ACTIVE PACKET FILTERING AGAINST DDoS ATTACK

by

Tina Seifpoor

B.S., Electrical Engineering-Electronics, Islamic Azad University of Lahijan-Iran, 2007

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Electrical and Electronics Engineering
Boğaziçi University
2014

ACTIVE PACKET FILTERING AGAINST DDoS ATTACK

APPROVED BY:

Prof. Emin Anarim (Thesis Supervisor)	....
Assoc. Prof. Mehmet Akar	....
Prof. Fatih Alagöz	....

DATE OF APPROVAL: 03.01.2014

ACKNOWLEDGEMENTS

Above all, I would like to acknowledge help and support of my principal advisor Prof. Emin Anarim, not to mention his patients throughout the thesis and my studying. I am also extremely grateful to Prof. Bülent Sankur for both academic and encouraging supports and Prof. Güneş Karabulut Kurt for her good advices.

I would like to give my special thanks to my friends, specially my classmate and co-worker Ramin Fadaei Fouladi, who helped me literally all through my thesis. I am also grateful to Mrs. Derya Erhan, for her technical and academical cooperation.

This thesis would not have been possible without the help, support and love of my friends and family, my brother and specially my parents, Mohammad and Nahid Seifpoor who gave their gratis support in all possible aspects, as always, for which my words of gratitude does not suffice. Last, but by no means least, I thank my dear uncle, Ali Nouri, whom I am indebted this thesis to, cause I would have never started it without his encouragements.

ABSTRACT

ACTIVE PACKET FILTERING AGAINST DDoS ATTACK

Distributed Denial of Service (DDoS) attacks are one of the dominant and persistent threats to the security of the Internet nowadays. The aim of these attacks are mainly resource or the bandwidth consumption with enormous number of normal packets. Their target are at layer three or four of the network which are network and transport layers, where distinguishing a normal packet from a malicious one is an arduous task. However, these are not the only precarious aspects of DDoS attacks. A DDoS attacker may easily spoof its source IP address, to hide the origins of the attack. Therefore, developing a distributed defense filtering strategy which can efficiently detect and drop attack packets with the least possible false negative probability is crucial. In this thesis, we propose an incorporated filtering scheme in victim host and edge routers, which detects and drops the illegitimate packets while mitigating the huge amount of data coming toward the victim in edge routers. First, a novel anomaly detection based on feature statistical behavior and payload characteristics of normal and attack traffic is proposed. In the second step, a host-based filtering strategy that detects spoofed packets with a combination of IP history based and hops counting filters, is applied in victim side by means of an advanced matrix bloom filter. Along with this filter, the defense and availability of the service on the target is guaranteed by turning off several edge routers by optimization system. This optimization, selects edge routers to be turned off for the good throughput to reach to the victim via two optimization algorithms, (i) Genetic evolutionary algorithm and (ii) linear programming algorithm.

ÖZET

DDoS ATAKLARINA KARŞI AKTİF PAKET FİLTRELEME

Dağıtık servis engelleme (DDoS) saldırıları Internet güvenliğini tehdit eden ve en çok karşılaşılan saldırılardan biridir. Bu saldırılarda amaç, kaynak ve bant genişliğini, yüksek sayıda normal paket göndererek tüketmektir. Saldırılarda, kötücül olan veya olmayan ulaşımın ayırt edilmesi güç olan, taşıma ve ağ katmanları hedef alınır. Bunların yanında DDoS saldırılarını ciddileştiren başka mekanizmalar da mevcuttur. Bir DDoS saldırganı, saldırının kaynağını maskelemek için sahte bir adresi kaynak adresi olarak gösterebilir. Bu sebeple, DDoS paketlerini en az sezim hatası ile verimli bir şekilde tespit edip engelleyecek filtrelerin geliştirilmesi çok önemlidir. Bu tezde, hedef yönlendiricilere gelen büyük miktardaki veriyi azaltarak, saldırı paketlerini tespit edip düşüren, uç yönlendiriciler için bir filtreleme yöntemi önerilmiştir. Öncelikle, yararlı yük karakteristiği ve ağ trafik özniteliklerinin istatistiksel davranışlarına dayanan yeni bir anormallik tespiti önerilmiştir. İkinci bir adım olarak, uç-tabanlı filtreleme strajesi ile geliştirilmiş bir bloom filtresi, hedef tarafında kullanılmıştır. Bu filtre, sahte IP paketlerinin tespiti için IP geçmişi ve hoplama sayısı değerlerini kullanmaktadır. Bu filtrenin yanısıra, hedefin Internet kullanılabilirliği ve güvenliği, eniyileme sistemi yardımıyla uç yönlendiricilerinin bir veya birkaçı kapatılarak sağlanır. Bu eniyileme, hedefe istenilen paketlerin ulaşması için kapatılacak uç yönlendiricileri, iki eniyileme algoritması ile seçer, (i) Genetik evrimsel algoritması ve (ii) doğrusal programlama algoritması.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS/ABBREVIATIONS

ANP	Average Number of Packets per flow
CBF	Counting Bloom Filter
CFN	Correlative Flow number
CPU	Central Processing Unit
CRAM	Card Random Access Memory
DDoS	Distributed Denial of Service
DoS	Denial of Service
FN	Flow Number
GA	Genetic Algorithm
GT	Good Throughput
HIP	History-based IP Filter
IAD	IP address Database
ICMP	Internet Control Message Protocol
IP	Internet Protocol
IPv4	Internet Protocol version 4
ISP	Internet Service Provider
LP	Linear Programming
ODGS	One Direction flow Generating Speed
OS	Operating System
PCF	Proportion of Correlative flows to number of Flows
RAM	Random Access memory
TCP	Transmission Control Protocol
TTL	Time To Travel
UDP	User Datagram Protocol

1. INTRODUCTION

The Internet, with over billion users today, has revolutionized the computer and communications world into an unprecedented integration of online servicing in academic, military, government, business and commercial fields both locally and globally. Nevertheless, the original Internet [1] was designed for scientific purposes, not as a commercial environment. The policies were based on openness and resilience and security generally was built on mutual respect and honor which was appropriate in the time but no more applicable with today's applications of Internet.

By commercialization of Internet [2] in the early 1990s, the Internet has become a popular target to attacks. With the growth in security incidents, the vulnerability of network infrastructure to failures caused malicious attacks become ever more apparent. Considering the expenses of recovering from such failures which often costs more than prevention, an automated, robust and easy operable defense system is of significant concern. Malicious users are often able to steal information or halt normal computer operation, with motives ranging from financial or political objective to personal hostility.

1.1. DoS attacks

Any attack which leads to unavailability on an online service is considered as a Denial of Service (DoS) attack [3]. The targets (victim) in a DoS attack, depending on the goal of the attack, might be a specific computer, the whole network, a service or even a port on the victim system, firewall or to any other system ability. The attack may be comprised of a single packet exploiting software bugs in a server Operating System (OS) and insert a malformed packet to subvert the legitimate communications. Such attacks are known as Vulnerability attacks or Semantic attacks. On the other hand, it can involve sending traffic streams with an immense number of packets higher than target's provisioned to manage, that congests the target's server or network and results in denial-of-service to the victim's legitimate clients. These attacks are called as flooding or Brute force attacks. With the success of these attacks, the system may fail to reply to any request comparing to normal state, and consequently the legitimate client's access to some expected services might be denied or limited. Vulnerability attacks are usually performed by a few packets with specific type or content to elicit the susceptibility of the victims thus they can be handled simply by patching the corresponding bugs of the protocol or application and detecting the special types of packets and drop them. Nevertheless flooding attacks are not easy to analyze and counter; they overwhelm the victim's resources by extreme volumes of data while the packets can be any type with any content limiting the capability of any defense mechanism toward such attack. In DoS attacks, although the attacker rarely uses her own machine to execute the attack, it usually involves only one attacking machine and Internet connection. Figure 1.1 depicts a simple DoS attack from three attackers. Although it is possible that different attackers perform the attack on one victim in parallel, each attacker use only one machine to employ the attack.

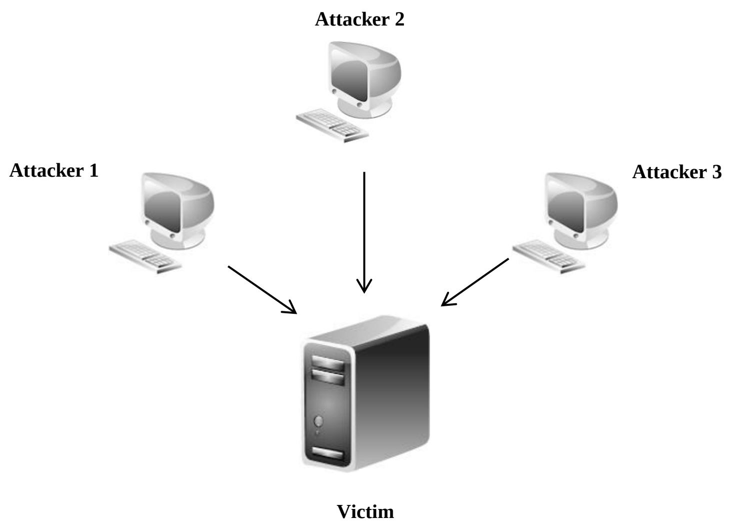


Figure 1.1. A simple DoS attack with three parallel attackers.

1.2. DDoS attacks

Distributed Denial-of-Service attacks [3] are simply DoS attacks performed from multiple depraved machines by the attacker, generate and send malicious packets toward a victim simultaneously in an attempt to inflict damage on victim by exhausting its resources which can be bandwidth, CPU, RAM, CRAM, and all parts related to compositional ability of the system. A DDoS attack is typically implemented via a network of two series of compromised machine infected by malicious codes as Master zombies and Slave zombies. A small DDoS attack scenario of an attacker with Master and Slave zombies is shown in Figure 1.2. These hosts are vulnerable systems found by an automated scanning and then attack tools installed on them. The attacker sends the attack command to Masters which are hibernated. Then, the Masters trigger Slave zombies, ordering them to forward large amount of useless data to the target to deplete it's bandwidth or resources.

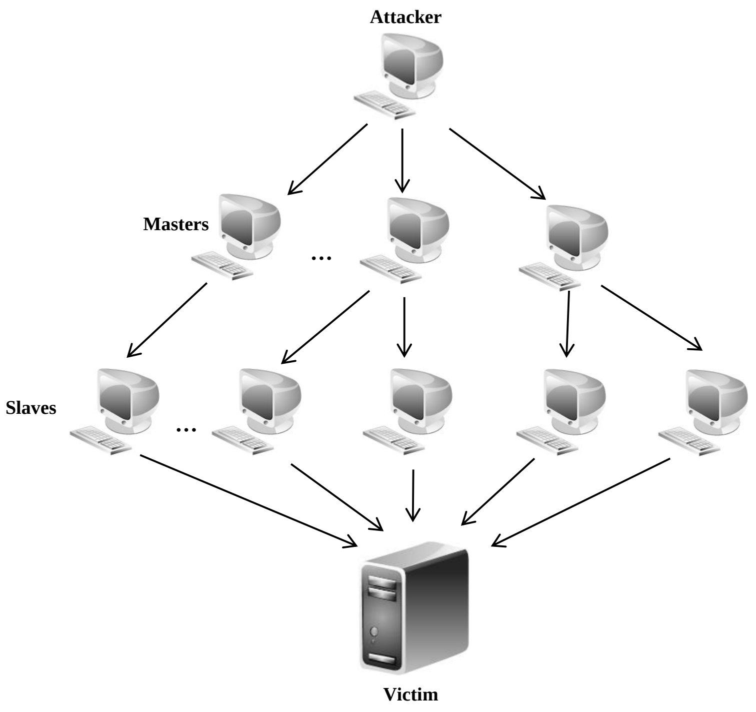


Figure 1.2. A simple DDoS attack scenario with Master and Slave hosts.

1.3. Method of application

The Process of a DDoS attack consists of four major steps: scanning to recruit vulnerable systems, compromising the machines with malicious codes known as attack tools, deployment and propagation of the attack from compromised Handlers (Masters) and Agents (Slaves). In the earlier days of DDoS technology development, all phases (scanning, compromise, deployment and propagation) would be done manually by the attacker. Over time, more and more automation has been brought into each of these steps which give the possibility of launching such attack to any normal user.

The first step which is the process of recruiting of the attack network (army) is usually performed automatically; the attacker downloads a scanning tool and deploys it from other compromised machines under its command namely Masters. There are plenty of techniques for scanning tools to find and infect vulnerable systems in the network. These machines are scanned by scan probes to find security glitches. The infection process with the attack software will be done through these security holes automatically. Each machine will be used for further deployment of new slaves after first infection phase. In the second phase, when the vulnerable system are found, they will become one of the zombies and attacker will give the instruction of copying and installing attack toolkit on selected vulnerable machines by Masters. During and after attack attempt, all logs showing malicious activity are erased to destroy evidence. Furthermore attack scripts under system directories are hidden with obscure, non-suspicious given names so they will not attract a user's attention. Sometimes they fix the bugs used for the exploit, to prevent other hackers from subverting the machine. Afterwards, the attacker uses masters to control the attack by specifying desired type of attack, the IP address of the victim and the time and duration of attack and they, subsequently, command the slaves to launch the attack with victim IP address as their packet's destination address and other commanded details. Further detailed information on operation can be found in [4] and [5].

1.4. Drawbacks of defense against DDoS attacks

DDoS attacks are inconveniently difficult to detect and complete prevention or defense against them is extremely hard to achieve. According to [6] and [7] there are two major drawbacks concerning defense against DDoS attack: One is the insecure design of Internet architecture in mind and second is the characteristics of DDoS attack.

1.4.1. Insecurity of Internet architecture

The Internet was designed for easy access and openness, not security, and it was indeed successful in its goal at the start. It provides its users with fast and effortless communication mechanisms, accomplished in higher-level protocols to ensure reliable quality of service. Because of the distributed characteristic of Internet, there is no possibility that common policies could be applied among its users. Such design, though successful in its goal sense, costs in several security issues such as proneness to DDoS. First of all, the connotation of security on the Internet is eminently interdependent. It means, even a victim with a high level of security is susceptible to a DDoS attack through insecurity of networks and host connected to it. Limitation in resources is another problem which is a target of depletion for DDoS attackers. Furthermore, accessible high bandwidth presents a good field for malicious users to misuse it as a path to deliver excessive data to victim.

1.4.2. DDoS attack characteristics

The premier obstacle against DDoS attack is its similarity with normal traffic patterns. Attack traffic is usually legitimate packets forwarded from slaves in high quantities, thus distinguishing these packets among normal client packets is almost impossible in packet basis and further classification is essential in detecting a DDoS attack. The distributed nature of the attack is another problem causing difficulties in both detection and mitigation of such attacks. Attacking packets are generated from numerous illegitimate machines distributed all over the Internet while it is mingled in edge routers of the victim. The defense system must control a large portion of the total attack to alleviate the denial-of-service effect on the victim. This indicates that a system must either be a single-point system located near the victim or a distributed system whose defense nodes cover a significant portion of the Internet. Moreover, the attackers usually alter the source IP address to masquerade themselves as legitimate users, which makes the detection of attack packets from normal ones even more troublesome. This method, known as “IP address spoofing”, has been used in most of preceded DDoS attacks which makes the task of trace back almost impossible besides making the detection difficult.

1.5. Defenses against DDoS attacks

To negate DDoS attacks, there are two distinct approaches: router-based and victim-based. Router based approaches are embedded in routers structure far from the victim to detect and mitigate the attack in early stages, while the victim based defense systems enhance the stability and resistance of Internet servers close to potential victims. Although it is very efficient to detect attack in early stages and close to their sources but these methods are usually costly and needs a distributed filtering deployment in upriver routers and network. The victim-based approaches, on the other hand, are immediately employable. Detection is much easier since the victim can observe the behavior of the normal traffic closely and distinguish any anomalies. Furthermore since the victims suffer the largest damage from DDoS attacks, it is more likely for them to have stronger motivation to invest and set up defense mechanisms compared to network service providers [3]. The defense mechanism should detect and discard spoofed traffic.

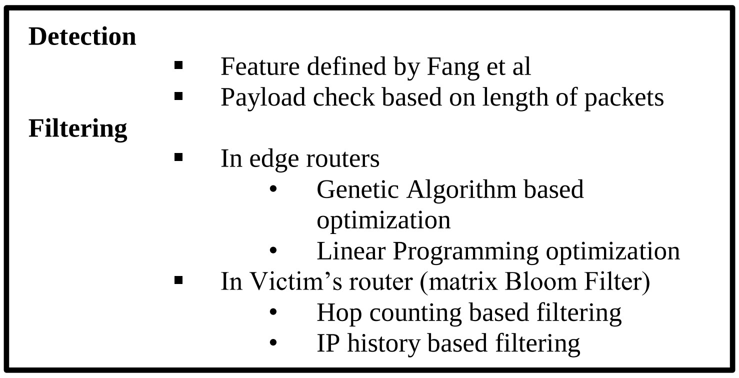
In this thesis, we propose a victim-based filtering strategy which detects spoofed packets and filters them by IP history based [8] and hop counting [9] distinguish filters. First, detection is done by selecting specific features extracted and calculated from the behavior of traffic in host's server router to calculate the probability of occurrence of an attack. This Section determines the state (alert or active) of the filter. When there is no attack, the database of the filtering system is trained which is explained in detail in Chapter 3. Chapter 4 contains the details and experimental results of the filtering in victim's end. When an attack is detected, filter starts to compare the source IP and Hop count value (Hc) of the packets with the data base (white list), if it belongs to the list the packet may pass, if not it will be dropped. While a router in victim side can detect and filter attack packets, the availability of the service and even defense mechanism is not guaranteed. To overcome this problem we apply some optimization on the good traffic coming toward the victim router by turning off some of the edge routers. This optimization system is applied by two methods: Genetic algorithm and linear programming which are evaluated in Chapter 5.

2. METHODOLOGY

As a consequence of remarkable damage that DoS and DDoS has cost to networks and Internet, a great deal of research has been performed on detecting and finding a suitable defense strategy against them. The large scale of the attack and usage of IP spoofing in different levels, makes trace back almost impossible and since DDoS tools use legitimate packets for attack, characterization of attack is a hard confusing labor which respectively makes detection difficult. The distributed nature of attack needs distributed cooperation between administrative domains which is costly to achieve. Thus a thorough filtering system which meets the security requirements of network while alleviating the drawbacks is necessary. According to [10] general class of filters against DDoS attacks are generally consist of three parts:

- Detection of attack
- Identification of the attack source
- Filtering of suspicious packets

However, since the second part which is known as trace-back is a cumbersome task and out of the scope of this thesis, our filtering scheme consists of the detection and filtering parts. Figure 2.1 represents the outline of our filtering scheme. In detection section, a combined method of feature selection introduced in [11] and a payload analysis is used to reduce the false alarm in detection of the DDoS attack. In the payload checking part, the characteristics of the length of packets in an interval of time is modeled for different protocols. The incoming packets are compared to those models for abnormalities based on a distance metric and threshold. Filtering parts activate when a DDoS attack is detected and Filtering policies are applied on two points of victim's network, one in the victim's router and in edge routers. Two optimization methods, Genetic algorithm and Linear programming, are performed and compared to enhance the good traffic flow toward the victim avoiding a crash down of victim server in edge routers. Furthermore, an improved adaptive Bloom based Filter, drops the packets in victim router by considering two features of packet's information i.e. the Hop Count value and source IP address.



Detection

- Feature defined by Fang et al
- Payload check based on length of packets

Filtering

- In edge routers
    - Genetic Algorithm based optimization
    - Linear Programming optimization
- In Victim's router (matrix Bloom Filter)
    - Hop counting based filtering
    - IP history based filtering

Figure 2.1. Principal outline of our filtering scheme against DDoS attack.

3. DETECTION AND FEATURE SELECTION

Feature selection is the process of extracting a subset of features from information in the IP heathers of packets, analyzing and comparing them to the characteristic of such features in normal traffic to detect the possibility of existence of an attack. It is important to select a correct and optimal subset of features which have the advantage of processing less data while obtaining more accuracy on attack detection. There are two types of anomaly based detections. One is based upon a set of rules relying on human expertise and manual intervention to define a normal behavior while in the other one, the properties of a normal behavior is learned automatically through a training of the system. In this chapter, a detection system based on feature characteristic of traffic is explained. All the data in this section is extracted from DeterLab [12] and the topology of the test-bed of our experiment is displayed in Figure 3.1. The network comprises of 10 clients, 5 routers, and one server. The clients include both attackers and legitimate users. Clients n_8, n_9 and n_{17} contribute in an attack to the victim, and the rest of the computers are normal users. we gathered data from n_{10} which is the last router connected to the victim. All the codes and programs are written in MATLAB-R2010a [13].

3.1. Flow based feature selection and detection

Feng *et al.* [11] introduced statistical features based on five keys (Source IP, Destination IP, Source and Destination ports and protocol) and the concept of Micro and Macro flows. They defined a Micro-Flow as a group of packets with the same time interval of Internet where all packets have the same specific characteristics in mentioned key features while a Macro-Flow is all the packets belonging to one time interval. There is no necessity that the packets would be in sequential order to be in the same Flow but they should be in same interval. To have a more comprehensive notion of the concept of Micro-Flows, Table 3.1 depicts an example of a short time interval data of packets with distinct Micro-Flows. Packet 1 and 2 are considered as one Flow and Packet No. 3 with packets 5 and 6 is another Flow. Each flow consists

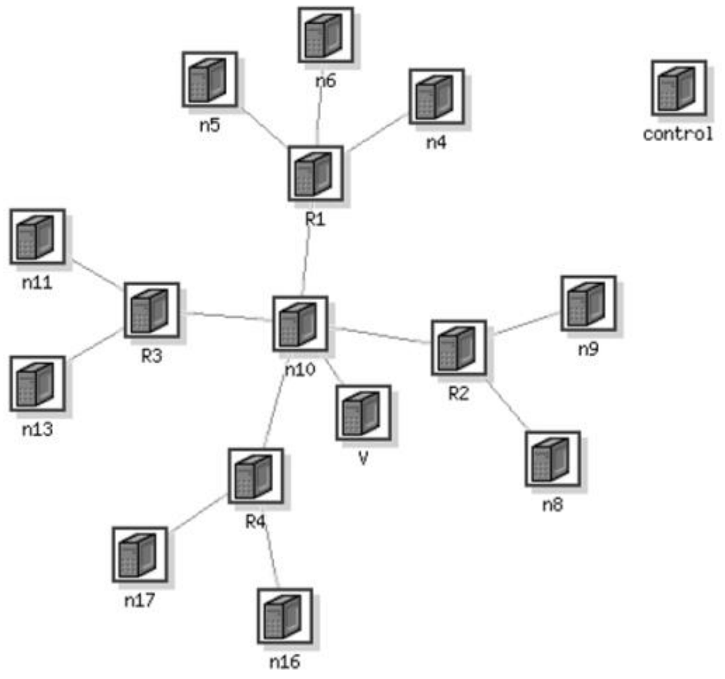


Figure 3.1. Topology of test-bed in DeterLab with three attackers n_8, n_9 and n_{17}.

of the packets with similar keys that for less computing time and effort we use three most reliable features (source IP, Destination IP, and protocol).

Table 3.1. Micro-Flows in an interval of time of data information.

No.	Time	Source IP	Destination IP	Protocol	Length
1	0	10.1.11.2	10.1.1.2	TCP	68
2	1.10E-05	10.1.11.2	10.1.1.2	TCP	68
3	0.001004	192.168.1.184	192.168.253.1	TCP	204
4	0.001478	192.168.253.1	192.168.1.184	STUN	192
5	0.001501	192.168.1.184	192.168.253.1	TCP	68
6	0.001573	192.168.1.184	192.168.253.1	TCP	216
7	0.002976	10.1.14.2	10.1.1.2	TCP	1516
8	0.002987	10.1.14.2	10.1.1.2	TCP	1516
9	0.007193	Source IP	10.1.14.2	TCP	80
10	0.0072	Source IP	10.1.14.2	TCP	80

3.1.1. Average number of packet in each Micro flow

Average number of packet in each Micro flow [11] is the first feature which differs significantly when an attack is going on. During attack the number of flows in an interval becomes close to the number of packets. In another words, flows with same keys would consist of just packet. This is mostly because by the means of the IP spoofing. The source IP addresses change so frequent and there would be fewer packets with similar IP addresses. Equation 3.1 is the mathematical formulate of this feature:

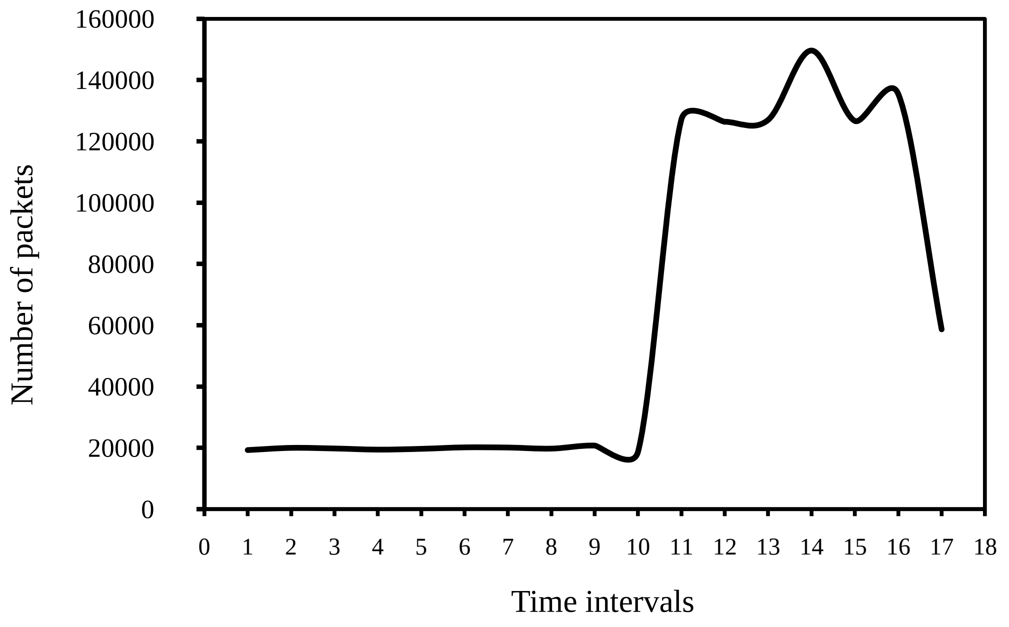
$$\text{ANP} = \frac{\text{Number of Packets in a Macro Flow}}{\text{Number of Micro Flows}} \tag{3.1}$$

This feature helps us to distinguish between a flash crowd and attack. Although both the number of flows and the number of packets in each interval rise when an attack is going on, growth rate of number of flows is much more than the increase in number of packets. Figure 3.2 gives a preview of these to features over time intervals of 30 seconds where attack starts at 11^{th} time interval.

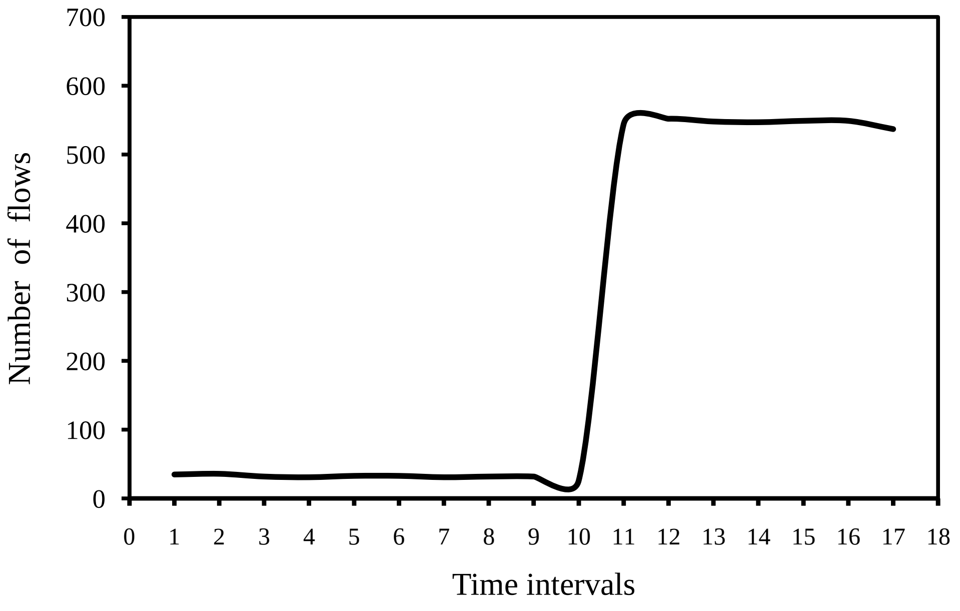
An experimental comparison of ANP (Average Number of Packets in each flow) between normal traffic and DDoS traffic is depicted in Figure 3.3 for 7 different intervals of time in normal and attack traffic.

3.1.2. Correlative packet flows over number of flows

Proportion of number of correlative packet flows (CFN) to number of flows (FN) in each interval of time [11] is another feature which is useful to detect the attack. This is an index of correlativity of the source IPs and Destination IPs and the ability of the victim to reply to receiving packets. During attack, since the source IP addresses are spoofed, even though the victim is still capable of answering attacking packet's requests, the replying packets cannot reach the attacking machines. To extract this feature, we need to find those flow number whom are answered by the victim. For more illustration, in Table 3.1 two correlative flows can be seen where first flow is packet 7 and 8, and the other flow is consists of packet 9 and 10. The second flow packets are



(a) Number of packets per interval



(b) Number of flows per interval

Figure 3.2. Number of packets versus number of flows in each time interval.

sent as an answer to the first flow source IP. The Proportion of CFN to FN is defined as :

$$\text{PCF} = \frac{\text{Number of Correlative flows}}{\text{Number of Micro Flows}} \tag{3.2}$$

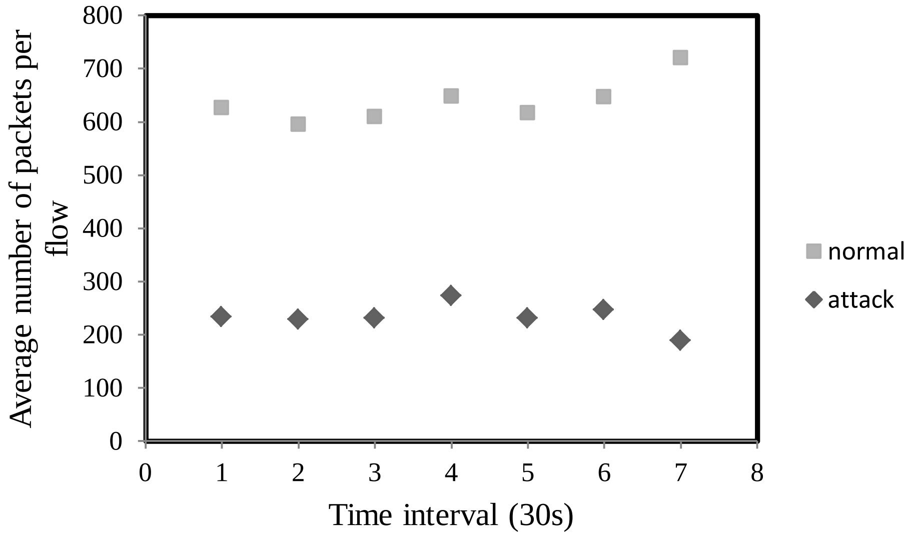


Figure 3.3. Average number of packets per Micro flow in attack and normal traffic.

To find the number of correlative flows in each interval, we defined two vectors for each interval; one with Source IP addresses and Destination addresses in sequence, and for the second one the place of the source and destination IPs were changed. Afterward, A Bubble Sorting Algorithm [14] was applied to find the similar elements in the first vector with the whole second vector. The similar elements were sorted and deleted from the list after counting. This process goes on recursively until all elements of vectors are analyzed to find CFN.

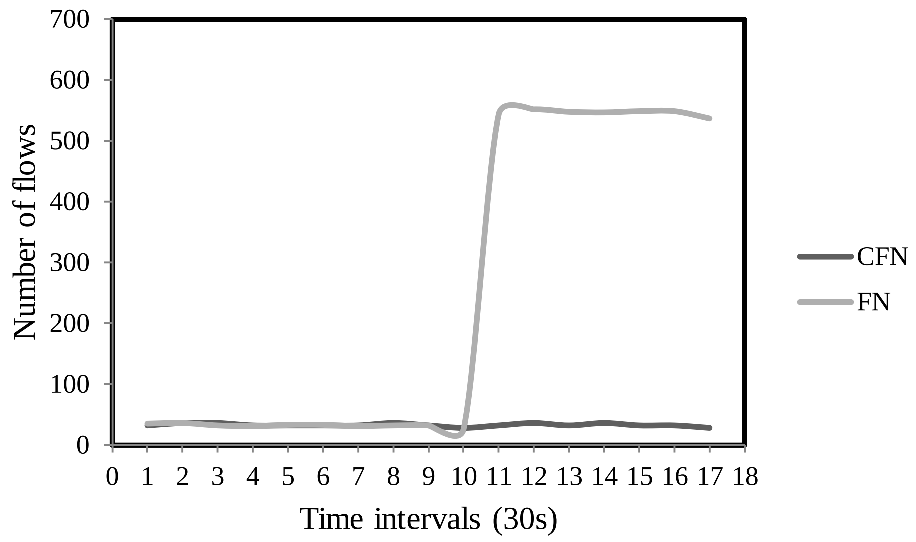


Figure 3.4. A comparison of CFN and FN where attack starts at 11th till 17th time interval.

As it can be seen in Figure 3.4, when an attack is going on, the change in number of correlative flows is negligible while the number of flows rises dramatically. Consequently, PCF in attack times decreases which can be used as an alarm for attack detection. Figure 3.5 is the result of PCF in each interval of time in both attack and normal traffic.

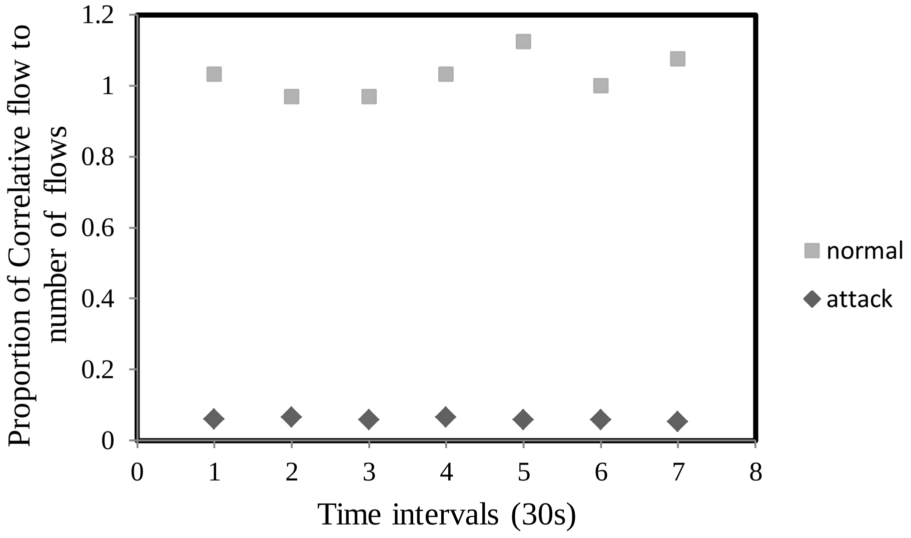


Figure 3.5. PCF in each interval of time for both attack and normal traffic.

3.1.3. One direction flow generating speed

One Direction Flow Generating Speed (ODGS) [11] is the last Micro flow based feature which is defined as in Equation 3.3.

$$\text{ODGS} = \frac{\text{Number of Flow} - \text{Number of correlative Flow}}{\text{Time interval duration}} \tag{3.3}$$

Flow generating speed increases considerably in attack or flash crowd situations. We use ODGS which is an indicator of number of one-directional flows generated in an interval of time traffic which is a small value for normal traffic to segregate these two different conditions. As it is obvious in Figure 3.6 This value increases significantly during a DDoS attack.

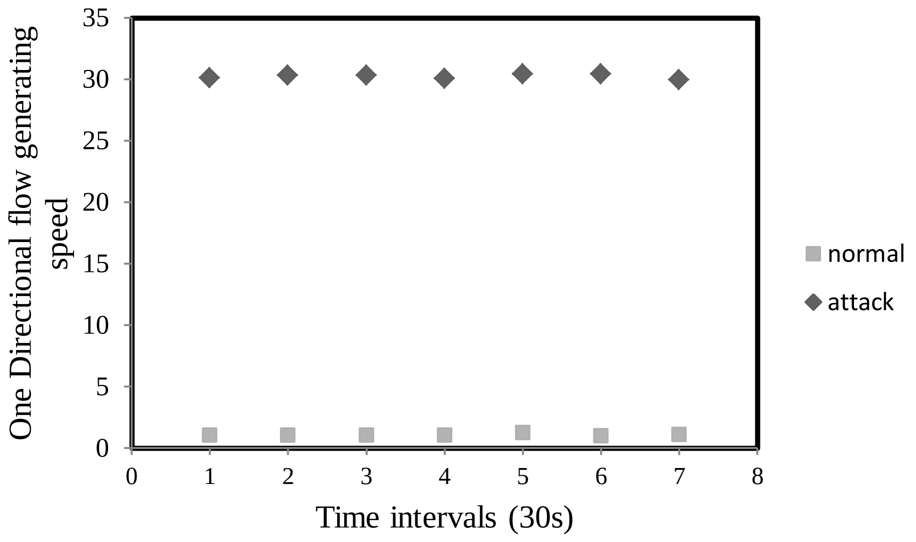


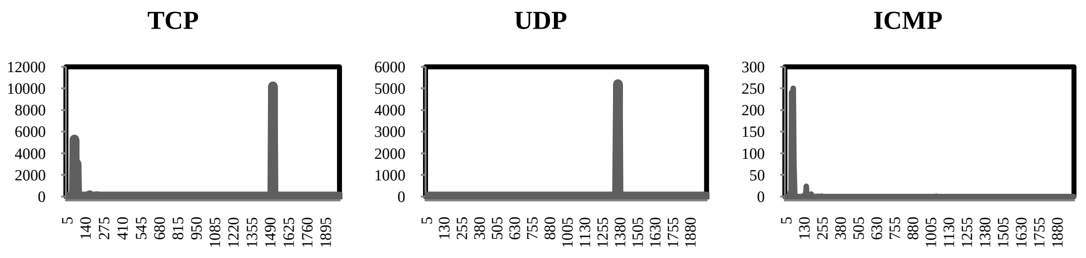
Figure 3.6. A comparison of ODGS in normal and attack traffic.

3.2. Payload length based detection

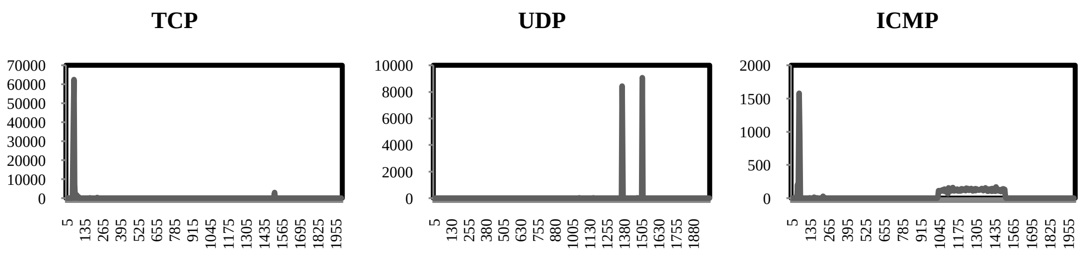
The described features are defined based on the Micro flows which give us a great measure of statistic attitudes of the traffic passing throw the routers close to the server that Intrusion Detection System is applied on. However the payload of the packets in Macro flow scale is another valuable feature. Attacking packet's contents are usually left empty or only filled with small futile bytes. As a consequent, the number of abnormal packets in payload sense increases during the attack. To achieve a self-regulating detection system, we propose a detector which analyze and model the length of payload of normal traffic in an interval of time. More specifically, The system first learns a profile of the expected payload length of packets delivered to a server during an interval of normal operation of the network where each model is the frequency distribution of packets length in a specific time interval. This modeling method is applied for three protocols; TCP, UDP and ICMP separately for more consistency. According to the characteristic of DDoS attack, the range of length distribution differs from normal traffic. Therefore, the incoming traffic is captured by the detector and modeled in each interval to be compared with the corresponding model. A distance metric and a threshold is employed to verify the propriety of flow of data in each interval of time.

3.2.1. Payload length model

The minimum length of a packet is 20B (20 bytes of header plus 0 byte data) and the maximum value is around 64KB. Since there is no specific limitation for the length of the payload, to detect abnormality in the behavior of the traffic without manual intervention a modeling structure is convenient. We propose to model the length of payload frequency distribution in an interval of time and then use a comparison metric to detect an attack. Figure 3.7 represents models of three protocol (TCP, UDP, ICMP) in both normal and attack situations. Each model is in fact the distribution of packets regarding their length for that specific protocol.



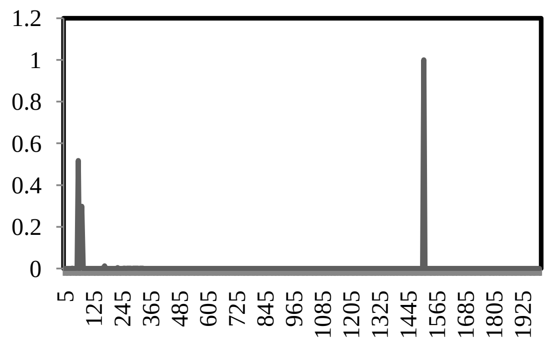
(a) Normal traffic payload length histograms



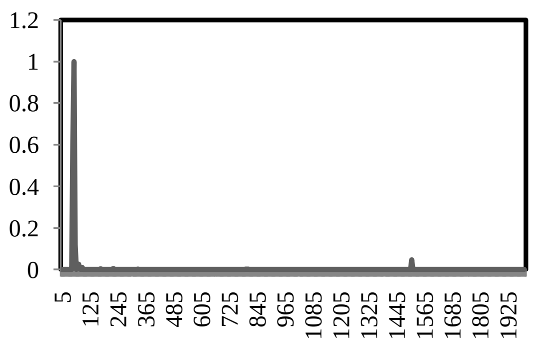
(b) Attack traffic payload length histograms

Figure 3.7. Payload length distribution of packets in an 30s time interval for TCP, UDP and ICMP protocols.

The number of packets increases dramatically during a DDoS attack which makes the comparison of its distribution with normal mostly based on the number of packets instead of its distribution. For example Euclidean distance may give us a remarkable difference between two histograms but the divergence is greatly dependent on the difference of number of packets in one bin of the histogram instead of the distribution of the histogram. To avoid encountering this problem, a normalization is utilized to adjust the distribution for comparison. In Figure 3.8 normalized models of TCP protocol is represented where Figure 3.10a and Figure 3.10b are the normal and attack situations respectively .



(a) TCP normal traffic



(b) TCP attack traffic

Figure 3.8. Normalized payload length distribution of packets in an 30s time interval for TCP protocol.

3.2.2. Distance metric

There are several measures to compare histograms which can be categorized in two major classes. Bin to Bin dissimilarity measures which assume that histogram bins are aligned and they only compare contents of corresponding histograms bins. The second class is Cross bin distances that also compare non-corresponding bins of histograms by the means of ground distance, defined as the distance between the representative features for different bins of histogram. The first group of distances are sensitive to bins boundaries. Furthermore, cross bin distances are more appropriate for the purpose of finding and comparing the characteristic of length of packet generation in a DDoS attack. Quadratic Form distance [15] is one of cross bin class measures which is defined as

$$\begin{aligned} &D_A = \sqrt{(X-Y)^T A (X-Y)} \\ &X \in R^n \\ &Y \in R^m \\ &A \in R^{n\times m} \end{aligned} \tag{3.4}$$

where X and Y are the histograms with $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m$ bins respectively and $A = [a_{ij}]$ is a positive-definite bin similarity matrix where a_{ij} denotes the similarity between bin i and j. To make a proper metric from this distance, we consider $a_{ij} = 1 - (d_{ij}/d_{max})$, where d_{ij} is the Euclidean distance between i and j bins of the histograms and d_{max} is the maximum value of d_{ij} [15].

To accomplish the detection, the incoming traffic is captured by the detector in every 30s (time interval)and analyzed the packet length distribution for each protocol. In our experiment each histogram is consisting of 400 bins and A is 400×400 matrix. Then, the detector compares the distribution with corresponding profiles via the quadratic distance mentioned in Equation 3.4. A threshold is defined for each protocol based on empirical results. Figure 3.9 displays a plot of the distance values of the normal traffic against the attack traffic in each protocol from a normal model histogram. As illustrated, the difference between normal intervals with normal models are remarkably small (around 0.00001) and by choosing a proper threshold, detection is prosecutable.

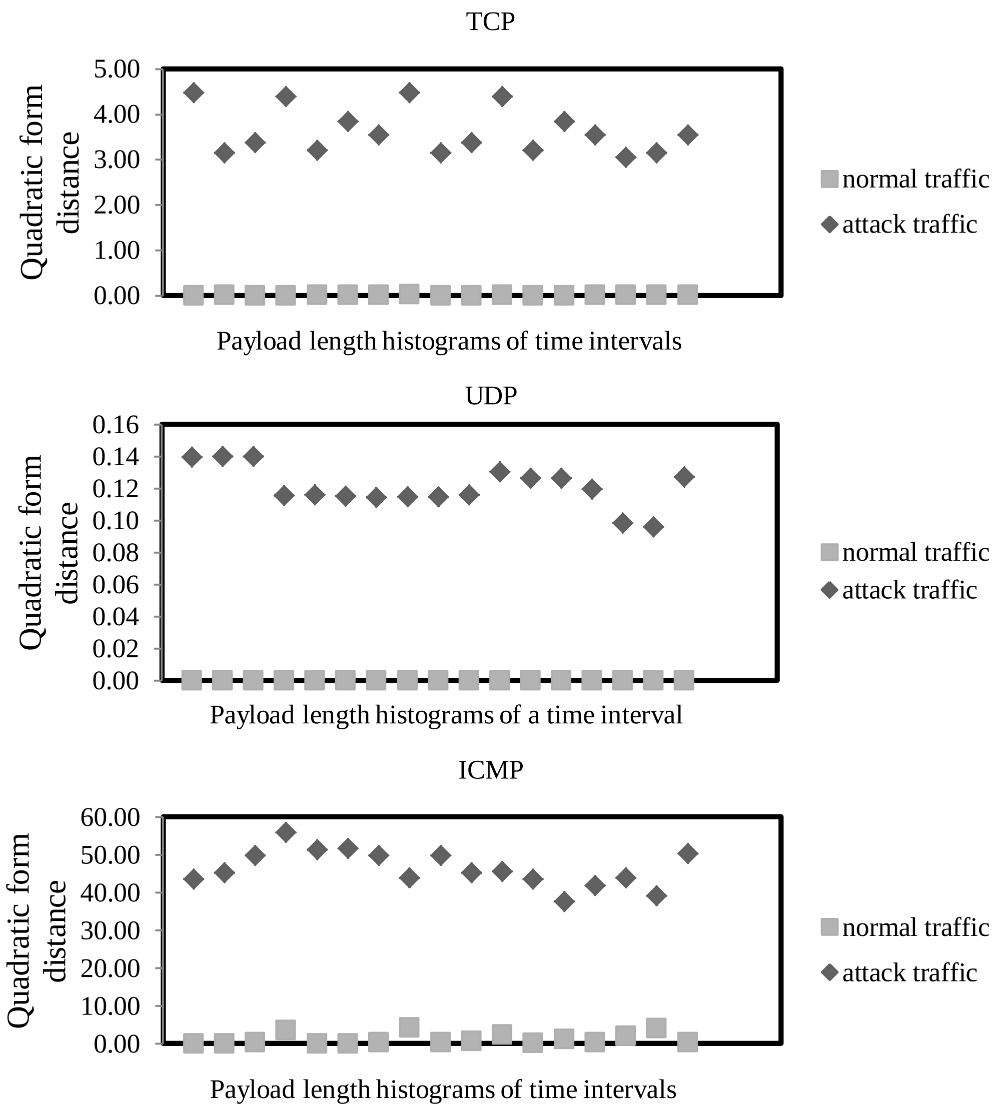


Figure 3.9. Computed distance of normal and attack traffic with a normal model histogram.

4. FILTERING SCHEME IN VICTIM'S ROUTER

Defense mechanism and filtering system majority have been designed on victim end. There are two main reasons for this matter: first, it is easier to model and inspect traffic in victim side. Second, victim suffers the largest damage from a DDoS attack and therefore, it is more motivated to invest in a defense system in comparison of other side of network. In this chapter, a filtering mechanism in victim part is explained including background information in regard with the filtering system.

4.1. IP history based filtering

The source IP addresses of normal traffic of a specific network can be used to make a white list table of IP addresses that each of which represents a regular legitimate client of the network. Around 82.9 percent of all IP addresses in observed flash crowds have been seen at least once before [16]. Therefore, the IP addresses that had been seen with a router or web server can be considered as a IP history database to detect attack traffic. Peng *et al.* [8] proposed to save all the IP addresses of the previous safe network connections to make an IP address database namely an IP address white list. They designed their detection mechanism based on this hypothesis, called History-based IP Filtering (HIF) for the victim router to filter incoming packets according to a built IP address white list. When network experiences a blockage of traffic, it discards packets whose source IP addresses have not restored in IP address database yet. In their method, a frequent IP address is defined under two specific criteria. The number of days which an IP is observed is the first criterion. Normally, clients use the Internet at regular times. Thus, an IP address can be considered to be frequent based on the number of days it has appeared in the network which is the basic idea behind rule1. More specifically, $P_1(d)$ represents the collection of IP addresses that each has been recorded in at least d days. The second rule is the number of packets per IP addresses. In normal states, frequent IP addresses send a certain number of packets to the network. $P_2(u)$ represents a group of IP addresses that the server received at least u packets from them. By combining all these two rules, which can be shown as

$F_c = P_1(d) \cap P_2(u)$ where F_c is the collection of all IP addresses under these two rules an accurate and efficient IP address database (IAD) can be made.

4.2. Hop-count based filtering

Although IP History based Filtering gives acceptable accuracy in dropping illegitimate packets with spoofed source IP addresses, if the attacker use IP addresses same as the legitimate ones of the network, false negative may arise dramatically. Hop-Count Filtering [9] can be a solution to such inefficiency. There is a value in IP header of each packet named Time-to-Live (TTL). Under the Internet Protocol, TTL is an $8-bit$ field. In the IPv4 header, TTL is the 9^{th} octet of 20 and in the IPv6 header, it is the 8^{th} octet of 40. The maximum TTL value is 255 which is the maximum value of a single octet. This value is decreased by one when the packet passes through an intermediate. When a packet reaches its destination, the number of intermediate hops ,known as Hop count (Hc), can be obtained by subtracting final TTL value from the initial TTL. An attacker may alter everything in an IP header but the number of hops that a packet passes to reach its destination can not be changed. Therefor, Hc can be used as a good measure for detecting spoofed IPs. The problem in hop-count computation is that initial TTL values varies for different operating systems (OS) and the destination router can only check the final TTL value. To tackle this problem, they considered alternative static initial TTL values for each IP address under consideration that most Operating systems use only a few selected initial TTL values: $30, 32, 60, 64, 128$ and 255. They selected the smallest initial value in the set that was larger than its final TTL as the initial TTL value of a packet. In the cases of $\{30, 32\}$, $\{60, 64\}$, and $\{32, 60\}$ they computed a hop-count value for each of the possible initial TTL values, and if there was a match with one of the two possible hop-counts the packet was considered as legitimate. In their work, the source IP address and the final TTL value of each IP packet is extracted. After deriving the initial TTL value, the hop count value is inferred by subtracts the final TTL value from initial value. The source IP address is then used as the index to find the correct hop-count for this IP address. If the computed hop-count matches the saved hop-count in the table, the packet is detected as a normal packet and if not, then the packet is spoofed and will be discarded. There is a small possibility that spoofed IP address has the same hop-count as the one from a zombie to the victim where HCF will not be able to identify the spoofed IP packet. Otherwise, HCF is highly effective in identifying spoofed IP addresses even though the the range of Hc values are limited.

4.3. Filtering scheme with modified Bloom Filter

Although both HIF and HCF may lead to reasonable results by themselves but they are not without inefficiencies. Skilled attackers may conduct attacks with subnet spoofing, using the same IPs as normal traffic which can causes dramatic flaws in the performance of HIF. Furthermore there are spoofed packets with the same TTL values that can pass HCF filter easily. To avoid such predicaments, we propose a new filtering scheme by utilizing the advantages of previous works, stand on the relation of Source IP address and TTL value of normal packets. With our method there would lie a very small probability that a spoofed IP address happen to be in our IP data space with the same hop values. The attacking packets with random IP spoofing would be discarded for their source IP addresses. Moreover, by considering the packets hops the attacking packets with subnet IP spoofing would also be filtered out since their hop values would not accord with normal packet's hop values. To accomplish such purpose, we need a table of information about all legitimate IP addresses with their established hop values regularly appeared in a server. However administering such a table is costly not only due to the huge memory space that it needs, but also in the time which takes to look up in such space. To override this obstacle, we may employ a modified Bloom Filter. Bloom filter has been used in defense against DDoS attack repeatedly e.g. in [17–19]. Nevertheless, even though the memory usage reduced considerably, but still there are many parameters configured manually in most of the methods which is not satisfactory regarding the adaptive model needed in this application field. Usually these filters were deployed in routers with massive amount of data to be saved which may lead to overflowing of the filter without manual supervision. Our filtering model, besides time and memory efficiencies is applied in victim server with bounded information limited to IP address and Hop values. The filter consists of a matrix database which each column is an enhanced Counting Bloom Filter, a look up algorithm with specific hash functioning method and a renewal algorithm to make the filter adaptive. Each part will be described in the following sections in detail.

4.3.1. Basic Bloom filter

Bloom Filter (BF), introduced originally at by Bloom (1970) [20] for the purpose of spell checking, is a vector-structure utilized to represent a set in a space efficient manner in order to check membership queries with some probability of error. In other words, space efficiency is obtained at the expense of small probability of false positives. This probability of error means that a Bloom filter could consider some entries even if they do not belong to the set. However Bloom filters have such space and time advantages over other data structures in data structuring, such as self-balancing binary search trees or hash tables that this false positive can be negligible. Regarding the time efficiency Bloom filters also have the property of independence from number of elements in the set so the time of adding or checking entries is constant. A basic BF is a vector,V, of m bits all initially set to zero to exhibit a set $S = \{x_1, x_2, \cdots, x_n\}$ with n elements. k independent hash functions $h_1, h_2, \cdots, h_k$ are used to map each elements of S into the bit vector and set the locations to 1s. These k hash function are assumed to be uniformly distributed over the range of m which is the size of BF vector. The probability that a certain bit is not set to one by a hash function is:

$$1 - \frac{1}{m} \tag{4.1}$$

Where m is the size of the BF vector. Then since there are k hash functions, the probability of any of them not having set a specific bit to one is given by:

$$(1 - \frac{1}{m})^k. \tag{4.2}$$

When an element is inserted, the probability that a given bit is still zero is:

$$(1 - \frac{1}{m})^{kn}. \tag{4.3}$$

The probability that a certain bit b in V is still zero can be calculated as in Equation 4.4:

$$P_r(0) = (1 - \frac{1}{m})^{kn} \approx e^{\frac{-kn}{m}}. \tag{4.4}$$

Therefore, the false positive probability is the product of the probabilities that k different bits would be all set to ones. The approximation of this probability can be seen in Equation 4.5.

$$P_r(false\_positive) = (1 - P_r(0))^k = (1 - e^{\frac{-kn}{m}})^k \approx e^{k \ln(1 - e^{\frac{-kn}{m}})} \tag{4.5}$$

Considering g as an approximation of the probability of false positive in Equation 4.5, we can calculate the optimize number of hash function k to minimize the probability equation by differentiate it with respect to k .

$$\frac{\partial g}{\partial k} = \ln(1 - e^{\frac{-kn}{m}}) + \frac{kn}{m} \frac{e^{\frac{-kn}{m}}}{1 - e^{\frac{-kn}{m}}} \tag{4.6}$$

Considering Equation 4.6 equal to zero, the optimized k is calculated as:

$$k_{opt} = \frac{m}{n} \ln(2) \approx \frac{9m}{13n}. \tag{4.7}$$

This k guarantees having the minimum possible false positive which is equals to:

$$P_{min}(false\_positive) = (\frac{1}{2})^k = (0.6158)^{\frac{m}{n}}. \tag{4.8}$$

Therefore, the probability of error will remain constant, if the length of a bloom filter linearly increases with the number of elements fed to the filter. The size of the vector m for the desired number of elements n and false positive probability p, is then as:

$$m = -\frac{n \ln p}{(ln2)^2}. \tag{4.9}$$

In practice, a balance should be found between minimization of false positive probability and the time of hash function computations which are inversely proportional. The possible operations in basic BF involves mapping elements to the set, which means making the database of our method, and querying for element membership in the filled vector. Unfortunately basic BF does not support deletion of elements or it results in false negative while having no false negative is the main advantage of BF. However, a number of extensions have been developed on basic BF that also support removal which is explained later as Counting Bloom Filter (see Section 4.3.2).

An element $x \in S$ can be inserted into the filter by setting the bits $h_i(x)$ to one where i is between zero and k. These Allocations can be set to 1 multiple times, but only the first change has an effect for each $h_i(x)$. Figure 4.1, presents a pseudo code for the insertion operation in Algorithm 1. Conversely, as shown in Algorithm 2, For the membership query if $y \in S$, we check if $\forall i, h_i(y) = 1$. If $\exists hi \neq 1$, then$y \notin S$. If $\forall i, h_i = 1$ is true, we can assume $y \in S$ with the mentioned false positive rate.

```
Input element x
Find all hash function $h_1$ to $h_k$
For i=1:k
   If $V_i$== 0
   then $V_i$=1 ;
   else
  do nothing;
end
```

Algorithm 1 : pseudo code for element insertion

```
Input element x
Find all hash function $h_1$ to $h_k$ for x
   If $V_i$== 1
   For all i=1:k
   then $x \in S$;
   else
   $x \notin S$ ;
end
```

Algorithm 2 : pseudo code for element query

Figure 4.1. Bloom filter insertion and query pseudo-codes.

4.3.2. Matrix Bloom Filter

Counting Bloom filter (CBF), introduced by Fan *et al.* [21], are simply standard bloom filter with the ability of counting the inserted elements which also provides the delete operation on a Bloom filter without repeating the filter construction. In counting bloom filters the array positions (buckets) are changed to a n-bit counter instead of a one bit binary value. In other words, Bloom filters are in the family of counting filters with a bucket size of one bit. In the insert operation, the value of the buckets can be added up until the bucket size is full and in the query operation each required buckets is checked to be a non-zero value instead of one. The delete operation, consists of decreasing the respective buckets value. One of problems in CBF is overflowing of the buckets. If this occurs, adding and deletion operations should not change the buckets set to the maximum possible value. The size of counters is usually 3 or 4 bits and consequently the use three or four times space more than normal bloom filters. Since we are interested in deleting IP addresses which had been seen less than two times in one week, a two-bit bucket is enough for our purpose of deletion.

Based on our definition of calculating a hop count value, selecting the smallest initial value in the set that is larger than its final TTL, this value will be between 0 and 31. Referring to this matter we built our data base as a cell matrix with 31 rows and m columns where m is the size of the standard CBF. The first column are the Hcs possible and the rest m columns are buckets of the CBF related to that Hc value. Equation 4.10 is our cell matrix which is, in fact, the data base of our filter.

$$\begin{bmatrix} x_{0,1} & x_{0,2} & \cdots & x_{0,m} \\ x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{31,1} & x_{31,2} & \cdots & x_{31,m} \end{bmatrix} \tag{4.10}$$

After acquiring set of regular legitimate source IP addresses, based on rule1 and rule2 mentioned before, the possible Hc's of each element is calculated and hashed through our hash functioning system. The corresponding values of these hashes are decreased in related row vectors of each hop count. It is important to consider that we may have to increment more than one element in our matrix due to the fact that different Hc values were taken in to account for close initial TTLs to avoid ambiguity. After training the data with all the elements of our frequent IP addresses in their corresponding Hcs row, the system is ready to check the new incoming packets. The possible Hc is calculated for each packet, the source IP address is then hashed and checked in the corresponding buckets of each Hc. If any of the elements would be more than zero, the packet can pass, and if not the packet would be discarded.

4.4. Statistical analysis and optimization of matrix Bloom Filter

An IP address is a 32 bit number in IPv4. To make a table and address all the existing IP addresses in, we need a table with 2^{32} bisections which needs immense memory to compile. However we can consider the IP addresses as strings and apply Hash functions on them. Any algorithm that maps data of variable lengths to one fixed length data is considered as a Hash function [22]. It is necessary that the hashes we use be deterministic, uniform and reliable on uniqueness as much as possible. The major performance metrics of Bloom filters are: computation time which depends on the number of hash functions k, size of the filter or elements which are m and n respectively and the probability of error which is the false positive probability of the filter calculated by Equation 4.5. This probability counts as the false negative rate of our filter against DDoS attack. With a constant array size, as it can be seen in Figure 4.2, more elements cause more false positive probability. However, we suppose the IP addresses that a server receives would alter only on their last 16 bits, we are faced with 2^{16} different IP addresses. This is the maximum possible number of the elements of our CBF design, $n = 65536$. the false positive can be attenuated in the cost of additional memory; the bigger the size of array, the less false positive probability which is far more obvious in log scale of the graphs in Figure 4.3. In the case of number of hash functions (k), there are two competing forces ; using more hash functions leads to a better chance of finding a 0 bit in checking an element existence. However this property holds to an extent which is optimum k computed by Equation 4.7. After this value any increase in k would lead to higher probability of false positive as it boosts the fraction

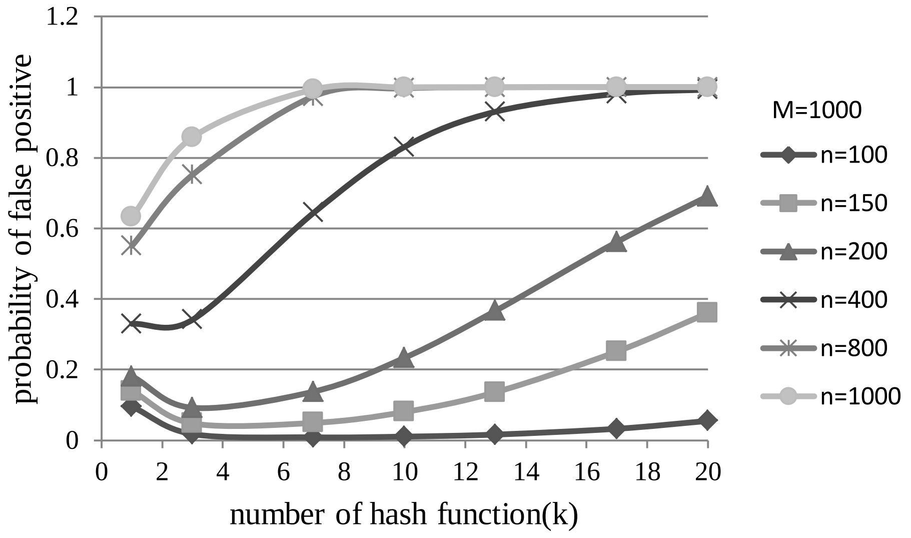


Figure 4.2. A comparison of probability of false positive with different elements numbers where array size is constant.

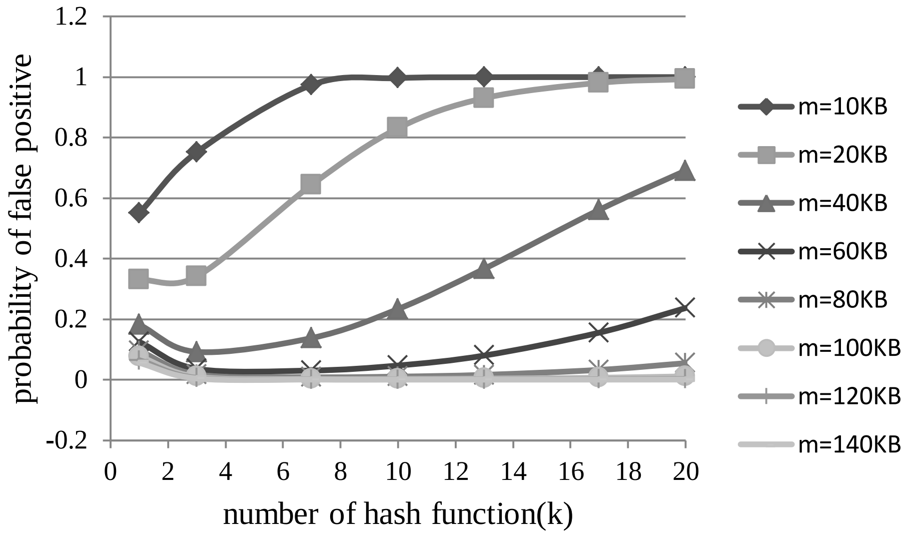
of 1 bits in filter's array. The affects of number of hash functions (k) and the size of the array (m) on false positive probability of the filter is depicted in Figure 4.3.

4.4.1. Optimization of Matrix Bloom Filter

4.4.1.1. Time and size optimization of filter. As mentioned before, we consider IP addresses that a server receives would alter on their last 16 bits. Therefor we encounter at most with 2^{16} different IP addresses $n = 65536$. According to Equation 4.11 and 4.12, holding to probability of false positive around 0.01, we need 8 hash functions and a 620Kb sized vector as our BF. Since our Bloom Filter is Counting Bloom Vector with buckets of 2 bits and we have 31 vectors the memory that our Filter requires is about 5MB.

$$m = -\frac{n \ln p}{(ln2)^2} = 623029 \approx 620Kb\,. \tag{4.11}$$

$$k_{opt} = \frac{m}{n}\ln(2) \approx \frac{9m}{13n} \approx 8\,. \tag{4.12}$$



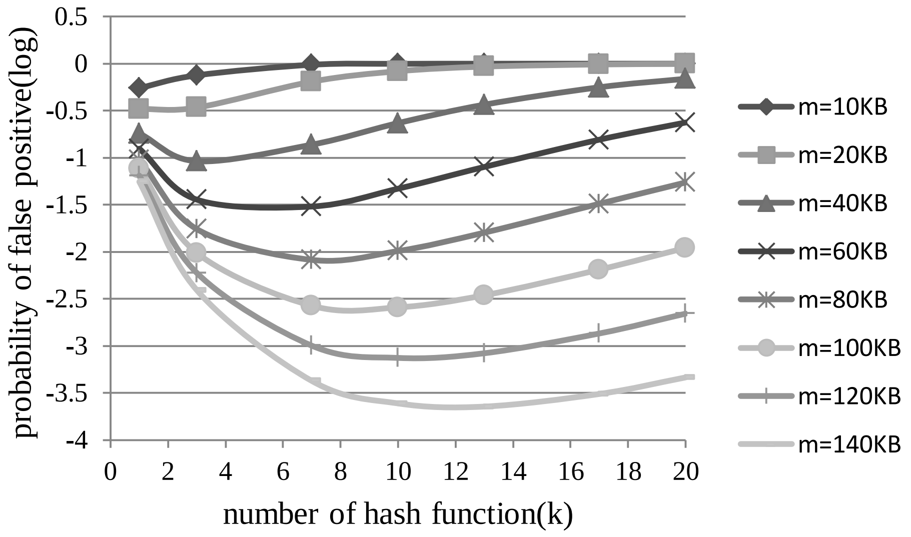


Figure 4.3. A comparison of probability of false positive with different array sizes and hash function numbers where number of elements is constants n=65536.

In order to have a robust hashing system, we submit two enhancements over the process of standard hashing. First, to reduce time consumption for hashing an IP address and make our hash function as uniform as possible, we propose to use one hash function SHA-1 [23] and divide it into eight parts and consider each as a distinct hash value instead of going through the process of calculating 8 different hash functions. It may seem this method causes collision in our bloom vector. On the contrary, it has no effects on the result of false positive when the number of divided hashes are more than four. Thus we can improve our training and look up algorithm regarding the time it takes to search the table with this method. Second, since SHA-1 gives a hash value with 160 bits, by partitioning it to eight parts, each subdivision would have 20 bits and addressing all these needs a vector with 2^{20} lengths. To adapt this hashes with the size of our filter which is around 620Kb, we use a modulo function with m as the divisor. This not only helps to qualify our hash values with the filter but also, if we consider the modulo function as part of hash function, make the hash function more uniform over the size of the vector m.

4.4.1.2. Adaptivity of filter with time deletion algorithm. So far, we tried to take all advantages of BFs and filling gaps of its deficiencies in our method. However, our Filtering scheme is prone to overflowing with IPs, leading to the failure of Bloom Filter in performing dramatically. To avoid such ill behavior, we propose a flag based method so the data structure of the filter has a regular deletion mechanism over certain period of times. Our filter is consisting of 2-bits cells, each of which capable to count till four. We consider a matrix with the same size of our filter's matrix, with flags all set to zero. For each cell, which is higher than 2, the corresponding flag element becomes 1. Afterward, all the elements corresponding to that zero flags are deleted in the filter. An example of such mechanism in a vector scale is given in Figure 4.4.

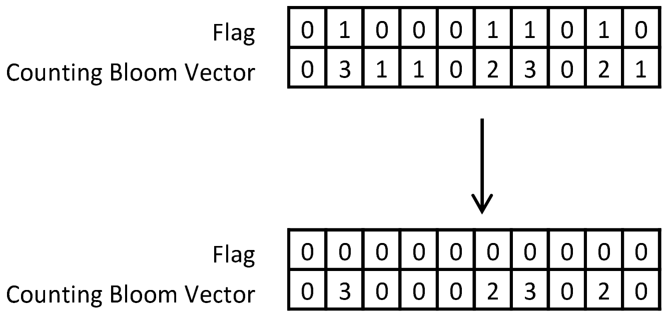


Figure 4.4. An example of flag deletion mechanism in vector scale.

Applying this deletion mechanism weekly, we delete the source IP addresses in white list which has been seen less than two times.

4.5. System accuracy evaluation

To evaluate the performance of our filter, we conduct two different analysis methods. In the first one, we consider a constant value for array size of the filter (m=1000) to evaluate the system performance for different numbers of elements (n) and number of hash functions (k). To perform this experiment, we trained the filter with 100 random IP addresses in Figure 4.5a. Then we examined the filter performance with 1000 random IP addresses. Same experiments are conducted for n=200, 400, and 800, and the results are shown in Figure 4.5b,c,d respectively. Probability of false negative is compared with the theoretical expected probability of false negative. As it can be inferred from Figure 4.5. with higher number of hash functions the error reduces considerably.

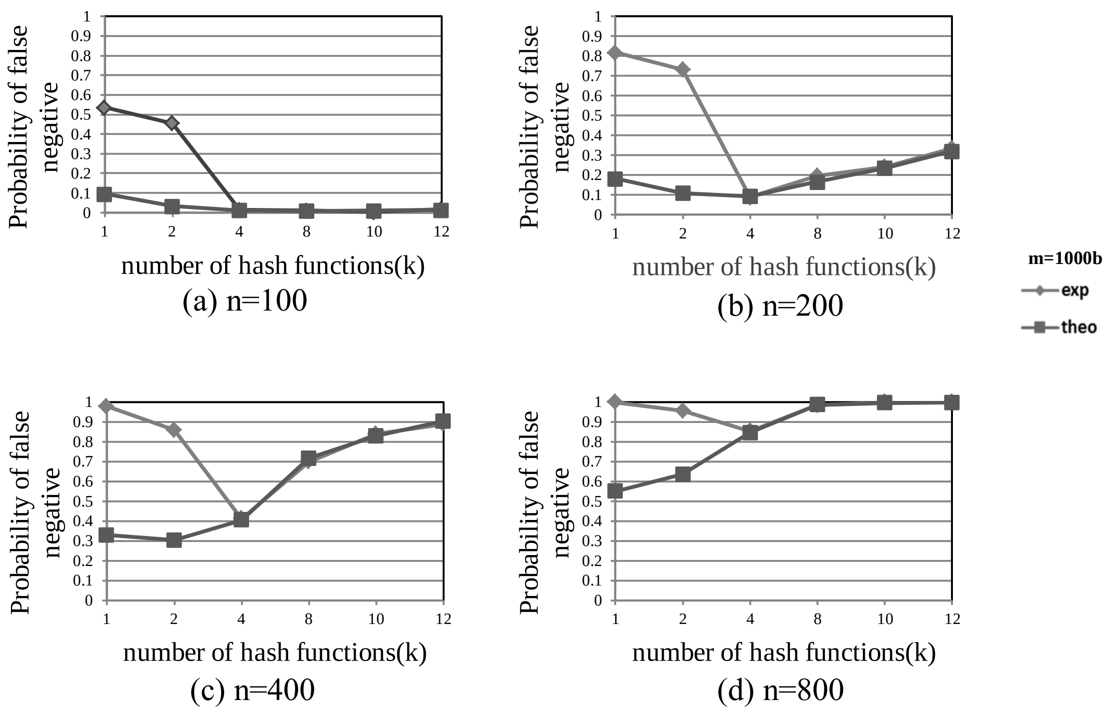


Figure 4.5. Probability of false negative considering m is constant at 1000b.

In the second analysis, four different sizes are considered for filters vector (m), while number of elements are kept constant (n=1000). This experiment is conducted over four different filter sizes m = 5Kb, 10Kb, 100Kb, and 1Mb. Figure 4.6 displays comparison results of theoretical values and tentative measurement of the probability of false negative for different numbers of hash functions. Although for small k, the difference between theoretical and experimental false negative is high, However by increasing the number of hash functions, this value reduces significantly, and they almost overlap with k's higher than four. Considering k is bigger than four, by increasing the size of the filter the false negative approaches to zero (Figure 4.6).

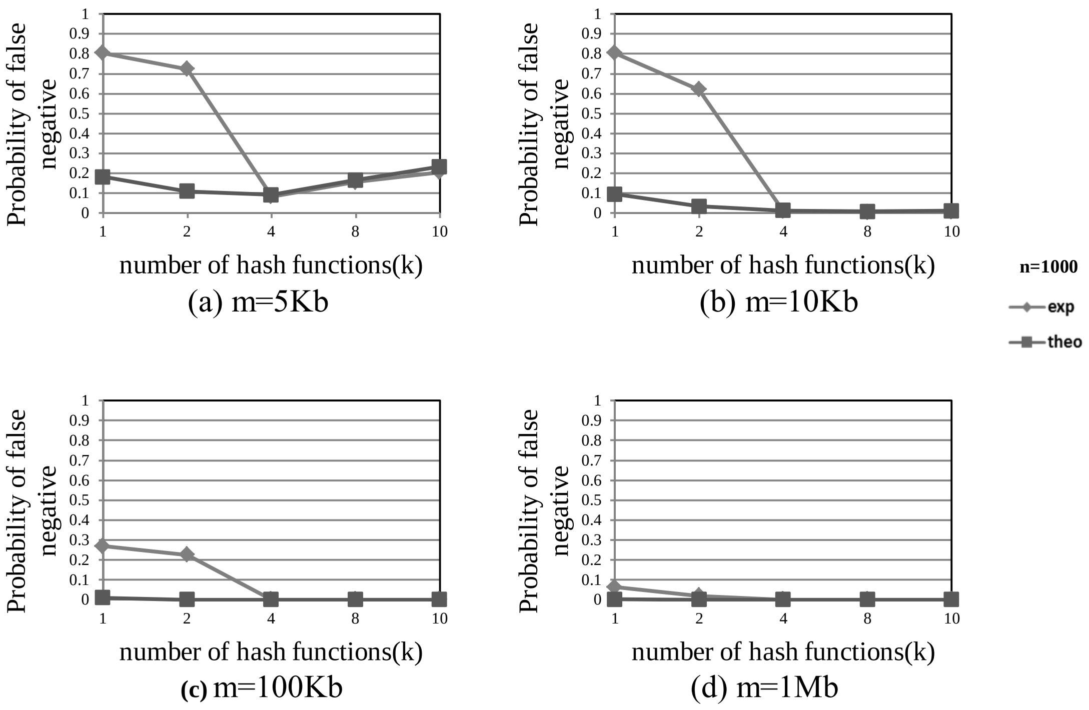


Figure 4.6. Probability of false negative considering n is constant at 1000.

To evaluate our matrix filtering system, we train the database of the filter with IP addresses with known TTLs, and then the filter is examined to determine the accuracy of filter in regard with attack packets and legitimate ones by calculating the false positive and false negatives. As it was mentioned before, the probability of error of Bloom Filters which is named as false positive is the probability of accepting a packet as part of the set while it is not. However, in filtering DDoS attack, this means considering an attack packet as a legitimate one which is the definition of false negative. Therefore we use the notion of false negative instead of probability of error in our Matrix Bloom Filter. With the new definition, the false positive means the probability of dropping a packet which belongs to our white list that should had been zero in Bloom filters. To be able to examine the filter accuracy, we apply our experimental assessment on small scale. In 5 days there were around 1700 IP addresses fed to filter from DARPA data set [24] as the legitimate addresses with their TTL's. These are the union of source IP addresses of training data in the dataset (attack free) of one week. Then the filter is ready to be evaluated toward attack packets. IP addresses of one of the same day were examined with the filter and the probability of false positive was zero as expected. However, there can be a small amount of false positive which is not related to the accuracy of our filter but the TTL differences of the new data. With a randomly made DDoS IP addresses, the contribution of the filter in encountering with randomly spoofed IPs and subnet spoofed IPs is summarized in Table 4.1. MBF works best when a subnet spoofing used in attacks because of Hc checking method in the filter.

Table 4.1. A comparison on filtering efficiency of normal bloom filter and matrix bloom filter in respect to different DDoS attacks.

Attack type	P_{f-n} with BF	P_{f-n} with MBF
Random IP spoofing	0.016	0
Subnet IP spoofing	1	0.002

5. FILTERING SCHEME IN EDGE ROUTERS

Although our filtering mechanism drops most of the DDoS packets in victim router, there is no guarantee that victims bandwidth or resource is secure from consumption with unwanted traffic. Moreover, to have the filter work properly there should be another filtering system in edge routers of the victim (victim router) to avoid any depletion in victim side. This filtering system should block part of the traffic, which is more likely to be from attackers by deciding on turning off some edge routers. When bandwidth depletion of DDoS happens, the traffic quota of edge routers can be observed via ISP. Meanwhile the filter calculates which routers are forwarding attack traffic and should drop the traffic ratio to zero. In the next step, the victim alarms its own ISP so the traffic blockage be applied on identified routers. In fact, the filtering problem can be considered as an optimization problem of receiving maximum good throughput in victim server with discarding maximum possible attack packets. Throughput of each upriver router can be considered as a weight of each router and our aim is then to maximize good put, with the weight that we measure in normal traffic as known variables of our problem [11]. We can identify our filtering model as an optimization problem as follows:

$$\begin{aligned} &\text{Maximize} \textstyle\sum_{i=1}^{n} P_i X_i \\ &\text{subject to} \textstyle\sum_{i=1}^{n} W_i X_i \leq C \\ &\text{with} P_i \geq 0, W_i \geq 0, C \geq 0 \end{aligned} \tag{5.1}$$

where n is the number of edge routers and C is the maximum throughput that our victim router can handle. W_i is the traffic weight transmitted from router i and P_i is the profit we assume for that router regarding its normal traffic quota and clients. If a router routes traffic to victim server, we define $X_i = 1$ and if not, we define $X_i = 0$.

Computational optimization techniques to solve such optimization problems differ in regard of problem classes. Classical exact methods generally take the form of exhaustive search in the space of restrains, while heuristic methods are usually a model of evaluation in a natural process. The latter solutions do not guarantee ideal optimal but an approximate solution which is desired mostly because of restrictions on computing power and time. In this thesis, we apply two optimization algorithms from each group, Linear programming (LP) and Genetic Algorithm (GA), and assess the advantages of each in our specific maximization problem.

5.1. Linear Programming optimization

Linear programming refers to a class of optimization problem (minimization or maximization) of a linear function with respect to a set of inequalities or equalities as its constrains. This method finds a point in which the function has maximum or minimum value by searching through a poly-tope made from the constrains, and a real-valued affine function defined on it. The linear programming problem stated as follows in [25]: "Among all feasible solutions, find one that minimizes (or maximizes) the objective function.". A set of feasible vectors, those meeting the criteria of corresponding constraints, is known as constraint set. If a constraint set of a linear programing function is a non-empty set, the problem is said to be feasible [25]; on the hand it is an infeasible one.". The general form of a linear program is:

$$\begin{aligned}
&\text{maximize } c_1x_1 + c_2x_2 + \ldots + c_nx_n \\
&\text{subject to } \quad \begin{aligned} a_{11}x_1 + \ldots + a_{1n}x_n &\le b_1, \\ &\vdots \\ a_{m1}x_1 + \ldots + a_{mn}x_n &\le b_m \end{aligned} \\
&\text{where } x_1 \ge 0, \ldots, x_n \ge 0
\end{aligned} \tag{5.2}$$

and its canonical from is:

$$\begin{aligned}
&\text{maximize } C^TX \\
&\text{subject to } AX \le B \\
&\text{where } X \ge 0
\end{aligned} \tag{5.3}$$

where C and B are vectors while A is a matrix, all with known coefficients. X is a vector of variables whose values are to be determined to maximize the objective function. B is a vector of size where m emphasizes the number of constrains plus n non-negativity restrictions of the variables. Other forms of minimization problems, alternative constrains forms or problems with negative variables can be rewritten in standard form. For instance, multiplying a minimum problem by -1 will change it to a maximum problem. Evidently, we can fit our model of optimization into this model by considering C as the Profit vector; A is the Weight vector while B is maximum throughput. X vector is the variable of our problem, which gives the routers status.

5.2. Genetic Algorithm optimization method

Genetic Algorithm (GA) [26] is a class of adaptive heuristic search algorithm based on the evolutionary principles of natural selection and genetics that was first laid down by Charles Darwin. It simulates the survival of the fittest among individuals, based on a fitness function, over consecutive generation for solving a problem, which can be an optimization problem. The Algorithm usually starts with a set of randomly selected solutions (represented by chromosomes) called population. Then it evaluates the fitness of each chromosome in the population. A new population is created by the means of selection, recombination and mutation. Selection is the process of selecting parent's chromosomes from the initial population according to their fitness values. Recombination with crossover represents crossing two parents to make a new offspring population with combining chromosomes from each parent and mutation introduces random modification in the chromosomes of each new offspring. Although this algorithm is modeled after natural processes, we can design our own encoding of information, our own mutations, and our own selection criteria. The genetic algorithm can be modeled by defining arrays of bits or characters that represent the chromosomes in GA. Figure 5.1 shows the chart of a genetic algorithm optimization in which a constrain is a stop criteria. The details of genetic algorithm steps are tabulated in Appendix A.

```
Initialize a population
Determine the fitness of the population
While the constrain is not met
    Select parents from population (selection)
    Perform crossover on parents to create new population (crossover)
    Perform mutation on new population (mutation)
    Determine the fitness of the new muted population
    Check if the constrain is met with new fitness value
End
```

Figure 5.1. Steps of Genetic Algorithm in optimization problems.

5.3. Experimental results of LP and GA optimization methods in DDoS filtering

To evaluate Genetic Algorithm and Linear Programming accuracy in solving the problem, we consider a network with 20 routers with different numbers of nodes connected to our victim server. The detailed information of the routers, clients and throughput is given in Table 5.1. Second column shows the number of users connected to each router while two next columns represent the number of normal users and attackers from the number of nodes. We considered 0.2 Mb throughput for each normal client and the Good throughput (Gt) is calculated in regard with this value and the number of normal clients. The last column displays the traffic weight (W) passing from each router. Maximum throughput that our victim server can handle is 10Mb.

The evaluation process consists of two experiments with both algorithm applications on mentioned network. The first experiment is based on uniformly distributed priority. In other words, there are no specific priority on status of any routers. Table 5.2 shows the state of each router while in Table 5.3 the number of attackers and percentage of the good throughput the victim receives after applying each GA and LP is summarized. As it is evident, not only GA gives a better result in blocking attackers but also it is faster than LP. This is mostly because LP is an exact optimization algorithm, which without priorities it tries to maximize the whole throughput no matter of attack or normal traffic under specified constrains while GA technique, models this problem in a way that cannot be modeled as accurately as with other approaches, which makes it more potentially useful in this problem. As mentioned before, GA is faster in

Table 5.1. Network information with 0.2Mb throughput from each normal client.

router	nodes	normal clients	Attacker	GT(Mb)	W(Mb)
R1	4	4	0	0.8	0.8
R2	2	0	2	0	1.2
R3	3	2	1	0.4	1
R4	3	3	0	0.6	0.6
R5	5	2	3	0.4	2.2
R6	2	0	2	0	1.2
R7	1	1	0	0.2	0.2
R8	5	3	2	0.6	1.8
R9	3	0	3	0	1.8
R10	1	1	0	0.2	0.2
R11	1	0	1	0	0.6
R12	2	0	2	0	1.2
R13	3	3	0	0.6	0.6
R14	1	0	1	0	0.6
R15	1	0	1	0	0.6
R16	5	4	1	0.8	1.4
R17	2	1	1	0.2	0.8
R18	1	1	0	0.2	0.2
R19	5	5	0	1	1
R20	3	0	3	0	1.8
total	53	30	23	6	19.8

finding an approximation of optimized answer because of its stochastic characteristics.

Table 5.3. Accuracy evaluation of LP and GA with uniform distributed priority.

Algorithm	blocked attackers	GT percentage
LP	12/23	66.6%
GA	15/23	93.3%

Table 5.2. State of routers after applying LP and GA algorithm where 1 means "on" and 0 means "off" with uniformly distributed priority.

Algorithm	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
LP	1	1	1	1	0	1	0	0	0	1
GA	1	0	1	1	0	0	1	1	0	1
Algorithm	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	x_{17}	x_{18}	x_{19}	x_{20}
LP	1	1	1	1	1	1	0	0	0	1
GA	1	1	1	0	0	1	1	1	1	0

In the second experiment, by considering 20 points for each normal client connected to router, we try to optimize our result by choosing proper priorities P_i . In fact, we can define our problem as a conditional problem with specific expectations from the behavior of both optimization approaches and apply constraints on the status of specific routers. For more illustration, consider a router which in normal situation has 4 normal nodes attached to it which can be estimated in real networks. By giving proper priority points to such a router we oblige our optimization algorithm to keep this router 'on' with most probability in solution (see Table 5.4). LP optimization algorithm gives the maximum possible good throughput with given priorities which is about 5.8Mb of total good throughput of 6Mb. As it is obvious in both Tables we gained same accuracy with both algorithms. However, GA takes less time for finding the solution, which is explicable with randomness attributes of GA. In other words, LP goes through each possible points in the polytop of constrains while GA uses random points on the same polytop which at most will take the same time as LP.

Table 5.4. State of routers after applying LP and GA algorithm where 1 means "on" and 0 means "off" with designated priorities.

Algorithm	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
LP	1	0	1	1	1	0	1	1	0	1
GA	1	0	1	1	1	0	1	1	0	1
Algorithm	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	x_{17}	x_{18}	x_{19}	x_{20}
LP	0	0	1	0	0	1	0	1	1	0
GA	0	0	1	0	0	1	0	1	1	0

Table 5.5 demonstrates the result of such priority designation with 20 points for each normal client.

Table 5.5. Accuracy evaluation of LP and GA with designated priorities.

Algorithm	blocked attackers	GT percentage
LP	16/23	96.6%
GA	16/23	96.6%

6. CONCLUSIONS

A DDoS attack is a distributed, large-scale attack made by malicious users to deplete the victim's resources with an enormous volume of data traffic, which consequently exhausts the victim's network of resources such as bandwidth, computing power or RAM. DDoS tools provide several spoofing levels, e.g. IP address spoofing, in order to open the path for malicious packets into victim. Characterization and filtering attack traffic major difficulty is because disguised legitimate packets are used for attacks. The distributed description of the attacks needs a distributed filtering scheme, but administrative domains cooperation is costly and a troublesome task. The main mechanisms against DDoS after detection, is filtering, which stops DDoS attack packets from reaching the victim in proximity of victim (victim based) or in upriver routers (edge router based). In comparison to the router-based method, the victim-based approach is employable immediately and it has more potential to invest on a defense mechanism rather than a network provider. In this thesis, we proposed a host based filtering approach, which selects specific features from the behavior of traffic in host's server router such as payload statistic behavior to detect the probability of occurrence of an attack and consequently determine the state (alert or active) of the filter in next step. Furthermore, it is used as an alarm mechanism for filtering mechanism in edge routers to mitigate the traffic flowing to victim's router, to avoid filtering process from getting overwhelmed by the huge amount of traffic passing throw it. However, if the detection mechanism does not detect the spoofed packets and discard them, they can easily reach the victim server as a legitimate packet. To have a fast, memory efficient look up process, we utilized an advance Bloom Filter to train and build the database of source IP addresses. Filter outlines are improved by combining two packet filtering methods, IP history based and Hop Counting filtering method. The filter consists of a matrix data-base where each column is an enhanced Counting Bloom Filter, a look up algorithm with specific hash functioning method and a renewal algorithm to make the filter adaptive in time. The probability of false alarm is reduced from 0.016 to 0.002 in subnet spoofed attack case. While a filter can detect and drop attack packets , it may not be able to preserve the availability of the service. To overcome this problem, optimization based data mitigation is done in edge routers of the victim server after detecting an intrusion. In that section two optimization, Genetic Algorithm and linear programming is used to optimize the good throughput under the constraint of the maximum throughput that the victim router can manage and with appropriate priority acquisition, we obtained up until 96.6% of good throughput where GA was relatively faster due to randomness characteristic of substructure steps of genetic evolution such as crossover and mutation which leads to an estimated optimization of the problem.

APPENDIX A: GENETIC ALGORITHM STEPS

A.1. Encoding

To use genetic algorithm for optimization purposes we need to define our model, our chromosomes, selection, crossover and mutation rules and also what we want to optimize in a function model which is named fitness function $f(x)$ where x is the chromosome. This process is called encoding. Encoding parameters of optimization problem to chromosomes is the first step of solving it with GA. There are several different methods to encode the chromosomes which depends on the problem. Among different encoding methods, the binary encoding is the most popular one. In this method all chromosomes consist of just one binary string. One or more characteristics of the problem is presented by the bits inside the string. Moreover, the string can be a number by itself. An example of binary encoding is given in Table A.1. Permutation encoding which is usually used in ordering problems, such as traveling Salesman Problem, is one of the encoding methods in which every chromosome is defined as a string of numbers. There are other methods such as value encoding or Tree encoding, that can be chosen regarding the problem one may face.

Table A.1. Binary encoded chromosomes examples.

Chromosome 1	1 1 1 0 0 1 0 1
Chromosome 2	1 0 0 0 1 1 0 1

A.2. Selection

When chromosomes are encoded, they should be selected from the initial population as parents and then left to the process of Crossover. The individuals with a higher fitness must have a higher probability of having offspring. To obtain the best chromosome modeling, there are several methods i.e. roulette wheel selection, Boltzman selection, tournament selection, rank selection, steady state selection. In roulette wheel selection, the probability of parenthood is considered proportional to their fitness value and then parents are randomly chosen for making the next generation like a roulette wheel. This method would lead to unfit selection if the fitness values have huge differences. Individuals, who have a much higher fitness, could be the parents for every child in the next generation. Another selection method is tournament selection, which selects four or more individuals, and their fitness values are compared two by two and ones with higher values are selected as parents. This selection continues until a new population is created.

A.3. Crossover

After encoding the chromosomes and selection, there is the matter of recombination which is done by cross over. Crossover can be done randomly or uniformly over the range of chromosomes. An example of one point crossover can be seen in Table A.2. The line shows where two parents are passed through crossover and two offspring are made by combining the parts of parents. Crossover probability is one of

Table A.2. One point crossover in chromosomes which leads to two distinct offsprings.

Chromosome 1	1 1 1 | 0 0 1 0 1
Chromosome 2	1 0 0 | 0 1 1 0 1
Offspring 1	**1 1 1** 0 1 1 0 1
Offspring 2	1 0 0 **0 0 1 0 1**

the parameters of GA, which is an indication of how often the crossover is performed on the chromosome. If it is 0 percent, it means the new generation is made from same chromosomes of old population. However it does not mean that the new generation is exactly a copy of the last population which is mainly because of mutation.

A.4. Mutation

The mutation is performed after cross over in GA. It is mostly performed to prevent the solution on falling into a local maximum or minimum. Mutation is in fact a slight random change in the new offspring. In binary encoding it can be a randomly chosen bits changes from 1 to 0 or vice versa. For better illustration see Table A.3 Mutation probability indicates how often parts of chromosome are mutated.

Table A.3. Example of mutation in Offsprings.

Offspring 1	1 1 1 0 1 1 0 1
Offspring 2	1 0 0 0 0 1 0 1
Offspring 1 after mutation	1 **0** 1 0 1 1 0 1
Offspring 2 after mutation	1 0 0 **1** 0 1 0 1

0 probability of mutation means there is no mutation and offspring is taken directly after crossover directly. If mutation probability is 100 percent, then it means all the bits of chromosome is changed. Mutation is made to prevent GA from falling into local extreme, but it should not occur very often, because then GA will in fact change to a random search.

REFERENCES

1. Abbate, J. E., *From ARPANET to Internet: A History of ARPA-sponsored Computer Networks, 1966–1988*, Ph.D. Thesis, 1994.

2. Press, L., “Commercialization of the Internet”, *Communications of the ACM*, Vol. 37, No. 11, pp. 17–21, 1994.

3. Mirkovic, J., S. Dietrich, D. Dittrich and P. Reiher, *Internet Denial of Service: Attack and Defense Mechanisms (Radia Perlman Computer Networking and Security)*, Prentice Hall PTR, 2004.

4. Mankin, A., D. Massey, C.-L. Wu, S. F. Wu and L. Zhang, “On Design and Evaluation of Intention-Driven ICMP Traceback”, *Computer Communications and Networks, 2001. Proceedings. Tenth International Conference on*, pp. 159–165, IEEE, 2001.

5. Mirkovic, J., G. Prier and P. Reiher, “Attacking DDoS at the Source”, *Network Protocols, 2002. Proceedings. 10th IEEE International Conference on*, pp. 312–321, IEEE, 2002.

6. Zaroo, P., “A Survey of DDoS Attacks and some DDoS Defense Mechanisms”, *Advanced Information Assurance (CS 626)*, 2002.

7. Mirkovic, J., G. Prier and P. Reiher, “Source-end DDoS Defense”, *Network Computing and Applications, 2003. NCA 2003. Second IEEE International Symposium on*, pp. 171–178, IEEE, 2003.

8. Peng, T., C. Leckie and K. Ramamohanarao, “Protection from Distributed Denial of Service Attacks using History-based IP Filtering”, *Communications, 2003. ICC’03. IEEE International Conference on*, Vol. 1, pp. 482–486, IEEE, 2003.

9. Wang, H., C. Jin and K. G. Shin, “Defense against Spoofed IP Traffic using Hop-

count Filtering", *IEEE/ACM Transactions on Networking (TON)*, Vol. 15, No. 1, pp. 40–53, 2007.

10. Tan, H.-X. and W. K. Seah, "Framework for Statistical Filtering against DDoS Attacks in MANETs", *Embedded Software and Systems, 2005. Second International Conference on*, pp. 8–pp, IEEE, 2005.

11. Feng, Y., R. Guo, D. Wang and B. Zhang, "Research on the Active DDoS Filtering Algorithm Based on IP Flow", *Natural Computation, 2009. ICNC'09. Fifth International Conference on*, Vol. 4, pp. 628–632, IEEE, 2009.

12. Mirkovic, J. and T. Benzel, "Teaching Cybersecurity with DeterLab", *Security & Privacy, IEEE*, Vol. 10, No. 1, pp. 73–76, 2012.

13. Guide, M. U., "Mathworks", *Inc., Natick, MA*, Vol. 5, 1998.

14. Veldhuizen, T., "Template Metaprograms", *C++ Report*, Vol. 7, No. 4, pp. 36–43, 1995.

15. Hafner, J., H. S. Sawhney, W. Equitz, M. Flickner and W. Niblack, "Efficient Color Histogram Indexing for Quadratic Form Distance Functions", *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, Vol. 17, No. 7, pp. 729–736, 1995.

16. Jung, J., B. Krishnamurthy and M. Rabinovich, "Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites", *Proceedings of the Eleventh International Conference on World Wide Web*, pp. 293–304, ACM, 2002.

17. Sung, M. and J. Xu, "IP Traceback-based Intelligent Packet Filtering: a Novel Technique for Defending against Internet DDoS Attacks", *Parallel and Distributed Systems, IEEE Transactions on*, Vol. 14, No. 9, pp. 861–872, 2003.

18. Kandula, S., D. Katabi, M. Jacob and A. Berger, "Botz-4-sale: Surviving Organized DDoS Attacks that Mimic Flash Crowds", *Proceedings of the Second Confer-*

ence on Symposium on Networked Systems Design & Implementation-Volume 2, pp. 287–300, USENIX Association, 2005.

19. Ye, F., H. Luo, S. Lu and L. Zhang, "Statistical En-route Filtering of Injected False Data in Sensor Networks", *Selected Areas in Communications, IEEE Journal on*, Vol. 23, No. 4, pp. 839–850, 2005.

20. Bloom, B. H., "Space/time Trade-offs in Hash Coding with Allowable Errors", *Communications of the ACM*, Vol. 13, No. 7, pp. 422–426, 1970.

21. Fan, L., P. Cao, J. Almeida and A. Z. Broder, "Summary Cache: a Scalable Wide-area Web Cache Sharing Protocol", *IEEE/ACM Transactions on Networking (TON)*, Vol. 8, No. 3, pp. 281–293, 2000.

22. Coron, J.-S., Y. Dodis, C. Malinaud and P. Puniya, "Merkle-Damgård revisited: How to construct a Hash Function", *Advances in Cryptology–CRYPTO 2005*, pp. 430–448, Springer, 2005.

23. Eastlake, D. and P. Jones, "US Secure Hash Algorithm 1, SHA1 (2001)", *Dostupné online (marec 2011): http://tools. ietf. org/html/rfc3174, Request for Comment*, Vol. 3174.

24. Lippmann, R., J. W. Haines, D. J. Fried, J. Korba and K. Das, "The 1999 DARPA off-line Intrusion Detection Evaluation", *Computer Networks*, Vol. 34, No. 4, pp. 579–595, 2000.

25. Bazaraa, M. S., J. J. Jarvis and H. D. Sherali, *Linear Programming and Network Flows*, Wiley. com, 2011.

26. Whitley, D., "A Genetic Algorithm Tutorial", *Statistics and Computing*, Vol. 4, No. 2, pp. 65–85, 1994.