# Distributed Detection of DDOS Attack

K.SANTHOSH

A Thesis Submitted to

Indian Institute of Technology Hyderabad

In Partial Fulfillment of the Requirements for

The Degree of Master of Technology

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology
Hyderabad

Department of Computer Science and Engineering

July 2011

# Declaration

I declare that this written submission represents my ideas in my own words, and where ideas or words of others have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the Institute and can also evoke penal action from the sources that have thus not been properly cited, or from whom proper permission has not been taken when needed.

(Signature)

(K.SANTHOSH)

(CS09G003)

# Approval Sheet

This Thesis entitled Distributed Detection of DDOS Attack by K.SANTHOSH is approved for the degree of Master of Technology from IIT Hyderabad

(Dr. P. Rajalakshmi ) Examiner
Dept. of Electrical Engineering
IITH

(Dr. T. Bheemarjuna Reddy) Examiner
Computer Science and Engineerng
IITH

(Dr. M.V.Panduranga Rao) Adviser
Dept. of Computer Science and Engineerng
IITH

(Dr. Ashudeb Dutta) Chairman
Dept. of Electrical Engineering
IITH

## Acknowledgements

I am deepely indebted to my adviser Dr. M.V. Panduranga Rao for his constant guidance and support right from my problem selection to thesis documentation. He patiently listened to all the problems I faced during the course of this work and appropriately guided me with his full support.

I thank all the faculty members of the Department of Computer Science & Engineering for sharing their views and giving valuable suggestions during the discussion of my work in department seminars.

I thank all my classmates for their friendly support who made the stay at this institute enjoyable, we shared joy and knowledge. I thank all my friends at IIT Hyderabad for the same.

I thank our director Prof. U.B.Desai for his friendly administrative support in getting all our requirements done as quickly as possible.

Finally, I thank all my family members for their constant love and support.

# Dedication

To My Parents, My dear friend Nishanth

# Abstract

Denial Of Service (DOS) and Distributed Denial Of Service (DDOS) attacks are attempts to make a server resources unavailable to its intended users. Information Security has three fundamental objectives: they are information integrity, confidentiality and availability. Denial Of Service attack is an attack on availability. In this attack the attacker makes the server busy in processing illegitimate requests thereby making server resources unavailable for legitimate clients. In Distributed Denial Of Service attack, multiple DOS attacks are carried out from several slaves (infected systems which are choosen as attacking agents) at a time on the victim (target server). SYN flooding DDOS attack is one type of DDOS attack. In SYN flooding DDOS attack, TCP SYN packets are used as attack packets. In SYN flooding DDOS attack, the attacker sends flood of SYN packets to victim server with spoofed source IP addresses. Server stores the state information of each of these attack connections. Server responds with SYN-ACK packets which are destined to spoofed IP addresses, so attacker do not recieves SYN-ACK packets. It causes the wastage of server resources in storing connection information of half open connections (Half open connection is a connection which is established from only one side of communicating parties). The victim server is busy in processing SYN requests which are originted from attacker, thus server is in a position to not serve for legitimate clients.

Our thesis mainly focuses on detecting SYN flooding DDOS attack. There are number of solutions proposed in the literature for detectinging SYN flooding DDOS attack. Most of the techniques depends on some of the assumptions in fields in the packet header, for example in [1] the authors assumed that attackers can not modify TTL (Time To Live) value. In their approach they used TTL field for calculating hop count. But in fact if the attacker spoofs the intial TTL values, then the proposed algorithm may not work properly. In [2] the blocking of TCP SYN flooding attack is done at a centralised, so in this approach also even if the attacker want to knockout the router, attacker can overwhelm the router with flood of spoofed IPs, thereby making the router to not work properly.

Ganguly et al [3] have proposed a solution that can be used to track SYN flooding DDOS attack effectively. In their approach they have used an variant of Flajolet Martin algorithm called distinct-count-sketch. The distinct-count-sketch is used to keep track of top-K destinations having most number of half open connections with distinct sources. The distinct-count-sketch algorithm performs detection of DDOS attack in a centralised manner. In our approach we propose a technique to reduce traffic in the network and load on server which is very high in Ganguly et al [3] centralised approach. We propose a distributed detection of SYN flooding DDOS attack. The aim of this thesis is to compare the results in detection of DDOS attack in both centralised monitoring and distributed monitoring approaches. To validate our distributed detection of DDOS attack we conducted experiments in NS2 (Network Simulator 2) [4]. We have conducted three types of experiments to check validity of our approach. In all experiments the results in the centralised and distributed approaches matched well. This encourages research in distributed detection of SYN flooding DDOS attack instead of centralised approach.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Denial Of Service (DOS) and Distributed Denial Of Service (DDOS) attacks are attempts to make a server resources unavailable to its intended users. Information Security have three fundamental objectives: they are Information Integrity, Confidentiality and Availability. Denial Of Service attack is an attack on availability. In this attack the attacker makes the server busy in processing illegitimate requests thereby making server resources unavailable for legitimate clients. In Distributed Denial Of Service attack, multiple DOS attacks are carried out from several slaves (infected systems which are chosen as attacking agents) at a time on the victim (target server). The DOS attack can be carried out in various forms such as crashing servers, crashing routers, overwhelming the network with high traffic, damaging server critical resources etc. Denial of Service attack victim can be either server, operating system, protocol which is used in network communication, network bandwidth, disk space, routing information etc. If the victim is server then we can have various types of Denial Of Service attacks like reflector attack, smurf attack etc. If the operating system is the victim then the attack is carried out by knowing the vulnerabilities in the design or implementation of operating system. Denial Of Service attack can be performed by knowing the vulnerabilities in the protocol thereby making protocol not working. We present a brief history of DDOS attacks and damage inflicted by such attacks. The extent of historical and potentially future damage will also serve as motivation of this thesis.

## 1.1 History of DOS and DDOS Attacks

The DOS attacks are as old as internet. Several new techniques were employed in the 90s for performing DOS attack. We now discuss some important attacks in the as a motivation for the study of DDOS detection.

**The Morris Worm:** On November 2nd 1988 the first DDOS attack was launched on the world wide web. It is referred as Morris Worm [5]. As a result of the attack about 15% (about 6000) of the systems which were connected to the network were infected and stopped their functioning.

**SYN flooding attack:** The TCP SYN flooding attack was discovered in 1994 by Bill Cheswick and Steve Bellovin[6][7]. The SYN flooding attack is explained in detail in section 1.3.2.

**Ping of Death:** This attack was carried out in 1995 [8]. It is the first popular DOS attack based on anomalies in TCP/IP design. This attack uses fragmentated packets as sources of DOS attack.

The target of this attack is the operating system. A ping is normally 56 bytes in size (or 84 bytes when IP header is considered). Many computers could not handle a ping packet larger than the maximum IPv4 packet size, which is 65,535 bytes. Sending a ping of more than this size could crash the target computer. Various variants of window such as Windows3.1x, Windows95 and Windows NT operating systems, as well various versions of Linux versions prior to 2.0.32 and 2.1.63 are vulnerable to this attack.
**Smurf Attack:** In 1998 this attack was first carried out. It is named after the exploit program which is used for the attack [9]. In this attack ICMP echo replies are used as attack traffic. It is an example of reflector attack. In Reflector attack, instead of sending attack packets directly to victim the attacker broadcasts ICMP echo requests to a network with source IP as victim IP. The system which are in the network will respond with ICMP echo reply there by making the server busy in processing ICMP echo reply which is wastage of server processing time.
**DDOS Attack on University of Minnesota:** On August 17th, 1999, large scale DDOS attacked the college network of University of Minnesota. The network which was used by faculty, students internet got shut down for several hours because of this attack. In this attack used trinoo DDoS tool was used. In this attack a flood of UDP packets with a 2-byte payload was generated and did not use IP spoofing [10]. It was reported that in this attack, the attacker had used 227 zombies, which includes 114 zombies which were from high speed, high capacity internet [11].
**DDOS Attack week:** February 7th, 8th and 9th 2000 three days are important in the history of internet for DDOS attack, because during these three days several high profiles servers like yahoo, amazon, e-bay as well as other world major web sites were shut down for several hours because of DDOS attack [12]. This attack is stand as a special case in the history of DDOS attack, because DDoS attacks were a combination of four types of DDoS attacks tools: Trinoo, TFN, TFN2K and Stacheldraht [13]. In this attack, attack traffic consists of UDP, SYN, ICMP and smurf packets. The DDOS attack of a few hours caused huge economical loss for these sites. The loss was in millions of dollars, for example yahoo advertising revenue that amounted to 500,000 dollars lost because of its shut down for 3 hours.
**Attack on wikileaks:** In February 2008, there is a attack on wikileaks. In this attack, attacker used DNS request as attack traffic. In this attack, a huge amounts of requests sent to the Web site's DNS servers. The bogus traffic from the DDoS attack reached peaks of 500Mbps and quickly flooded the DNS servers, bringing the whole Web site down [14].

## 1.2 DOS Attack

In Denial Of Service attack attacker sends flood of requests to the victim, thereby making the victim(target server) in a position to not serve for legitimate clients. The victim can be network, protocol, operating system, infrastructure etc. If the victim server is a then we can have various types of Denial Of Service attacks like SYN flooding attack, reflector attack, smurf attack etc. If the victim is a protocol we can implement Denial Of Service attack by knowing the vulnerabilities in the protocol. SYN flooding attack is a result of weakness in TCP 3-way handshake procedure, ping of death is a result weakness in IP fragmentation technique. Details of ping of death is explained in section 1.3.2. If the operating system is the victim then the attack is carried out by knowing the vulnerabilities in the design or implementation of operating system. Ping Of Death is an attack on operating system. In this attack the operating system of the victim crashes.

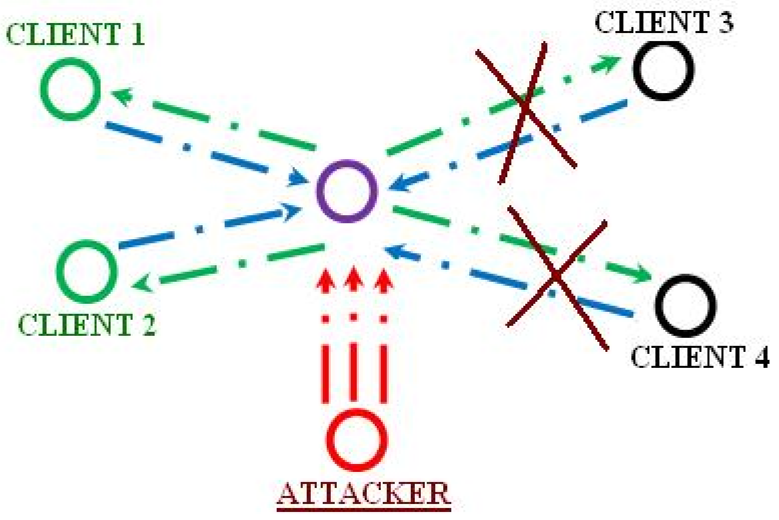

Figure 1.1: DOS Attack

## 1.3 DDOS Attack

Distributed Denial Of Service attack is an extension of DOS attack. Wherein the attackers starts attacking the victim with DOS attacks at the same time in coordination.

In DDOS attack there is one master(who is attacker) and number of zombies (attacking agents). Master is responsible for issuing control commands for zombies, and the zombies are responsible for generating actual attack traffic. As number of attacking agents are participating in DDOS attack, there is very less probability for the zombies to get detected by victim.

Initially the master searches for systems on the internet having vulnerabilities that allows the systems to generate attack traffic. Once the master finds such systems it recruits them as attacking agents. Once recruitment of zombies is done, the master installs neccessary code which contains control commands for DDOS attack in zombies. The control commands includes setting of target server, type of attack packets to be generated, start time of attack, duration of attack etc. Upon recieving control commands from master, zombies starts attacking the victim with neccessary attack traffic. The type of communication required between master and zombie is configured in the attack code.

### 1.3.1 Phases of DDOS attack

DDOS Attack is carried out in in two phases. In the first phase attacker (master) recruits the zombies, in the second phase actual attack will takes place. First phase is called as Deployment

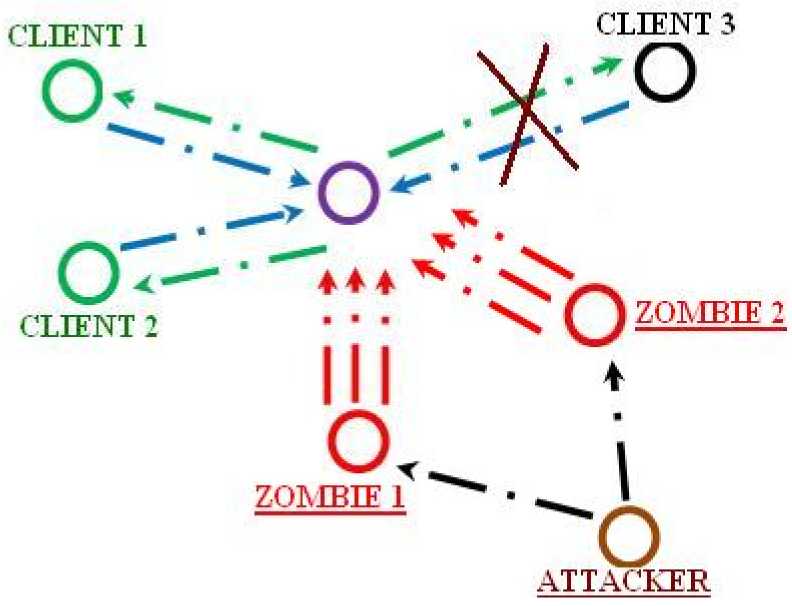

Figure 1.2: DDOS Attack

phase, second phase is called as Attack phase [15].

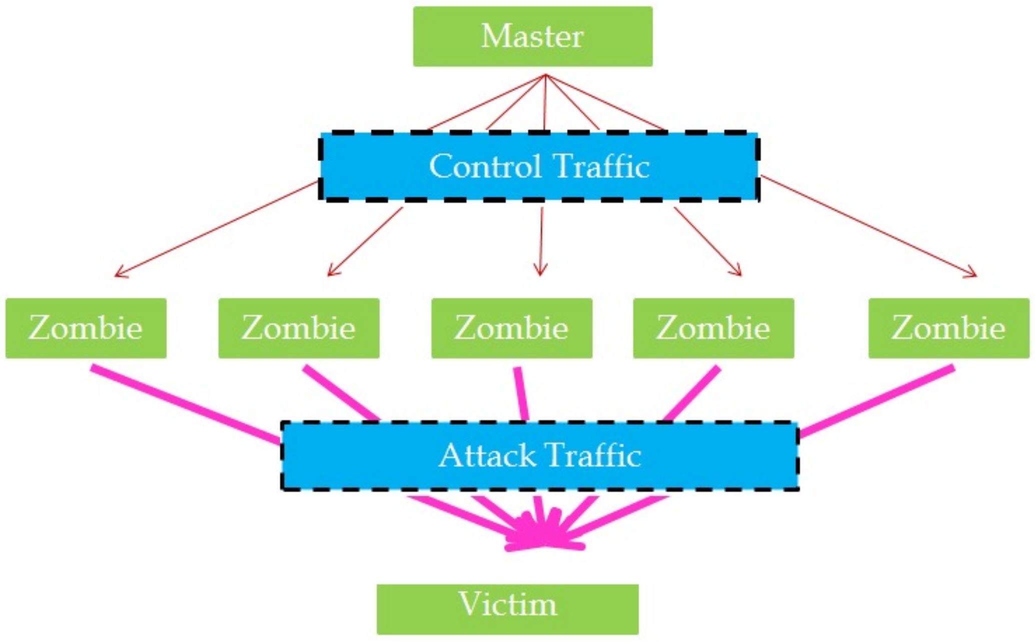

Figure 1.3: DDOS Attack phases

### 1.3.2 Types of DDOS Attack

There are a number of mechanisms to carry out DDOS. Following are a few impotant examples of DDOS.
**SYN Flooding Attack:** TCP SYN Flooding is result of weaknesses in TCP Protocol design. It uses the flaws in the TCP 3-way handshake mechanism [15]. The TCP 3 way handshake mechanism is shown in Figure 1.4. In SYN flooding attack, the attacker sends flood of SYN packets to victim

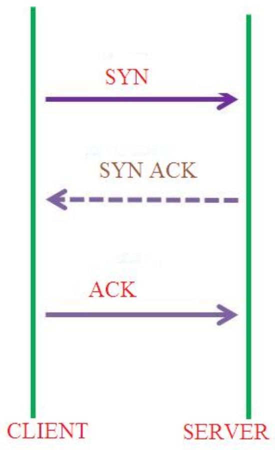

Figure 1.4: TCP 3 way handshake

server with spoofed source IP addresses [16]. Server stores the state information of each of these attack connections. State information includes, the source IP addres, source port, destination IP address and destination port etc. Server responds with SYN-ACK packets which are destined for spoofed IP addresses, so attacker does not recieves SYN-ACK packets. It causes the wastage of server resources in storing connection information of half open connections (Half open connection is a connection which is established from only one side). The victim server is busy in processing SYN packets which are originated from attacker, thus server is not in a position to serve legitimate clients.
**UDP flooding Attack or Fraggle:** In UDP flooding attack, the attacker sends flood of UDP packets with spoofed source IP to victims random ports. If the ports of the victim are closed then victim responds with ICMP packets to spoofed source IPs saying that destination is not reachable. If number of UDP packets are considerably high, this attack will consume most of victim server time for replying with ICMP packets. so the victim performance may goes down [17].
**Ping Of Death:** A Ping Of Death is an attack on operating system. A ping is normally 56 bytes in size (or 84 bytes when IP header is considered). However, a larger packet can be transmitted if it is fragmented. On a vulnerable system, a buffer overflow can occur when the packet is reassembled, causing the victim to freeze or crash. Many computers could not handle a ping packet larger than the maximum IPv4 packet size, which is 65,535 bytes. Sending a ping of more than this size could crash the target computer [8].
**DNS Attack:** DNS attack is attack on network bandwidth [15]. In DNS attack attacker sends multiple DNS requests to different Name Servers in the network with source IP of request is victim IP. The Name Servers which recieved DNS requests responds with DNS Reply. The attackers sends

multiple DNS requests to different servers at a time, so the victim server recieves the flood of DNS replies from different DNS servers at the same time. DNS reply have more size than that of DNS request, so it results in wastage of network bandwidth [17].

## 1.4 Detection Approaches for DDOS Attack

We can fight DDOS attacks either by detecting them (it allows DDOS attack to happen) or by preventing them in the first pahse. In this thesis, we focus only on DDOS detection mechanism, in particular detecting SYN flooding attacks.

In order to detect attack traffic, the detection system should be able to distinguish between the normal traffic and attack traffic. DDOS attack detection can be done in various ways such as:

**Pattern Detection:** In this type of detection technique the attack is detected by comparing the attack packet with known signature of DDOS attack. In this type of detection approach already known DDOS attack signatures are stored at detection system. Detection system is employed to compare each incoming packets with already known attack signature. If it finds similarity between the attack signature and incoming packet signature, then it will raise an alarm about DDOS attack happening [18].

**Anamoly Detection:** In this type of detection technique, the server is trained with normal traffic pattern ( or traffic distribution) [19], [20]. By considering the pattern anomalies in the various network parameters such as anomalies in inter packet arrival time,drop rate of packets at queue, the size of the packets, IP hop values, IP destination address, network performance etc. the detection system is in a position to detect the happening of DDOS attack. The detection system periodically computes the incoming traffic pattern, once it finds an anomaly in the traffic pattern the detection system will raise the alarm about DDOS attack.

**Third party detection:** In this technique, servers do not employ detection system by itself, they rely on third party attack detection system. The third party can detect attack by following any one of the above detection techniques [21]. In DDOS Attack as the attackers uses spoofed IP address, so it is difficult for the victim to detect the source of Attack. As the internet is built on end-to-end strategy, and intermediate systems do not maintain information about packets, it is difficult to trace the source of attack .

**ICMP Traceback:** In this technique every router samples the forwarded packets with a very low probability (e.g., 1 out of 20,000) and sends an ICMP Traceback message to the destination. An ICMP Traceback message contains various useful information such as previous and next hop addresses of the router, timestamp, and authentication information. while packets are traversing in their path from the attacker A to the victim V, the intermediate routers (R1, R2, R3, R4, R5,and R6) take some of these packets and send ICMP Traceback messages to the destination. With enough messages, the victim can trace the network path A - V . There is a drawback in this technique, there is an extra overhead and delay in sending ICMP packets to destination [21].

**Packet Marking:** Packet Marking technique overcomes the drawback in ICMP Traceback technique mentioned above. Instead of sending sampled packets back to destination, the packets stores traversed routers id. Victim side server processes the packets, server extracts the path information from the information stored in the packets. By considering probabilistic distribution of number of packets from different paths, and taking top frequent regions of all the packets, we can identify the

source of the attack. The packet marking scheme [22] can be either by storing information of all routers or storing the probablistic routers id only. If we use probabilistic routers id, the amount of storage required to store path information is low.
**Rate limiting of various types of packets:** The rate of different types of packets in the network can be used for detecting the DDOS attack. For example in a normal traffic the number of SYN packets will be 2 to 5 percentage of overall traffic. In normal scenerion the number of UDP packets, ICMP packets destined to server are very low, So whenever server recieve the packets which are in a out range of normal values, then server can activate the packet filtering system to detect the attack traffic from normal traffic [23].
**Filtering:** Firewalls, IDS (Intrusion Detection Systems) can be used as DDOS attack detection systems. But these IDS should be trained properly in order have high true positives and low false positives of attack packet detection [24].

Many techniques depend on the anamolies in some of the fields in the packet header. For example in [1] the authors assumed that attackers can not modify hop count. In their approach, they used TTL field for calculating hop count. But in fact if the attacker spoofs the initial TTL values, then the proposed algorithm may not work properly.

In [2] the blocking of TCP SYN flooding attack is done in a centralised way, so even if the attacker want to knockout the router, attacker can overwhelm the router with flood of spoofed IPs, thereby making the router to not work properly, so the detection system for DDOS attack detection may fail. In [25], detection of DDOS attack is achieved by calculating distance from TTL value, but it is always not possible to calculate true distance from TTL. In [26], detection of DDOS attack is done by considering anamolies in source IP address. Anamolies in source IP addresses can not reflect the syn flooding DDOS attack, so it can not detect syn flooding DDOS attack.

**Ganguly et al Real Time Detection of DDOS Attack:** Ganguly et al [3] have proposed a solution which can be used for tracking SYN flood attack effectively. In their approach they have used a variant of Flajolet Martin algorithm called as Distinct Count Sketch. The Distinct Count Sketch is used to keep track of top-K destinations having most half open connections from distinct sources. The advantage in their approach is that it allows deletion of legitimate connections from the pool of observation. The Ganguly et al [3] approach is as follows: for each of incoming SYN packet the count for that destination is incremented by 1 and for each of the ACK count for the destination is decremented by 1. Each system in the network sends a tuple (Source, Destination, $\pm$ 1) is sent to the centralised DDOS monitor. The processing of this data is done at Centralised DDOS monitor. The Distinct Count Sketch algorithm runs at this Centralised DDOS monitor. The input for Distinct count sketch is collection of such tuples. The output is top K destinations.

They believe that a distinct-source frequency metric for a destination IP address provides a very robust indicator of potential DDoS activity. So the problem, is seeks to find the top-k destinations connected to the most distinct sources. As Count Sketch synopses incur a small (logarithmic) number of steps to process each streaming update. Their algorithm is guaranteed logarithmic time to find an approximate set of top-k destinations (and, corresponding distinct frequencies) that is provably close (with high probability) to the actual top-k set. As it is taking less time it can be readily deployed to monitor large ISP networks transmitting large volumes of IP packet data. A distinct-count sketch imposes a small space overhead, and can be efficiently maintained by performing a guaranteed small (i.e., logarithmic) number of simple hash operations per element in the stream.

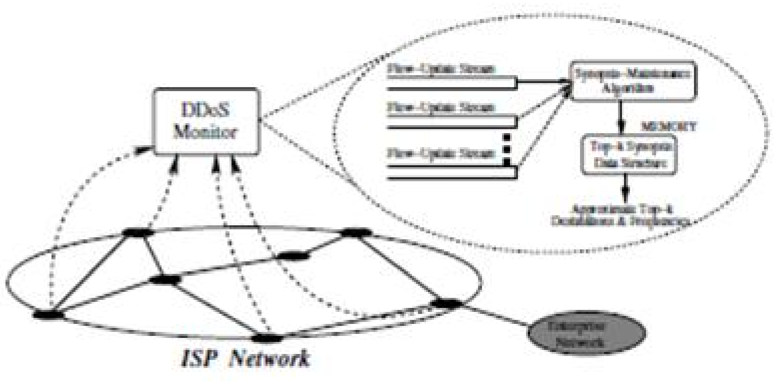
DDoS Monitor
Flow-Update Stream
Flow-Update Stream
Flow-Update Stream
Synopsis-Maintenance Algorithm
MEMORY
Top-k Synopsis Data Structure
Approximate Top-k Destinations & Frequencies
ISP Network
Enterprise Network

**Figure 1.** Update-Stream Processing Architecture.

Figure 1.5: Ganguly et al approach

In [27] paper, they discussed classification and characteristics of the source address spoofing. Source address spoofing can be classified in six categories. The characteristics of first category of source address spoofing are analyzed by the statistical analysis of packets collected by the CAIDA network telescope. They propose a technique to identify source IP address spoofing based on classification of IP addresses.

## 1.5 Our Approach to detect DDOS Attack

The approach of Ganguly et al [3] has the problem that tuples of the form (Source, destination,$\pm 1$) for all connections in the ISP have to be sent to a centralized DDOS monitor where the DDOS detection algorithm is run. This results in a high network overload, and high processing overhead at the DDOS monitor. In this thesis, we propose a technique that reduces both the the traffic in the network and load on the DDOS attack monitoring server. Instead of sending all the tuples to DDOS attack monitoring server, just send the output of local DDOS detection algorithm to DDOS attack monitoring server. DDOS attack monitoring server merges outputs from all routers in the network and it perform DDOS attack detection on the outputs recieved from all routers in the network. In thesis, we mainly focus on the SYN flooding DDOS attack detection. We have conducted experiments in which the DDOS attack monitoring is done both in centralised and in distributed ways, and the outputs in both cases are match well. This thesis is organised as follows in chapter 2 discusses some preliminaries required for our approach. In third chapter we discuss our approach and experimental results and discuss a technique for preventing IP-spoofing. In the last chapter we conclude with discussion and future work.

# Chapter 2

# Preliminaries

In our approach we used data stream algorithms such as Misra Gries Algorithm and count-min sketch. In addition to these algorithms we used heap sort algorithm to maintain top-K destinations. In this chapter we discuss count-min-sketch and Misra Gries algorithm.

## 2.1 Count Min Sketch

Count min sketch was developed by Graham Cormode and S.Muthu Krishnan [28]. It is used to answer frequency related queries in the data stream processing. Count min sketch is a variant of Bloom Filter [29], [30]. It is used to store and retrieve the frequencies (counts) of input elements efficiently. The name of count-min-sketch is derived based on the two operations which are performed in count-min sketch. The first operation is counting frequency of input elements. The second operation is computing minimum. Count min sketch maintains a 2-D array of size w $*$ d, where w is the width (number of columns in array) of array and d is depth (number of rows in array) of array. Count min sketch maintains d hash functions $h_1, h_2, h_3, ..., h_d$. Each of these d hash functions are associated with one row in count-min-sketch array as follows: function $h_1$ is associated with first row, function $h_2$ is associated with second row, function $h_3$ is associated with third row etc. The Figure 2.1 shows the schematic diagram of count-min-sketch.

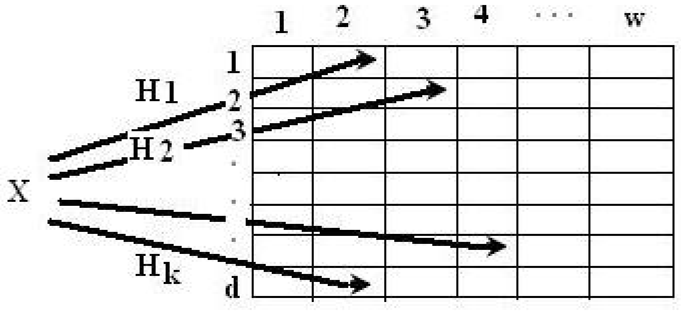

Figure 2.1: Count-min-sketch

The input parameters for count-min-sketch are accuracy ($\epsilon$), certainty ($\delta$). The hash table size and number of hash functions required for count-min-sketch are calculated from the input parameters $\epsilon$ and $\delta$ as follows:

Hash table size (w)= e/$\epsilon$

Number of hash functions (d)= ln 1/$\delta$.

Initially all entries in the count-min-sketch array are intialized to zero, which means that counts of all input elements is intialized to zero. The d hash functions are used to insert frequencies of elements into array and deletion of frequencies of elements from array. The update procedure for inserting (or deleting) an element to (from) count-min-sketch is as follows:

### 2.1.1 Update Procedure:

To insert an element X into count-min-sketch array, the procedure is as follows: we calculate hash values of X w.r.t all hash functions $h_1(X)$, $h_2(X)$, $h_3(X)$, ... $h_d(X)$. The count of counters which are at i$^{th}$row, $h_i(X)^{th}$ column of (where i=1, 2, 3, ..., d) count-min-sketch array are incremented by 1. To delete an element X from count-min-sketch array, we follows same procedure as insertion procedure of count-min-sketch, but instead of incrementing value of counters we decrement the value of counters by 1.

### 2.1.2 Procedure To Answer A Query:

The query for count-min-sketch is finding the frequency of an input element. The procedure to find frequency of an element X is as follows: First we calculate the hash values of X w.r.t all d hash functions. Next the minimum count among all counts which are at calculated hash values is taken as true count for that element. The proof of algorithm correctness can be found in [28].

## 2.2 Misra Gries Algorithm:

Misra Gries algorithm is used find the frequent elements in an input data stream or input array. The input for the algorithm is set of elements in a data stream or an array of elements. The parameter for Misra Gries algorithm is a constant number K. The output of the algorithm is set elements which are having frequency greater than N/K, where N is number of elements already inserted into algorithm. The basic idea of Misra Gries algorithm is Pigeon hole principle. Lets recall the Pigeonhole principle. There are n pigeons, m holes and m $<$ n. This implies that $\geq$ 2 pigeons have to go into one hole.

Now the idea of this algorithm is explained by considering K=2, then we generalise the idea of algorithm for K $>$ 2.

Let us assume that in an input array of size N, there are d (where d $<$ N) distinct elements $Y_1$, $Y_2$, $Y_3$, ..., $Y_d$, each with frequencies $f_1$, $f_2$, $f_3$, ..., $f_d$ respectively. Now we consider the possibilities of existence of an element X with frequency greater than N/2.

For values of d (where d $<$ N/2), we can construct N/2 tuples with size of each tuple is equal to 2. An element is said to be in majority if it is having frequency greater than N/2, it follows that the number of remaining elements is less than N/2 (because sum of frequency of majority element and sum of frequencies of all minorities elements is equal to N). If we constraint ourself to pairing of 2 different elements, the majority element should pair with one minority element, it results in some of the majority elements gets unpaired which means that those are majority in number. This result (majority elements gets ended up with unpaired) is key element of Misra Gries algorithm.

Now we correlate the above basic idea with Misra Gries algorithm. In our basic idea we constrained ourself with pairing of 2 different elements. The pairing of elements can be done by using an array of size $2 * 2$, where zeroth row represents element and first row represents the count of element. For an input element, if element is already exists in the array just increment the count of the element (so that it should reflect the amount by which the element is in majority). If on the otherhand if element is not found, but some free space found in the array, then insert the element into array with count 1. On inserting an element if the array is filled, then it means that we reached our requirement of pairing of different elements. Now we can cancel out the all elements in array by amount 1. It means that we are erasing a tuple of size 2. To erase a tuple of size 2, we just decrement the count of all elements in the array by 1. If we foud some element in the array with count greater than 1, it means that the element is in majority upto that instance of time. After cancellation of a tuple, if we found some element with count 0, we drop out that element from array, so that we can allot a free room for a new incoming element.

For values of d where d $\geq$ N/2 there exists no element with majority in frequency.

The above basic idea can be extended for K > 2. Let us assume that in an input array of size N, there are d (where d < N) distinct elements $Y_1$, $Y_2$, $Y_3$, ..., $Y_d$, each with frequencies $f_1$, $f_2$, $f_3$, ..., $f_d$ respectively. In this case, there exists at most K-1 elements which are having frequencies greater than N/K. Now consider pairing of K-different elements as a tuple, we can form at most N/K such tuples. N/K such tuples are possible where all input elements are having frequencies equal to K. We can form zero tuples where there are just less than K different elements. So at any case we can form at most N/K such tuples. So if we follow erasing of tuples, we can erase atmost N/K tuples. In the cancellation of all elements (different) in a tuple, an element gets erased if it is having frequency less than N/K and an element remains in the output if the frequency of element is greater than N/K. The below algorithm performs insertion of element and erasing of a tuple from Misra Gries array. The pseudo code for the Misra Gries Algorithm is as follows: [31], [32].

**Misra Gries Algorithm** Intialise: A $\leftarrow$ (Empty Array) Process J:

```
1. if J ∈ keys(A) then
2. A[J] ← A[J] + 1
3. else if | keys(A)| < k - 1 then
4.       A[J] ← 1
5. else
6.       for each l ∈ keys(A) do
7.            A[l] ← A[l] - 1
8.            if A[l] = 0 then remove l from A
```

OUTPUT: On query a, if a $\in$ keys(A), then report fa=A[a], else fa=0

## 2.3 Heap Sort Algorithm

The heap sort algorithm used to sort elements either in ascending order or in descending order. This algorithm uses heap data structure. Heap is complete binary tree in which all levels upto last level are filled, last level may not get filled fully. The heap sort algorithm can sort the elements in the tree either in ascending order or in descending order. This algorithm makes use of three methods they are heap insert, max heapify and shift down. whenever we insert a new element into heap tree, heap insertion method is called. Heap insertion algorithm inserts the element into appropriate position of heap. Max heapify and shift down methods are used to maintain max heap property on every node in the tree.

# Chapter 3

# Distributed Detection

Ganguly et al [3] proposed an approach for tracking SYN flood attacks effectively. In their approach they have used a variant of Flajolet Martin algorithm called the distinct count sketch algorithm. The Distinct Count Sketch is used to keep track of top-K destinations having most number of half open connections from distinct sources. The advantage in Ganguly et al. approach is that it allows deletion of legitimate connections from the pool of observation. Ganguly et al [3] approach is as follows: for each of incoming SYN packet the count for that destination is incremented by 1 and for each of ACK packet the count for that destination is decremented by 1. Each system in the network maintains a tuple (Source, Destination, +1/-1). The tuple is sent to the centralised DDOS attack monitor every time. The processing of this data is done at DDOS attack monitor. The distinct count sketch algorithm runs at this centralised DDOS attack monitor, for tracking of top destinations that have half open connections from the most number of sources. The input for distinct count sketch is a collection of such tuples. The output is top K destinations.

The key idea here is that the legitimate connections are associated with SYN and ACK, whereas the attack connections are associated with only SYN packets as there is no acknowledgement for attack connection. In Ganguly et al [3] approach as all tuples from all systems in the network are goes through a DDOS attack monitor, it causes lot of traffic in the network and lot of processing overhead on DDOS attack monitor.

## 3.1 Our Approach

We propose a technique to reduce traffic in the network and load on server which is very high in Ganguly et al [3] approach. Our approach uses ideas similar to Ganguly et al [3] but instead of centralised DDOS monitoring we propose to use distributed DDOS monitoring. Our technique is as follows: Every router in the network runs its own DDOS attack monitoring algorithm, and the output of router's DDOS attack monitoring algorithm is shared with centralised DDOS attack monitor. The centralised DDOS attack monitor merges all the outputs from all routers and it runs DDOS attack detection on merged output. The merging operation can be done by using some sufficiently large arrays or we can directly input the local routers output to centralised algorithm. As the size of output produced by router's DDOS attack detection algorithm is very low when compared with total size of all tuples for each connection in the network, the traffic in our approach is reduced and the processing overhead on the DDOS attack monitor is also reduced. The aim of this thesis is to compare the results in both centralised monitoring and distributed monitoring.

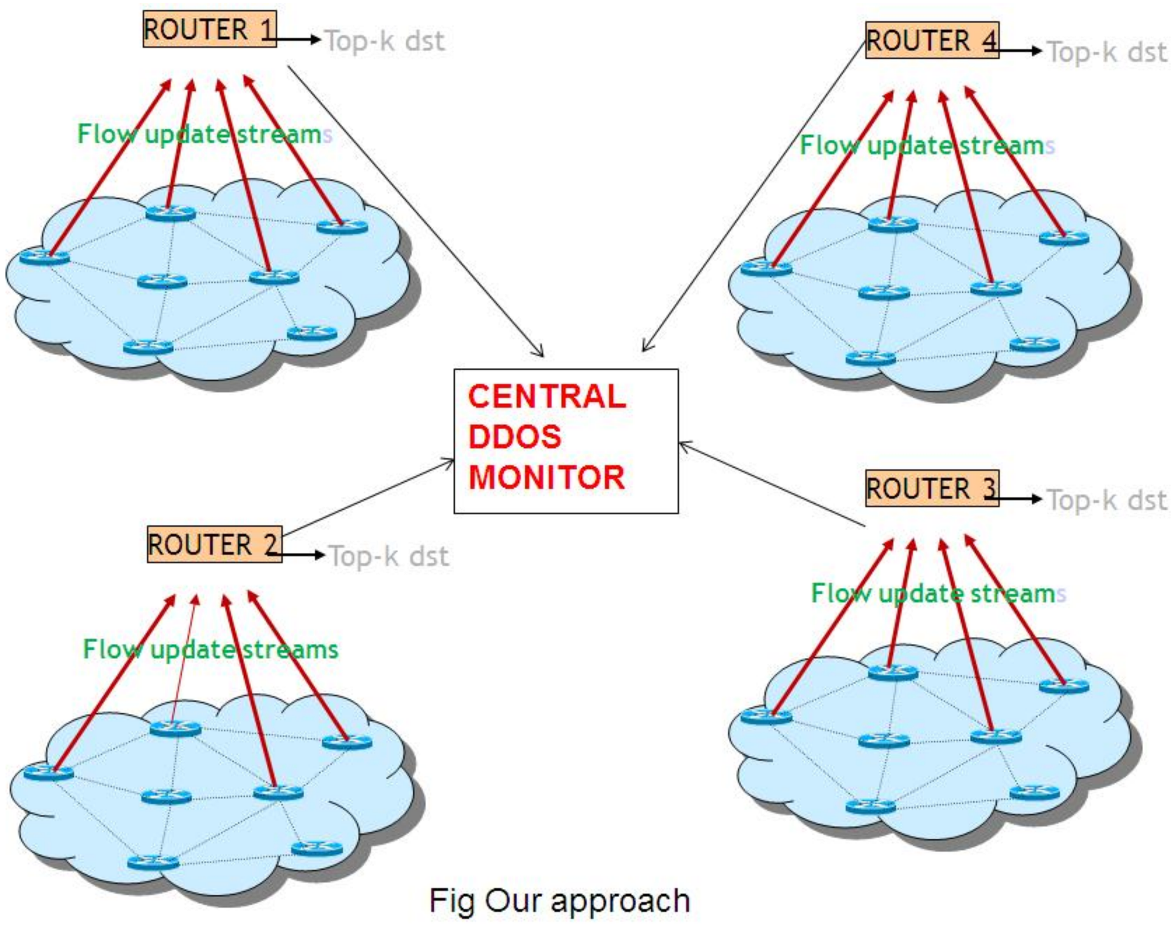

Figure 3.1: Our Approach

## 3.2 Experiments and Results

We have conducted three experiments to check validity of approach, of which two experiments are done in NS2 (Network Simulator 2) simulator [4]. The experiments are conducted by considering various random network topologies each with differenet sizes. Each topology in our experiments is built randomly, The random paramters in our experiments are as follows: topology of network, number of agents attached to every node, start time of each traffic generator, end time of each traffic generator. In all experiments, we tabulated our results as shown below. The columns in all tables named appropriately, but because space constraint we used some shortcut. The meaning of shortcuts we used are as follows: Dis or D means Distributed, Cen or C means Centralised, % Found means percentage of destinations detected by algorithm. False +ves reflect, false +ves in Detection of detections. We now explain our experiments and results in detail:

### 3.2.1 Experiment 1

This experiment is a static one, where we generate only arrays as input for the Misra Gries algorithm. To simulate the event that some legitimate connections might also be open at the time of making the measurement in real, we deliberately put every tuple in the array with a small probability. This serves as false positives.

The experimental setup of this experiment is as follows: We have constructed various network topologies various random parameters. The sizes of network topologies used in this experiment are 60, 120, 240 nodes. The random parameters in this experiment are as follows: The links between nodes (systems in real world) in the network, number of agents(protocols in real world) attached to a node, the start time and end time of traffic generator.

In this experiment we have used Misra Gries algorithm. We have experimented DDOS monitoring with low percentage (P) insertion of legitimate connections into Misra Gries algorithm and the attack connections are inserted with probability 1. We conducted this experiment in both centralised and distributed DDOS detection approaches. The results in both approaches match in most of the cases as shown in Tables 3.1 - 9. As expected, for high percentage insertion of legitimate connections the detection of victims which are under DDOS attack is low. For low percentage insertion of legitimate connections the detection of victims which are under DDOS attack is high. We have conducted this experiment by varying size of topology in the network. For a fixed size topologyd we varied percentage of legitimate that goes into Misra Gries algorithm. We have considered three different sizes of topology (60, 120, 240 nodes). We considered three different percentages (5, 40, 80) by which legitimate connections are inserted into Misra Gries algorithm. For each of these variables combination (topology size, percentage by which legitimate connections inserted into Misra Gries algorithm), we conducted 5 trials. We have taken the average of all results in all 5 trial as result of experiment with those variable (topology size, percentage by which legitimate connections inserted into Misra Gries) combination. We believe that average result of number of trials will reflect the true result with more accuracy.

| Trial | Output | | | Total Found | | % Found | | false +ves | |
|---|---|---|---|---|---|---|---|---|---|
| | Actual | Distributed | Centralised | Distributed | Cen | Dis | Cen | Dis | Cen |
| 1 | 13, 51, 47 | 47, **0**, 51 | 51, 13, 47 | **2** | **3** | **66** | **100** | **1** | **0** |
| 2 | 28, 55, 53 | **50** | 28, 53, 55 | **0** | **3** | **0** | **100** | **1** | **0** |
| 3 | 39, 40, 6 | - | 40, 6, 39 | **0** | **3** | **0** | **100** | **0** | **0** |
| 4 | 40, 53, 32 | 40 | 40, 32, 53 | **1** | **3** | **33** | **100** | **0** | **0** |
| 5 | 16, 24, 56 | - | 24, 16, 56 | **0** | **3** | **0** | **100** | **0** | **0** |
| TOTAL | | | | **3** | **15** | **20** | **100** | **2** | **0** |

Table 3.1: Experiment 1: 60 nodes, P=5

| Trial | Output | | | Total Found | | % Found | | false +ves | |
|---|---|---|---|---|---|---|---|---|---|
| | Actual | Distributed | Centralised | Distributed | Cen | Dis | Cen | Dis | Cen |
| 1 | 37, 8, 54 | 37, 8, 54 | 37, 54 | **3** | **2** | **100** | **66** | **0** | **0** |
| 2 | 24, 53, 48 | 24 | 48, 53, 24 | **1** | **3** | **33** | **100** | **0** | **0** |
| 3 | 37, 54, 17 | 17, 54 | 17, **59** | **2** | **1** | **66** | **33** | **0** | **1** |
| 4 | 17, 15, 53 | 53, 17 | 53, 15, 17 | **2** | **3** | **66** | **100** | **0** | **0** |
| 5 | 43, 21, 39 | 43 | 39, 21, 43 | **1** | **3** | **33** | **100** | **0** | **0** |
| TOTAL | | | | **9** | **12** | **60** | **80** | **0** | **1** |

Table 3.2: Experiment 1: 60 nodes, P=40

| Trial | Output | | | Total Found | | % Found | | false +ves | |
|---|---|---|---|---|---|---|---|---|---|
| | Actual | Distributed | Centralised | Distributed | Cen | Dis | Cen | Dis | Cen |
| 1 | 37, 49, 28 | 37 | 49, **48** | **1** | **1** | **33** | **33** | **0** | **1** |
| 2 | 55, 26, 27 | 55 | 55 | **1** | **1** | **33** | **33** | **0** | **0** |
| 3 | 16, 41, 6 | 41 | 41 | **1** | **1** | **33** | **33** | **0** | **0** |
| 4 | 15, 10, 56 | 19, 56 | 15, 56, 19 | **2** | **3** | **66** | **100** | **0** | **0** |
| 5 | 25, 8, 26 | 26, 25 | 25 | **2** | **1** | **66** | **33** | **0** | **0** |
| TOTAL | | | | **7** | **7** | **47** | **47** | **0** | **1** |

Table 3.3: Experiment 1: 60 nodes, P=80

| Trial | Output | | | Total Found | | % Found | | false +ves | |
|---|---|---|---|---|---|---|---|---|---|
| | Actual | Distributed | Centralised | Distributed | Cen | Dis | Cen | Dis | Cen |
| 1 | 32, 59, 23 | 23, 59, 32 | 23, 32, 59 | **3** | **3** | **100** | **100** | **0** | **0** |
| 2 | 6, 52, 12 | 6, 12, 52 | 12, 6, 52 | **3** | **3** | **100** | **100** | **0** | **0** |
| 3 | 42, 98, 71 | - | 42, 71, 98 | **0** | **3** | **0** | **100** | **0** | **0** |
| 4 | 24, 11, 115 | - | 115, 24, 11 | **0** | **3** | **0** | **100** | **0** | **0** |
| 5 | 39, 56, 77 | 56, 39 | 77, 39, 56 | **2** | **3** | **66** | **100** | **0** | **0** |
| TOTAL | | | | **8** | **15** | **54** | **100** | **0** | **0** |

Table 3.4: Experiment 1: 120 nodes, P=5

| Trial | Output | | | Total Found | | % Found | | false +ves | |
|---|---|---|---|---|---|---|---|---|---|
| | Actual | Distributed | Centralised | Distributed | Cen | Dis | Cen | Dis | Cen |
| 1 | 58, 3, 56 | 13 | 58, 13 | **1** | **2** | **33** | **66** | **0** | **0** |
| 2 | 65, 31, 42 | 65, 31 | 65, 31 | **2** | **2** | **66** | **66** | **0** | **0** |
| 3 | 84, 116, 21 | - | 21, 84, 116 | **0** | **3** | **0** | **100** | **0** | **0** |
| 4 | 89, 54, 99 | 89, 99 | 99, 89 | **2** | **2** | **66** | **66** | **0** | **0** |
| 5 | 55, 77, 95 | 77, 95 | 95, 55 | **2** | **2** | **66** | **66** | **0** | **0** |
| TOTAL | | | | **7** | **11** | **47** | **74** | **0** | **0** |

Table 3.5: Experiment 1: 120 nodes, P=40

| Trial | Output | | | Total Found | | % Found | | false +ves | |
|---|---|---|---|---|---|---|---|---|---|
| | Actual | Distributed | Centralised | Distributed | Cen | Dis | Cen | Dis | Cen |
| 1 | 46, 35, 51 | **112**, 35 | **112** | **1** | **0** | **33** | **33** | **1** | **1** |
| 2 | 67, 36, 0 | - | 36, **113** | **0** | **1** | **0** | **33** | **0** | **1** |
| 3 | 81, 60, 35 | 60, **1** | 81, **22** | **1** | **1** | **33** | **33** | **1** | **1** |
| 4 | 39, 67, 57 | 67, **111, 3** | **111**, 39 | **1** | **1** | **33** | **33** | **2** | **1** |
| 5 | 8, 51, 111 | 8 | **16**, 111 | **1** | **1** | **33** | **33** | **0** | **1** |
| TOTAL | | | | **4** | **4** | **27** | **27** | **4** | **5** |

Table 3.6: Experiment 1: 120 nodes, P=80

| Trial | Output | | | Total Found | | % Found | | false +ve | |
|---|---|---|---|---|---|---|---|---|---|
| | Actual | Distributed | Centralise | Dis | Cen | Dis | Cen | D | C |
| 1 | 154, 32, 212 | 32, 154, 212 | 212, 32, 154 | **3** | **3** | **100** | **100** | **0** | **0** |
| 2 | 187, 164, 141 | 141, 164, 187 | 141, 187, 164 | **3** | **3** | **100** | **100** | **0** | **0** |
| 3 | 155, 236, 213 | 155 | 213, 236, 155 | **1** | **3** | **33** | **100** | **0** | **0** |
| 4 | 138, 184, 211 | 184 | 211, 184, 138 | **1** | **3** | **33** | **100** | **0** | **0** |
| 5 | 92, 207, 163 | 92, 163, 207 | 92, 163, 207 | **3** | **3** | **100** | **100** | **0** | **0** |
| TOTAL | | | | **11** | **15** | **74** | **100** | **0** | **0** |

Table 3.7: Experiment 1: 240 nodes P=5

| Trial | Output | | | Total Found | | % Found | | false +ve | |
|---|---|---|---|---|---|---|---|---|---|
| | Actual | Distributed | Centralised | Dis | Cen | Dis | Cen | D | C |
| 1 | 173, 13, 201 | - | 13, **229** | **0** | **1** | **0** | **33** | **0** | **1** |
| 2 | 232, 49, 207 | 49, 232, **190** | 232, **214** | **2** | **1** | **66** | **33** | **1** | **1** |
| 3 | 126, 235, 133 | 126, 133, 235 | 235, 126, 133 | **3** | **3** | **100** | **100** | **0** | **0** |
| 4 | 184, 65, 193 | 65 | 65 | **1** | **1** | **33** | **33** | **0** | **0** |
| 5 | 105, 48, 93 | 93 | 48 | **1** | **1** | **33** | **33** | **0** | **0** |
| TOTAL | | | | **7** | **7** | **47** | **47** | **1** | **2** |

Table 3.8: Experiment 1: 240 nodes, P=40

| Trial | Output | | | Total Found | | % Found | | false +ves | |
|---|---|---|---|---|---|---|---|---|---|
| | Actual | Distributed | Centralised | Dis | Cen | Dis | Cen | Dis | Cen |
| 1 | 44, 26, 156 | 156 | 44 | **1** | **1** | **33** | **33** | **0** | **0** |
| 2 | 173, 27, 71 | 173 | 173 | **1** | **1** | **33** | **33** | **0** | **0** |
| 3 | 130, 160, 81 | **7**, 81, **204** | **7**, **204** | **1** | **0** | **33** | **0** | **2** | **2** |
| 4 | 107, 129, 103 | 129 | **24** | **1** | **0** | **33** | **0** | **0** | **1** |
| 5 | 212, 100, 113 | **10**, 212 | **79**, 212 | **1** | **1** | **33** | **33** | **1** | **1** |
| TOTAL | | | | **5** | **3** | **33** | **20** | **3** | **4** |

Table 3.9: Experiment 1: 240 nodes, P=80

### 3.2.2 Experiment 2

We have conducted this experiment to compare results in both distributed approach and centralised approach for detecting top destinations with maximum number of half open connections. To find top destinations we have used Misra Gries algorithm. In distributed approach each router in the network runs its own copy of Misra Gries algorithm, the outputs of routers Misra Gries algorithms is merged at a router and the merged output is given as input for local Misra Gries. In centralised approach, Misra Gries algorithm runs at a centralised DDOS monitor the input for this Misra Gries algorithm is all traffic in the network.

We have conducted this experiment using NS2 (Network Simulator 2) simulator [4]. The experimental setup of this experiment is as follows: We have constructed various network topologies with various parameters chosen randomly. The sizes of network topologies used in this experiment are 20, 30, 40 nodes. In this experiment, the parameters which choosen randomly are as follows: The links between nodes in the network, number of agents attached to a node, the start time and end time of traffic generator. This experiment is conducted to simulate the idea (insertion of all connections into pool of observation and deletion of legitimate connections from pool of observation) of Ganguly et al [3] with Misra Gries algorithm. The aim of this experiment is to compare the results in both distributed and centralised DDOS detection approaches with Misra Gries algorithm. As Misra Gries algorithm does not maintain the exact count of each element in data stream, there is no provision for deletion of element from Misra Gries algorithm. For simulating idea of Ganguly et al [3] in Misra Gries, we have used a sufficiently large array at each node in the network. The data structure which used for simulating idea of Ganguly et al [3] is in the form of (source, destination, destination port, 0/1). This data structure is maintained at each node in the network. The simulation of idea of Ganguly et al [3] is done as follows: at each node in the network from which SYN packet is passing, a tuple (source, destination, destination port, 1) is inserted into array at that node. For each ACK which is passing through a node, previous tuple which corresponds to (source, destination, destination port) is reset to 0, i.e. a tuple of the form (source, destination, destination port, 0) overwrites the tuple (source, destination, destination port, 1) in the array at that node. At end of simulation, we give all the destinations with value 1 in the array as input to Misra Gries algorithm. In distributed approach each node in the network implements their own copy of Misra Gries algorithm and the output of each node's Misra Gries algorithm is given as input to centralised node. This centralised node is responsible to merge all outputs of all nodes. In centralised approach all the arrays are mereged into one big array, and this big array is fed as input to Misra Gries algorithm. For small topology the results in both distributed approach and centralised approach match well and for larger topology there is large difference between the centralised approach and distributed approach. The experimental results are shown in Tables 3.10 - 12.

| Trial | Output | | | Total Found | | % Found | | false +ve | |
|---|---|---|---|---|---|---|---|---|---|
| | Actual | Distributed | Centralise | Dis | Cen | Dis | Cen | D | C |
| 1 | 3, 18, 2 | 2, 3, 18 | 2, 3, 18 | **3** | **3** | **100** | **100** | **0** | **0** |
| 2 | 17, 1, 16 | 16, 17 | 16, 17 | **2** | **2** | **66** | **66** | **0** | **0** |
| 3 | 5, 17, 14 | **8**, 14 | **8**, 14 | **1** | **1** | **33** | **33** | **1** | **1** |
| 4 | 5, 0, 19 | **6**, 0, 5 | **6**, 0, 5 | **2** | **2** | **66** | **66** | **1** | **1** |
| 5 | 3, 14, 12 | - | - | **0** | **0** | **0** | **0** | **0** | **0** |
| TOTAL | | | | **8** | **8** | **54** | **54** | **2** | **2** |

Table 3.10: Experiment 2: 20 nodes

| Trial | Output | | | Total Found | | % Found | | false +ves | |
|---|---|---|---|---|---|---|---|---|---|
| | Actual | Distributed | Centralised | Dis | Cen | Dis | Cen | Dis | Cen |
| 1 | 18, 24, 27 | 27, 24 | 18, 27, 24 | **2** | **3** | **66** | **100** | **0** | **0** |
| 2 | 22, 28, 2 | 2, **3**, 28 | 2, 22, 28 | **2** | **3** | **66** | **100** | **1** | **0** |
| 3 | 25, 27, 20 | 25, 20, 27 | 20, 27, 25 | **3** | **3** | **100** | **100** | **0** | **0** |
| 4 | 24, 3, 14 | 24, **4** | 3, 24, 14 | **1** | **3** | **66** | **100** | **1** | **0** |
| 5 | 19, 10, 18 | 10, 19, **3** | 10, 18, 19 | **2** | **3** | **66** | **100** | **1** | **0** |
| TOTAL | | | | **10** | **15** | **66** | **100** | **3** | **0** |

Table 3.11: Experiment 2: 30 nodes

| Trial | Output | | | Total Found | | % Found | | false +ve | |
|---|---|---|---|---|---|---|---|---|---|
| | Actual | Distributed | Centralised | Dis | Cen | Dis | Cen | D | C |
| 1 | 10, 36, 30 | **11** | 10, 36 | **0** | **2** | **0** | **66** | **1** | **0** |
| 2 | 6, 26, 37 | 37, **7** | 6, 26 | **1** | **2** | **33** | **66** | **1** | **0** |
| 3 | 23, 20, 22 | 20, 23 | 22, 20, 23 | **2** | **3** | **66** | **100** | **0** | **0** |
| 4 | 24, 32, 11 | **14** | 32, 24 | **0** | **2** | **0** | **66** | **1** | **0** |
| 5 | 0, 37, 30 | **1, 32** | 30, 37 | **0** | **2** | **0** | **66** | **2** | **0** |
| TOTAL | | | | **3** | **11** | **20** | **74** | **5** | **0** |

Table 3.12: Experiment 2: 40 nodes

### 3.2.3 Experiment 3

We have conducted this experiment to compare results in both distributed approach and centralised approach for detecting top destinations with maximum number of half open connections. To find top destinations we have used Count min sketch. The aim & experimental setup of this experiment is same as Experiment 2, the only difference is that in algorithms we used for finding top destinations.

We have conducted this experiment using NS2 (Network Simulator 2) simulator [4]. The experimental setup of this experiment is as follows: We have constructed various network topologies with various random parameters. The sizes of network topologies used in this experiment are 60, 120, 180 and 210 nodes. The random parameters in this experiment are as follows: The links between nodes in the network, number of agents attached to a node, the start time and end time of traffic generator. This experiment is conducted to simulate the idea (insertion of all connections into pool of observation and deletion of legitimate connections from pool of observation) of Ganguly et al [3] with count min sketch algorithm. The aim of this experiment is to compare the results in both distributed and centralised DDOS detection approaches with count min sketch algorithm. In this experiment, at each node in the network we have used a 2 D array of size wd (count min sketch array).

In this experiment, for each of SYN packet we increment the count of the destination in the count min sketch array by 1. The procedure for incrementing count of a destination is explained in detail in section 2.1. The count of a destination is decremented on recieving a ACK packet from client to that destination. Decrementing count of an element in count min sketch is also explained in detail in section 2.2. The count of the count min sketch is given as input for the heap sort algorithm, heap sort algorithm takes destination id and count of the destination as input. The tuple (destination id, destination count) is treated as a single data structure. The heap tree is constructed by taking these data structures as nodes of the tree. The heap sort algorithm always maintains the max heap property on the destination count. After inserting a new tuple into heap tree, max heapify procedure is called. Max heapify procedure is responsible for maintaining max heap property on destination count. The output results in both approaches found to be match well in both cnetralised and distributed DDOS detection approaches. The experimental results are shown in Table 3.13 - 16.

| S.No | Output | | | Found | | % Found | | false +ve | |
|---|---|---|---|---|---|---|---|---|---|
| | Actual | Distributed | Centralise | D | C | D | C | D | C |
| 1 | 44, 40, 8, 36 | 36, 44, 8, 40 | 36, 44, 40 | **4** | **3** | **100** | **75** | **0** | **0** |
| 2 | 0, 35, 13, 41 | 41, 13, 0, 35 | 41, 13, 0, 35 | **4** | **4** | **100** | **100** | **0** | **0** |
| 3 | 23, 52, 20, 8 | 23, 52, 20, 8 | 52, 8, 23, 20 | **4** | **4** | **100** | **100** | **0** | **0** |
| 4 | 45, 36, 0, 13 | 0, 36, 45, **2** | 0, 36, 45, 13 | **3** | **4** | **75** | **100** | **1** | **0** |
| 5 | 29, 19, 18, 8 | 29, 9, **0**, 18 | 19, 18, 8, 29 | **3** | **4** | **75** | **100** | **1** | **0** |
| | | | TOTAL | **18** | **19** | **90** | **95** | **2** | **0** |

Table 3.13: Experiment 3: 60 nodes

| S.No | Output | | | Found | | % Found | | false | |
|---|---|---|---|---|---|---|---|---|---|
| | Actual | Distributed | Centralise | D | C | D | C | D | C |
| 1 | 82, 87, 77, 99 | **18**, 87, 77, **37** | **18**, 87, **37**, 77 | **2** | **2** | **50** | **50** | **2** | **2** |
| 2 | 92, 32, 55, 26 | 92, 26, 55, **119** | 92, 55, **119**, 26 | **3** | **3** | **75** | **75** | **1** | **1** |
| 3 | 75, 113, 63, 17 | 17, 113, 75, **47** | 17, 113, **47**, 63 | **3** | **3** | **75** | **75** | **1** | **1** |
| 4 | 50, 100, 105, 93 | 100, 105, 93, **78** | 93, 100, **78**, 105 | **3** | **3** | **75** | **75** | **1** | **1** |
| 5 | 61, 113, 22, 58 | 113, 61, **81**, 58 | 113, 61, 58, 22 | **3** | **4** | **75** | **100** | **1** | **0** |
| | | | TOTAL | **14** | **15** | **70** | **75** | **6** | **5** |

Table 3.14: Experiment 3: 120 nodes

| S.No | Output | | | Found | | % Found | | false | |
|---|---|---|---|---|---|---|---|---|---|
| | Actual | Distributed | Centralise | D | C | D | C | D | C |
| 1 | 166, 101, 55, 79 | 79, 101, 166, 55 | 79, 101, **129**, 166 | **4** | **3** | **100** | **75** | **0** | **1** |
| 2 | 4, 2, 3, 1 | 4, **100**, 3, 1 | 1, **100**, 4, **80** | **3** | **2** | **75** | **50** | **1** | **2** |
| 3 | 145, 134, 142, 51 | 51, 142, 145, **10** | 51, 142, **62, 10** | **3** | **2** | **75** | **50** | **1** | **2** |
| 4 | 71, 148, 32, 2 | 148, 32, **176**, 71 | 148, 32, 2, **176** | **3** | **3** | **75** | **75** | **1** | **1** |
| 5 | 88, 128, 74, 127 | **21, 45**, 74, **107** | 74, **45**, 88, **2** | **1** | **2** | **25** | **25** | **3** | **2** |
| | | | TOTAL | **14** | **12** | **70** | **60** | **6** | **8** |

Table 3.15: Experiment 3: 180 nodes

| S.No | Output | | | Found | | % Found | | false | |
|---|---|---|---|---|---|---|---|---|---|
| | Actual | Distributed | Centralise | D | C | D | C | D | C |
| 1 | 154, 109, 200, 19 | 19, 154, 200, 109 | 19, 154, 200, 109 | **4** | **4** | **100** | **100** | **0** | **0** |
| 2 | 9, 27, 56, 128 | 27, 56, 128, **200** | 27, 56, 9, **200** | **3** | **3** | **75** | **75** | **1** | **1** |
| 3 | 26, 94, 44, 207 | 297, 180, 44, 26 | 180, 44, 26, 207 | **4** | **4** | **100** | **100** | **0** | **0** |
| 4 | 192, 164, 46, 165 | **185**, 46, 164, 192 | **185**, 46, 165, 192 | **3** | **3** | **75** | **75** | **1** | **1** |
| 5 | 202, 154, 69, 84 | 84, 69, **148, 35** | **35**, 84, 154, 69 | **2** | **3** | **50** | **75** | **2** | **1** |
| | | | TOTAL | **16** | **17** | **80** | **85** | **4** | **3** |

Table 3.16: Experiment 3: 210 nodes

## 3.3 A technique for preventing IP spoofing

IP spoofing is a key element for DDOS Attack. With IP Spoofing the attacker is able to generate a flood of packets with spoofed IPs. Our Internet is based on End-to-End architecture. This End-to-End architecture is the main source of IP spoofing. End-to-End architeture means all the functionalities in the network are performed at the end systems, for example error checking, authentication checking, data reassembling etc. are performed at end systems. In this chapter we propose a new idea to prevent IP spoofing. Our idea is very simple, at the same time it requires less processing time and less storage. Our proposed technique attempts to prevent local IP spoofing. Local IP spoofing means a system in a LAN (Local Area Network) uses IP address of another system which is in the same LAN. In our technique router is responsible to check authentication of each connection which are passing through it.

The detailed idea is as follows: Intially each system in the network selects two large prime numbers (Q,R) randomly. The selected prime numbers kept as secret by the system. The selected prime numbers are considered as full key, product of the selected prime numbers is considered as partial key(P) for first connection from that source. At the time of registration of IP address of system with local router, system shares it's partial key with the router. The product is considered as partial key for next new connection from that source. Whenever a new connection is going from a system, that system should give its full key (Q, R) in the first packet of connection. i.e. system should give the selected prime numbers (Q, R) in the first packet of connection. Once packet arrives at the router, router calculates the product of the two prime numbers which are stored in the packet, if calculated product (cP) matches with the stored product (sP) which corresponds that source IP address, then it means that the packet is from genuine system. For sharing the selected new prime numbers with router we can use either options field in the IP header or special packet.

Once a local router authenticates the source of connection, the local router should overwrite the full key field in the packet with the routers full key (Q,R) so that it is used by its downstream router for checking authenticity of connection by downstream router. Our algorithm is very strong in the sense that if we give a product of two prime numbers there is no known polynomial time algorithm which can find the factors the product. The above procedure is valid for only first connection of a system, and next connection's partial key(nP) can be shared with the router by using previous connection. i.e. while sending a connection full key (Q, R) from a system, it should also send the next connection partial key (nP) to router. The router perform authentication check by using full key (Q. R) in the packet, if connection is verified as it is from legitimate source, router overwrite the stored partial key (sP) which is stored in it's local memory with nP value. This nP value is used for authenticaition checkup of next connection from the same source IP. The above procedure of sending next partial key (nP), along with current full key (Q, R) should be followed for every connection. The below figure shows schematic procedure of our proposed idea for preventing IP Spoofing.

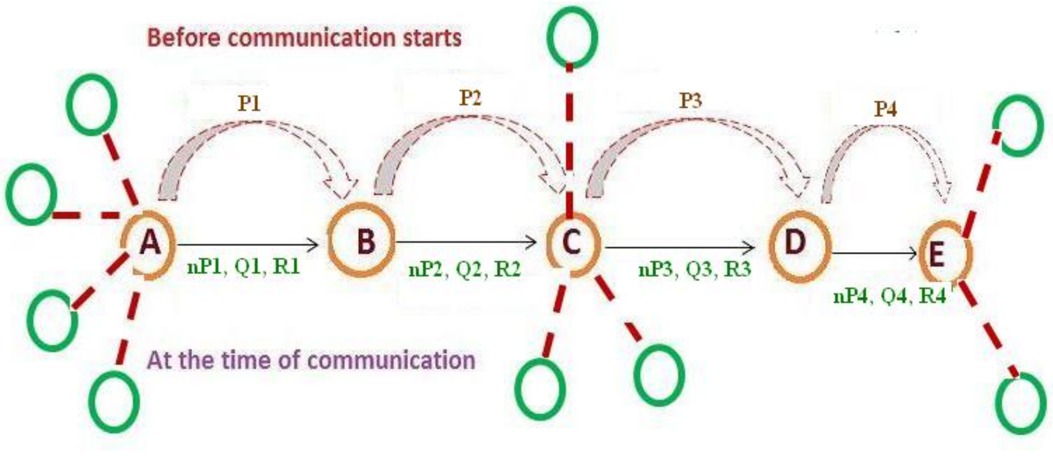

Figure 3.2: An idea to prevent IP Spoofing

# Chapter 4

# Conclusions and Future Work

This thesis is motivated by the idea of Ganguly et al [3] that tracking destinations having half open connections with a large number of distinct sources is useful in identifying potential targets. This tracking is done at a central DDOS monitor.

Our technique aims to reduce traffic in the network and load on the DDOS monitor which is very high in their approach. We conducted experiments in NS2 (Network Simulator 2) [4] to compare results in centralised and distributed DDOS attack detection. Experimental results show that the detection of DDOS attack in both centralised approach and distributed approach match well in several cases. Since a distributed approach decreases traffic on the network and computational overheads at the central DDOS monitor, we believe that it could be an interesting alternative to explore.

While our experiments were run on NS2, it will be interesting run them on real networks. We have validated our approach with Misra Gries algorithm and count-min-sketch. It would also be interesting to compare the results of the experiments in this thesis with a distributed version of the distinct-count-sketch based streaming algorithms of Ganguly et al.

# References

[1] I.B.Mopari, S.G.Pukale, and M.L.Dhone. Detection and Defense Against DDOS Attack with IP Spoofing. In ICAC3 '09. 2009 .

[2] Y. Kim, J.-Y. Jo, J. Chao, and F. Merat. High Speed Router Filter for Blocking TCP flooding under DDOS Attack 2003.

[3] S. Ganguly, M. N. Garofalakis, R. Rastogi, and K. K. Sabnani. Streaming Algorithms for Robust, Real-Time Detection of DDoS Attacks. In ICDCS. 2007 4.

[4] T. Issariyakul and E. Hossain. Introduction to Network Simulator NS2. 1st edition. Springer Publishing Company, Incorporated, 2008.

[5] H. K. Orman. The Morris Worm: A Fifteen-Year Perspective. *IEEE Security & Privacy* 1, (2003) 35–43.

[6] http://tools.ietf.org/html/rfc4987ref B96. syn flood history 2007.

[7] http://memex.org/meme2 12.html. syn flood.

[8] http://linuxmafia.com/faq/Security/ping-of death.html. ping of death.

[9] http://www.pentics.net/denial-of-service/white papers/smurf.cgi. smurf attack.

[10] http://denialofservice.uw.hu/ch03lev1sec3.html. DOS and DDOS evaluation.

[11] M. Sachdeva, K. Kumar, G. Singh, and K. Singh. Performance Analysis of Web Service under DDoS Attacks. In Advance Computing Conference, 2009. IACC 2009. IEEE International. 2009 1002 –1007.

[12] http://news.cnet.com/2100 1017-236683.html. feb 2000 ddos attack.

[13] http://book.soundonair.ru/cisco/ch14lev1sec4.html. Common Network Attacks.

[14] http://www.secure64.com/news-wikileaks-dns-server fire. wikileaks 2008.

[15] J. Mirkovic and P. Reiher. D-WARD: A Source-End Defense against Flooding Denial-of-Service Attacks. *IEEE Transactions on Dependable and Secure Computing* 2, (2005) 216–232.

[16] http://tools.ietf.org/html/rfc4987. syn flood.

[17] H. Chau. Network Security Defense Against DoS/DDoS Attacks.

[18] http://www.snort.org/. The Open Source Network Intrusion Detection System.

[19] J. Yan, S. Early, and R. Anderson. The XenoService  A Distributed Defeat for Distributed Denial of Service. *Proceedings of ISW 2000* .

[20] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. *SIGCOMM Comput. Commun. Rev.* 32, (2002) 62–73.

[21] B. S. L. M and T. Taylor. ICMP Traceback Messages 2003.

[22] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical network support for IP traceback. 2000 295–306.

[23] T. M. Gil and M. Poletto. MULTOPS: a data-structure for bandwidth attack detection. In In Proceedings of 10th Usenix Security Symposium. 2001 23–38.

[24] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827 (Best Current Practice) 2000. Updated by RFC 3704.

[25] Y. You, M. Zulkernine, and A. Haque. A Distributed Defense Framework for Flooding-Based DDoS Attacks. In ARES. IEEE Computer Society, 2008 245–252.

[26] T. Peng, C. Leckie, and K. Ramamohanarao. Detecting Distributed Denial of Service Attacks Using Source IP Address Monitoring. In In Proceedings of the Third International IFIP-TC6 Networking Conference (Networking 2004. Springer, 2002 771–782.

[27] J. Bi, P. Hu, and P. Li. Study on Classification and Characteristics of Source Address Spoofing Attacks in the Internet. In Proceedings of the 2010 Ninth International Conference on Networks, ICN '10. IEEE Computer Society, Washington, DC, USA, 2010 226–230.

[28] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms* 55, (2005) 58–75.

[29] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* 13, (1970) 422–426.

[30] http://sites.google.com/site/countminsketch/home/expositions. count-min sketch ampersand its applications.

[31] A. Chakrabarti. CS85: Data Stream Algorithms Lecture Notes, Fall 2009.

[32] J. Misra and D. Gries. Finding Repeated Elements. *Sci. Comput. Program.* 2, (1982) 143–152.