# EARLY DETECTION AND PREVENTION OF DENIAL-OF-SERVICE ATTACKS

By

**R.RANJITH**

**Reg. No: 0820108013**

**of**

**KUMARAGURU COLLEGE OF TECHNOLOGY**

**(An Autonomous Institution Affiliated to Anna university, Coimbatore )**

**COIMBATORE – 641 006**

**A PROJECT REPORT**

**Submitted to the**

**FACULTY OF INFORMATION AND COMMUNICATION ENGINEERING**

*In partial fulfillment of the requirements for the award of the degree of*

**MASTER OF ENGINEERING IN**

**COMPUTER SCIENCE AND ENGINEERING**

**MAY, 2010**

## BONAFIDE CERTIFICATE

Certified that this project report titled "**Early Detection and Prevention of Denial-of-Service Attacks**" is the bonafide work of **R.Ranjith (0820108013)** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report of dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

[illegible]

**GUIDE**

**(Dr.A.MUTHUKUMAR)**

[illegible]

**HEAD OF THE DEPARTMENT**

**(Dr.S.THANGASAMY)**

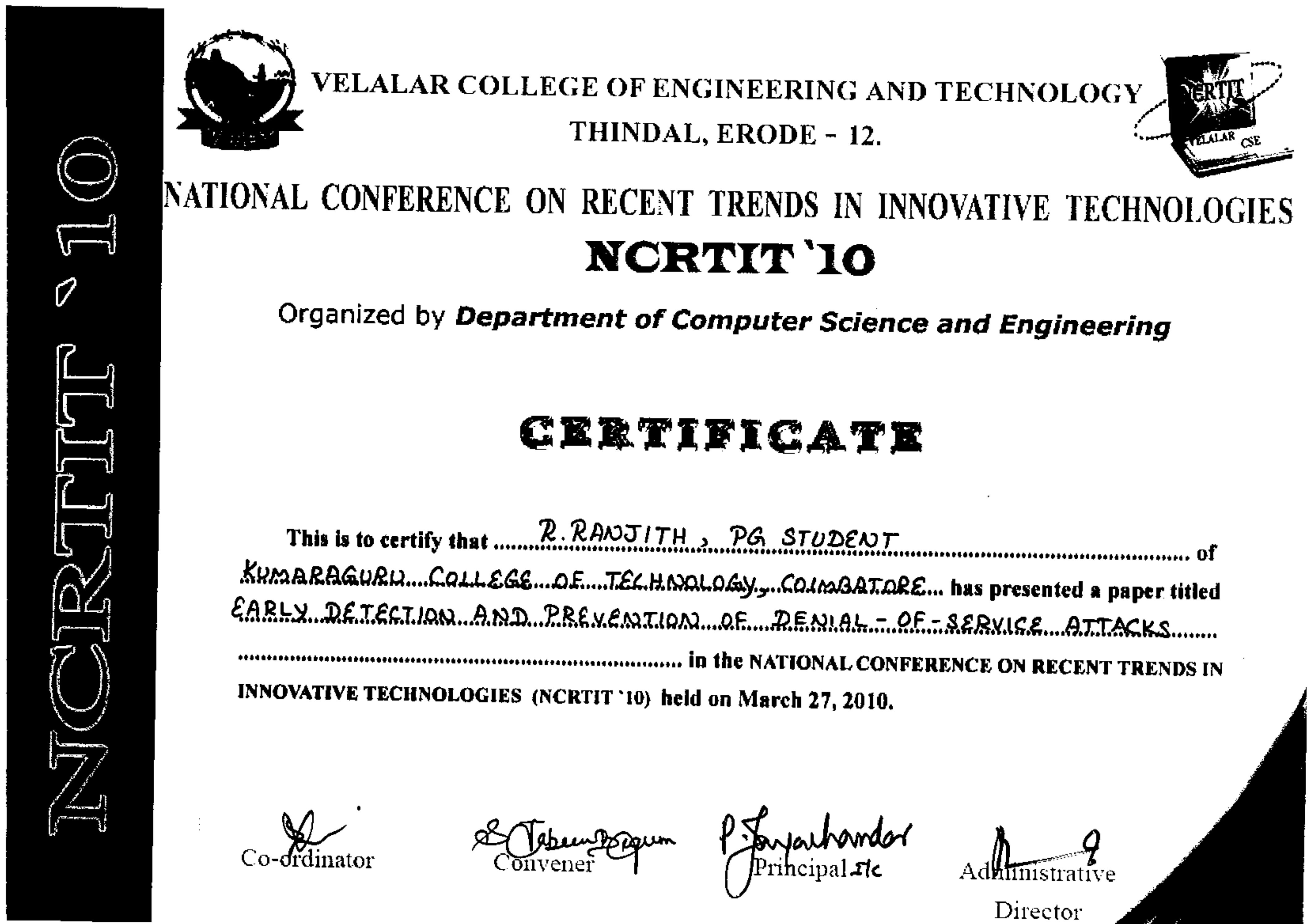
The candidate with **University Register No. 0820108013** was examined by us in Project Viva-Voce examination held on 17/5/10

[illegible] 17/5/10

**Internal Examiner**

[illegible]

**External Examiner**



NCRTIT `10

VELALAR COLLEGE OF ENGINEERING AND TECHNOLOGY
THINDAL, ERODE - 12.

NATIONAL CONFERENCE ON RECENT TRENDS IN INNOVATIVE TECHNOLOGIES

NCRTIT `10

Organized by ***Department of Computer Science and Engineering***

# CERTIFICATE

**This is to certify that** ....R. RANJITH , PG STUDENT.............. **of** KUMARAGURU COLLEGE OF TECHNOLOGY, COIMBATORE... **has presented a paper titled** EARLY DETECTION AND PREVENTION OF DENIAL - OF - SERVICE ATTACKS....... .............. **in the NATIONAL CONFERENCE ON RECENT TRENDS IN INNOVATIVE TECHNOLOGIES (NCRTIT `10) held on March 27, 2010.**

[illegible]
Co-ordinator

[illegible]
Convener

P. Jayachandar
Principal i/c

[illegible]
Administrative Director

# ABSTRACT

A major threat to the information economy is denial-of-service (DoS) attacks. These attacks are highly prevalent despite the widespread deployment of perimeter-based countermeasures. Therefore, more effective approaches are required to counter the threat. This requirement has motivated to propose a novel, distributed, and scalable mechanism for effective early detection and prevention of DoS attacks at the router level within a network infrastructure. This project work presents the design details of the new mechanism. Specifically, this project shows how the mechanism combines both stateful and stateless signatures to provide early detection of DoS attacks and, therefore, protect the enterprise network. More importantly, this discusses how a domain-based approach to an attack response is used by the mechanism to block attack traffic. This project approach enables the blockage of an attack to be gradually propagated only through affected domains toward the attack sources. As a result, the attack is eventually confined within its source domains, thus avoiding wasteful attack traffic overloading the network infrastructure. This approach also provides a natural way of tracing back the attack sources, without requiring the use of specific trace-back techniques and additional resources for their implementation.

## ஆய்வுச்சுருக்கம்

ஒரு பிணையத்தில் அங்கிகாரம் பெற்ற கணினிகள் பல உள்ளன அதில் அங்கிகாரம் பெறாத உறுபினர்கள் சில தேவையற்ற தரவுகளை செர்வெர்க்கு அனுப்பிகொண்டே இருக்கும், அப்படி அனுப்பும் போது வழங்கி செயலிழந்து அங்கிகாரம் பெற்ற உறுப்பினர்களுக்கு தேவையான செய்திகளை தர முடியாமல் போய்விடுகிறது அது ஒரு பிணையத்தின் செயல்ப்பாட்டை குறைக்கும்.

அந்த தேவையற்ற தரவுகளால் ஒரு பிணையத்தின் வேகமான செயல்ப்பாட்டை மிகவும் குறைக்கும் அதனால் பணி இடங்களில் ஒழுங்கான பிணையத்தை உருவாக்க முடியாமல் போகிறது.

இந்த ஆய்வில் சீ2 வழங்கி என்ற உருப்பை பயன்படுத்தி அந்த தெவையற்ற தரவுகளை தடுக்க முடியும். உள்வாங்கும் தரவுகளை சோதித்து ஒரே தரவு பல எண்ணிக்கையில் இருந்தால் அதனை தடுத்து தேவையான தரவை மட்டும் வழங்கிக்கு அனுப்பும், இதனால் ஒரு பிணையத்தில் சீரான தகவல் பரிமாற்றத்தை பெற முடியும்

## ACKNOWLEDGEMENT

I express my profound gratitude to our Chairman **Padmabhusan Arutselver Dr. N. Mahalingam B.Sc, F.I.E** for giving this great opportunity to pursue this course.

I would like to begin by thanking to **Dr.S.Ramachandran**, *principal* for providing the necessary facilities to complete my thesis.

I take this opportunity to thank **Dr. S.Thangasamy,** Dean ,*Head of the Department*, Computer Science and Engineering, for his precious suggestions.

I register my hearty appreciation to **Dr.A.Muthukumar,** *Professor,* my thesis advisor. I thank for his support, encouragement and ideas. I thank him for the countless hours he has spent with me, discussing everything from research to academic choices.

I thank all project committee members for their comments and advice during the reviews. Special thanks to **Mrs.V.Vanitha, M.E, (Ph.D),** *Assistant professor,* Department of Computer science and Engineering, for arranging the brain storming project review sessions.

I would like to convey my honest thanks to all **Teaching** staff members and **Non Teaching** staffs of the department for their support. I would like to thank all my classmates who gave me a proper light moments and study breaks apart from extending some technical support whenever I needed them most.

I dedicate this project work to my **parents** for no reasons but feeling from bottom of my heart, without their love this work wouldn't possible.

## TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF ABBREVIATIONS

| | |
|---|---|
| DDoS | Distributed Denial of Service |
| DiDDeM | Distributed DoS Detection Mechanism |
| DoS | Denial of Service |
| PF | Pre Filters |
| $C^2$ | Command and Control |
| JRI | Joint Response Issuer |
| PFH | Pre Filter Handler |
| EPM | Enhanced Probabilistic Marking |
| AMD | Attack Mitigation Decision-making |
| PPF | Preferential Packet Filtering |
| TLD | Top-Level-Domain |
| IRC | Internet chat program |

# CHAPTER 1

# INTRODUCTION

## 1.1 OVERVIEW OF DENIAL OF SERVICE

### 1.1.1 Problem Definition

A major threat to the information economy is denial-of-service (DoS) attacks. These attacks are highly prevalent despite the widespread deployment of perimeter-based countermeasures. Therefore, more effective approaches are required to counter the threat. This requirement has motivated us to propose a novel, distributed, and scalable mechanism for effective early detection and prevention of DoS attacks at the router level within a network infrastructure. This paper presents the design details of the new mechanism. Specifically, this paper shows how the mechanism combines both stateful and stateless signatures to provide early detection of DoS attacks and, therefore, protect the enterprise network. More importantly, this paper discusses how a domain-based approach to an attack response is used by the mechanism to block attack traffic. This novel approach enables the blockage of an attack to be gradually propagated only through affected domains toward the attack sources. As a result, the attack is eventually confined within its source domains, thus avoiding wasteful attack traffic overloading the network infrastructure. This approach also provides a natural way of tracing back the attack sources, without requiring the use of specific trace-back techniques and additional resources for their implementation.

## 1.2 OVERVIEW OF THE DIDDEM

This section provides an overview of our DiDDeM-based system with particular reference to its cooperative working for early DoS detection and prevention. The key components of the system will be presented in detail in the subsequent sections. If DoS attacks are allowed to reach their intended targets, the attacks are able to succeed in their objective of denying resources to legitimate users. The goal of the DiDDeM-based system is to provide early detection and response to DoS attacks before they denigrate the services and resources of their targets. The system integrates a set of cooperative DiDDeM domains. Each domain is comprised of a single command and control server and a set of prefilters (PFs)/traffic monitors. The acts as a server, located on a designated network node or router, to the PFs within the domain. The key services that the providers are the management of PFs, responses to attacks detected and reported by PFs, and cooperation with adjacent domains. A PF is mainly responsible for attack detection through stateful and stateless signatures as well as attack reporting to the in its domain. Stateful signature analysis applies statistical methods to collected data over a period of time. This data is then analyzed to generate some specific values: for example, traffic thresholds or user profiles to define normal or abnormal behavior. Stateless signature analysis compares an event, such as a network packet, to a list of known signatures, rather than having to hold information, or state, between events. The PF is located on a router within the domain, and no more than one PF runs on a single router. In this paper, we assume that all and PFs are operating in trusted environments. The issues of how to establish such trusted environments for the protection of the PFs are beyond the scope of this paper.

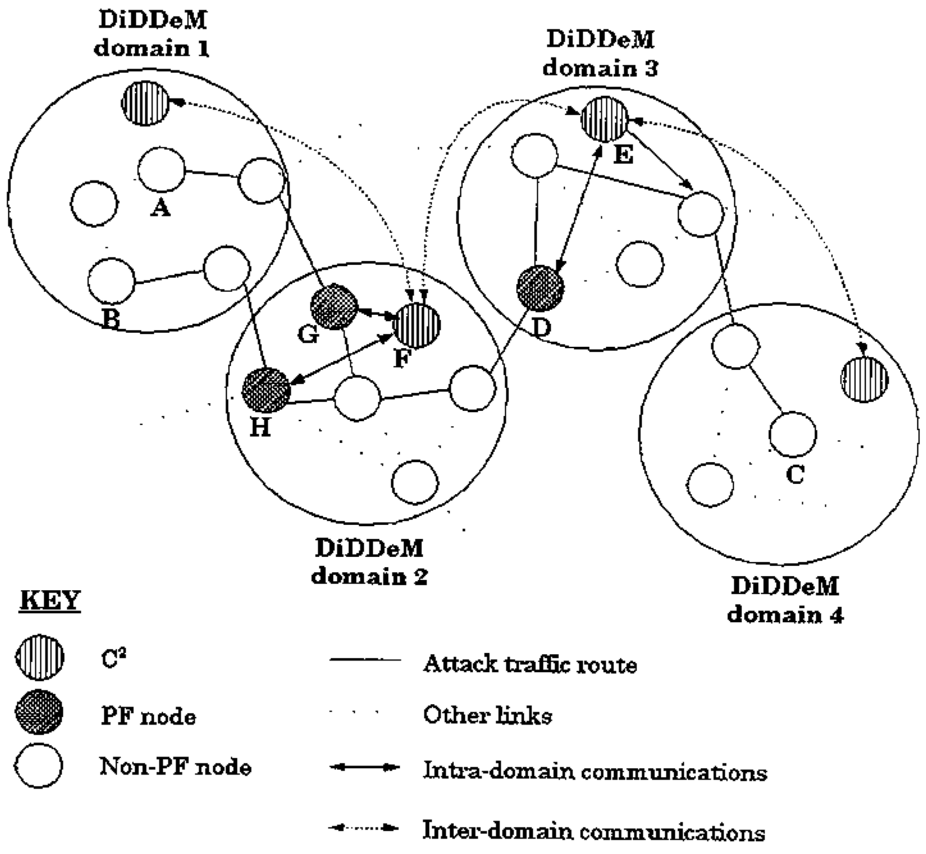


Fig 1.1: DiDDeM domain cooperation example.

A $C^2$ has a number of subordinate routers within its domain. To ensure that a large workload is not placed on the $C^2$ server by having a large number of PFs, only the domain ingress-edge routers run PFs. That is, they are the first routers within the domain that packets from other domains pass through, e.g., the routers/PF nodes highlighted by the darker shading in Fig. 1.1. The placement of PFs at the ingress edge of the domain ensures that attack traffic is detected, reported, and blocked when it is just entering into the domain from adjacent ones. In this way, the other routers within the domain have no need to operate PFs for the same attack detection and response.

### 1.2.1 DIDDEM PREFILTER (PF):

This section presents the design detail of one of the key DiDDeM components, PF. DoS attacks often require a high level of traffic throughput to achieve their objective of reducing availability. Malicious traffic will be interspersed with benign traffic. Therefore, a large amount of traffic must be considered within very tight temporal constraints. In order to facilitate the filtering of such a large volume of traffic, the more benign traffic that can be filtered out by identifying patterns in the TCP/IP headers, the better. This can greatly reduce the processing load of monitoring as the amount of suspicious traffic to be analyzed is kept low. Stateful information of a system, such as unusual rises in traffic volume of a particular protocol, is a major indicator of the possibility of a DoS attack. Beyond the perimeter of a local network, the maintenance of stateful information is more complex and will lead to performance degradation due to the much heavier workload on wider system monitoring. Therefore, a challenge met by the PF design is the inference of state information from stateless information in order to reduce the monitoring workload. Another important issue of the PF design is real-time notification in that an attack alert must be generated in real time in order that a response can be issued to counter the attack. This requires that an attack must be detected while it is ongoing but before the attack traffic reaches its target.

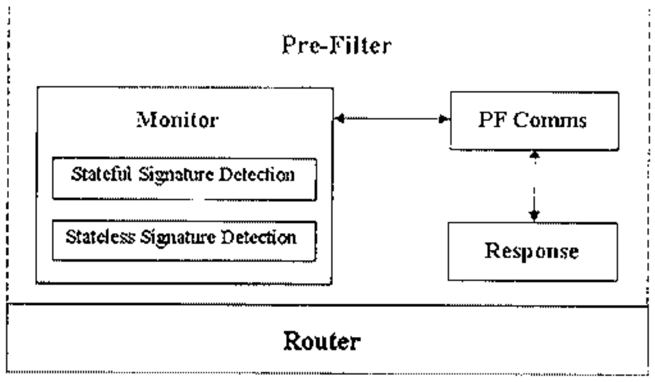


Fig 1.2: PF modules.

The PF design ensures that it is able to infer stateful information about the network being monitored from stateless information for DoS attack detection, report a detected attack to the corresponding $C^2$, and executes response actions instructed by the $C^2$. The PF is located on a router. The part surrounded with the dashed line in Fig. 1.2 represents the PF running on the router. Once a high volume of traffic directed at a particular target is detected, the PF applies stateless signatures to a statistical sample of packets. If a signature match is found, the PF sends this information to its $C^2$ and awaits a response decision.

- **PF Comms:**

  This module is responsible for interaction with the overseeing. It provides the initial authentication with its, as well as other communications between the PF node and server.

- **Monitor**:

  This is comprised of two sub modules: stateful signature detection and stateless signature detection. The stateful signature detection detects unusual rises in network traffic passing through the router directed at a particular host, network, or domain by making use of the router's congestion algorithm.

➢ **Response:**

When the PF receives a response directive from its overseeing, the module instructs the router to enforce the directive. Here, we assume that the router is capable of dropping packets with respect to a specified set of parameters, e.g., a particular packet type and destination, at a required rate. The in a DiDDeM domain is located on a dedicated server (or router). It is responsible for the registration and management of all the PFs within the domain, registration of the servers of all the adjacent DiDDeM domains, issuing of attack responses to relevant PFs within the domain, and coordination with the adjacent domains for joint responses to attacks. The $C^2$ comprises three modules, *Coordinator,·Comms* and *Registration*, as illustrated in Fig. 1.3. The functionality of these modules is described as follows.

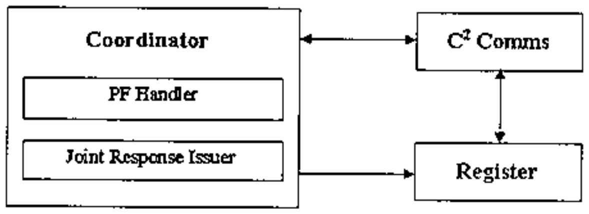


Fig. 1.3: $C^2$ modules.

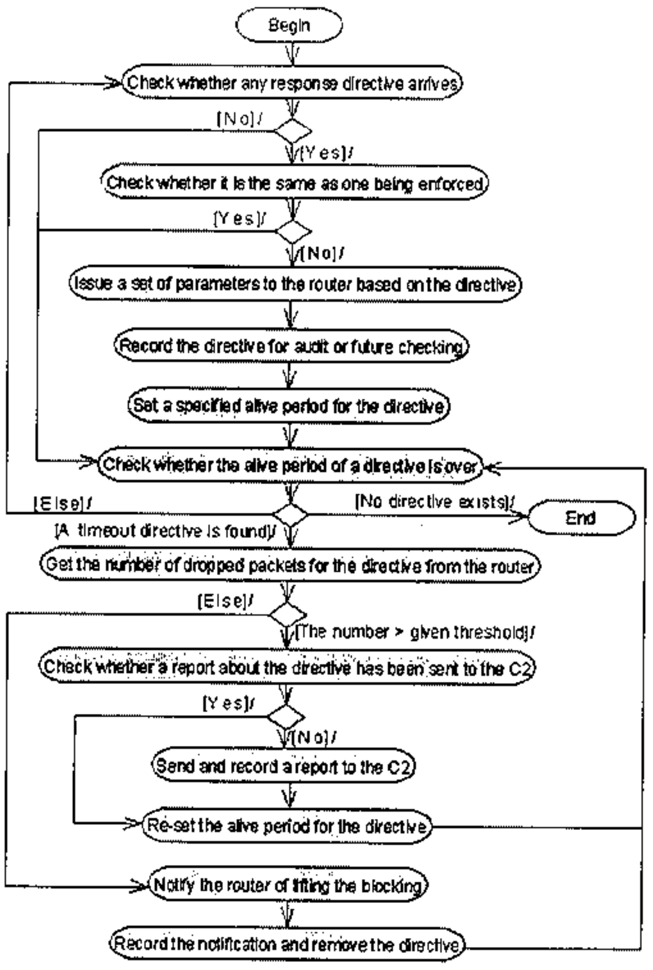

Begin
Check whether any response directive arrives
[No]/
[Yes]/
Check whether it is the same as one being enforced
[Yes]/
[No]/
Issue a set of parameters to the router based on the directive
Record the directive for audit or future checking
Set a specified alive period for the directive
Check whether the alive period of a directive is over
[Else]/
[No directive exists]/
End
[A timeout directive is found]/
Get the number of dropped packets for the directive from the router
[Else]/
[The number > given threshold]/
Check whether a report about the directive has been sent to the C2
[Yes]/
[No]/
Send and record a report to the C2
Re-set the alive period for the directive
Notify the router of lifting the blocking
Record the notification and remove the directive

Fig 1.4: Operation process for PF response.

The response process performed by the response module is illustrated in Fig. 1.4. When the module receives a response directive from the C$^2$, it forms a set of parameters in accordance with the directive received, and passes them to the router to begin filtering attack packets to be dropped based on these parameters.

### 1.2.2 DIDDEM Command and Control

➢ **Comms:**

This module is similar to that located in a PF, namely, it is in charge of authenticated communications with not only each PF within the domain but also the in every adjacent domain.

➢ **Register**:

This module is responsible for the registration of contact details and connectivity for all the PFs within the domain and the servers of all the adjacent domains. It also provides the coordinator module with access to the contact details as will be described below.

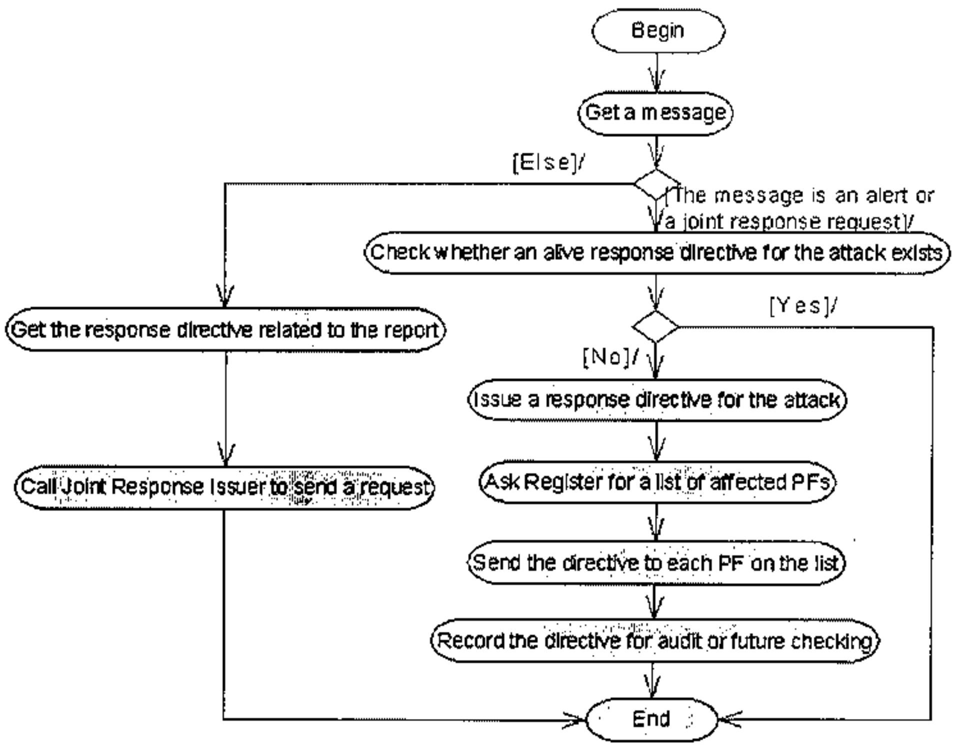
➢ **Coordinator**:

This module controls and coordinates attack responses once an attack has been detected and reported by some PFs. The module is comprised of two sub modules, PF handler and joint response issuer, as depicted in Fig. 1.3. The operations of the two sub modules are explained in detail next.

➢ **PF handler**:

This sub module is in charge of issuing a response directive in relation to an attack discovered and reported by a PF or an adjacent domain, and then distributing the directive to relevant PFs within the domain for blocking the attack traffic. The operational process of the sub module is illustrated in Fig.1.5.

If the message received is an attack alert or a joint response request, the PF handler checks whether there exists a response directive still being enforced, which is about the same attack defined in the message. This checking is based on a specified alive period for the directive. The period begins when the directive is issued, and it sets the minimal duration for the directive to be implemented by relevant PFs. After the alive period, the directive may no longer be implemented by all the PFs. Thus, if an alive directive for the attack stated in the received message exists, there is no need for any further response action. Otherwise, a new response directive should be issued to relevant PFs for blocking attack traffic.



Begin
Get a message
[Else]/
[The message is an alert or a joint response request]/
Check whether an alive response directive for the attack exists
[Yes]/
Get the response directive related to the report
[No]/
Issue a response directive for the attack
Call Joint Response Issuer to send a request
Ask Register for a list of affected PFs
Send the directive to each PF on the list
Record the directive for audit or future checking
End

Fig 1.5: Operation process for PF handler.

➢ **Joint response issuer**:

This sub module issues a joint response request in relation to a given response directive and a particular PF identity, as illustrated in Fig. 1.6. The directive helps to define attack packets to be blocked by adjacent domains. The PF identity is used to identify a list of adjacent domains each of which has a direct link to the router with the PF running on it, namely, the issuer should send the joint response request only to these domains. The issuer asks the register module to carry out this identification as the register manages the connectivity and contact information about all the adjacent domains. It is possible that the issuer has already sent the same joint request to an identified adjacent domain in response to a report from a different PF, and the request is still alive. In this case, the issuer has no need to resend the request to avoid unnecessary communication load.

In summary, the responses decided by the $C^2$ in a DiDDeM domain can be divided into three cases. First, if an alert about a new attack is received from a PF within the domain, the $C^2$ defines the parameters needed for attack packets to be blocked, and uses them to issue a response directive to relevant PFs in the domain to implement the blocking. Second, if a joint response request for blocking an attack is received from an adjacent domain and no alive response directive is for the same attack, then the $C^2$ generates a new response directive based on the parameters given in the received request, and sends it to relevant PFs for blocking the attack traffic.

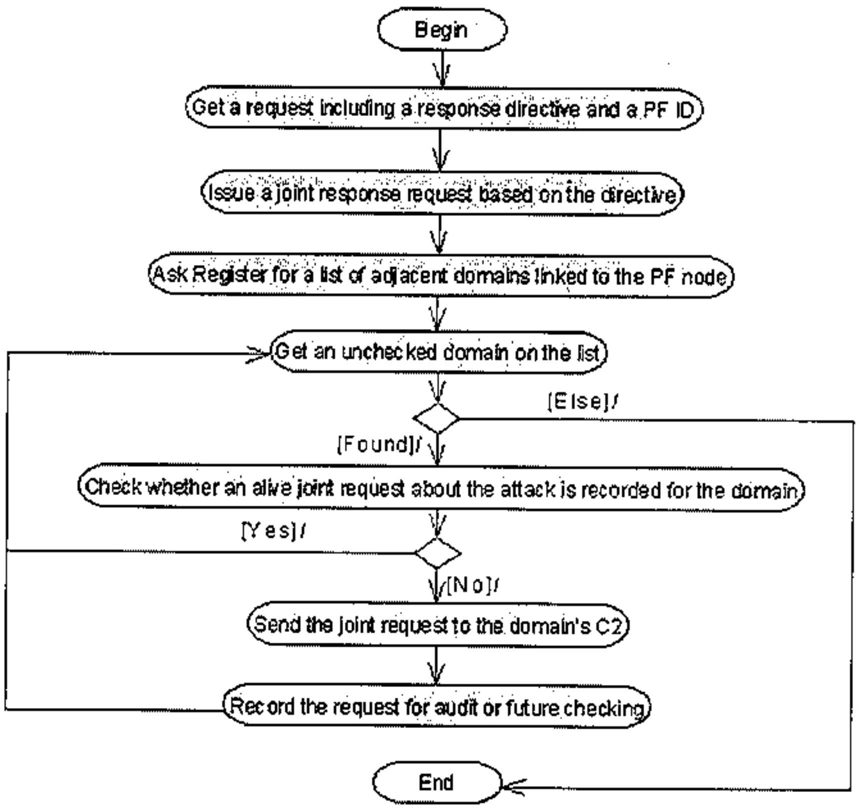

Begin
Get a request including a response directive and a PF ID
Issue a joint response request based on the directive
Ask Register for a list of adjacent domains linked to the PF node
Get an unchecked domain on the list
[Else]/
[Found]/
Check whether an alive joint request about the attack is recorded for the domain
[Yes]/
[No]/
Send the joint request to the domain's C2
Record the request for audit or future checking
End

Fig 1.6: Operation process for joint response issuer.

# CHAPTER 2
# LITERATURE SURVEY

## 2.1 IP TRACE BACK-BASED INTELLIGENT PACKET FILTERING:

**Basic Operation:**

It is a novel technique that can effectively filter out the majority of DDoS attacks by tracking the location of the attackers. It works by performing "smart filtering" dropping DDoS traffic with high probability while allow most of the legitimate traffic to go through. This clearly requires the victim to be able to statistically distinguish legitimate traffic from DDoS traffic.

The proposed scheme leverages on and generalizes the IP trace back schemes to obtain the information concerning whether a network edge is on the attacking path of an attacker or not. The proposed scheme will mark the attacker edges on its path as infected but the edges on the path of a legitimate path will mostly be clean.

1. Enhanced Probabilistic Marking (EPM) module.
2. Attack Mitigation Decision-making (AMD) module.
3. Preferential Packet Filtering (PPF) module.

**Limitations**:

It is not able to mitigate the effect of the attack while it is raging on

## 2.2 TRANSIENT PERFORMANCE OF PACKET SCORE FOR BLOCKING DDOS ATTACKS

**Basic Operation:**

The transient performance of the Packet Score system has been explained by how the performance changes with different attack types, different attack intensities and different measurement window times.

It implements a statistics-based packet scoring mechanism to distinguish between the legitimate and non-legitimate packets there by discarding packets based on the packet score. In Packet Score we perform online traffic profiling of the incoming traffic and compare it with the nominal traffic profile for abnormality detection. The key concept in profiling is the notion of "Conditional Legitimate Probability" (CLP).CLP can be viewed as a score which estimate the legitimacy of the suspicious packet. The threshold used for the score-based selective packet discard decision is dynamically adjusted based on the score distribution of recent incoming packets

A. Conditional Legitimate Probability

B. Off-line Nominal Profile Generation

C. On-line Packet Score Operation

D. Sample score distribution

The algorithm used to implement the above method of trace back system is as follows.

```
for each packet pkt
  u := uniform(0,1)
  if (u < q) then
    v := uniform(0,1)
    if (v < r) then
      encode mark in a way suitable for IP traceback
      set flag to 1 to indicate signaling subchannel
      pkt.hopcount := 0
    else
      encode mark in a way suitable for preferential filtering
      set flag to 0 to indicate data subchannel
  else
    if (flag == 1) pkt.hopcount++
```

**Limitations:**

The packet score can be mislead by changing the attack types and intensities.

## 2.3 SECURED GOSSIP-BASED MULTICAST PROTOCOL:

**Basic Operation:**

It is a simple gossip-based multicast protocols which eliminate single point of failure using redundancy and random choices. Gossip-based protocols can be extremely vulnerable to DoS attacks targeted as a small subset of the processes. Drum is a simple gossip protocol which achieves DoS resistance using a combination of pull and push operations, separate resource bounds for different operations, and the use of random ports in order to reduce the chance of a port being attacked.

**Limitations:**

Inevitable performance degradation will be there.

## 2.4 A PACKET FILTERING SCHEME FOR DETECTION AND PREVENTION OF DDOS ATTACKS.

**Basic Operation:**

This paper introduces a DDoS defense scheme that supports automated online attack characterizations and accurate attack packet discarding based on statistical processing. The Packet Score is used as a packet filtering scheme for detection and prevention the DDoS attacks.

The packet score formulation is done to obtain a threshold value, there by discarding the unwanted packets. It has been done using CLP. CLP can be viewed as a score which estimates the legitimacy of a suspicious packet.

We have reviewed the architecture of the PacketScore scheme that defends against DDoS attacks and studied its transient performance under changing attacks. PacketScore can tackle never-seen-before DDoS attack types by providing a statistics-based adaptive differentiation between attacking and legitimate packets. It is capable of blocking virtually all kinds of attacks as long as the attackers can't precisely mimic the site's traffic characteristics. The packets following the nominal traffic profile have higher score while others have

**Benefits:**

- It reduces the unwanted packet traffic
- Better service quality for the legitimate user request

# CHAPTER 3
## DETAILS OF METHODOLOGY

### 3.1 Network analysis:

The existing network should be analyzed whether there is denial of service attack. It should be analyzed based on non legitimate and legitimate users. Analyze whether the attack packets reaching the target i.e. the main server. Those attack packets are from non legitimate users should be considered as shown in fig 3.1.

Organizations and users alike have become heavily dependent on their network connections. This has given rise to the dichotomy faced by those partaking in the information economy paradigm. The more organizations become reliant on their networks, the more potential damage can be inflicted by attacks launched over these networks.

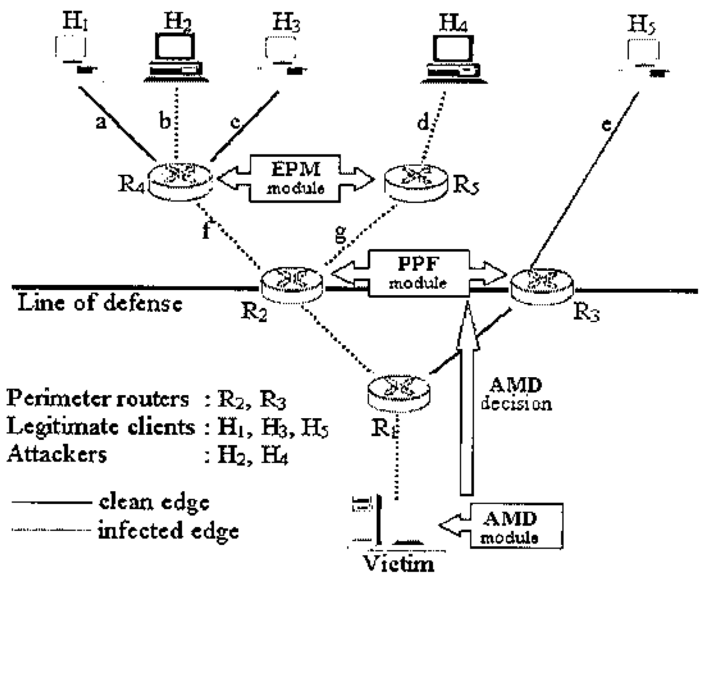


Fig 3.1: Example network as seen from the victim.

Background traffic was necessary to determine the false alarm rates of intrusion detection systems. A large amount of web, telnet, and mail traffic was generated between the inside PC's and workstations and the outside workstations and web sites. In addition, there are many user automata of various types (e.g. secretaries, programmers, managers) on outside workstations who perform work using telnet and other services on the three inside victim machines and the other inside workstations. The three gateway machines contain operating system kernel modifications similar to those used in conjunction with custom software web, mail, telnet, and other servers to allow a small number of actual hosts to appear as if they were 1000's of hosts with different IP addresses. The contents of network traffic such as file transfers are either statistically similar to live traffic, or sampled from public-domain sources. For example, some email message contents are created using statistical bigrams frequencies to preserve word and two-word sequence statistics from a sampling of roughly 10,000 actual email messages to and from computer professionals filtered using a 40,000 word dictionary to remove names and other private information. Other email messages are actual messages from a variety of public-domain mail list servers. Similar approaches were used to produce content for FTP file transfers.

Content of the web servers was initially captured using a custom web automaton that was run on the real Internet. This automaton was programmed to visit thousands of web sites popular with university and government personnel with a frequency that depended on the site's popularity and to visit a random number of links at each site before traversing to another site. It generated a large database of public-domain site content, which was transferred to the evaluation test bed. This was necessary because the evaluation test bed was disconnected from the Internet for security reasons and live web sites could thus not be accessed. When the test bed was run, browsing automata accessed web pages through the outside web gateway. This gateway used custom software to emulate thousands of web sites. Telnet sessions were generated from statistical profiles of user types that were used to generate interactive sessions. These statistical profiles indicated the frequency of occurrence of different UNIX commands, typical login times and telnet session durations, typical source and destination machines, and other information. For example, programmers primarily edited C programs, compiled these programs, sent mail, read the manual pages, and ran programs. Secretaries edited documents, and sent mail. There were also a large number of users who primarily sent and received mail and browsed web sites. Public domain sources were used to obtain software programs created by simulated programmers, documents created by secretaries, and other content. As suggested, a modified version of the expect language was used to create user automata which behaved as if they were users typing at keyboards. Human actors performed more complex tasks. They upgraded software, added users, changed passwords, remotely accessed programs with graphical user interfaces, and performed other system administration tasks.

## 3.2 Constructing PREFILTER (PF):

Constructing prefilter is to detect the attack packets from the non legitimate users. It will maintain a count to identify the attack packets that are from the same users.

Stateful information of a system, such as unusual rises in traffic volume of a particular protocol, is a major indicator of the possibility of a DoS attack. Beyond the perimeter of a local network, the maintenance of stateful information is more complex and will lead to performance degradation due to the much heavier workload on wider system monitoring. Therefore, a challenge met by the PF design is the inference of state information from stateless information in order to reduce the monitoring workload.

Another important issue of the PF design is real-time notification in that an attack alert must be generated in real time in order that a response can be issued to counter the attack. This requires that an attack must be detected while it is ongoing but before the attack traffic reaches its target.

The PF design ensures that it is able to infer stateful information about the network being monitored from stateless information for DoS attack detection, report a detected attack to the corresponding $C^2$, and executes response actions instructed by the $C^2$. The PF is located on a router. The part surrounded with the dashed line represents the PF running on the router. Once a high volume of traffic directed at a particular target is detected, the PF applies stateless signatures to a statistical sample of packets. If a signature match is found, the PF sends this information to its $C^2$ and awaits a response decision.

## 3.3 Command and Control server:

It is responsible for the registration and management of all the PFs within the domain. Registration of the servers of all the adjacent DiDDeM domains Issuing of attack responses to relevant PFs within the domain Coordination with the adjacent domains for joint responses to attacks.

If the message received is an attack alert or a joint response request, the PF handler checks whether there exists a response directive still being enforced, which is about the same attack defined in the message. This checking is based on a specified alive period for the directive. The period begins when the directive is issued, and it sets the minimal duration for the directive to be implemented by relevant PFs. After the alive period, the directive may no longer be implemented by all the PFs. Thus, if an alive directive for the attack stated in the received message exists, there is no need for any further response action. Otherwise, a new response directive should be issued to relevant PFs for blocking attack traffic. Note that it is possible that some of these PFs are still enforcing a previous directive equivalent to the new one. In this case, these PFs simply ignore the new directive.

When receiving a message through the $C^2$ Comms module, the PF handler determines that the message is a report from a PF about an attack being blocked, an attack alert from a PF, or a joint response request from an adjacent domain. The first case indicates that an excessive amount of attack traffic is entering into the router associated with the PF. Hence, the handler needs to call the joint response issuer sub module to generate and send a joint response request to each adjacent domain with a direct link to the affected router to block the attack traffic. This means that the attack blocking is propagated from the current domain to its adjacent one closer to a source of the attack. It is easy to see that the more such propagation is repeated by relevant domains, the more closely to the attack source the attack blocking moves.

# CHAPTER 4
# EXPERIMENTAL RESULTS

## 4.1 C$^2$ SERVER:

C$^2$ in a domain is located in a dedicated server. It is responsible for the registration and management of all the PFs within the domain, registration of the c2 servers of all the adjacent domains, issuing of attack responses to relevant PFs within the domain, and coordination with the adjacent domains for joint responses to attacks.

The C$^2$ in a DiDDeM domain is located on a dedicated server (or router). It is responsible for the registration and management of all the PFs within the domain, registration of the C$^2$ servers of all the adjacent DiDDeM domains, issuing of attack responses to relevant PFs within the domain, and coordination with the adjacent domains for joint responses to attacks. The C$^2$ comprises three modules, Coordinator, C$^2$ Comms and Registration.

All unnecessary services should be removed. The less there are applications and open ports in hosts, the less there are vulnerabilities to be exploited by an attacker. Default installations of operating systems often include many applications not needed by a user. Especially many home-users do not even know what services are running on their systems. A vulnerability scanner can be used to detect what network services (open ports) are available in a network.

A firewall (or a router with similar abilities) should be used to control access to a network. Even if there are many services available from local hosts, not all of these services need to be accessible from the public Internet.

All relevant security patches should be installed timely. The DDoS tool deployment phase and many logic DoS attacks are based on exploiting vulnerabilities in host software. Removing known security holes prevents re-exploitation of vulnerabilities for example with publicly available scripts. In practice, this important defense is often neglected which makes it possible for available exploits to have lifetime.

Attackers should not be able to get unauthorized access to hosts, e.g., by exploiting weak passwords. A minimum requirement is to use passwords which are difficult to guess with or without existing password cracking tools.

The antivirus software should be using the most recent virus definition database. This helps detecting known worms and viruses. Antivirus software can thus be considered as IDS.

Register module is responsible for the registration of contact details and connectivity for all the PFs within the domain and the $C^2$ servers of all the adjacent domains. It also provides the coordinator module with access to the contact details as will be described below.

Joint response issuer sub module issues a joint response request in relation to a given response directive and a particular PF identity. The directive helps to define attack packets to be blocked by adjacent domains. The PF identity is used to identify a list of adjacent domains each of which has a direct link to the router with the PF running on it, namely, the issuer should send the joint response request only to these domains. The issuer asks the register module to carry out this identification as the register manages the connectivity and contact information about all the adjacent domains. It is possible that the issuer has already sent the same joint request to an identified adjacent domain in response to a report from a different PF, and the request is still alive. In this case, the issuer has no need to resend the request to avoid unnecessary communication load.

Automatic DDoS attacks automate the use phase in addition to the recruit, exploit and infect phases, and thus avoid the need for any communication between attacker and agent machines. The start time of the attack, attack type, duration and victim are preprogrammed in the attack code. Deployment mechanisms of this attack class minimal exposure to the attacker, since he is only involved in issuing a single command at the start of the recruitment process. The hardcoded attack specification suggests a single-purpose use of the DDoS network, or the inflexible nature of the system. However, the propagation mechanisms usually leave a back door to the compromised machine open, enabling easy future access and modification of the attack code. Further, if agents communicate through IRC channels, these channels can be used to modify the existing code. Coordinator module controls and coordinates attack responses once an attack has been detected and reported by some PFs. The module is comprised of two sub modules, PF handler and joint response issuer, as depicted.

### 4.1.1 PERPETRATE DDOS ATTACKS:

The main goal is to inflict damage on the victim. Frequently the ulterior motives are personal reasons (a significant number of DDoS attacks are perpetrated against home computers, presumably for purposes of revenge), or prestige (successful attacks on popular Web servers gain the respect of the hacker community). However, some DDoS attacks are performed for material gain (damaging a competitor's resources or blackmailing companies) 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.

In this project, we propose a protocol-independent DDoS defense scheme that is able to dramatically improve the throughput of legitimate traffic during a DDoS attack. It works by performing "smart filtering": dropping DDoS traffic with high probability while allowing most of the legitimate traffic to go through. This clearly requires the victim to be able to statistically distinguish legitimate traffic from DDoS traffic. The proposed scheme leverages on and extends IP traceback techniques to gather "intelligence": information such as whether or not a network edge is on the path from an attacker ("infected"). By preferentially filtering out packets that are inscribed with the mark (identity) of an "infected" edge, the proposed scheme filters out most of the traffic from attackers since each and every edge on an attacker's path to the victim is infected. Packets from a legitimate client, on the other hand, with high probability will not be filtered out, since, typically, most of the edges on the client's path to the victim are not infected. To evaluate its effectiveness in defending against DDoS attacks, the proposed scheme is simulated on three sets of real-world Internet topologies with varying operating parameters. Simulation results demonstrate that the throughput of the legitimate traffic can be increased by three to seven times. To the best of our knowledge, this is the first research that leverages on IP traceback for automatic DDoS response. What makes this approach more appealing is that the operations required of routers (probabilistic marking) are fully in line with the operations of IP traceback.

Recently there were occurrences of attacks that varied the set of agent machines active at any one time, avoiding detection and hindering traceback. We regard this technique as important since it invalidates assumptions underlying many defense mechanisms that agents are active throughout the attack and can thus be traced back following the path of the attack traffic. We divide attacks, based on the persistence of the agent set, into attacks with constant agent set and attacks with variable agent set.

During attacks with a constant agent set, all agent machines act in a similar manner, taking into account resource constraints. They receive the same set of commands and are engaged simultaneously during the attack. Examples are an attack in which all agents start sending attack traffic simultaneously, or they engage in a pulsing attack but the periods for pulses match over all agent machines.

During attacks with a variable agent set, the attacker divides all available agents into several groups and engages only one group of agents at any one time | like the army general who deploys his battalions at different times and places. A machine could belong to more than one group, and groups could be engaged again after a period of inactivity. One example attack of the variable agent set type is an attack in which several agent groups take turns pulsing, thus flooding the victim with a constant flow of packets.

### 4.1.2 Application:

Application attacks target a given application on the victim host, thus disabling legitimate client use of that application and possibly tying up resources of the host machine. If the shared resources of the host machine are not completely consumed, other applications and services should still be accessible to the users. For example, a bogus signature attack on an authentication server ties up resources of the signature verification application, but the target machine will still reply to ECHO requests, and other applications that do not require authenticated access should still work. Detection of application attacks is challenging because other applications on the attacked host continue their operations undisturbed, and the attack volume is usually small enough not to appear anomalous. The attack packets are virtually indistinguishable from legitimate packets at the transport level (and frequently at the application level), and the semantics of the targeted application must be heavily used for detection. Since there are typically many applications on a host machine, each application would have to be modeled in the defense system and then its operation monitored to account for possible attacks. Once detection is performed, the host machine has sufficient resources to defend against these small volume attacks, provided that it can separate packets that are legitimate from those that are part of the attack.

Host attacks disable access to the target machine completely by overloading or disabling its communication mechanism or making a host crash, freeze or reboot. An example of this attack is a DoS attack. All attack packets carry the destination address of the target host. If protocols running on the host are properly patched, the host attacks likely to be perpetrated against it are reduced to attacks that consume network resources. The high packet volume of such attacks facilitates detection, but the host cannot defend against these attacks alone. It must usually request help from some upstream machine.

Resource attacks target a critical resource in the victim's network such as a specific DNS server, a router or a bottle-neck link. The paths of the attack packets merge before or at the target resource and may diverge afterwards. These attacks can usually be prevented by replicating critical services and designing a robust network topology.

Network attacks consume the incoming bandwidth of a target network with attack packets whose destination address can be chosen from the target network's address space. These attacks can deploy various packets (since it is volume and not content that matters) and are easily detected due to their high volume. The victim network must request help from upstream networks for defense since it cannot handle the attack volume itself.

Infrastructure attacks target some distributed service that is crucial for global Internet operation. Examples include the attacks on domain name servers, large core routers, routing protocols, certificate servers, etc. The key feature of these attacks is not the mechanism they deploy to disable the target (e.g., from the point of view of a single attacked core router, the attack can still be regarded as a host attack), but the simultaneity of the attack on multiple instances of a critical service in the Internet infrastructure. Infrastructure attacks can only be countered through the coordinated action of multiple Internet participants.

Content of the web servers was initially captured using a custom web automaton that was run on the real Internet. This automaton was programmed to visit thousands of web sites popular with university and government personnel with a frequency that depended on the site's popularity and to visit a random number of links at each site before traversing to another site. It generated a large database of public domain site content, which was transferred to the evaluation test bed. This was necessary because the evaluation test bed was disconnected from the Internet for security reasons and live web sites could thus not be accessed. When the test bed was run, browsing automata accessed web pages through the outside web gateway. This gateway used custom software to emulate thousands of web sites. Telnet sessions were generated from statistical profiles of user types that were used to generate interactive sessions. These statistical profiles indicated the frequency of occurrence of different UNIX commands, typical login times and telnet session durations, typical source and destination machines, and other information. For example, programmers primarily edited C programs, compiled these programs, sent mail, read UNIX manual pages, and ran programs. Secretaries edited documents, and sent mail. There were also a large number of users who primarily sent and received mail and browsed web sites. Public domain sources were used to obtain software programs created by simulated programmers, documents created by secretaries, and other content. As suggested in a modified version of the expect language was used to create user automata which behaved as if they were users typing at keyboards. Human actors performed more complex tasks. They upgraded software, added users, changed passwords, remotely accessed programs with graphical user interfaces, and performed other system administration tasks.

## 4.2 ROUTER:

Router is responsible for running PFs. A $C^2$ has a number of subordinate routers within its domain. To ensure that a large workload is not place on the c2 server by having a large number of PFs, only the domain ingress-edge routers run PFs. That is they are the first routers within the domain that packets from other domains pass through.

Router comprises of registration of clients in the domain, settings, signatures List.

In the Registration all the clients in this domain is saved. For example, the IP Address of clients, Host Name. Settings contains $C^2$ server, router, Prefilter are registered.

Signatures List comprises of stateless and statefull virus headers. All these can be visualized in the Router.

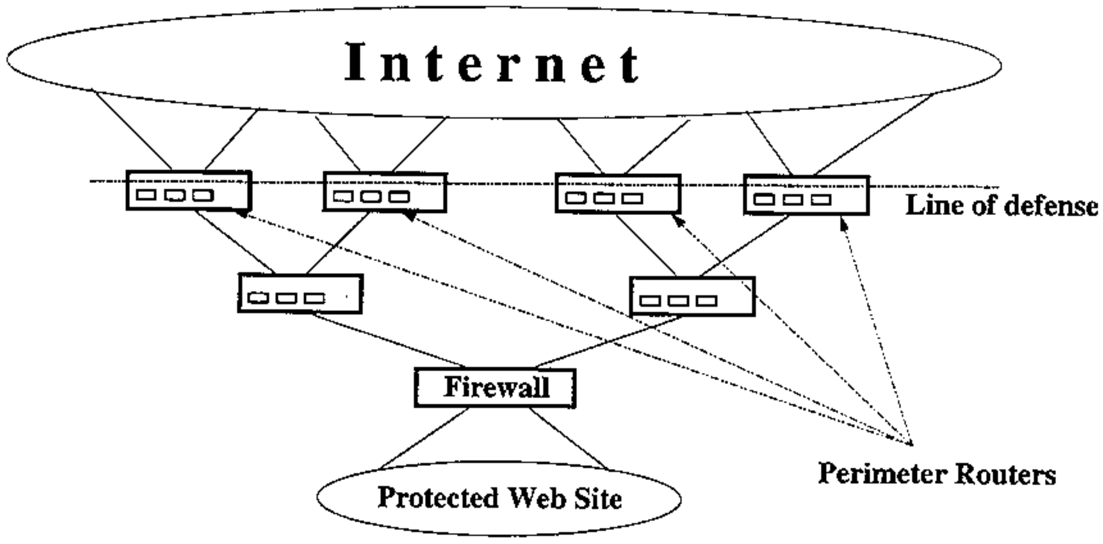


Fig 4.1: The perimeter router model

### 4.2.1 DOS ATTACKS AGAINST DNS:

Two of the most common DoS attacks against DNS are direct DoS attacks and amplification attacks. For the direct DoS attacks, attacker tries to overwhelm the server by sending an excess traffic from single or multiple sources. Based on the measurement of DNS during the periods of distributed DoS attacks, a huge number of query packets were received by the target name server. Therefore, the name servers flooded by DoS attacks will experience packet loss and cannot always respond to every DNS request. The percentage of lost incoming DNS requests due to the excessive load is based on the DoS attack intensity. It also points that the packet size of DNS data flow is small and the message amount is little that make the process of anomalous behaviors detection more difficult.

On the other hand, attackers establish the most sophisticated and modern type of DoS attacks known as amplification attacks to increase the effect of normal DoS attacks. The attacks against TLD (Top-Level-Domain) name servers are all forms of DNS amplification attacks. The reason behind the name of amplification attack is that the attacker makes use of the fact that small queries can generate much larger UDP packets in response. Nowadays, the attackers use the recent extensic ı of DNS protocol (RFC 2671) to magnify the amplification factor. For example a 60 bytes DNS request can be answered with responses of over 4000 bytes. This yields an amplification factor of more than 60. Several researchers have studied the effects of reflected amplification attacks. Based on their analyses, patterns of these attacks include a huge number of nonstandard packets larger than the standard DNS packet size which is 512 bytes.

Data collection is usually a basic requirement for any DoS. The type of data that we should collect is based on the type of DoS. There are two approaches for normal and intrusion traffic simulations. One is to simulate using a tested in a real environment. The other is to simulate using simulator softwares. When accessing to a real environment for traffic simulation is hard, we exploit the power of network simulators. According to our knowledge, there are no available generated dataset for DoS attacks against DNS. Therefore, the required data for our experiments was generated using a network simulator. The network topology of our simulation contains a single legitimate client, an attacker, and two servers. All nodes are connected to the same router. All the links are 100Mbps and 10ms except the link between target server and router that is 10Mbps and 10ms delay. A queue size of 100 packets, with a drop-tail queuing strategy was used. There are two types of traffic generated in the network which are legitimate traffic and attack traffic. A modified version of Agent/Ping with a maximum of 3 retransmissions with 5-second timeouts is used for DNS as implemented. The request inter arrival period is fixed at 10s. The attacker is expected to flood the target name server with excess traffic. We chose different values of delay for applying to the attack start time in order to achieve variability.

This router describes two different types of DoS attacks against attack packets and their patterns. Based on these patterns the required traffic data was simulated. We proposed a machine learning based system for detecting and classifying DoS attacks against DNS. For this purpose three different neural networks were evaluated. The results show that a three layered BP neural network with a structure can give us good accuracy and a good classification rate for direct DoS and amplification attacks comparing to Trace back and Gossip-Based Multicast networks.

Similar to the off-line packet profile, an on-line profile is generated as packets arrive. Rather than performing expensive calculation of the CLP on the fly, we employ a scorebook approach. A frozen set of recent histograms in period is used along with the stored off-line nominal profile to generate a set of "scorebooks" which maps a specific combination of attribute values to its corresponding "score".

Most intrusion detection systems provide some degree of configuration to allow experts to customize the system to a given environment. To avoid learning how to run and customize each intrusion detection system, to reduce the time required to perform the evaluation, and to perform a fair comparison, we elected to perform an off-line blind evaluation of all systems. Two sets of data were provided to participants. First, seven weeks of training data were provided from July to mid September 1998. This training data contained normal background traffic and labeled attacks. Expert users or system developers configured their systems and trained any learning algorithms to achieve the highest detection rates and the lowest false alarm rates on this training data. Then two weeks of unlabeled test data was provided at the end. Participants ran their intrusion detection systems on this test data and returned a list of all attacks detected, without knowledge of the locations or of the types of attacks. This approach made it easy to participate ensured that all participants are evaluated fairly and with minimum bias, and lead to the development of evaluation corpora that can be used by many researches for system design and refinement. Practical

### 4.2.2 INTERNET RESOURCES ARE LIMITED:

Each Internet entity (host, network, service) has limited resources that can be consumed by too many users.

An end-to-end communication paradigm led to storing most of the intelligence needed for service guarantees with end hosts, limiting the amount of processing in the intermediate network so that packets could be forwarded quickly and at minimal cost. At the same time, a desire for large throughput led to the design of high bandwidth pathways in the intermediate network, while the end networks invested in only as much bandwidth as they thought they might need. Thus, malicious clients can misuse the abundant resources of the unwitting intermediate network for delivery of numerous messages to a less provisioned victim.

Attacks with indirect communication use some legitimate communication service to synchronize agent actions. Recent attacks have used IRC (Internet chat program) channels. The use of IRC services replaces the function of a handler, since the IRC channel users sufficient anonymity to the attacker. The agents do not actively listen to network connections (which means they cannot be discovered by scanners), but instead use the legitimate service and their control packets cannot be easily differentiated from legitimate chat trace. The discovery of a single agent may lead no further than the identification of one or more IRC servers and channel names used by the DDoS network. From there, identification of the DDoS network depends on the ability to track agents currently connected to the IRC server (which may be the subset of all subverted machines). To further avoid discovery, attackers frequently deploy channel- hopping, using any given IRC channel for short periods of time. Since IRC service is maintained in a distributed manner, and the IRC server hosting a particular IRC channel may be located anywhere in the world, this hinders investigation. Although the IRC service is the only known example of indirect communication so far, there is nothing to prevent attackers from subverting other legitimate services for similar purposes.

Although the proposed scheme may leverage on any of the existing IP traceback schemes in this paper, we show how it builds on the Advanced Marking Scheme proposed. The advantage of the AMS is that it provides faster reconstruction and higher accuracy (hence, fewer false positives in identifying attackers) than other IP traceback schemes, when there are more than one attackers. However, it assumes that the victim is able to obtain a map of upstream routers, which is a stronger (arguably less practical) assumption than used in other IP traceback. Our future research will study how our scheme works with other IP traceback techniques.

AMS employs a technique similar to the Bloom filter as follows: It uses eight independent hash functions to encode network edges. When a packet goes through an edge e and the identity of the edge is to be marked, it will be chosen uniformly between 1 and 8, and the mark is written into the IP header of the packet (representing concatenation). The reconstruction algorithm determines that an attacker has e on its path if and only if the algorithm has received attacking packets with at least k out of the eight mark values. The tunable parameter larger k results in longer "attack graph" reconstruction time, but fewer false positives when identifying infected edges. We view this as a variant of Bloom filter since all the edges that an attacker has traversed can be viewed as a set, and it is represented by a bit array indexed by the values generated by these hash functions.

We begin by evaluating the three protocols in a failure-free scenario, and in situations where crash failures occur. We assume that the crashes occur before M is generated, and that the source does not crash. We also assume that the crashes are not detected by the correct processes, i.e., they try to gossip with crashed processes as well. Our aim is to validate two known results: 1) the propagation time of gossip-based multicast protocols can be seen, with a logarithmic x-axis, and 2) the performance of such protocols degrades gracefully as crash failures amount, as depicted. We can see that Push and Pull slightly outperforms Drum in these experiments. This is due to the fact that the bounds on the pull and push channels in Drum are strict, i.e., even if in a specific round, no messages have arrived via the push channels, only requests from at most two distinct processes will be handled, although the process is capable of handling four such requests. Conversely, Push and Pull have only one bound, which guarantees that messages will not be discarded if they can be processed. The ability to perform well even when many processes crash stems from the random choice of communication partners each round.

## 4.3 PREFILTER:

PF ensures that it is able to infer stateful information about the network being monitored from stateless information for Dos attack detection, report a detected attack to the corresponding c2, and executes response actions instructed by the c2.The PF is located on a router. Once high volume of traffic directed at a particular target is detected, the PF applies stateless match is found, the PF sends this information to its C2 and awaits a response decision. PF comprises of three modules. Monitor, PF Comms and Response.

Monitor is comprised of the two sub modules statefull signature detection and stateless detection. The stateful signature detection detects unusual risks in network traffic passing through the router directed at a particular host, network, or domain by making use of the router's congestion algorithm.

PF Comms module is responsible for interaction with the overseeing $C^2$. It provides the initial authentication with its $C^2$, as well as other communications between the PF node and $C^2$ server. Response is received from $C^2$ by the PF. This module instructs the router to enforce the directive. Here the router is capable of dropping packets with respect to a specific set of perimeters eg: a particular packet type and destination.

Having observed the vulnerabilities of traditional protocols, we turn to search for ways to eliminate these vulnerabilities. Specifically, our goal is to design a protocol that does not allow an attacker to increase the damage it causes by focusing on a subset of the processes. We are not familiar with any previous protocol that achieves this goal. We are familiar with only one previous work that addresses DoS attacks on a DoS based protocol. However, the problem they consider differs from ours in a way that renders their approach inapplicable to our setting and, moreover, they only deal with limited attack strengths.

As a first defense, one may protect a system against DoS attacks using network-level mechanisms. These mechanisms involve rate-limiting incoming traffic and filtering packets according to their headers. However, network-level filters cannot detect DoS attacks at the application level, when the traffic seems legitimate. Even if means are in place to protect against network-level DoS, an attack can still be performed at the application level, as the bandwidth needed to perform such an attack is usually lower. This is especially true if the application performs intensive computations for each message, as occurs, e.g., with secure protocols based on digital signatures.

As network-level DoS-mitigation solutions are increasingly available, application level DoS attacks are becoming a major concern. Consequently, vendors have begun employing some measures against DoS attacks at the application layer. Such solutions are commonly deployed at the network/firewall level, although they are application-specific. However, these measures are usually just hard-coded validity checks for well-known protocols, and do not contain means to deal with resource exhaustion caused by the application. In this paper, we are concerned with coping with DoS attacks in application-level multicast protocols. The basic idea is to assume simple and general mechanisms at the network/firewall level and to exploit them at the application (multicast protocol) level.

Another important issue of the PF desigr is real-time notification in that an attack alert must be generated in real time in order that a response can be issued to counter the attack. This requires that an attack must be detected while it is ongoing but before the attack traffic reaches its target.

We now explain how the combination of push, pull, random port selections, and resource bounds achieves resistance to targeted DoS attacks. A DoS attack can flood a port with fabricated messages. Since the number of messages accepted on each port in a round is bounded, the probability of successfully receiving a given valid message M in a given round is inversely proportional to the total number of messages arriving on the same port as M in that round. Thanks to the separate resource bounds, an attack on one port does not reduce the probability for receiving valid messages on other ports.

In order to prevent a process from sending its messages using a push operation, one must attack (flood) the push offer targets, the ports where push-replies are awaited, or the ports where data messages are awaited. However, the push destinations are randomly chosen in each round, as are the push-reply and data ports. Thus, the attacker has no way of predicting these choices.

Similarly, in order to prevent a process from receiving messages during a pull operation, one needs to target the destination of the pull-requests or the ports on which pull replies arrive. However, the destinations and ports are randomly chosen. Thus, using the push operation, Drum achieves resilience to targeted attacks aimed at preventing a process from sending messages, and using the pull operation, it withstands attacks that try to prevent a process from receiving messages.

### 4.3.1 DDoS attacks performance:

A DDoS attack is carried out in several phases. The attacker first recruits multiple agent machines. This process is usually performed automatically through scanning of remote machines, looking for security holes that will enable subversion. The discovered vulnerability is then exploited to break into recruited machines and infect them with the attack code. The exploit/infect phase is frequently automated, and the infected machines can be used for further recruitment of new agents. Another recruit/exploit/infect strategy consists of distributing attack software under disguise of a useful application (these software copies are called Trojans). This distribution can be performed, for instance, by sending E-mail messages with infected attachments. Subverted agent machines are used to send t!.e attack packets. Attackers often hide the identity of subverted machines during the attack through spoofing of the source address yield in attack packets. Note, however, that spoofing is not always required for a successful DDoS attack. With the exception of reflector attacks, all other attack types use spoofing only to hinder attack detection and characterization, and the discovery of agent machines.

## 4.4 CLIENT:

After all the settings done in the Router and $C^2$ server all the clients in the domain are run. Thereafter the communication between clients is happened. All the packets from clients are directed to destination through Router. Router will enforce PF to check the incoming packets. Here the checking is to identify the packet type and destination. Here before sending any packets to router they have to join by enter host name as username.

In all of our evaluations, we stage various DoS attacks. We assume that the DoS attacks are launched from outside the system. DoS from inside the group is essentially just one source (or more) generating excessive traffic. This can happen regardless of any malicious nodes being part of the multicast group, e.g., in a heterogeneous system. Consequently, this is, in fact, a flow-control problem, as one cannot differentiate between a malicious attack and legitimate excessive traffic. Flow control in gossip-based multicast has been dealt with.

In each DoS attack, the adversary focuses on a fraction of the processes and sends each of them x fabricated messages per round. We note that randomly choosing the attack targets every round does not make any difference, as the communication partners are re chosen uniformly at random each round. We denote the total attack strengthens. We assume that the message source is being attacked (this has no impact on the results of Push). We consider attacks either of a fixed strength, where B is fixed and increases or of increasing strength, where either x is fixed and increases, or vice versa. Examining fixed strength attacks allows us to identify protocol vulnerabilities, e.g., whether an adversary can benefit from targeting a subset of the processes. Increasing strength attacks enable us to assess the protocols' performance degradation due to increasing attack intensity.

In a case study implemented in ns2 and detailed in the above method for the derivation of stateful information was implemented. During the simulation, approximately 19 500 attack packets were directed at a victim node by two attacking nodes. This represents an attack consisting of approximately 1000 packets/s. Once the congestion algorithm was invoked by the router, 798 attacks and legitimate packets were to be dropped. Of this number, 742 packets were actual attack packets while the remainders were legitimate traffic. Therefore, out of a total of 19 500 attack packets, only about 4% of this volume was inspected. The 697 packets detected out of the 742 inspected by the DiDDeM-enhanced router ensure a 94% detection rate. In addition, only two legitimate packets were detected as attack packets, thus providing a false positive rate of 4%. In a simulation of stateless signature detection, stateless TCP SYN flood signatures were applied to 432 packets, of which 50 were actual attack packets. For TCP SYN flood detection, packets were searched for instances of SYN flags within the header, which are indicative of an attack. If a flag is found, it is compared with other packets within the stream to ascertain whether neighboring flags also have the same flag set. For example, the stream was searched for flags within three packets of one another. The 50 attack packets and eight benign packets were all identified, providing a 100% detection rate.

The objective of DoS is to prioritize the packets based on their CLP values and discarding the most likely attack packets. Since an exact prioritization would require offline, multiple-pass operations, e.g., sorting, we take the following alternative approach to realize an online, one-pass operation. First, all the incoming packets during are scored. Second, we construct a cumulative distribution function (CDF) with the CLP scores of the packets. Then a congestion algorithm is used to determine the fraction of arriving packets to be discarded in order to control the utilization of the victim to be below a target value. Once the required packet-discarding percentage is determined, the corresponding CLP discarding threshold is looked up from the CDF of the CLP scores. We then discard a suspicious packet if it's CLP score is below the threshold. While CLP-computation is always performed for each incoming packet, selective packet discarding only happens when the system is operating beyond its safe (target) utilization level target. Otherwise, the overload control scheme will set to zero.

### 4.4.1 Client implementation:

A denial-of-service attack is characterized by an explicit attempt to prevent the legitimate use of a service. A distributed denial-of-service attack deploys multiple attacking entities to attain this goal. This paper is solely concerned with DDoS attacks in the computer realm, perpetrated by causing the victim to receive malicious trace and some damage as a consequence.

One frequently exercised manner to perform a DDoS attack is for the attacker to send a stream of packets to a victim; this stream consumes some key resource, thus rendering it unavailable to the victim's legitimate clients. Another common approach is for the attacker to send a few malformed packets that confuse an application or a protocol on the victim machine and force it to freeze or reboot. In September 2002 there was an onset of attacks that over-loaded the Internet infrastructure rather than targeting special victims. Yet another possible way to deny service is to subvert machines in a victim network and consume some key resource so that legitimate clients from the same network cannot obtain some inside or outside service. This list is far from exhaustive. It is certain that there are many other ways to deny service on the Internet, some of which we cannot predict, and these will only be discovered after they have been exploited in a large attack.

If the average queue size exceeds the maximum threshold, congestion occurs. Prior to dropping packets, as would occur, the packets to be dropped are inspected by the corresponding PF. Within the prototype, this is set to two packets being compared. However, this variable may be set to a different number by a system administrator to ensure effective packet comparisons. A packet to be dropped is drawn from the queue based on the congestion algorithm used by the router. The destination address is compared with the previous inspected packet's destination address, which is stored by the router for this type of comparison. If both destination addresses match, then these packets are passed for stateless signature analysis. If they do not match, the current destination address is stored for comparison with the next packet and the current packet is dropped.

The analysis and simulations measure latency in terms of gossip rounds: We measure the message's propagation time, which is the expected number of rounds it takes a given protocol to propagate a message to all (in the closed-form analysis) or to 99 percent (in the simulations) of the correct processes. We chose a threshold of 99 percent since the message may fail to reach some of the correct processes due to old message purging or link loss. Note that correct processes can be either attacked or non attacked. In both cases, they should be able to send and receive data messages. We turn to measure actual performance on a cluster of workstations. Our goal for this evaluation is twofold: First, we wish to ensure that the simplifying assumptions made in the analysis and simulations have little impact on their results. For example, in the implementation, rounds are not synchronized and the push-offer mechanism is used. Second, we seek to measure the consequences of DoS attacks not only on actual latency, but also on the throughput of a real system, where multiple messages are sent, and old messages are purged from processes message buffers.

### 4.4.2 Enhanced Probabilistic Marking (EPM) module:

This module is an extension of the probabilistic marking module in IP traceback schemes. This lightweight module is running on each participating Internet router, whether or not there are DDoS attacks. Recall that in IP traceback, each Internet router needs to inscribe, with a certain probability, a mark into infrequently-used IP header fields. Our scheme further splits such marks into two types, described later. Packets of the first type are used by the AMD module below for IP traceback and packets of the second type are used by the PPF module below for intelligent filtering.

### 4.4.3 Attack Mitigation Decision-making (AMD) module:

Running on the victim or the border gateway device (e.g., firewall) of the victim site, this module implements two functions: 1) reconstructing attack paths using existing IP traceback algorithms based on information contained in the aforementioned marks of the first type once an attack is detected, and 2) making algorithmic decisions on the probability of dropping a packet based on the mark inscribed into its header and the results from 1). The decisions from 2) will be conveyed to the perimeter routers to be carried out. We will show that there is little communication overhead in transporting such information.

### 4.4.4 Preferential Packet Filtering (PPF) module:

This module is running on every perimeter router. These modules will differentially filter a packet (destined for the victim) that contains the aforementioned marks of the first type, based on the instructions issued to them from the AMD module, once an attack is detected. We will show that little processing overhead is incurred at the perimeter routers: Each filter/pass decision requires only the computation of a hash value and a table lookup.

The proposed scheme extends and leverages on existing IP traceback schemes. In IP traceback, each Internet router inscribes, in a probabilistic way, a mark into a low-entropy IP header field (called "mark field" hereafter) set aside for this purpose. These marks collectively allow the victim to reconstruct the "attack graph," which consists of the network edges that the packets from the attackers have traversed. We can view the marked fields inscribed at the packet headers by IP traceback schemes as a communication channel. IP traceback uses this channel for only one purpose: to convey the encoded information about the "attack graph." Our scheme, on the other hand, splits this channel into the following two subchannels.

One subchannel, called the signaling subchannel, will continue to carry the same information needed for IP traceback. It occupies about 5 percent of the channel bandwidth, i.e., only 5 percent of the marks are signaling marks. This implies that the reconstruction of the whole attack graph will be 20 times slower than in the underlying IP traceback scheme. Fortunately, the preferential filtering becomes very effective in improving the throughput of legitimate traffic as soon as a critical portion of the attack graph is reconstructed. As we will show, the nature of the probabilistic marking used in IP traceback determines that it takes much less (less than 10 percent of the latter) packets/time to obtain this critical portion than obtaining the whole attack graph.

The other subchannel, called the data subchannel, will consume the remaining 95 percent of the channel bandwidth. Information contained in the data subchannel, combined with the "attack graph" reconstructed from the signaling subchannel, will allow the perimeter routers to infer whether the packet is more likely to come from an attacker or a legitimate host. A packet will be preferentially filtered out or passed by the perimeter routers if it is determined to be more likely to come from the attackers or legitimate hosts, respectively.

The marks contained in packet headers will correspondingly be split into two types, namely, signaling marks and data marks. They carry signaling and data subchannel information, respectively. The enhanced marking algorithm that performs such splitting is shown. When a packet pkt arrives at a router, the router decides whether it will overwrite the current mark with marking probability. If it decides to overwrite the mark, the router then decides whether this mark should be a signaling mark or a data mark, with probability, respectively. For a signaling mark, the hop count is maintained in the same way as in IP traceback; it will be reset to 0 if the router overwrites the mark, or simply incremented by 1 otherwise. For a data mark, there is no hop count field.

These approaches to DoS attacks have several weaknesses. Hence, there is a requirement for an alternative, yet complementary, approach to the DoS problem. To meet this challenge, we have proposed a system for early defense against DoS. At the heart of this system is a novel Distributed DoS Detection Mechanism (DiDDeM) providing the means by which DoS attacks are detected early, beyond the perimeter of the network under attack, so as to enable an early propagated response to block the attack through net- work routers, particularly those close to the attack sources. This approach has been implemented as a prototype, further details of which are discussed. This paper is focused mainly on the presentation of the new mechanism design.

# CHAPTER 5
# CONCLUSION AND FUTURE OUTLOOK

The flow of information is the most valuable commodity for organizations and users alike, and DoS attacks pose a great threat to this flow. These attacks are highly prevalent despite the widespread deployment of network security countermeasures such as firewalls and intrusion detection systems. Current countermeasures find DoS extremely problematic, therefore, a number of other approaches have been proposed to counter the problem. However, these approaches are not without their problems, so a new approach to more effective detection and prevention of DoS attacks is required.

This requirement has motivated us to propose the novel distributed mechanism, DiDDeM, for effective early detection and prevention of DoS attacks. In this paper, we have demonstrated that the DiDDeM makes use of stateful and stateless signatures in conjunction for attack detection, which differs from the other related work that mainly employs one of the two signature approaches. The main benefit from the combination of the two approaches is that not all malicious packets have to be inspected in order to ascertain the presence of an attack, thus improving detection efficiency and making attack detection feasible within the routing infrastructure. Moreover, the DiDDeM offers a novel, distributed and scalable approach to attack responses.

Our future work is to take two directions. First, we will refine the DiDDeM-based system for improved defense against DoS attacks. For example, we will continue our implementation and experiment on a wider range of attack scenarios, expand the PF deployment to routers not at the ingress edge of a domain, and further explore congestion algorithms for stateful signature analysis and its closer integration with stateless signature analysis. These will help to reduce false alarm rates and improve the system efficiency. Second, we will exploit the potential of the DiDDeM approach for its application to attack scenarios other than DoS. For example, utilizing the DiDDeM within an organizational boundary may enhance the detection and prevention of network worms, as these programs require a high volume of traffic during their spreading periods.

An analysis of the research systems and experiments with the baseline system suggest that two characteristics of the research systems and of the evaluation led to improved performance of the research systems. First, attack signatures were similar between training and test data. Although new attacks in the test data exploited different system weaknesses, the visible signature in tcp dump data was similar to that in training data. It was primarily caused by attackers creating interactive shells with root-level privilege and by stealthy techniques used to prepare for the attack and run the attack. Second, examples of normal sessions and of both clear and stealthy attacks were provided in training. These examples were used by system developers to create rules or signatures with low false alarm rates and high detection rates. For example, one system specifically extracted features to detect root-level shells including typical root-shell command-line prompts and a string printed out by some of the buffer-overflow attacks to indicate when a root-level shell was successfully created. This same system also used discriminate training, which relied heavily on normal and attack training data, to extract rules with high detection rates and low false alarm rates. Experiments were performed to determine whether selecting new attack-specific keywords and using discriminate training with keyword counts as input features could improve the performance of the simple baseline system. Neither approach alone was sufficient to obtain the good performance demonstrated by the two best research systems. A combination of adding new keywords and using discriminate training, however, increased the performance of the baseline system on attacks to be similar to that of the two best research systems.

# 6 REFERENCES

[1] R. Lippmann, J.W. Haines, D. J. Fried, J. Korba, and K. Das, "The 1999 DARPA off-line intrusion detection evaluation," *Comput. Netw.*, vol. 34, pp. 579–595, 2000.

[2] D. Moore, G. M. Voelker, and S. Savage, "Inferring Internet denial-ofservice activity," in *Proc. 10th Usenix Security Symp.*,Washington, DC, 2001.

[3] R. Richardson, "The eighth annual CSI/FBI computer crime and security survey 2004," Comput. Security Inst./Federal Bureau of Investigation, Tech. Rep., 2003.

[4] J. Haggerty, T. Berry, Q. Shi, and M. Merabti, "DiDDeM: A system for early detection of TCP SYN flood attacks," in *Proc. GLOBECOM 2004*, Dallas, TX, Nov.–Dec. 29–3, 2004, pp. 2037–2042.

[5] T. M. Gil and M. Poletto, "MULTOPS: A data-structure for bandwidth attack detection," in *Proc. USENIX Security Symp.*, Washington, DC, 2001.

[6] C. Douligeris and A. Mitrokotsa, "DDoS attacks and defense mechanisms: Classification and state of the art," *Comput. Netw.*, vol. 44, pp. 643–666, 2004.

[7] A. Kazmanovic and E. W. Knightly, "Low-rate TCP-targeted denial of service attacks," in *Proc. Symp. Commun. Arch. Protocols*, Karlesruhe, Germany, 2003, pp. 345–350.

[8] C. Meadows, "A cost-based framework for analysis of denial of service in networks," *J. Comput. Security*, vol. 9, pp. 143–164, 2001.

[9] J. C. Brustoloni, "Protecting electronic commerce from distributed denial- of-service attacks," in *Proc. WWW2002*, Honolulu, HI, 2001, pp. 553–561.

[10] P. Papadimitratos and Z. J. Haas, "Securing the Internet routing infrastructure," *IEEE Commun. Mag.*, vol. 40, pp. 76–82, Oct. 2002.

[11] S. Shyne, A. Hovak, and J. Riolo, "Using active networking to thwart distributed denial of service attacks," in *Proc. IEEE Aerosp. Conf.*, Big Sky, MT, 2001, pp. 3/1103–3/1108.

[12] D. Sterne, K. Djahandari, R. Balupari, W. La Cholter, B. Babson, B. Wilson, P. Narasimhan, and A. Purtell, "Active network based DDoS defense," in *Proc. DARPA Active Netw. Conf. Expo.*, San Francisco, CA, 2002, pp. 193–203.

[13] J. Ioannidis and S. M. Bellovin, "Implementing pushback: Router-based defense against DDoS attacks," in *Proc. Netw. Distrib. Syst. Security Symp.*, San Diego, CA, 2002.

[14] A. Hussain, J. Heodemann, and C. Papadopoulos, "A framework for classifying denial of service attacks," in *Proc. Symp. Commun. Arch. Protocols*, Karlesruhe, Germany, 2003, pp. 99–110.

[15] J. Haggerty, Q. Shi, and M. Merabti, "Statistical signatures for early detection of flooding denial-of-service attacks," in *Proc. IFIP/SEC*, Chiba, Japan, May-Jun. 30–1, 2005, pp. 327–342.

# APPENDICES

## Appendix A

## Source Code:

**Sample Coding C² server:**

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.sql.*;
import java.lang.*;
import java.util.*;

public class C2Server implements ActionListener
{
se s;
C2ServerThread td;
C2Stateless st;
C2Statefull stf;
Cserver_router cr;
Cserver_prefilter cp;

JTextArea output;
JScrollPane scrollPane;
boolean flag;

JMenuBar menuBar;
JMenu menu,menu1,menu2,menu3,menu4;
JMenuItem menuItem1,menuItem2,menuItem3,mi4,mi5,mi6,mi7,mi8,mi9,mi10;

public JMenuBar createMenuBar()
{

//Create the menu bar.
```

```
menuBar = new JMenuBar();


//Build the first menu.
menu = new JMenu("Actions");
menu.setMnemonic(KeyEvent.VK_O);
menu.getAccessibleContext().setAccessibleDescription("The only menu in this
program that has menu items");
menuBar.add(menu);

//a group of JMenuItems
menuItem1 = new JMenuItem("Start",KeyEvent.VK_S);
//menuItem.setMnemonic(KeyEvent.VK_T); //used constructor instead
menuItem1.setAccelerator(KeyStroke.getKeyStroke(
KeyEvent.VK_1, ActionEvent.ALT_MASK));
menuItem1.getAccessibleContext().setAccessibleDescription("This doesn't really do
anything");
menuItem1.addActionListener(this);
menu.add(menuItem1);


//a group of JMenuItems
menuItem2 = new JMenuItem("Stop",KeyEvent.VK_T);
//menuItem.setMnemonic(KeyEvent.VK_T); //used constructor instead
menuItem2.setAccelerator(KeyStroke.getKeyStroke(
KeyEvent.VK_1, ActionEvent.ALT_MASK));
menuItem2.getAccessibleContext().setAccessibleDescription("This doesn't really do
anything");
menuItem2.addActionListener(this);
menu.add(menuItem2);

//a group of JMenuItems
menuItem3 = new JMenuItem("Exit",KeyEvent.VK_E);
frame.setContentPane(c.createContentPane());
```

```
//Display the window.

frame.setBounds(300,150,550,550);
frame.setVisible(true);
}

public static void main(String[] args)
{
//Schedule a job for the event-dispatching thread:
//creating and showing this application's GUI.
javax.swing.SwingUtilities.invokeLater(new Runnable()
{

public void run()
{
createAndShowGUI();
}
});
}
}
```

**Sample coding of Router:**

```
import java.awt.*;
import java.applet.*;
import javax.swing.*;
import java.awt.event.*;
import java.net.* ;
import java.io.* ;
import java.lang.* ;
import java.util.* ;
import java.sql.*;

//main class
```

```
public class Router extends JFrame
{

public Router()
{
Container c=getContentPane();

JTabbedPane tp=new JTabbedPane();

tp.add("Register",new Register());
tp.add("Action", new Action());
tp.add("Requests", new Requests());
tp.add("Virus Signatures", new SignatureList());
tp.add("settings",new settings());
tp.add("Help", new Help());
tp.add("Exit",new Exit());



c.add(tp);

}

public static void main(String args[])
{
Router rout=new Router();
rout.setTitle("Router");
rout.pack();

rout.setBounds(200,50,650,650);
rout.setVisible(true);
}
}


//For Registration tab


class Register extends JPanel
{

add a;
```

```
Cancel c;
Modify m;
View v;


Register()
{

JButton b1=new JButton("Add");
add(b1);

JButton b2=new JButton("Cancel");
add(b2);

JButton b3=new JButton("Modify");
add(b3);

JButton b4=new JButton("View");
add(b4);



b1.addActionListener(new ActionListener()
{
public void actionPerformed(ActionEvent e)
{
a=new add();
}
});


b2.addActionListener(new ActionListener()
{
public void actionPerformed(ActionEvent e)
{
c=new Cancel();
}
});



b3.addActionListener(new ActionListener()
{
public void actionPerformed(ActionEvent e)
{
m=new Modify();

}
});
b4.addActionListener(new ActionListener()
{
public void actionPerformed(ActionEvent e)
{
v=new View();
}
});

}

}
```

**Sample coding of prefilter:**

```
import java.net.* ;
import java.io.* ;
import java.lang.* ;
import java.util.* ;
import java.sql.*;
public class ChatServerThread extends Thread
{
static final int BUFFER_SIZE = 512;    //Sets Max Packet Size
private DatagramPacket IncomingPacket;    //Socket to Communicate with
ClientsTableEntry CTE;
DatagramSocket OutToClientSocket;
Hashtable ClientsTable;
byte[] RecievedBuffer = new byte[BUFFER_SIZE];
byte[] SendAllBuffer = new byte[BUFFER_SIZE];
byte[] SendPrivateBuffer = new byte[BUFFER_SIZE];

//variables for statfull operaion

String arr[]=new String[100];
String arr1[]=new String[100];
String arr2[]=new String[100];
String temp="";
int temp_count=0;
String sign="";
String sign1;
String SendPrivateString;
String SendAllString;
String Blank = " ";
String Username;
String RecieverName;
String SenderName;
String Message;

InetAddress RecieverIPAddress;
InetAddress ClientIPAddress;
int RecieverPort;
int ClientPort;
boolean check=false;
Socket s;
String message;
BufferedReader in;
PrintWriter out;
int length=0;
int i;
public      ChatServerThread(DatagramPacket      ArrivedPacket,      Hashtable
MasterClientsTable, Socket s1)

{

//Create buffers to hold incoming & outgoing messages
SendPrivate = false;
SendAll = true;
}
```

```
}

//Incoming Packet is a Message Packet: Must validate Sender and Reciever,
//or decide if the message is to be broadcast

else if (PacketType == 3)
{

System.out.println("conversation between clients");
if(ULength==0)
{
System.out.println("Message is to be broadcast");

}
catch(Exception ae)
{
}

}
else
{
SendPrivateBuffer = MakePacket(3, SenderName, Message);
SendPrivate = true;
SendAll = false;
PrivateTo = true;
try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con=DriverManager.getConnection("jdbc:odbc:dsn","","");
Statement stmt=con.createStatement();
boolean checkk3=CheckSenderID(ClientIPAddress,ClientPort);
boolean checkk4=CheckRecieverID(Username);
stmt.executeUpdate("insert                    into                    IntraDomain
values('"+SenderName+"','"+ClientIPAddress+"',"+ClientPort+",'"+RecieverName+"',
'"+CTE.GetClientIP()+"','"+Message+"')");
stmt.executeUpdate("insert                    into                    IntraDomain_full
values('"+SenderName+"','"+ClientIPAddress+"',"+ClientPort+",'"+RecieverName+"',
'"+CTE.GetClientIP()+"','"+Message+"')");
System.out.println("Values inserted sucessfully");
stmt.close();
con.close();
}
catch(Exception a)
{
}
}
}

//packet exceeds the queue size
SendAll = false;
}
}
}
```

**Sample coding of Client:**

```
import javax.swing.UIManager;
import java.awt.*;
import java.net.*;
import java.util.*;
import java.io.*;
public class ClientInterface {
boolean packFrame = false;
DatagramSocket ChatSocket;
/**Construct the application*/
public ClientInterface() {
try{
```

```
//create the master thread for the new client interface
ChatSocket = new DatagramSocket();
}
catch (SocketException se)
{
System.err.println(se);
}
catch (IOException e)
{
System.err.println(e);
}
//create the new UI frame and recieving thread.
/**Main method*/
public static void main(String[] args) {
try
{
UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
}
catch(Exception e)
{
e.printStackTrace();
}
new ClientInterface();
}
}
```

## Appendix B

### SCREENSHOTS:

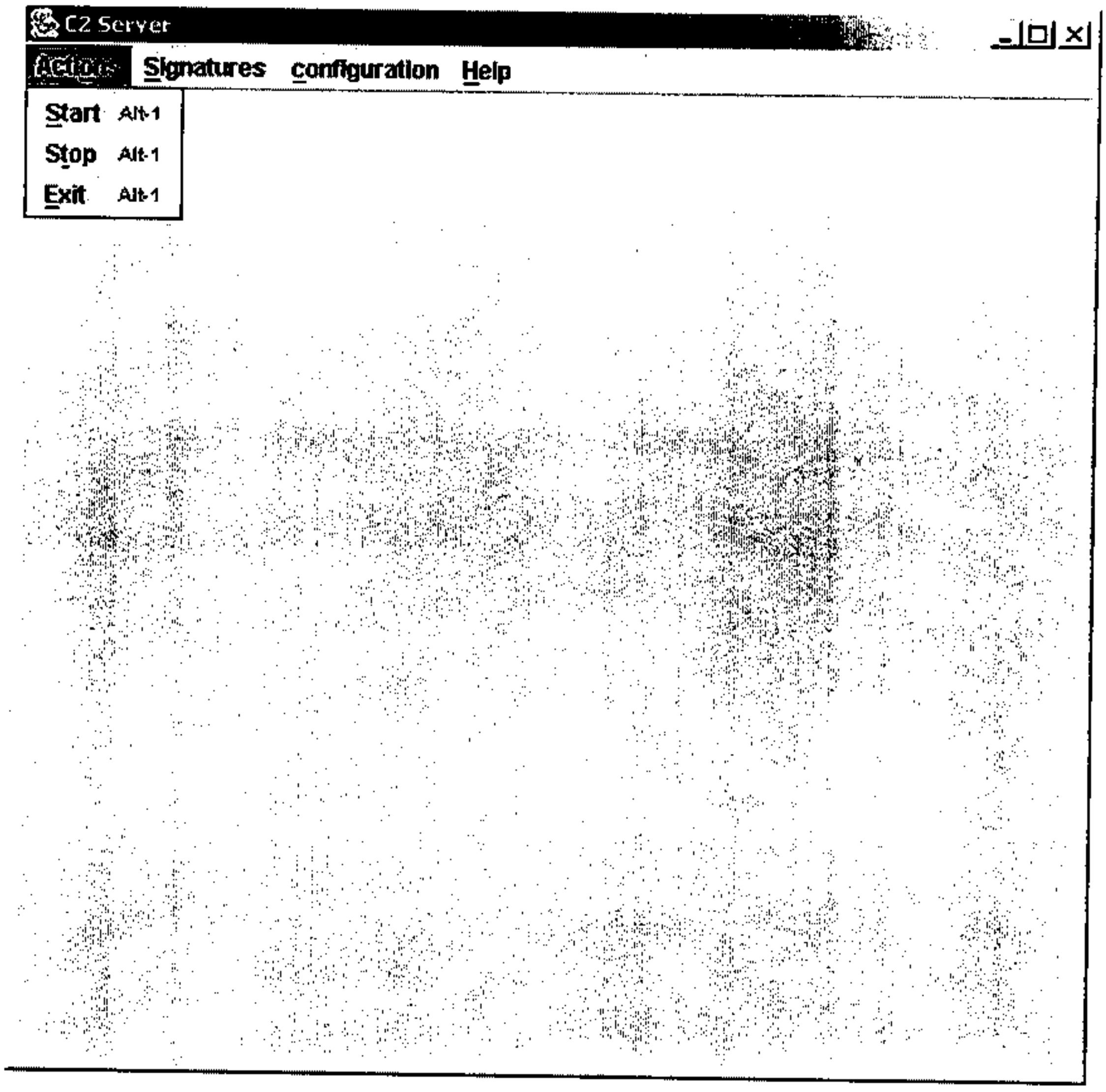
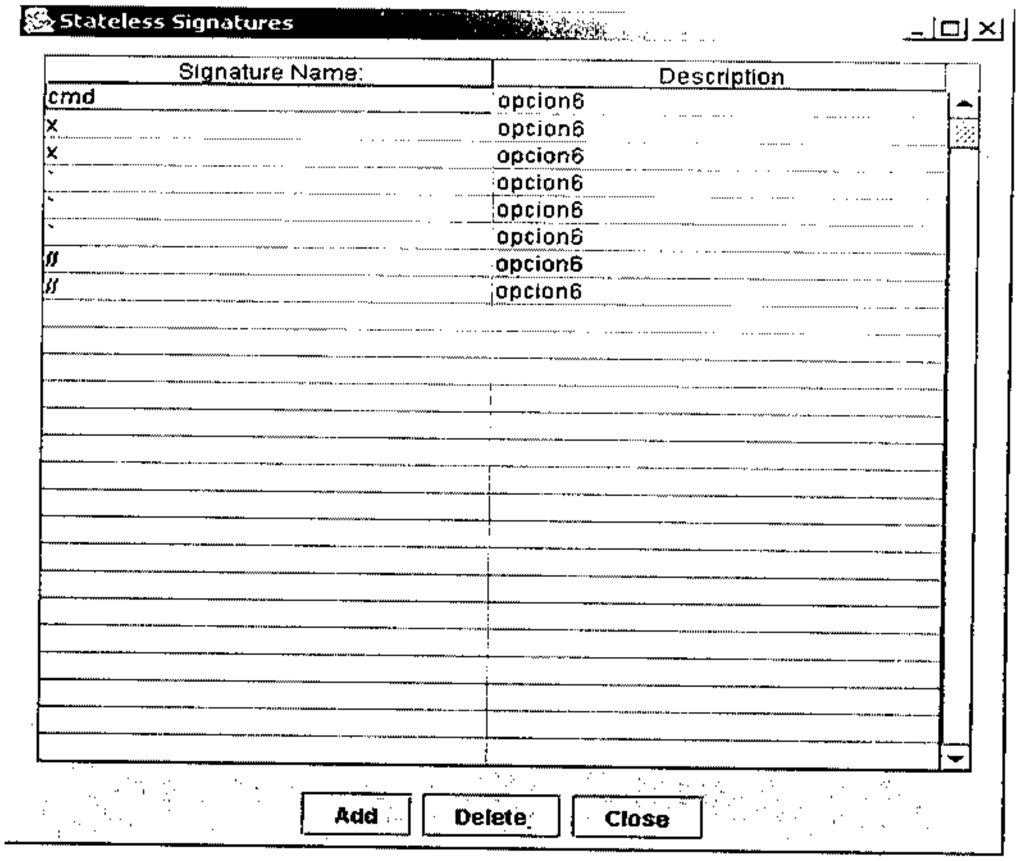
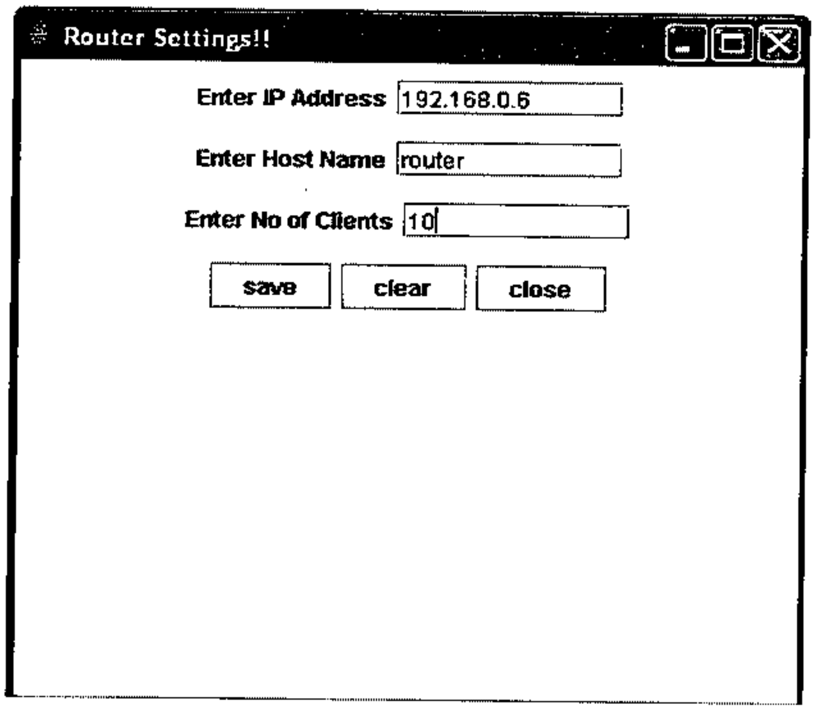


Fig 7.1: $C^2$ Server Actions



| Signature Name: | Description |
|---|---|
| cmd | opcion6 |
| x | opcion6 |
| x | opcion6 |
| ` | opcion6 |
| ` | opcion6 |
| ` | opcion6 |
| // | opcion6 |
| // | opcion6 |

Fig 7.2: Stateless Signature

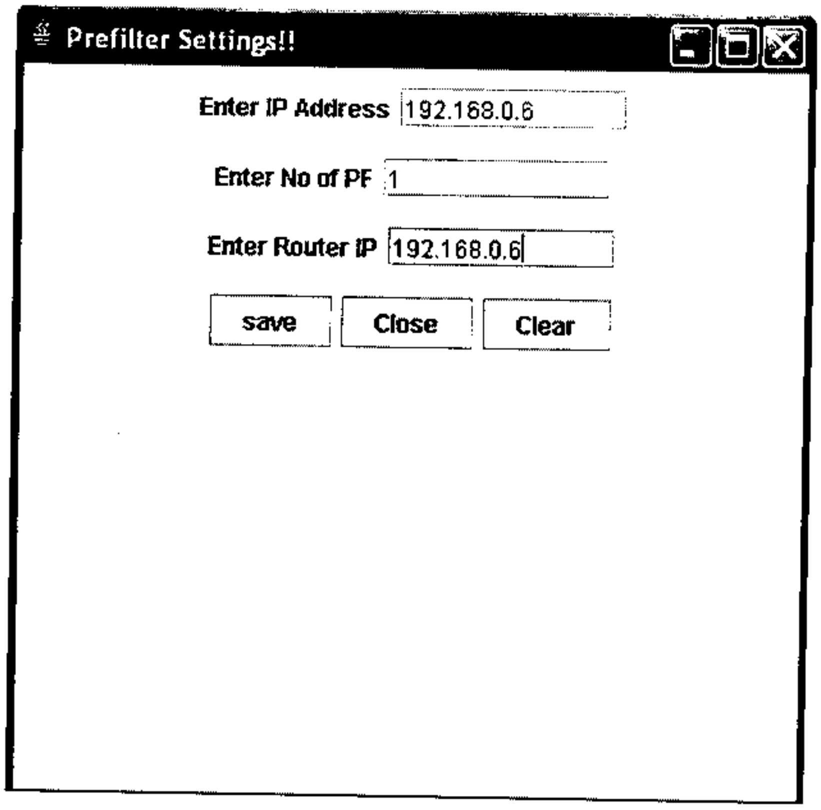


Enter IP Address 192.168.0.6

Enter Host Name router

Enter No of Clients 10

Fig 7.3: Router Settings

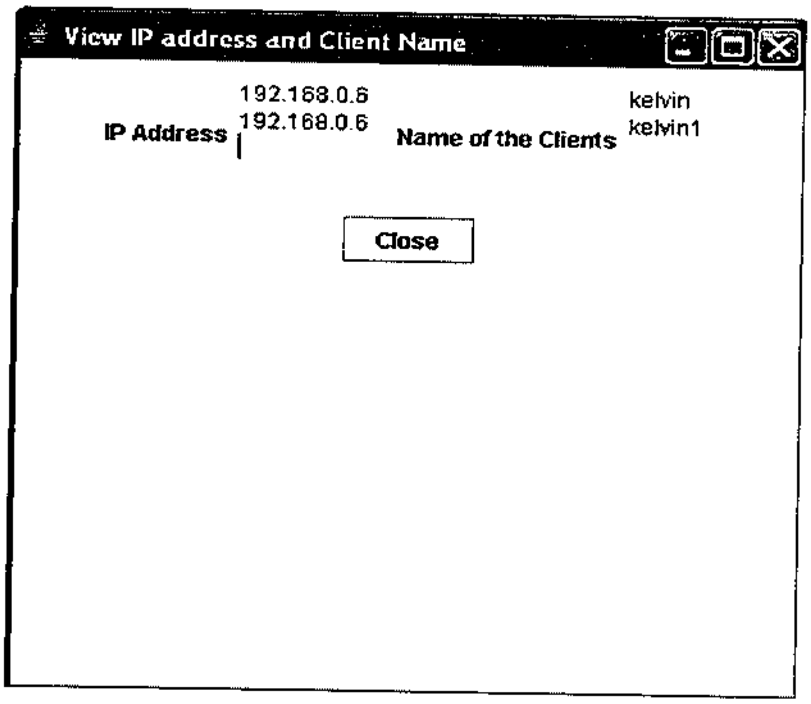

Prefilter Settings!!

Enter IP Address 192.168.0.6

Enter No of PF 1

Enter Router IP 192.168.0.6

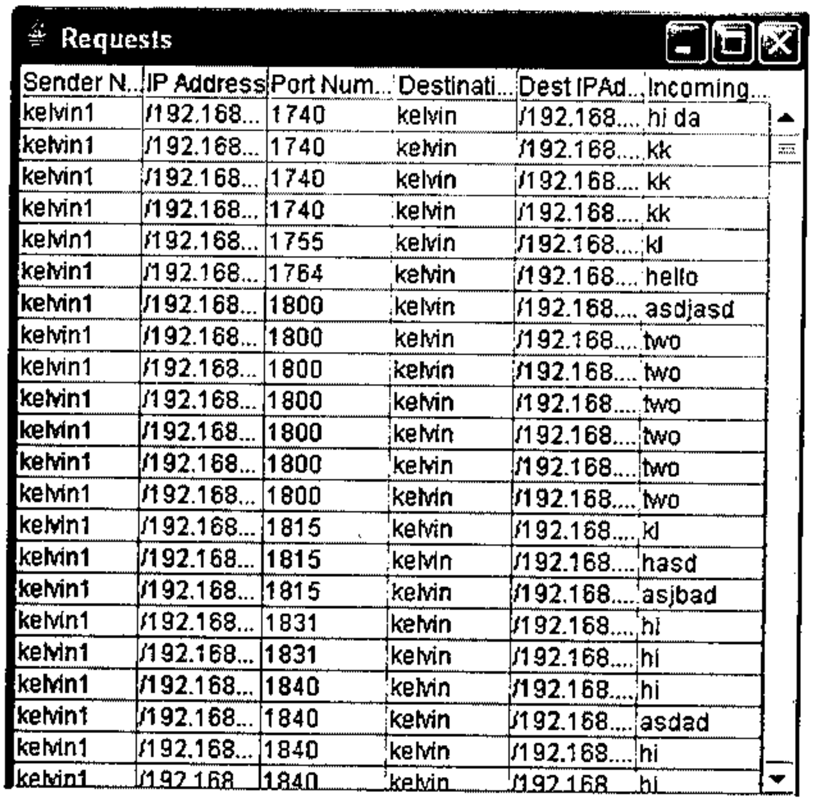
Fig 7.4: PreFilter Settings


View IP address and Client Name

IP Address 192.168.0.6 192.168.0.6

Name of the Clients kelvin kelvin1

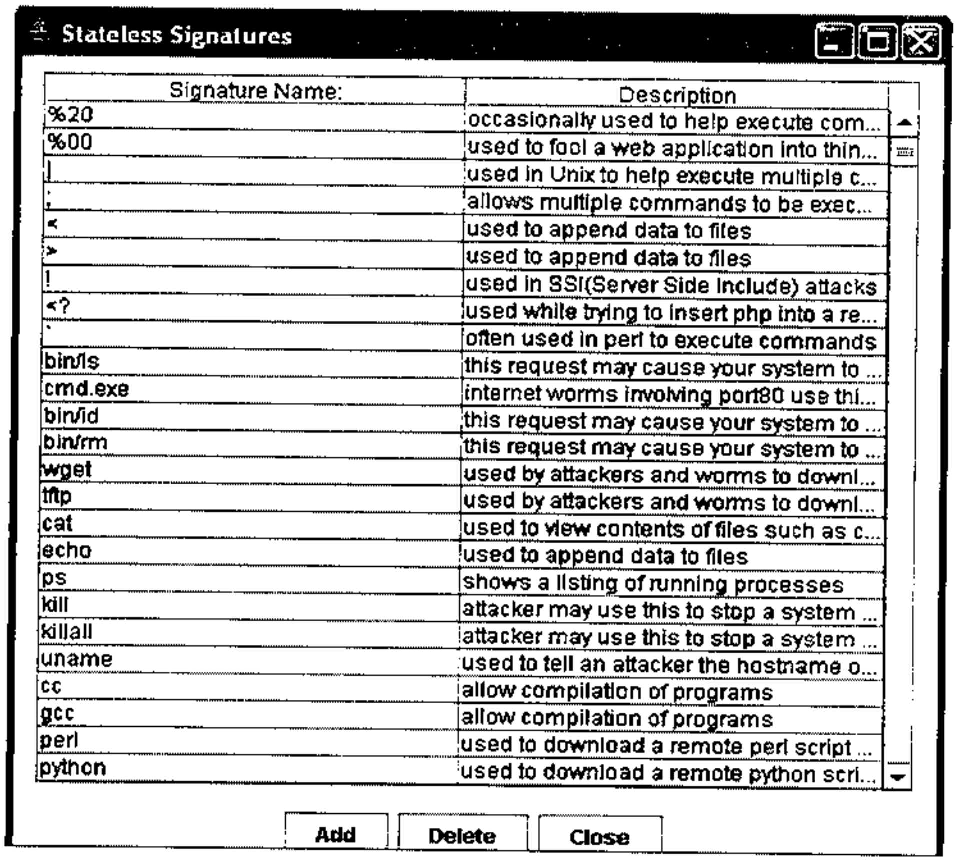
Fig 7.5: Client IP address and client name



Requests

| Sender N... | IP Address | Port Num... | Destinati... | Dest IPAd... | Incoming... |
|---|---|---|---|---|---|
| kelvin1 | /192.168... | 1740 | kelvin | /192.168.... | hi da |
| kelvin1 | /192.168... | 1740 | kelvin | /192.168.... | kk |
| kelvin1 | /192.168... | 1740 | kelvin | /192.168.... | kk |
| kelvin1 | /192.168... | 1740 | kelvin | /192.168.... | kk |
| kelvin1 | /192.168... | 1755 | kelvin | /192.168.... | kl |
| kelvin1 | /192.168... | 1764 | kelvin | /192.168.... | hello |
| kelvin1 | /192.168... | 1800 | kelvin | /192.168.... | asdjasd |
| kelvin1 | /192.168... | 1800 | kelvin | /192.168.... | two |
| kelvin1 | /192.168... | 1800 | kelvin | /192.168.... | two |
| kelvin1 | /192.168... | 1800 | kelvin | /192.168.... | two |
| kelvin1 | /192.168... | 1800 | kelvin | /192.168.... | two |
| kelvin1 | /192.168... | 1800 | kelvin | /192.168.... | two |
| kelvin1 | /192.168... | 1800 | kelvin | /192.168.... | two |
| kelvin1 | /192.168... | 1815 | kelvin | /192.168.... | kl |
| kelvin1 | /192.168... | 1815 | kelvin | /192.168.... | hasd |
| kelvin1 | /192.168... | 1815 | kelvin | /192.168.... | asjbad |
| kelvin1 | /192.168... | 1831 | kelvin | /192.168.... | hi |
| kelvin1 | /192.168... | 1831 | kelvin | /192.168.... | hi |
| kelvin1 | /192.168... | 1840 | kelvin | /192.168.... | hi |
| kelvin1 | /192.168... | 1840 | kelvin | /192.168.... | asdad |
| kelvin1 | /192.168... | 1840 | kelvin | /192.168.... | hi |
| kelvin1 | /192.168 | 1840 | kelvin | /192.168 | hi |

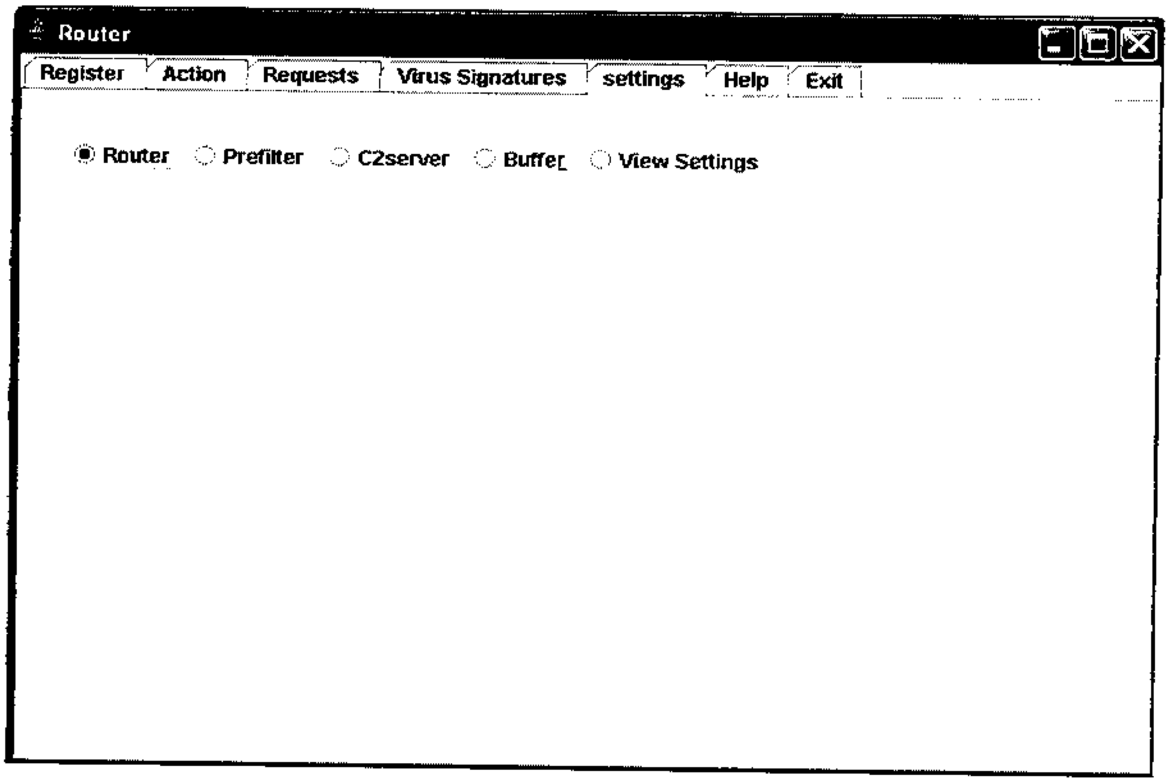
Fig 7.6: Request for Packet transfer



Stateless Signatures

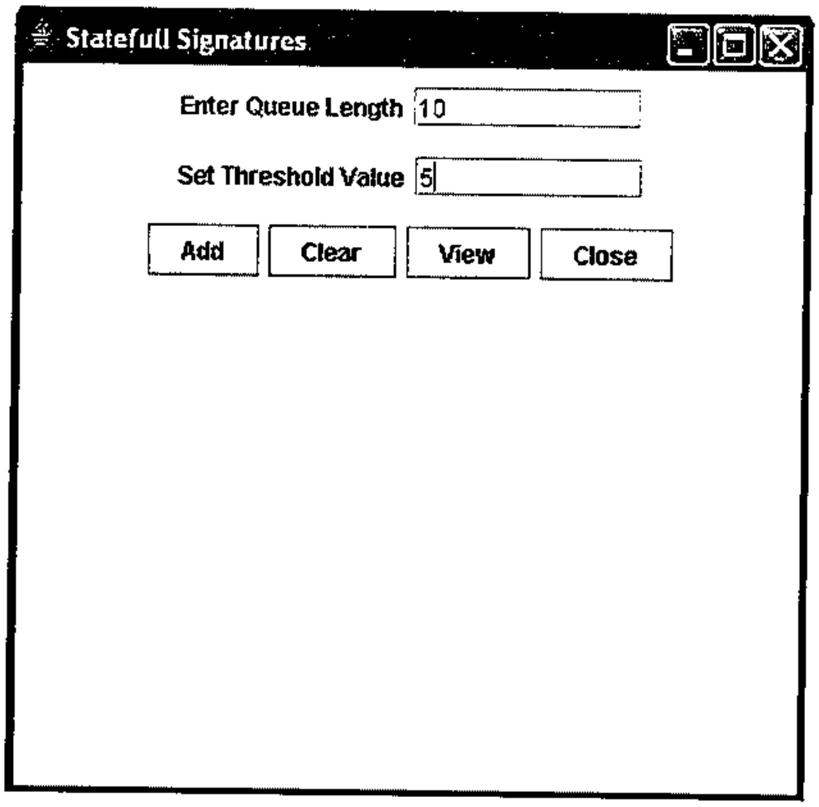
| Signature Name: | Description |
|---|---|
| %20 | occasionally used to help execute com... |
| %00 | used to fool a web application into thin... |
| \| | used in Unix to help execute multiple c... |
| ; | allows multiple commands to be exec... |
| < | used to append data to files |
| > | used to append data to files |
| ! | used in SSI(Server Side Include) attacks |
| <? | used while trying to insert php into a re... |
| ` | often used in perl to execute commands |
| bin/ls | this request may cause your system to ... |
| cmd.exe | internet worms involving port80 use thi... |
| bin/id | this request may cause your system to ... |
| bin/rm | this request may cause your system to ... |
| wget | used by attackers and worms to downl... |
| tftp | used by attackers and worms to downl... |
| cat | used to view contents of files such as c... |
| echo | used to append data to files |
| ps | shows a listing of running processes |
| kill | attacker may use this to stop a system ... |
| killall | attacker may use this to stop a system ... |
| uname | used to tell an attacker the hostname o... |
| cc | allow compilation of programs |
| gcc | allow compilation of programs |
| perl | used to download a remote perl script ... |
| python | used to download a remote python scri... |

Fig 7.7: Stateless Signature Description



[● Router] ○ Prefilter ○ C2server ○ Buffer ○ View Settings

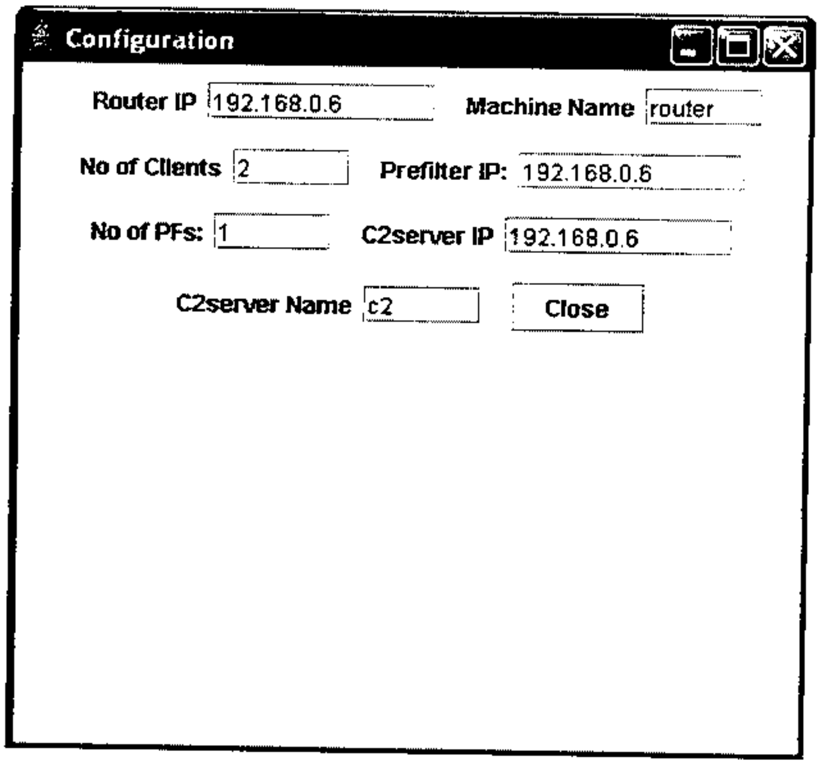
Fig 7.8: Settings view of Router



Enter Queue Length 10

Set Threshold Value 5

Fig 7.9: Statefull Signatures


Configuration

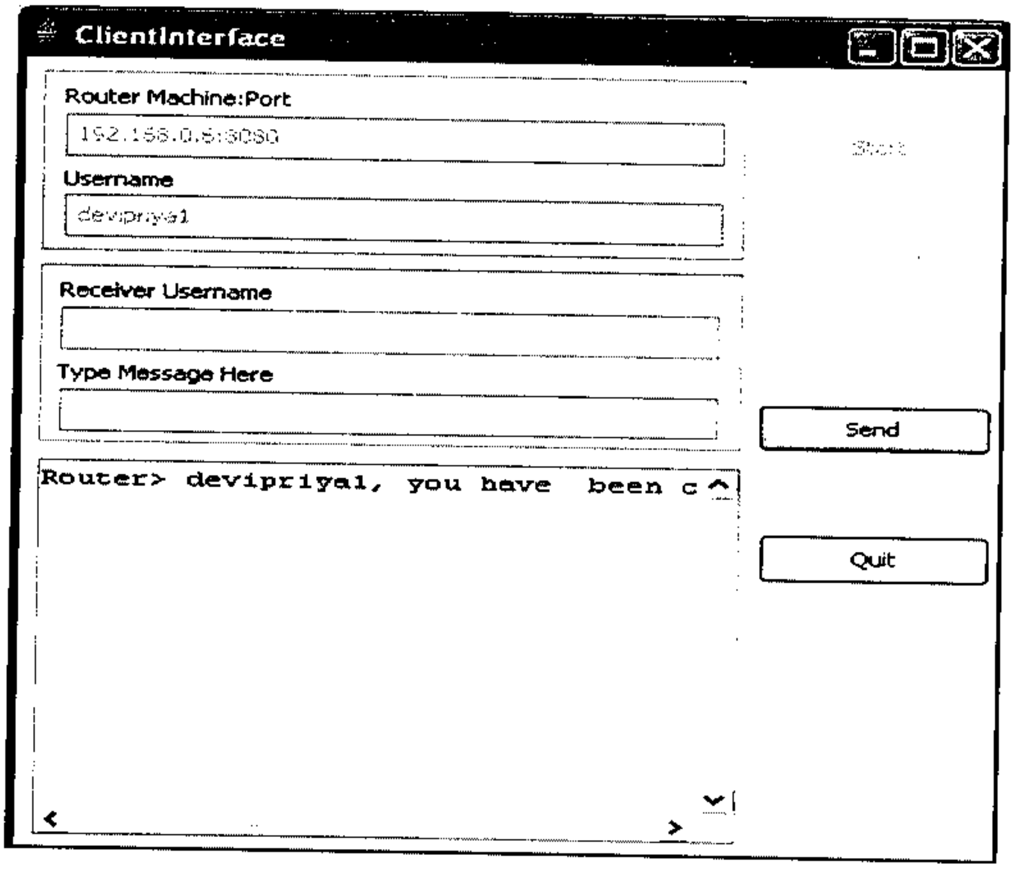
Router IP 192.168.0.6 Machine Name router

No of Clients 2 Prefilter IP: 192.168.0.6

No of PFs: 1 C2server IP 192.168.0.6

C2server Name c2

Fig 7.10: Configuration of Router,Prefilter,C² server


ClientInterface

Router Machine:Port
192.168.0.6:8080

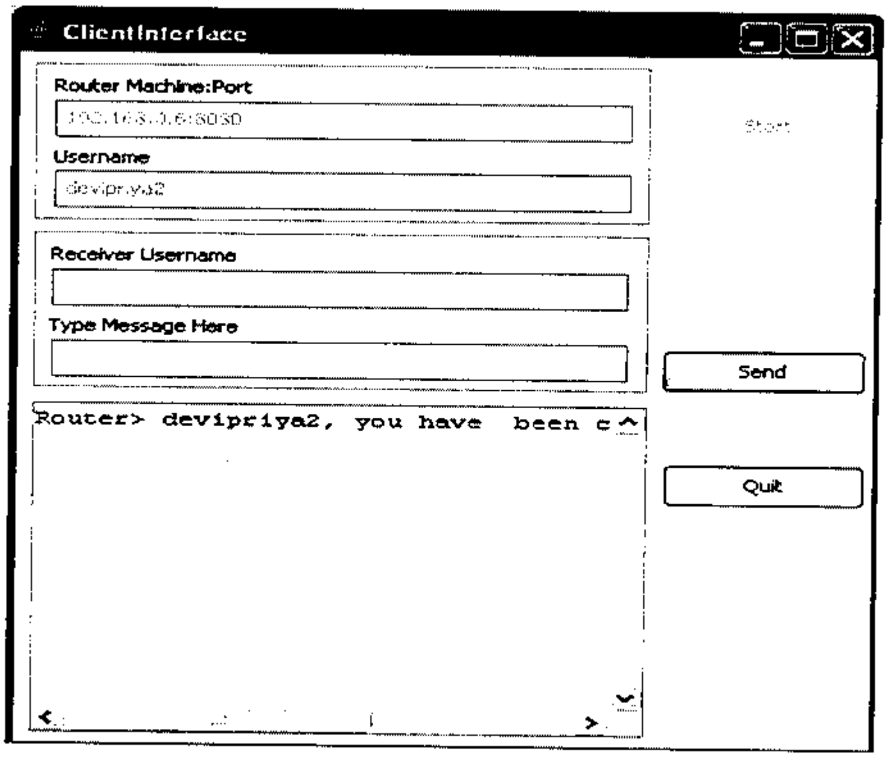
Username
devipriya1

Receiver Username

Type Message Here

```
Router> devipriya1, you have  been c
```

Fig 7.11: Client 1 Interface



192.168.0.6:8080

devipriya2

```
Router> devipriya2, you have  been c
```

Fig 7.12: Client 2 Interface