CHAPTER 1

INTRODUCTION

CHAPTER 1
INTRODUCTION

1.1 Background

DDOS attacks first came to public attention in the first quarter of 2000 when a number of prominent US web sites were attacked. In 2001 the first version of the Code Red worm was designed so that hosts compromised by the worm would perform a denial of service attack on the White House in USA. More recently, the Slapper worm, which targeted Linux machines in September 2002, was designed to create a massive denial of service network of infected machines controllable by the virus writer.

Given the wide availability of DDOS tools there is a significant risk that, if not properly protected, a large number of machines could be used to mount a DDOS attack. Too many machines have not been properly patched or protected and are exploitable as 'zombies'. They are also vulnerable to the attentions of 'script-kiddies', whose aim is to infect as many machines as possible. The threat of DDOS attacks occurring however will then depend on the intentions of the attacker controlling the DDOS network.

To cite a few cases, on February 7,2000 the DDOS attack led to shutdown of major commercial websites like Amazon, buy.com, CNN, eBay, Yahoo etc. just at one go. On July 20, 2004 Russian and British police arrested an extortion group in St Petersburg for carrying out DOS attack for money for extortion. In the US, where DOS attacks is most common, the government has made DOS attack a serious federal crime under the National Information Infrastructure protection Act of 1996 with penalties that include years of imprisonment and many other countries are thinking of adopting similar Laws. Though the goal of this attack is not to take control of the computer like other virus attacks rather it is constraining a legitimate user from accessing a resource. DOS attacks get very little respect from the Hackers community because these attacks are relatively easier to implement but they can be highly effective. Damage due to DOS to organizations is very large, second only to damage due to theft of propriety information. Therefore it is currently the most expensive computer crime for victim organizations.

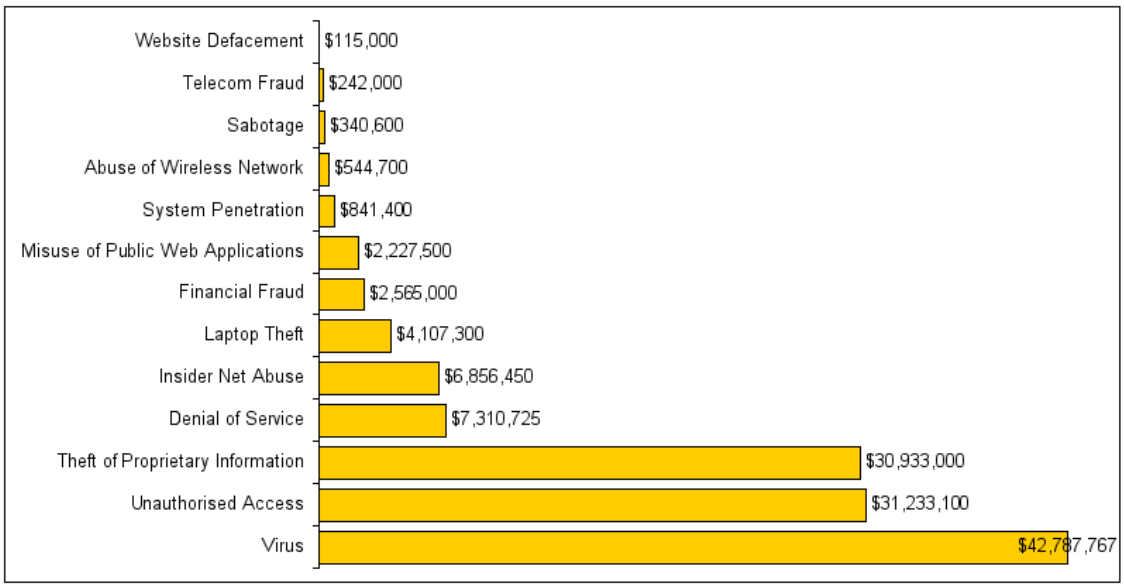

Figure 1.1: Report Indicating Loss Due to Computer Crimes [21]

1.2 Motivation

Network security breaches have become a major threat to business institutions providing online services and in the end costs in the tune of billions of dollars every year. As per statistical details by CERT, the rates of network attacks have gone up from a mere 171 vulnerabilities in 1995 to 7236 in 2007 and the attack reached a staggering level of 4110 in the first half of 2008. The list might not represent the actual no. of breaches as most of them go unreported. Denial-of-Service attacks are among the various forms of attacks with possibilities of highest level of harm in simplest possible form. This "Denial of Service" could impact the network in following forms.

- Occupying the already scarce resources of network further limiting its availability for legitimate users.
- Damaging the network configuration through overloading resulting malfunctioning of various software and hardware components of the network.
- To systematically restrict the access of a particular individual to a service.
- To interrupt a specific service or the system that provides that service.

Hence the main motive of all such attacks is to obstruct the functioning of a service provider as a potential competitor in online business or preventing an individual from getting the benefits of a given service.

The design of the internet makes it more vulnerable and hence a heaven for DDOS attackers. Hence the complex part of this network falls at the two ends with one being the server and the other being the client. So, the misbehaviour at one end could easily affect the other end as the intermediate entities would do almost nothing to protect them from further damage. Tools like firewall can protect the victim but still it has to respond to be requests that are being sent from various units that include both legitimate as well as attacker and here DDOS attackers make their impact as this form of attack relies on intermediate component of the internet and hence easily impact the internet. As the significant change in the performance of the net as a whole.

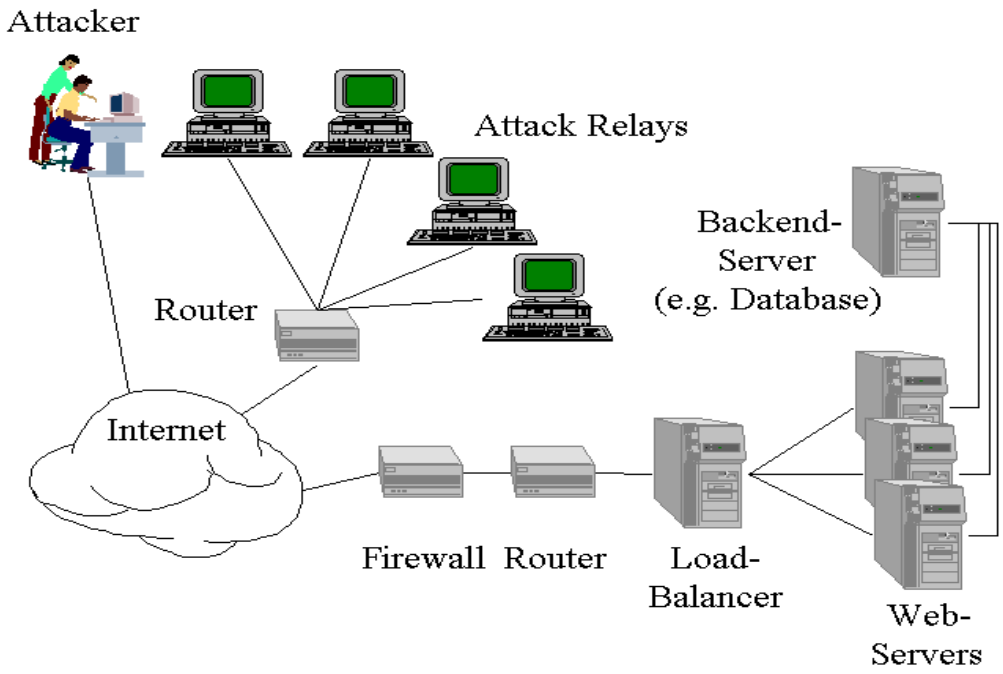

Figure 1.2: Illustration of DDOS attack scenario [22]

This report concentrates on the fundamental understanding of router throttling as a mechanism against DDOS attacks. Further advancement has been made over control-theoretic model that makes it useful for better understanding of the system behaviour under the medley of limits as well as operating conditions. The implementation of an adaptive throttle algorithm that can enhance the security features of the server as well as protect it from further resource overload and makes it free enough for further requests from legitimate users.

1.3 Attack Network

Many recent DOS attacks (also called DDOS attacks) were launched from distributed attacking hosts. A DDOS attack is launched in two phases. First, an attacker builds an attack network which is distributed and consists of thousands of compromised computers (called zombies, bots, or attacking hosts). Then, the attacking host flood a tremendous volume of traffic towards victims either under the command of the attacker or automatically. To build an attack network, the attacker looks for computers that are poorly secured, such as those not having been properly patched. In general, a vulnerable host can be compromised via two types of approaches. One is to entice users to run malicious programs, such as a virus, a spy ware or a Trojan horse carried in malicious emails, files, or web pages. The other approach is via automated malicious programs, such as worms that can automatically scan vulnerable remote computers. The vulnerability in these computers is then exploited to allow the attacker to break into and install DOS attacking programs that further scan other hosts, install backdoors and flood packets. The attacker is thus called the master of these compromised computers (zombies). Some DOS programs have the ability to register the compromised computers as a member in the attack network controlled by the attacker. In addition, the newly compromised computers will automatically repeat the scanning and exploiting process to look for other vulnerable computers. Because of self-propagation, a large attack network can quickly be built to include hundreds or thousands of computers.

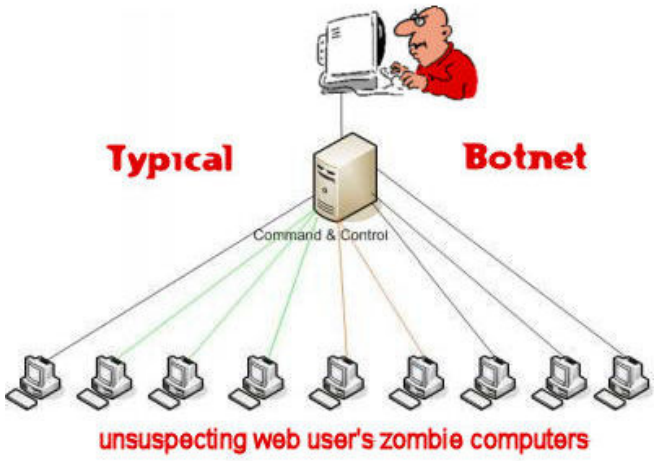

Figure 1.3: Attack Network (BotNet) [23]

BotNet is an example of attack networks (depicted in Figure1.3) in which an attacker controls a large number of zombies. In the attack network, zombies are called hosts. They can communicate with the attacker and synchronize their actions via covert channels. My Doom virus provides another example of building such a DOS attack network different from BotNets. The attack network was not solely built through technological vulnerabilities. The attacking program in compromised computers was designed to automatically and simultaneously flood towards www.sco.com on February 1, 2004 and www.microsoft .com on February 3, 2004.

1.4 Attack Scenario

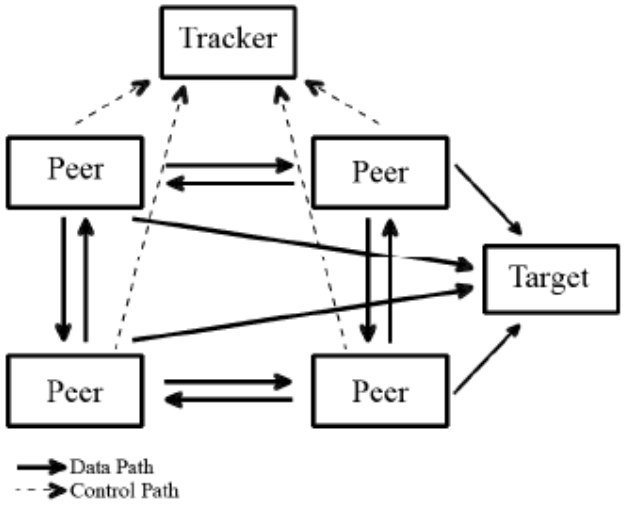

Figure 1.4: Overview of attack scenario [24]

A DDOS is a variant of the DOS attack, except that it involves the use of several attacking computers. In other words, in a DOS attack there is a single stream of attack traffic but in a DDOS attack there are multiple streams of attack traffic. Sometimes a single attacking in computer (i.e sync-flood, Land attack), these attacks are commonly referred to as DOS attacks. Although, the goal of the attack is primarily to consume resources of a victim, the attack will always be more effective with the use of multiple attacking computers. In this case, these attacks are referred to as DOS attacks. The primary goal of DOS attack is to overwhelm the victim server and its secondary goal is to consume bandwidth of the network surrounding the victim. A DDOS attack typically consists of an attacker (or multiple attackers), several handler computers and many attack zombies. During the DDOS attack setup phase, the attacker(s) will relentlessly probe many computers looking for various weaknesses in their systems and then make attempts to infiltrate them and convert them to handlers or zombies. When an attacker

gains control of the computer, it has the ability to communicate back with itself and other zombies. Recent attacks have shown that attackers are becoming more sophisticated recruiting zombies and how they are commanded following their conversion. It is conceivable that an attacker could recruit zombies over a period of several years and then when they have accumulated a large number begin flooding packets in the direction of the victim.

When the attack begins, the attackers notify the handlers to invoke the zombies to begin flooding useless data towards the victim. It should be noted that the traffic is not always useless data; an attacker may disguise their traffic to resemble legitimate traffic to avoid any filtering mechanism that is designed to detect attack packets. In Figure1.4 there is an image of typical DDOS attack scenario. The victim is the end host or the routers that are adjacent to the end host. Depending on the intensity of the attack, the end host may become overwhelmed with incoming traffic or the adjacent routers will become overwhelmed with traffic. Nonetheless, the victim in the attack is the server in addition to the countless number of users that are no longer able to communicate with the server due to the severe congestion. Due to the nature of the attack, this attack is commonly referred to as a *flooding attack.*

1.5 DDOS Attack Architecture

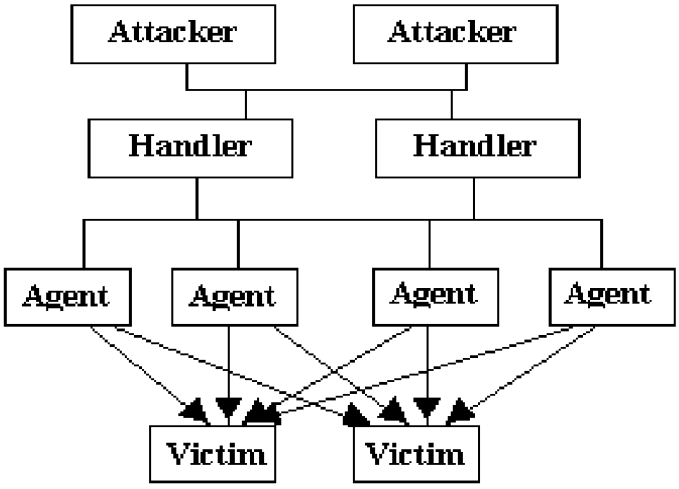

Fig 1.5: DDOS Agent-Handler Attack Model [25]

The Agent-Handler model of a DDOS attack consists of clients, handlers and agents (Figure 1.5). The client is where the attacker communicates with the rest of the DDOS attack system. The handlers are software packages located throughout the Internet that the attacker's client uses to communicate with the agents. The attacker communicates with any number of handlers

to identify which agents are up and running, when to schedule attacks or when to upgrade agents. The owners and users of the agent systems typically have no knowledge that their system has been compromised and will be taking part in a DDOS attack. Depending on how the attacker configures the DDOS attack networks, agents can be instructed to communicate with a single handler or multiple handlers.

CHAPTER 2

DISTRIBUTED DENIAL OF SERVICE – AN INTRODUCTION

2.1 Denial of Service Attack: An Overview

In a denial-of-service (DOS) attack, an attacker attempts to prevent legitimate users from accessing information or services. By targeting your computer and its network connection or the computers and network of the sites you are trying to use. An attacker may also be able to prevent you from accessing email, web sites, online accounts (banking etc.) or other services that rely on the affected computer.

The most common and obvious type of DOS attack occurs when an attacker "floods" a network with information. The server can only process a certain number of requests at once, so if an attacker overloads the server with requests, it can't process your request. This is a "denial-of-service" because you can't access that site. The Denial of Service attack is one of the major internet security threats. The other threat includes eavesdropping (snooping/sniffing), inserting messages, impersonation, and hijacking. The denial of service attack is considered to as threatening as other security attacks.

As the primary objective of DOS attack is *"non availability of resources"* to the legitimate user, it can be termed as an event in which a user or organization is deprived of the services of a resource they would normally expect to have. Typically, the loss of service is the inability of a particular network service, such as e-mail, to be available or the temporary loss of all network connectivity and services. Denial of Service Attack can also destroy programming and files in a computer system.

2.2 Distributed denial-of-service (DDOS) attack: a DOS variant

In a distributed denial-of-service (DDOS) attack, an attacker may use your computer to attack another computer. By taking advantage of security vulnerabilities or weaknesses, an attacker could take control of your computer. He or she could then force your computer to send huge amounts of data to a website or send spam to particular email address. The attack is "distributed" because the attacker is using multiple computers, including yours, to launch the denial-of-service attack.

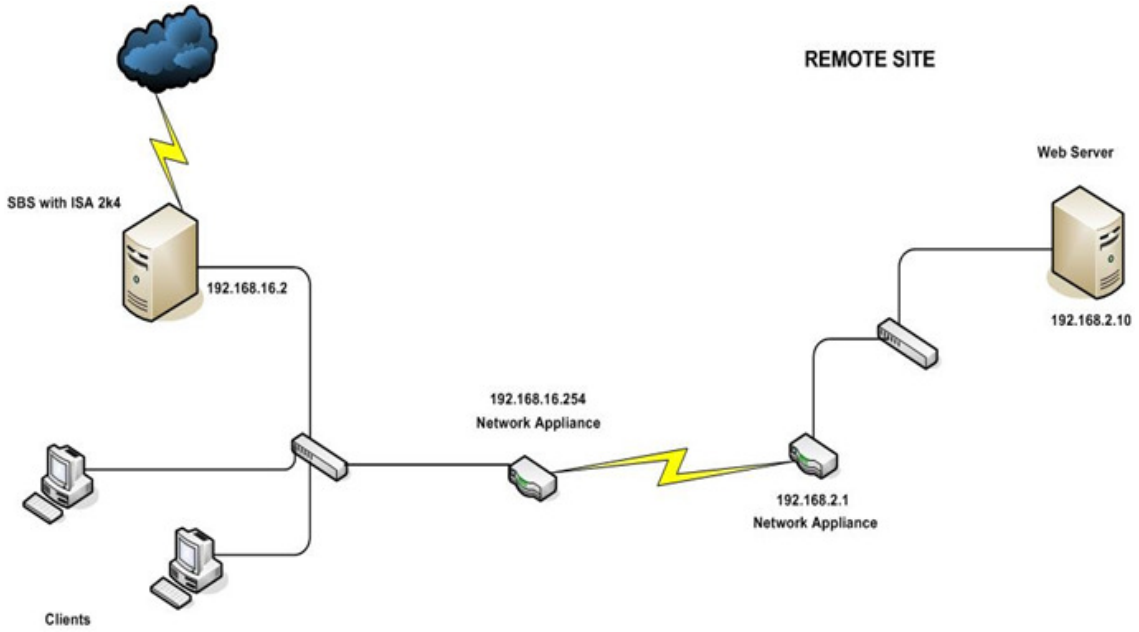

Figure 2.1: Packets drop under DDOS attack [26]

Zombies are called bots and the attack network is called botnet in hacker's community. The zombie machines under control of handlers as shown in Fig 2.1 send attack packets which converge at victim or its network to exhaust either its communication or computational resources. DDOS is basically, a resource overloading problem. The resource can be bandwidth, memory, CPU cycles, file descriptors and buffers etc. The attackers bombard scarce resource either by flood of packets or a single logic packet which can activate a series of processes to exhaust the limited resource. Here in the Fig 2.1 packets drop due to congested access link in victim network and buffer overflow at victim due to large number of requests are depicted.

A denial of service, DOS attack is characterized by an explicit attempt by attackers to prevent legitimate users of a service from using that service. This service can be either a specific destination e.g. a host or the medium that connects the host to other hosts e.g. a network. It is an active threat in the sense that an attacker has to actively create these requests in contrast to e.g. traffic analysis and release of message contents. A distributed denial of service attack often abbreviated DDOS attack is a denial of service attack. Which instead of using one host as the base of attack uses multiple hosts, hence the name distributed. A DDOS or DOS attack differs from an attempt to break into a computer, this is often called a crack or often also falsely referred to as a hack, in several ways some of the most prevalent are:

A DDOS or DOS attack doesn't break into the computer attacked; it simply floods the target with so many requests to the service being attacked that it can't handle the legitimate requests coming from legitimate users. There is no break in done during the actual attack but in the acquiring of a zombies a break in is needed.

A DDOS or DOS attack doesn't itself provide an attacker with information stored on the target system; it can of course be used as a part of an attack chain incidents leading to that global but is in itself not providing such information.

This is a two phase attack, namely

- Preparation phase
- Attack phase

In the preparation phase, an attack network is established by infecting poorly secured computers with a DOS agent, such as a Trojan horse. Highly vulnerable computers are those with out-of-date or non-existent antivirus software. They are termed as Zombies. In the second phase the intermediate computers are used to carry out the attack by instructing them to flood garbage data to the target. Zombies are generally unaware that are carrying on the attack on behalf of the attacker.

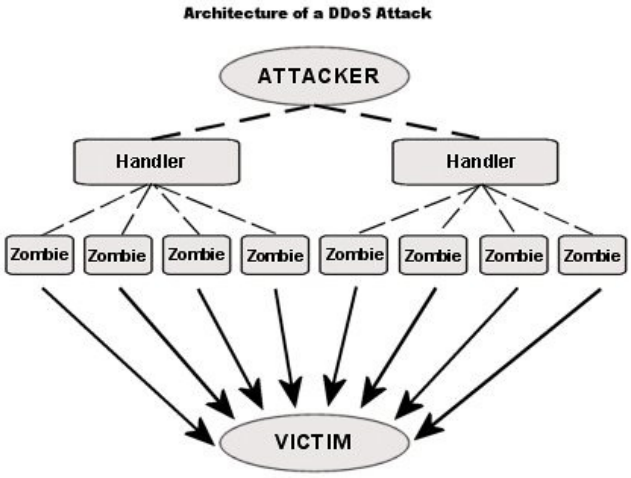

Figure 2.2: DDOS Attack diagram [27]

2.3 Characteristics of Distributed Denial of Service Attacks

A denial of service attack is characterized by an explicit attempt by an attacker to prevent legitimate users of a service from using the desired resources. Examples of denial of service attack include:

- Attempts to "flood" a network, thereby preventing legitimate network traffic.

- Attempts to disrupt connections between two machines, thereby preventing access to a service.
- Attempts to prevent a particular individual from accessing a service.

A distributed denial of service attack is composed of four elements, as shown in Fig 7. First, it involves a victim i.e. the target host that has been chosen to receive the brunt of the attack. Second, it involves the presence of the attack daemon agents. These are agent programs that actually conduct the attack on target victim. The third component of a distributed denial of service attack is the control master program. Its task is to coordinate the attack. Finally, there is the real attacker, the mastermind behind the attack. By using a control master program, the real attacker can stay behind the scenes of the attack. The following steps take place during a distributed attack:

1. The real attacker sends an 'execute" message to the control master program.
2. The Control master program receives the "execute" message and propagates the Command to the attack daemons under its control.
3. Upon receiving the attack command, the attack daemons begin the attack on the Victim.

2.4 Types of Attacks

The ultimate motive of a DDOS attack is to incapacitate a system's ability to process requests for its legitimate users. The various forms of attack are as follows:

2.4.1 Resource Starvation

Its aim is to disrupt of physical network components such as a resource on a particular machine. The resource can be memory or CPU cycles etc. For example:

- Starting some infinite loop.
- Starting some process which consumes the CPU cycles completely.

2.4.2 Bandwidth Consumption

It involves flooding traffic for consumption of computational resources such as bandwidth and disk space. Its primary objective is to block all access to a network thereby preventing any relevant information to flow across the network. For example:

- Websites may go down because an attacker is creating a large amount of requests to an HTTP server.

- A mail server (SMTP/POP etc) can fail if its "mail bombed". This is the act of sending hundreds or thousands of bogus emails in a very short amount of time.

2.4.3 Programming Flaws

There exist some programming flaws which lead to exceptional conditions which cannot be handled by Operating Systems, resulting in failure of application. For example:

- A very long data input
- Unparseable data input

2.4.4 Destruction of Configuration Information

An attacker destroys or alerts configuration information, which prevents the victim from using the network. For example:

- Changing routing tables or DNS catches.

2.5 Various modes of Attack

There are two categories of attacks possible in DOS:

- Direct Attack
- Indirect Attack

2.5.1 Direct Attack

In the Direct attack the attacker attacks the victim directly without hiding his identity.

The various forms of direct attacks are:

2.5.1.1 Land Attack

In this attack IP packets with same source and destination address are used, which causes the machine enters into the loop, thereby consuming CPU cycles.

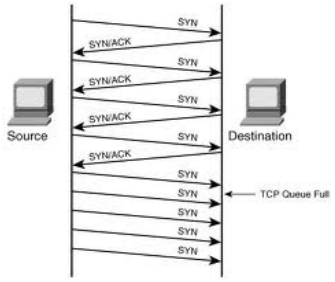

Figure 2.3 Land Attack [28]

2.5.1.2 Teardrop Attack

Teardrop is a program that sends IP fragments to a machine connected to the internet or a network. Teardrop exploits the overlapping IP fragments bug present in Windows 95, Windows NT, and Windows 3.1 machines. The bug causes the TCP/IP fragmentation re-assembly the code to improperly handle overlapping IP fragments. This attack has not shown any significant damage to system and a simple reboot may be a remedy. The Teardrop Attack uses IP's packet fragmentation algorithm to send corrupted packets to the victim machine. This confuses the victim machine and may hang it.

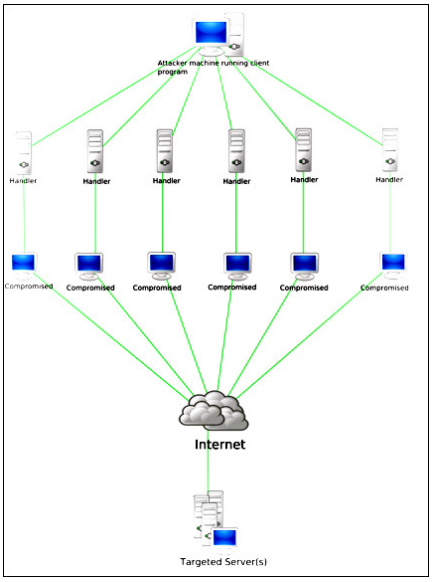

Figure 2.4 Teardrop Attack [29]

2.5.1.3 Ping of Death

This attack causes ICMP ping messages longer than TCP/IP specification of 65,536 octets to crash or freeze the system. This was a major cause of concern in early versions of various operating systems which would react in an unpredictable fashion when receiving these oversized IP packets.

The Ping of Death attacks relies on a bug in the Berkley TCP/IP stack which also existed on most systems which copied the Berkley network code. The ping of death was simply sending ping packets larger than 65,535 bytes to the victim. This denial of service attack was as simple as: ping -1 86600 victim.org

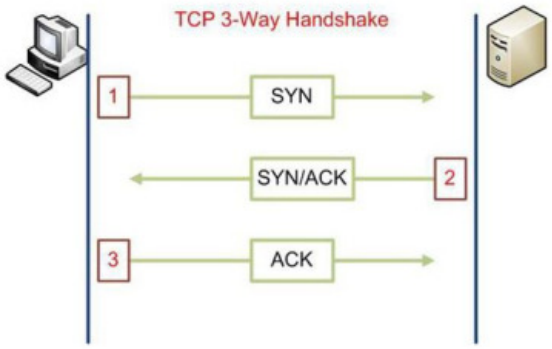

Figure 2.5: Ping of Death Attack [30]

2.5.1.4 Flood Attack

The earliest form of denial of service attack was the flood attack. The attacker simply sends more traffic than the victim could handle. This requires the attacker to have a faster network connection than the victim. This is the lowest technique of the denial of service attacks and also the most difficult to completely prevent.

In a DDOS flood attack the zombies flood the victim system with IP traffic. The large volume of packets sent by the zombies to the victims system slows it down, crashes the system or saturates the network bandwidth. This prevents legitimate users from accessing the victim.

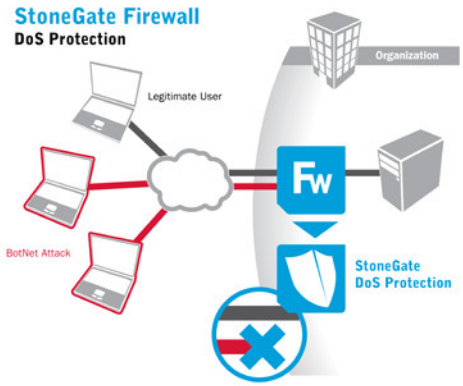

Figure 2.6: Flood Attack [31]

2.5.1.5 Synchronization Flood (SYN Attack)

In this attack the attacker opens multiple half-open connections with the victim. The Victim keeps these connections open waiting for the acknowledgements to complete the connection. Hence the Kernel data structures get used up without opening any real connections. In the TCP protocol, handshaking of network connections is done with SYN and ACK messages. The

system that wishes to communicate sends a SYN message to the target system. The target system that responds with an ACK message. In a SYN attack, the attacker floods the target with SYN messages spoofed to appear to be from unreachable Internet addresses.

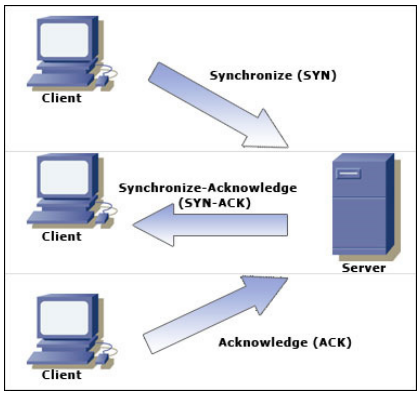

Figure 2.6: Diagram Indicating Synchronization Flood [32]

2.5.1.6 Amplification Attacks

A DDOS amplification attack is aimed at using the broadcast IP address feature found on most routers to amplify and reflect the attack. This feature allows a sending system to specify a broadcast IP address as the destination address rather than a specific address.

For this type of DDOS attack, the attacker can send the broadcast message directly to the attacker which can use the agents to send the broadcast message to increase the volume of attacking traffic. If the attacker decides to send the broadcast message directly, this attack provides the attacker with the ability to use the system within the broadcast network as zombies without needing to infiltrate them or install any agent software.

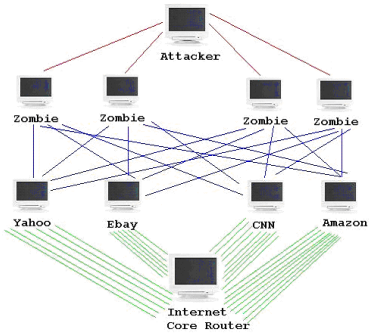

Figure 2.7: Amplification Attack [33]

2.5.2 Indirect Attack

The Indirect attacks are the attacks in which the attacker is able to attack the victim hiding his own identity, either by using intermediary machines or by spoofing packets.

The Attacks are:

- Smurf Attacks
- Distributed DOS

2.5.2.1 Smurf Attack

This attack involves sending ping request with fake source IP address to IP broadcast address. The IP broadcast address may be a gateway to a LAN. This kinds of attacks relies on carelessly configured network devices that allow packets to be sent to all computer host on a particular network via the broadcast address of the network rather than a specific machine. The whole network then serves as a smurf amplifier. For one ping(ICMP echo) request, hundreds of ping replies can be generated which is in turn send to the victim. Hence the victim is overloaded with the ping replies. In the Smurf Attack, the attacker sends a ping request to a broadcast address at a third-party on the network. The ping request is spoofed to appear to come from the victims network.

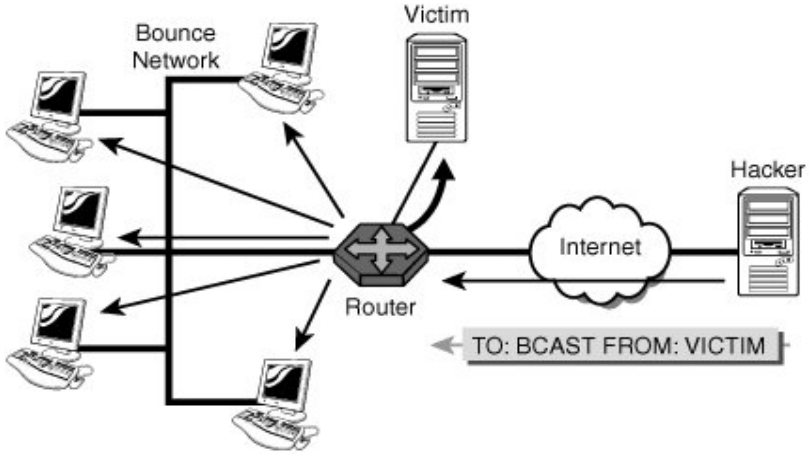

Figure 2.7: Diagram Demonstrating Smurf Attack [34]

2.5.2.2 Distributed DOS

In a Distributed Denial of Service (DDOS) Attack, the attacking computers are zombies with broadband connections that have been infected by viruses or Trojan horse programs that allow

the attacker to control the zombie machine and perform the attack. It is the newest and the most widely used mode of the DOS Attack.

This is a two phase attack that involves:

- Stealthy Preparation Phase
- The Attack Phase

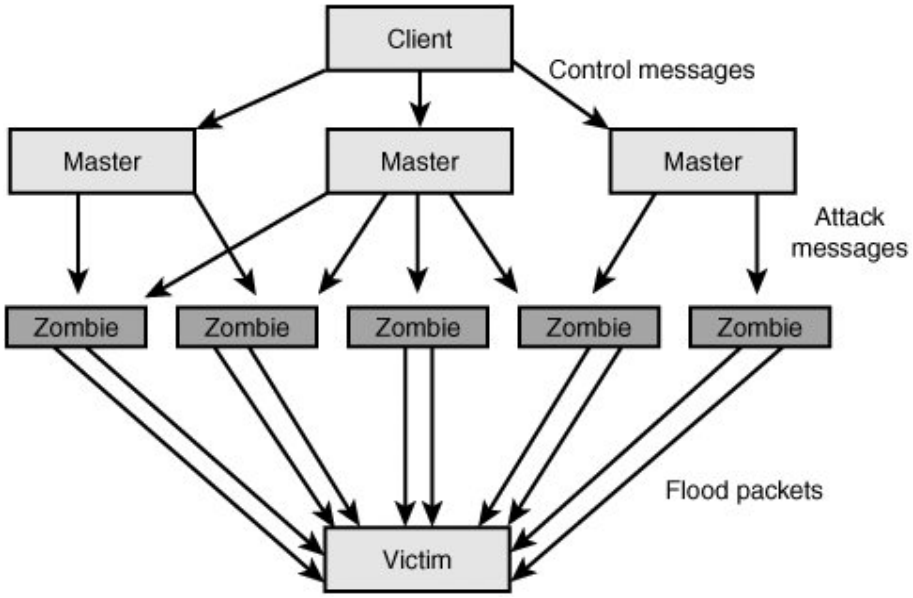

Figure 2.8: Diagram Demonstrating DDOS Attack [28]

CHAPTER 3

DDOS DEFENCE MECHANISMS

3.1 Recruitment of Agent Network

Having understood the varied nature of DDOS attacks it would be useful to be aware of the whole process of a DDOS attack right from the recruitment of an agent network to the different tools used to wage an attack, the scale o an attack and the last but not the least defence mechanism to fend off attackers. Depending on the scale and type of attack, the attacker intends to initiate, he searches for vulnerable hosts (hosts with high network bandwidth and poor administration), manually, semi automatically, in a process known as scanning. Once the vulnerable nodes are identified DOS daemons are installed on these systems. Earlier it was the installation process that was automated but now there are scripts to automate the search for vulnerable hosts and installation of attack code. Examples of automated scanning tools are *scan, bots and worms.*

The various stages of the DDOS attack model are:

1. Attacker
2. Communication Channel
3. Attacking Host
4. Victim

Thousands of people connect to an IRC server and form the thousands of different chat channels. It is impossible to identify the attack channel in such a huge number of channels and therefore the identity of the attacker is left unknown. In the Agent Handler, Figure 13, the communication channel consists of one or more hosts that are attacker has compromised. The attacker issues commands to handler which in turn dispatches commands to the agent to launch an attack against the victim.

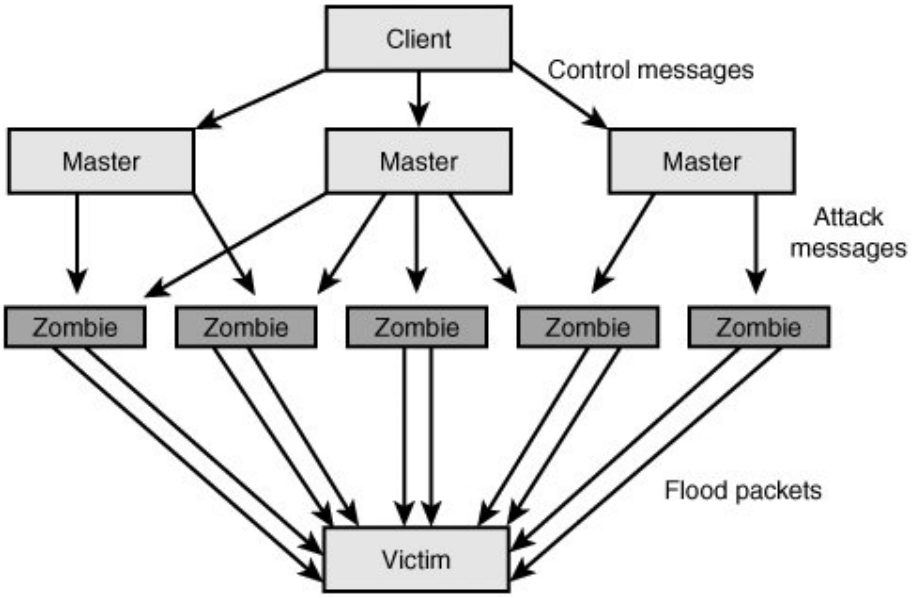

Figure 3.1: A Handler/Agent Control Structure [26]

3.2 DDOS Agent Setup

These are both active and passive methods that attackers use to install malicious code onto a secondary victim system in order to set up an Agent-handler based DDOS attack network. Active methods typically involve the attacker scanning the network or systems with known vulnerabilities. Upon identifying such vulnerable systems, he can stealthily install the DDOS Agent Software. Thus the system is compromised as a secondary victim and can be used as a zombie in a DDOS attack. Passive methods typically involve the attacker sharing corrupt files or building web sites that take advantage o known vulnerabilities in a secondary victim's web browser. Upon accessing a file or website with an embedded DDOS Agent, the secondary victim system is compromised and the DDOS agent code may be installed.

3.3 Defence Principles and Challenges

As DDOS is a distributed attack and because of high volume and rate of attack packets distributed instead of centralized defence. Second, High Normal Packet Survival Ratio(NPSR) hence less collateral damage is the prime requirement for a DDOS defence. Third, a DDOS defence method should provide secure communication for control messages in terms of

confidentiality, authentication of sources, integrity and freshness of exchanged messages between defence nodes. Fourth, a partially and incrementally deployable defence model is successful as there is centralized control for autonomous systems (AS) in Internet. Fifth, a defence system must take into account future compatibility issues such as interfacing with other systems and negotiating different defence policies. However with the present technology, development and implementation of a DDOS defence model which can satisfy all of these defence principles in general is very difficult in practice due to several challenges such as:

a) Large number of unwitting participants
b) No common characteristics of DDOS streams
c) Use of legitimate traffic models by attackers
d) No administrative domain cooperation
e) Automated tools
f) Hidden identity of participants
g) Persistent security holes on the Internet
h) Lack of attack information

3.4 Defence against Attacks

Many observers have stated that there are currently no successful defence against a fully distributed denial of service attack. This may be true. Nevertheless, there are numerous safety measures that a host or network can perform to make the network and neighbouring networks more secure. These measures include:

3.4.1 Filtering Routers

Filtering all packets entering and leaving the network protects the network from attacks conducted from neighbouring networks and prevents the network itself from being an unaware attacker. This measure requires installing ingress and egress packet filters on all routers.

3.4.2 Disabling IP Broadcasts

By disabling IP broadcasts, host computers can no longer be used as amplifiers in ICMP flood and smurf attacks. However, to defend against this attack, all neighbouring networks need to disable IP broadcasts.

3.4.3 Disabling Unused Services

If UDP echo or charges services are not required, disabling them will help to defend against the attack. In general, if network services are unneeded or unused, the services should be disabled to prevent tampering and attacks.

3.5 An Introduction to DOS Countermeasures

A comprehensive countermeasure for DOS attacks has four distinct elements:

Prevention, detection, mitigation and trace back. Before an attack occurs, there should be existing prevention mechanisms that are capable of eliminating the threat of the attack. When the attack does occur , only a successful and timely detection of attack will allow the appropriate mitigation mechanism to be deployed. During or following the attack, a method called tree back can be used to determine the source of the attack. In addition, trace back can help improve future methods for detecting and preventing a DOS attack. The dependence upon all four of these items is crucial for a successful DDOS countermeasures.

In order to design the most effective countermeasures against DOS attacks, we must address the security vulnerabilities at each layer and introduce defence mechanisms that are capable of mitigating specific threats at the appropriate layer. The only way to defend a computer from attacks is to design and employ a number of protection mechanism that are designs to combat a specific threat.

3.5.1 Prevention

Prevention is the first step towards defending a machine from DOS/DDOS attack. The key to prevention is awareness and vigilance. System patches and updates are very important to protection but they only fix known problems. Thus systems are always vulnerable to unknown attacks and undiscovered weakness. Therefore, it is necessary for many to understand how attacks occur and to always remain aware and watching for new potential threats.

3.5.2 Detection

One of the most important components in designing a DOS countermeasure is to determine and establish the optimal methodology to detect an ongoing attack. Most detection mechanism rely on some form of an application or software that resides on a host system or within the network that can observe traffic patterns or resource usage. These programs typically are configured to detect anomalies or deviations from normal behaviour. When anomalies are detected, alerts are created so that either a system administrator or an automated program can quickly determine

the type of the attack and decide which actions to take to safely minimize the effects of the attack and return the system back to its original state. Most attacks are detected by an end host or server. However, in many cases a DDOS can affect the routers within the network.

3.5.3 Mitigation

Mitigation is the process of minimizing the effects of the ongoing attacks. The simplest and easiest form of mitigation is to simplify drop packets that belong to an attacker and allow packets that belong to a client to reach their destination. However, the main obstacle with this method is how to accurately determine if a packet belongs to an attacker or to a legitimate client. Attackers have shown great sophistication in their ability to disguise themselves as clients, making it virtually impossible to determine if a packet belongs to a legitimate client or an attacker. In this report, the mitigation techniques that are researched do not necessarily deal with making this distinction. Instead, the techniques presented in this report allow each client to prove itself as being legitimate before being granted access.

3.5.4 Trace Back

The process of trace back involves the methods used to determine the source of the attack. This technique is commonly referred to as IP trace back. In many previous DOS/DDOS attacks, the attackers have demonstrated the ability to spoof the identity of the attack packets by selecting a different source IP address than the IP address of the computer that is actually sending the packet. This technique is referred to as IP spoofing. Before a router forwards a packet, it verifies that the source IP address of the packet is valid.

Common trace back methods involve packet marking, a technique where routers place a unique mark within the header of each packet that it forwards. When a packet traverses all the way to server, each router will have already marked the packet in such a way that the combination of all of the marks creates a unique signature that is measured to identify the client. Thus, when the end host receives a packet, the total mark will be used to differentiate between a client and an attacker.

3.6 Mitigate or Stop the Effects of DDOS Attack

3.6.1 Load Balancing

For network providers there are a number of techniques used to mitigate the effects of a DDOS attack. Providers can increase bandwidth on critical connections to prevent them from going down in the event of an attack. Replicating servers can help provide additional failsafe protection in the event some go down during a DDOS attack. Balancing the load to each server in multiple-server architecture can improve both normal performance as well as mitigate the effect of a DDOS attack.

3.6.2 Throttling

One proposed method to prevent servers from going down is to use Max-min Fair server-centric router throttles. This method sets up routers that access a server with logic to adjust (throttle) incoming traffic to levels that will be safe for the server to process. This will prevent flood damage to servers. Additionally, this method can be extended to throttle DDOS attacking traffic versus legitimate user traffic for better results. This method is still in experimental stage; however similar techniques to throttling are being implemented by network operators. The difficulty with implementing throttling is that it is still harder to decipher legitimate traffic from malicious traffic. In the process of throttling, legitimate traffic may sometimes be dropped or delayed and malicious traffic may be allowed to pass to the servers.

3.6.3 Drop Requests

Another method is to simply drop request when the load increases. This can be done by the router or the server. Alternatively, the requester may be induces to drop the request by making the requester system solve a hard puzzle that takes a lot of computer power or memory space before continuing with request. This causes the users of zombie systems to detect performance degradation and could possibly stop their participation in sending DDOS attack traffic.

CHAPTER 4

LITERATURE REVIEW

4.1 Literature Review

Abraham Yaar [1] presented a new packet marking approach i.e. Pi (short for Path identifier) in which path fingerprint is embedded in each packet which enables a victim to identify packets traversing same paths. In this scheme each packet traversing the same path carries the same identifier. Path identifier fits in each single packet so the victim can immediately filter traffic after receiving just one attack packet .Xiuli Wang [16] proposed Pushback to mitigate DDoS attacks. It is based on improved Aggregate based congestion control (IACC) algorithm and is applied to routers to defend against bandwidth consumption attacks. In this scheme we first match the attack signature of the packet, if it is matched packet is sent to the rate limiter which will decide whether to drop the packet or not. From the rate limiter the packet is sent to the Pushback daemon which will drop these packets with the help of upstream routers.

Ruiliang Chen and Jung- Min Park [12] combined the packet marking and pushback concepts to present a new scheme called as Attack Diagnosis. In this scheme an Intrusion Detection System is installed at the victim that detects the attack. The victim instructs the upstream routers to start marking packets with trace back information based on which victim reconstructs the attack paths and finally upstream routers filter the attack packets.

Antonis Michalas, Nikos Komninos, Neeli R. Prasad and Vladimir A. Oleshchuk [2] presented Client puzzle approach to prevent DDoS attacks in ad hoc network. In this approach every node that is trying to contact another node has to solve two puzzles. The first one is a discrete logarithm problem (DLP) and the solution of this puzzle will help the connection initiator to create and solve the second puzzle which is considered to be the most difficult.

Biswa ranjan Swain and Bibhudatta Sahoo [4] presented Probabilistic Approach and HCF method to mitigate these attacks. In this approach the researchers have developed a probabilistic approach to find out the number of malicious packets arriving at the server. After calculating the Probability of the packets being malicious we filter the packets from the given no. of packets. They proposed a formula to calculate the Probability.

Yinan Jing, Xueping Wang Xiao and Gendu Zhang [17] presented IP Traceback based Rate Limiting approach to mitigate DDOS attacks. Max-min fairness algorithm used by previous

researchers it is found that it punishes both attackers and legitimate users equally under a meek attack. The survival ratio(i.e the percentage of legitimate packets received by the victim in all legitimate packets) of IP Trace back is very high reaching over 90%, therefore, this algorithm not only regulates the traffic to ensure the victim's load but also improves the survival ratio of legitimate packets.

4.2 Related Works

Countermeasures to bandwidth-exhaustion attacks

It is a mechanism that enables the system to counter DOS yielded bandwidth-exhaustion attacks through the concepts of aggregate-based congestion control; trace back and filtering. Another mechanism being the Aggregate-based congestion control (ACC) that extends its scope to traditional flow based so that data packets could be managed to finer granules. An aggregate is defined as a collection of packets that share some property (signature). ACC provides mechanisms for detecting and controlling aggregates at a router using an attack signature, a mechanism to propagate aggregate control requests (and the attack signature) to upstream routers. ACC critically depends on the mechanism by which attacks are detected and an attack signature is formulated and this can be a source of difficulty against an intelligent adversary that varies its traffic characteristics over time. A goal of Congestion Puzzles (CP) is to avoid the need to formulate attack signatures.

A related congestion control mechanism is level-k max-min fair router throttles. The mechanism differs from ACC in that a congested server is responsible for issuing congestion control requests to routers k hops away (denote the set of these routers by R(k) to help maximize the bandwidth allocated to those receiving the smallest allocation (max-min optimization). This approach does not depend on formulating attack signatures, but offers fairness only to the extent that the routers in R(k) can provide it. With a low deployment depth (small k), it is possible that legitimate clients' flows may aggregate to a relatively high bandwidth flow before reaching a router in R(k), thus being subjected to rate limiting. Another limitation of the mechanism is the assumption that all routers are trusted, which makes it vulnerable to attacks from compromised routers. Several methods focus primarily on filtering or tracing spoofed traffic, such as ingress filtering, hop-count filtering, and numerous works on trace back. These approaches are of less utility against non-spoofed traffic and thus permit DDOS attacks from zombies using their real source addresses. In addition, many of these schemes rely upon some way of distinguishing attack packets from legitimate ones, thereby again raising the difficulties of generating attack signatures. Finally, some filtering schemes

consider coordination among routers. Our approach also supports such coordination within the context of the CP mechanism. Recently, an approach that uses overlay network is compromised of a set of nodes across the internet. A client who wants to connect to the web server has to first pass a reverse Turing test posed by an overlay node, which then tunnels the clients' connection to an approval location so as to reach the web server. This approach, however does not solve the general bandwidth-exhaustation problem: First, adversaries might still be able to use other protocols or the traffic addressed to a less sensitive server to congest routers on paths to the web server. Second, this solution is tailored to protocols driven by human users, who can be called upon to pass a reverse Turing test. Third, once adversaries have implanted zombies at overlay nodes or routers, they might circumvent the defence mechanism.

4.3 Proposed Approach

Our goal in this project is to protect a server system from having to deal with excessive service request arrivals over a global network. (However the approach can be easily generalized to protecting an intermediate routing point under overload). To do so, we adopt a *proactive* approach: Before aggressive packets can converge to overwhelm a server, we ask routers along forwarding paths to regulate the contributing packet rates to more moderate levels thus forestalling an impending attack. The basic mechanism is for a server under stress, say S, to install a router throttle at an upstream router several hops away. The throttle limits the rate at which packets destined for S will be forwarded by the router. Traffic that exceeds the rate limit can either be dropped or rerouted to an alternate server, although we will focus exclusively on dropping solution in this project.

A key element in the proposed defence system is to install appropriate throttling rates at the distributed routing points, such that, globally, S exports its full service capacity Us to the network, but no more. The "appropriate" throttles should depend on the current demand distributions and so must be negotiated dynamically between server and network. Our negotiation approach is server-initiated. A server operating below the designed load limit needs no protection and need not install any router throttles. As server load increases and crosses the designed load limit, Us however, the server may start to protect itself by installing and activating a rate throttle at a subset of its upstream routers. After that, if the current throttle fails to bring down the load at s to below Us, then the throttle rate is reduced. On the other hand, if the server load falls below a low-water mark Ls (where Ls<Us), then the throttle rate is increased (i.e., relaxed). If an increase does not cause the load to significantly increase over

some observation period, then the throttle is removed. The goal of the control algorithm is to keep the server load within [Ls, Us] whenever a throttle is in effect.

Although throttling is space-efficient, the total amount of state information needed at a router is nevertheless linear in the number of installed throttles. Hence, it may not be possible for the routers to maintain state about every Internet server. However, the approach can be feasible as an on-demand and selective protection mechanism. The premise is that DDOS attacks are the exception rather than the norm. At any given time, we expect at most only a minor portion of the network to be under attack, while the majority remaining portion to be operating in "good health". Moreover, rouge attackers usually target "premium sites" with heavy customer utilization, presumably to cause maximal user disruption and to generate the most publicity. These selected sites may then elect t protect themselves in the proposed architecture, possibly by paying for the offered services.

Work in this report targets a network architecture for countering denial-of-service (DOS) attacks directed at an Internet server. The basic mechanism is for a server under stress to install a router throttle at selected upstream routers. The throttle mechanism would be highly effective in preferentially dropping attacker traffic over good user traffic. Also, this throttling can regulate the experienced server load to below its design limit in the presence of user dynamics so that the server can remain operational during a DOS attack. In our study, we are supposed to simulate a distributed denial of service attack using ns-2 network simulator. We examine how various queuing algorithm implemented in a network router perform during an attack and whether legitimate users can obtain desired bandwidth. We find that under persistent denial of service attacks, class based queuing algorithms can guarantee bandwidth for certain classes of input flows.

4.5 The contributions

The contributions in this Report are:

- Contributed to the fundamental understanding of router throttling as a mechanism against DDOS attacks. In particular, we advance a control-theoretic model useful for understanding system behaviour under a variety of parameters and operating conditions:-
- Presented a throttle algorithm that can effectively protect a server from resource overload and increase the ability of good user traffic to arrive at the intended server.

- Showed how max-min fairness can be achieved across a potentially large number of flows and the implication of a notion of convergence on DDOS attacks.
- Studied how throttling may impact real application performance. Specifically, demonstrated via simulations the performance impact on an HTTP web server.

Convention 1: All traffic rate and server load quantities stated in this paper are in unit of kb/s, unless otherwise stated.

Modelled a network as a connected graph *G= (V, E)*, where *V* is the set of nodes and *E* is the set of edges. All leaf nodes are hosts and thus can be a traffic source. An internal node is a router, a router cannot generate traffic but can forward traffic received from its connected hosts or peer routers. We denote by R the set of internal routing nodes. All routers are assumed to be trusted. The set of hosts, *H=V-R*, is partitioned into the set of ordinary "good" users, *Hg*, and the set of attackers i, E models the network links, which are assumed to be bi-directional. Since our goal is to investigate control against *server* resource overload, each link is assumed to have infinite bandwidth. The assumption can be relaxed if the control algorithm is also deployed to protect routers from overload.

In this report, we designate a leaf node V as the target Server S. A good user sends packets to S at some rate chosen from the range [*0,rg*]. An attacker sends packets to S at some rate chosen from the range *[0,ra]*. In principle, while rg can actually be set to a reasonable level according to how users normally access the service at S (and we assume *rg <<Us*), it is hard to prescribe constraints on the choice of ra. In practice, it is reasonable to assume that *ra* is significantly higher than rg. This is because if every attacker sends at a rate comparable to a good user then an attacker must recruit or compromise a large number of hosts to launch an attack with sufficient traffic volume.

When S is under attack, it initiates the throttle defense mechanism (for ease of presentation, we assume that an overloaded server is still capable of initiating the defense actions) The throttle does not have to deployed at every router in the network. Instead, the deployment points are parameterized by a positive integer k and are given by R(k). Specifically, R(k) contains all the routers that are either k hops away from S or less than k hops away from S but are directly connected to a host. Fig4.1 shows an example network

topology. In the figure 4.1, a square node represents a host, while a round node represents a router. The host on the far left is the target server S.

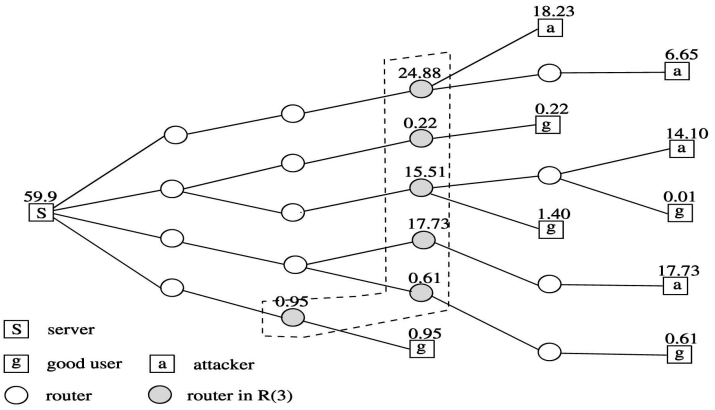

Figure 4.1: Network Topology illustrating R(3) deployment points of routers throttle, and offered traffic rates [35]

The routers in R(3) are shaded in the figure. Notice that the bottom-most router in R(3) is only two hops away from S, but is included because it is directly connected to a host. Given our system model, an important research problem is how to achieve *fair* rate allocation of the server capacity among the routers in R(k). To that end, we define the following notion of convergence.

4.6 Proposed Algorithm

The following algorithm has been proposed to mitigate DDOS Attacks

Throttle Algorithm

The report here presents an algorithm in which each router throttles traffic for S by forwarding only a fraction f(0 <=f<=1) of the traffic. The fraction f is taken to be one when no throttle for S is in effect. In adjusting f according to current server congestion, the algorithm mimics congestion control. Specifically, f is reduced by a multiplicative factor α when S is congested and sends the router a rate reduction signal. It is increased by an additive constant β- subject to the condition that f<1 – when S has extra capacity and sends te router a rate increase signal.

The algorithm that S runs is specified in Fig 15.It is to be invoked when either (i) the current server load (measured as traffic arrival rate to S) crosses Us or (ii) a throttle is in effect and the current server load drops below Ls. In case (i) S multicasts a rate reduction signal to R (k), in case (ii) it multicasts a rate increase signal. The algorithm can take multiple rounds until a server load within [Ls, Us] is achieved. Also, if the server load is below Ls and the next rate

increase signal raises the server load by an insignificant amount (i.e by less than □), we remove

the throttle. The monitoring window should be set to be somewhat larger than the maximum round trip time between S and a router in R (k).

In the throttle based algorithm we install at each router in R(k) a uniform leaky bucket rate (i.e the throttle rate) at which the router can forward traffic for S. Fig. 15 specifies the algorithm by which S determines the throttle rate to be installed. In the specification, rs is the current throttle rate to be used by S. It s initialized to (Ls, Us) / *f* (k), where *f* (k) is either some small constant, say 2, or an estimate of the number of

Algorithm

```
ρ last :  = -∞;

Hv     :  =  Us;

Lv     :  =  0;

Rs     :   = (Ls + Us)/f(k) ; /*initialize throttle rate*/

While(1)

If (ρ >= Us)

 Hv  : =  Rs;

If (ρ last - ρ < €)

  Lv  := 0;

.     end if

   else if (  ρ <= Ls)

          Lv  := Rs ;

           If (ρ  - ρ last < €)

               Hv  : =  Us;

            end if;

end if;

 ρ last :  = ρ;

Rs  :  = (Hv + Lv)/2);
        end while;
```

Iteration	Rs	ρ
1.	11	34.78
2.	5.5	18.38
3.	14.7	45.88
4.	7.3	24.68
5.	6.4	20.98

Table 1: TRACE OF THROTTLE AND ACHIEVED SERVER LOAD FOR THE ALGORITHM

throttle points typically needed in R(k).

The throttle algorithm when called multicast a rate rs throttle to R (k). This will cause a router in R (k) to regulate traffic destined for S to a leaky bucket with rate rs. The algorithm may then continue in the while loop that it iteratively adjusts rs to an appropriate value.

Notice that the iterative process aims to keep the server load in [Ls,Us] whenever a throttle is in effect.

In the example network shown in Fig 4.1, let the number above each host(except S) denote the current rate at which the host sends traffic to S.
The number above each router denotes the offered rate of traffic at the router, destined for S. Also, let Ls= 20,Us=24. Initially, the total offered load to S exceeds Us, and hence the algorithm is invoked at S.

We apply the algorithm to the given example scenario in Fig.4.1. We initialize rs to (Ls + Us)/2 = 11. Table 1 shows how rs and the aggregate server load evolve. When the algorithm is first invoked with throttle rate 11, the aggregate load at S drops to 34.78. Since the server still exceeds Us, the throttle rate is made 5.5 and the server load drops below Ls to 18.38. As a result the throttle rate is increased to 14.7 and the server load becomes 45.88. Since the server still exceeds Us, the throttle rate is decreased to 7.3 and the server load becomes 24.68. Since this still exceeds Us, the throttle rate is further decreased to 6.4 and the server load becomes 20.98. Since, 20.98 is within the target range [20,24], the throttle algorithm terminates. When that happens, the forwarding rates of traffic for S at the deployment routers (from top to bottom in the figure) are 6.4, 0.22, 6.4, 6.4, 0.61 and 0.95 respectively.

4.7 Specific Knowledge Required

This project demands the working knowledge of the following.

4.7.1 Tool Command Language

Tool Command Language (TCL) is regarded as the “best-kept secret in the software industry”. It has many uses, namely, Web and Desktop Applications, Network Programming, Embedded Development, Testing, General Purpose Programming, System Administration and for Databases The advantage of using this language is flexible integration, elegant networking model and thread support.

4.7.2 Using Network Simulator

We have used the Network Simulator version 2.29 for the simulation of the various attacks and the counter mechanisms. The operating system used is Linux fedora Core 4. TCL version 8.0.5 release 1 was used as the scripting language. The post-analysis bandwidth consumption graph was drawn using X-Graph version 12.1. Some other simulator like *‘Opnet’* is also used for simulation of a network. But Network Simulator along with the Linux operating system holds the pride of being capable enough to support almost all the protocols available that are used in the wide network of computers.

4.7.3 X-Graph

X-Graph is a graph plotting application that is included within the Network Simulator. It supports drawing trace graphs of wired, wireless, satellite, new trace and networks. Its features includes 238 2D graph, 12 3D graph, delay, jitter, processing time, round trip time, throughput graph and statistics.

CHAPTER 5

OBSERVATIONS AND RESULTS

5.1 Implementation

The work of the topic" A Throttle Based Approach to mitigate the Distributed Denial of Service Attacks" has been implemented in Network Simulator (NS2) Version 2.29 in Linux operating System.

There are various nodes which have been created with the help of network simulator then actual flow of data packets among the nodes which shows the actual traffic rates between the server and end users.

5.2 Simulations

Performance evaluation is a critical component of systems research that allows the evaluation of new ideas, identification of problems and bottlenecks and optimization of existing systems. There are three general approaches to performance evaluation: (1) prototyping: build it (or a scaled down version of it) and see how it works; (2) analytical modeling: build a mathematical model of it and use it to analyze the system; and (3) Simulation: build a software model of the system. Prototyping is often not feasible, or time consuming especially for large scale systems; it also provides limited controllability and observability. Similarly, analytical modeling cannot capture highly complex systems. Thus, simulation has emerged as an attractive alternative that is heavily used in performance evaluation of computer systems. Network simulation enables us to predict behavior of a large-scale and complex network system such as the Internet at low cost under different configurations of interest and over long periods. Many network simulators, such as NS-2, SSFNet, Opnet, Qualnet, etc., are widely available. We will use NS-2 for this project. NS-2 is a discrete event simulator written in C++, with an OTcl interpreter shell as the user interface that allows the input model files (Tcl scripts) to be executed. Most network elements in NS-2 are developed as classes, in object-oriented fashion. The simulator supports a class hierarchy in C++, and a very similar class hierarchy in OTcl. The root of this class hierarchy is the TclObject in OTcl. Users create new simulator objects through the OTcl interpreter, and then these objects are mirrored by corresponding objects in the class hierarchy in C++. NS2 provides substantial support for simulation of TCP, routing algorithms, queuing algorithms, and multicast protocols over wired and wireless (local and satellite) networks, etc. It is freely distributed, and all source code is available.

5.2.1 Attack Simulation

In Distributed Denial of Service Attack, the whole attack revolves around four main Parties

- Vicious Attacker
- Intended Attacker
- Legitimate User
- Innocent Zombies

The flooding technique used for 'bandwidth consumption' at the intended victim is buffer overflow.

5.3 System Performance

Aggregate Server Load Vs Time

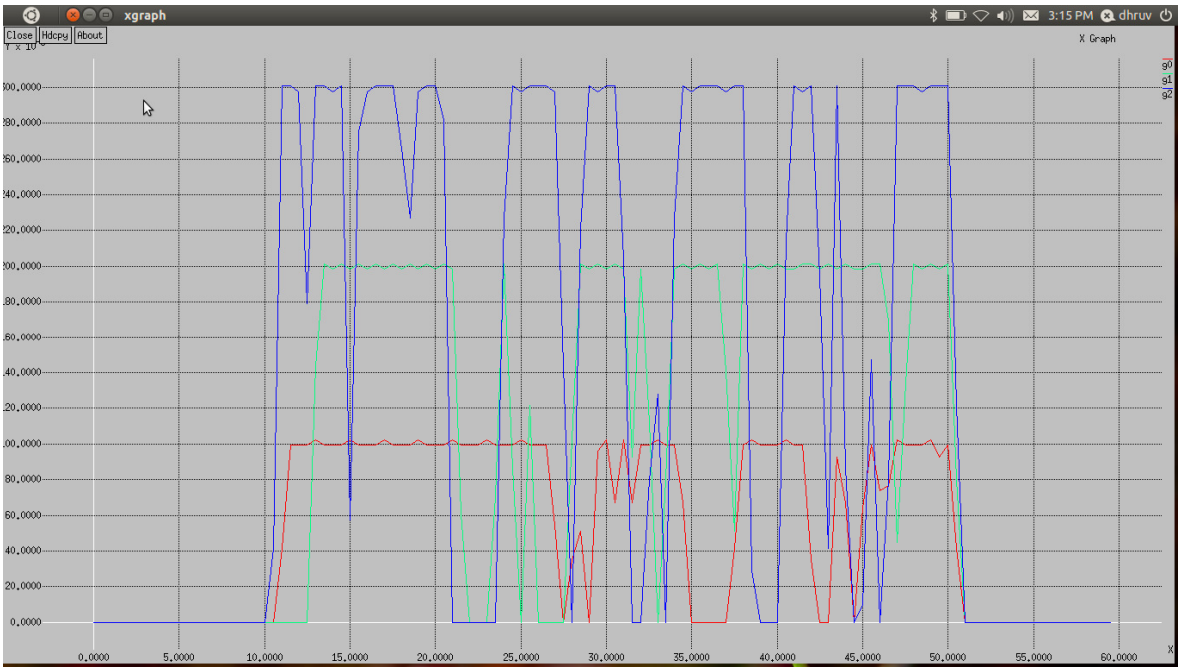

Figure 5.1 : System Performance for Us= 24 , Ls = 20

Here Co-ordinates are

X- Axis--------> Time

Y Axis--------> Aggregate Server Load

Here X axis represents time in Seconds and y axis represents aggregated traffic rate in Kb. We have analyzed system performance for various watermark values. Blue curve represents system performance. Here upper traffic load is defined as Us = 24Kb and lower load Ls = 20kb. Here system is not more stable and lie between upper limit and lower limits. As traffic increases beyond the maximum capacity of router, throttle algorithm is applied on the routers as described and traffic will be decreased for that routers.

For green colour graph system is more stable than previous one because after applying throttle algorithm the rate of flow of packets will be constant.

Throughput Vs Time

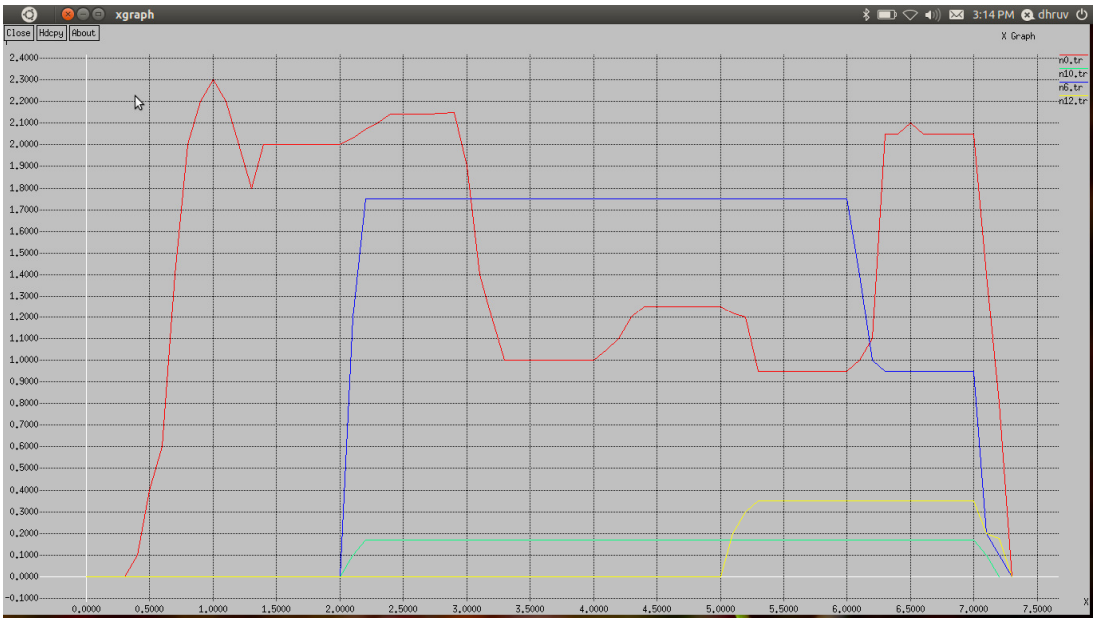

Figure 5.2 : System Performance between throughput and Time

Here Co-ordinates are

X- Axis-----→ Time

Y- Axis------→Throughput

Here X axis represents time in Seconds and y axis represents throughput. We have analyzed system performance. Red curve, Blue curve, yellow curve and green curve represents various system states at different time intervals.

The Distributed Denial of Service (DDOS) attack is a two phase attack, which involves: first, infecting Zombie computer and then, causing them to flood the target. Hence the original attacker is able to mark his identity. Due to the two phases of attack the network has complex topology and the network congestion is high. Hence a countering mechanism which involves addition of extra packets in the network for trace back is unsuitable. Packet marking therefore can be used as a trace back mechanism, as it involves simple marking of packets instead of creating new packets. A mark is a random number added by the router between the header and payload of the IP packet. Each router on marking a packet sends a random number to the Network Manager.

5.5 Observations

Several observations are in order about the practical deployment of our defence mechanism. First, we must achieve reliability in installing router throttles otherwise, the throttle mechanism can itself be a point of attack. To ensure reliability, throttle messages must be authenticated before an edge router (assumed to be trusted) admits them into the network. Notice that only edge routers that have made arrangement with the (relatively few) server sites that desire protection have to do authentication. Also, throttle requests must be efficiently and reliably delivered from source to destination, which can be achieved by high network priority for throttle messages and retransmissions in case of loss. Since throttle messages are infrequent and low in volume, the cost of authentication and priority transmissions should be acceptable.

Second, because of the feedback nature of the control strategy, it is possible that the server will transiently experience resource overload. To ensure that the throttle mechanism remains operational during these times, we can either use a coprocessor on the server machine that is not concerned with receive-sie network processing, or deploy a helper machine, whose job is to periodically ping the server and initiate defence actions when the server is not responsive.

Third, the throttle mechanism may not be universally supported in a network. Our solution remains applicable provided at least one router supports the mechanism on a network path that

sees substantial attacker traffic. Depending on the position of such a router, the feasible range of *k* may be more restricted.

Fourth, we have adopted a generic notion of max-min optimization in our study, which makes it easy to manage and deploy.

5.6 Summary

DDOS is a complex problem involving hosts distributed all over the Internet and affecting numerous networks. While localized defences alleviate damage from small-scale, easily charecterizable attacks, more sophisticated threats can only be handled through a cooperative, Internet-wide defence. With the right direction of attack dissection strategy discussed which is Detection, Segregation and mitigation has made it easy for the researchers move forward in three directions. Various IDS based systems has been studied which are geographically distributed has been proposed. Secure Overlay Networks has been proposed as part of Active networks where the trace back is mitigated to other nodes in the overlay network. The main phase is detection and readiness after an attack is detected forms the complete security chain.

CONCLUSION(S)

Conclusions

A number of conclusions can be drawn from understanding DDOS attacks and from looking at some of the defensive measures that are being researched or implemented. DDOS attacks are quiet advanced methods of attacking a network system to make it unusable to legitimate network users. These attacks are an annoyance at a minimum and if they are against a critical system, they can be severely damaging. Loss of network resources costs money, delays work and cuts off communication between network users. The negative effects of a DDOS attack make it important that solutions and security measures be developed to prevent these types of attacks.

The project presents a server-centric approach to protect a server system under DDOS attacks. The approach limits the rate at which an upstream router can forward packets to the server, so that the server exposes no more than its designed capacity to the global network. In allocating the server capacity among the upstream routers, we studied a notion of level –k max – min fairness, which is policy free and hence easy to deploy and manage.

In addition, algorithm effectiveness was evaluated using a realistic global network topology and various models for attacker and good user distributions and behaviours. Our results indicate that the proposed approach can offer significant relief to the server that is being flooded with malicious attacker traffic. First, for aggressive attackers, the throttle mechanism can preferentially drop attacker traffic over good user traffic, so that a larger fraction of good user traffic can make it to the server as compared with no network protection. In particular, level-k max-min fairness performs better than recursive pushback of max-min fair rate limits previously proposed in the literature. This is especially the case when attackers are evenly distributed over the network. Second, for both aggressive and “meek” attackers, throttling can regulate the experienced server load to below its design limit, so that the server can remain operational during a DDOS attack. Moreover, our related implementation results show that throttling has low computation and memory overheads at a deployment router. Our results indicate that server-centric router throttling is a promising approach to countering DDOS attacks. Our focus has been on DDOS attacks in which attackers try to overwhelm a victim server by directing an excessive volume of traffic to the server. Other forms of attacks are possible that do not depend on the sheer volume of attack traffic. However, more sophisticated attack analysis (e.g. intrusion detection) is usually feasible to deal with these other forms of attacks.

Limitations and Future Scope

My contribution in the work is two-fold. First, I have identified the problem of proving DDOS, discussed a potential usage scenario where it could be important and applicable and proposed a solution to the problem. Second, I have proposed a DDOS attack identification algorithm that estimates the DDOS attacks magnitude and with very high accuracy even in the presence of more than 2000 attackers and 60% legacy routers in the network. As future work on this, I would like to relax some restrictive assumptions that I have made and also try optimizing some algorithm. Here is the summary of future work items, along with the approach I propose to adopt:

- Relax the assumption on upstream router map.
- Optimize the packet verification algorithm.
- Relax the assumption on a unique path from PPM routers to victim. Add a few bits of bath identification information in packet marking.
- Design a practical key management scheme between the validator and PPM routers. Explore using the time-released key chain approach.

APPENDIX

APPENDIX1: COMPARISON OF VARIOUS MITIGATION TECHNIQUES

TABLE A: COMPARISON OF MITIGATION TECHNIQUES

S. No	Technique	Formula Used	Description
1.	HCI-MPR	$P(m, l) = e^{-\lambda_E}(p^m\lambda^m)/m!.e^{-\lambda_L(1-p)}(1-p)^l\lambda^l/l!$	This is based on mathematical equations to find the malicious packets and proposes an inspection algorithm to mitigate DDOS attack.
2.	Client Puzzle	$\Pi(n) = \frac{1}{2}.[\frac{n}{2}].n+\frac{1}{8}.(-1)^{1+n}.\frac{3}{4}.[\frac{n}{2}]$	In this method cryptographic puzzles are generated that a client must answer correctly before it is given services, which pushback the load to the source of an attack in case of overload.
3.	Egress Filtering	$e_f^{(t+1)} = \left(q_f^{\wedge(t+1)} - S_{max}(r_f^{\wedge(t+1)}) + \right) / q_f$	The IP header of packet leaving are checked for filtering criteria, if criteria is met packet is routed otherwise it is not sent to destination host.
4.	Ingress Filtering	$dl_f^{(t+1)} = (1-\frac{ce_i^{(t)}}{ci_i^{(t)}(1+\beta_i^{(t)}}$)+	In this method filters identify the packets entering the domain and drops the traffic with IP address that does not match the domain prefix connected to a ingress router.
5.	Preferential Dropping	$d_i^{(n+1)} = (1-\frac{l_i^{(n)}}{m_i^{(n)}(1+\beta_i^{(n)}}$ +	This method simply drops the request when the load increases either by server or router with the requester making the request system to solve a hard puzzle.
6.	Probabilistic HCF	$P\{N_1=n,N_2=m\} = e^{-\lambda_E}\frac{(\lambda p)^m}{n!}e^{-\lambda(1-p)}\frac{(\lambda(1-p)^m}{m!}$	In this method the average arrival rate of packets and error probability of packets is used to calculate the number of malicious packets then filtering is done using the HCI-Algorithm.
7.	Pushback	-------	In this method when the congestion level reaches a certain threshold, sending router starts dropping the packets and illegitimate traffic can be calculated by counting the number of packets dropped for a particular IP address as attackers change their IP address constantly.
8.	Distributed Throttling	$r_s := (L_s+U_s) / f(k)$	This method sets the routers that access the server with a logic to adjust the incoming traffic to levels that are safe and prevent flood attacks.
9.	IP Trace Back- Rate Limiting	$N_L F_L < B < (N_A+N_L)$ x F_L	In this Internet traffic is trace back to the true source rather spoofed IP address which helps in identifying attackers traffic and possibly the attacker.

10.	Path Fingerprint	P[M(R_i→R1)=M(R_j→R2)) □ (M(R1→R3)=M(R2→R3))]= $\frac{1}{2^{2n}}$	Path Fingerprint represents the route an IP packet takes and is embedded in each IP packet, IP packet with incorrect path fingerprint are considered spoofed.
11.	Simple HCF	------	Hop Count value i.e. difference between initial TTL and final TTL of the spoofed IP packets is not consistent with legitimate IP packet is used to build HCF table which helps in mitigating DDOS attacks.

λ – poisson's distribution of packets arrival rate at the server

p – probability of malicious packets arriving at the server

1-p – probability of non-malicious packets arriving at the server

n – joint probability of malicious packets among total traffic

m - joint probability of non-malicious packets among total traffic

M – total no. of packets arrived with poisson's distribution λ

$d\theta_f^{(t+1)}$ - dropping probability for egress filtering

$q_f^{(t+1)}$ - no. of packets from ISP

$\eta_f^{(t+1)}$ – no. of packets to the ISP

S_{max} – maximum packet symmetry during normal case

$di_f^{(t+1)}$ - dropping probability for ingress filtering

S_{min} - minimum packet symmetry during normal case

η_g - throttle rate

l_w - Lower water mark

U_w - upper water mark

f(k) – estimate of no. of throttle points

$l_i^{(n)}$ - no. of packets in hash table 2

$l_i^{(n)}$ - no. of packets in hash table 2

M (Ri) – n-bit marking that router Ri inserts

$d_i^{(n+1)}$ - dropping probability for n+1 time

$m_i^{(n)}$ - no. of packets in hash table 1

N_L - pieces of legitimate flows

F_L - rate of flow of packets

N_A - pieces of attack flows

B – bandwidth

N_2- no. of non malicious packets

N_1- no. of malicious packets

APPENDIX 2: TCL SCRIPT

In many points TCL is similar to C, especially for loop structures, function definitions and mathematical or conditional expressions. In other points such as expressions evaluation and list data structures TCL has inherited the benefits from Scheme Language.
In TCL all data is represented as Strings.

Each Tcl command call is a sentence of the form : *command arg1 arg2 arg3*

The Tcl evaluator take each word of this sentence and evaluate it. After evaluation of each word, the first word (command) is considered to be a function name and this function is executed with as arguments the following words.

To evaluate a word, the interpretor has to do the following substitutions in the word string :

- If the word is surrounded by *" "*, this word may contain spaces, but substitution is still applicable inside the quotations. Inside the quotation, there may be spaces and carriage returns.
- If a word is surrounded by *{ }*, this word is unaffected (substitution is thus not applicable on this word). Inside the braces, there may be spaces and carriage returns. Moreover, the *{ }* braces may be nested.
- If a part of the word is surrounded by *[ ]*, this part is considered as a command sentence : the text within the brackets is evaluated as a Tcl command and replaced with the result.
- where substitution is applicable, every string beginning with *$* is replaced with the variable represented by this string. This string is ended by a space, a '-' or a ','.

Some TCL Features

All operations are commands, including language structures. They are written in prefix notation.

- Commands are commonly variadic.
- Everything can be dynamically redefined and overridden.
- All data types can be manipulated as strings, including source code.
- Event-driven interface to sockets and files. Time-based and user-defined events are also possible.
- Variable visibility restricted to lexical (static) scope by default, but uplevel and upvar allowing procs to interact with the enclosing functions' scopes.
- All commands defined by Tcl itself generate error messages on incorrect usage.
- Extensibility, via C, C++, Java, and Tcl.
- Interpreted language using bytecode
- Full Unicode (3.1) support, first released 1999.
- Cross-platform: Windows API; Unix, Linux, Macintosh, etc.
- Close integration with windowing (GUI) interface Tk.
- Multiple distribution mechanisms exist:
- Full development version (e.g., ActiveState Tcl)
- tclkit (kind of single-file runtime, only about 1 megabyte in size)
- starpack (single-file executable of a script/program, derived from the tclkit technology)
- freewrapTCLSH turns TCL scripts into single-file binary executable programs.
- BSD licenses, freely distributable source.[20][21]

APPENDIX 3: CODING

```
#Create a simulator object

set ns [new Simulator]
$ns rtproto DV

#Open the output files

set f0 [Open g0.tr w]
set  f1 [Open g1.tr w]
set  f2 [Open g2.tr w]

#Adding these four lines.
#Open the output files

set f0 [Open n0.tr w]
set f1 [Open n10.tr w]
set f2 [Open n6.tr w]
set f3 [Open n12.tr w]

#Adding these four lines.
#Open the output files

set f0 [Open r0.tr w]
set f1 [Open r10.tr w]
set f2 [Open r6.tr w]
set f3 [Open r12.tr w]

#Define different colors for data flows

$ns color 1 Blue
$ns color 1 Red
$ns color 1 Black
$ns color 1 green
```

```
#Open the nam trace file

set nf [open  out. nam w]
$ns namtrace-all $nf

#define different stepsizes

set stepsizea 0.1
set stepsizeb 0.9
set stepsizec 6.0
set stepsized 10.0

#define function for throughout vs time

proc throughputTime { }  {
set packet_delivered 0
set packet_sent 0
set throughput 0.0

#Get the current time

        set now [$ns now]

#Calculate the bandwidth (in MBit/s)

        puts   "$now  [expr $bw0/$time*8/1000000] "
        puts   "$now  [expr $bw1/$time*8/1000000] "
        puts   "$now  [expr $bw2/$time*8/1000000] "

throughput  [expr  ($packet_delivered/$packet_sent)*100]

#Re-schedule the procedure

        $ns at [expr  $now+$time]  "record"
        Trace_file.out -> throughput.data

#Call xgraph plotting program to plot throughput vs time
```

```
exec xgraph throughput.data  &
exit 0

}

#Define a 'finish' procedure

proc finish { } {
        global ns nf
        $ns flush-trace
       #Close the trace file


        close $nf

      #Execute nam on the trace file

exec nam out.nam &
#Adding these 8 lines
        global f0 f1 f2 f3
        #Close the output files
        close $f0
        close $f1
        close $f2
        close $f3
#Call xgraph to display the results

        exec xgraph n0.tr n10.tr n6.tr n12.tr –geometry 600x400 &
                exit 0
}

#Define a 'finish1' procedure

proc finish1 { } {
        global ns nf
        $ns flush-trace
        #Close the trace file
```

```
        close $nf
        #Execute nam on the trace file
        exec nam out.nam &

global f0 f1 f2
#Close the output files

        close $f0
        close $f1
        close $f2
#Call xgraph to display the results

exec xgraph g0.tr g10.tr g6.tr g12.tr –geometry 800x400 &
                exit 0
}
#Create 27 nodes

set n0 [$ns node]
$n0 set X_10
$n0 set Y_10
$n0 set Z_0

set n1 [$ns node]
$n1 set X_15
$n1 set Y_12
$n1 set Z_5

set n2 [$ns node]
$n2 set X_20
$n2 set Y_17
$n2 set Z_0

set n3 [$ns node]
$n3 set X_25
```

```
$n3 set Y_22
$n3 set Z_0

set n4 [$ns node]
$n4 set X_200
$n4 set Y_60
$n4 set Z_0

set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]
set n8 [$ns node]

# node n9 to n14 will later have throttle router.
set n9 [$ns node]
set n10 [$ns node]
set n11 [$ns node]
set n12 [$ns node]
set n13 [$ns node]
set n14 [$ns node]

#node n15 is a good user.
set n15 [$ns node]

#node n16 is an attacker.
set n16 [$ns node]
set n17 [$ns node]

#node n18 is a good user.
set n18 [$ns node]
set n19 [$ns node]

#node n20 is a good user.
set n20 [$ns node]
```

```
set n21 [$ns node]
set n22 [$ns node]

#node n23 and n24 are attackers.
set n23 [$ns node]
set n24 [$ns node]

#node n25 is a good user.
set n25 [$ns node]

#node n26 is an attacker.
set n26 [$ns node]

#node n27 is a good user.
set n27 [$ns node]

#Create links between the nodes
$ns duplex-link $n1 $n0 1Mb 10ms DropTail
$ns duplex-link $n2 $n0 1Mb 10ms DropTail
$ns duplex-link $n3 $n0 1Mb 10ms DropTail
$ns duplex-link $n4 $n0 1Mb 10ms DropTail
$ns duplex-link $n5 $n1 1Mb 10ms DropTail
$ns duplex-link $n6 $n2 1Mb 10ms DropTail
$ns duplex-link $n7 $n2 1Mb 10ms DropTail
$ns duplex-link $n8 $n3 1Mb 10ms DropTail
$ns duplex-link $n9 $n4 1Mb 10ms DropTail
$ns duplex-link $n10 $n5 1Mb 10ms DropTail
$ns duplex-link $n11 $n6 1Mb 10ms DropTail
$ns duplex-link $n12 $n7 1Mb 10ms DropTail
$ns duplex-link $n13 $n8 1Mb 10ms DropTail
$ns duplex-link $n14 $n8 1Mb 10ms DropTail
$ns duplex-link $n15 $n9 1Mb 10ms DropTail
$ns duplex-link $n16 $n10 1Mb 10ms DropTail
$ns duplex-link $n17 $n10 1Mb 10ms DropTail
```

```
$ns duplex-link $n18 $n11 1Mb 10ms DropTail
$ns duplex-link $n19 $n12 1Mb 10ms DropTail
$ns duplex-link $n20 $n12 1Mb 10ms DropTail
$ns duplex-link $n21 $n13 1Mb 10ms DropTail
$ns duplex-link $n22 $n14 1Mb 10ms DropTail
$ns duplex-link $n23 $n17 1Mb 10ms DropTail
$ns duplex-link $n24 $n19 1Mb 10ms DropTail
$ns duplex-link $n25 $n19 1Mb 10ms DropTail
$ns duplex-link $n26 $n21 1Mb 10ms DropTail
$ns duplex-link $n27 $n22 1Mb 10ms DropTail
$ns duplex-link $n10 $n6 1Mb 10ms DropTail
$ns duplex-link $n8 $n4 1Mb 10ms DropTail

#set the value of buffer size by user
puts “Enter buffer value, and then press Enter: “;
 gets stdin myUserInput;

#setting a limit on the maximum buffer size of the queue in the link
Between nodes <n1> and <n2>.
#$ns _ queue-limit <n1> <n2> <limit>

$ns queue-limit $n16 $n10 myUserInput
$ns queue-limit $n23 $n17 myUserInput
$ns queue-limit $n17 $n10 myUserInput
$ns queue-limit $n16 $n10 myUserInput
$ns queue-limit $n10 $n5 myUserInput
$ns queue-limit $n10 $n6 myUserInput
$ns queue-limit $n5 $n1 myUserInput
$ns queue-limit $n6 $n2 myUserInput
$ns queue-limit $n2 $n0 myUserInput
$ns queue-limit $n1 $n0 myUserInput
$ns queue-limit $n18 $n11 myUserInput
$ns queue-limit $n11 $n6 myUserInput
$ns queue-limit $n25 $n19 myUserInput
```

```
$ns queue-limit $n24 $n19 myUserInput
$ns queue-limit $n20 $n12 myUserInput
$ns queue-limit $n19 $n12 myUserInput
$ns queue-limit $n12 $n7 myUserInput
$ns queue-limit $n7 $n2 myUserInput
$ns queue-limit $n27 $n22 myUserInput
$ns queue-limit $n22 $n14 myUserInput
$ns queue-limit $n14 $n8 myUserInput
$ns queue-limit $n26 $n21 myUserInput
$ns queue-limit $n21 $n13 myUserInput
$ns queue-limit $n13 $n8 myUserInput
$ns queue-limit $n8 $n3 myUserInput
$ns queue-limit $n8 $n4 myUserInput
$ns queue-limit $n3 $n0 myUserInput
$ns queue-limit $n15 $n9 myUserInput
$ns queue-limit $n9 $n4 myUserInput
$ns queue-limit $n4 $n0 myUserInput
```

#Monitoring the queue-length in each node

```
$ns duplex-link-op $n16 $n10 queuePos 0.5
$ns duplex-link-op $n23 $n17 queuePos 0.5
$ns duplex-link-op $n17 $n10 queuePos 0.5
$ns duplex-link-op $n16 $n10 queuePos 0.5
$ns duplex-link-op $n10 $n5 queuePos 0.5
$ns duplex-link-op $n10 $n6 queuePos 0.5
$ns duplex-link-op $n5 $n1 queuePos 0.5
$ns duplex-link-op $n6 $n2 queuePos 0.5
$ns duplex-link-op $n1 $n0 queuePos 0.5
$ns duplex-link-op $n1 $n0 queuePos 0.5
$ns duplex-link-op $n18 $n11 queuePos 0.5
$ns duplex-link-op $n11 $n6 queuePos 0.5
$ns duplex-link-op $n25 $n19 queuePos 0.5
$ns duplex-link-op $n24 $n19 queuePos 0.5
```

```
$ns duplex-link-op $n20 $n12 queuePos 0.5
$ns duplex-link-op $n19 $n12 queuePos 0.5
$ns duplex-link-op $n12 $n7 queuePos 0.5
$ns duplex-link-op $n7 $n2 queuePos 0.5
$ns duplex-link-op $n27 $n22 queuePos 0.5
$ns duplex-link-op $n22 $n14 queuePos 0.5
$ns duplex-link-op $n14 $n8 queuePos 0.5
$ns duplex-link-op $n26 $n21 queuePos 0.5
$ns duplex-link-op $n21 $n13 queuePos 0.5
$ns duplex-link-op $n13 $n8 queuePos 0.5
$ns duplex-link-op $n8 $n3 queuePos 0.5
$ns duplex-link-op $n8 $n4 queuePos 0.5
$ns duplex-link-op $n3 $n0 queuePos 0.5
$ns duplex-link-op $n15 $n9 queuePos 0.5
$ns duplex-link-op $n9 $n4 queuePos 0.5
$ns duplex-link-op $n4 $n0 queuePos 0.5

# Create a UDP agent and attach it to node n16

set udp1 [new Agent/UDP]
$udpl set class_ 1
$ns attach-agent $n16 $udp1

#Create a CBR traffic source and attach it to udp1
set cbr1 [new Application/Traffic/ CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1

#Create a UDP agent and attach it to node n23
set udp2 [new Agent/UDP]
$upd2 set class_ 2
$ns attach-agent $n23 $udp2
```

```
#Create a CBR traffic source and attach it to udp2
set cbr2 [new Application/Traffic/CBR]
$cbr2 set packetsize_  500
$cbr2 set interval_ 0.005
$cbr2 attach-agent $udp2

#Create a TCP agent and attach it to node n18
set tcp0 [new Agent/TCP]
$tcp0 set class_ 0
$ns attach-agent $n18 $tcp0

#Create a CBR traffic source and attach it to tcp0
set cbr3 [new Application/Traffic/CBR]
$cbr3 set packetsize_  80
$cbr3 set interval_ 0.005
$cbr3 attach-agent $tcp0

#Create a TCP agent and attach it to node n24
ser tcp1 [new Agent/TCP]
$tcp1 set class_ 1
$ns attach-agent $n24 $tcp1
set sink [new Agent/TCPSink]
$ns attach-agent $ns24 $sink
$ns connect $tcp1 $sink
#$tcp set fid_ 19

#Create a CBR traffic source and attach it to tcp1
set cbr4 [new  Application/Traffic/CBR]
$cbr4 set packetsize_ 200
$cbr4 set interval_ 0.005
$cbr4 attach-agent $tcp1

#Create a UDP agent and attach it to node n20
set udp5 [new Agent/TCP]
```

```
$udp5 set class_ 5
$ns attach-agent $n20 $udp5

#Create a CBR traffic source and attach it to udp5
set cbr5 [new Application/Traffic/CBR]
$cbr5 set packetsize_  150
$cbr5 set interval_ 0.005
$cbr5 attach-agent $udp5

#Create a UDP agent and attach it to node n15
set udp6 [new Agent/TCP]
$udp6 set class_ 6
$ns attach-agent $n15 $udp6

#Create a CBR traffic source and attach it to udp6
set cbr6 [new Application/Traffic/CBR]
$cbr6 set packetsize_  150
$cbr6 set interval_ 0.005
$cbr6 attach-agent $udp6

#Create a UDP agent and attach it to node n26
set udp7 [new Agent/UDP]
$udp7 set class_ 7
$ns attach-agent $n26 $udp7

#Create a CBR traffic source and attach it to udp7
set cbr7 [new Application/Traffic/CBR]
$cbr7 set packetsize_  200
$cbr7 set interval_ 0.005
$cbr7 attach-agent $udp7

#Create a UDP agent and attach it to node n27
set udp8 [new Agent/UDP]
$udp8 set class_ 8
```

```
$ns attach-agent $n27 $udp8

#Create a CBR traffic source and attach it to udp8
set cbr8 [new Application/Traffic/CBR]
$cbr8 set packetsize_ 135
$cbr8 set interval_ 0.005
$cbr8 attach-agent $udp8

#Create a UDP agent and attach it to node n25
set udp9 [new Agent/UDP]
$udp9 set class_9
$ns attach-agent $n25 $udp9

#Create a CBR traffic source and attach it to udp9
set cbr9 [new Application/Traffic/CBR]
$cbr9 set packetsize_ 50
$cbr9 set interval_ 0.005
$cbr9 attach-agent $udp9

#Create a UDP agent and attach it to node n16
set udp10 [new Agent/UDP]
$udp10 set class_ 1
$ns attach-agent $n16 $udp10

#Create a CBR traffic source and attach it to udp10
set cbr10 [new Application/Traffic/CBR]
$cbr10 set packetsize_  500
$cbr10 set interval_ 0.005
$cbr10 attach-agent $udp10

#Create a TCP agent and attach it to node n23
set tcp2 [new Agent/TCP]
$tcp2 set class_ 2
$ns attach-agent $n23 $tcp2
```

```
#Create a CBR traffic source and attach it to tcp2
set cbr11 [new Application/Traffic/CBR]
$cbr11 set packetsize_  500
$cbr11 set interval_ 0.005
$cbr11 attach-agent $tcp2

#Create a UDP agent and attach it to node n24
set udp12 [new Agent/UDP]
$udp12 set class_ 4
$ns attach-agent $n24 $udp12

#Create a CBR traffic source and attach it to udp12
set cbr12 [new Application/Traffic/CBR]
$cbr12 set packetsize_  200
$cbr12 set interval_ 0.005
$cbr12 attach-agent $udp12

#Create a UDP agent and attach it to node n26
set udp13 [new Agent/UDP]
$udp13 set class_ 7
$ns attach-agent $n26 $udp13

#Create a CBR traffic source and attach it to udp13
set cbr13 [new Application/Traffic/CBR]
$cbr13 set packetsize_  80
$cbr13 set interval_ 0.005
$cbr13 attach-agent $udp13

#Create a UDP agent and attach it to node n16
set udp14 [new Agent/UDP]
$udp14 set class_ 7
$ns attach-agent $n16 $udp14
```

```
#Create a CBR traffic source and attach it to udp14
set cbr14 [new Application/Traffic/CBR]
$cbr14 set packetsize_  80
$cbr14 set interval_ 0.005
$cbr14 attach-agent $udp14

#Create a UDP agent and attach it to node n10
set udp15 [new Agent/UDP]
$udp15 set class_ 7
$ns attach-agent $n10 $udp15

#Create a CBR traffic source and attach it to udp15
set cbr15 [new Application/Traffic/CBR]
$cbr15 set packetsize_  80
$cbr15 set interval_ 0.005
$cbr15 attach-agent $udp15

#Create a UDP agent and attach it to node n6
set udp16 [new Agent/UDP]
$udp16 set class_ 7
$ns attach-agent $n6 $udp16

#Create a CBR traffic source and attach it to udp16
set cbr16 [new Application/Traffic/CBR]
$cbr16 set packetsize_  80
$cbr16 set interval_ 0.005
$cbr16 attach-agent $udp16

#Create a UDP agent and attach it to node n10
ser udp17 [new Agent/UDP]
$udp17 set class_ 7
$ns attach-agent $n10 $udp17

#Create a CBR traffic source and attach it to udp17
```

```
set cbr17 [new Application/Traffic/CBR]
$cbr17 set packetsize_  80
$cbr17 set interval_ 0.005
$cbr17 attach-agent $udp17

proc throttle { }  {

#min load of the server
set Ls 100
#max load of the server
set Us 300
#small constant
set fk 2
#d represent constant additive step
set d 0.1
#here p represent traffic arrival step
set p 10
set rs [expr $Ls + $Us/$fk]
while {$1} {
#multicast current rate-throttle to Rk;
#monitor traffic arrival rate p for time window w;
#throttle rate not string enough
If ($p > $Us)
{
  Set rs [ expr $rs / $2]
}
#throttle too string
else if ($p < $Ls)
  {
        if ([expr $p - $ Plast] < e)
  {
        #remove rate throttle from Rk;
        break;
  }
```

```
  else
  {
        #try relaxing throttle by additive step
        set Plast = $P
        set rs [expr $rs + $d]
        }
}

else
{
   break;
}

}
}


#adding this function

#Define a procedure that attaches a UDP agent to a previously created node
# 'node' and attaches an Expoo traffic generator to the agent with the Ls=100 and Us=300
#characteristic values 'size' for packet size 'burst' for burst time
#'idle' for the idle time and 'rate' for burst peak rate. The procedure connects
# the source with the previously defined traffic sink 'sink' and returns the
#source odject.
proc attach-expoo-traffic {node sink size burst idle rate }  {
        # Get an instance of the simulator
        set ns [Simulator instance]

        #Create a UDP agent and attach it to the node
        set source [new agent/UDP]
        $ns attach-agent $node $ source

        #Create and Expoo traffic agent and set its configuration parameters
```

```
        set traffic [new Application/Traffic/Exponential]
        $traffic set packetSize_ $size
        $traffic set burst_time_ $burst
        $traffic set idle_time_ $idle
        $traffic set rate_ $rate

                #Attach traffic source to the traffic generator
                $traffic attach-agent $source
        #Connect the source and the sink
        $ns connect $source $sink
return $traffic
#Define a procedure which periodically records the bandwidth received by the
#three traffic sinks sink0/1/2 and writes it to the three files f0/1/2.
proc record {} {
                global sink 0 sink 1 sink 2 f0 f1 f2
        #Get an instance of the simulator
        set ns [Simulator instance]
        #Set the time after which the procedure should be called again
                set time 0.5
        # How many bytes have been received by the traffic sinks?
                set bw0 [$sink0 set bytes_]
                set bw1 [$sink1 set bytes_]
                set bw2 [$sink2 set bytes_]
        #Get the current time
                set now [$ns now]
        #Calculate the bandwidth (in MBit/s) and write it to the files
                puts $f0 "$now [expr $bw0/$time*8/1000000]"
                puts $f1 "$now [expr $bw1/$time*8/1000000]"
                Puts $f2 "$now [expr $bw2/$time*8/1000000]"
        #Reset the bytes_values on the traffic sinks
                $sink0 set bytes_ 0
                $sink1 set bytes_ 0
                $sink2 set bytes_ 0
        #Re-schedule the procedure
```

```
                $ns at [expr $now+$time] “record”
}

# Create three traffic sinks and attach them to the node0 , node0 , node 0
set sink0 [new Agent/LossMonitor]
set sink1 [new Agent/LossMonitor]
set sink2 [new Agent/LossMonitor]
$ns attach-agent $n0 $sink0
$ns attach-agent $n0 $sink1
$ns attach-agent $n0 $sink2

#Create three traffic sources
set source0 [attach-expoo-traffic $n0 $sink0 200 2s 1s 100k]
set source1 [attach-expoo-traffic $n1 $sink1 200 2s 1s 200k]
set source2 [attach-expoo-traffic $n2 $sink2 200 2s 1s 300k]

#Create  a Null agent (a traffic sink) and attach it to node n3
set null0 [new Agent/Null ]
$ns attach-agent $n0 $null0

set null1 [new Agent/Null ]
$ns attach-agent $n10 $null1

set null2 [new Agent/Null ]
$ns attach-agent $n6 $null2

Set null3 [new Agent/Null ]
$ns attach-agent $n12 $null3

#Connect the traffic sources with the traffic sink
$ns connect $upd1 $null0
$ns connect $upd2 $null0
$ns connect $tcp0 $null0
$ns connect $tcp1 $null0
```

$ns connect $upd5 $null0

$ns connect $upd6 $null0

$ns connect $upd7 $null0

$ns connect $upd8 $null0

$ns connect $upd9 $null0

$ns connect $upd10 $null1

$ns connect $tcp2 $null1

$ns connect $upd12 $null3

$ns connect $upd13 $null0

$ns connect $upd14 $null1

$ns connect $upd15 $null2

$ns connect $upd16 $null0

$ns connect $upd17 $null0

#Schedule events for the CBR agents

$ns at 0.3 “$cbr1 start”

$ns at 0.4 “$cbr2 start”

$ns at 0.5 “$cbr3 start”

$ns at 0.6 “$cbr4 start”

$ns at 0.7 “$cbr5 start”

$ns at 0.8 “$cbr6 start”

$ns at 0.6 “$cbr7 start”

$ns at 0.9 “$cbr8 start”

$ns at 1.0 “$cbr9 start”

$ns at 2.0 “$cbr10 start”

$ns at 2.0 “$cbr11 start”

$ns at 5.0 “$cbr12 start”

$ns at 2.2 “$cbr13 start”

$ns at 6.0 “$cbr2 start”

$ns at 6.0 “$cbr4 start”

$ns at 6.0 “$cbr7 start”

$ns at 2.0 “$cbr17 start”

#Start logging the received bandwidth

$ns at 0.0 “record”

#Start the traffic sources
$ns at 10.0 “$source0 start”
$ns at 10.0 “$source1 start”
$ns at 10.0 “$source2 start”
$ns at 2.1 “$cbr14 start”
$ns at 2.01 “$cbr15 start”
$ns at 2.02 “$cbr16 start”

$ns at 2.0 “$cbr1 stop”
$ns at 2.0 “$cbr2 stop”
$ns at 7.0 “$cbr3 stop”
$ns at 5.0 “$cbr4 stop”
$ns at 7.0 “$cbr5 stop”
$ns at 7.0 “$cbr6 stop”
$ns at 2.0 “$cbr7 stop”
$ns at 7.0 “$cbr8 stop”
$ns at 7.0 “$cbr9 stop”
$ns at 7.0 “$cbr10 stop”
$ns at 6.0 “$cbr11 stop”
$ns at 7.0 “$cbr12 stop”
$ns at 7.0 “$cbr13 stop”
$ns at 7.0 “$cbr14 stop”
$ns at 7.0 “$cbr2 stop”
$ns at 7.0 “$cbr4 stop”
$ns at 7.0 “$cbr7 stop”
$ns at 7.0 “$cbr17 stop”
$ns at 7.0 “$cbr15 stop”
$ns at 7.0 “$cbr16 stop”

#Stop the traffic sources
$ns at 50.0 “$source0 stop”

```
$ns at 50.0 “$source1 stop”
$ns at 50.0 “$source2 stop”

#node failure at node n3 at time 1.0 sec
$ns rtmodel- at 1.0 down $n3
$ns stmodel-at 2.0 up $n3

#link failure between node n6 & n2 at time 3.0sec

$ns rtmodel-at 3.0 down $n6 $n2
$ns rtmodel-at 4.0 up $n6 $n2

#define throttle rate vs no. of packets
proc throttlerate { } {
set noofpackets 10
set t 0.0
set n 10.0
set ti 1.2
set ri 1.0
set Ti expr [$ti * $ri]
#Get the current time
                    set now [$ns npw]

For {set i 0} {$i < $n} {incr i} {
   If([expr ($t-$ti)] < $Ti)
    {
  set Ti [expr ($t - $ti]
     }
noofpackets = $noofpackets+10
}

#Calculate the bandwidth (inMBit/s)
          puts  “$now [expr $bw0/$time*8/1000000]”
          puts  “$now [expr $bw1/$time*8/1000000]”
```

```
        puts  "$now [expr $bw2/$time*8/1000000]"
#Re-schedule the procedure
        $ns at [expr $now+$time] "record"
# Call xgraph plotting program to plot throttle rate vs no. of packets
exec  xgraph throttle.data &
exit 0
}

#Define a 'finish2' procedure
proc  finish2 {} {
        global ns nf
        $ns flush-trace
       #Close the trace file
        close $nf
      #Execute nam on the trace file
        exec nam out.nam &
#Adding these 8 lines
        global f0 f1 f2 f3
        #Close the output files
        close $f0
        close $f1
        close $f2
        close $f3

#Call xgraph to display the results
        exec xgraph r0.tr r10.tr r6.tr r12.tr –geometry 600x400 &
        exit 0
}

#Call the finish procedure after 60 seconds of simulation time
$ns at 60.0 "finish"

#Call the finish procedure after 8 seconds of simulation time
$ns at 8.0 "finish1"
```

#Call the finish procedure after 10 seconds of simulation time

$ns at 10.0 “finish2”

#Run the simulation

$ns run [36]

APPENDIX 4: PUBLICATION

Conference Name: International Conference on Advanced Computing , Communication and Networks'11

Paper Title: DDOS Mitigation Techniques – A Survey

Authors: Dhwani Garg

Location: CHANDIGARH, INDIA

Conference Name: National Conference on Advances in Computational Intelligence

Paper Title: DDOS Attacks Router Throttling Mitigation Technique (Paper Accepted)

Authors: Dhwani Garg

Location: ROHTAK,INDIA

APPENDIX 5: REFERENCES

[1] Abraham Yaar, Adrian Perrig, Dawn Song "Pi: A Path Identification Mechanism to Defend against DDOS Attacks" IEEE 2003.

[2] Antonis Michalas, Nikos Komninos, Neeli R. Prasad, Vladimir A. Oleshchuk"New Client Puzzle Approach for DoS Resistance in Ad hoc Networks" IEEE 2010.

[3] Antonio Challita, Mona El Hassan, Sabine Maalouf, Adel Zouheiry "A Survey of DDOS Defense Mechanisms"

[4] Biswa Ranjan Swain, Bibhudatta Sahoo "Mitigating DDoS attack and Saving Computational Time using a Probabilistic approach and HCF method" IEEE International Advance Computing Conference, March 2009.

[5] Fasheng Yi, Shui Yu, Wanlei Zhou, Jing Hai and Alessio Bonti,"Source based Filtering Scheme Against DDOS Attacks" International Journal Database of Theory and Application.

[6] Gal Badishi, Amir Herzberg, Idit Keidar, Oleg Romanov, Avital Yachin "An Empirical Study of Denial of Service Mitigation Techniques" IEEE 2008.

[7] Nicholas A. Fraser, Douglas J. Kelly, Richard A. Raines, Rusty O. Baldwin and Barry E. Mullins "Using Client Puzzles to Mitigate Distributed Denial of Service Attacks in the Tor Anonymous Routing Environment" ICC,2007.

[8] Palvinder Singh Mann, Dinesh Kumar "A Reactive Defense Mechanism based on an Analytical Approach to Mitigate DDoS Attacks and Improve Network Performance" International Journal of Computer Applications, January 2011.

[9] Ping Du, Akihiro Nakao "Mantlet Triology: DDOS Defense Deployable with Innovative Anti-Spoofing, Attack Detection and Mitigation" IEEE 2010.

[10] Rajesh Sharma, Krishan Kumar, Kuldip Singh, R.C. Joshi "Shared Based Rate Limiting; An ISP level Solution to deal DDOS Attacks" IEEE 2006.

[11] Raktim Bhattacharjee, S. Sanand, and S.V. Raghavan. "Path Attestation Scheme to avert DDoS Flood Attacks" International Federation for Information Processing,2010.

[12] Ruiliang Chen, Jung-Min Park, Randolph Marchany "A Divide –and-Conquer Strategy or Thwarting Distributed Denial-of-Service Attacks" IEEE 2007.

[13] Ruiliang Chen, Jung-Min Park "Attack Diagnosis: Throttling Distributed Denial of Service Attacks Close to the Attack Sources" IEEE 2005.

[14] V.Praveena, N.Kiruthika "New Mitigating Technique to Overcome DDOS Attack" World Academy of Science, Engineering and Technology 2008.

[15] W.J. Blackert, D.M. Gregg, A.K. Castner, E.M. Kyle, R.L. Hom, R.M. Jokerst "Analyzing Interaction Between Distributed Denial Of Service Attacks and Mitigation Technologies" IEEE 2003.

[16] Xiuli wang "Mitigation of DDOS Attacks through Pushback and Resource Regulation" International Conference on Multimedia and Information Technology 2008.

[17]Yinan Jing, Xueping Wang, Xiaochun Xiao, Gendu Zhang"Defending Against Meek DDOS Attacks By IP Trace-back based Rate Limiting"IEEE 2006.

[18] Yinghong Fan, Hossam Hassanein and Patrick Martin "Proactive Control of Distributed Denial of Service Attacks with Source Router Preferential Dropping" IEEE,2005.

[19] http://en.wikipedia.org/wiki/Tcl

[20] http://users.belgacom.net/bruno.champagne/tcl.html

[21] http://www.acunetix.com/general/images/websitesecurity/web-hacks.jpg

[22] http://www10.org/cdrom/papers/409/scenario-overview.gif

[23] http://global.crazyengineers.com/wp-content/uploads/2011/03/botnet.jpg

[24]http://4.bp.blogspot.com/_TWV_4RpG9Lo/TTF9kfhUMAI/AAAAAAAAAa0/cdJCjhQwyBk/s1600/viewer2.png

[25]http://t3.gstatic.com/images?q=tbn:ANd9GcRaPZnkmt3Kez3V2M_p3K32t1g_ZBVqJFzyY_wam9mkYCIxDUWjjbPiuoX3

[26] http://www.huawei.com/products/datacomm/dm/datacomm/enus/gc/1035_img/2.JPG

[27] http://blog.goodvikings.com/wp-content/uploads/dos-attack-index2.gif

[28] http://www.blogymate.com/BlogPost/BlogyMate.com-BlogTH2612011111741.jpg..

[29] http://www.blogymate.com/BlogPost/BlogyMate.com-BlogTH2612011111741.jpg.

[30] http://www.securitydocs.com/images/papers/dosfaq-1.png.

[31] http://tula.bofh.ru/articles/539/2005-dos3.gif.

[32] http://www.isi.edu/nsnam/ns/tutorial/

[33] http://www.blogymate.com/BlogPost/BlogyMate.com-BlogTH2612011111741.jpg.

[34] http://tula.bofh.ru/articles/539/2005-dos3.gif.

[35] http://www.isi.edu/nsnam/ns/tutorial/

[36] David K.Y Yau, John C.S Lui, Feng Liang and Yeung Yam,"Defending Against Distributed Denial-of Service Attacks With Max-Min Fair Server-Centric Router Throttles" IEEE Transactions on Parallel and Distributed Systems, July 2001