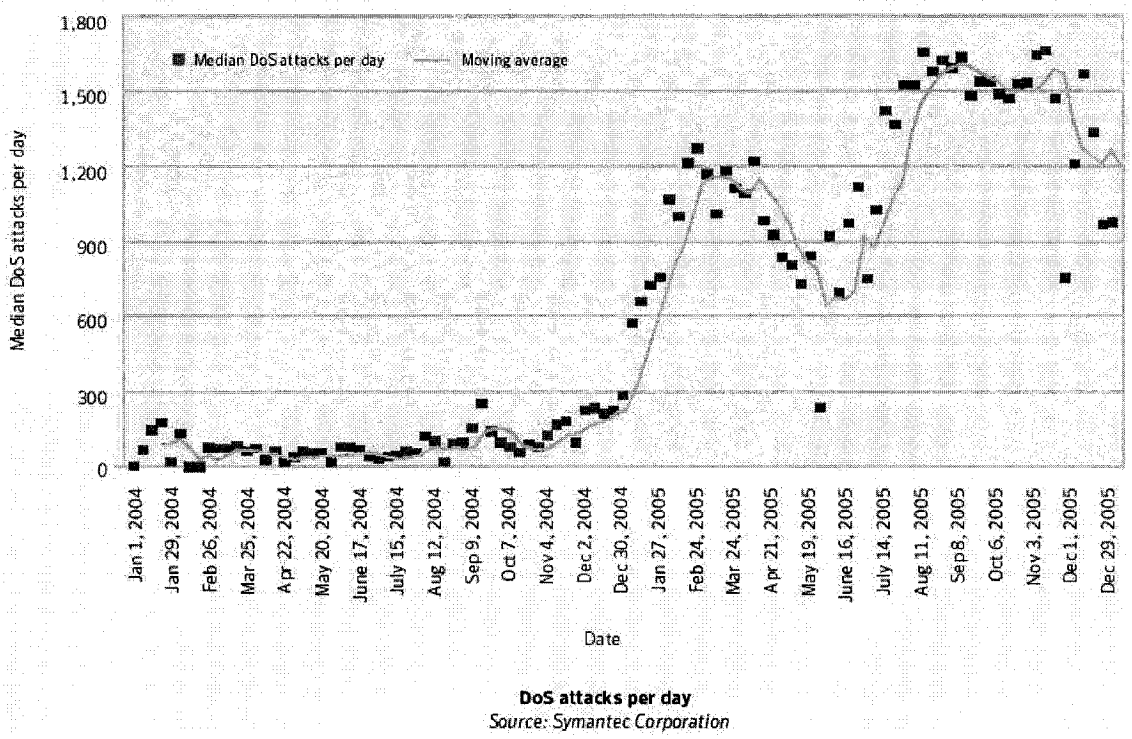


**Figure 2-7: DDoS attacks per day [4]**

# CHAPTER 3

# DDoS DEFENCE MECHANISMS

DDoS defence mechanisms can be classified in several ways. The two most commonly used classifications are based on: a) the defence mechanism's goal, that is, preventive versus reactive; and b) the deployment location, that is, the victim network, the intermediate network, and the source network [35]. Another DDoS classification is based on the activity deployed [39]: intrusion prevention, intrusion detection, intrusion tolerance and mitigation, and intrusion response. In this chapter, we adopt the deployment location classification since our concern is solutions deployed at the source network where attack packets are generated. Figure 3-1 depicts DDoS defence mechanisms' deployment locations.

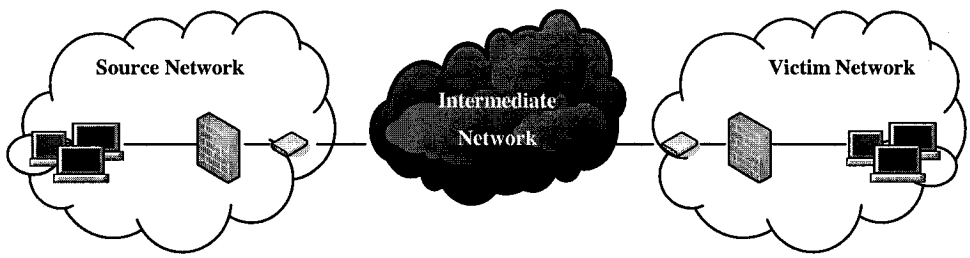


**Figure 3-1: DDoS defence mechanisms' deployment locations**

Historically, source-end defence mechanisms have lacked deployment motivations mainly because a DDoS attack inflicts the most damage on the main victim. In the past, agent machines could still essentially function normally despite performance degradation and bandwidth loss. However, with new DDoS attack trends, agent machines are also at risk because sensitive data could be gathered after the intrusion. Symantec reported that 54 percent of identified threats are able to expose confidential information [10]. Handler and

agent machines have always been second, third, and $n^{th}$ layer victims; nevertheless, DDoS defence mechanisms are mostly deployed at the main victim network. We believe this paradigm should change in order to tackle the DDoS threat at its root: thwart agent machine participation in DDoS attacks.

Another motivation for a source network DDoS defence mechanism is related to legal liability for damages, which is of more concern to Internet Service Providers (ISPs) than end users. The growing number of DDoS attacks compounded with the escalating amount of economic damage which ensues have raised liability debates. Nevertheless, due to the distribute nature of attacks, identifying agents is not always possible and legal systems also vary around the world. If the targeted server is in one country but agents are distributed around other countries, enforcing liability would be very complicated. Additionally, a final victim seeking financial compensation for damages resulting from a DDoS attack is more likely to confront ISPs than end-users (as in the case of copyright infringement and Peer-to-Peer networks). In the future, if agents are identified, an ISP could face liability charges; thus, ISPs would benefit if end-users' machines are protected.

## *3.1 Source-end Defence Mechanisms*

Available source-end defence mechanisms can be divided into three categories: academic research level, corporate level, and personal level. At the time of this writing we are not aware of any solution aimed at detecting attack traffic at each particular agent machine in any of the three categories. The following sections explain similar approaches in each of the three categories. An approach is considered to be similar to our approach if it can be implemented at the source network.

### 3.1.1 Academic Research Level Solutions

At the academic research level, the following solutions are comparable to our approach. All of these solutions are implemented at the edge of networks in order to have access to all incoming and outgoing network traffic.

#### 3.1.1.1 MULTOPS (Multi-Level Tree for Online Packet Statistics) [40]

Gil and Poletto propose MULTOPS [40], which is a data-structure that network monitors and routers can employ to detect ongoing bandwidth attacks. MULTOPS is a tree

of nodes containing traffic ratio statistics at different aggregation levels. The tree contracts or expands depending on traffic patterns. Figure 3-2 depicts a high-level diagram of MULTOPS. The two interfaces provide the required inputs to measure incoming and outgoing traffic in one direction. Attack detection is based on the heuristic that under normal operation, traffic flows going to and coming from one direction are proportional.

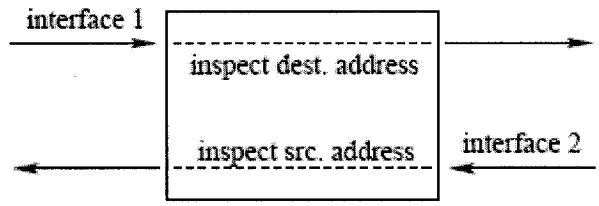


**Figure 3-2: MULTOPS [40]**

MULTOPS has two modes of operation: victim-oriented mode, and attacker-oriented mode. More than one MULTOPS is needed to detect both attacker and victim addresses. The first mode identifies the victim of an attack, and the second identifies the source(s) of an attack. Depending on the operation mode, a flow is classified as an attack if the traffic ratio drops below a threshold (victim-oriented mode) or if the traffic ratio surpasses a threshold (attacker-oriented mode). Once flows are classified as an attack, these flows are rate-limited. One drawback of MULTOPS is that if IP spoofing is employed, MULTOPS may impose collateral damage by dropping legitimate packets when operating in victim-oriented mode. Moreover, MULTOPS might not detect the attack at all when operating in attacker-oriented mode. Another drawback of MULTOPS is possible misclassification of non-TCP traffic due to the fact that the heuristic of proportional flows is based on TCP operations (packet acknowledgements).

#### 3.1.1.2 D-WARD (DDoS Network Attack Recognition and Defense) [13]

Mirkovic proposed D-WARD [13], which detects and discards DDoS attack traffic originating at the network where the mechanism is deployed. Figure 3-3 depicts D-WARD's deployment location. D-WARD works by collecting two-way traffic statistics from the border router and comparing these statistics with previously built legitimate traffic models. When an anomaly is detected, rate limiting policies are applied. D-WARD also prevents

random IP spoofing. Rate limiting attack traffic is preferred over blocking attack traffic to allow faster recovery from false positives (legitimate traffic classified as attack).

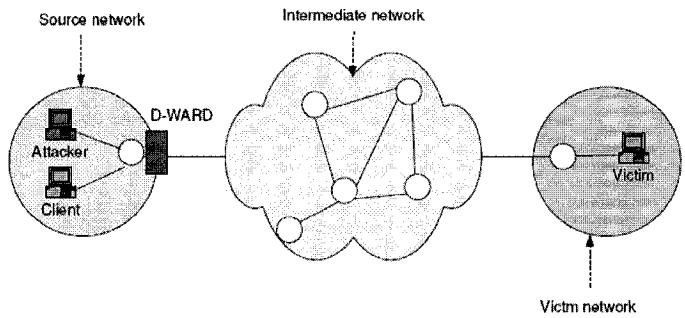


**Figure 3-3: D-WARD deployment location [13]**

Due to the fact that D-WARD applies rate limiting policies, the system is quite complex in terms of its implementation. For example, D-WARD distinguishes between flow traffic and connection traffic such that legitimate traffic is favoured during an attack (Figure 3-4 shows flow and connection concepts).

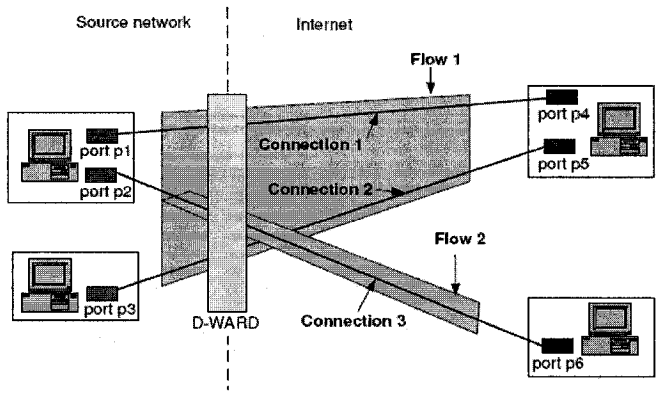


**Figure 3-4: D-WARD flows and connections [13]**

3.1.1.3 Stub-Domain DDoS Detection [41]

Kommareddy et al. proposed Stub-Domain DDoS Detection [41], a system consisting of two components: one for identifying attack flows, and the other for eliminating false positives by analysing inputs from the first component. This system is focused on detecting TCP-based DDoS attacks. Traffic monitors are attached to routers in order to avoid affecting their operation, thus making the system passive. Figure 3-5 shows the monitor's architecture.

Monitors are associated with routers at the Autonomous System (AS) where the attack packets are generated. An AS is a group of networks and routers usually administered for one entity under a common routing policy [42]. This is the first source-domain system designed to work in stub-networks. A stub-network is one that usually carries packets destined to a local host [43]. Unlike MULTOPS and D-WARD, this is a distributed system that is able to manage arbitrary stub-network topologies. Traffic monitors are distributed throughout the AS. This system works with asymmetric traffic and it bases its decision on traffic ratio as in MULTOPS. Different monitors communicate with one and other in order to decide on suspicious flows and decrease the number of false positives.

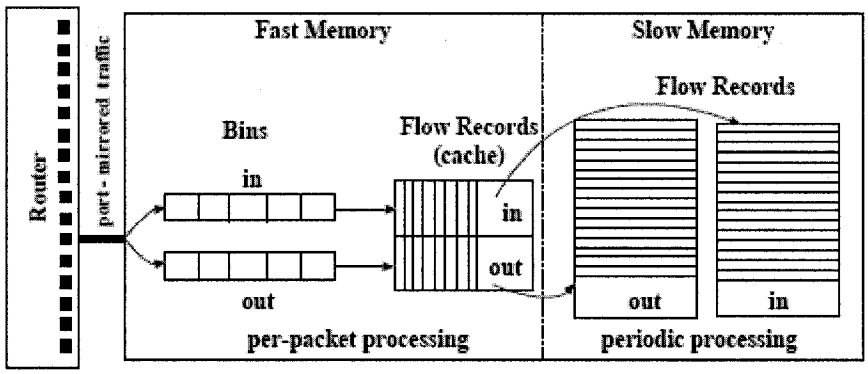


**Figure 3-5: Monitor architecture [41]**

3.1.1.4 FDS (Flooding Detection System) [44]

Wang et al proposed FDS [44], which monitors traffic at leaf routers that connect end hosts to the Internet. The FDS can be located either at the first-mile leaf router (near the source) or at the last-mile leaf router (near the targeted victim). Figure 3-6 depicts FDS installation's location. FDS bases attack detection on the relationship between SYN packets and FIN packets. Under normal circumstances, a SYN packet will eventually generate a FIN within a normal flow. The authors deal with the case of connections terminated with RST packets by empirically recognizing some of the total RST packets as active RST (which is equivalent to a FIN packet).

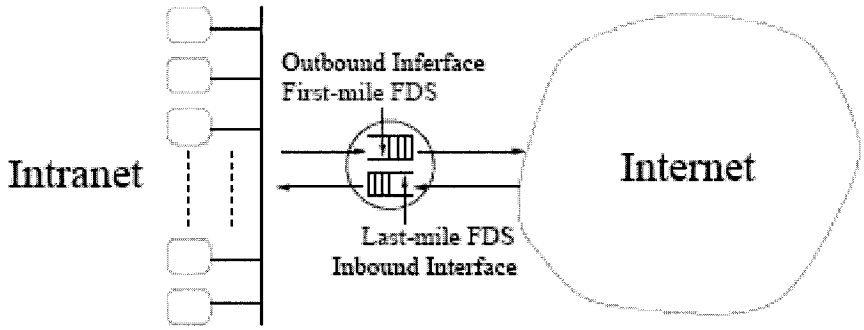


**Figure 3-6: FDS deployment locations [44]**

3.1.1.5 A Covariance Analysis Model for DDoS Attack Detection [45]

S. Jin and D. Yeung proposed a covariance model to detect DDoS attacks [45]. SYN flooding attacks are used as an application of the proposed method. As with FDS, this detection method can be implemented near the targeted victim or near the source of the attack. This covariance analysis model is a statistics-based method, and for SYN flooding attacks, all TCP flags are used as features in the model. The correlations among the features (the TCP flags) differentiate normal traffic from attack traffic. One advantage of this method over other statistical methods is the absence of assumptions about packet distribution.

### 3.1.2 Corporate Level Solutions

At the corporate level, the reverse firewall offered by Cs3 [46] is a comparable approach.

Traditional firewalls protect networks from incoming malicious traffic whereas the reverse firewall protects the external network from flooding DDoS attacks. By monitoring two-way connections, packets are filtered out and rate-limiting policies are applied if anomalies are detected. The reverse firewall is a hardware-based solution targeted for ISPs, universities, and corporations. Figure 3-7 shows one example of network architecture with two reverse firewalls. An advantage of the reverse firewall (over other solutions) is it supposedly does not require updates to detect new types of attacks.

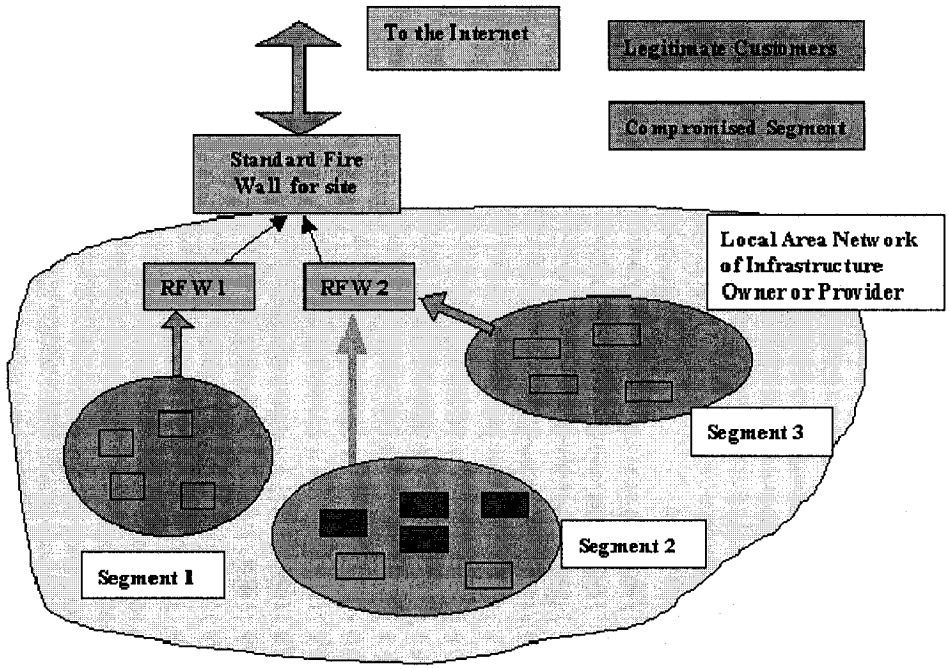


**Figure 3-7: The reverse firewall [46]**

### 3.1.3 Personal Level Solutions

At the personal level, user defence mechanisms are personal firewalls ([47], [48], [49], [50], [51]), port scanners ([52], [53], [54]) and antivirus software (see [30] for a list of available antivirus applications). None of these solutions is particularly aimed at detecting attack traffic. If they are layered and properly configured, intrusions might be prevented and the same final goal of eliminating agents' participation in DDoS attacks could be achieved. In fact, bot infection rate dramatically decreased from 30,000 per day in mid July 2004 to below 5,000 per day at the end of the same year. This drastic reduction is attributed to the release of Windows XP Service Pack 2 along with regular vulnerability patching [10]. Nevertheless, it should be recalled that 67 percent of the general public do not use personal firewalls [9]; this reluctance could be due to unawareness of security breaches as well as difficulty of the installation process. Additionally, 5,000 bot recruitments per day is still a very large number. Another important aspect of personal firewalls is the interaction with the end-user; depending on the configuration, the firewall might ask the user whether or not access should be allowed. Attack tools frequently masquerade themselves by using a general

file on the operating system to gain access to the Internet; therefore, it is not obvious that it is an attack tool requesting access to the Internet.

Regarding port scanning tools, due to the increasing use of legitimate services to perpetrate DDoS attacks, obscure ports are no longer used to suspiciously listen for connections. Therefore, these tools are of little help to the general public, especially if the users have limited technical knowledge. Precisely, most of these applications present open ports in terms of the processes using the port (for example, svhost.exe in Windows-based systems), which does not make it immediately obvious whether it is a legitimate application using the port. Figure 3-8 shows the Active Ports [52] port scanning software user interface. Moreover, in the case of Active Ports, Norton Antivirus reports it as a security threat, which will confuse even more end-users.

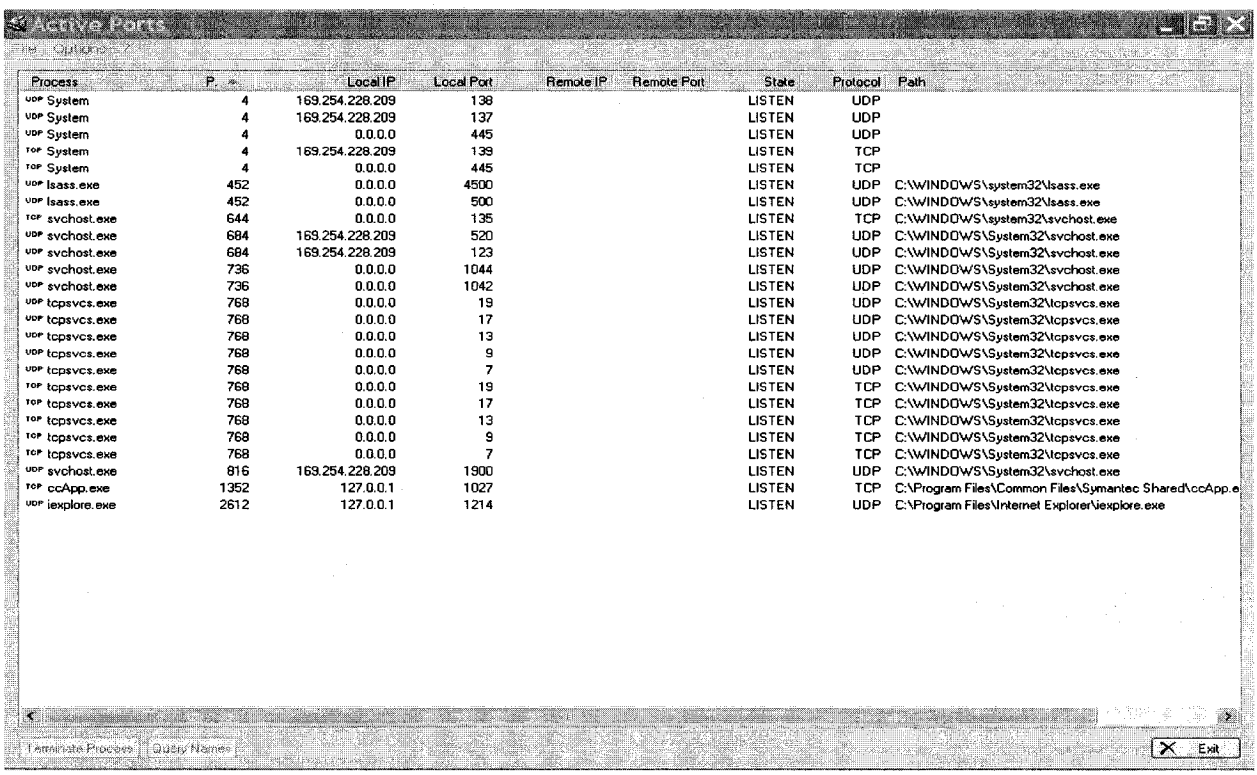


**Figure 3-8: User interface of Active Ports software [52]**

Finally, due to the increasing number of new vulnerabilities, it is not realistic to rely on antivirus software since usually vulnerabilities are known, at least in the attacker

community, before patches or updates are available. Moreover, attack tools frequently block access to security sites and also disable antivirus software.

As previous sections have shown, attackers manage to keep recruiting agent machines in spite of current defence mechanisms. Detecting intrusion is difficult for the general public not only because of the lack of an easy-to-use tool, but also because of the fact that usually attackers disable firewalls and antivirus software to avoid detection [37]. Based on the fact that agent machines are $2^{nd}$, $3^{rd}$, and $n^{th}$ layer victims, we propose to implement an easy-to-use tool for monitoring two-way communications at the agent machines in order to detect TCP DDoS attack packets.

# CHAPTER 4

# TCP DDoS ATTACKS

As discussed in Chapter 2, DDoS attacks' success is greatly based on the ability of attackers to recruit a large army of agent machines by varying attack methodology incredibly fast at a very low cost. Due to this diversity in DDoS attack strategies, it is not realistic to tackle all type of attacks with one single solution. DDoSniffer focuses on TCP-based attacks not only because most Internet traffic is two-way communications (TCP is about 90 percent of Internet packets [13]), but also because more than 90 percent of attacks use TCP [44]. In order to have two-way communications, TCP clients and TCP servers must keep track of several parameters (for each way) such as sequence numbers, port numbers, and acknowledgement numbers among others. However, a TCP connection is uniquely identified by a pair of sockets, that is, the source IP address and the source port number, and the destination IP address and the destination port number [55]. Figure 4-1 depicts an example of five TCP connections.

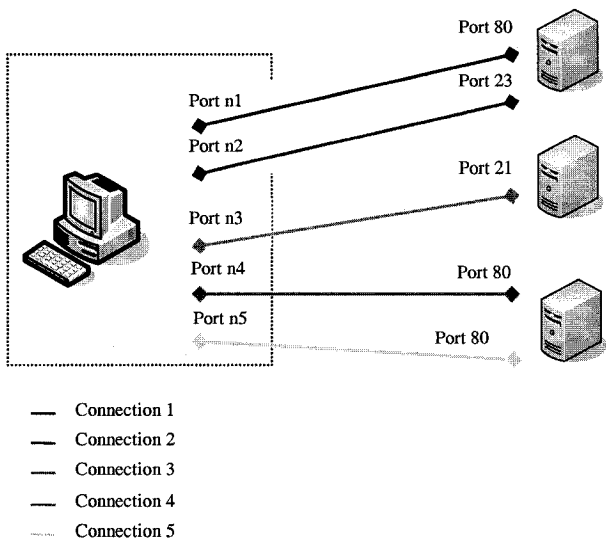


**Figure 4-1: TCP connections**

As TCP is a connection-oriented protocol, before a TCP client and a TCP server can send data to one and other, a TCP connection must be established between these two ends. TCP establishment starts with the three-way handshake; after completing the handshake the TCP connection enters the established state in which data flows between end points (i.e. the client and the server). To terminate the connection, either end sends a FIN packet which is then acknowledged by the other end, and at this point the TCP connection is said to be half-close. The end that closes the connection could still receive data because closing a connection means that this particular end has no more data to be sent. The connection is completely close when the remaining end sends a FIN packet, which is also acknowledged. Thus, opening a connection requires three packets and closing it requires four packets [56]. It is also possible to terminate a connection by sending a RST packet which does not require acknowledgement from the other end. Under normal conditions, legitimate TCP connections are expected to have more than four packets since only the establishment phase requires three packets. Figure 4-2 depicts an example of a TCP packet flow including establishment and termination packets. In the example depicted in Figure 4-2, the client finalizes its connection first (by sending the FIN packet), but it should be noted that this is not always the case.

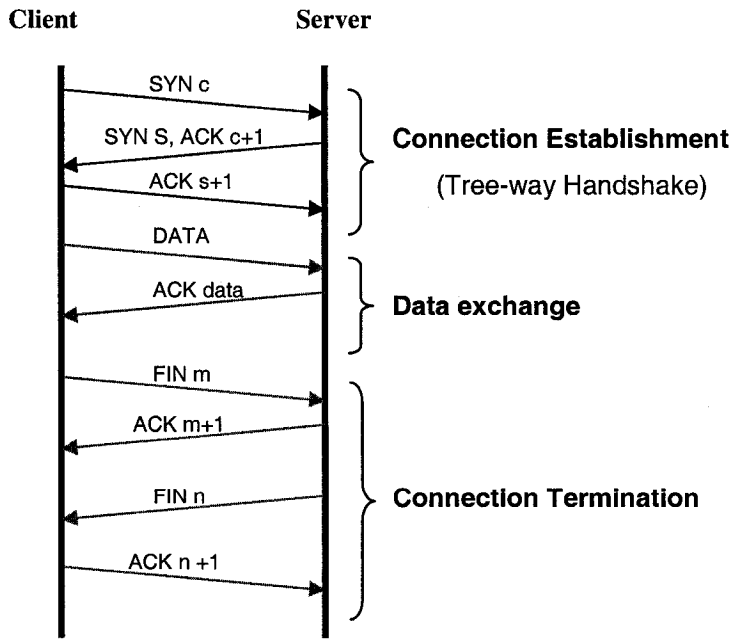


**Figure 4-2: Example of TCP packet flow**

Another characteristic of a legitimate TCP connection is its traffic ratio which is given by the number of outgoing packets divided by the number of incoming packets, as shown in equation 4-1. For example, the traffic ratio for the connection depicted in Figure 4-2 is 1.25.

$$ConnectionRatio = NumberOfSentPackets / NumberOf\,\mathrm{Re}ceivedPackets \quad \textbf{(4-1)}$$

TCP provides reliable two-way communication between hosts; communication reliability is accomplished by requesting acknowledgments from the receiver-end, which also controls the amount of data sent by the sender-end so that buffers do not overflow [55]. Due to the TCP flow control mechanism, the difference between received packets and sent packets in a TCP connection is expected to be relatively small (since packets are acknowledged). Recall that a TCP connection is defined by the source and destination port numbers and by the IP addresses of the source and destination machines [55]; legitimate TCP connections can be characterized by the ratio of sent packets to and received packets from a given destination [13].

Under normal circumstances, the lack of acknowledgments could be attributed to network congestions or other related problems; however, during a DDoS attack, no acknowledgments will be sent out because the victim will be overwhelmed. For the duration of a DDoS attack, attack connections keep sending out packets regardless of the lack of acknowledgments. Consequently, the traffic ratio of an attack connection is higher than usual because the number of outgoing packets is greater than the number of incoming packets. Determining a traffic threshold to distinguish legitimate traffic from attack traffic is a key factor in order to avoid a large number of false positives in attack detections. A false positive, also referred as to false alarm, occurs when a defence mechanism signals the presence of an attack while there is no attack ongoing. As the distribution of traffic is unknown, heuristics observations are necessary to determine typical values for the traffic ratio. More details about the threshold for normal traffic ratios is presented in section 6.1.3.

On the other hand, TCP attack vectors vary among attacks, but in their basis, TCP-based DDoS attacks generally break typical TCP operation in two ways: a) by generating connections with four packets or less, or b) by generating connections with traffic ratios

larger than usual. We refer to these two types of attacks as connection attacks and bandwidth attacks, respectively. As mentioned in Chapter 2, a given attack could be classified as both a flooding attack and a vulnerability attack. Based on this conventional attack classification, it is difficult to affirm what kind of attack DDoSniffer will detect, but based on how TCP semantics is broken, DDoSniffer will detect what we call connection attacks and bandwidth attacks.

## 4.1 Connection Attacks

Connection attacks are DDoS attacks that violate typical behaviour of TCP connections by generating connections with four packets or less. In normal conditions, legitimate TCP clients will complete the three-way handshake and will exchange data starting at the fourth packet. Even if a client finishes the connection at the fourth packet by sending a FIN packet, a legitimate TCP connection is expected to have more than four packets because each direction must be closed separately. In the case of flooding attacks, connections participating in an attack will have four packets or less: the three-way handshake (if completed) and probably one more packet with a service request. The case of vulnerability attacks is more complex because it depends on how the vulnerability is exploited. Precisely, if exploiting the vulnerability generates new connections with four packets or less, DDoSniffer should detect the ongoing attack.

SYN flooding attacks are the most common attacks among its class; however, there are several distinct attack vectors to mount a DDoS flooding attack. An attack vector is a mean by which attackers are able to mount a given attack. Table 4-1 shows several flooding attack vectors and the corresponding server's response.

| **Attack Packet** | **Server Response** |
|---|---|
| SYN (to a opened port) | SYN-ACK |
| SYN (to closed port) | RST |
| ACK | ACK |
| RST | No response |
| NULL | RST |
| FIN (to a closed port) | RST |

**Table 4-1: TCP flooding attack vectors**

Typical SYN flooding attacks are mounted to an open port; this type of SYN flooding attacks are usually referred to as SYN-ACK flooding attacks to distinguish them from the SYN-RST flooding attacks mounted to a closed port. In general, what applies to SYN flooding attack vectors also applies to other flooding attacks; this is why SYN flooding attacks will be explained in more details and differences will be pointed out when necessary.

### 4.1.1 SYN Flooding Attacks

Based on whether or not IP spoofing is used, current SYN flooding attacks can be divided into two categories: a) spoofed IP address attacks, and b) non-spoofed IP address attacks. Originally, SYN flooding attacks used spoofed IP addresses mainly to avoid tracing back attack packets to the source and thus exposing the attacker's identity. To countermeasure diverse defence mechanisms such as egress filtering [57], SYN-Cookies [58], [59], and [60], and TCP Intercept [61] along with others, current SYN flooding attacks usually employ the agent machines' real IP addresses. Nevertheless, spoofed address SYN flooding attacks cannot be ignored since they still represent a security threat.

In an attack employing IP spoofing, the attacker sends a SYN packet to the server with a source address other than its own. The server replies with a SYN-ACK packet that will be sent to the legitimate owner of the spoofed address. If a SYN-ACK packet from the server reaches the legitimate owner of the spoofed address, the legitimate owner's machine will generate a RST packet which is returned to the server. This frees up space in the server's connection request queue (if the request is still in the queue). Therefore, attack SYN packets must be generated at a rate fast enough to fill the server's connection request queue before the legitimate owners' RST packets reach the server.

To keep the server's connection request queue filled, attack packets must keep arriving at rate that forces the server to erase older received SYN requests from its queue such that if a RST packet comes from the legitimate machine, this request is not on the server's queue anymore and no space will be freed [62]. Moreover, a legitimate client trying to reach the server is effectively blocked out because its SYN request has been flooded out of the queue. Figure 4-3 shows an example of a SYN flooding attack employing IP spoofing. This type of SYN flooding attack will generate connections with less than four packets;

specifically, every attack connection will have only one packet, the SYN packet. DDoSniffer should detect this type of attacks based on its connection attack detection algorithm that monitors the number of connections with four packets or less. More details in the detection algorithm are given in section 5.1.

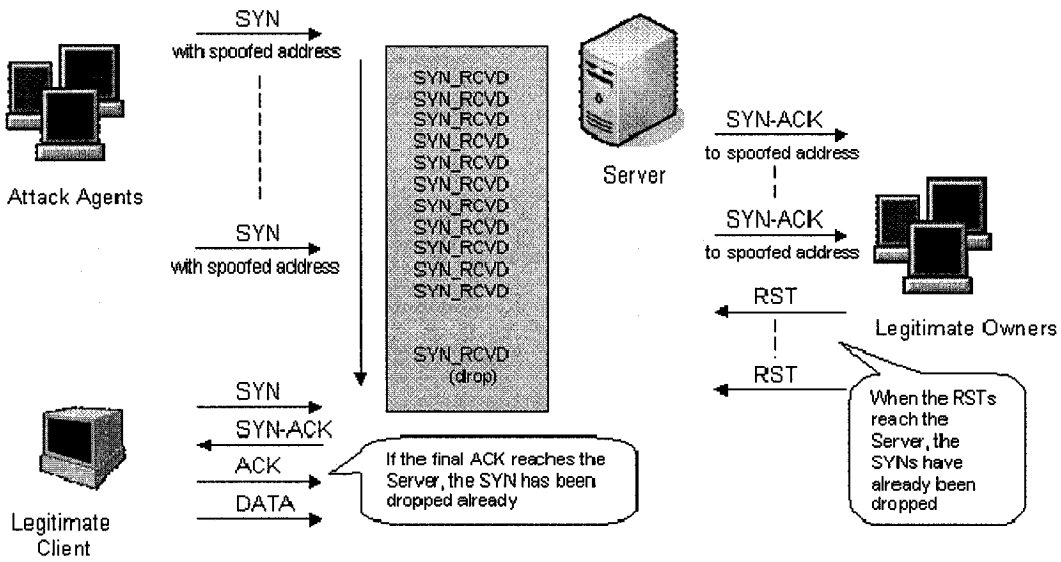


**Figure 4-3: SYN flooding attack**

Regarding the case of non-spoofed IP addresses, attackers have generally three options:

- Modifying the impersonated machine TCP/IP stack to silently discard the acknowledgement packet from the server [3]. The modification to the TCP/IP stack is needed to avoid the stack sending a RST packet which will free the space on the server's connection request queue. This type of attack will produce connections with two packets: the SYN and the SYN-ACK packets.
- Sending attack packets faster than SYN + ACK Round Trip Time [62]. The SYN + ACK Round Trip Time is the time between sending the SYN-ACK packet and receiving the final ACK packet. If the final ACK reaches the

server when this particular connection request is not on the server's queue, the connection will not be completed, and no space will be freed on the queue. Moreover, if a legitimate client manages to get its connection request on the server's queue, by the time this client's final ACK reaches the server, the connection request will have been displaced by the attacker's requests and no connection will be completed (the attacker succeeds). This attack is shown in Figure 4-4. Attacks of this kind will create connections with three packets: the SYN packet, the SYN-ACK packet, and the RST packet. The RST packet is sent because the TCP/IP stack was not modified. When the server is totally overwhelmed, the SYN-ACK packet will not be sent out, and attack connections will only have one packet per connection (the SYN packet).

- Completing the three-way handshake and exhausting a different queue or resource in the server. To avoid some defence mechanisms that prevent traditional SYN flooding attacks, attackers complete the connection and sometimes request a service from the victim. An example of this type of attack is the HTTP-GET attack, after completing the three-way handshake, an HTTP GET packet is sent requesting a web page. In this case, the attack aims to consume the maximum number of established connections the targeted server can concurrently handle. This type of attack is very difficult to detect and prevent because the attacker uses legitimate service requests. Connection patterns of this attack methodology depend on whether or not a service is requested after completing the three-way handshake. If no service is requested, attack connections will have a maximum of three packets (the three-way handshake); when the attack attains the denial of service stage, connections will have less than three packets because the server will be overwhelmed and no SYN-ACK will be sent out. On the other hand, if a service is requested, connections will have four packets: the three-way handshake plus the service request packet. Probably, at the beginning of the attack, connections will have more than four packets because the server will be able to reply to the service request, but after a given time the victim will be overwhelmed.

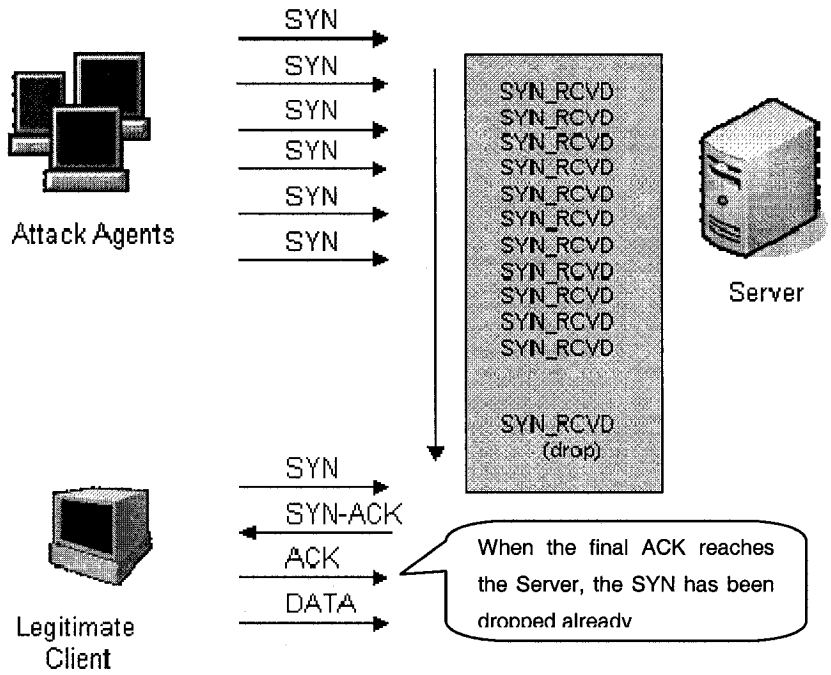


**Figure 4-4: Non-Spoofing SYN flooding attack**

Regarding SYN flooding attacks to a closed port, when a SYN packet is sent to a closed port on the victim, a RST packet will be sent out by the server. SYN flooding attacks to a closed port can be part of a SYN flooding attack which is not directed to a specific port; instead the attacker sends SYN packets to several ports probably aiming to exhaust the victim's bandwidth. The volume of RST packets consumes the targeted server's bandwidth, but also open ports will be compromised (based on the same principles of the SYN flooding attacks to open ports). SYN flooding attacks to a closed port can also be part of a scanning process in order to determine vulnerable ports and applications. Once open ports are identified, SYN packets are sent to these ports to consume the server's connection request queue. SYN flooding attacks to closed ports will generate connections with two packets: the SYN packet and the RST packet.

### 4.1.2 ACK Flooding Attacks

Another very well known attack is the ACK flooding attack, also called ACK storm attack. In an ACK flooding attack, attack packets are sent with wrong sequence numbers; the

server replies with an ACK packet containing the expected sequence number, and the attacker replies (or re-send) an ACK packet with a wrong sequence number; this exchange of ACKs continues indefinitely causing the server to exhaust its resources [63]. Depending of the attacker's methodology, an ACK flooding attack will generate connections with four packets or less, or it will generate connections with traffic ratios larger than usual (in either case DDoSniffer will detect this type of attack).

### 4.1.3 RST Flooding Attacks

The TCP flow control mechanism allows the receiver-end to control the number or TCP segments or packets the sender-end could transmit without waiting for an acknowledge packet [55]. It is the 16-bit window field in the TCP header that indicates the number of packets the sender of the packet will be accepting from the other party. For example, if an ACK is received for packet sequence number 2001 and the window field has a value of 3000, then the sender of this packet is expecting packets with sequences numbers between 2001 and 5001. An attacker can terminate an established TCP connection by sending a RST packet with an ACK field that falls within the window of sequence numbers the server is expecting to receive. Figure 4-5 shows an example of a connection termination through a RST packet.

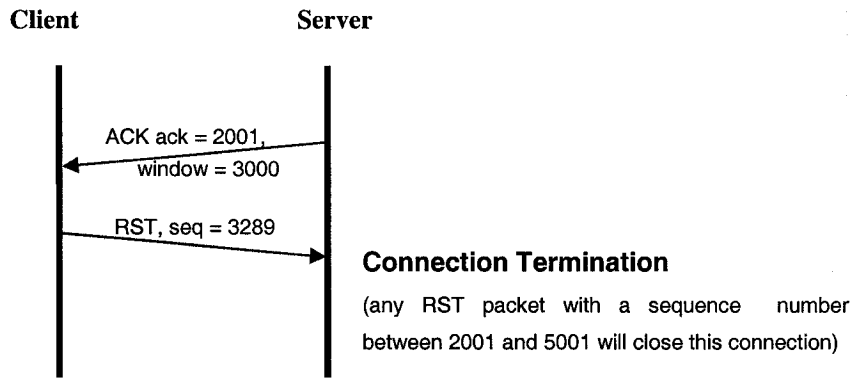


**Figure 4-5: Example of connection termination**

Originally, the brute force required to perform a RST attack was wrongly estimated because the window of sequence numbers was not taking into account. It was said that attackers needed to guess the IP address, the port number, and the sequence number. Nevertheless, it is not necessary to find a match of the sequence number instead the attack

packet's sequence number only needs to fall within the expected window, and the larger the window the most chances to reset (terminate) the connection [64]. A RST flooding attack will generate connections with only one packet, the RST packet, because every change in the IP address and/or port number will generate a different connection.

It is also possible to mount an RST attack by setting the SYN flag [64]. In this case, the server will interpret this packet as a duplicated SYN packet. The TCP specification indicates that the server should reply with a RST packet and close the connection; this procedure was designed to allow recovery from a system crash or duplicated old SYN packets [55]. Still, attackers need to perform brute force to guess the IP addresses and port numbers in order to send a SYN packet that falls into the established connection. Attack connections of this methodology will have at the most two packets: the SYN packet (used to reset the connection), and the RST packet sent by the server.

### 4.1.4 NULL Flooding Attacks

A NULL flooding attack is similar to the traditional SYN flooding attack, but the three-way handshake semantics is broken by sending a packet with a null flag (all flags set to zero). This null flag packet can cause the connection to enter an invalid state (instead of the established state). The targeted victim can become unresponsive due to these connections; for example, a device running the operating system CatOS of Cisco will stop working and reload [65]. A NULL flooding attack will generate connections with three packets: the SYN packet, the SYN-ACK packet, and the null flag packet.

### 4.1.5 FIN Flooding Attacks

These attacks are directed to closed ports to generate a flood of RST packets in respond to the FIN packets. A FIN flooding attack consumes the network resources of the victim. Attack connections of this type of attacks will have two packets: the FIN packet, and the RST packet.

By monitoring connections with four packets or fewer, the above mentioned attacks could be detected because for the duration of a DDoS flooding attack, the number of connections that remain with four packets or less will grow very fast. As attack traffic resembles legitimate traffic, classification of traffic cannot be based on per-packet

observations. DDoSniffer keeps connections with four packets or less in a table called *new-connection* table; a legitimate TCP connection should remain in this table for a very short period of time. Detection of connection attacks is based on the size of the new-connection table. More details on the detection algorithm is given in section 5.1.2.

## 4.2 Bandwidth Attacks

DDoS bandwidth attacks violate the semantics of the TCP flow control mechanism. Conforming to the flow control specification, a TCP client will send packets according to the TCP server's permission. This permission is based not only on the server's resources, but also on the network conditions. For instance, packets lost due to network congestions or buffer overflows will force a legitimate client to slow down its transmission rate to adapt to the current situation. An impersonated client (attacker), on the other hand, will not wait for the server's acknowledgements; instead this client will keep aggressively sending attack packets to consume the victim's resources. In normal conditions, a legitimate connection will have a small difference between the number of sent packets and the number of received packets. Bandwidth attack detection is based on this difference of sent packets and received packets; precisely, bandwidth attack detection is based on traffic ratio (as defined by equation 4-1).

An attack connection will have a traffic ratio larger than usual due to its aggressive behaviour of continuously sending packets out regardless of the lack of response from the victim. Determining benchmarks for bandwidth attack detection, based on traffic ratio, poses a big challenge because the traffic ratio of a given connection depends on the following factors:

- TCP implementation - The TCP specification does not define an explicit algorithm to manage the window of sequence numbers. The window field in the TCP header indicates the number of packets the sender is allowed to send without waiting for acknowledgments. In other words, the sender does not need to stop its transmission and wait for an acknowledgement for every single packet. This flow control mechanism is called sliding-window [56]. Moreover, usually a TCP client/server does not acknowledge receiving a packet immediately; instead the TCP client/server waits to see if there is data

to be sent in the same direction as the ACK [56]. This technique is referred to as piggybacking the acknowledgement with the data. Due to the sliding-window protocol as well as the delays in sending acknowledgements the relationship of sent packets and received packets is not 1 to 1. Still the difference between the number of sent packets and the number of received packets should be small enough to expect the traffic ratio to fluctuate between 1 and 4 empirically determined as shown in section 6.1.3.

- User's behaviour - TCP uses different algorithms to manage diverse types of data. FTP and electronic mail applications, for instance, generate bulk data whereas Telnet produces interactive data. Delayed acknowledgments, selective acknowledgments, the Nagle algorithm [66], window scaling [67] and other extensions to the original TCP specification cause variations in the traffic ratio. Moreover, these variations depend on user's behaviour because it is the user who decides which applications to run.

Regarding the attackers' methodologies, several techniques discussed in section 4.1 can also be used to mount a bandwidth attack; for example, an HTTP-GET attack that keeps requesting different web pages without waiting for a response from the server will be a bandwidth DDoS attack. During a HTTP-GET flooding attack, several connections are concurrently open and each connection requests a web page. During a HTTP-GET bandwidth attack, the same connection requests several web pages. The final effect of both attacks is the same: a denial of service; one is accomplished by filling out the server's queue, and the other by exhausting the server's bandwidth. Figure 4-6 illustrates an example of bandwidth attack using HTTP-GET.

In general, attack classification is becoming very fuzzy because attackers are able to change attack vectors very easily. In particular, what is important is to detect ongoing attacks regardless of their classification and methodology. By monitoring both the number of connections with four packets or less and the traffic ratio per connection, DDoSniffer is able to detect several TCP-based DDoS attacks.

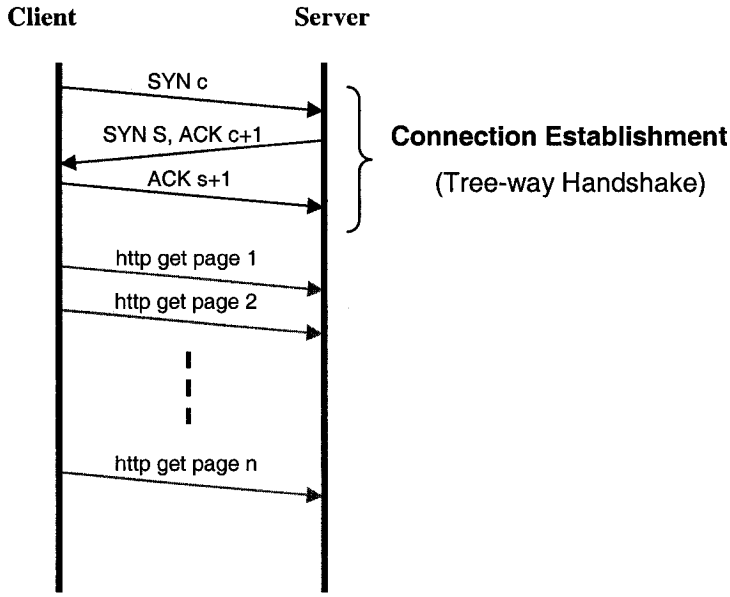


**Figure 4-6: Example of a bandwidth attack with HTTP-GET**

# CHAPTER 5

# DDoSniffer

DDoSniffer is an attack detection tool deployed at the source network; precisely, DDoSniffer monitors network traffic at each agent machine participating in DDoS attacks. Figure 5-1 illustrates DDoSniffer's deployment location. DDoSniffer detects TCP-based DDoS attacks by monitoring network traffic and building connection tables for distinguishing attack traffic from legitimate traffic. DDoSniffer is a passive packet sniffer, in other words, the machine's network traffic is not affected by our software tool. The ultimate goal of DDoSniffer is to detect outgoing TCP-based DDoS attacks and raise an alarm to the end-user notifying the existence of the attack traffic.

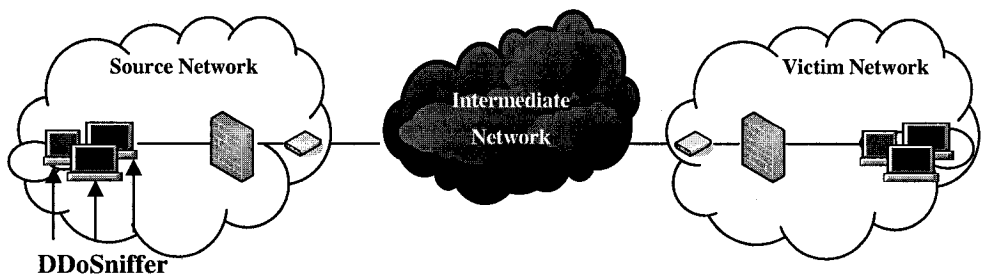


**Figure 5-1: DDoSniffer deployment location**

Another implicit goal of DDoSniffer is to keep processing time per packet as short as possible to be able to handle all packets, especially during heavy traffic load. If packets are not measured in a timely manner, DDoSniffer may fail by raising a false alarm or by missing detection of an ongoing attack. A false positive, also referred to as a false alarm, is generated if the detection mechanism signals an attack alarm when no attack is ongoing. On the other hand, a false negative is when an ongoing attack is not detected by the mechanism. Accuracy

of a detection mechanism is usually measured in terms of the false positive and false negative rates. A detection mechanism that generates a significant number of false positives is not reliable. Similarly, a detection mechanism whose false negative rate is high is not efficient. To keep both rates (false positives and false negatives) low, DDoSniffer attack detection is not based on per-packet observation; instead traffic patterns are studied over time to achieve more exactness.

DDoSniffer is designed to detect TCP-based attacks that fall within the two categories introduced in chapter 4: connection attacks, and bandwidth attacks. An extra feature of DDoSniffer is IP spoofing detection which is mostly employed in reflection attacks (as explained in Chapter 2.)

## 5.1 DDoSniffer Architecture

DDoSniffer monitors TCP traffic and builds two different tables in order to keep traffic statistics such that anomalies in traffic are detected. As discussed in section 4.1, TCP-based DDoS attacks generally break the TCP semantics in two manners: a) by generating connections that have less than or equal to four packets; or b) by generating connections that have traffic ratios greater than a threshold. Each of these two attacks is detected by different processes (running in parallel), and every one of these processes depends on one of the two connection tables: 1) the new-connection table, and 2) the connection table.

Distinguishing legitimate traffic from attack traffic is based on observing these two connection tables and comparing current traffic patterns with what is considered to be normal traffic patterns. As attack traffic resembles legitimate traffic, a given packet cannot be classified upon its arrival to the capturing module; instead it has to be put into the tables for a later classification. This classification of traffic is done over time in order to be more accurate in attack detection. Another reason to avoid classifying packets immediately is to prevent missing out packets due to the time spent processing each packet.

Three parallel components form the architecture of DDoSniffer: 1) the *capturing module*, 2) the *new-connection module*, and 3) the *classification module*. Figure 5-2 illustrates a high level diagram of DDoSniffer's architecture. The three modules run in parallel sharing access to the two tables (the new-connection table and the connection table).

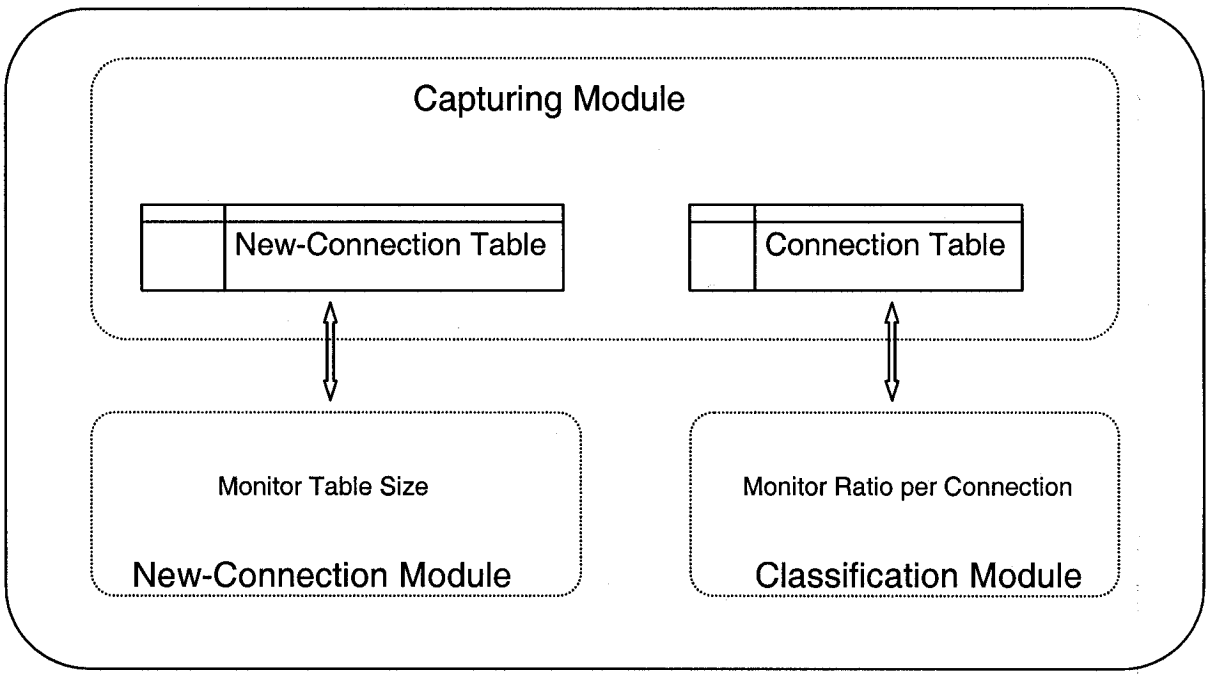


**Figure 5-2: DDoSniffer's architecture**

### 5.1.1 Capturing Module

This is the core module of DDoSniffer. The capturing module monitors network traffic, filters out all but TCP traffic, and builds the two tables in which the other two modules base their operations on. This module does not affect the network traffic in any manner because it captures the packets from the TCP/IP stack. The new-connection table and the connection table are shared among the three modules. They are hash-tables indexed by the pair of sockets participating in the connection of each record. Figure 5-3 shows the order in which the parameters are used to generate the index for each table.

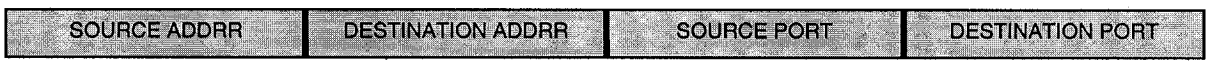


**Figure 5-3: Index for DDoSniffer tables**

Although for attack detection only the number of outgoing packets and the number of incoming packets might be sufficient, ten fields are stored in each record of both tables because some extra information is necessary in order to base detection on overall changes in

packet behaviour instead of on an individual per packet basis. The following ten fields are part of a connection record in each of the two tables:

1. The source IP address
2. The destination IP address
3. The source port
4. The destination port
5. The number of incoming packets
6. The number of outgoing packets
7. The traffic ratio – Defined as in equation 4-1 (the number of outgoing packets to the number of incoming packets). This field is computed only for connections with more than four packets.
8. The connection status – *Suspicious* if the number of packets is less or equal to four, *Normal* if the traffic ratio is less than $R$, and *Attack* if the traffic ratio is greater than $R$.
9. A time stamp – To signal the time of the last operation
10. Last status – Same values as field number 8.

The first four fields have the value of the first packet of the connection. For example, if the first packet of a given connection is an outgoing packet, the source IP address field contains the IP address of the machine DDoSniffer is running on (DDoSniffer machine henceforth), the destination address field takes on the destination address of the packet, and so forth. For the following packets of that connection, the first four fields will remain the same. The main reason not to update these fields is to avoid unnecessary manipulation of data that do not contribute in any manner to attack detection. Regarding the connection status field, when a connection is first added to the tables its status is set to "suspicious" because a new connection should be considered suspicious till its number of packets is greater than four. Once the connection has more than four packets its status will be updated by the classification module.

Captured packets are handled as follows:

1. Before passing the packet to the tables, the capturing module first verifies whether or not IP spoofing is being used; if IP spoofing is detected, the user is notified.
2. Once it has been verified that IP spoofing is not taking place, the next step is to check if the captured packet belongs to an existing connection.
    a. In the case that this packet belongs to a new connection, the connection is added to both tables.
    b. If the packet is part of an existing connection, the respective packet counter (incoming or outgoing) is incremented.
        i. It is necessary to verify the total number of packets on the connection the captured packet belongs to.
            1. If the connection has more than four packets, this connection is removed from the new-connection table and its ratio is computed and saved on the connection table.
            2. If the connection has less than four packets, no action is taken.

Figure 5-4 depicts the flow of a captured packet in the capturing module.

It can be argued that one table would be not only sufficient for attack detection, but one table would also simplify the capturing module's operations. Up to a certain degree, this is a valid argument. Nevertheless, keeping connections with four packets or less in a separate table in fact facilitates attack detection as it is explained in the following section.

### 5.1.2 New-Connection Module

The new-connection module monitors the size of the new-connection table which keeps record of current open connections that have four packets or fewer. Legitimate connections remain in the new-connection table until the three-way handshake is completed and the data exchange starts; under normal circumstances a legitimate connection completes the three-way handshake in fractions of a second. During a DDoS flooding attack, the new-

connection table's size increases in a few seconds because attack connections remain open with less or equal than four packets (as explained in section 4.1). If the size of this table is greater than *NEWCONN* for four consecutive observation periods, an alarm of an ongoing attack should be raised to the end-user. NEWCONN is an upper limit to the size of the new-connections table because only NEWCONN connections are allowed to be in the table for four consecutive observation periods. NEWCONN is a system parameter that is empirically adjusted as explained in section 6.1.2. Recall that the new-connection table is a hash-table indexed by the source and destination addresses and the source and destination ports. This module detects TCP flooding attacks as well as any other attack that breaks the TCP semantics by generating connections with four packets or fewer.

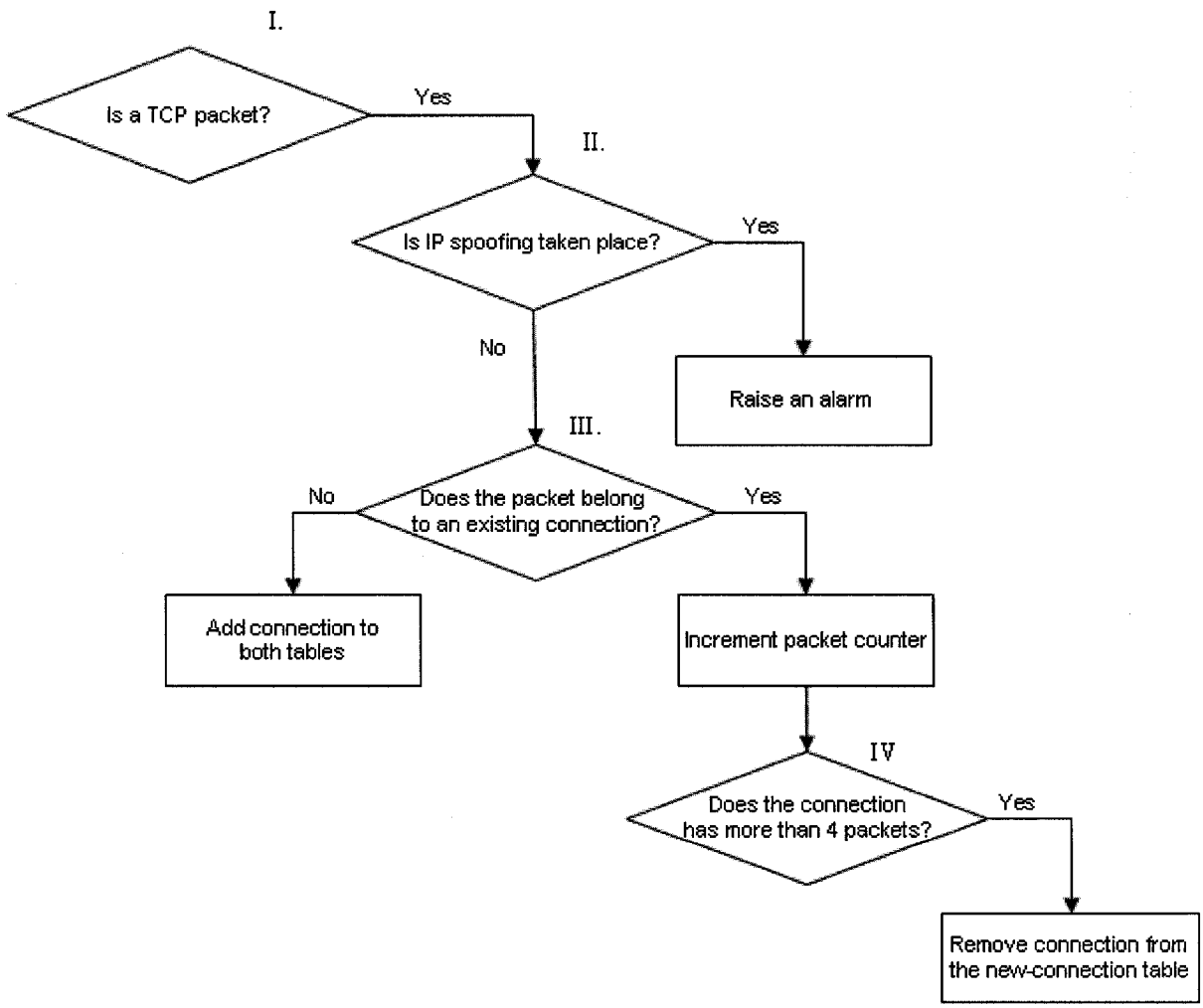


**Figure 5-4: Flow of a packet in the capturing module**

This module's operation is simplified due to the fact that connections with four packets or less are kept in a separate table. Instead of having to read every record in the connection table to count the number of connections that has four packets or fewer, the size of the new-connection table is retrieved at every observation interval. Under normal circumstances, the size of the new-connection table should ideally be near zero because legitimate connections complete their three-way handshake in fractions of a second. It could be possible that in one observation interval some legitimate connections have not completed their three-way handshake, but most likely for the next observation interval the situation will be different. It is very unlikely that for four consecutive observation intervals more than NEWCONN legitimate connections remain in the new-connection table.

DDoSniffer attack detection sensitivity highly depends on the adjustment of the NEWCONN parameter. A shorter value provides faster detection of connection attacks, but it also increments the number of false positives. Recall a false positive occurs when no attack is ongoing but an alarm of an attack is raised. There is a trade-off between detection time and detection accuracy; section 6.1.2 explains how the NEWCONN parameter is adjusted.

The flow chart of this module is as follows:

1. After obtaining the size of the new-connection table,
    a. If the size of the table is less or equal to NEWCONN,
        i. The counter of consecutive periods is reset to zero.
    b. If the size of the table is greater than NEWCONN,
        i. The next step is to check the number of consecutive observation periods the size of the table has been greater than NEWCONN.
            [1] If number of consecutive observation periods is less than 4, the counter is incremented.
            [2] An alarm is raised to the end-user only after four consecutive periods of observing the size of the table being greater than NEWCONN.

Figure 5-5 illustrates the flow of the new-connection module.

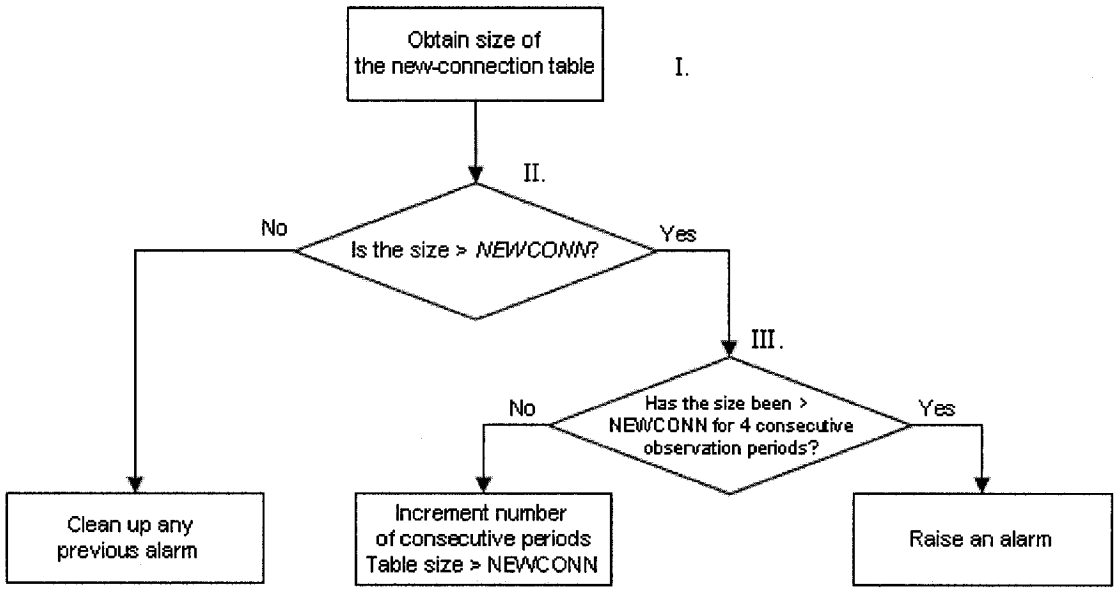


**Figure 5-5: Flow of the new-connection module**

### 5.1.3 Classification Module

The classification module reads the connection table which registers all open connections regardless of the number of packets in each connection. Connections are classified as normal or attack based on its traffic ratio. In this case, an attack alarm is raised if at least one connection has its traffic ratio greater than *R* for two consecutive observation periods. As in the case of connection attack detection, the sensitivity of the DDoSniffer for detecting bandwidth attacks depends on a system's parameter, in this case: *R* (empirically determined as discussed in section 6.1.3). A shorter value allows faster detection, but it also increases the number of false positives in bandwidth attacks. Likewise the new-connection table, the connection table is a hash-table indexed by the pair of sockets participating in the connection. By keeping two different tables for connections, we avoid the overhead of having to verify the number of packets each connection has at every observation period, which in time of an attack could be very time consuming due to the large size of the connection table. The classification module detects bandwidth attacks.

This module bases its operation on the connection table built for the capturing module. The traffic ratio of each connection is updated by the capturing module and not by the classification module. In this manner, no extra time would be unnecessarily spent computing traffic ratios for connections that have not completed the three-way handshake. Figure 5-6 depicts the flow of the classification module.

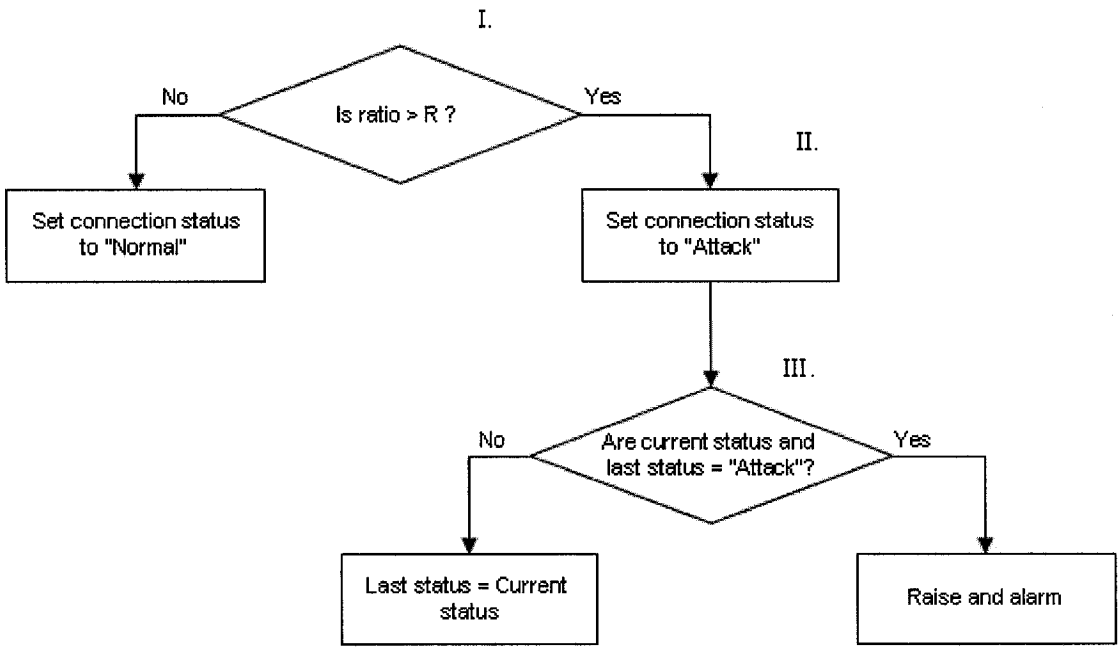


**Figure 5-6: Flow of the classification module**

# CHAPTER 6

# EXPERIMENTAL SETUP

Ultimately, the measure of DDoSniffer's usefulness as a DDoS defence mechanism greatly depends on its ability to identify attack activity. Diverse experiments were designed to test DDoSniffer in simulated real-world conditions. Particularly, the experiments facilitate the definition of metrics from which the success (or failure) of DDoSniffer can be measured. The simulated real-world conditions are derived from data collected by a network technology firm.

As DDoSniffer is deployed at each agent machine, a large network testing environment is not necessary. Instead, DDoSniffer's performance is measured in terms of the lowest detectable attack rate as well as the shortest attack duration. Whether or not this lowest-rate attack is sufficient to mount a successful DDoS attack depends on the number of agent machines participating in the attack. However, from DDoSniffer's standpoint, the number of agent machines forming part of the attack is irrelevant. Figure 6-1 illustrates the network architecture employed during experimentation.

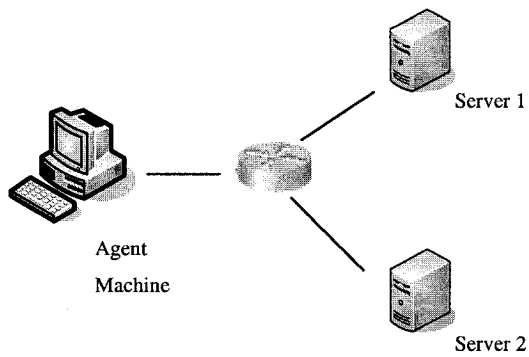


**Figure 6-1: Network architecture employed for experimentation**

DDoSniffer was implemented using the Microsoft Visual C++ .NET development platform and is intended for use on Windows XP machines. The implementation was done as

a part of this thesis as well as part of a research project for Cistech Limited [68]. The experimental work consists of three parts:

1. DDoSniffer – The detection tool developed following the design principles discussed in chapter 5.
2. Background traffic generator – A tool specifically designed and implemented to simulate legitimate traffic during the tests.
3. Attack tool – To evaluate DDoSniffer, an attack tool was deployed to test detection of diverse attack types.

It should be noted that all three of these tools run on the agent machine shown in Figure 6-1.

This chapter consists of three sections. Section 6.1 explains the implementation details of DDoSniffer. The background traffic generator tool is described in Section 6.2. In Section 6.3, the attack tool is presented.

## *6.1 DDoSniffer*

DDoSniffer is a multithreaded software tool. As discussed in chapter 5, DDoSniffer consist of three components: the capturing module, the new-connection module, and the classification module. Each of these three modules is deployed as a separate thread sharing access to the two main resources: the new-connection table, and the connection table. Additionally, DDoSniffer has two parameters to be configured: *NEWCONN*, the number of new connections allowed in the new-connection table for consecutive observation periods, and *R*, the traffic ratio threshold for bandwidth attack detection. Table 6-1 shows the values for the system's parameters; both parameters affect DDoSniffer's performance. The following sections explain how these values were adjusted.

| Parameter | Value |
|---|---|
| NEWCONN | 3 |
| R | 4 |

**Table 6-1: System configuration parameters**

### 6.1.1 Capturing Module

The capturing module sniffs traffic sent to and sent from the DDoSniffer machine (the machine on which DDoSniffer is running). The open-source Windows Packet Capture

Library WinPcap [69] is employed for this module. The network traffic is filtered by the WinPcap kernel-level packet filtering feature. In this manner, the traffic that enters the capturing module is TCP traffic. As the capturing module is the core module of DDoSniffer, this module is always running as a background thread. The capturing variables for the WinPcap library are configured in a manner such that every packet (either incoming or outgoing) triggers the capturing procedure.

The built-in hash-table class of the development environment (Visual C++ .NET) is employed to build the new-connection table and the connection table. A hash-table provides fast access to any of its record data provided that the index is chosen effectively. Due to the fact that a given connection is uniquely identified by the pair of IP addresses and ports participating in the connection, these parameters are used to generate the index into each of the tables. One important factor that must be taken into account when generating the table index is the order of the parameters. If these parameters are taken directly from the captured packet, incoming and outgoing packets belonging to the same connection will be saved in different records. This issue is circumvented by always taking the IP address of the DDoSniffer machine as the reference source address.

After the information is saved in the respective tables, the raw data is also saved for further analysis (this is intended for research purposes). Additionally, another capturing application (Ethereal [70]) is used to compare this data with the data provided by DDoSniffer. This comparison can provide insights on situations where DDoSniffer fails to measure all traffic.

### 6.1.2 New-connection Module

Adjusting the value of the NEWCONN parameter is an important aspect of the implementation of this module. As explained in section 5.1.2, there is a trade-off between detection time and the number of false positives. A larger value of NEWCONN delays attack detection, but it also prevents false positives in the case where legitimate packets get lost due to network congestion. Similarly, a shorter value of NEWCONN expedites attack detection, but it also increases the number of false positives (as the adjustment process proved – discussed later on). NEWCONN is an upper limit to the size of the new-connection table in

the sense that only NEWCONN connections will be allowed in the new-connection table for four consecutive observation intervals.

The adjustment process for NEWCONN was as follows. Using the developed attack tool to generate a one-minute long HTTP-GET attack that opens one connection per second, values from one to ten were tested in order to select the optimum value for NEWCONN. For each value, the experiment was repeated five times with a different random seed used in the background traffic generator each time. Two metrics were measured: average detection time, and average number of false positives. Figure 6-2 shows the obtained results. As expected, with the smallest value of NEWCONN, the attack was detected with the shortest average time (4.4 seconds), but also with the largest number of false positives of all (6.8). The optimum value for NEWCONN is the value that has a short number of false positives while detecting attacks in a reasonable time. From Figure 6-2, it can be seen that three is the first value for which the average number of false positives is zero (no false alarm occurred). Consequently, NEWCONN was set to 3 for experimentation.

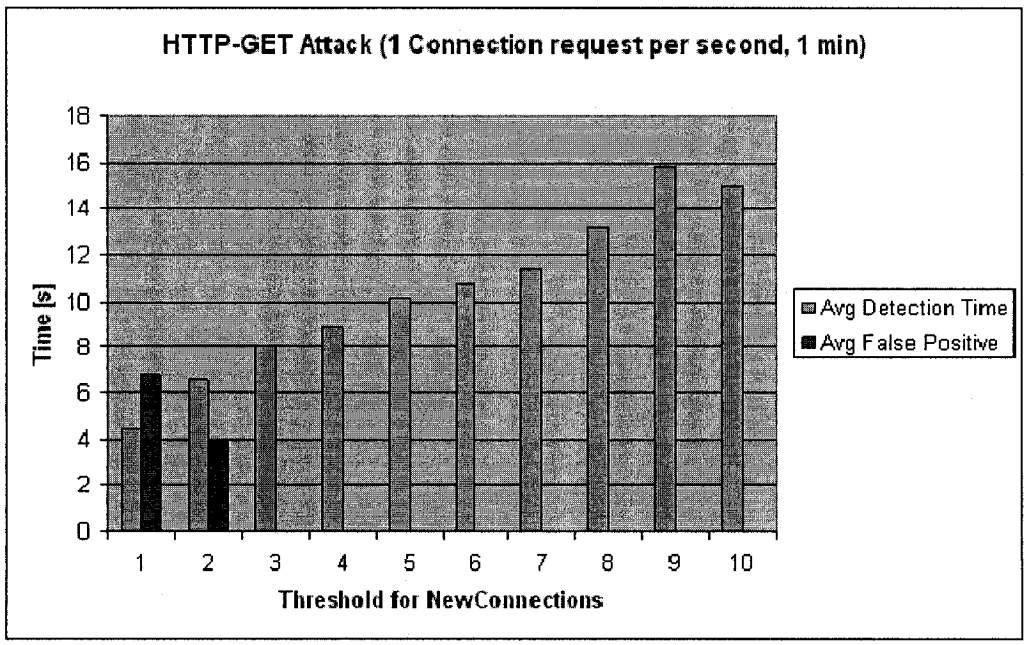


**Figure 6-2: NEWCONN experiments results**

### 6.1.3 Classification Module

The operation of this module (detection of bandwidth attacks) is heavily influenced by the value of the threshold $R$ for the traffic ratio. As explained in section 4.2, the traffic ratio of a connection depends on several factors including the type of traffic. Due to the unknown distribution of the traffic, it is necessary to build legitimate traffic models in order to empirically determine the value of the threshold $R$ for the traffic ratio on which attack detection will be based. Three weeks of traffic data provided by Cistech Limited were examined and traffic patterns were studied on a daily basis. Cistech Limited uses IP Service Engine (IPSE) [71] to keep record of their network activities; we had access to data from three different locations. Figure 6-3 depicts Cistech Limeted's Network Architecture.

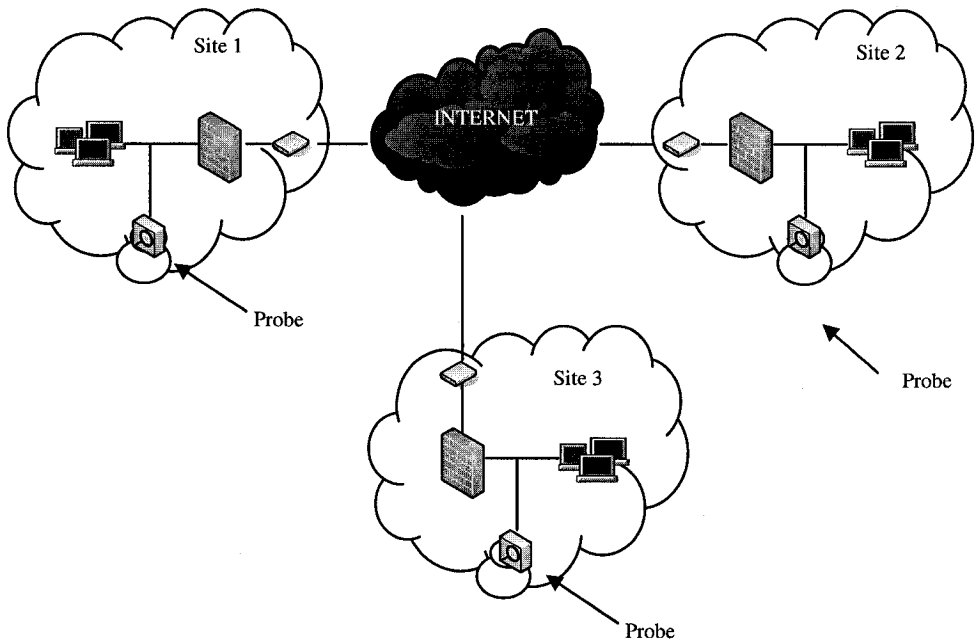


**Figure 6-3: Cistech Limited network architecture**

IPSE is a hardware and software based system that can provide data from the three different locations either separately or all together (due to the three probes shown in Figure 6-3). To acquire a larger and more representative set of data, the information from the three sites in Figure 6-3 were taken all together. Additionally, only HTTP traffic was analyzed

because it was the dominant type of traffic during the period of study. From the obtained data, the traffic ratio for each user was calculated. Table 6-2, Table 6-3, and Table 6-4 present the average traffic ratio as well as the maximum value of the traffic ratio reached each day for Week 1, Week 2, and Week 3, respectively.

| | **Day 1** | **Day 2** | **Day 3** | **Day 4** | **Day 5** |
|---|---|---|---|---|---|
| **Avg. Ratio** | 0.9954 | 1.0065 | 1.0847 | 1.0436 | 1.0845 |
| **Variance** | 0.0883 | 0.1095 | 0.1162 | 0.1267 | 0.1194 |
| **STD** | 0.2972 | 0.3310 | 0.3409 | 0.3560 | 0.3456 |
| **Max Ratio** | 3 | 3.31538 | 2.5 | 2.5 | 2.5 |

**Table 6-2: HTTP traffic during week 1**

| | **Day 1** | **Day 2** | **Day 3** | **Day 4** | **Day 5** |
|---|---|---|---|---|---|
| **Avg. Ratio** | 0.9840 | 0.9503 | 0.9828 | 0.9858 | 1.0014 |
| **Variance** | 0.0906 | 0.0681 | 0.0882 | 0.0922 | 0.0993 |
| **STD** | 0.3011 | 0.2611 | 0.2971 | 0.3038 | 0.3151 |
| **Max Ratio** | 2.4444 | 2.4545 | 3.3366 | 3.3421 | 3.3104 |

**Table 6-3: HTTP traffic during week 2**

| | **Day 1** | **Day 2** | **Day 3** | **Day 4** | **Day 5** |
|---|---|---|---|---|---|
| **Avg. Ratio** | 1.0221 | 0.9656 | 0.9395 | 0.9844 | 0.9424 |
| **Variance** | 0.0996 | 0.0769 | 0.0787 | 0.0854 | 0.0812 |
| **STD** | 0.3156 | 0.2774 | 0.2806 | 0.2923 | 0.2850 |
| **Max Ratio** | 3.0952 | 2.8214 | 2.5714 | 2.6238 | 2.5454 |

**Table 6-4: HTTP traffic during week 3**

Over these three weeks, the average maximum ratio was 2.8240, and the largest value was 3.3366 during the third day of the second week. The threshold $R$ for attack detection was set to 4. Although a smaller value of $R$ could have been chosen, there is at least one case of a legitimate connection with a traffic ratio equal to four. Specifically, if a connection is

established and then terminated with a RST packet immediately after sending a GET packet (or any other request), this connection's traffic ratio is equal to four (outgoing packets: SYN, ACK, GET, RST, incoming packets: SYN-ACK).

This module classifies any connection with a traffic ratio larger than 4 as an attack connection. This traffic threshold deduced from HTTP traffic is used regardless of the application running on top of TCP assuming that other TCP traffic generates similar traffic patterns (in terms of the traffic ratio). Another implicit assumption comes from the data itself, it is assumed that no attack occurred during the period the data was taken.

## 6.2 Background Traffic Tool

This implemented tool simulates legitimate traffic during the experiments. Attack detection is influenced not only by the attack traffic, but also by the amount of legitimate traffic generated by the DDoSniffer machine. DDoSniffer only sees part of the attack traffic because it is a source defence mechanism and attack traffic is distributed among the agent machines participating in the attack. The biggest challenge of DDoSniffer is to detect a low-rate attack in the presence of high-rate legitimate traffic.

Establishing benchmarks for legitimate traffic is one of the many open aspects that testing a DDoS defence mechanism faces. There are no well-known and established standards to be adopted. A commonly used approach is to obtain traces from real networks such as research laboratories at Universities, private companies collaborating with the academic research community, and so forth. The approach followed in this thesis belongs to the second category. The same set of three weeks of data used to determine the threshold traffic ratio $R$ was filtered out to obtain:

- The top ten users during these three weeks – With the average number of total packets per hour for these ten users, a packet rate per minute was estimated (426) and used in the generation of a trace for legitimate traffic. This trace was called TR01.
- The top user of all three weeks – The packet rate per minute (4,200) of this user was employed in the generation of a second trace for legitimate traffic. This trace was named TR02.

Using a customized application that randomly opens connections, two traces were generated (one for each packet rate). Legitimate traffic consisted of HTTP and FTP. The length for each trace was 5 minutes; thus, the first trace had 2,130 legitimate packets, and the second trace had 21,000 legitimate packets. The second trace, TR02, posed a more significant challenge to DDoSniffer due to the higher packet rate. Forty different random seeds were used to generate 40 runs of this trace. Regarding the other trace, TR01, 10 runs were generated by using the same first 10 random seeds employed for TR02.

## 6.3 Attack Tool

This tool generates DDoS attacks to test DDoSniffer under diverse circumstances. As discussed in section 4.1, TCP-based DDoS attacks can be divided into two groups: connection attacks that generate connections with four packets or fewer, and bandwidth attacks that generate connections with traffic ratios larger than $R$.

To evaluate DDoSniffer's performance with regards to connection attacks, it is unnecessary to test several flooding attacks. Instead, by testing with an HTTP-GET type of attack, other flooding attacks are implicitly tested as well. Moreover, detecting HTTP-GET attacks is more demanding for DDoSniffer because these types of attacks generate more packets per connection and this causes more manipulation of both DDoSniffer's tables (the new-connection table and the connection table). As these tables are shared resources, when one thread is accessing the tables, the other threads must wait to gain access to the tables. It is expected that the amount of time required for the new-connection module to detect connection attacks increases as the number of packets arriving at the capturing module increases. The attack tool opens one new connection every second for the duration of the attack, and each connection sends one GET packet over HTTP. In this manner, we are testing the smallest possible rate: 1 connection per second. Table 6-5 summarizes the HTTP-GET attack set up.

| | **Attack rate** | **Attack duration** |
|---|---|---|
| **Attack 1** | 1 connection per second | 1 minute |
| **Attack 2** | 1 connection per second | 10 seconds |

**Table 6-5: HTTP-GET attacks set up**

Regarding bandwidth attacks, the methodology is much simpler: the attack tool opens a TCP connection, and then one packet per second is continuously sent over the same connection for the duration of the attack. As in the case of connection attacks, two different attack lengths were tested: 1 minute, and 10 seconds.

# CHAPTER 7
# PERFORMANCE RESULTS

In order to measure the performance of DDoSniffer, two metrics are significant: the lowest detectable attack rate and the shortest attack length. The number of false negatives is another important performance metric because false negatives diminish the value of real detections. A closely related metric to false negatives is the detection rate, that is, the percentage of attacks that were in fact detected. The system's parameters *NEWCONN* and *R* were set to the values mentioned in sections 6.1.2 and 6.1.3, respectively.

This chapter presents the performance statistics of the detection tool DDoSniffer during experimentation as well as an analysis of the results. Section 7.1 shows DDoSniffer's performance when detecting connection attacks. Section 7.2 explains the results of bandwidth attacks' tests. Section 7.3 presents a comparison of DDoSniffer's results and other related approaches.

## *7.1 Connection Attacks*

As explained in chapter 6, due to the fact that DDoSniffer only sees a portion of the attack traffic, low-rate attacks were our focus of interest. Attacks started 2 minutes 30 seconds after the legitimate traffic tool had started. Recall all experiments lasted 5 minutes (the length of the legitimate traffic traces).

Figure 7-1 depicts the results for TR01 (2,130 legitimate packets) for the case of attack 1. The x-axis shows the run number (recall that for TR01, the experiment was repeated ten times with a different random seed used in each trial), and the y-axis shows the detection time. The average detection time for this combination of attack-trace was 6.7 seconds with a standard deviation of 0.4830, and the false negative rate was 0% which means that the detection rate was 100%. This high percentage of attack detection is attributable to the low-rate of legitimate traffic as well as the duration of the attack.

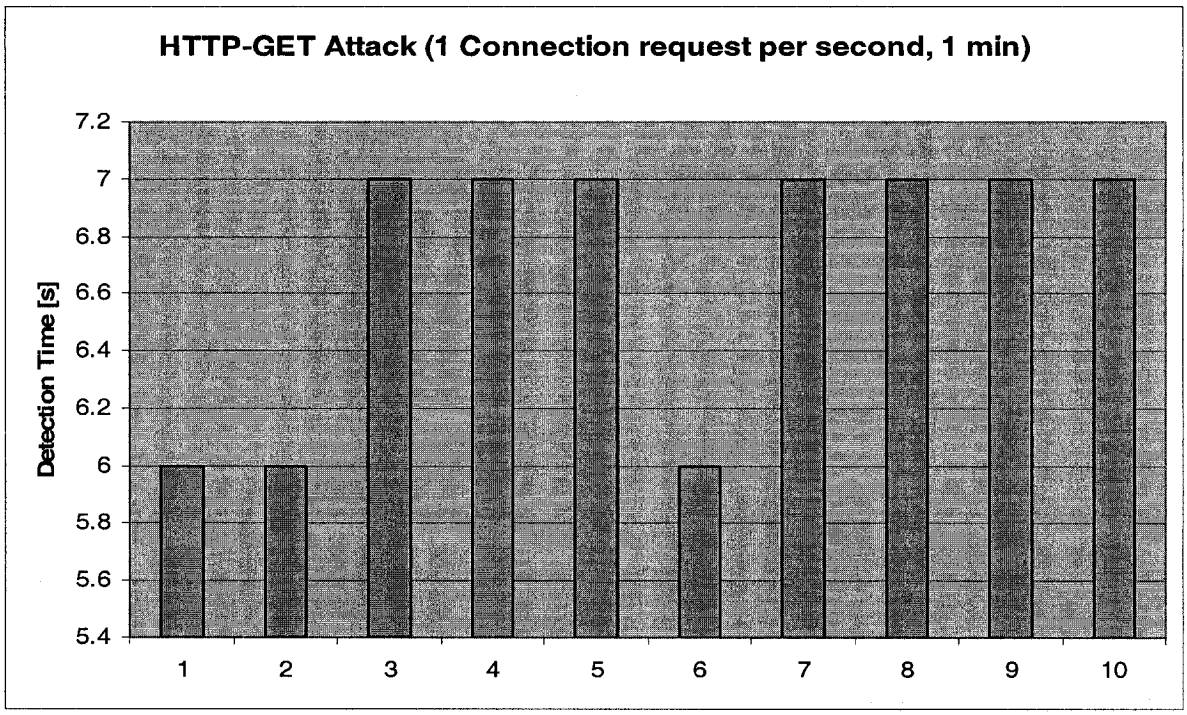


**Figure 7-1: Detection time for each trial (TR01- Attack 1)**

Figure 7-2 presents the results obtained for TR02 (21,000 legitimate packets) and attack 1. In this case the average detection time was 8.05 seconds with a standard deviation of 1.6938. The false negative rate and the detection rate were 0% and 100%, respectively. However, there was an average of 1.42% of packet loss in 35% of the runs. Additionally, 50% of the runs during which packets were lost also experienced misclassification of connections. It should be noted that all the occurrences of packet loss were during the attack. Packet loss in this context means that DDoSniffer was not able to measure all the packets.

As expected, the detection time was longer for TR02 than TR01 because of the higher rate of legitimate traffic (4,200 packets per minute vs. 426 packets per minute respectively). Due to the fact that access to the new-connections table and the connection table is shared among the three modules of DDoSniffer, the detection time increases as the total number of packets increases.

Figure 7-3 shows the results for TR02 and attack 2. The average detection time increased to 8.225 seconds with a standard deviation of 1.7901. The false negative rate was 27% and the detection rate was 73%. However, the percentage of runs experiencing packet loss decreased considerably to 2.5% (from 35%). There was only one run with packet loss.

This one run experienced packet loss of 0.12% (which also occurred during the attack) and no misclassification of connections occurred.

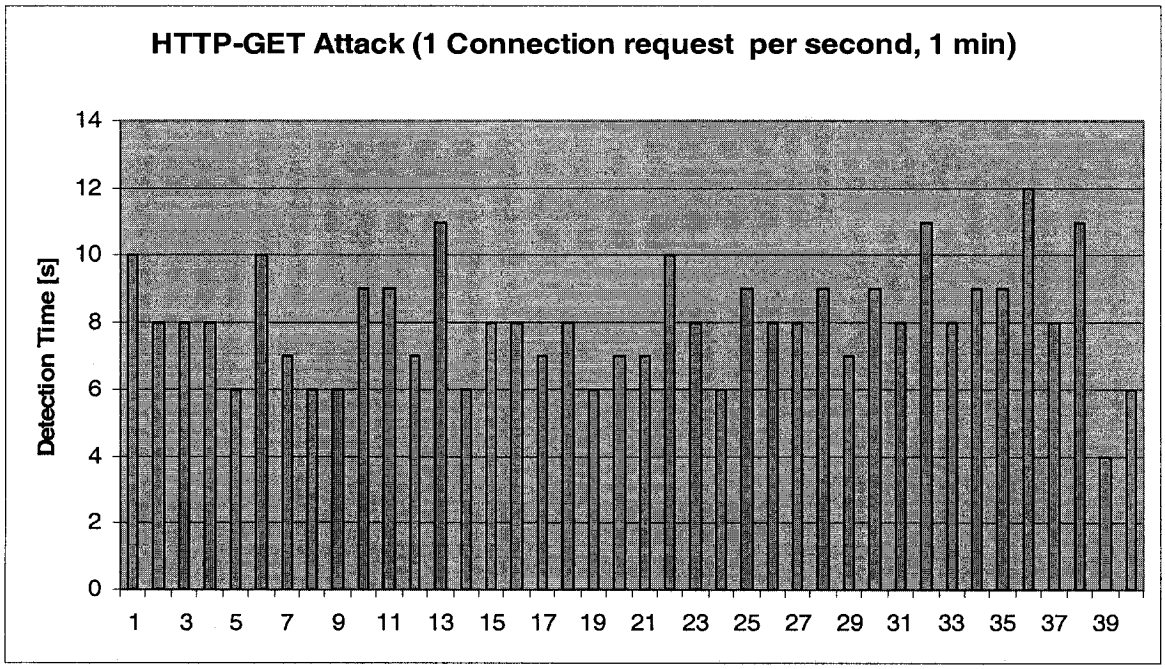


**Figure 7-2: Detection time for each trial (TR02 -Attack 1)**

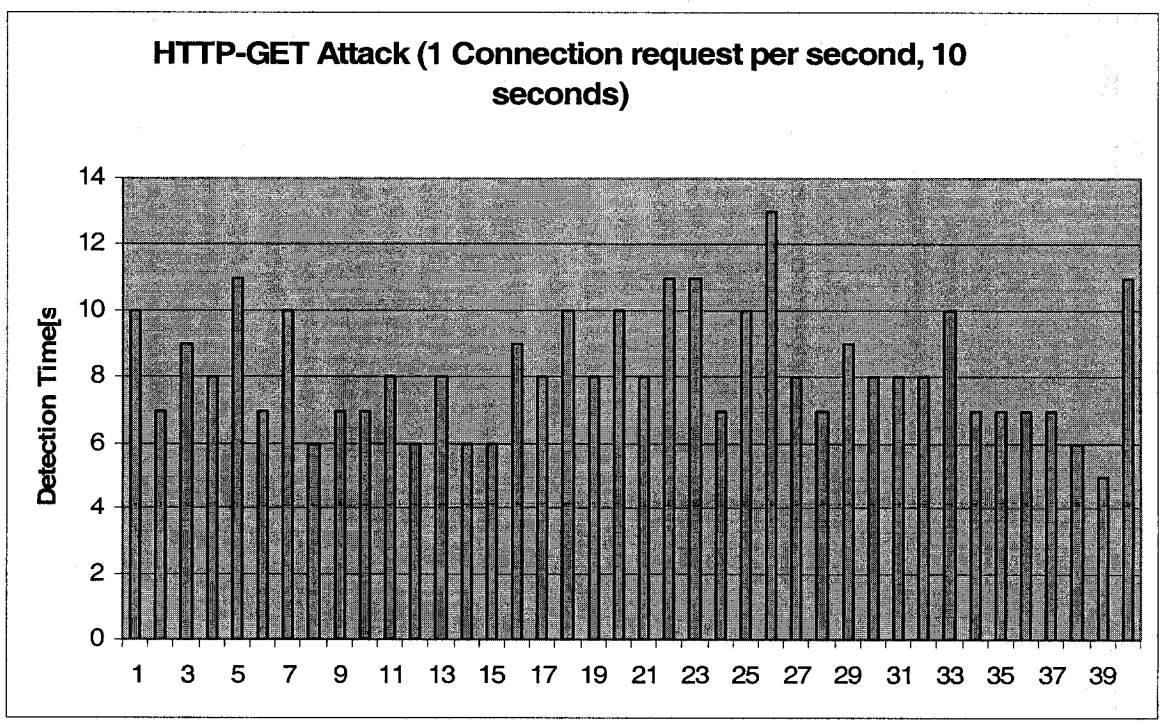


**Figure 7-3: Detection time for each trial (TR02 - Attack 2)**

Although attack 2 was shorter (i.e., fewer packets in total), the detection time was slightly longer than the previous attack. As the percentage of packets loss was considerably lower for attack 2, the total number of packets ended up being larger and this explains the longer detection time.

## 7.2 Bandwidth Attacks

For the case of bandwidth attacks, the attack tool opens one connection and sends one packet per second for the duration of the attack. The attacks started 2 minutes 30 seconds after the background traffic tool had started.

Figure 7-4 depicts the results for TR01 and attack 3. The x-axis represents the run number and the y-axis shows the detection time of the attack. In this situation, the average detection time was 3.5 seconds with a standard deviation of 0.5270. The false negative rate was 0% and the detection rate 100%. As in the case of connection attacks, these results are attributable to the low-rate of legitimate traffic as well as attack duration.

Figure 7-5 shows the results for TR02 and attack 3. The average detection time was 9.8 seconds with a standard deviation of 4.5566. The false negative rate was 0%, and the detection rate was 100%. Nevertheless, 25% of the time, packet loss occurred. The average packet loss was 1.7%. In 50% of the cases where packets were lost, misclassification of connections occurred as well. All packet loss took place during the attack.

As in the case of connection attacks, the detection time for TR02 was longer than TR01. This was expected to be the case due to the higher packet rate of TR02 over TR01. Sharing access to the connection table causes longer detection time for higher packet rates. Figure 7-6 illustrates the obtained results for TR02 and attack 4. The average detection time fell to 7.9 seconds with a standard deviation of 2.0853. The false negative rate rose to 13% and the detection rate was 87%. The packet loss occurrence dropped to 2.5%. In the run which suffered packet loss, only 0.43% of packets were lost and no misclassification occurred.

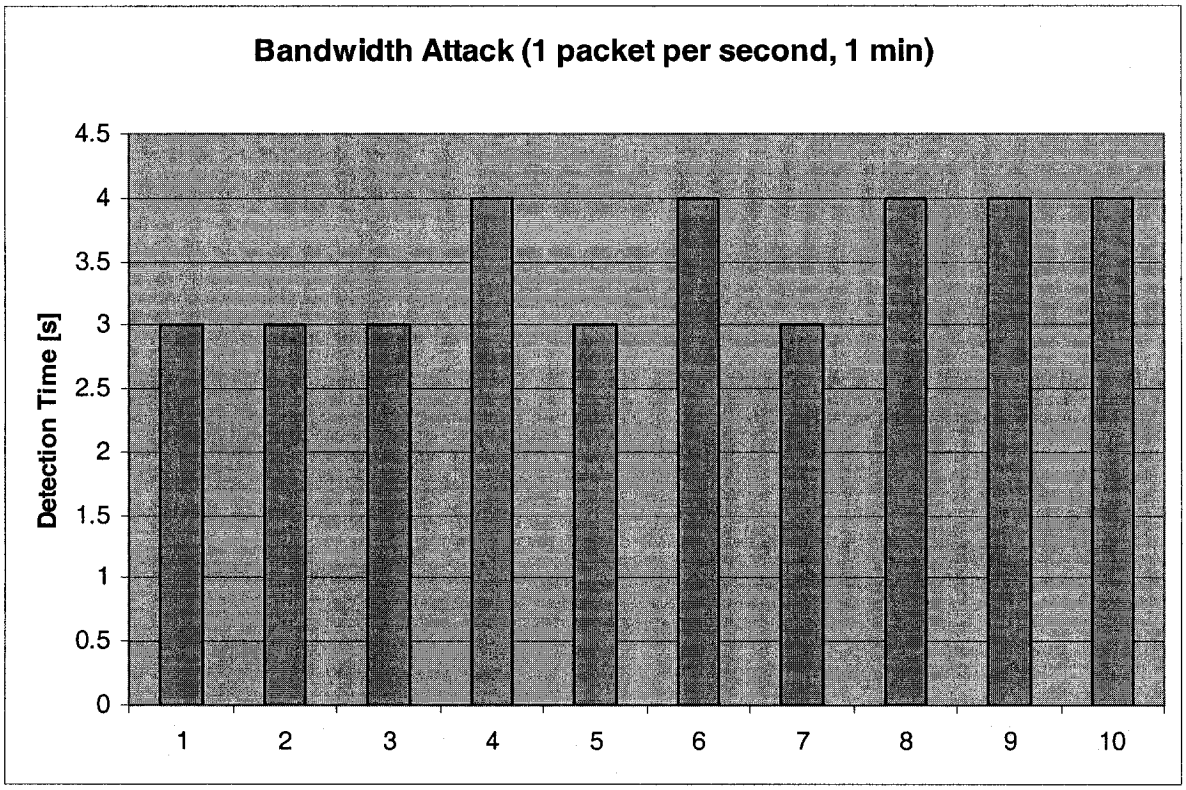


**Figure 7-4: Detection time for each run (TR01 - Attack 3)**

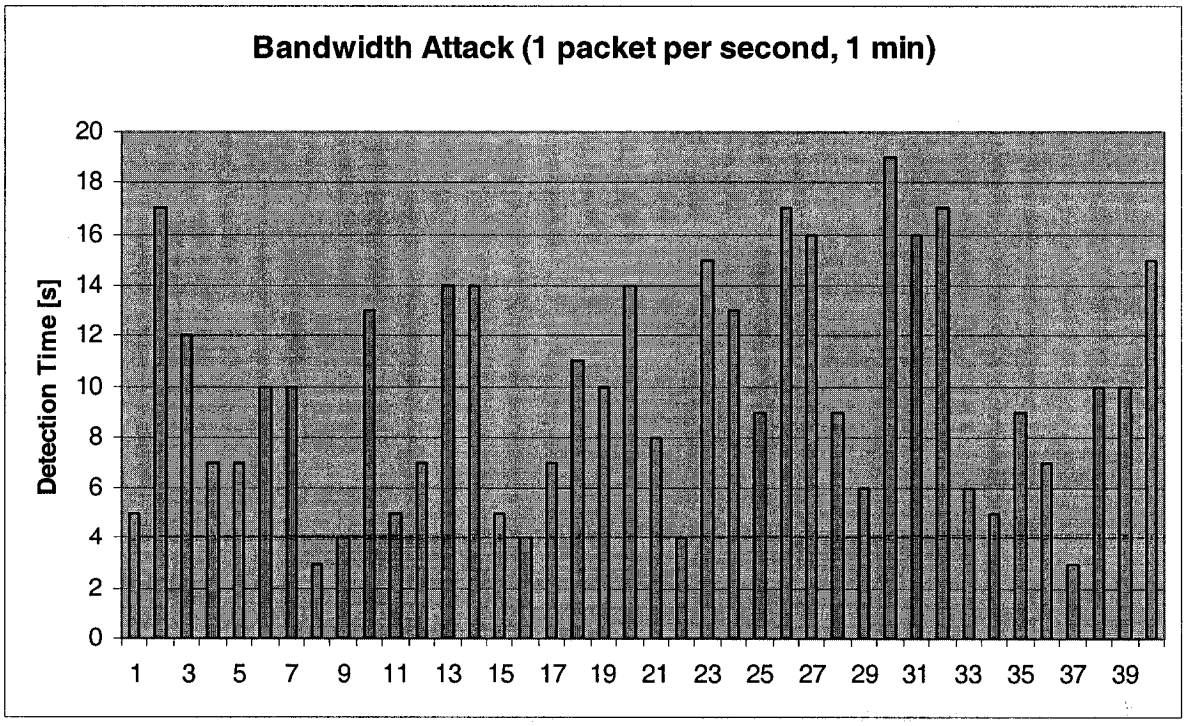


**Figure 7-5: Detection time for each trial (TR02 - Attack 3)**

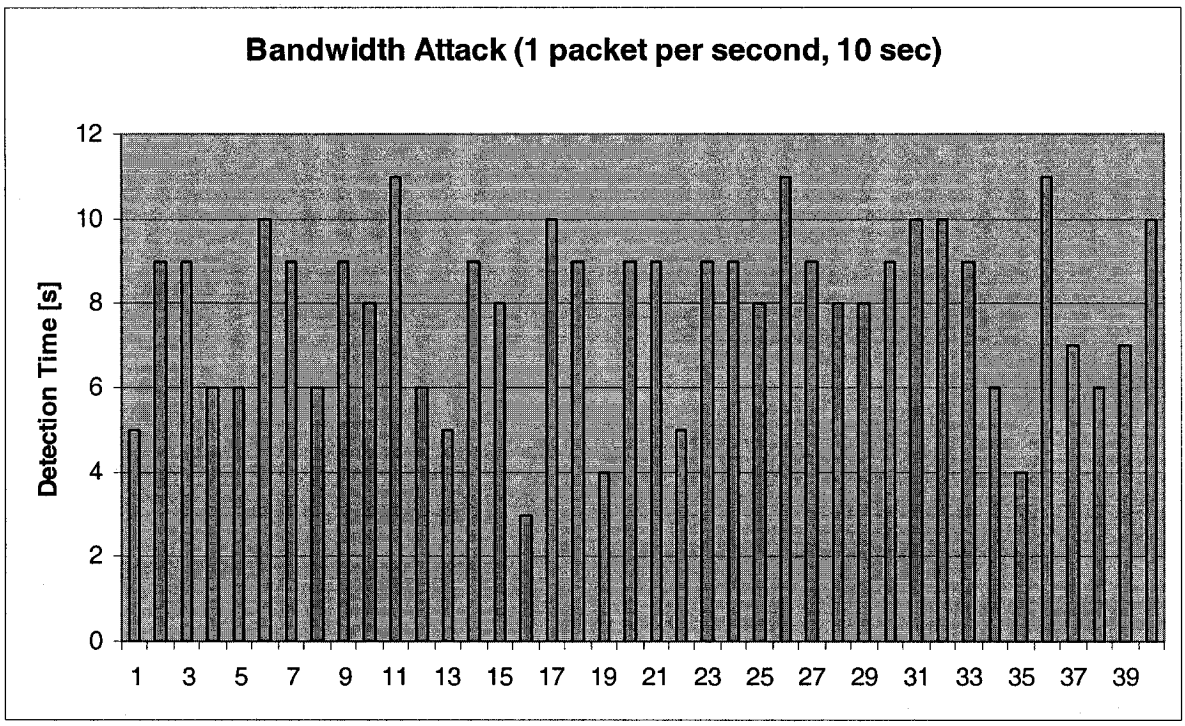


**Figure 7-6: Detection time for each trial (TR02 - Attack 4)**

Experiments showed that DDoSniffer detected TCP-based attacks in 7.36 seconds (average time). Table 7-1 summarizes the experiment results discussed in sections 7.1 and 7.2. Both connection (CONN) and bandwidth (BW) attacks shown in Table 7-1 had a rate of 1 packet per second. Additionally to these attacks, IP spoofing is also detectable by DDoSniffer as explained in section 5.1.1.

| Trace | Attack type | Attack duration | Detection Time [sec] | Detection Rate | Packet loss |
|---|---|---|---|---|---|
| TR01 | CONN | 1 min | 6.7 | 100% | 0% |
| | BW | 1 min | 3.5 | 100% | 0% |
| TR02 | CONN | 10 sec | 8.225 | 73% | 0.12% |
| | CONN | 1 min | 8.05 | 100% | 1.42% |
| | BW | 10 sec | 7.9 | 87% | 0.43% |
| | BW | 1 min | 9.8 | 100% | 1.7% |

**Table 7-1: Results summary**

## 7.3 Limitations

Since higher rate traffic introduces delay in attack detection, an attacker wishing to circumvent DDoSniffer can open a large number of legitimate connections to overwhelm the system. To overcome this limitation, the sizes of the new-connection table and the connection table can be bounded. Once this upper limit is reached, some connections are to be removed from the tables. The criteria for connection removal need to be investigated in the future because it cannot be only based on time.

Another possible method to thwart DDoSniffer detection algorithm is by pulsing attacks. Instead of constantly sending packets, a pulsing attack is active for a given period of time (henceforth referred to as the "on time" $T_{on}$) and inactive for another period of time (henceforth referred to as the "off time" $T_{off}$). During $T_{on}$, the attacker transmits packets, and during $T_{off}$ no action is taken. Frequently, pulsing attacks are used in a synchronized manner such that sets of agent machines are on during the $T_{off}$ period of other sets of agent machines. An attacker seeking to evade DDoSniffer by using a pulsing attack needs to send fewer packets than NEWCONN during $T_{on}$ which has to be less than 7 seconds to avoid detection. This approach is not impossible, but it puts severe time constrains on the attacker. Synchronizing agent machines under such short time periods is a challenge the attacker needs to solve to achieve a successful denial of service attack.

## 7.4 Performance comparison

To gain a deeper understanding of DDoSniffer's achievements, our results are summarized and compared to other approaches in Table 7-2, and Table 7-3. However, it should be emphasized that DDoSniffer supplements existing defence mechanisms rather than competes against them. For example, an ISP can benefit from adopting any of these solutions and providing DDoSniffer as a detection tool for the end-hosts.

| Defence Mechanism | Attack type | Detection time | Attack rate [pps] | Attack duration | Detection rate |
|---|---|---|---|---|---|
| **DDoSniffer** | TCP | 8.225 sec | 1 | 10 sec | 73% |
| | TCP | 8.05 sec | 1 | 60 sec | 100% |
| | TCP BW | 7.9 sec | 1 | 10 sec | 87% |
| | TCP BW | 9.8 sec | 1 | 60 sec | 100% |
| **D-WARD** | TCP | 7 sec | 1 | 100 sec | N/A |

**Table 7-2: Performance comparison**

| Defence Mechanism | Attack type | Detection time | Attack rate [pps] | Attack duration | Detection rate |
|---|---|---|---|---|---|
| **Stub-Domain** | TCP | 10.14 sec | 100 | 8 min | 100% |
| | TCP | 13.35 sec | 50 | 8 min | 100% |
| | TCP | 27.88 sec | 20 | 8 min | 100% |
| | TCP | 106.25 sec | 10 | 8 min | 99% |
| **FDS** | TCP SYN | 40 sec | 80 | 10 min | N/A |
| | TCP SYN | 6 min | 35 | 10 min | N/A |

**Table 7-3: Other approaches' results**

Important aspects to point out:

- Attack rate - DDoSniffer detects the lowest attack rate and only D-WARD equals this achievement. However, it is not clear if other approaches can detect this low rate.
- Attack variety – DDoSniffer not only detects a large variety of TCP-based attacks, but also detects the occurrence of IP spoofing. Similar features are only offered by D-WARD. In the case of FDS, it is not evident if other type of TCP-based attacks will be detected. Moreover, FDS detection is based on the relationship between SYN packets and FIN packets, but any of the TCP flags can be modified by the attacker to avoid detection.

- Detection time – DDoSniffer's average detection time is 7.36 seconds which is slightly longer than D-WARD's detection time. Other solutions tested higher attack rates and still they detection times were longer than DDoSniffer. It is expected that higher attack rates will be detected faster.
- Deployment cost - Additionally, all of the above solutions are implemented at the border of networks which incur hardware costs which DDoSniffer does not.
- Collateral damage - Whether or not collateral damage can be caused by the defence mechanism depends on the actions taken after attack detection. Only DDoSniffer and Stub-Domain passively monitor network traffic, and thus no collateral damage is imposed in case of false alarms.

# CHAPTER 8
# CONCLUSIONS AND FUTURE WORK

The previous chapters have shown that despite the diversity of TCP-based DDoS attacks, it is possible to detect ongoing attacks effectively. Detection is the minimum requirement of all defence mechanisms and DDoSniffer is capable of detecting a broad range of attacks within seconds. Remarkably, the detection algorithms were design in such a manner that attackers must come up with novel strategies under strict time constrains in order to circumvent our tool.

Different scenarios were tested to evaluate the performance of DDoSniffer when detecting what we classified as connection attacks and bandwidth attacks. The former attacks generate connections with four packets or fewer. The latter attacks create connections with traffic ratios larger than usual. The traffic ratio of a connection is given by the number of outgoing packets to the number of incoming packets. DDoSniffer detected 10-second connection attacks (which generated one connection per second) in 8.225 seconds with a detection rate of 73%. For bandwidth attacks (that sent one packet per second) the detection time was 7.9 seconds with an 87% detection rate for 10-second attacks as well.

We have shown that it is feasible to develop a software tool for detecting ongoing DDoS attack traffic at the agent machines. Our research contributions are as follows:

- In-depth analysis of the DDoS problem – We studied the DDoS threat from a general perspective to narrow down the roots of attackers' success: agent machine participation in DDoS attacks.
- Break down of TCP-based DDoS attack strategies – We described in detail the methods employed by attackers to successfully mount a DDoS attack. Although attack strategies are very diverse, this diversity can be classified into two categories: attacks that generate connections with four packets or

fewer, and attacks that generate connections with traffic ratios larger than usual. We called these two types of attacks connection attacks and bandwidth attacks.

- Based on the decomposition of the strategies, we designed and developed a software tool to prove that it is possible to detect attacks in a reasonable short time (7.36 seconds average time).

The distributed and complex nature of DDoS attacks require several defence mechanisms interacting together. An extension to DDoSniffer is to trigger antivirus software and personal firewall installed on the DDoSniffer machine to complement our functionalities. Another extension is to include a component to detect UDP-based attacks.

Additionally, three important factors must be addressed to handle long runs of DDoSniffer without compromising its performance in detecting DDoS attacks. First, the size of both tables (the connection table and the new connection table) must be bounded. As explained in chapter 7, higher traffic rates introduce delay in attack detection due to the size of the tables. An attacker could generate legitimate connection patterns aiming at delaying attack detection long enough to succeed. Second, due to the length of the experiments, it was not necessary to clean-up the tables. However, in the long run, inactive connections must be removed (regardless of the size of the tables). The timestamp field was included in the tables for this reason. In the case of the new-connection tables, any connection that has been longer than 75 seconds should be removed. Seventy-five seconds is the time the TCP specification indicates a half-open connection is kept on the queue waiting to be completed. In the case of the connection table, which connections must be removed is an open research aspect that must be further investigated. Third, the size of the log file must also be limited. Although, originally raw data was recorded to provide insights into DDoSniffer operations (specifically when DDoS fails to capture all traffic), this log file can also be used by an advanced user in order to determine whether or not an alarm was due to a real attack or some other reasons. If the log file is not limited in size, the DDoSniffer machine's storage capacity can be exhausted.

# REFERENCES


[1] C. Duffy Marsan, and C. Garretson, "Net Security Gets Root-Level Boost," *Network World Fusion*, October 2003; http://www.networkworld.com/news/2003/1027ddos.html

[2] BBC NEWS, "Google recovers after virus hits", July 2004; http://news.bbc.co.uk/2/hi/technology/3927963.stm

[3] J. Mirkovic, S. Dietrich, D. Dittrich, and P. Reiher, *Internet Denial of Service: Attack and Defense Mechanisms*, 1st ed., Upper Saddle River, NJ: Prentice Hall, 2004.

[4] Symantec, "Symantec Internet Security Threat Report, Trends for July 05–December 05 Volume IX," March 2006

[5] CERT Coordination Center, "Botnets as a Vehicle for Online Crime," December 2005; http://www.cert.org/archive/pdf/Botnets.pdf

[6] CERT Coordination Center, "Results of the Distributed-Systems Intruder Tools Workshop," December 1999; http://www.cert.org/reports/dsit_workshop.pdf

[7] B. Krebs, "'Sasser' Worm Tip of the PC Bug Invasion," *SecurityFocus*, 12 May 2004; http://www.securityfocus.com/news/8573

[8] Cisco Systems Inc., "Service Provider Security: Mitigating Worm Outbreaks," November 2004; ftp://ftp-eng.cisco.com/cons/isp/security/NANOG-32-PowerSession-Nov-2004/

[9] B. Acohido and J. Swartz, "Unprotected PCs can be hijacked in minutes," *USA Today*, 29 November 2004; http://www.usatoday.com/money/industries/technology/2004-11-29-honeypot_x.htm

[10] Symantec, "Symantec Internet Security Threat Report, Trends for July 04–December 04 Volume VII," March 2005; http://www.symantec.com/region/se/sepress/download/istr_no7.pdf

[11] K. Dunham, "Battling the Bots," *Information Systems Security*, May/June 2005; http://www.infosectoday.com/IT%20Today/bots.pdf

[12] Sophos, "Sophos Security Threat Management Report 2005," 6 December 2005; http://www.sophos.com.au/pressoffice/news/articles/2005/12/toptensummary05.html

[13] J. Mirkovic, "D-WARD: Source-End Defense Against Distributed Denial-of-Service Attacks," doctoral dissertation, Dept. of Computer Sciences, University of California, Los Angeles, 2003.

[14] V. Laurens, A. El Saddik and A. Nayak, "Requirements for Client Puzzles to defeat the Denial of Service and the Distributed Denial of Service attacks," *International A Journal of Information Technology*, pp: 326-333, Vol. 4, No. 3, 2006.

[15] CERT Coordination Center, "TCP SYN Flooding and IP Spoofing Attacks," *Tech Tips*, November 2000.

[16] D. Moore et al, "Inferring Internet Denial-of-Service Activity," in *ACM Transactions on Computer Systems*, v. 24 n. 2, May 2006.

[17] Cisco Systems Inc., "Cisco Security Advisory: Multiple Vulnerabilities in CBOS," 7 August 2001; http://www.cisco.com/en/US/products/products_security_advisory09186a00800b13cb.shtml

[18] Cisco Systems Inc., "Cisco Security Advisory: CBOS Web-based Configuration Utility Vulnerability," August 2001; http://www.cisco.com/en/US/products/products_security_advisory09186a00800b13ce.shtml

[19] S. Gibson, "Distributed Reflection Denial of Service," http://grc.com/files/drdos.pdf

[20] D. Moore and C., "The Spread of the Code-Red worm (CRv2)," *Cooperative Association for Internet Data Analysis (CAIDA)*, 30 July 30 2001; http://www.caida.org/analysis/security/code-red/coderedv2_analysis.xml

[21] Wikipedia, "Mydoom (computer worm)," http://en.wikipedia.org/wiki/Mydoom

[22] Microsoft, "Mydoom, Zindos, and Doomjuice Worm Removal Tool," 8 March 2005; http://support.microsoft.com/?scid=kb;en-us;836528

[23] PCWORLD, "Mydoom: The Trend of Worms to Come?," 20 February 2004; http://www.pcworld.com/news/article/0,aid,114881,00.asp

[24] D. Dittrich, "Analysis of the 'Power' bot," August 2001; http://staff.washington.edu/dittrich/misc/power.analysis.txt

[25] Cisco Systems Inc., "Strategies to Protect Against Distributed Denial of Service (DDoS) Attacks," 2003; http://www.cisco.com/warp/public/707/newsflash.pdf

[26] C. Huegen, "Smurf Attack Information," 08 February 2000; http://www.pentics.net/denial-of-service/white-papers/smurf.cgi

[27] D. Dittrich, "The DoS Project's "trinoo" distributed denial of service attack tool," 21 October 1999; http://staff.washington.edu/dittrich/misc/trinoo.analysis.txt

[28] D. Dittrich, "The "Tribe Flood Network" distributed denial of service attack tool," 21 October 1999; http://staff.washington.edu/dittrich/misc/tfn.analysis.txt

[29] D. Dittrich, "The "stacheldraht" distributed denial of service attack tool," 31 October 1999; http://staff.washington.edu/dittrich/misc/ stacheldraht.analysis.txt

[30] Cisco Systems Inc., "Botnet and DDoS System Mitigation for ISPs," November 2004; ftp://ftp-eng.cisco.com/cons/isp/security/CPN-Summit-2004/Paris-Sept-04/

[31] CERT Coordination Center, "Trends in Denial of Service Attack Technology," October 2001; http://www.cert.org/archive/pdf/DoS_trends.pdf

[32] National Infrastructure Security Co-ordination Centre, "Botnets – the threat to the Critical National Infrastructure," 17 October 2005; http://www.niscc.gov.uk/niscc/docs/ botnet_11a.pdf

[33] The Honeynet Project & Research Alliance, "Know your Enemy: Tracking Botnets," 13 March 2005; http://www.honeynet.org/papers/bots/

[34] L. McLaughlin, "Bot Software Spreads, Causes New Worries," *IEEE Distributed Systems Online*, v. 5, n. 6, June 2004; http://csdl2.computer.org/comp/mags/ds/2004/06/o6001.pdf

[35] J. Mirkovic, and P. Reiher, "A Taxonomy of DDoS Attack and DDoS Defense Mechanisms," in *ACM Special Interest Group on Data Communications (SIGCOMM) Computer Communications Review*, v.34 n.2, April 2004

[36] Windows XP SP2: Nmap Fix and Further Information, *Nmap Hackers*, 26 August 2004; http://seclists.org/lists/nmap-hackers/2004/Jul-Sep/0003.html

[37] B. Krebs, "How to Remove the 'Sasser' Worm," *Washington Post*, 11 May 2004; http://www.washingtonpost.com/wp-dyn/articles/A62330-2004May3.html

[38] Dittrich, D., "DDoS: A look back from 2003," *I2 DDoS Workshop*, August 2003; http://staff.washington.edu/dittrich/talks/I2-ddos.ppt

[39] C. Douligeris, and A. Mitrokotsa, "DDoS attacks and defense mechanisms: classification and state-of-the-art," *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 44, (5) pp. 643 – 666, 2004.

[40] T. M. Gil, and M. Poletto, "MULTOPS: A Data-Structure for bandwidth attack detection," *Proceeding of 10th Usenix Security Symposium*, August 2001, pp 23-38.

[41] C. Kommareddy, "Detecting DDoS Attacks in Stub-Domain," doctoral dissertation, University of Maryland, 2006.

[42] Wikipedia, "Autonomous system (Internet)," http://en.wikipedia.org/wiki/ Autonomous_system_(Internet)

[43] Webopedia, "Stub network", http://www.webopedia.com/TERM/S/stub_network.html

[44] H. Wang, D. Zhang, and K. G. Shin, "Detecting SYN flooding attacks," *Proceedings of the IEEE INFOCOM'02*, 2002.

[45] S. Jin, and D. Yeung, "A covariance analysis model for DDoS attack detection", *IEEE International Conference on Communications* (ICC'2004), Paris, France, 20-24 June 2004.

[46] Cs3, "The Reverse Firewall: Defeating DDoS Attacks," 2003; http://www.cs3-inc.com/rfw.html

[47] Zone Labs, "ZoneAlarm," 2005; http://www.zonelabs.com/store/content/home.jsp

[48] Sygate Technologies Inc., "Award Winning Personal Firewall Provider," 2005; http://smb.sygate.com/

[49] Agnitum Security, "Outpost Personal Firewall," 2005; http://www.agnitum.com/products/outpost/

[50] Kerio Technologies Inc., "Kerio Personal Firewall 4," 30 March 2005; http://www.kerio.com/us/kpf_download.html

[51] Omniquad, "Omniquad Personal Firewall," 2003; http://www.omniquad.com/tshnew.htm

[52] SmartLine Inc., "Active Ports," 4 July 2002; http://www.protect-me.com/freeware.html

[53] Blue's Port Scanner, 2005; http://www.bluebitter.de/portscn2.htm

[54] D. Petri, "Quickly Find Local Open Ports," 2005; http://www.petri.co.il/quickly_find_local_open_ports_gui.htm

[55] Information Sciences Institute, *Transmission Control Protocol*, IETF RFC793, September 1981; http://www.ietf.org/rfc/rfc0793.txt

[56] W.R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*, 1st ed., Addison Wesley, 1994.

[57] P. Ferguson, *Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing*, IETF RFC2827, May 2000; http://www.ietf.org/rfc/rfc2827.txt

[58] P. Karn, *Photuris: Session-Key Management Protocol*, IETF RFC2522, March 1999; http://www.ietf.org/rfc/rfc2522.txt

[59] D. Maughan, M. Schertler, M. Schneider, and J. Turner, *Internet Security Association and Key Management Protocol (ISAKMP)*, IETF RFC2408, November 1998; http://www.ietf.org/rfc/rfc2408.txt

[60] H. Orman, *The OAKLEY Key Determination Protocol*, IETF RFC2412, November 1998; http://www.ietf.org/rfc/rfc2412.txt

[61] Cisco Systems Inc., "TCP Intercept," http://www.cisco.com/univercd/cc/td/doc/product/software/ios112/intercpt.pdf

[62] Cisco Systems Inc., "Advances in SYN-FLOOD Mitigation," November 2004; ftp://ftp-eng.cisco.com/cons/isp/security/NANOG/NANOG-32-PowerSession-Nov-2004/

[63] Sun Microsystems, "Solaris Hosts are Vulnerable to a Denial of Service Induced by an Internet Transmission Control Protocol (TCP) 'ACK Storm'," http://sunsolve.sun.com/search/document.do?assetkey=1-26-102206-1

[64] Cisco Systems Inc., "Cisco Security Advisory: TCP Vulnerabilities in Multiple IOS-Based Cisco Products," 20 April 2004; http://www.cisco.com/warp/public/707/cisco-sa-20040420-tcp-ios.shtml

[65] Cisco Systems Inc., "Cisco Security Advisory: Cisco CatOS Telnet, HTTP and SSH Vulnerability," 9 June 2004;http://www.cisco.com/warp/public/ 707/cisco-sa-20040609-catos.shtml

[66] J. Nagle, *Congestion Control in IP/TCP Internetworks*, IETF RFC896, January 1984; http://www.ietf.org/rfc/rfc0896.txt

[67] V. Jacobson, R. Braden, and D. Borman, *TCP Extensions for High Performance*, IETF RFC1323, May 1992; http://www.ietf.org/rfc/rfc1323.txt

[68] http://www.cistech.ca/

[69] http://www.winpcap.org

[70] http://www.ethereal.com/

[71] http://www.nakinasystems.com/