A NOVEL MITIGATION AND TRACEBACK SCHEME FOR DDoS ATTACK

A DISSERTATION

Submitted in partial fulfillment of the requirements for the award of the degree of

MASTER OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

By

HARSH

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE - 247 667 (INDIA)

JUNE, 2009

Candidate's Declaration

I hereby declare that the work, which is being presented in the dissertation entitled "A **NOVEL MITIGATION AND TRACEBACK SCHEME FOR DDoS ATTACK"** towards the partial fulfillment of the requirement for the award of the degree of **Master of Technology** in **Computer Science and Engineering** submitted in the Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee (India) is an authentic record of my own work carried out during the period from July 2007 to June 2009, under the guidance of **Dr. R. C. Joshi, Professor, Department of Electronics and Computer Engineering, IIT Roorkee.**

I have not submitted the matter embodied in this dissertation for the award of any other degree or diploma.

Date: 10 June, 09

Place: Roorkee

Harsh

(HARSH)

Certificate

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

Date: 10·6·09

Place: Roorkee

[illegible]

(Dr. R. C. JOSHI)

Professor

Department of Electronics and Computer Engineering

IIT Roorkee – 247 667

Acknowledgements

I would like to take this opportunity to extend my heartfelt gratitude to my guide and mentor **Dr. R. C. Joshi**, Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, for his trust in my work, his able guidance, regular source of encouragement and assistance throughout this dissertation work. I would state that the dissertation work would not have been in the present shape without his inspirational support and I consider myself fortunate to have done my dissertation under him.

I also wish to thank Ms. Anjali Sardana for her valuable suggestions and timely help.

Finally, I would like to say that I am indebted to my parents and my husband for everything that they have done for me. All of this would have been impossible without their constant support.

Harsh

HARSH

Abstract

Distributed Denial of Service (DDoS) attacks pose a severe security threat to the steady functioning of any network. These attacks aim at depleting the resources of a server or an administrative network by overwhelming it with enormous and useless traffic. The outcome of this is the fact that legitimate users are denied service.

Though a large number of schemes have been proposed and implemented for defense against the DDoS attacks, an end-to-end approach for the same is still missing. Great amount of research has been carried out in the areas of the detection of the presence of these attacks, differentiating the legitimate flows from the attack ones, tracing the identity of the attackers and fortifying the server in order to minimize the impact of the attack. But there is still a paucity of effective frameworks that encompass multiple stages of the process of defense against DoS attacks.

In this work "A NOVEL MITIGATION AND TRACEBACK SCHEME FOR DDoS ATTACK", a novel solution is proposed which deals with mitigating the influence of the attack, and identification of the path traversed by the flow once it has been characterized as an attack flow.

In the proposed strategy, the packets are marked a hash of the router in there identification field. The TTL field is used to calculate the distance from the marked router to the Victim. These packets are subjected to characterization module and if attack is diagnosed the modified pushback as well as the traceback to the attacker starts. Pushback gives the immediate relief to the Victim as well as gives time to trace the attacker. The effectiveness of the approach is validated with simulation in ns-2 on a Linux platform.

Contents

CANDIDATE'S DECLARATION .... **i**

ACKNOWLEDGEMENTS .... **ii**

ABSTRACT .... **iii**

TABLE OF CONTENTS .... **iv**

CHAPTER 1: INTRODUCTION AND STATEMENT OF PROBLEM **1**

1.1 Introduction 1

1.2 Motivation 2

1.3 Statement of the Problem 4

1.3 Organization of the Dissertation 4

CHAPTER 2: DISTRIBUTED DENIAL OF SERVICE ATTACK **5**

2.1 Attacker Goals 6

2.2 Modus Operandi 6

2.3 Commonly Observed Attacks 8

2.4 Distributed Denial of Service Defenses 11

2.5 Defense Challenges 11

2.5.1 Technical Challenges 11

2.5.2 Social Challenges 13

2.6 Defense Goals 13

2.7 Defense Approaches 14

2.8 Related Work and Research Gaps 15

2.8.1 Related Work 15

2.8.2 Research Gaps 19

CHAPTER 3: DEFENSE STRATEGY AGAINST DDoS ATTACKS **21**

3.1 Assumptions 21

3.2 Proposed Solution 21

CHAPTER 4: CHARACTERIZATION, MITIGATION AND TRACEBACK.... 25

4.1 Characterization.. 25

4.1.1 Packet Process.. 26

4.1.2 Algorithm Used... 27

4.2 Mitigation and Traceback.. 28

4.2.1 Packet Marking.. 29

4.2.2 Calculating Distance.. 30

4.2.3 Pushback.. 31

4.2.4 Map Reconstruction.. 32

4.2.5 Path Reconstruction.. 32

CHAPTER 5: SYSTEM DESIGN AND IMPLEMENTATION............................ 34

5.1 System Design... 34

5.1.1 Network Simulator... 34

5.1.2 System Components... 34

5.1.3 Simulation Model... 35

5.2 Implementation.. 36

5.2.1 Simulation Topology... 36

5.2.2 Procedures.. 36

5.2.2.1 Characterization.. 36

5.2.2.2 Mitigation and Traceback.. 37

5.2.3 Simulation Parameters... 37

5.2.4 Performance Evaluation Metrics....................................... 38

CHAPTER 6: RESULTS AND DISCUSSION... 39

6.1 Results for Characterization... 39

6.1.1 Sample Output... 41

6.2 Mitigation and Traceback.. 43

CHAPTER 7: CONCLUSIONS AND FUTURE WORK..................................... 47

7.1 Conclusions.. 47

7.2 Suggestions for Future Work.. 48

REFERENCES **49**

LIST OF PUBLICATIONS **54**

APPENDIX: SOURCE CODE LISTING **55**

Chapter 1

Introduction and Statement of Problem

1.1 Introduction

Distributed denial-of-service attacks (DDoS) attacks consist of an overwhelming quantity of packets being sent from multiple attack sites to a victim site. These packets arrive in such a high quantity that some key resource at the victim (bandwidth, buffers, and CPU time to compute responses) is quickly exhausted. The victim either crashes or spends so much time handling the attack traffic that it cannot attend to its real work. Thus legitimate clients are deprived of the victim's service for as long as the attack lasts.

Distributed denial-of-service attacks are widely regarded as a major threat to the Internet. They have adversely affected service to individual machines, major Internet commerce sites, and even core Internet infrastructure services. Occasionally, a very large-scale DDoS attack occurs (usually as the by-product of a virus or worm spread), crippling Internet-wide communications for hours. While services are restored as soon as the attack subsides, the incidents still create a significant disturbance to the users and costs victim sites millions of dollars in lost revenue. Furthermore, the Internet is used daily for important communications such as stock trades, financial management and even some infrastructure services. Many of these transactions must be processed in a timely manner and can be seriously delayed by the onset of a DDoS attack. The seriousness of the threat is further increased by the ease of how these attacks are performed. Any unsophisticated user can easily locate and download DDoS tools and engage them to perform successful, large-scale attacks. The attacker runs almost no risk of being caught. All of these characteristics have contributed to a widespread incidence of DDoS attacks. The first large-scale appearance of distributed denial-of-service (DDoS) attacks occurred in mid-1999. Today, Almost Ten years later, researchers are still struggling to devise an effective solution to the DDoS problem. The damage done by DDoS attacks is increasing day by day as the records shown in Figure 1.1 [1]. The biggest damage was done by virus and was estimated as approximately 5.5 crore dollars. DDoS was the cause for the second largest damage in 2004 causing a loss of more then 2 crore dollars and the condition are still not improved.

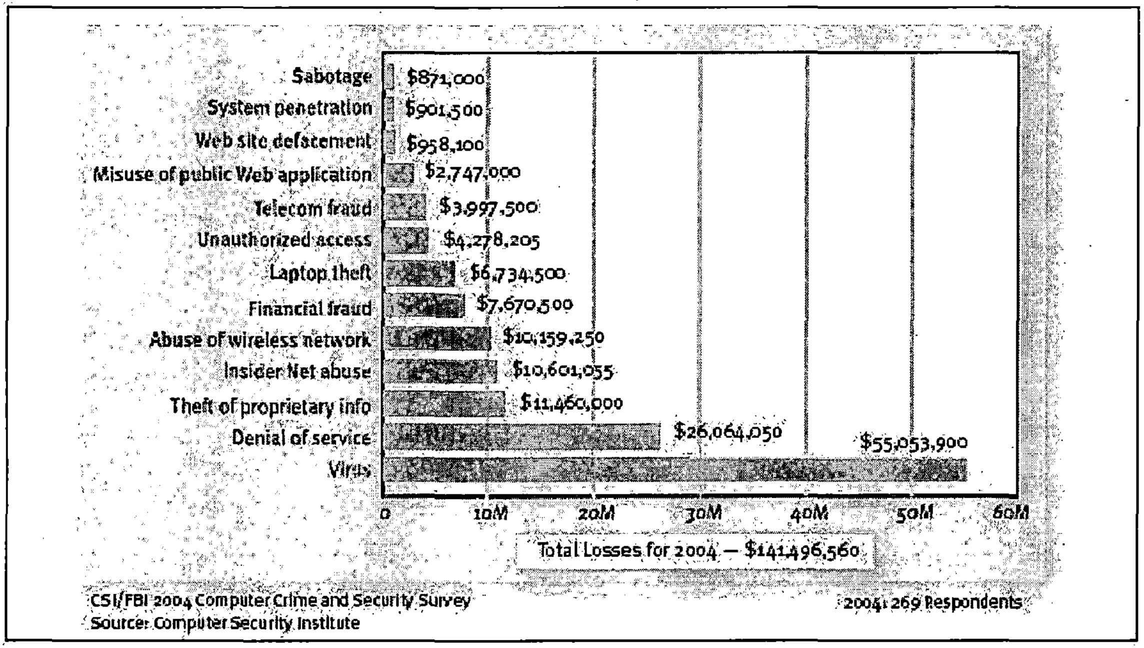


Figure 1.1 Dollar Amount of Losses by Type

1.2 Motivation

Regardless of the diligence, effort, and resources spent securing against intrusion, Internet connected systems face a consistent and real threat from DoS attacks because of two fundamental characteristics of the Internet.

- The Internet is comprised of limited and consumable resources.

The infrastructure of interconnected systems and networks comprising the Internet is entirely composed of limited resources. Bandwidth, processing power, and storage capacities are all common targets for DoS attacks designed to consume enough of a target's available resources to cause some level of service disruption. An abundance of well-engineered resources may raise the bar on the degree an attack must reach to be effective, but today's attack methods and tools place even the most abundant resources in range for disruption.

- Internet security is highly interdependent.

DoS attacks are commonly launched from one or more points on the Internet that are external to the victim's own system or network. In many cases, the launch point consists of one or more systems that have been subverted by an intruder via a security-related compromise rather than from the intruder's own system or systems. As such, intrusion defense not only helps to protect Internet assets and the mission they support, but it also helps prevent the use of assets to attack other Internet-connected networks and systems. Likewise, regardless of how well defended your assets may be, your susceptibility to many types of attacks, particularly DoS attacks, depends on the state of security on the rest of the global Internet.

There are four different ways to defend against DoS attacks [2]:

(1) Attack prevention aims to fix security holes, such as insecure protocols,
(2) Attack detection aims to detect DoS attacks in the process of an attack,
(3) Attack source identification aims to locate the attack sources,
(4) Attack reaction aims to eliminate or curtail the effects of an attack.

The number and assortment of both the attacks as well as the defense mechanisms is monstrous. Though an array of schemes has been proposed for the detection of the presence of these attacks, characterization of the flows as normal or malicious, identifying the source(s) of the attacks and mitigating the effects of the attacks once they have been detected, there is still a dearth of complete frameworks that encompass multiple stages of the process of defense against DDoS attacks. These observations have motivated the need of a solution against the DDoS attacks which should be efficient, scalable and easy to implement.

1.3 Statement of the Problem

To propose a solution for defending DDoS attacks and encompasses the following activities in defense against DDoS attacks:

(i) *Mitigation* of the effect of the attack on the victim node or network,
(ii) Accurate *traceback* of the source(s) of the attack flows.

1.4 Organization of Dissertation

This report comprises of seven chapters including this chapter that introduces the topic and states the problem. The rest of the dissertation report is organized as follows.

Chapter 2 gives an overview of the DoS and DDoS attacks and gives a brief view of the challenges faced in defending these attacks. Also, tools used for realizing DDoS attacks are discussed in brief. It discusses the related work and research gaps in the various phases of the defense against DDoS attacks.

Chapter 3 gives an overall structure of the proposed framework. It gives a big picture of the solution and leaves the details for the subsequent chapters.

Chapter 4 explains in detail the characterization work of the proposed scheme. It also describes the mechanism to achieve the dual functionality of mitigation and traceback.

Chapter 5 describes the system design that includes the system components and the simulation model. The implementation details are also charted out in terms of the topology used for simulation purposes, procedures and simulation parameters.

Chapter 6 discusses the simulation results and displays the effectiveness of the proposed mechanism for defense against DDoS attacks.

Chapter 7 concludes the work and gives the directions for future work.

Chapter 2
Distributed Denial of Service Attacks

In general, a denial of service (DoS) attack is any attack which makes an on-line service (e.g. Web Service) unavailable. The attack could involve a single packet (e.g. the "land" attack [3]) exploiting software bugs in a server, or a traffic stream with a tremendous number of packets that congest the target's server or network. We define a bandwidth attack as any attack that consumes a target's resources through a massive traffic volume. In this dissertation, we focus on bandwidth attacks, and henceforth we mean bandwidth attack when we refer to denial of service attacks unless otherwise stated. The distributed denial of service (DDoS) attack is a bandwidth attack whose attack traffic comes from multiple sources. To launch a DDoS attack, an attacker usually compromises many insecure computers connected to the Internet first. Then a DDoS attack is launched from these compromised computers. The reflector attack is an attack where innocent third-parties (reflectors) are used to bounce attack traffic from the attacker to the target. A reflector can be any network device that responds to any incoming packet, for example, a web server. The attacker can make the attack traffic highly distributed by using many reflectors. The reflector attack is a type of DDoS attack. To summarize, the relations between different types of attacks are illustrated in Figure 2.1

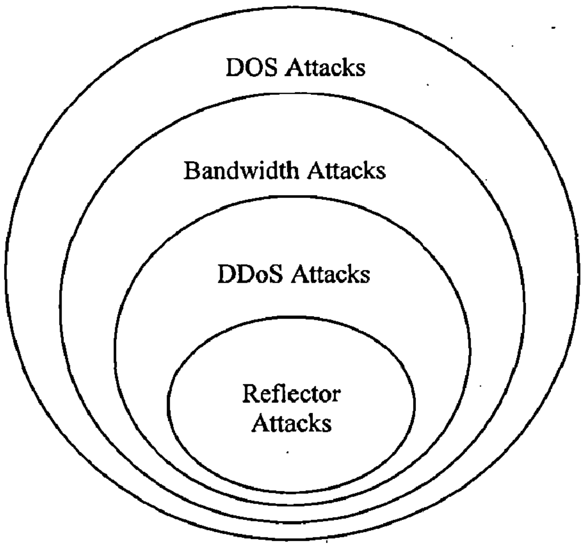


Figure 2.1: The relation of different types of attacks

2.1 Attacker Goals

The goal of a DDoS attack is to inflict damage on the victim. Frequently the ulterior motives are personal reasons (a significant number of DDoS attacks are perpetrated presumably for purposes of revenge), or prestige (successful attacks on popular Web servers gain the respect of the hacker community). However, it is not unlikely that some DDoS attacks are performed for material gain (damaging competitor's resources, such as the recent case of Linux fans attacking SCO [4] because of its lawsuit against IBM) or for political reasons (a country at war could perpetrate attacks against its enemy's critical resources, potentially enlisting a significant portion of the entire country's computing power for this action). In some cases, the true victim of the attack might not be the actual target of the attack packets, but others who rely on the target's correct operation. For example, in September 2002 there was an onset of attacks that overloaded the Internet infrastructure rather than targeting specific victims [5].

It also frequently happens that a DDoS attack is perpetrated accidentally, as a by-product of another malicious activity, such as worm spread [6]. Inefficient worm-spreading strategies create massive traffic that congests the Internet and creates a denial-of-service effect to numerous clients. While ordinary home users are less likely to become victims of DDoS attacks than large corporate networks, no one is free from the DDoS threat. The next attack may target AOL servers, denying service to many home users, or the next worm may congest the Internet so severely that no one can receive service. DDoS is an Internet-wide problem and all parties should cooperate to find a suitable solution.

2.2 Modus Operandi

A distributed denial-of-service is carried out in several phases. The attacker first recruits multiple agent (*slave*) machines. This process is usually performed automatically: the attacker downloads a scanning tool and deploys it from other compromised machines under its command (*masters/handlers*). The tool scans remote machines, probing for security holes that will enable subversion. Vulnerable machines are then exploited—broken into using the discovered vulnerability. They are subsequently infected with the attack code. The exploit/infect phase is also automated, and the infected machines can be used for further recruitment of new agents. Attackers attempt to cover the fact that agent machines have been compromised. They erase all logs showing malicious activity to destroy evidence that could incriminate them. They also hide attack scripts under system directories and give them obscure, non-suspicious names so they will not attract a user's attention and be erased. Sometimes they patch the vulnerability used for the exploit, to prevent other hackers from taking over the machine. Current exploit/infection scripts contain automated tools for covering tracks, so even inexperienced attackers do not leave much evidence of the subversion.

During a DDoS attack, agent machines are engaged to send the attack packets to the victim. The attacker orchestrates the onset of the attack, and scenario details such as the desired type and duration and the target address from the master to the agent machines. Agent machines usually fire out the packets at a maximum possible rate to increase the attack's chances of success. However, there have been attacks where agents were generating packets at a small rate (to prevent agent discovery) or where agent machines were periodically pausing the attack to avoid detection (*pulsing attacks*). Attackers usually hide the identity of subverted machines during the attack through spoofing of the source address field in attack packets. Note, however, that spoofing is not always required for a successful DDoS attack. With the exception of reflector attacks that use spoofing as an attack tool, all other attack types' use spoofing only to hinder detection and discovery of agent machines.

Figure 2.2 shows the architecture of the DDoS Attacks.

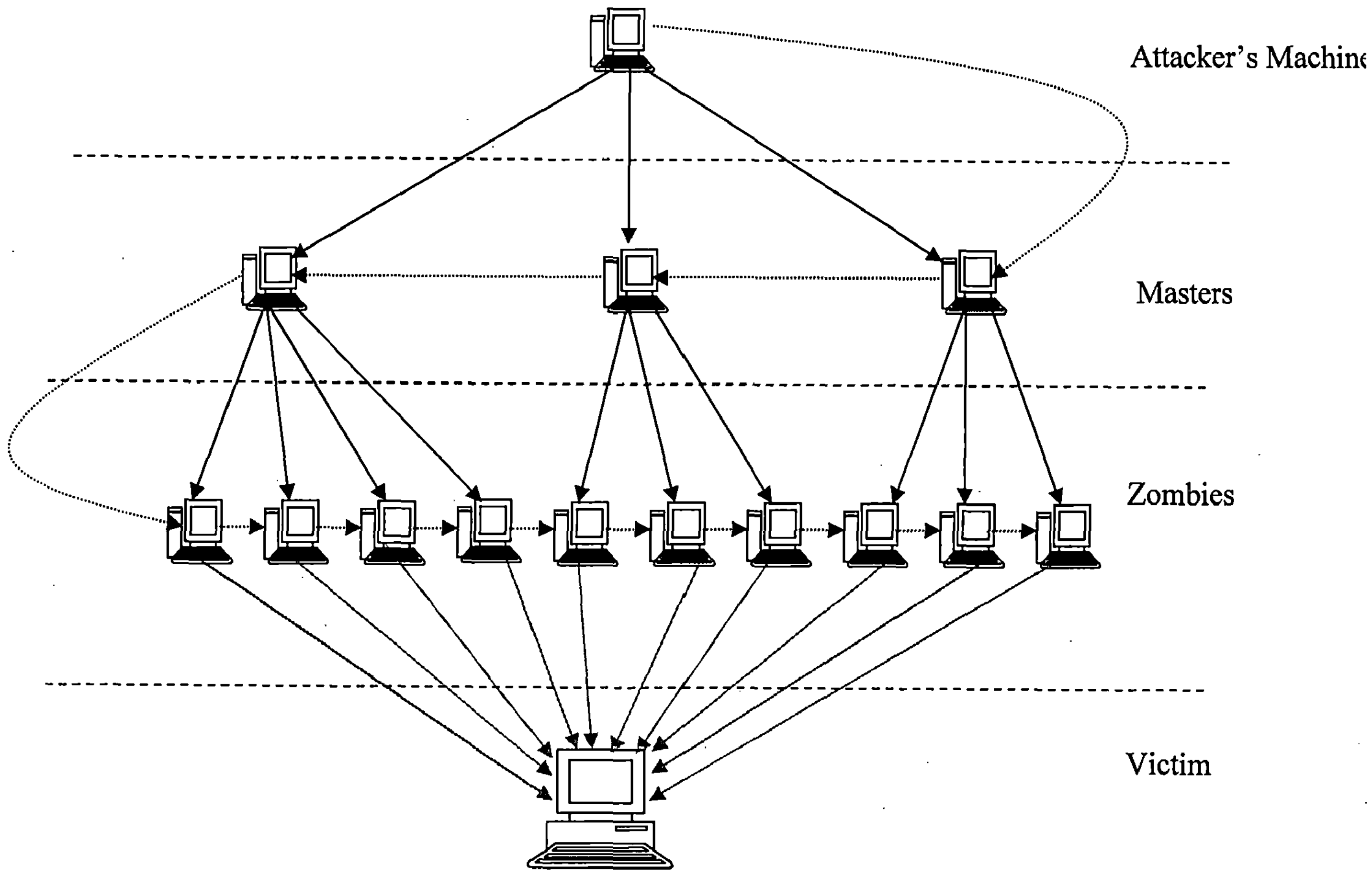


Figure 2.2: Architecture of DDoS Attacks

2.3 Commonly Used Attack Tools

While there are numerous scripts that are used for scanning, compromise and infection of vulnerable machines, there are only a handful of DDoS attack tools that have been used to carry out the engagement phase. A detailed overview of these tools, along with a timeline of their appearance, is given in [7]. DDoS attack tools mostly differ in the communication mechanism deployed between masters and slaves, and in the customizations they provide for attack traffic generation. The following paragraphs provide a brief overview of these popular tools.

Trinoo [8] deploys a master/slave architecture, where an attacker sends commands to the master via TCP and masters and slaves communicate via UDP. Both master and slaves are password protected to prevent them from being taken over by another attacker. Trinoo generates UDP packets of a given size to random ports on one or multiple target addresses, during a specified attack interval.

Tribe Flood Network (TFN) [9] also deploys master/slave architecture. Agents can wage a UDP flood, TCP SYN flood, ICMP ECHO flood and Smurf attacks at specified or random victim ports. The attacker communicates with masters using any of a number of connection methods (e. g., remote shell bound to a TCP port, UDP based client/server remote shells, ICMP-based client/server shells such as LOKI [10], SSH terminal sessions, or normal "telnet" TCP terminal sessions.) Remote control of TFN agents is accomplished via *ICMP ECHOREPLY* packets. All commands sent from master to slaves through ICMP packets are coded, not clear text, which hinders detection.

Stacheldraht [11] (German for "barbed wire") combines features of Trinoo and TFN tools and adds encrypted communication between the attacker and the masters. Stacheldraht uses TCP for encrypted communication between the attacker and the masters, and TCP or ICMP for communication between master and agents. Another added feature is the ability to perform automatic updates of agent code. Available attacks are UDP flood, TCP SYN flood, ICMP ECHO flood and Smurf attacks.

Shaft [12] is a DDoS tool similar to Trinoo, TFN and Stacheldraht. Added features are the ability to switch master servers and master ports on the fly (thus hindering detection by intrusion detection systems), a "ticket" mechanism to link transactions, and a particular interest in packet statistics. Shaft uses UDP for communication between masters and agents. Remote control is achieved via a simple telnet connection from the attacker to the master. Shaft uses "tickets" for keeping track of its individual agents. Each command sent to the agent contains a password and a ticket. Both passwords and ticket numbers have to match for the agent to execute the request. A simple letter-shifting (Caesar cipher) is used to obscure passwords in sent commands. Agents can generate a UDP flood, TCP SYN flood, ICMP flood, or all three attack types. The flooding occurs in bursts of 100 packets per host (this number is hard-coded), with the source port and source address randomized. Masters can issue a special command to agents to obtain statistics on malicious traffic generated by each agent. It is suspected that this is used to calculate the yield of a DDoS network.

Tribe Flood Network 2000 (TFN2K) [13] is an improved version of the TFN attack tool. It includes several features designed specifically to make TFN2K traffic difficult to recognize and filter, to remotely execute commands, to obfuscate the true source of the traffic, to transport TFN2K traffic over multiple transport protocols including UDP, TCP, and ICMP, and features to confuse attempts to locate other nodes in a TFN2K network by sending "decoy" packets. TFN2K obfuscates the true traffic source by spoofing source addresses. Attackers can choose between random spoofing and spoofing within a specified range of addresses. In addition to flooding, TFN2K can also perform some vulnerability attacks by sending malformed or invalid packets.

mstream [14] generates a flood of TCP packets with the ACK bit set. Masters can be controlled remotely by one or more attackers using a password protected interactive login. The communications between attacker and masters, and a master and agents, are configurable at compile time and have varied significantly from incident to incident. Source addresses in attack packets are spoofed at random. The TCP ACK attack exhausts network resources and will likely cause a TCP RST to be sent to the spoofed source address (potentially also creating outgoing bandwidth consumption at the victim).

Trinity [15] is the first DDoS tool that is controlled via IRC or ICQ. Upon compromise and infection by Trinity, each machine joins a specified IRC channel and waits for commands. Use of legitimate (IRC or ICQ) service for communication between attacker and agents eliminates the need for a master machine and elevates the level of the threat. Trinity is capable of launching several types of flooding attacks on a victim site, including UDP, IP fragment, TCP SYN, TCP RST, TCP ACK, and other floods.

Flitz [16] is a DDoS tool which features spoofed ip/tcp/udp flood, flooding in parallel, distributed smurf attack and status report of the slave. With one stop command, you can stop all the slaves at once.

BlackEnergy [17] is a web-based distributed denial of service (DDoS) bot used by the Russian hacker underground. BlackEnergy gives the attackers an easy to control web-based bot that can launch various attacks and control the bots using a minimal syntax and structure. The BlackEnergy HTTP C&C is built on PHP, MySQL. The BlackEnergy botnet uses HTTP to communicate to its controlling servers by sending a POST message to the server.

2.4 Distributed Denial of Service Defenses

The seriousness of the DDoS problem and the increased frequency, sophistication and strength of attacks has led to the advent of numerous defense mechanisms. Yet, although it has been several years since the first distributed attacks were perpetrated, and many solutions have been developed since then, the problem is hardly dented, let alone solved.

2.5 Defense Challenges

The challenges to designing DDoS defense systems fall roughly into two categories: Technical Challenges and Social Challenges. Technical challenges encompass problems associated with the current Internet protocols and characteristics of the DDoS threat. Social challenges, on the other hand, largely pertain to the manner in which a successful technical solution will be introduced to Internet users, and accepted and widely deployed by these users. The main problem that permeates both technical and social issues is the problem of large scale. DDoS is a distributed threat that requires a distributed solution.

2.5.1 Technical Challenge

The distributed nature of DDoS attacks and use of legitimate traffic models and IP spoofing represent the main technical challenges to designing effective DDoS defense systems. In addition to that, the advance of DDoS defense research is hindered by the lack of attack information and absence of standardized evaluation and testing approaches. The following list summarizes and discusses technical challenges for DDoS defense:

1. Need for a distributed response at many points on the Internet.

There are many possible DDoS attacks, very few of which can be handled only by the victim. Thus it is necessary to have a distributed, possibly coordinated, response system. It is also crucial that the response be deployed at many points on the Internet to cover diverse choices of agents and victims. Since the Internet is administered in a distributed manner, wide deployment of any defense system (or even various systems that could cooperate) cannot be enforced or guaranteed. This discourages many researchers from even designing distributed solutions.

2. Lack of detailed attack information.

It is widely believed that reporting occurrences of attacks damages the business reputation of the victim network. Therefore, very limited information exists about various attacks, and incidents are reported only to government organizations under obligation to keep them secret. It is difficult to design imaginative solutions to the problem if one cannot become familiar with it. Note that the attack information should not be confused with attack tool information, which is publicly available at many Internet sites. Attack information would include the attack type, time and duration of the attack, number of agents involved (if this information is known), attempted response and its effectiveness, damages suffered, etc.

3. Lack of defense system benchmarks.

Many vendors make bold claims that their solution completely handles the DDoS problem. There is currently no standardized approach for testing DDoS defense systems that would enable their comparison and characterization. This has two detrimental influences on DDoS research: (1) since there is no attack benchmark, defense designers are allowed to present those tests that are most advantageous to their system, and (2) researchers cannot compare actual performances of their solutions to the existing defenses; instead they can only comment on design issues.

4. Difficulty of large-scale testing.

DDoS defenses need to be tested in a realistic environment. This is currently impossible due to the lack of large scale testbeds, safe ways to perform live distributed experiments across the Internet, or detailed and realistic simulation tools that can support several thousands of nodes. Claims about defense system performance are thus made based on small-scale experiments and simulations, and are not credible.

2.5.2 Social Challenges

Many DDoS defense systems require certain deployment patterns to be effective. Those patterns fall into several categories:

1. Complete deployment
2. Contiguous deployment
3. Large-scale, widespread deployment
4. Complete deployment at specified points in the Internet
5. Modification of widely deployed Internet protocols, such as TCP, IP or HTTP
6. All (legitimate) clients of the protected target deploy defenses.

None of the above requirements are practical for general purposes (although they may work well to protect an important server or application that communicates with a selected set of clients). The Internet is extremely large and is managed in a distributed manner. No solution, no matter how effective, can be deployed simultaneously in hundreds of millions of disparate places.

2.6 Defense Goals

The primary goal of DDoS defense is to provide good service to a victim's legitimate clients during the attack, thus cancelling the denial-of-service effect. Ideally, clients should perceive little or no service degradation while the attack is ongoing. The secondary goal is to alleviate the effect of the attack on the victim so that its resources can be dedicated to legitimate clients or preserved. Last, attack attribution (locating with high accuracy agent machines and perpetrators of the attack) will serve as a strong deterrent to DDoS incidents, as attackers could face the risk of discovery and punishment.

2.7 Defense Approaches

DDoS defense approaches can roughly be divided into three categories: preventive, survival and responsive approaches.

Preventive approaches introduce changes into Internet protocols, applications and hosts, in order to patch existing vulnerabilities and reduce the incidence of intrusions and exploits. Their goal is to prevent vulnerability attacks, and to impede the attacker's attempts to gain a large agent army. While preventive approaches are necessary for improving Internet security, they need to be deployed widely to constrain the DDoS threat. As long as large numbers of machines are insecure, attackers can still wage large-scale attacks. There is no reason to believe that preventive approaches will successfully undermine the power of the DDoS threat in the foreseeable future.

Survival approaches enlarge a victim's resources, enabling it to serve both legitimate and malicious requests during the attack, thus cancelling the denial of service effect. The enlargement is achieved either statically — by purchasing more resources, or dynamically — by acquiring resources at the sign of possible attack from a set of distributed public servers and replicating the target service.

Responsive approaches detect the occurrence of the attack and respond to it ("fight back") either by controlling attack streams, or by attempting to locate agent machines and invoking human action. In order to be successful, response approaches must meet following requirements:

1. **Accurate detection.** The system must be able to detect all attacks that inflict damage at the victim.
2. **Effective response.** The system must stop the attack flows, regardless of their volume or distribution. Alternately, in the case of response by agent identification, the system must be able to accurately identify the majority of attack machines

regardless of their distribution. This identification must be prompt so that the action can be taken while the attack is on-going. Ideally, identification responses should identify not only the agent machines, but also the master and the attacker machines.

3. **Selective response**. The system must differentiate between legitimate and attack packets, and ensure good service to legitimate traffic during the attack. Collateral damage due to the response must be lower than the damage suffered by legitimate clients in the absence of response. This requirement does not pertain to agent identification approaches.

2.8 Related Work and Research Gaps

2.8.1 Related work

Burch and Cheswick introduce the concept of network trace back. They identify attack paths by selectively flooding network links and monitoring the changes caused in attack traffic [18]. The scheme could easily fooled by the attackers by creating stealth traffic to match the required parameters.

Savage et al. Propose the Fragment Marking Scheme (FMS) for IP traceback [19]. They suggest that routers probabilistically mark the 16 bit IP identification field, and that the receiver reconstructs the IP addresses of routers on the attack path using these markings. The FMS do not work well if only a small number of routers implement them.

Bellovin [20] proposes the idea of ICMP traceback messages, where every router samples the forwarded packets with a very low probability (e. g., 1 out of 20,000) and sends an ICMP Traceback message to the destination. An ICMP Traceback message contains the previous and next hop addresses of the router, timestamp, portion of the traced packet, and authentication information.

Incoming packets to a network domain can be filtered by ingress routers. These filters verify the identity of packets entering into the domain, like an immigration security system at the airport. Ingress filtering, proposed by Farguson and Senie [21], is a restrictive mechanism that drops traffic with IP address that does not match a domain prefix connected to the ingress router.

Goodrich presents a marking scheme that marks nodes instead of links into packets [22]. Because this approach does not use a distance field, it has issues with attack graph reconstruction and does not scale to a large number of attackers.

Snoeren et al. propose SPIE, a mechanism using router state to track the path of a single packet [23]. The main advantage of SPIE is that it enables a victim to trace back a single packet by querying the router state of upstream routers; however, it does require routers to keep a large amount of state. Li et al. have further developed their approach, lowering the required router state, at the expense of a large communication overhead for traceback [24]. Consider the case where 50% of the routers implement the SPIE mechanism. Let's consider (very conservatively), that a router has 10 neighbouring routers on average.

With the SPIE mechanism, we find that a given router forwarded an attack packet, and we attempt to find out from which neighbouring router it came from. Thus, we need to contact the 9 neighbouring routers which potentially forwarded the packet (we do not need to query the next-hop router towards the victim). Let's assume that 5 of the neighbouring routers implement SPIE, but that 4 do not implement it. Besides the 5 SPIE-enabled routers, we also need to contact all neighbours of the 4 legacy routers, about 40 additional routers. However, 20 of those routers are legacy routers themselves, so we need to contact all of their neighbours as well. It is clear that this approach scales poorly if an attack path traverses several legacy routers.

Dawn et al. proposed the Advanced Marking Scheme and the Authenticated Marking Scheme [25], which allows the victim to traceback the approximate origin of spoofed IP packets. It used the identification field of IP header for storing the hash as well as the distance from the router which marked the packet. The AMS approach suffers from a problem. The upstream map of routers used by AMS is gathered using the trace route tool, which does not distinguish between AMS-enabled and legacy routers. However, the AMS distance field only counts hops of AMS-enabled routers, which leads to the following problem.

Assuming the victim has identified a router at distance x, when receiving an edge marking from distance x+1, the victim will have to test the IP addresses of all the routers at distances greater than x (rather than just those at distance x + 1) because the edge between two AMS-enabled routers may traverse several non-marking legacy routers. This effect will lead to an increase in the false-positive rate of the scheme, particularly with high percentages of legacy routers present. AMS also suffers from lack of the enough range for the hash values. With 11 bits, we can have only 2^{11} hashing values. With 2^{32} possible IP address space, the probability of collision will be very high.

Park and Lee [26] propose route-based distributed packet filtering, which rely on route information to filter out spoofed IP packets. The authors of the paper shows that with partial deployment of route-based filters, about 20% in the Internet AS topologies, it is possible to achieve a good filtering effect that prevents spoofed IP flows reaching other ASes. These filters need to build route information by consulting BGP routers of different ASes. Since routes on the Internet change with time, it is a challenge for route-based filters to be updated in real time. Finally, all filters proposed in the literature so far fall short to detect IP address spoofing from the domain in which the attacker resides.

Dean et al. suggest algebraic traceback, an algorithm to encode a router's IP address as a polynomial in the IP identification field [27]. Adler presents a theoretical analysis of traceback, presenting a one-bit marking scheme [28]. This work is primarily of theoretical interest, and does not scale to large numbers of attackers.

Ratul Mahajan et al. [29] presented the term PUSHBACK. In this system the packets arrive at router in the input queues from where they are sent to the module which matches them with the congestion signature. The packets survived from match go to the output queue directly. The packets whose signature match with congestion signature (may be attack packets or legitimate) are sent to the rate limiter which drops a significant amount of traffic depending upon parameters set by "pushbackd" daemon. The pushback method introduced by Ratul et al. performed pretty well but it could not stop the legitimate users to suffer.

Ruiliang Chen et al. [30] proposed Attack Diagnosis (AD), a novel attack mitigation scheme that combines the concepts of Pushback and packet marking. AD's architecture is inline with the ideal DDoS attack countermeasure paradigm, in which attack detection is performed near the victim host and attack mitigation is executed close to the attack sources. AD is a reactive defense that is activated by a victim host after an attack has been detected.

Rajesh Sharma et al. [31] proposed Shared Based Rate Limiting, whose basic mechanism was to have monitoring, rate limiting and filtering routers at various levels of ISPs. The participating routers, start there function after getting a signal from a server under attack.

The scheme was invoked only during attack times, and was able to mitigate attack traffic through dynamic filtering. Server tells edge routers to rate limit the traffic according to the share of traffic which was being passed through particular routers. The solution proposed was an ISP level solution.

Bhawna et al. [32] proposed An Integrated Framework for Proactive Mitigation, Characterization and Traceback of DDoS Attacks, a novel integrated framework which deals with proactively mitigating the influence of the attack, characterization of the TCP flows as attack or legitimate, and identification of the path traversed by the flow once it has been characterized as an attack flow. In the proposed framework, generation of copies of TCP/IP headers by intermediate routers provides for the dual functionality of proactive mitigation and traceback. The characterization of the flows has been achieved by an innovative Exactly Periodic Subspace Decomposition (EPSD) based approach.

Zhaoyang Qu et al. [33] proposed A Novel Two-step Traceback Scheme for DDoS Attacks traceback scheme to track DDoS attack source by dividing the tracing process into two steps. In the first step, Packet Marking Method based on Autonomous System (ASPMM) is adopted to determine the attack-originating Autonomous System (AS). In the second step, Non-repeated Probabilistic Packet Marking (NRPPM) is used to identify the exact origin of the attacks in the specific AS. Compared with previous algorithms, the two-step traceback scheme has the benefits of low bandwidth consumption, quick convergence speed, light computational overhead of address recombination; it can decrease the number of packets the path reconstruction needs, and improve the efficiency of path reconstruction, hence making it possible to trace the DDoS attack source rapidly.

2.8.2 Research Gaps

Burch and Cheswick [18] could be easily fooled by attackers by creating stealth traffic as well as the enormous load on the network created causes the users to suffer a lot.

FMS [19], SPIE [23], ingress filtering [21] had implementation issues. Partial implementation of these solutions caused loopholes in system which could be easily exploited to attack victims.

FMS [19] proposed an 11 bit hash value to be included in the identification field of the IP header. Research proved that 11 bit hash value fails (start to collide) after the total number of attacker exceeds 60[34].

ICMP traceback messages [20] could be easily created by attackers thus they can be mixed with actual packets generated by routers to misguide victim's calculations.

AMS [25] biggest disadvantage was to rely on external tools for topology creation. AMS Calculated the distance from victim to marked router depending upon AMS enabled routers whereas the topology created by the tools also included the legacy routers. This difference causes a loophole in the system.

Route Based Filtering [26] relied on BGP routers for updating which changes very frequently so the real time updating was a big issue.

Algebric Traceback [27] does not scale to a large number of nodes because of its extensive calculations and requirement for a high performance routers at each node.

Ruiliang Chen et al. [30] did not have the capability of scaling well. If the attackers increase there power the system could easily be nailed down.

Rajesh Sharma et al. [31] and Zhaoyang Qu et al. [33] solutions biggest drawback was that it depends upon ISP or Autonomous Systems for routing information. The most difficult part of the scheme will be to bring all the AS to a consent to use the solution so obviously they had implementation issues.

Bhawna et al. [32] framework depends upon header forwarding to the victim but the solution provided was very fragile. It did not scale well due to the storage required at the routers for the packets so practically can not be implemented.

From all above gaps found in the earlier work done by many of the authors, following is the summary of some of the properties which are crucially required in a proposed scheme.

1. Proposed solution must work even if partially deployed across routers in the Internet.
2. It must require only a small hardware change on routers.
3. It must allow a victim to identify the attack path after only a small number of packets.
4. It could scale to a large number of attackers.
5. It must allow a victim to Traceback locally, without communicating with any router or ISP.

Chapter 3
Defense Strategy Against DDoS Attacks

Every solution works on the basis of some of the facts of the system in which it is going to be implemented, so is our solution. Before proposing we assume some assumptions which should be fulfilled by the network.

3.1 Assumptions

We assume the following network environment. Every host, either a client or a server, is connected to its local edge router. Edge routers are in turn interconnected by core routers. The server being attacked is called the victim. A study [34] has shown that 95% of the routes observed in the Internet have fewer than five observable daily changes. So we make the reasonable assumption that every route from a client to the victim is fixed during the timeframe of interest. We also assume that Internet routers are not compromised.

We use the term false negative to denote a zombie machine whose attack packets have not been filtered, and use the term false positive to denote a legitimate client whose packets have been incorrectly throttled.

Like other packet marking based mitigation schemes, we assume the existence of an IDS module installed at the victim (or at its firewall), which is able to identify and collect malicious packets.

3.2 Proposed Solution

The proposed solution provides for reactive mitigation of the effect of DDoS attacks as described next. Whenever a packet arrives at a router (belonging to a predefined set) to be forwarded to the potential victim server, instead of sending that packet directly on the outbound link, the router mark the packet and then forward it to the server for characterization. This marking is from the family of PPM (Probability Packet Marking [35].

The technique to be used in this solution for mitigation provides the *dual* functionality of IP Traceback as well. The 16-bit IP *Identification* field in the header of the original packet will be used for traceback purposes.

The packets sent to the victim server will be subject to the characterization test described next. For characterization, Traffic Measurement Analysis (TMA) [36] technique will be used as part of this Solution. The TMA is a well proved technique for characterization proposed by Lersak Limwiwatkul et al. Any other technique can be used convincingly rather then TMA without affecting the efficiency of the proposed solution.

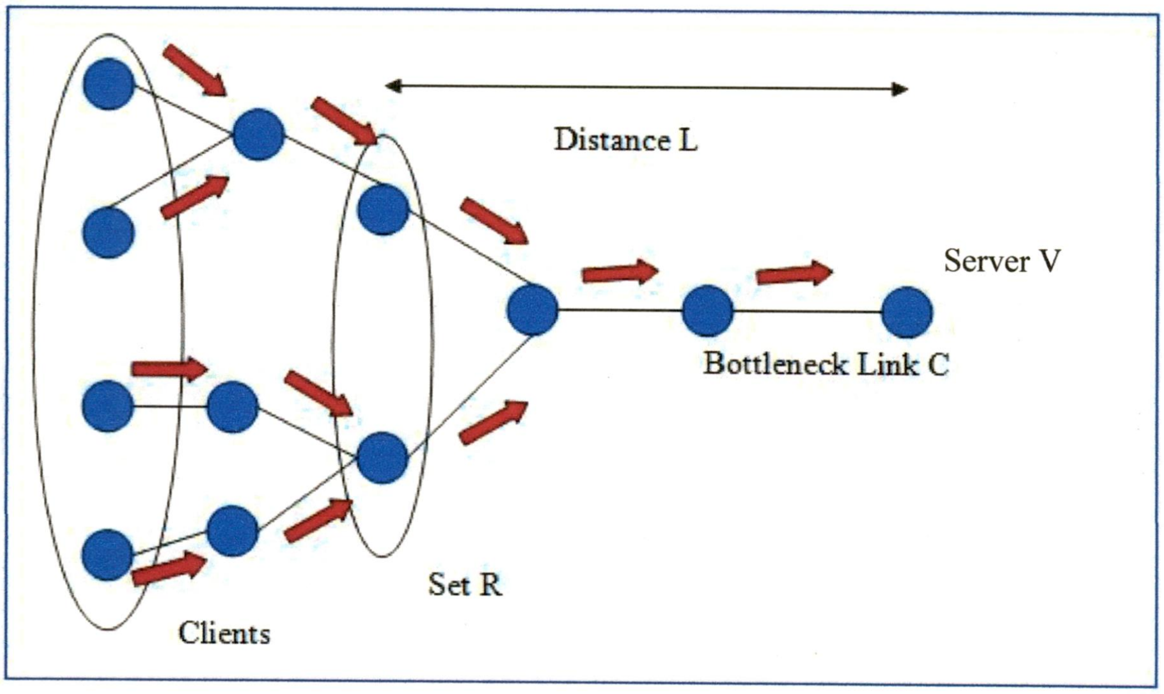
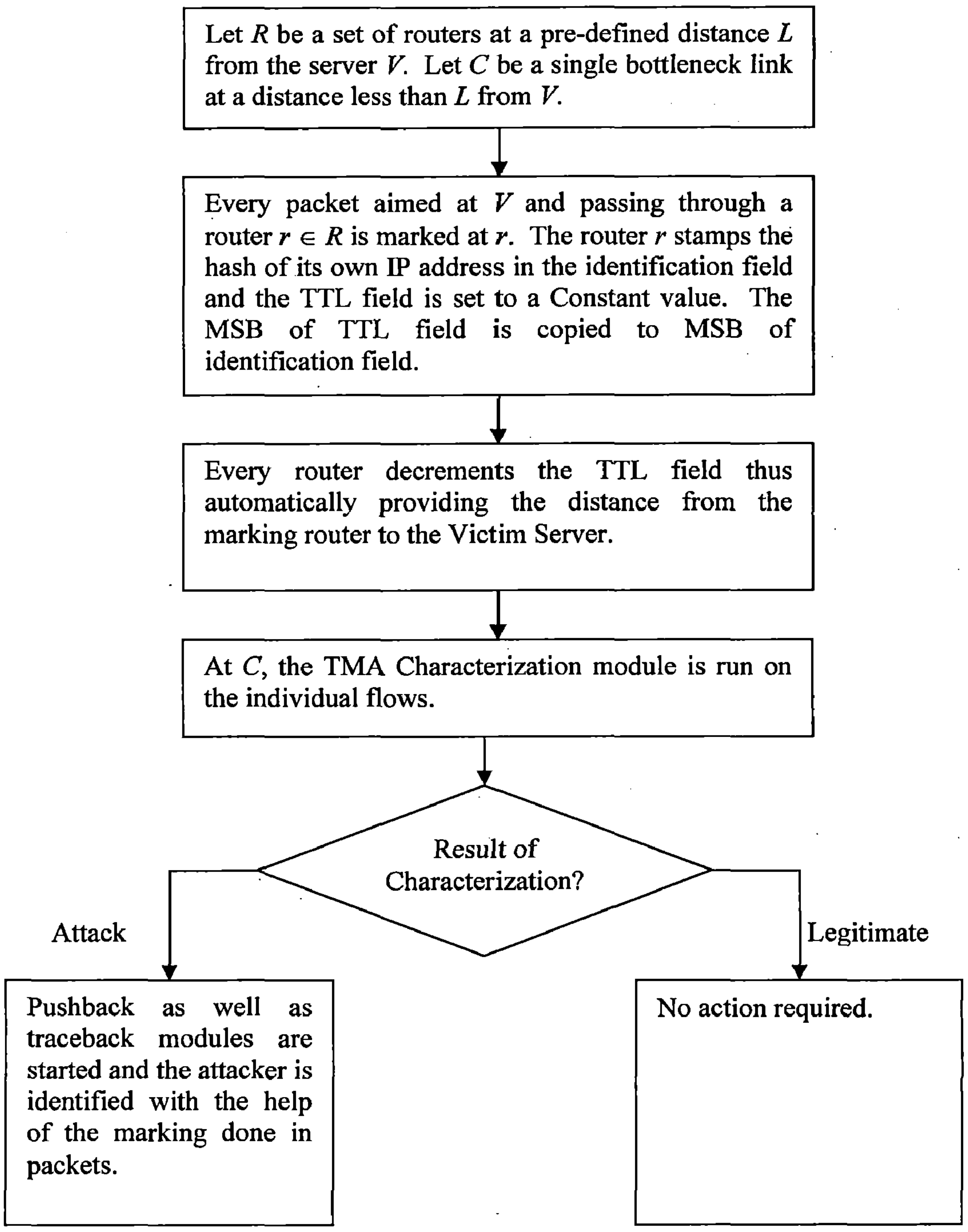


Figure 3.1: Sample topology to illustrate proposed solution.

To get a better understanding of the proposed model, consider a sample topology shown in Figure 3.1. The topology considered is similar to the one used traditionally to depict a typical client-server scenario in the Internet for simulation purposes [37].



Let R be a set of routers at a pre-defined distance L from the server V. Let C be a single bottleneck link at a distance less than L from V.
Every packet aimed at V and passing through a router $r \in R$ is marked at r. The router r stamps the hash of its own IP address in the identification field and the TTL field is set to a Constant value. The MSB of TTL field is copied to MSB of identification field.
Every router decrements the TTL field thus automatically providing the distance from the marking router to the Victim Server.
At C, the TMA Characterization module is run on the individual flows.
Result of Characterization?
Attack
Legitimate
Pushback as well as traceback modules are started and the attacker is identified with the help of the marking done in packets.
No action required.

Figure 3.2: Flowchart depicting details.

The clients (attack and legitimate) send their requests (indicated by thick arrows) to the server V. The routers (set R) en route from the clients to the server will mark these packets and send the packets to V (indicated by thin arrows). Once these packets reach the bottleneck link C, they will undergo the TMA test and thus the flows will be characterized as attack or legitimate. If a flow is characterized as a legitimate flow, No action is required. If a flow is characterized as an attack flow, then the router of Victim V informs the routers in upward direction to start pushback for giving immediate relief to the Victim. As well as the method to find the attacker (traceback) starts on Victim server to identify the attacker. A flowchart depicting the solution is illustrated in Figure 3.2. The effort of creating an attack graph is saved in case of legitimate flows.

This chapter only gives a brief overview of the proposed Solution. The individual phases of the Solution are considered in detail in the subsequent chapters. The aim of this chapter is to show the big picture so that the forest is not missed for the trees. The details of the Characterization, Mitigation and Traceback techniques applied in the framework are covered in the subsequent chapters.

Chapter 4
Characterization, Mitigation and Traceback

An overview of the entire solution implemented as part of this work has been presented in Chapter 3. In this chapter, the three phases of Characterization, Mitigation and IP Traceback are covered in detail.

4.1 Characterization

Characterization is the process of identifying accurately which of the flows aimed at the victim are attack flows and which ones are legitimate. Lersak Limwiwatkul et al. [36] proposed the Traffic Measurement Analysis to identify any kind of suspected flows in the incoming traffic flows. In this solution, Traffic Measurement Analysis based characterization scheme is used for characterizing DDoS attacks.

The method works on the basis of general characteristics of the Traffic. It proposes to measure various characteristics like volume, distribution and ratio of the packets and any discrepancy in the measurements of the characteristics will prove any suspicious move.

Before applying Traffic Measurement Technique we can apply some rules like header check on the packet header which allow us to subcategorize the packets on which measurement is to be done. After having header check traffic measurements should be applied to study the attack traffic signature. The Algorithm proposed by the authors is:-

1. Input is the Network Packet.
2. Consider whether packet is matched with Designed conditional rules or not, if not, it should be declined.
3. Consider whether the test period is within the considering period (ΔT) or not, if not, the period should be changed.
4. If packet matched with rules and within ΔT, packet will be analyzed by Traffic Measurement Analysis.

5. The data from the analysis can be used further; for example creating the graph for monitoring.

The matched packets can be processed for the traffic measurement analysis using the following rules:
Let ΔT be the defined time period; For example 0.5 Seconds

- Volume Measurement Analysis equation is-

$$\text{Volume} = \frac{\text{Total Number of packet}}{\Delta T}$$

Here, we count the number of packet occurring during a period of time.

- Distributed Measurement Analysis Equation is-

$$\text{Distributed} = \frac{\text{Total Number of distinct packet}}{\Delta T}$$

Here we measure the number of distinct packets of the observing data packets per ΔT.

- Ratio Measurement Analysis Equation is-

$$\text{Ratio} = \frac{\text{Total Number of packets incoming}}{\text{Total Number of packets outgoing}}$$

Here we divide the total number of incoming packets with the total number of outgoing packets.

4.1.1 The Packet Process

TCP is a sliding-window and acknowledgement (ACK) based transport protocol. The window size of a TCP flow limits the number of in-flight packets it can have in the network. The window size is determined by the advertised window size of the receiver and the estimated congestion level of the network. TCP is a window- based, and Acknowledgement (ACK) - based transport protocol widely used in the Internet. Every data packet arriving at the receiver can permit the receiver to transmit an ACK packet to the sender. Similarly, every ACK packet arriving at the sender allows it to place a new data packet on the network. Thus, if we monitor the network at any point between the sender and the receiver and if we observe a certain number of packets belonging to a particular flow, then it is quite probable that the same number of packets belonging to that flow will be visible after one round – trip time between the sender and the receiver. This introduces periodicity in a normal TCP flow and causes the ratio of incoming versus outgoing packets equals to one. Even if the TCP uses SACK (Selective Acknowledgement) and doesn't send the acknowledgement for every packet, still the ratio can not exceed three (Assumption). In an attack flow, the attackers overwhelm the server by not obeying the TCP policy of waiting for ACK packets before the outstanding data packets can be sent. Thus, there is the loss of a constant ratio of incoming to outgoing packets in such flows.

4.1.2 Algorithm Used

Let T_{sample} be the time interval after which the flow statistics (packet arrivals) are monitored continuously per flow. Let $N_{current}$ be the number of packets arrived till the sample instant from the time the flow was active minus the number of packets arrived till the last sample instant. Let volume_stats (named as volumeudp. txt and volumetcp. txt in coding) be a file of which stores the value of $N_{current}$ for the last all instants. The statistics of these volumes will be used for validating our results later into the simulation. Once the volume for the links is placed in the volume_stats, the traffic measurement analysis starts on the incoming packets.

If the ratio of the incoming packets to the outgoing packets is less then three, the flow is tagged as the legitimate flow otherwise tagged as an attack, hence declared as the suspicious flow in the output.

The detailed steps are shown in the form of a flowchart in Figure 4.1.

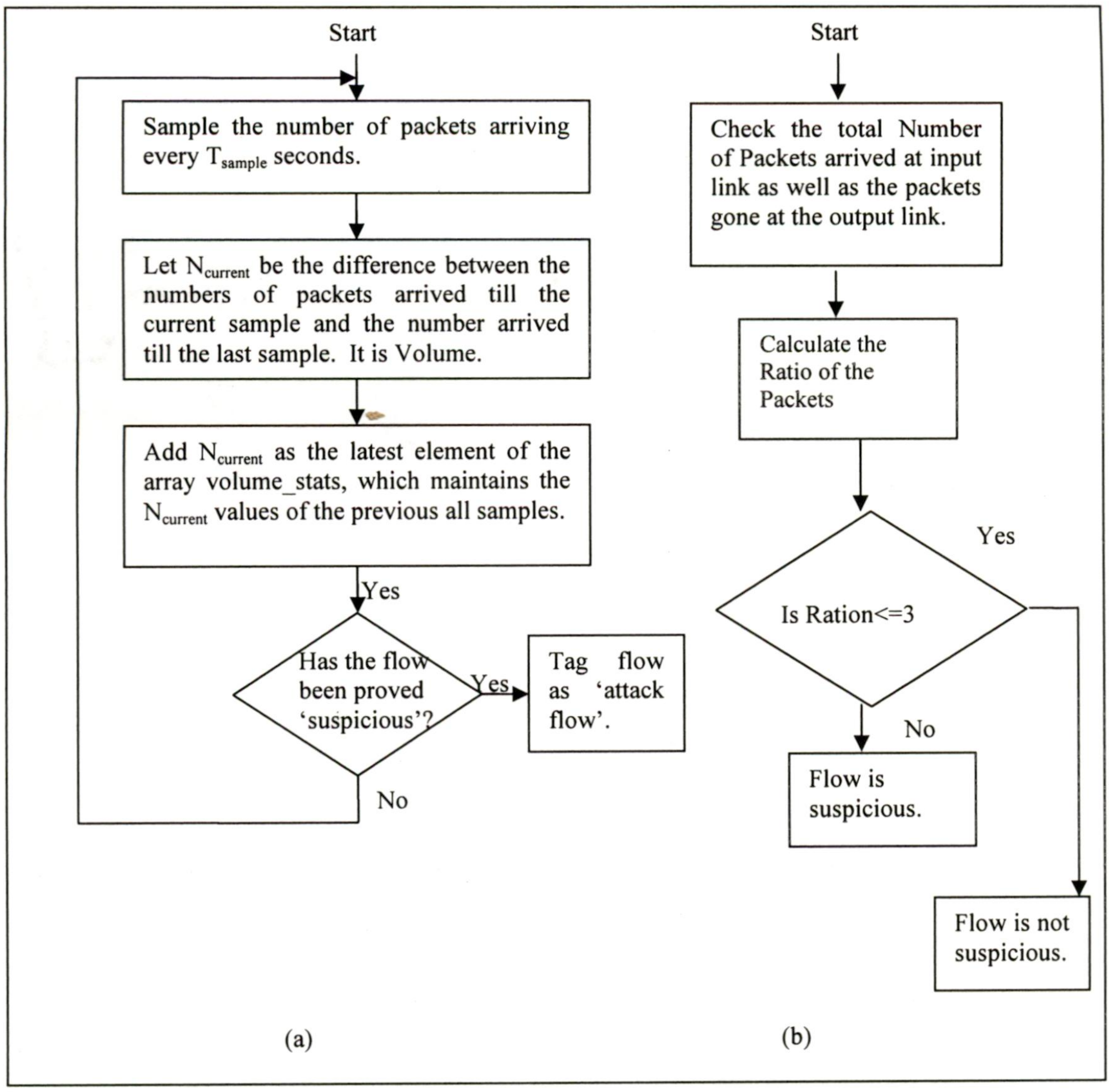

Start
Sample the number of packets arriving every T_{sample} seconds.
Let $N_{current}$ be the difference between the numbers of packets arrived till the current sample and the number arrived till the last sample. It is Volume.
Add $N_{current}$ as the latest element of the array volume_stats, which maintains the $N_{current}$ values of the previous all samples.
Yes
Has the flow been proved 'suspicious'?
Yes
Tag flow as 'attack flow'.
No
(a)
Start
Check the total Number of Packets arrived at input link as well as the packets gone at the output link.
Calculate the Ratio of the Packets
Is Ration<=3
Yes
No
Flow is suspicious.
Flow is not suspicious.
(b)

Figure 4.1: (a) Flowchart for sampling the number of packets per flow. (b) Flowchart for invoking the Traffic Measurement Analysis functionality for the online methodology.

4.2 Mitigation and Traceback

The proposed traceback mechanism is in the family of PPM (Probabilistic Packet Marking) traceback schemes, and consists of 3 parts: a packet marking scheme to be deployed at routers, pushback to be implemented at router level and, map and path reconstruction algorithms used by end hosts receiving the packet marking. Figure 4.2 shows the notation we use in this dissertation.

c	Bit replacement for the 5 LSB of the TTL
P.dist bit	The distance bit in packet P
P.hash	The hash in packet P
q	Marking probability
TTL [0]	Least significant bit (LSB) of the TTL
TTL [5]	Sixth bit of the TTL
TTL [4::0]	The five least significant bits of the TTL
H (IP)	Compute a cryptographic hash function on the IP address, e.g., SHA - 1(IP)
b|c	Concatenation of the values b and c
nmap	Number of unique fragments needed for single IP address map reconstruction

Fig 4.2: Notation used

In proposed scheme, an attack victim is assumed to have constructed a map of upstream routers and their IP addresses using packet markings received before the attack itself occurs. Routers mark the 16-bit IP ID field of certain forwarded packets. The packet markings contain two elements: a hash of the marking router's IP address, and a distance field. Based on the distance field and the TTL of a given packet, the attack victim can determine from how many hops away the marking is generated. The victim uses the hash fragments and distance calculation from the markings in the malicious packets in conjunction with its router map to identify a candidate set of marking routers. After a number of different hash fragments matching a particular router arrive at the victim, that router is added to the reconstructed attack path.

4.2.1 Packet Marking

In the proposed scheme, as in all other PPM schemes, routers mark (overwrite) the 16 bit IP Identification (IP ID) field of the IPv4 header of a small percentage of the packets that they forward. A router marks a forwarded packet with a certain probability, q, which is a global constant among all solution enabled routers. A packet mark is divided into two fields, as shown in Figure 4.3. The first field, denoted as b, is the 1-bit distance field. The second field involves the router's hash. Each router pre-calculates a hash of its IP address.

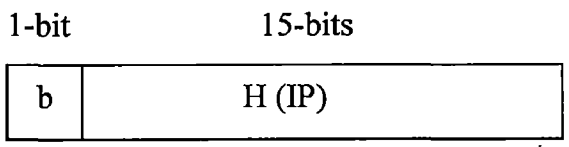


Figure 4.3: Marking field diagram. (The distance field b is one bit and the remaining 15 bits are used for the hash.)

Unlike other PPM schemes, solution has a deterministic marking aspect. For each packet that a particular router has not probabilistically marked, that same router calculates a boundary check based on the packet's TTL field and distance bit. The boundary check contains a calculation of the minimum bound on the distance, in solution-enabled and legacy router hops, since the packet was last marked. If the packet was not marked for the past 32 hops then the boundary check evaluates to true and the packet is automatically marked by the forwarding router. The boundary check is evaluated as: (b|c- TTL [5::0]) mod 64 > 32, where b|c denotes the concatenation of the distance bit b in the packet with the global constant c, and TTL [5::0] denotes the six least significant bits of the TTL field.

When marking a packet a router sets the 5 least significant bits of the packet's TTL to a global constant c, and stores the 6th bit of the TTL in the distance field b. This last step allows the next solution-enabled router, or the packet receiver, to determine the distance since the router's mark. We explain the details of calculating the distance in the following section. Finally, if a router does not mark the packet then it will not change any part of the IP ID field.

4.2.2 Calculating Distance

Recall the distance related operations a marking router performs in solution: it sets the 5 least-significant bits of the packet's TTL field to a global constant c, and stores the sixth bit of the TTL in the distance field b. When a packet arrives at its destination, the distance at which the packet was marked is computed as: d = (b|c - TTL[5::0]) mod 64, where b|c denotes concatenation of the one bit distance field b with the five bit TTL replacement constant c. Because legacy routers decrement the TTL, the distance is representative of the exact number of hops from a marking router, rather than just the number of hops of traceback enabled routers. Invalidating the distance space (through boundary check) and having solution-enabled routers automatically mark packets with distances in the invalid range will increase the percentage of marked packets relative to other traceback schemes using the same marking probability, q.

4.2.3 Pushback

Proposed solution uses a modified version of pushback. It removes the disadvantage of pushback which makes the legitimate users suffer. According to the system we consider that we already have means to detect an attack. The attack alarm will be supported by a sudden increase in the arrivals of hashed packets that signifies an abnormal increase in traffic. This alarm activates the pushback as well as the path reconstruction (As described in Section 4.2.5). Rather then stopping the complete traffic whose signature match with congestion signature the pushback just limits the traffic up to the maximum capability of the connecting link of the routers (which was decided as a part of TCP/IP handshake), as our victim is capable of handling that much load easily. For example if the capacity of the link between the victim and the router next to victim is having a capacity of 100Mbps. At the time of attack, there is 150Mbps data arriving, out of which 50Mbps is from legitimate user and 100 Mbps is from attacker.

The original pushback orders the upstream router to stop all the traffic whose signature match with congestion signature so it will stop all the 100Mbps traffic as well as some part of legitimate user (whose signature match with congestion signature). This make the innocent user suffer. The modified version of pushback stops the only extra 50Mbps rather then stopping all 100 Mbps. It will make less legitimate users suffer due to unintentional match of signature. Also it would not affect the victim as victim is already capable of handling that much of load.

4.2.4 Map Reconstruction

Proposed scheme need the map of upstream routers for traceback. For map reconstruction any of the tools available can be used as the proposed scheme does not have a difference in the distance given by the tool to the distance calculations done by the scheme. For simulations purposes we have not developed the map reconstruction module as we are well aware of the topology. This is a pre-processing step which should be done offline so it does not affect the actual efficiency of the system.

In this section, we describe how the victim can generate this upstream router map. From Section 4.2.1, every packet mark consists of an IP address hash fragment, a fragment number, and a distance bit. Its map reconstruction leverages the fact that an endhost can group together packets that traverse the same path during a TCP connection. When receiving packet markings from the same distance and TCP connection, an endhost can assume that the markings come from the same router. Thus, the endhost collects nmap unique fragments from a particular distance, scans through the space of all possible IP addresses, and adds the IP address whose hash matches the nmap fragments to the upstream router map.

4.2.5 Path Reconstruction

The purpose of an IP Traceback mechanism is to reconstruct the IP addresses of the routers on the path from the attacker to the victim. We assume that the victim has completed the map reconstruction phase that we outline in the previous section (i.e., generated the map of upstream routers). Similar to all previous IP Traceback mechanisms, we assume that the victim has a mechanism to identify malicious packets, so that it can perform traceback. In the path reconstruction phase, the victim uses its router map and marked attack packets to reconstruct the attack path, which is the set of all routers that forwarded attack packets. Earlier we describe that the victim can detect how many routers the packet traversed since it was marked, using the one bit distance field b, the last six bits of the TTL, and the five bit TTL replacement constant c: $d = (b|c - TTL[5::0]) \bmod 64$.

Based on these values, the victim can identify candidate attack path routers after receiving only a single marked packet as follows. The victim compares the hash fragment it receives with the hash fragments of all routers at the distance d in its router map, and marks any router with a matching fragment. In the case that the victim's map contains a unique path from the reconstructed router to the victim, the victim can knows that the router, and all its downstream routers, are on the attack path as well.

Chapter 5
System Design and Implementation

5.1 System Design

To investigate the effectiveness of the proposed framework in defending against DDoS attacks, the simulation on a simplified topology has been carried out on Network Simulator 2(ns-2. 33) [38]. A large number of scenarios are explored.

5.1.1 Network Simulator

Network simulations for Ns-2[38] are composed of C++ code, which is used to model the behaviour of the simulation nodes, and oTcl scripts that control the simulation and specify further aspects, for instance the network topology. This design choice was originally made to avoid unnecessary recompilations if changes are made to the simulation set-up. Back in 1996 when the first version of ns-2 was released, this was a reasonable intent, as the frequent recompilation of C++ programs was indeed time-consuming and slowed down the research cycle. However, from today's perspective, the design of ns-2 trades off simulation performance for the saving of recompilations, which is questionable if one, is interested in conducting scalable network simulations.

We chose Ns2 over the other simulation tools available like OPNET, OMNET++, SWANS, JiST, and SimPy because of the vast options available in Ns2 for academic research. Some of the tools mentioned above are not freeware which makes them restricted to the developer community and some of them provide the facility to work in Ad-Hoc networks only. As my work was focussed in wired networks, Ns2 was the best option to choose.

5.1.2 System Components

The system consists of the following components:

Clients: Two types of clients are considered: Legitimate Clients and Attackers. The legitimate clients obey the TCP protocol, whereas it is not expected of the attackers to adhere to the TCP congestion avoidance protocols.

Server: The service provided by the server is a generic TCP-based service. The legitimate clients connect to the server with the aim of achieving file downloads, whereas the attackers aim at clogging the bottleneck link leading to the server in order to make the service unavailable to the legitimate clients.

Agents on Intermediate Routers: one new agent (IFS) is created in order to provide for the functionality of the proposed framework. They are deployed at certain intermediate routers. They are discussed in detail next.

5.1.3 Simulation Model

Clients: The legitimate clients are modelled by FTP applications run on TCPNewReno (a flavour of TCP). They obey the constraints imposed by the TCP protocol. The attackers are modelled by CBR traffic on UDP. This choice is done as a UDP sender does not need to wait for any acknowledgement from the receiver before sending out further outstanding packets. This property is apt to model an attacker as an attacker would normally send out large bursts of packets continuously with the aim of flooding the links leading to the server under attack.

Server: The server is modelled by a simple TCPSink which sends out ACK packets for packets it receives.

IFS: *Integrated Functional System* agent is deployed on the routers that are located at a certain pre-determined distance L from the server. This agent receive packets from the clients (legitimate and attackers) that are actually aimed for the server and mark them before sending to the server.

Flow Monitor: ns-2 provides for the Flow Monitor feature which is used to monitor individual flows. To provide the functionality of Characterization of flows as attack or legitimate, the Flow Monitor is modified. The statistics of every flow are sent to characterizing modules which label the traffic as legitimate or Attack.

5.2 Implementation

5.2.1 Simulation Topology

Figure 5.1 illustrates the simulated network topology. The topology considered is similar to the one used traditionally to depict a typical client-server scenario in the Internet for simulation and validation purposes [32].

The simulation is carried out in Network Simulator ns-2 [38]. The legitimate clients are TCP agents that request files of size 1 Mbps each. The attackers are modelled by UDP agents. The rate is kept very high (3Mbps) which is very typical of an attack flow. A UDP connection is used instead of a TCP one because in a practical attack flow, the attackers would normally never follow the basic rules of TCP, i.e. waiting for ACK packets before the next window of outstanding packets can be sent, etc.

Each of the links is a duplex link of 5 Mbps bandwidth, with the exception of the high bandwidth bottleneck link which is modelled by a combination of two simplex links of 20 Mbps bandwidth each. The Flow Monitor tailored for the purpose of Characterization is attached to the bottleneck link. As illustrated in Figure 5.1, the legitimate clients, the attackers, the agent *IFS* are differentiated by colour.

5.2.2 Procedures

The procedures used to implement the various features discussed are described next.

5.2.2.1 Characterization

For characterization the method used is Traffic Measurement Analysis [36]. The scheme is already a proved solution for DDoS attack characterization. The most

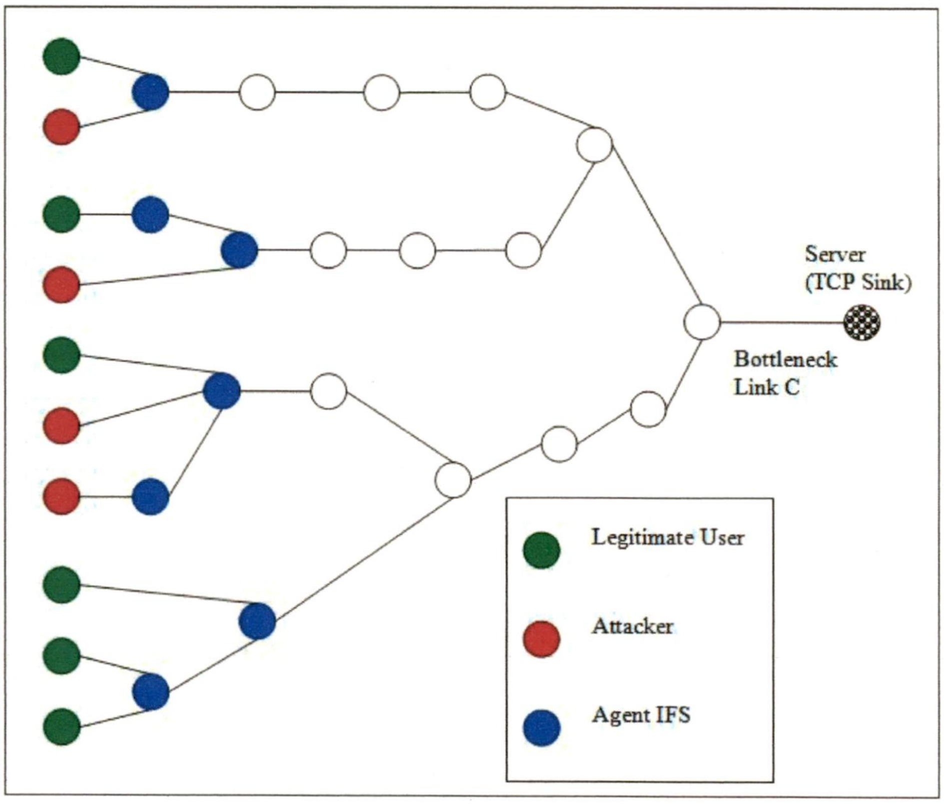


Figure 5.1: Topology used for Simulation.

powerful advantage of the scheme is that it did not make flash crowd as a DDoS attack as done by many other solutions available.

Any of the other solution for characterization can comfortably replace the used solution for characterization. The Flow Monitor in NS2 is modified to characterize the attack as well as to start the pushback and traceback at victims end.

5.2.2.2 Mitigation and Traceback

IFS: The *IFS* agent is implemented by the class *IfsAgent*. Its *recv* function provides the functionality of receiving the packets, marking them according to the rules described and then forwarding it to the receiver.

5.2.3 Simulation Parameters

Table 5.1 lists the simulation parameters, their values and description of these parameters used in the simulation.

5.2.4 Performance Evaluation Metrics

The main purpose for simulation is to study the cost and benefit of the proposed framework. The cost is incurred by the overhead in terms of the additional router capacity needed as well as the network capacity required for implementing the solution. The benefit is measured in terms of the improvement in the congestion window of the legitimate users when the system is under attack.

Parameter	Value	Description
Simulator	ns-2	Simulation tool
Number of Nodes	30	Network nodes
Client Load	0. 1 - 0. 4	Relative load issued by client requests
Attack Load	0-0. 9	Relative load due to attack traffic.
Simulation time	0-300 sec	Simulation duration
Attack time	150 – 180 sec	Attack duration
Legitimate Traffic type	TCP	File Transfer Protocol
Attack Traffic Type	UDP	Constant Bit Rate
Client-Router link BW	5 Mbps	Bandwidth
Attacker-Router link BW	5 Mbps	Bandwidth
Router-Router link BW	5 Mbps	Bandwidth
Router-Server link BW	40 Mbps	Bandwidth

Table 5.1: Simulation Parameters.

Chapter 6
Results and Discussion

An analysis of the simulation experiments carried out in ns-2 [38] is carried out next.

6.1 Results for Characterization

Legitimate Flow (T $_{sample}$=0. 2 seconds):-

For T_{sample} = 20 ms. a sample result is as shown next. The figure 6.1 shows the expected behaviour of a user when no attack is active. The starting peak of the traffic is due to slow start phase. Once the slow start phase is over and the window size is fixed the legitimate flow goes steady.

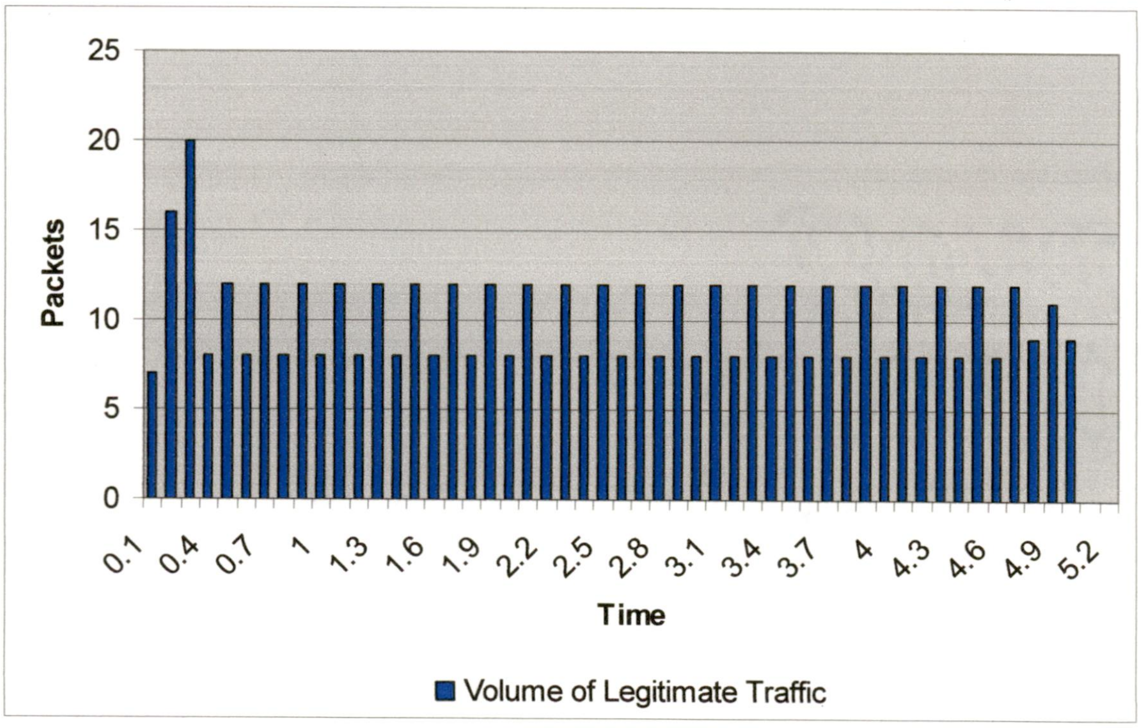


Figure 6.1: Traffic Volume for a legitimate flow.

The packet process shows the desired behaviours of the users. Once the slow start phase is over they should never send packets more then the window size allotted.

Attack Flow:-

The packet process of an attack flow is illustrated in Figure 6.2.

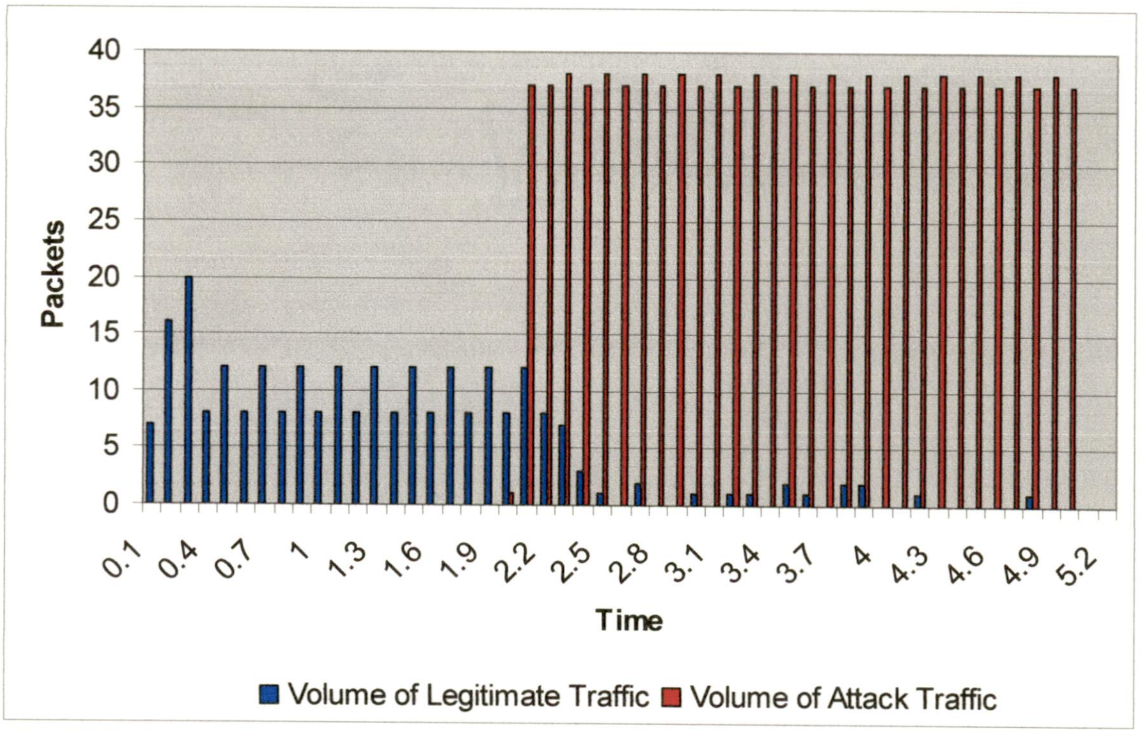


Figure 6.2: Traffic volume for an attack flow.

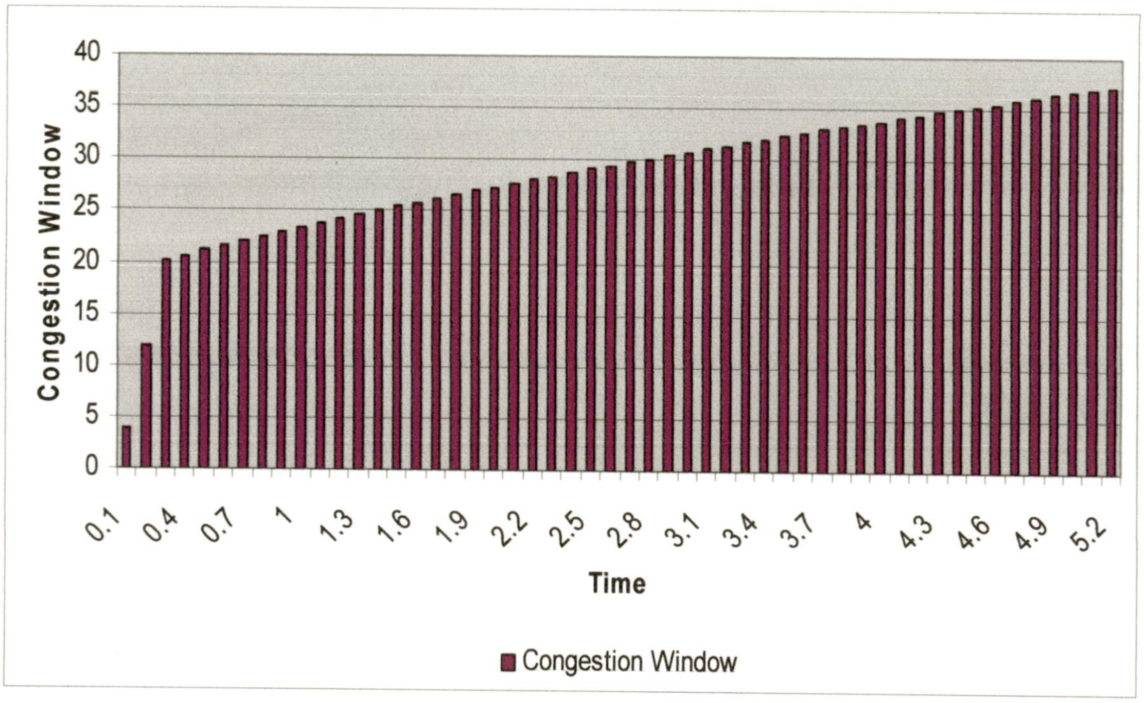


Figure 6.3: Congestion Window of a legitimate flow.

The packet process highlights the fact that it was a bursty DDoS attack. The process kills the traffic of the legitimate users. When the attack is on the volume of the legitimate users almost falls to zero. The attack volume is so high that it consumes all the bandwidth between bottleneck router and the Destination and does not allow any of the legitimate traffic to flow.

The behaviour of the flow can also be judge by seeing the congestion window of the TCP links as shown in Figure 6.3. We will evaluate our mitigation and traceback on the basis of this congestion window only.

6.1.1 Sample Output

The characterization technique is applied to the simulation topology and the results obtained are shown next. The 6 legitimate sources are started first at same time (0.0); hence they are allotted flow ids 0,2,4,7 to 9. Then the 4 attackers are started at the same time (2.0) to depict the real-world scenario where multiple attackers (zombies) would synchronize their start times and end times in order to make their attack more effective. Since they are started after the first 6 sources are already active, they are assigned flow ids 1, 3, 5 and 6. A few relevant lines of output are shown below. After each 0.2 seconds the method gives us an output that which of the incoming flows are suspicious. Nothing suspicious turns up unless time 2.0 and that is the time when attack starts.

```
Nothing suspecious from Node 0
Nothing suspecious from Node 1
Nothing suspecious from Node 2
Nothing suspecious from Node 3
Nothing suspecious from Node 4
Nothing suspecious from Node 5
Nothing suspecious from Node 6
Nothing suspecious from Node 7
Nothing suspecious from Node 8
Nothing suspecious from Node 9

------------------------------------
Nothing suspecious from Node 0
Nothing suspecious from Node 1
Nothing suspecious from Node 2
Nothing suspecious from Node 3
Nothing suspecious from Node 4
Nothing suspecious from Node 5
Nothing suspecious from Node 6
Nothing suspecious from Node 7
Nothing suspecious from Node 8
Nothing suspecious from Node 9

------------------------------------
Nothing suspecious from Node 0
Nothing suspecious from Node 1
Nothing suspecious from Node 2
Nothing suspecious from Node 3
Nothing suspecious from Node 4
Nothing suspecious from Node 5
Nothing suspecious from Node 6
Nothing suspecious from Node 7
Nothing suspecious from Node 8
Nothing suspecious from Node 9
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .

------------------------------------
Nothing suspecious from Node 0
Suspecious behaviour from Node 1
Nothing suspecious from Node 2
Suspecious behaviour from Node 3
Nothing suspecious from Node 4
Suspecious behaviour from Node 5
Suspecious behaviour from Node 6
Nothing suspecious from Node 7
Nothing suspecious from Node 8
Nothing suspecious from Node 9

------------------------------------
Nothing suspecious from Node 0
Suspecious behaviour from Node 1
Nothing suspecious from Node 2
Suspecious behaviour from Node 3
Nothing suspecious from Node 4
Suspecious behaviour from Node 5
Suspecious behaviour from Node 6
Nothing suspecious from Node 7
Nothing suspecious from Node 8
Nothing suspecious from Node 9
```

6.2 Mitigation and Traceback

There are some of the advantages which proposed scheme offer as compared to the already available solutions. The advantages are as Follows:

1. Proposed solution works even if partially deployed across routers in the Internet.
2. It requires only a small hardware change on routers.
3. It allows a victim to identify the attack path after only a small number of packets.
4. It can scale to a large number of attackers.
5. It allows a victim to Traceback locally, without communicating with any router or ISP.

In proposed solution marking scheme allows 15 bit hash to be included in the identification field because it saves the distance field by adding it to the TTL field. With a 15 bit hash value the false positives are very less as compared to the solution which have an 11 bit hash value due to the distance field. The data provided by skitter project [39] (Number of unique routers vs. Distance) can be used to make comparisons. Figure 6.4 shows the data provided by the skitter project.

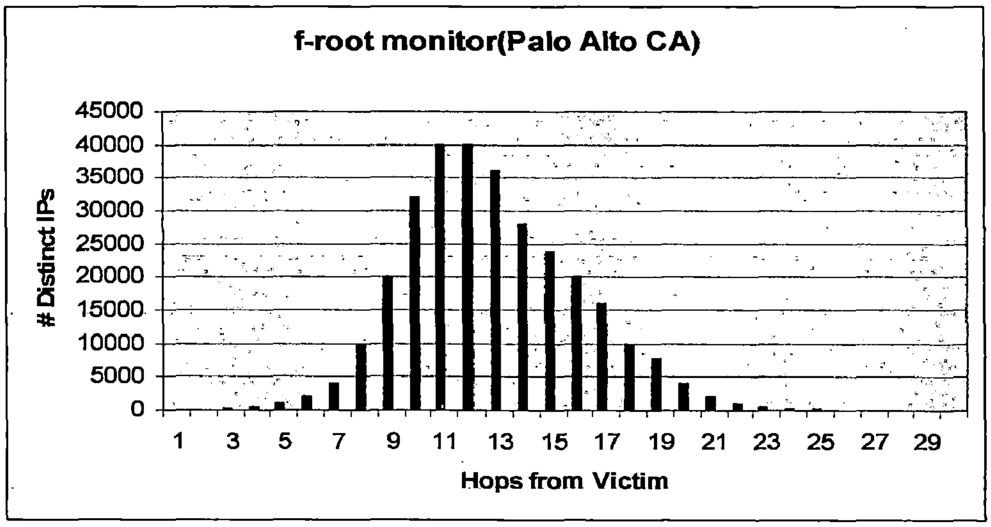


Figure 6.4: Number of unique IP addresses at each hop away from the f-root Skitter monitor

According to the data provided by the skitter project, if we calculate the false positives generated by proposed solution that will be comparatively far less then other schemes with 11 bit hash values as shown in table 6.2

Hops from Victim	**False positives**	**Any other 11 bit hash scheme**
5	0. 03	0. 48
10	0. 97	15. 6
15	0. 73	11. 7
20	0. 12	1. 95
25	0. 006	0. 09

Table 6.2: False positives

It can safely Traceback even with single packet arrival from the attacker. It can safely calculate d that is the distance of the router which marked the packet. After getting the packet the victim matches the hash of the packet with the hash values of all the routers at a distance d (As Described in 4.2.5) from victim. In the case that the victim's map contains a unique path from the reconstructed router to the victim, the victim can knows that the router, and all its downstream routers, are on the attack path.

For showing the efficiency of the system we show the congestion window of the legitimate users in following four scenarios (Figure 6.5):

1. No attack and Proposed Solution not used.
2. No Attack and Proposed Solution used.
3. Attack and Proposed solution used.
4. Attack and Proposed solution not used.

The Solution does not impact the congestion window so the "Cost" is not in the form of any reduction in congestion window but it will be the extra computing capabilities required at each router. The "Cost" also includes the hardware and software requirements required at the victim end. We need the 15 bit hash value of each IP address which is valid but this calculation can be done offline so it does not provide any delay in the processing of the packets at routers. The "Benefit" of the system is the improvement in the congestion window when the system is under attack. As the

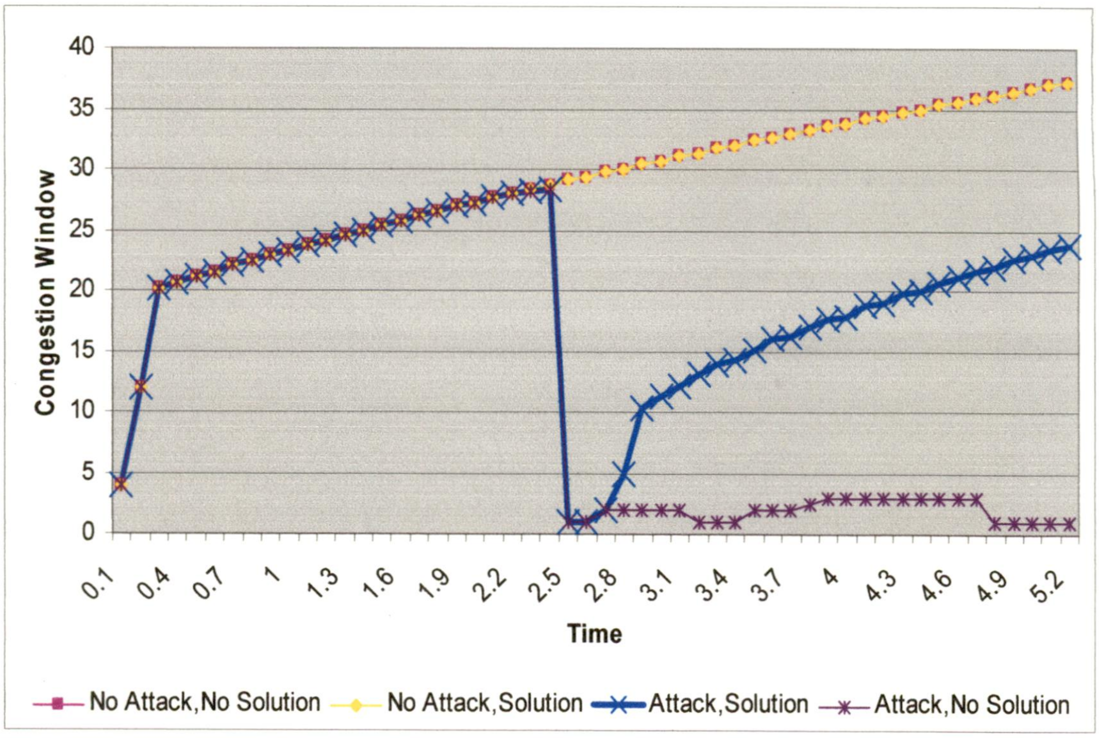


Figure 6.5: Cost-Benefit Analysis of the Mitigation Technique.

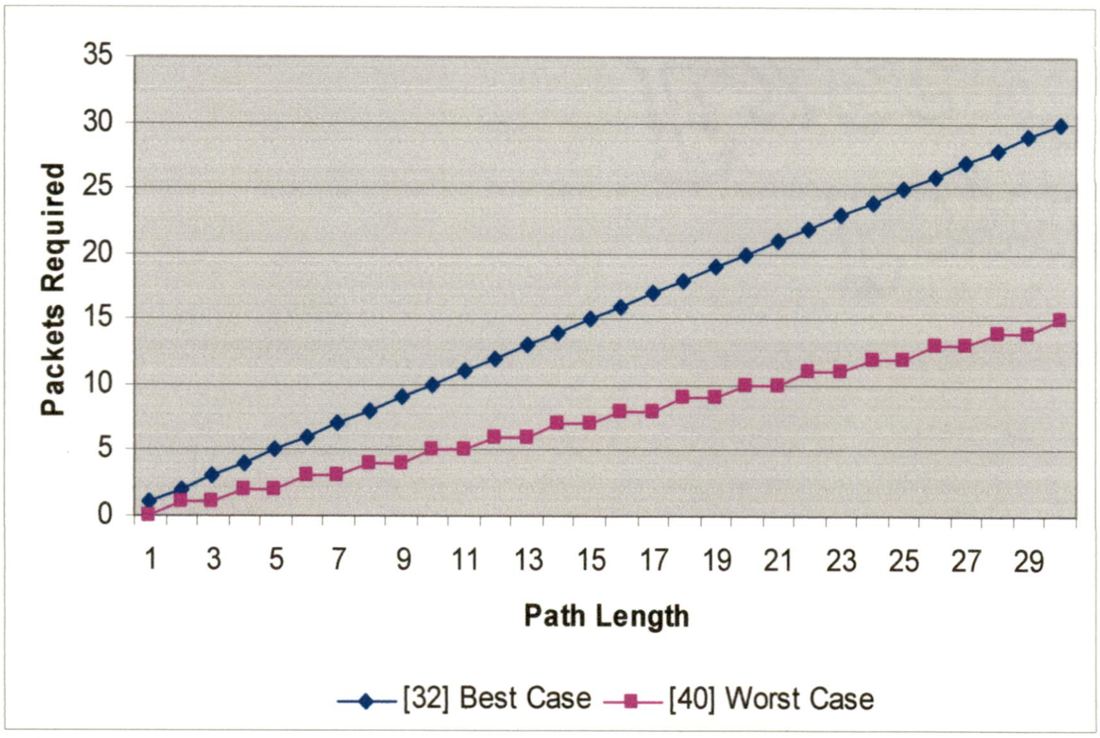


Figure 6.6: The comparison between the proposed scheme [40] and Bhawna et al. [32] framework

proposed solution is a reactive solution so as soon as attack starts the congestion window of the legitimate user falls drastically but because of the efficiency of the solution, the congestion window starts improving.

For comparison purpose, we preformed a similar simulation using Bhawna et al.[32] framework. Figure 6.6 illustrates the relation between path length and the number of packets required to reconstruct the attack paths. It is not easy to see, when reconstructing the path of the same path length, the number of packets the proposed scheme requires is approximately half of the number that the Bhawna et al.[32] framework requires. So it can reduce reconstruction time and improve the speed of traceback.

Chapter 7
Conclusions and Future Work

7.1 Conclusions

Efficiency and scalability are the key requirements in design of defense against Distributed Denial of Service Attacks. The proposed scheme provides a solution for defense against flooding-based DDoS attacks. The effectiveness of the scheme is illustrated by an appropriate simulation testbed.

The solution belongs basically to the family of PPM [35] but it improves some of the disadvantages which were earlier found in many of the techniques from the same family. It combines the two basic solutions available: Pushback and PPM. Packets are marked probalistically while on the way to the destination. These marked packets are characterized and if attack is alarmed, pushback starts to give immediate relief to the user. In the mean while the traceback module works on its way to the attacker.

As described in Section 6.2, there are five advantages offered by the proposed scheme. The advantages are as Follows:

1. The solution works even if partially deployed across routers in the Internet.
2. It requires only a small hardware change on routers.
3. It allows a victim to identify the attack path after only a small number of packets.
4. It can scale to a large number of attackers.
5. It allows a victim to Traceback locally, without communicating with any router or ISP.

We compared some of the available solutions to the proposed scheme on the basis of these properties and the table 7.1 shows the comparison. As seen the solution clearly outstands the existing works. Results in 6.5, 6.6 prove the superiority of the solution over the others.

hanism	1.	2.	3.	4.	5.
h and Cheswick[18]	X	X			
19]	X	X			X
25]		X		X	X
ich[22]	X	X			X
ric Traceback[27]	X	X			X
23]			X	X	
vna[32]	X		X		
osed Scheme[40]	X	X	X	X	X

Table 7.1: Available schemes Vs proposed scheme.

Suggestions for Future Work

'he assumption that the victim has a method available for characterizing the attacks -an be discarded and an efficient characterization technique can be added to make it a complete framework. The map reconstruction phase of the solution depends upon the third party tool which was a disadvantage in earlier strategies [25] due to the difference in distance calculations. The disadvantage is removed in the proposed scheme but still there is a scope of developing a module for map reconstruction as proposed.

Proposed scheme is a reactive technique. It mitigates the DDoS attacks very effectively but still once the victim has to face the attack and that will result in collateral damage. To reduce the effect the technique can be modified to be a proactive technique. For probabilistic packet marking the scheme used 16 bit "Identification" field. If the QoS is not the primary concern then the "Type of Service" field can be used along with the "Identification" field which will greatly improve the efficiency of the system. Moreover the Hash marked on packets can be fragmented to make scheme more robust.

REFERENCES

1. CERT Coordination Centre. Computer Crime and Security Survey, 2004. http://www.cert.org/archive/pdf/FBI2004.pdf

2. J. Mirkovic, J. Martin, and P. Reiher, "A taxonomy of DDoS attacks and DDoS defence mechanisms," *ACM SIGCOMM Computer Communications Review*, Vol. 34, No. 2, April 2004, pp. 39-53.

3. D. J. Marchette. Computer Intrusion Detection and Network Monitoring: A Statistical Viewpoint (Springer, 2001).

4. S. Shankland. "SCO Web site slammed by Net attack." ZDNet.com,May 2003. http://zdnet.com.com/2100-1105 2-999584.html.

5. R. Naraine. Massive DDoS Attack Hit DNS Root Servers, October2002. http://www.esecurityplanet.com/trends/article/0,,10751_1486981,00.html.

6. D. Moore. The spread of the code red worm (crv2). http://www.caida.org/analysis/security/codered/coderedv2 analysis.xml.

7. CERT Coordination Centre. Trends in Denial of Service Attack Technology, October 2001. http://www.cert.org/archive/pdf/DoS trends.pdf

8. .D. Dittrich. The DoS Project's trinoo distributed denial of service attack tool. http://staff.washington.edu/dittrich/misc/trinoo.analysis.

9. D. Dittrich. The Tribe Flood Network distributed denial of service attack tool. http://staff.washington.edu/dittrich/misc/tfn.analysis.txt.

10. Brown, L., Seberry, J., Pieprzyk, J., Dept. of Computer, S., Academy, A.D.F., and Centre for Computer Security, R.: 'LOKI: A cryptographic primitive for authentication and secrecy applications' (Springer, 1990. 1990)

11. D. Dittrich. The Stacheldraht distributed denial of service attack tool. http://staff.washington.edu/dittrich/misc/stacheldraht.analysis.

12. N. Long S. Dietrich and D. Dittrich. "An Analysis of the "Shaft" distributed denial of service tool." In Proceedings of LISA 2000, 2000. http://www.adelphi.edu/»spock/shaft-lisa2000.pdf.

13. CERT Coordination Center. Denial of Service Tools. http://www.cert.org/advisories/CA-1999-17.html.

14. D. Dittrich, G. Weaver, S. Dietrich, and N. Long. The mstream distributed denial of service attack tool. http://staff.washington.edu/dittrich/misc/mstream.analysis.txt.

15. D. Dittrich. Trinity distributed denial of service attack tool. http://staff.washington.edu/dittrich/misc/trinity.analysis.txt.

16. D. Dittrich. The Flitz distributed denial of service attack tool. http://staff.washington.edu/dittrich/misc/flitz.analysis.txt.

17. Arbor Networks October 2007 http://atlas-public.ec2.arbor.net/docs/BlackEnergy+DDoS+Bot+Analysis.pdf

18. Hal Burch and Bill Cheswick," Tracing anonymous packets to their approximate source" Unpublished paper, December 1999.

19. Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson., " Practical network support for IP traceback." In Proceedings of ACM SIGCOMM 2000, August 2000.

20. S. M. Bellovin. "ICMP traceback messages." Internet draft: draft-bellovin-itrace-00.txt, Mar. 2000.

21. P. Ferguson and D. Senie. "Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing agreements performance monitoring." RFC 2827, May 2000

22. Michael Goodrich., "Efficient packet marking for large-scale IP traceback" In Proceedings of the 9th ACM Conference on Computer and Communications Security, pages 117–126, November 2001

23. Alex C. Snoeren et al., "Hash-based IP traceback" In Proceedings of ACM SIGCOMM 2001, pages 3–14, August 2001.

24. J. Li, M. Sung, J. Xu, and L. Li., " Large-scale IP traceback in high-speed Internet: Practical techniques and theoretical foundation" In Proceedings of the IEEE Symposium on Security and Privacy, May 2004.

25. Dawn Song and Adrian Perrig, "Advanced and authenticated marking schemes for IP traceback". In Proceedings IEEE Infocomm 2001, April 2001.

26. K. Park and H. Lee. A proactive approach to distributed DoS attack prevention pusing route-based packet filtering. In *Proc. ACM SIGCOMM,* San Diego, CA, Aug. 2001.

27. Drew Dean, Matt Franklin, and Adam Stubblefield., "An algebraic approach to IP traceback" ACM Transactions on Information and System Security, May 2002.

28. Micah Adler et al.,"Tradeoffs in probabilistic packet marking for IP traceback." In Proceedings of 34th ACM Symposium on Theory of Computing (STOC), 2002.

29. Ratul Mahajan et al. "Controlling High Bandwidth Aggregates in the Network" ACM SIGCOMM CCR, Vol 32, No. 3, July 2002.

30. Chen, R.; Park, J.-M. "Attack diagnosis: throttling distributed denial-of-service attacks close to the attack sources" Computer Communications and Networks, 2005. ICCN 2005 Proceedings, 14th International Conference, Page(s):275 – 280, 17-19 Oct. 2005.

31. Rajesh Sharma et al. "Shared based Rate Limiting: An ISP level Solution to Deal DDoS Attacks" Annual India conference 2006, Page(s):1 – 6, Sept. 2006.

32. Bhavana Gandhi, R. C. Joshi, "An Integrated Framework for Proactive Mitigation, Characterization and Traceback of DDoS Attacks," *International Journal of Computer Science and Network Security IJCSNS*, Vol. 7, No. 3, March 2007, pp. 274-282.

33. Zhaoyang Qu; Chunfeng Huang; Ningning Liu, " A Novel Two-step Traceback Scheme for DDoS Attacks" Proceedings of the 2008 Second International Symposium on Intelligent Information Technology Application - Volume 01 2008, Page(s):879 – 883, December 20 - 22, 2008.

34. C. Jin, H. Wang, and K. G. Shin, "Hop-Count Filtering: An Effective Defense against Spoofed DoS Traffic," The Tenth ACM International Conference on Computer and Communications Security (CCS), Oct. 2003, pp. 30-41.

35. S. Savage, D. Wetherall, A. Karlin and T. Anderson, "Network Support for IP Traceback", ACM/IEEE Transactions on Networking, Vol.9, No.3, 2001, pp.226-237.

36. Lersak Limwiwatkul, Arnon Rungsawang, "Distributed Denial of Service Detection using TCP/IP Header and Traffic Measurement Analysis", International Symposium on Communications and Information Technologies(ISCIT 2004),October 26-29,2004.

37. Chen-Mou Cheng, H. T. Kung, and Koan-Sin Tan, "Use of Spectral Analysis in Defense Against DoS Attacks," *In the Proc. of Global Telecommunications*

Conference, 2002, GLOBECOM '02. IEEE, Vol. 3, Taipei, Taiwan, Nov. 2002, pp. 2143–2148.

38. NS-2 Network Simulator, available at, http://www.isi.edu/nsnam/ns/, 2009.

39. Skitter-based macroscopic topology data http://www.caida.org/tools/measurement/skitter/skitter_request.xml

40. Harsh, R. C. Joshi, "A Proposal for an Integrated Functional System for IP Traceback," *In the Proc. of* National *Journal on Emerging Trends in Software and Networking Technologies ETSNT'09*, April 2009, pp. 1-4.

LIST OF PUBLICATIONS

[1] **Harsh**, R. C. Joshi, "A Proposal for an Integrated Functional System for IP Traceback," *In the Proc. of* N*ational Journal on Emerging Trends in Software and Networking Technologies ETSNT'09*, April 2009, pp. 1-4.

[2] **Harsh**, R. C. Joshi, "DDoS Attacks: Detection and Prevention Schemes," *In the Proc. of International Conference on Art, Science, Management and Engineering,* April 2009.

APPENDIX

SOURCE CODE LISTING

#final.tcl

```
#Create the simulator instance
set ns [new Simulator]
$ns color 1 Red
$ns color 3 Red
$ns color 5 Red
$ns color 6 Red
$ns color 0 Blue
$ns color 2 Blue
$ns color 4 Blue
$ns color 7 Blue
$ns color 8 Blue
$ns color 9 Blue

#Open the NAM trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Open the trace file
set tf [open out_expo.tr w]
$ns trace-all $tf

proc finish {} {
        global ns nf
        $ns flush-trace
        close $nf
        exec /home/Harsh/Desktop/ns-allinone-2.33/nam-1.13/nam
/home/Harsh/Desktop/ns-allinone-2.33/ns-2.33/out.nam
        exit 0
}

#create 28 nodes
for {set i 0} {$i<28} {incr i} {
        set m($i) [$ns node]
}

#nodes forming the bottleneck link
set n2 [$ns node]
set n3 [$ns node]

#Create links between the nodes
$ns simplex-link $m(0) $m(10) 5Mb 10ms DropTail
$ns simplex-link $m(10) $m(0) 5Mb 10ms DropTail
$ns simplex-link $m(1) $m(10) 5Mb 10ms DropTail
$ns simplex-link $m(10) $m(1) 5Mb 10ms DropTail
$ns simplex-link $m(10) $m(16) 5Mb 10ms DropTail
$ns simplex-link $m(16) $m(10) 5Mb 10ms DropTail
$ns simplex-link $m(2) $m(11) 5Mb 10ms DropTail
$ns simplex-link $m(11) $m(2) 5Mb 10ms DropTail
$ns simplex-link $m(11) $m(12) 5Mb 10ms DropTail
```

```
$ns simplex-link $m(12) $m(11) 5Mb 10ms DropTail
$ns duplex-link $m(12) $m(17) 5Mb 10ms DropTail
$ns simplex-link $m(3) $m(12) 5Mb 10ms DropTail
$ns simplex-link $m(12) $m(3) 5Mb 10ms DropTail
$ns simplex-link $m(4) $m(13) 5Mb 10ms DropTail
$ns simplex-link $m(13) $m(4) 5Mb 10ms DropTail
$ns simplex-link $m(13) $m(18) 5Mb 10ms DropTail
$ns simplex-link $m(18) $m(13) 5Mb 10ms DropTail
$ns simplex-link $m(5) $m(13) 5Mb 10ms DropTail
$ns simplex-link $m(13) $m(5) 5Mb 10ms DropTail
$ns simplex-link $m(6) $m(14) 5Mb 10ms DropTail
$ns simplex-link $m(14) $m(6) 5Mb 10ms DropTail
$ns duplex-link $m(14) $m(13) 5Mb 10ms DropTail
$ns simplex-link $m(7) $m(19) 5Mb 10ms DropTail
$ns simplex-link $m(19) $m(7) 5Mb 10ms DropTail
$ns simplex-link $m(8) $m(15) 5Mb 10ms DropTail
$ns simplex-link $m(15) $m(8) 5Mb 10ms DropTail
$ns simplex-link $m(15) $m(19) 5Mb 10ms DropTail
$ns simplex-link $m(19) $m(15) 5Mb 10ms DropTail
$ns simplex-link $m(9) $m(15) 5Mb 10ms DropTail
$ns simplex-link $m(15) $m(9) 5Mb 10ms DropTail
$ns duplex-link $m(16) $m(20) 5Mb 10ms DropTail
$ns duplex-link $m(17) $m(21) 5Mb 10ms DropTail
$ns duplex-link $m(18) $m(22) 5Mb 10ms DropTail
$ns simplex-link $m(19) $m(22) 5Mb 10ms DropTail
$ns simplex-link $m(22) $m(19) 5Mb 10ms DropTail
$ns duplex-link $m(20) $m(23) 5Mb 10ms DropTail
$ns duplex-link $m(21) $m(24) 5Mb 10ms DropTail
$ns duplex-link $m(22) $m(25) 5Mb 10ms DropTail
$ns duplex-link $m(23) $m(26) 5Mb 10ms DropTail
$ns duplex-link $m(24) $m(26) 5Mb 10ms DropTail
$ns duplex-link $m(25) $m(27) 5Mb 10ms DropTail
$ns duplex-link $m(26) $n2 5Mb 10ms DropTail
$ns duplex-link $m(27) $n2 5Mb 10ms DropTail
$ns simplex-link $n2 $n3 20Mb 10ms DropTail
$ns simplex-link $n3 $n2 20Mb 10ms DropTail
$ns queue-limit $n2 $n3 150

#Set up the orientation
$ns duplex-link-op $m(0) $m(10) orient left-up
$ns duplex-link-op $m(1) $m(10) orient left-down
$ns duplex-link-op $m(10) $m(16) orient left-up
$ns duplex-link-op $m(2) $m(11) orient left-up
$ns duplex-link-op $m(3) $m(12) orient left-down
$ns duplex-link-op $m(4) $m(13) orient left-up
$ns duplex-link-op $m(5) $m(13) orient left-center
$ns duplex-link-op $m(6) $m(14) orient left-down
$ns duplex-link-op $m(7) $m(19) orient left-up
$ns duplex-link-op $m(8) $m(15) orient left-center
$ns duplex-link-op $m(9) $m(15) orient left-down
```

```
$ns duplex-link-op $n2 $n3 orient right-center

#Set up 6 TCP connections

set tcp0 [new Agent/TCP/Newreno]
$ns attach-agent $m(0) $tcp0
set sink0 [new Agent/TCPSink]
$ns attach-agent $n3 $sink0
$tcp0 set fid_ 0


set tcp1 [new Agent/TCP/Newreno]
$ns attach-agent $m(2) $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $n3 $sink1
$tcp1 set fid_ 2

set tcp2 [new Agent/TCP/Newreno]
$ns attach-agent $m(4) $tcp2
set sink2 [new Agent/TCPSink]
$ns attach-agent $n3 $sink2
$tcp2 set fid_ 4

set tcp3 [new Agent/TCP/Newreno]
$ns attach-agent $m(7) $tcp3
set sink3 [new Agent/TCPSink]
$ns attach-agent $n3 $sink3
$tcp3 set fid_ 7

set tcp4 [new Agent/TCP/Newreno]
$ns attach-agent $m(8) $tcp4
set sink4 [new Agent/TCPSink]
$ns attach-agent $n3 $sink4
$tcp4 set fid_ 8

set tcp5 [new Agent/TCP/Newreno]
$ns attach-agent $m(9) $tcp5
set sink5 [new Agent/TCPSink]
$ns attach-agent $n3 $sink5
$tcp5 set fid_ 9

#create FTP traffic sources

set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0

set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
```

```
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2

set ftp3 [new Application/FTP]
$ftp3 attach-agent $tcp3

set ftp4 [new Application/FTP]
$ftp4 attach-agent $tcp4

set ftp5 [new Application/FTP]
$ftp5 attach-agent $tcp5

#Setup 4 UDP connections

set udp0 [new Agent/UDP]
$ns attach-agent $m(1) $udp0
$udp0 set fid_ 1

set udp1 [new Agent/UDP]
$ns attach-agent $m(3) $udp1
$udp1 set fid_ 3

set udp2 [new Agent/UDP]
$ns attach-agent $m(5) $udp2
$udp2 set fid_ 5

set udp3 [new Agent/UDP]
$ns attach-agent $m(6) $udp3
$udp3 set fid_ 6

# Setup Null Agents

set null0 [new Agent/Null]
$ns attach-agent $n3 $null0

set null1 [new Agent/Null]
$ns attach-agent $n3 $null1

set null2 [new Agent/Null]
$ns attach-agent $n3 $null2

set null3 [new Agent/Null]
$ns attach-agent $n3 $null3

#Setup CBR over UDP connections

set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
$cbr0 set type_ CBR
$cbr0 set packetSize_ 1000
```

```
$cbr0 set rate_ 3Mb

set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
$cbr1 set type_ CBR
$cbr1 set packetSize_ 1000
$cbr1 set rate_ 3Mb

set cbr2 [new Application/Traffic/CBR]
$cbr2 attach-agent $udp2
$cbr2 set type_ CBR
$cbr2 set packetSize_ 1000
$cbr2 set rate_ 3Mb

set cbr3 [new Application/Traffic/CBR]
$cbr3 attach-agent $udp3
$cbr3 set type_ CBR
$cbr3 set packetSize_ 1000
$cbr3 set rate_ 3Mb

#Set the IFS Agents

set ifs0 [new Agent/Ifs]
$ns attach-agent $m(10) $ifs0

set ifs1 [new Agent/Ifs]
$ns attach-agent $m(10) $ifs1


set ifs2 [new Agent/Ifs]
$ns attach-agent $m(11) $ifs2

set ifs3 [new Agent/Ifs]
$ns attach-agent $m(12) $ifs3

set ifs4 [new Agent/Ifs]
$ns attach-agent $m(13) $ifs4

set ifs5 [new Agent/Ifs]
$ns attach-agent $m(13) $ifs5

set ifs6 [new Agent/Ifs]
$ns attach-agent $m(14) $ifs6

set ifs7 [new Agent/Ifs]
$ns attach-agent $m(19) $ifs7

set ifs8 [new Agent/Ifs]
$ns attach-agent $m(15) $ifs8
```

```
set ifs9 [new Agent/Ifs]
$ns attach-agent $m(15) $ifs9

#Make connectivity between TCP and IFS agents

$ns connect $tcp0 $ifs0
$ifs0 actual $tcp0
$ns connect $ifs0 $sink0

$ns connect $tcp1 $ifs2
$ifs2 actual $tcp1
$ns connect $ifs2 $sink1

$ns connect $tcp2 $ifs4
$ifs4 actual $tcp2
$ns connect $ifs4 $sink2

$ns connect $tcp3 $ifs7
$ifs7 actual $tcp3
$ns connect $ifs7 $sink3

$ns connect $tcp4 $ifs8
$ifs8 actual $tcp4
$ns connect $ifs8 $sink4

$ns connect $tcp5 $ifs9
$ifs9 actual $tcp5
$ns connect $ifs9 $sink5

#Make connectivity between UDP and IFS agents
$ns connect $udp0 $ifs1
$ifs1 destination $null0
$ns connect $ifs1 $null0

$ns connect $udp1 $ifs3
$ifs3 destination $null1
$ns connect $ifs3 $null1

$ns connect $udp2 $ifs5
$ifs5 destination $null2
$ns connect $ifs5 $null2

$ns connect $udp3 $ifs6
$ifs6 destination $null3
$ns connect $ifs6 $null3

#Attach A Flow Monitor
set r1fm [$ns makeflowmon SrcDestFid]
$ns attach-fmon [$ns link $n2 $n3] $r1fm
set flowdesc [open mon.tr w]
```

```
$r1fm attach $flowdesc
$r1fm set pdrops_
set fcl [$r1fm classifier]

#connect queue monitors for characterization
set qmon0 [$ns monitor-queue $m(16) $m(10) mff]
set qmon1 [$ns monitor-queue $m(10) $m(1) mff]
set qmon2 [$ns monitor-queue $m(12) $m(11) mff]
set qmon3 [$ns monitor-queue $m(12) $m(3) mff]
set qmon4 [$ns monitor-queue $m(18) $m(13) mff]
set qmon5 [$ns monitor-queue $m(13) $m(5) mff]
set qmon6 [$ns monitor-queue $m(14) $m(6) mff]
set qmon7 [$ns monitor-queue $m(22) $m(19) mff]
set qmon8 [$ns monitor-queue $m(19) $m(15) mff]
set qmon9 [$ns monitor-queue $m(19) $m(15) mff]

for {set i 0} {$i < 2} {incr i} {
        set x($i) 0.00
}

#procedure for passing parameters of queue monitor to flowmonitor
proc pass {} {
        global qmon0 qmon1 qmon2 qmon3 qmon4 qmon5 qmon6 qmon7 qmon8
qmon9 ns x r1fm
        set x(0) [$qmon0 set parrivals_]
        set x(1) [$qmon1 set parrivals_]
        set x(2) [$qmon2 set parrivals_]
        set x(3) [$qmon3 set parrivals_]
        set x(4) [$qmon4 set parrivals_]
        set x(5) [$qmon5 set parrivals_]
        set x(6) [$qmon6 set parrivals_]
        set x(7) [$qmon7 set parrivals_]
        set x(8) [$qmon8 set parrivals_]
        set x(9) [$qmon9 set parrivals_]
        $r1fm passing $x(0) $x(1) $x(2) $x(3) $x(4) $x(5) $x(6) $x(7) $x(8) $x(9)
}

#queue monitors for udp traffic measurement becoz of flaw in ns2
set udpmon0 [$ns monitor-queue $m(1) $m(10) mff]
set udpmon1 [$ns monitor-queue $m(3) $m(12) mff]
set udpmon2 [$ns monitor-queue $m(5) $m(13) mff]
set udpmon3 [$ns monitor-queue $m(6) $m(14) mff]

proc udppass {} {
        global udpmon0 udpmon1 udpmon2 udpmon3 ns r1fm
        set u0 [$udpmon0 set parrivals_]
        set u1 [$udpmon1 set parrivals_]
        set u2 [$udpmon2 set parrivals_]
        set u3 [$udpmon3 set parrivals_]
        $r1fm passudp $u0 $u1 $u2 $u3
```

```
}

#procedure to characterize flows
proc sample-flow {} {
        global ns r1fm
        set time2 0.2
        udppass
        pass
        $r1fm dump
        set now2 [$ns now]
        if { $now2>=5.5 } {
        finish
        exit 0
        } else {
        $ns at [expr $now2+$time2] "sample-flow"
        }

}

#Pushback
proc push-back {} {
                global ns r1fm ifs0 ifs1 ifs2 ifs3 ifs4 ifs5 ifs6 ifs7 ifs8 ifs9
                set a0 [$r1fm set attack0_]
                set a1 [$r1fm set attack1_]
                set a2 [$r1fm set attack2_]
                set a3 [$r1fm set attack3_]
                set a4 [$r1fm set attack4_]
                set a5 [$r1fm set attack5_]
                set a6 [$r1fm set attack6_]
                set a7 [$r1fm set attack7_]
                set a8 [$r1fm set attack8_]
                set a9 [$r1fm set attack9_]
                if { $a0 == 1 } {
                        puts "node 0 is attacking"
                        $ifs0 stop
                }

                if { $a1 == 1 } {
                        puts "node 1 is attacking"
                        $ifs1 stop
                }

                if { $a2 == 1 } {
                        puts "node 2 is attacking"
                        $ifs2 stop
                }

                if { $a3 == 1 } {
                                puts "node 3 is attacking"
                                $ifs3 stop
                }

                if { $a4 == 1 } {
                        puts "node 4 is attacking"
                        $ifs4 stop
                }

                if { $a5 == 1 } {
                        puts "node 5 is attacking"
                        $ifs5 stop
                }

                if { $a6 == 1 } {
                        puts "node 6 is attacking"
                        $ifs6 stop
                }

                if { $a7 == 1 } {
                        puts "node 7 is attacking"
                        $ifs7 stop
                }

                if { $a8 == 1 } {
                        puts "node 8 is attacking"
                        $ifs8 stop
                }

                if { $a9 == 1 } {
                        puts "node 9 is attacking"
                        $ifs9 stop
                }

                set time1 0.2
                set now1 [$ns now]
                $ns at [expr $now1+$time1] "push-back"
}

#Schedule events
$ns at 0.1 "$m(0) label \"Legitimate\""
$ns at 0.1 "$m(1) label \"Attack\""
$ns at 0.1 "$m(2) label \"Legitimate\""
$ns at 0.1 "$m(3) label \"Attack\""
$ns at 0.1 "$m(4) label \"Legitimate\""
$ns at 0.1 "$m(5) label \"Attack\""
$ns at 0.1 "$m(6) label \"Attack\""
$ns at 0.1 "$m(7) label \"Legitimate\""
$ns at 0.1 "$m(8) label \"Legitimate\""
$ns at 0.1 "$m(9) label \"Legitimate\""
$ns at 0.1 "$m(10) label \"IFS\""
$ns at 0.1 "$m(11) label \"IFS\""
```

```
$ns at 0.1 "$m(12) label \"IFS\""
$ns at 0.1 "$m(13) label \"IFS\""
$ns at 0.1 "$m(14) label \"IFS\""
$ns at 0.1 "$m(19) label \"IFS\""
$ns at 0.1 "$m(15) label \"IFS\""
$ns at 0.1 "$n3 label \"Sink\""

$ns at 2.0 "$cbr0 start"
$ns at 5.0 "$cbr0 stop"
$ns at 2.0 "$cbr1 start"
$ns at 5.0 "$cbr1 stop"
$ns at 2.0 "$cbr2 start"
$ns at 5.0 "$cbr2 stop"
$ns at 2.0 "$cbr3 start"
$ns at 5.0 "$cbr3 stop"

$ns at 0.0 "$ftp0 start"
$ns at 5.0 "$ftp0 stop"
$ns at 0.0 "$ftp1 start"
$ns at 5.0 "$ftp1 stop"
$ns at 0.0 "$ftp2 start"
$ns at 5.0 "$ftp2 stop"
$ns at 0.0 "$ftp3 start"
$ns at 5.0 "$ftp3 stop"
$ns at 0.0 "$ftp4 start"
$ns at 5.0 "$ftp4 stop"
$ns at 0.0 "$ftp5 start"
$ns at 5.0 "$ftp5 stop"

proc clean-up {} {
puts "in cleanup..."
global ns m(0) m(1) m(2) m(3) m(4) m(5) m(6) m(7) m(8) m(9)
$ns detach-agent $m(0) $tcp0
$ns detach-agent $m(2) $tcp1
$ns detach-agent $m(4) $tcp2
$ns detach-agent $m(7) $tcp3
$ns detach-agent $m(8) $tcp4
$ns detach-agent $m(9) $tcp5
$ns detach-agent $m(1) $udp0
$ns detach-agent $m(3) $udp1
$ns detach-agent $m(5) $udp2
$ns detach-agent $m(6) $udp3
}

$ns at 0.1 "sample-flow"
$ns at 0.1 "push-back"
$ns at 5.2 "finish"
$ns run
```

#FLOWMON.H

```
class FlowMon : public EDQueueMonitor {
public:
        FlowMon();
        void in(Packet*);        // arrivals
        void out(Packet*);       // departures
        void drop(Packet*);  // all drops (incl
        void edrop(Packet*); // "early" drops
        void mon_edrop(Packet*);      // " monitored early" drops
        int command(int argc, const char*const* argv);

        //added by ratul
        void setClassifier(Classifier * classifier) {
          classifier_ = classifier;
        }

        Flow * find(Packet* p) {
                return (Flow *)classifier_->find(p);
        }
        int value_;//ADDED BY HARSH
        int attack0_;
        int attack1_;
        int attack2_;
        int attack3_;
        int attack4_;
        int attack5_;
        int attack6_;
        int attack7_;
        int attack8_;
        int attack9_;
protected:
        void    dumpflows();
        void    dumpflow(Tcl_Channel, Flow*);
        void    fformat(Flow*);
        char*   flow_list();

        Classifier*     classifier_;
        Tcl_Channel     channel_;

        int enable_in_;          // enable per-flow arrival state
        int enable_out_;         // enable per-flow depart state
        int enable_drop_;        // enable per-flow drop state
        int enable_edrop_;       // enable per-flow edrop state
        int enable_mon_edrop_; // enable per-flow mon_edrop state

        //an excessive high value for large simulations using flow monitor
        char    wrk_[65536]; // big enough to hold flow list
        //Added By Harsh
        int fd[10][3];  //STATISTICS FOR EACH FLOW;
        int x[10];      //STATISTICS PASSED BY TCL FILE
```

#FLOWMON.CC

```
FlowMon::FlowMon() : classifier_(NULL), channel_(NULL),
        enable_in_(1), enable_out_(1), enable_drop_(1), enable_edrop_(1),
enable_mon_edrop_(1),value_(0),attack0_(0),attack1_(0),attack2_(0),attack3_(0),atta
ck4_(0),attack5_(0),attack6_(0),attack7_(0),attack8_(0),attack9_(0)
{
        bind_bool("enable_in_", &enable_in_);
        bind_bool("enable_out_", &enable_out_);
        bind_bool("enable_drop_", &enable_drop_);
        bind_bool("enable_edrop_", &enable_edrop_);
        bind("value_",&value_);//TESTING FOR CONNECTIVITY
        bind("attack0_",&attack0_);
        bind("attack1_",&attack1_);
        bind("attack2_",&attack2_);
        bind("attack3_",&attack3_);
        bind("attack4_",&attack4_);
        bind("attack5_",&attack5_);
        bind("attack6_",&attack6_);
        bind("attack7_",&attack7_);
        bind("attack8_",&attack8_);
        bind("attack9_",&attack9_);

        //one entry for each flow
        for(int i=0;i<=9;i++)
        {
                fd[i][0]=i;             //flowid
                fd[i][1]=0;             //pdrops
                fd[i][2]=0;             //parrivals
        }
        for(int i=0;i<=9;i++)
        {
                flags[i]=0;             //flags intialized to zero

        }
        for(int i=0;i<=9;i++)
        {
                x[i]=0;         //pkts sent which is to be compared with parrivals
        }
}

void FlowMon::in(Packet *p)
{
        Flow* desc;
        EDQueueMonitor::in(p);
        if (!enable_in_)
                return;
        if ((desc = ((Flow *)classifier_->find(p))) != NULL) {
                desc->setfields(p);
                desc->in(p);
        }
}

void FlowMon::out(Packet *p)
{
Flow* desc;
        EDQueueMonitor::out(p);
        if (!enable_out_)
                return;
        if ((desc = ((Flow*)classifier_->find(p))) != NULL) {
                desc->setfields(p);
                desc->out(p);
        }
}

void FlowMon::drop(Packet *p)
{
        Flow* desc;
        EDQueueMonitor::drop(p);
        if (!enable_drop_)
                return;
        if ((desc = ((Flow*)classifier_->find(p))) != NULL) {
                desc->setfields(p);
                desc->drop(p);
        }
}

void FlowMon::edrop(Packet *p)
{
        Flow* desc;
        EDQueueMonitor::edrop(p);
        if (!enable_edrop_)
                return;
        if ((desc = ((Flow*)classifier_->find(p))) != NULL) {
                desc->setfields(p);
                desc->edrop(p);
        }
}

//added for monitored early drops - ratul
void
FlowMon::mon_edrop(Packet *p)
{
        Flow* desc;
        EDQueueMonitor::mon_edrop(p);
        if (!enable_mon_edrop_)
                return;
        if ((desc = ((Flow*)classifier_->find(p))) != NULL) {
                desc->setfields(p);
                desc->mon_edrop(p);
        }
}

void FlowMon::dumpflows()
{
        register int i, j = classifier_->maxslot();
        Flow* f;
        for (i = 0; i <= j; i++) {
                if ((f = (Flow*)classifier_->slot(i)) != NULL)
                {
                        fd[i][0]=f->flowid(); //ADDED BY HARSH
                        fd[i][1]=f->pdrops();
                        fd[i][2]=f->parrivals();
                        //printf("for flow id %d the packet arrivals are %d
\n",fd[i][0],fd[i][2]);
                        dumpflow(channel_, f);
                }
        }

        if(j<10)
        {
                for(int l=j+1;l<=9;l++)                 //for currently inactive flows
                {
                        fd[l][0]=l;
                        fd[l][1]=0;
                        fd[l][2]=0;
                }
        }

}

char* FlowMon::flow_list()
{
        register const char* z;
        register int i, j = classifier_->maxslot();
        Flow* f;
        register char* p = wrk_;
        register char* q;
        q = p + sizeof(wrk_) - 2;
        *p = '\0';

        for (i = 0; i <= j; i++) {
                if ((f = (Flow*)classifier_->slot(i)) != NULL) {

                        z = f->name();
                        while (*z && p < q) {

                                *p++ = *z++;
```

```
                    }
                    *p++ = ' ';
                }
                if (p >= q) {
                    fprintf(stderr, "FlowMon:: flow list exceeded working
buffer\n");
                    fprintf(stderr, "\t  recompile ns with larger FlowMon::wrk_[]
array\n");
                    exit (1);
                }
        }
        if (p != wrk_)
                *--p = '\0';
        return (wrk_);
}

void FlowMon::fformat(Flow* f)
{
        double now = Scheduler::instance().clock();
#if defined(HAVE_INT64)
        sprintf(wrk_, "%8.3f %d %d %d %d %d %d " STRTOI64_FMTSTR " "
STRTOI64_FMTSTR " %d %d " STRTOI64_FMTSTR " " STRTOI64_FMTSTR "
%d %d %d %d %d %d %d %d %d",
#else /* no 64-bit int */
        sprintf(wrk_, "%8.3f %d %d %d %d %d %d %d %d %d %d %d %d %d
%d %d %d %d %d %d %d",
#endif
                now,            // 1: time
                f->flowid(),    // 2: flowid
                0,              // 3: category
                f->ptype(),     // 4: type (from common header)
                f->flowid(),    // 5: flowid (formerly class)
                f->src(),       // 6: sender
                f->dst(),       // 7: receiver
                f->parrivals(), // 8: arrivals this flow (pkts)
                f->barrivals(), // 9: arrivals this flow (bytes)
                f->epdrops(),   // 10: early drops this flow (pkts)
                f->ebdrops(),   // 11: early drops this flow (bytes)
                parrivals(),    // 12: all arrivals (pkts)
                barrivals(),    // 13: all arrivals (bytes)
                epdrops(),      // 14: total early drops (pkts)
                ebdrops(),      // 15: total early drops (bytes)
                pdrops(),       // 16: total drops (pkts)
                bdrops(),       // 17: total drops (bytes)
                f->pdrops(),    // 18: drops this flow (pkts) [includes edrops]
                f->bdrops(),    // 19: drops this flow (bytes) [includes edrops]
                f->qs_pkts(),   // 20: Quick-Start packets this flow
                f->qs_bytes(),  // 21: Quick-Start bytes this flow
                f->qs_drops()  // 22: dropped Quick-Start pkts this flow
        );
}

void FlowMon::dumpflow(Tcl_Channel tc, Flow* f)
{
        //ADDED BY HARSH
        for(int i=0;i<=9;i++)
        {
                if(x[i]==0)
                        x[i]=1;
                if((fd[i][2]/x[i])<=10)
                        printf("Nothing suspecious from Node %d \n",i);
                else
                {
                        printf("Suspecious behaviour from Node %d \n",i);
                        value_=1;
                        flags[i]=1;
                }
        }
        printf("\n-------------------------------------\n");
        //here is the imp for push back mark the attackers
        if(flags[0]==1)
                attack0_=1;
        if(flags[1]==1)
                attack1_=1;
        if(flags[2]==1)
                attack2_=1;
        if(flags[3]==1)
                attack3_=1;
        if(flags[4]==1)
                attack4_=1;
        if(flags[5]==1)
                attack5_=1;
        if(flags[6]==1)
                attack6_=1;
        if(flags[7]==1)
                attack7_=1;
        if(flags[8]==1)
                attack8_=1;
        if(flags[9]==1)
                attack9_=1;
        fformat(f);
        if (tc != 0) {
                int n = strlen(wrk_);
                wrk_[n++] = '\n';
                wrk_[n] = '\0';
                (void)Tcl_Write(tc, wrk_, n);
                wrk_[n-1] = '\0';
        }
```

```
}

int FlowMon::command(int argc, const char*const* argv)
{
        Tcl& tcl = Tcl::instance();
        if (argc == 2) {
                if (strcmp(argv[1], "classifier") == 0) {
                        if (classifier_)
                                tcl.resultf("%s", classifier_->name());
                        else
                                tcl.resultf("");
                        return (TCL_OK);
                }
                if (strcmp(argv[1], "dump") == 0) {
                        dumpflows();
                        return (TCL_OK);
                }


                if (strcmp(argv[1], "flows") == 0) {
                        //      printf("command says gimme flow list\n");
                        tcl.result(flow_list());
                        return (TCL_OK);
                }
        } else if (argc == 3) {
                if (strcmp(argv[1], "classifier") == 0) {
                        classifier_ = (Classifier*)
                                TclObject::lookup(argv[2]);
                        if (classifier_ == NULL)
                                return (TCL_ERROR);
                        return (TCL_OK);
                }
                if (strcmp(argv[1], "attach") == 0) {
                        int mode;
                        const char* id = argv[2];
                        channel_ = Tcl_GetChannel(tcl.interp(),
                                (char*) id, &mode);
                        if (channel_ == NULL) {
                                tcl.resultf("FlowMon (%s): can't attach %s for writing",
                                        name(), id);
                                return (TCL_ERROR);
                        }

                        return (TCL_OK);
                }
        } else if (argc == 12) {                        //ADDED BY HARSH
                if (strcmp(argv[1],"passing") == 0)
                {
                        //printf("parameter passed\n");
                        x[0] = atoi(argv[2]);
                x[1] = atoi(argv[3]);
                x[2] = atoi(argv[4]);
                x[3] = atoi(argv[5]);
                x[4] = atoi(argv[6]);
                x[5] = atoi(argv[7]);
                x[6] = atoi(argv[8]);
                x[7] = atoi(argv[9]);
                x[8] = atoi(argv[10]);
                x[9] = atoi(argv[11]);
                //printf("values of outgoing packets %d %d \n",x[0],x[1]);
            }
            return (TCL_OK);
        }else if (argc == 6){
            if (strcmp(argv[1],"passudp") == 0)
            {
                fd[1][2]=atoi(argv[2]);
                fd[3][2]=atoi(argv[3]);
                fd[5][2]=atoi(argv[4]);
                fd[6][2]=atoi(argv[5]);
            }
            return (TCL_OK);
        }
        return (EDQueueMonitor::command(argc, argv));
}
```

#IFS.H

```
/*
*File:Header for a new 'IFS' agent class for the network simulator
*/
#ifndef ns_ifs_h
#define ns_ifs_h

#include "agent.h"
#include "tclcl.h"
#include "packet.h"
#include "address.h"
#include "ip.h"
#include "tcp.h"
#include "object.h"

class IfsAgent:public Agent {
public:
        IfsAgent();
        virtual int command(int argc,const char* const* argv);
        virtual void recv(Packet*,Handler*);
        bool flag_;
        NsObject* ifstarget_;
        NsObject* ifsdest_;
        int stop_;

};

#endif
```

#IFS.CC

```
#include "ifs.h"
#include <stdio.h>
static class IfsClass : public TclClass {
public:
        IfsClass() : TclClass("Agent/Ifs") {}
        TclObject* create(int, const char*const*) {
                return (new IfsAgent());
        }
} class_ifs;


IfsAgent::IfsAgent() : Agent(PT_NTYPE),flag_(0),stop_(1)
{

}

void IfsAgent::recv(Packet* pkt, Handler* h)
{
 packet_t recvtype=HDR_CMN(pkt)->ptype_;
 if (recvtype==0 && stop_==1) {                    //client packet received

        // Access the IP header for the received packet:
        hdr_ip* hdrip = hdr_ip::access(pkt);
        hdr_cmn* hdrcmn = hdr_cmn::access(pkt);
        int x,y;
        //modify the ttl of the pkt:
        x=hdrcmn->addr() & 07fff;
        y=hdrip->ttl() & 8000 ;

        if(flag_==0)
        {
        hdrip->ttl()=256;
        flag_=1;
        }

        hdrip->daddr()=daddr();
        send(pkt,0);

 } else if (recvtype==0 && stop_==0)
        {
                Packet::free(pkt);
        } else if (recvtype==2 && stop_==1){
        ifsdest_->recv(pkt,h);


}else if (recvtype==2 && stop_==0)
        {
                Packet::free(pkt);
```

```
          }
          else{//ACK packet received
          hdr_ip* hdrip = hdr_ip::access(pkt);
          ifstarget_->recv(pkt,h);
 }

}

int IfsAgent::command(int argc, const char*const* argv)
{
 if (argc ==3) {
          if (strcmp(argv[1], "actual") == 0) {
          ifstarget_=(NsObject*)TclObject::lookup(argv[2]);
          return (TCL_OK);
          }
          if (strcmp(argv[1],"destination") == 0)
          {
            ifsdest_=(NsObject*)TclObject::lookup(argv[2]);
           return (TCL_OK);
          }
 } else if (argc==2) {
          if (strcmp(argv[1], "stop") == 0) {
                    stop_=0;
          }
      return (TCL_OK);

}

  // If the command hasn't been processed by IfsAgent()::command,

 // call the command() function for the base class

 return (Agent::command(argc, argv));

}
```