Chapter 4

A DDoS Detection and Mitigation System Based on Bidirectional Nature of Internet Communications

In the previous chapter, I discussed a light weight DDoS detection technique to detect an ongoing attack in near real time. However, a DDoS defense solution should be capable of not only to detect the attack but also to discriminate the attack packets from normal packets. In this chapter, I present a DDoS defense system for the detection and mitigation of different types of commonly seen DDoS attacks. The system assumes bidirectional traffic information at an edge router to detect and mitigate the attacks. A router might not always see the out going traffic corresponding to the incoming traffic carried by the router and which has always been a problem for other approaches which assume bidirectional traffic at the monitoring point. I introduce an agent-based technique which enables each edge router to validate the bidirectional nature of the incoming traffic passing through them. I present several experiments demonstrating the effectiveness of the proposed detection and mitigation system.

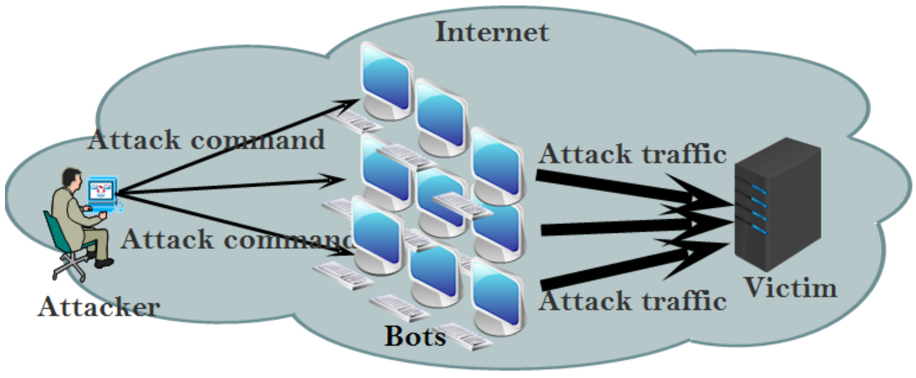
4.1 Introduction

In a DDoS attack the attacker first compromises a set of machines, called as bots(zombies).The entire network of such compromised machines controlled by a single user (*i.e.*, the actual attacker) is called a botnet. Once control over one or more botnet is achieved, the attacker commands the zombies to send Internet traffic to one or a set of selected servers called as victim(s) in an attempt to exhaust the resources of the victim such as CPU, memory and link bandwidth (the link could be any where between the first mile and last mile router). Under such a situation the legitimate users of the victim(s) experience high degradation of service or at worst situation, no service at all. Figure 4.1(a) shows a pictorial representation of a DDoS attack along with its participants such as attacker, bots and victim. However, it is not always necessary that the attacker needs to have a botnet of compromised machines to perform a successful DDoS attack. A variant of DDoS attack that doesn't require a botnet to launch the attack is known as reflection DDoS attack (DRDoS)[7]. In a DRDoS attack vulnerable public servers which response to queries, mostly over UDP such as DNS and NTP are used as reflectors to carry out a DDoS attack. Figure 4.1(b) shows a pictorial representation of DRDoS attack. To perform an attack, the attacker sends requests to such reflectors but spoofs the SIP of the request packets as the victim's IP. When the reflectors receive such requests they send the responses to the corresponding SIP *i.e.*, to the victim. The victim thus receives a large volume of response messages from many such reflectors sufficient to create a DoS situation at the victim. Two important advantages of DRDoS attacks are as follows.

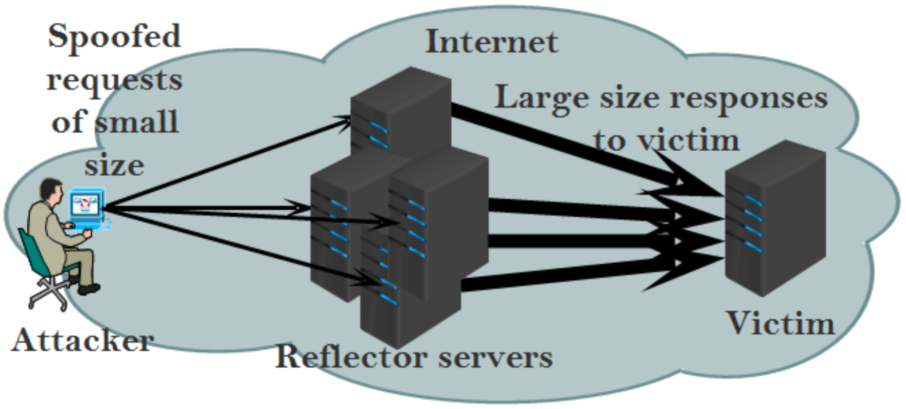
1. A large botnet is not required to carry out a massive attack. Even with a single machine an attacker can launch a massive DRDoS attack.

2. In DRDoS attack, the attacker can achieve a bandwidth amplification of the final attack traffic towards the victim. The size of the response messages of the reflector servers such as NTP and DNS, are typically many times bigger than that of the request messages. The bandwidth amplification can be calculated as the ratio of the number of bytes in the response messages to the number of bytes in the corresponding request message. The attacker can easily amplify the attack traffic by many hundred times using proper reflectors. The different types of reflectors and along with their amplification factors are well documented in [7].

In a DRDoS attack, the attacker usually aims to exhaust the network links of the victim by flooding the victim with a high volume of unwanted traffic. Two other most commonly used DDoS attacks are TCP SYN flooding attack and UDP flooding attack. In a TCP flooding attack the attacker instructs the bots to send TCP SYN packets to the victim with spoofed source IP in an attempt to exhaust



(a) DDoS attack and its participants


Spoofed requests of small size
Internet
Large size responses to victim
Attacker
Reflector servers
Victim

(b) A Reflection DDoS attack

Figure 4.1: DDoS attack and its variants

the TCP state table of the victim machine. In UDP flooding attack the bots are instructed to send huge number of UDP packets to the victim. Such attacks exploit the network/transport layer features of Internet Protocol to conduct the attack and commonly known as network/transport layer DDoS attack.

In this chapter I present a defense system to detect and mitigate commonly seen network/transport layer DDoS attacks. The proposed system is advantageous in view of the following points

- Accurate detection of an attack at the edge of a protected network in near real time.
- Effective mitigation of the attack irrespective of whether the SIP used in the attack packets are real or spoofed.

- Scalable and robust, in the sense that the attacker has very limited control over filling up the resources used to keep track of necessary information by the defense system.

The rest of the chapter is organized as follows. In section 4.2, I discuss the proposed DDoS defense system. Section 4.3 presents the experimental evaluation of the defense system under different types of attack scenarios. Conclusions are drawn in section 4.4.

4.2 DDoS Detection and Mitigation System (DDM)

The proposed defense system, DDM, operates at the edge routers of a network to protect the servers inside the network. Such a network could be a tire-3 network which is connected to the Internet through one or more Internet Service Providers. Figure 4.2 shows such a network. In the figure, the routers marked as PR_i ($i =$

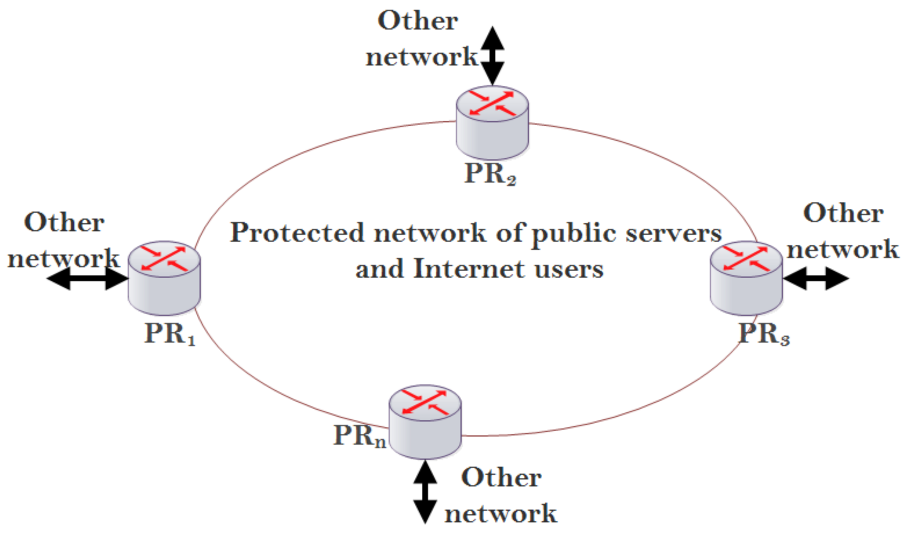


Figure 4.2: Topology of a protected network

$0, 1, 2, 3..$) are the edge routers of the network which connect the network with the rest of the Internet *i.e.*, all the incoming/outgoing traffic to/from the network are visible at these routers. The protected network consists of different public servers such as web servers, e-mail servers, DNS servers and NTP servers along with individual Internet users. Given such a network, the goal is to detect and mitigate one or more DDoS attacks going on against one or more servers inside the network.

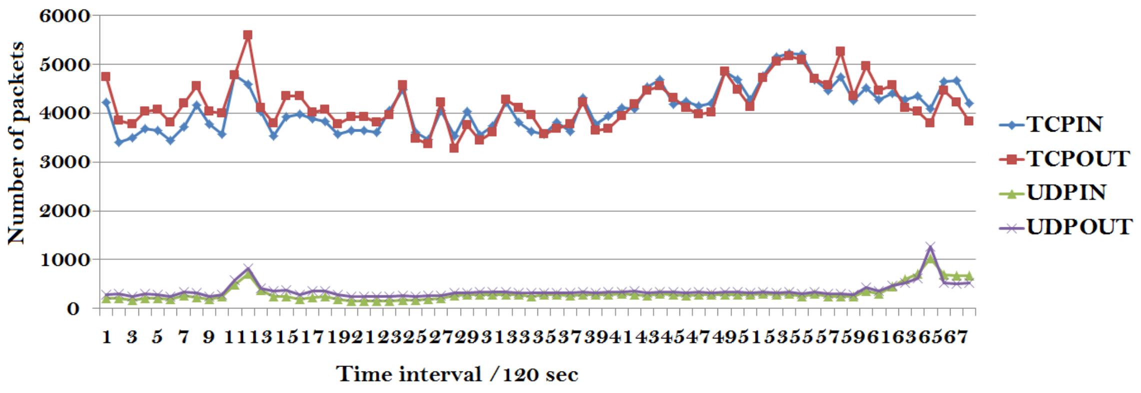
Table 6.1 includes a list of abbreviations and their meanings which are used to describe the rest of the chapter.

Table 4.1: Short hand notations and names used in chapter 4

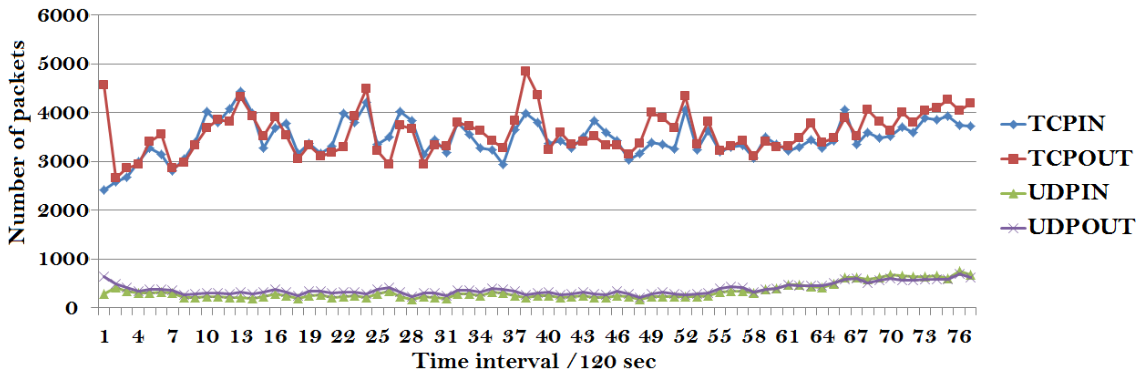
NAME	DESCRIPTION
PR	Perimeter Router and represents an edge router of the protected network
PRID	16 bit ID of a PR which it inserts into the 16 bit ID field of the incoming packets
SIP	Source IP address of a packet
FU	Forward Unit of the defense system
BU	Backward Unit of the defense system
FTT	Forward Traffic Table to keep track of incoming traffic information
BTT	Backward Traffic Table which is used by the BU to keep track of the control packets received from the *bi-agent* s. The FU receives copy of this table from the BU at each interval.
CAL	Current Arrival List, representing the set of newly arrived sources in a particular interval
BLT	Black List Table, which is used by the FU to keep track of the identified attack sources
GL	Good List, which records the sources of the communications validated by the FU over a period of time
$GoodFilter$	Filters an entry against the GL
ϵ	Maximum number of time intervals which can elapse between the first incoming packet from a source and its corresponding out going packet, before considering the source as unidirectional

The proposed defense mechanism is based on the observation that most of the Internet traffic is bidirectional in nature, *i.e.*, both the ends in a communication usually send packets to each other, might be at different rate. Not only TCP but UDP based protocols such as DNS, NTP and SNMP also possess bidirectional nature of communication. However, there are Internet traffic also which are completely unidirectional in nature such as those generated by push services. However, the overall traffic from such services are very little compared to the rest of the traffic and can hence be considered as special cases. Figure 4.3 shows the incoming vs outgoing traffic generated by TCP and UDP protocols for network traces AUK-VIII and WAIK-VIII. (Description of the traces are presented in Chapter 2).

We can see a high correlation between the incoming and outgoing traffic in both the protocols. Figure 4.4 shows the unidirectional traffic present in AUK-VIII and WAIK-VIII, which is very less ($< 0.025\%$) compared to the entire traffic. Manual inspection of these unidirectional traffic reveals that most of these traffic are UDP traffic. For our experiments we filtered such unidirectional communications from the traces.



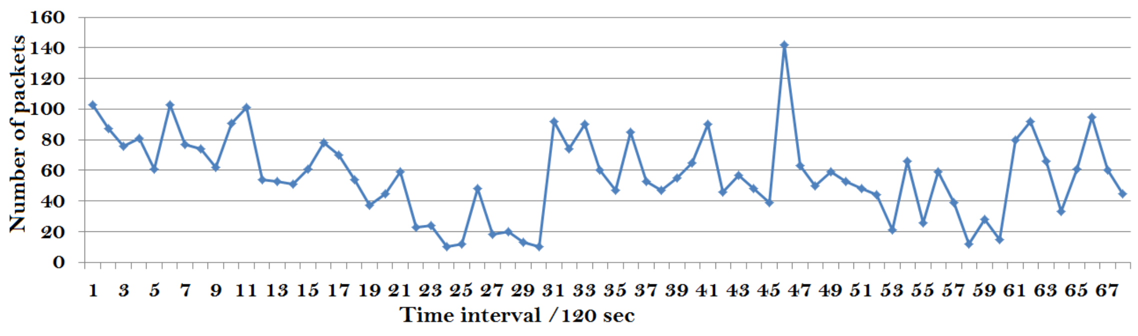
(a) AUK-VIII trace



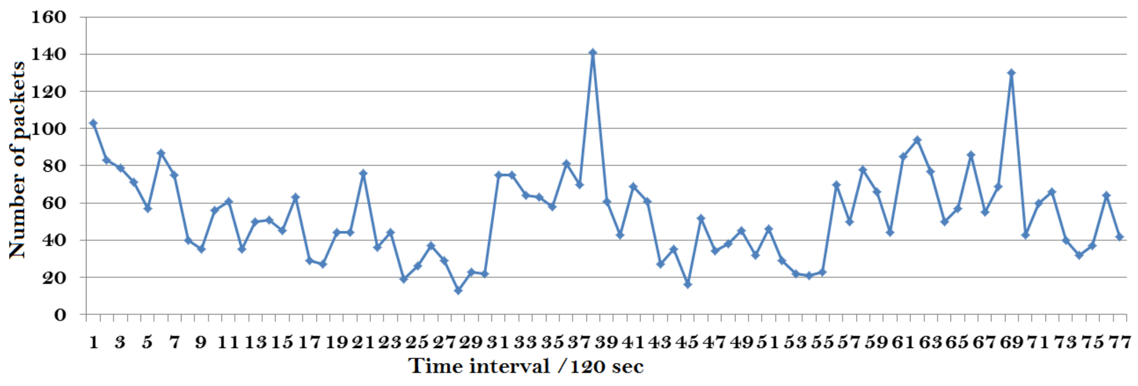
(b) WAIK-VIII trace

Figure 4.3: Incoming vs outgoing TCP and UDP traffic for normal traces

On the contrary to the bidirectional nature of a legitimate Internet communication, DDoS attack traffic generated in most of the DDoS attacks such as SYN flooding, UDP flooding, and Reflection DDoS are unidirectional in nature. Let us consider a TCP SYN flooding attack, where the bots continuously send TCP SYN packets to the victim, possibly with spoofed SIP. In such an attack, the bots do not participate in valid TCP communications with the victim. Although, the victim replies each SYN packet by a SYN/ACK packet but it will never sends a pure TCP ACK packet to a bot. Thus, if we consider a pure TCP ACK packet from a server to a specific IP as a sign of bidirectional TCP communication, then, under a TCP SYN flooding attack, we will observe a huge number of unidirectional communications with the victim server. Similarly, in a DRDoS attack, the victim receives a large number of response messages from a set of public servers (used as reflectors by the attacker) which it never requested for, causing a huge number of



(a) AUK-VIII trace



(b) WAIK-VIII trace

Figure 4.4: unidirectional traffic present in the normal traces

unidirectional communications towards the victim. Thus, bidirectional nature of a communication can be used to verify the source of the communication as either legitimate or an attack source.

In Figure 4.2 we see that the rest of the Internet interacts with the servers within the protected network only through the edge routers, *i.e.*, through the PRs of the protected network. We deploy our defense system at these routers, independent of each other, to monitor the bidirectional nature of the communications going through them to detect and mitigate possible DDoS attacks. Since our defense system checks bidirectional pattern of the communications going through the corresponding PR, it implies that all the outgoing traffic corresponding to the PR's incoming traffic should exit the network through the same PR. Such a requirement is difficult to achieve under practical situation. Figure 4.5 shows an example where the incoming (user P to server S) and outgoing (server S to user P) packets of a communication is visible at two different PRs of the network. To overcome this limitation we discuss here an agent-based technique, which enables each PR to have the information regarding the outgoing traffic corresponding to the traffic entered into the network through that particular PR.

4.2.1 Agent-based Bidirectional Information

To validate the bidirectional pattern of a communication, a PR needs to know the IP addresses to which the servers from the protected network are sending traffic in response to the incoming traffic through that PR. To achieve this goal the following assumptions are made

1. Each PR in the protected network is given a 16-bit unique ID, called as PRID. When a packet enters into the network, the corresponding PR inserts its PRID into the 16 bit ID field of the packet.
2. Each server is associated with an agent called as *bi-agent.* The *bi-agent* is installed between the server and the first router to which the server is connected with, as shown in Figure 4.5.

The *bi-agent* maintains a hash table to keep track of the PRIDs of the incoming packets to a particular server. The key of the hash table is the source IP of the incoming packet and the value is the 16 bit ID field of the incoming packet. For each incoming packet the *bi-agent* extracts the SIP and PRID and updates the hash table accordingly. For each outgoing packet from the server, if the packet is a valid bidirectional packet (for example a pure TCP ACK packet) the *bi-agent* generates a control packet with the 32 bit destination IP field of the outgoing packet as the payload of the control packet. The same destination IP is then used to retrieve the PRID from the hash table. The control packet is then sent to the PR pointed out by the PRID. In order to convert the 16 bit PRID to actual 32 bit IP address of an edge router, the *bi-agent*s maintain a static table which maps each PRID to an edge router. Figure 4.5 demonstrates an example of the technique explained above.

Let us assume, there are two different sources, *i.e.*, P and Q which send request packets p_{req} and q_{req}, respectively to a server S inside the network. Also, assume both these requests enter into the network through the same edge router PR_1. PR_1 inserts its 16 bit PRID into the packets ID field. When these packets reach server S, the corresponding *bi-agent* records the SIP and ID field of the packets in its hash table. Based on the server's response, the *bi-agent* generates and sends control packets $p_{control}$ and $q_{control}$ to PR_1. As shown in Figure 4.5, even if the actual

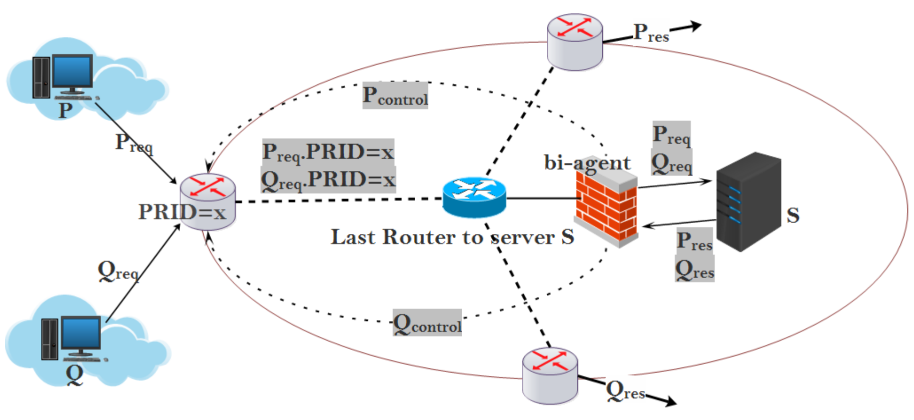


Figure 4.5: Demonstration of the working of a *bi-agent*

response packets P_{res} and Q_{res} do not exit through PR_1, PR_1 still can verify the bidirectional nature of the sources P and Q based on the control packets it received from the *bi-agent* .

The task of the *bi-agent* associated with a server is to keep track of the source IP addresses which communicate to the server. An attacker who is informed about the defense system might try to exhaust the *bi-agent* by sending attack traffic from a large number of sources. However, the following two points should be noted about a *bi-agent* .

1. For each SIP, a *bi-agent* needs to maintain only the 16 bit PRID and hence the *bi-agent* can easily keep track of millions of sources without getting overwhelmed.

2. A *bi-agent* receives attack traffic only during the initial period of an attack. As soon as an attack is detected by the defense system at the edge of the protected network, the attack traffic is dropped at that point itself.

Hence, an attacker's attempt to exhaust a *bi-agent* will only make the attack more prominent to the defense system, making it extremely difficult to achieve the attacker's malicious goal.

4.2.2 Deployment of the Defense System At An Edge Router

In this section we consider the deployment of the defense system at a PR which observes the incoming traffic as well as the control packets received from the internal servers to detect and mitigate any possible DDoS attacks. Each PR has two different units referred to as Forward Unit (FU) and Backward Unit (BU). The task of the FU is to maintain a list of all SIPs which have communicated through the corresponding PR to one or more servers inside the protected network. On the other hand the BU is responsible for collecting the control packets sent by the *bi-agent*s associated with the internal servers. In other words, FU maintains the list of SIPs from which packets are coming into the protected network, and BU maintains the list of SIPs to which packets are sent from the protected network. Thus at any instant of time, comparison of the lists of SIPs maintained by the FU and BU reveals the sources which are not bidirectional. The details of the FU and BU are discussed below

1. Backward Unit: As mentioned earlier, for each incoming packet through a PR, the *bi-agent* of the destination server sends a control packet to the PR. The corresponding BU receives such control packets and records the IP addresses encoded in the payload of the received packets in a table referred to as Backward Traffic Table (BTT). At the end of each interval, the BU forwards the recorded IPs in that interval (*i.e.*, a copy of BTT) to the FU and resets BTT for the next interval.

2. Forward Unit: The FU maintains a table referred to as Forward Traffic Table (FTT) to keep track of the sources of the incoming traffic. For each incoming packet other than marking its PRID in the 16 bit ID field of the packet, a PR inserts (or updates) an entry in its FTT which is indexed by the SIP of the incoming packet. The fields of a FTT record along with their meaning is presented in Table 4.2.

Table 4.2: Details of a FTT record

NAME	DESCRIPTION
Count	At any time it represents the total number of packets sent by SIP. This field is incremented every time a packet is seen from the corresponding SIP.
IsGood	Represents whether the corresponding SIP is verified of not.
LastGoodAt	Represents the interval number at which the most recent packet is received from the corresponding SIP when the SIP is already validated.
LastBadAt	Represents the interval number at which the most recent packet is received from the corresponding SIP when the SIP is not validated yet.
StartedAt	Represents the interval number at which the first packet from the corresponding SIP was seen.

Let us now see how an incoming packet is processed by a PR before it is forwarded to the next hop. A PR splits time into intervals of fixed width and assigns a monotonically increasing value to each interval. For each incoming packet, a PR checks its FTT using the SIP of the packet. If there exists no entry against the SIP (indicating a new communication), a new record is inserted against the SIP. The *Count* field is set to 1, *IsGood* field is set to 'false' indicating that the SIP is not yet verified, the *LastGoodAt* field is set to null and the *LastBadAt* and *StartedAt* fields are set to the current interval number. For each such new communications, the SIP is maintained in a list referred to as Current Arrival List (CAL). The CAL represents a list of SIPs which are seen by the PR in the current interval but was not seen in at least last T intervals. Here T is a user specified value which represents time out period of an entry in the FTT.

On the other hand, if the SIP lands in an existing record in the FTT then the *IsGood* field of the corresponding record is checked. A false value here indicates that the SIP is not verified by the PR. A PR verifies a communication at the end of each interval and hence *IsGood* is left as it is in this case. The *Count* field is incremented and the *LastBadAt* is set to the current interval. If *IsGood* is true then the *LastGoodAt* field is checked which gives the interval at which the most recent packet from the SIP was received. If the difference between the current interval and this field value is greater than the threshold T then the packet is considered and processed as a new communication. If the difference is less than T the *Count* field is incremented and also the *LastGoodAt* field is set to the current interval.

At the end of each interval the FU checks each entry in the CAL against the BTT forwarded to it by the BU. For each match the corresponding entry in the FTT is updated with *IsGood*=true and *LastGoodAt* as the value of *LastBadAt*, which represents the interval at which the last packet from the SIP was seen by the PR. Also the SIP is removed from the CAL. It might happen that some of the outgoing traffic generated by the incoming traffic in the interval CI_i might only appear at CI_{i+1} or in general $CI_{i+\epsilon}$, where ϵ is the maximum number of intervals that can elapse between a forward packet and its corresponding backward packet. Thus if an SIP from the CAL is not found in the backward list then the difference between the current interval and the *StartInterval* is calculated. If the difference is less than a threshold ϵ then the communication is allowed until the next interval, else the SIP involved in the communication is detected as an attack source. Once an attack source is detected we stop further traffic from the source. To achieve this, the FU maintains a table referred to as Black List Table (BLT). Whenever an IP is detected as malicious, it is pushed into the BLT and the corresponding entry from the FTT is removed. We now use BLT as a filter in the FU. As soon as a packet reaches the edge router, the corresponding FU checks the SIP of the packet against BLT. If the SIP is already black listed then the packet is dropped else it is processed as explained before. Figure 4.6 shows the architecture of the defense system in a PR.

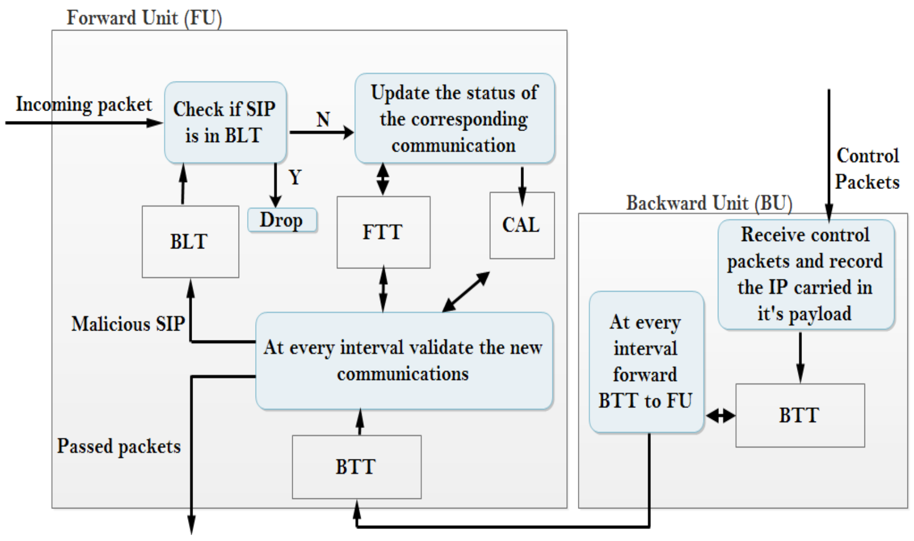

Forward Unit (FU)
Incoming packet
Check if SIP is in BLT
N
Update the status of the corresponding communication
Y
Drop
BLT
FTT
CAL
Malicious SIP
At every interval validate the new communications
Passed packets
BTT
Control Packets
Backward Unit (BU)
Receive control packets and record the IP carried in it's payload
At every interval forward BTT to FU
BTT

Figure 4.6: Defense system architecture

4.2.3 Implementation Details of DDM

To check how the above mentioned model works under normal as well as attack condition, we developed a program consisting of two units controlled by two separate threads. One thread executes the Forward Unit (FU) and the other one executes Backward Unit(BU). Periodically, the BU passes a copy of BTT to the FU using inter thread communication technique. In the FU, we need Black List Table(BLT), Forward Traffic Table(FTT) , Backward Traffic Table(BTT) and Current Arrival List(CAL). In case of BLT and BTT, we simply want to query whether an IP address is present in the list or not. We do not need to store any other information, not even the IP itself. Thus both these list are implemented as Bloom Filters[58, 139]. On the other hand the FTT keeps track of a set of values against an IP. This table is thus implemented as a Hash Table. At the end of each interval the entries in the CAL is read and processed one at a time, which is served well by a simple list structure.

4.3.3.1 Estimation of the Size of the Data Structures

Since we are using bloom filters and hash tables to store and retrieve our information, it might happen that we get false positive due to their probabilistic nature. We discuss here how one can set the size of such tables so that the false positive rate is below some tolerable threshold.

- Hash Table: We have used an universal hash function for our hash tables. The false positive of a hash table is caused by collision rate. The collision rate on the other hand depends on the load factor of the hash table and can be stated as

$$\alpha = \frac{n}{m} \tag{4.1}$$

  where α is the load factor of the hash table, n is the number of entries and m is the number of slots in the hash table. Thus to estimate the size of the FTT we first need to estimate the number of entries which the table has to record at any time. The following equation gives the estimation:

$$n = avgLifeTime \times ar \tag{4.2}$$

  where $avgLifeTime$ is the average life time of a communication, ar is the arrival rate of new IP into the network (as explained earlier a completely new IP and an IP appearing after T intervals both are considered as new to the system). The arrival rate in AUK-VIII and WAIK-VIII is approximately 10 IPs per second (can be seen in Figure 4.9(b)). The $avgLifeTime$ of a communication in AUK-VIII and WAIK-VIII is approximately 60 seconds. From Equation (4.2) we get n as 600 for both the traces. To keep the false positive rate low we set $\alpha = 0.01$. Thus from Equation (4.1), under normal condition number of slots (m) in FTT will be 60000. From 4.2 we see that a FTT record maintains 5 fields. Considering each field size as 16 bit, the memory needed by FTT under normal condition turns out as 4800000 $bits$ which is $< 5MB$.

- Bloom Filters: As mentioned above, both the BTT and BLT are implemented as Bloom Filters. A Bloom filter is a space-efficient but probabilistic data structure that supports membership queries against a set of elements. To represent and support membership queries for a set $X = \{x_1, x_2, ...x_n\}$ of n elements, a Bloom filter uses m bits and a set of k hash functions, $H\{h_1, h_2, , , , h_k\}$ with range $(1, 2, ...m)$. For theoretical analysis it is assumed that each hash function spreads the n items in the set uniformly over the range $(1, 2, ...m)$, independent of each other. For practical implementation we chose the hash functions from a family of universal hash function. For each element $x_j \in X, j = 0, 1, ...n$, $m[h_i(x_j)]$ is set to 1, where $i = 0, 1, ..k$. For a membership query of an element y, if $\exists h_l \in H, l = 0, 1, ..k$ such that $m[h_l(y)] = 0$, then the element y is certainly not there in the set. If $\forall h_l, l = 0, 1, ..k$ $m[h_l(y)] = 1$, then we assume that the element is a member of the set with a certain probability of being wrong, referred to as $false\ positive\ rate$. Let $X = \{x_1, x_2, ...x_n\}$ be the set of n elements, m be the number of bits in the Bloom filter, k is the number of hash function used by the Bloom filter and fpr is the maximum tolerable false positive rate of the Bloom filter then the following equations can be used to show the relationship among the parameters.

$$k = ln\frac{m}{n} \tag{4.3}$$

$$fpr = (0.6185)^{\frac{m}{n}} \tag{4.4}$$

From Equation (6.2), we see that the $false\ positive\ rate$ of a Bloom filter decreases if bits per element *i.e.*, $\frac{m}{n}$ increases. Also, from Equation (6.1) we see that the number of hash functions in a Bloom filter increases logarithmically with respect to $\frac{m}{n}$.

The number of entries in the BTT will be proportional to the number of entries in the FTT. Considering fpr as 0.01, $n = 600$ from Equation (6.2) we get $m < 6000\ bits$ or $m < 6\ KB$. On the other hand the size of the BLT depends on the number of SIP being blacklisted during an attack. For our experiments we use $n = 100000$ for the BLT. Considering fpr as 0.01 from Equation (6.2) we get $m < 10\ MB$.

4.3.3.2 Possibility of Attacking the Defense System Itself

Another point of interest is that whether the attacker can fill these tables or not to launch a DDoS attack at the defense system itself. First, consider the BTT maintained by the BU which it updates to the FU periodically. This table is fed by the control packets from the *bi-agent*s which are generated when the *bi-agent* sees a valid bidirectional packet towards the requesting source. Thus this table cannot be populated by the attacker as attack traffic are unidirectional in nature. On the other hand, the FTT maintained by the FU has to keep track of a source for a maximum of ϵ intervals before it can be verified as either good or bad. Thus if a source is malicious it will be detected eventually and will be blacklisted. The attacker will eventually exhaust all its bots within a certain time period which depends on the number of bots exposed per interval of time. So, it turns out that the attacker will simply reduce its attack length in an attempt to exhaust the FTT. Also in section 4.4.3, we will see that once the attack is detected by observing large black listing by the FU, the mitigation system gets activated and filters most of the new attack traffic at the PR without needing them to process and store by the FTT in the FU. Thus, after detection of the attack, the attacker has less control over the FTT table to blow it up. That way the filling of the BLT by the attacker also becomes difficult as the mitigation system allows less number of attackers after attack detection.

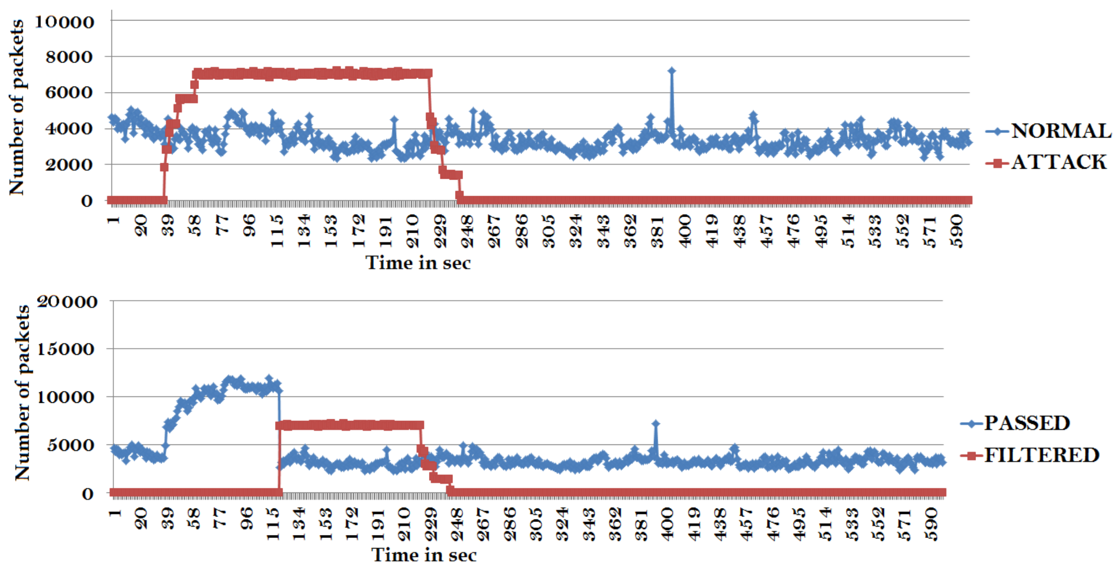
4.3 Experiments and Results

In this section, I mention the experiments performed to evaluate our defense system. To perform the experiments AUK-VIII and WAIK-VIII network traces are used as normal traffic reference. In the experiments the time interval is set to 60 seconds. The value of ϵ is set to 2 intervals.

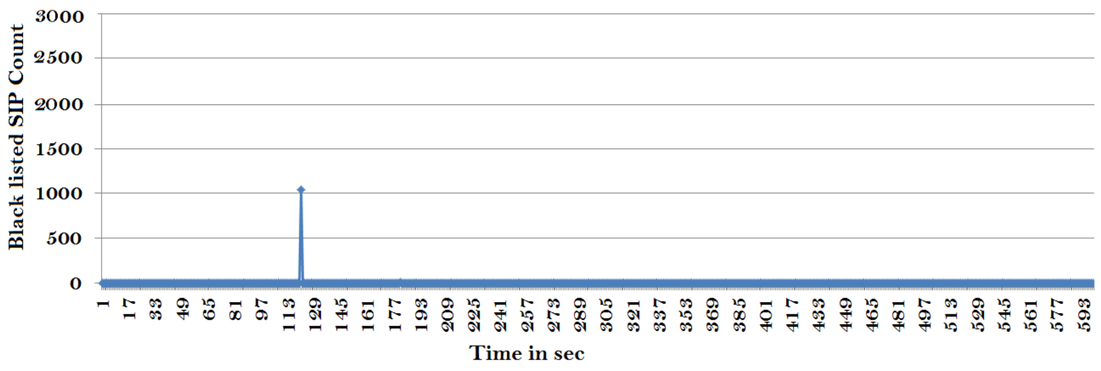
4.3.1 Experiment 1

DDoS attack with fixed number of sources: For this experiment UDP attack packets are generated mimicking a DDoS attack performed by 1000 bots using TUCANNON. The attack was performed for 3 minutes. As the defense system is not making use of protocol specific information to detect and mitigate an attack, TCP traffic would also have served the purpose here.

Figure 4.7(a) shows the attack traffic as well as the normal background traffic. In the top graph (which are seen by the FU as a single stream of packets) and the bottom graph shows the separation of the input stream into attack traffic and normal traffic by the FU, marked as $FILTERED$ and $PASSED$ respectively. We can see that for the first 2 minutes of the attack the entire attack traffic was allowed by the defense. After that the last 1 minute of the attack is detected and mitigated accurately. It happens because when the attack starts the FU records all the new sources and waits for ϵ=2 intervals for them to verify. As the attack sources did not verify at the end of the 2^{nd} interval, all of them is detected as malicious and thus marked as blacklisted and any further traffic from them is blocked by the blacklist filter. The fact can be seen in Figure 4.7(b) which shows a large number



(a) Attack vs normal traffic detected by the system



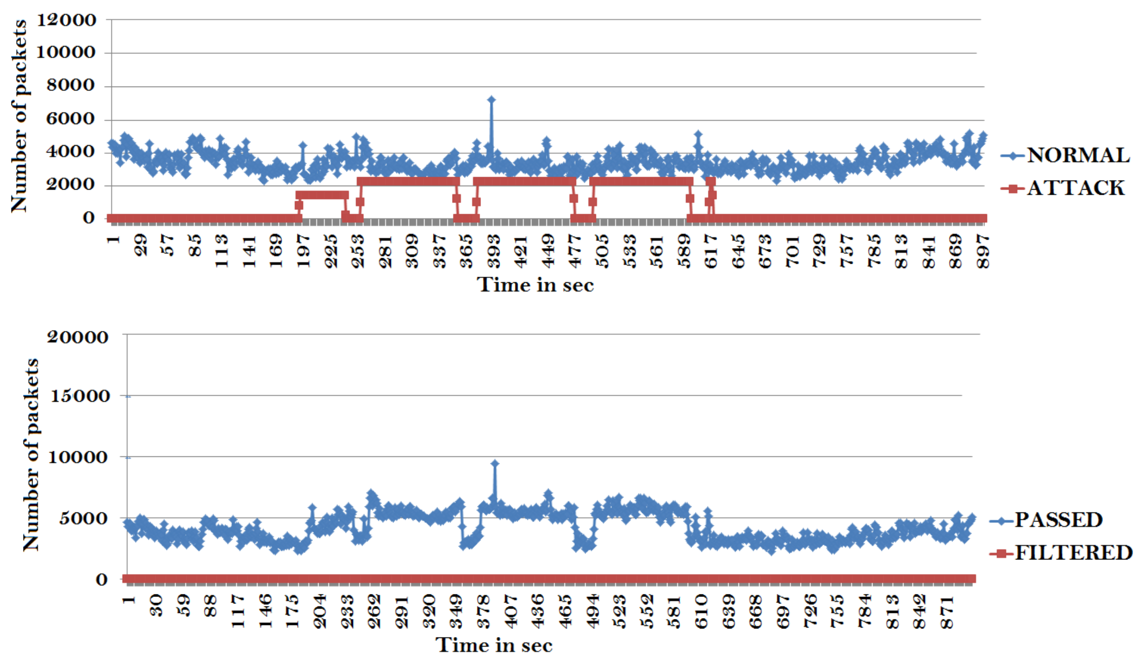
(b) Black listed SIP count per interval

Figure 4.7: $Experiment1$ result

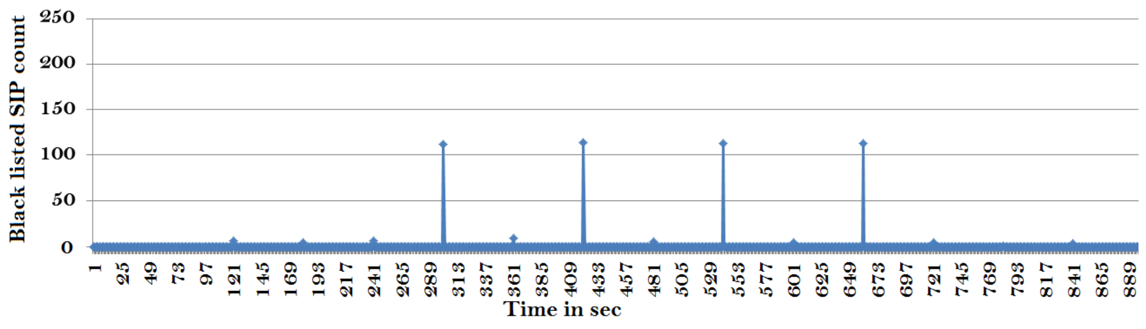
of blacklisting at around 118^{th} sec. Suppose the attacker has only a few thousands of bots to carry out the attack In that case the entire attack will be stopped by our defense system within a short interval of time. The situation of having a fixed number of attack sources can be seen in DRDoS attack. The attacker typically uses thousands of reflectors to generate the attack traffic. Also, the attacker does not have any control over the SIP field of the attacking packets. Thus, such a DRDoS attack will be detected and stopped by our defense immediately.

4.3.2 Experiment 2

DDoS attack with changing sources: Suppose, instead of exposing the entire botnet at a time, the attacker activates a fraction of it. TUCANNON is used to generate the attack traffic for the experiment where, each attack thread generates attack traffic using a fixed SIP. The SIP of the attack threads are changed periodically to mimic change of attacking groups in the attack. The total number of threads (over all the machines) are approximately 100, which is the number of active attack sources in an interval.



(a) Attack vs normal traffic detected by the system



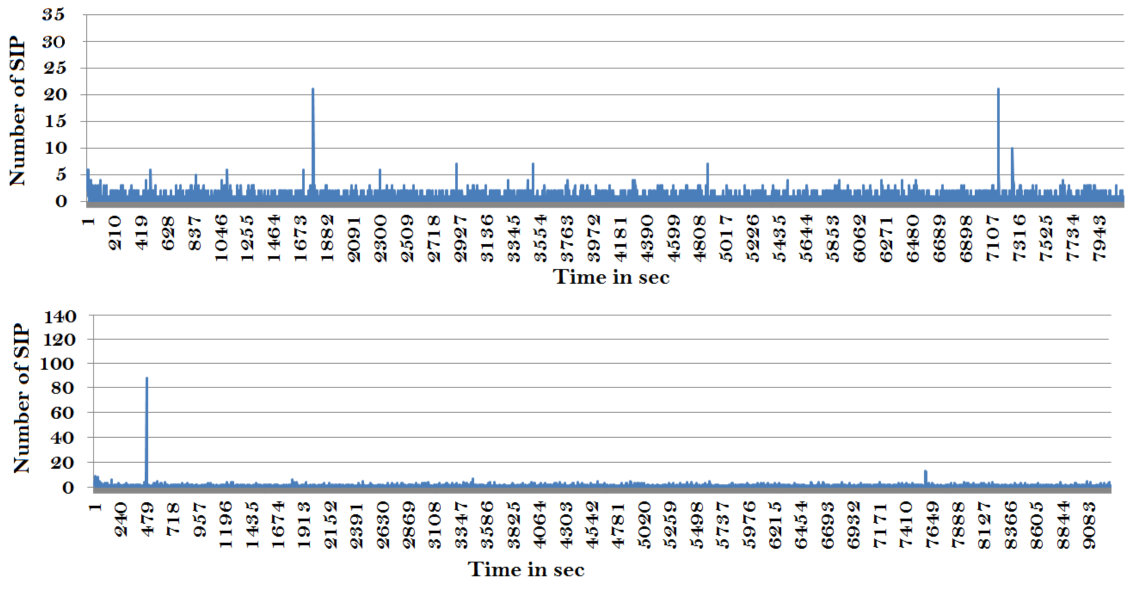
(b) Black listed SIP count per interval

Figure 4.8: *Experiment*2 result

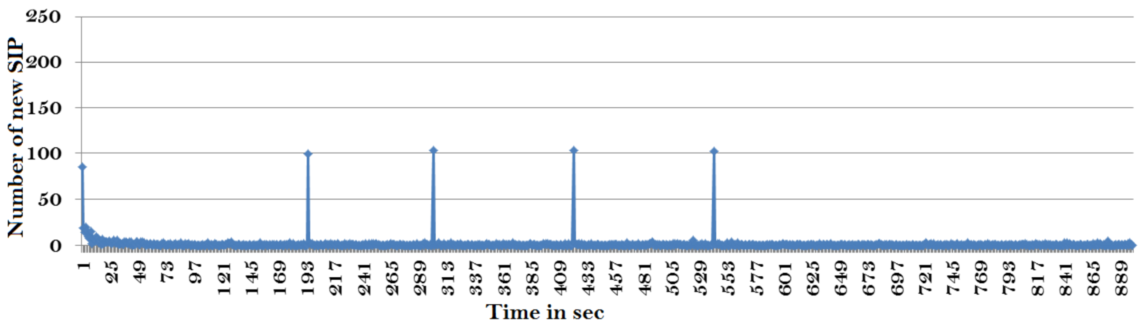
Figure 4.8(a) shows that the entire attack traffic pass through the defense system. Due to the changing nature of the attack sources the defense system is not capable of filtering out any attack traffic. If the size of the botnet (or the number of reflectors) is small, then to keep the attack alive for a longer period of time the attacker has to activate a small number of zombies at a time which makes the attack less effective. However, the attacker can try to inject attack packet at high rate to occupy the victim bandwidth. But such an attempt will make the attack source more visible to the defense system. The *count* field of an FTT record can be used to detect such high rate attack sources. However, if the attacker has a large botnet then it can activate a large size group at a time and thus can cause harm to the victim. Figure 4.8(b) shows that from the second interval from the starting point of the attack, a large number of sources are continuously blacklisted by the FU. *i.e.*, at every interval the attack is detected but due to the changing nature of the attack sources the defense system is not capable of filtering out any attack traffic. Here we will see how we can enhance the defense system to prevent such attack where the attack sources are changed periodically revealing a fraction of a large botnet. The FU maintains a list of SIPs participated in one or more valid communications through the PR in the past, referred to as Good List (GL). The duration of the history period could be from hours to weeks or larger than that. When the FU verifies a communication as 'good', its source IP is pushed into the GL. Also, the FU monitors the number of arrivals per interval. The number of arrivals per interval is nothing but the size of CAL at the end of an interval. Figure 4.9(a) shows the arrival of new SIP per interval in AUK-VIII and WAIK-VIII traces. Figure 4.9(b) shows the arrival per interval of the input stream prepared in experiment 2 with the consideration of AUK-VIII as the normal background traffic.

We see a stable arrival rate over a period of time for the normal traces with occasional bursts of arrival. However, during the attack the arrival of new SIP per interval goes high significantly. The FU considers a high blacklisting as an attack alarm and if in the subsequent intervals after the alarm contains burst of arrival, it activates its mitigation system. To achieve this the FU performs the following

- The FU triggers an alarm referred to as *AttackAlarm* as soon as it sees a high black list count.
- When the *AttackAlarm* is on and the size of CAL is also larger than a threshold then the FU activates a filter referred to as *GoodFilter* which allows only those SIP recorded in the GL into the network. The deactivation of the *GoodFilter* is done by observing the drop count by the *GoodFilter*. If the



(a) Arrival of SIP/interval in AUK-VIII and WAIK-VIII traces.



(b) Arrival SIP/interval in the attack trace considering AUK-VIII as background normal traffic.

Figure 4.9: Arrival rate of SIP/interval

count is less than a threshold the filter is deactivated by the FU.The modified FU is shown in Figure 4.10.

To study the behavior of the defense with this modification, experiment 3 is performed.

4.3.3 Experiment 3

DDoS attack mitigation based on SIP history: In this experiment the first half of AUK-VIII trace is used to fill the GL. The duration of the history is thus 90

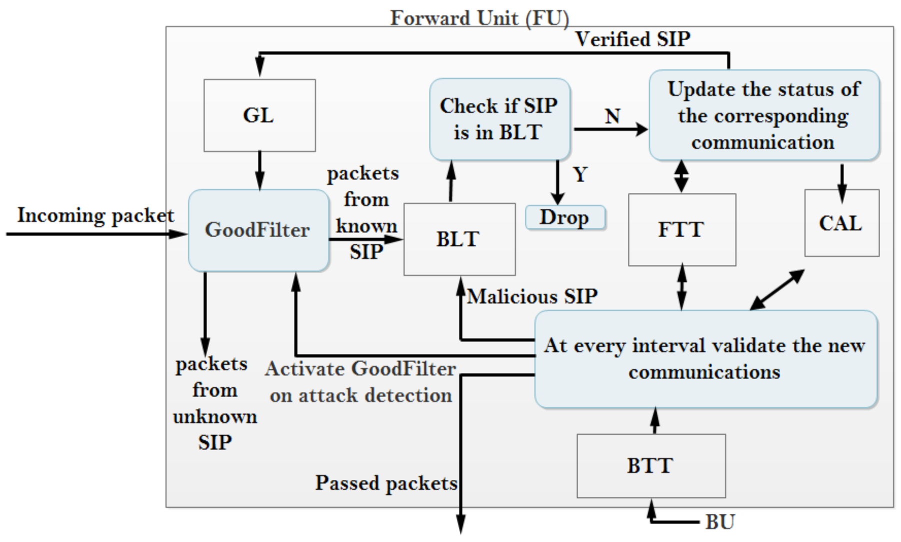


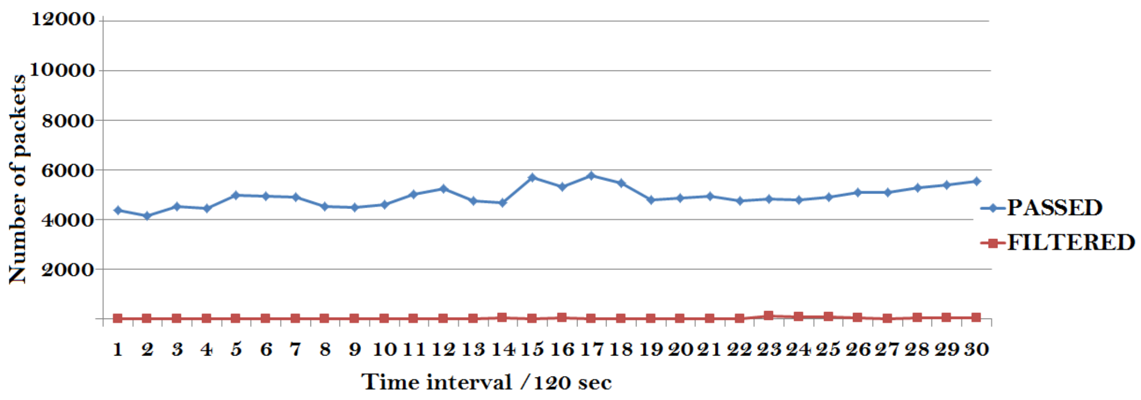
Figure 4.10: Modified FU

minutes. Figure 4.11(a) shows the passed and filtered traffic volume when the second half of AUK-VIII trace is filtered against the GL created from the first half of the trace. We observe that only a fraction of normal traffic is blocked by the *GoodFilter*. On the other hand, Figure 4.11(b) shows the output of the FU when the attack traffic generated in experiment 2 is merged with the second half of the trace, which ensures that most of the attack traffic is blocked by the *GoodFilter*.

Thus, even if the attacker possesses a large botnet and continues to attack with different bot groups at a time, our defense can mitigate the attack successfully. The effectiveness of the *GoodFilter* certainly depends on the history duration. As the history goes longer over time, the number of entries in the GL will also go high. To estimate the size of the *GoodFilter* for normal traces, we plot the number of completely new SIPs over time for AUK-VIII and WAIK-VIII traces. Figure 4.12 shows that the rate of new SIP to the GL goes down over time. We recorded around 1000 SIP in each of AUK-VIII and WAIK-VIII traces in 2 hours. Since GL is used only to check the membership of an entry it is implemented as a bloom filter. Considering false positive rate as 0.001 and number of potential SIP as 2000, from equation (6.2) we see that for such a GL will require $< 30\ KB$ of memory.

4.3.4 Experiment 4

DDoS attack with spoofed SIP: Until now we were assuming that the source IPs used in the attack traffic were not spoofed. Now, if we remove that constraint, the attacker can use almost all the 2^{32} IP addresses as the SIP of the attack packets.



(a) Filtering 2nd half based on 1st half of AUK-VIII trace

where τ_N = $detection\ time$ ρ_N = $normalized\ detection\ time\ after\ a\ change\ occurs.$ Inf represents $infimum$, n is the time when the attack started, N is user defined threshold value

(b) Attack vs normal traffic detected by the system

Figure 4.11: *Experiment*3 result

Let us discuss the consequences when each bot generates attack packets with randomly spoofed SIP. The defense system mitigates the attack in the same way as in Experiment 3, as shown in Figure 4.11(a). The FU will detect the set of spoofed SIPs used in interval $a + i$ at the end of $a + \epsilon + i$ interval, where a is the interval, when the attack started and $i = 0, 1, 2, ..n$. This will trigger the *AttackAlarm*. Since the arrival rate at each interval given by the size of the CAL remains high in the subsequent interval due to the randomly spoofed SIP used in the packets, the *GoodFilter* will eventually get activated to mitigate the attack. But since the attacker has the entire range of source IPs, he/she can perform the attack for a longer period of time in an attempt to make the *GoodFilter* obsolete and drop packets from legitimate but unknown users of the victim.

4.4 Discussion

In this chapter, I propose and demonstrate an effective DDoS attack detection and mitigation system(DDM) based on the observation that most of the Internet communication is bidirectional in nature. Experimental results show that the proposed

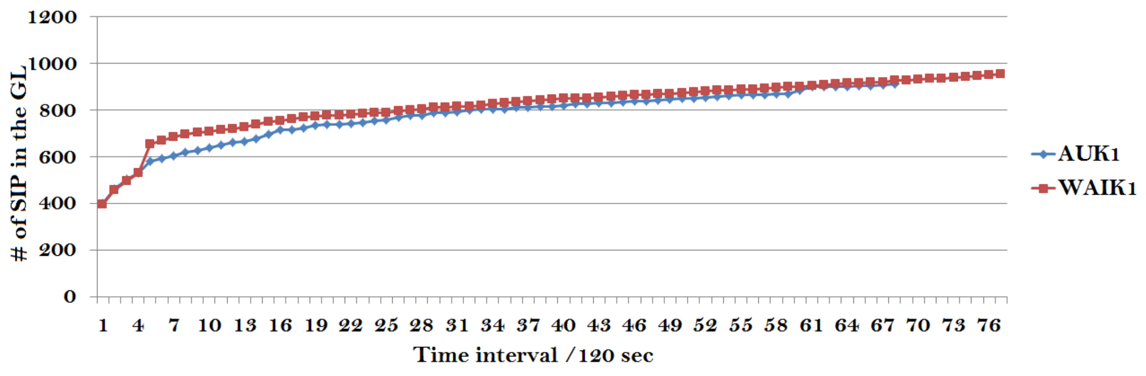


Figure 4.12: Entries in the GL over time

detection and mitigation system can defend against many different types of DDoS attacks. DDM is a victim end DDoS defense technique which, detects an attack by observing a high volume of unidirectional flows. DDM mitigates an ongoing DDoS attack by blocking all the subsequent packets from the SIPs which are detected as unidirectional. DDM achieves this by maintaining a *black list* of malicious SIPs. DDM also maintains a database of SIPs which have participated in a bidirectional communication with in a fixed duration history period, ranging from hours to days or even months. DDM performs effectively against non spoofing attacks such as DRDoS where, the attack packets received by the victim always carry the genuine SIPs of the senders of the packets. However, spoofing can be used by the attacker to degrade the performance of DDM. Spoofing can be of two types.

1. *Random spoofing:* In random spoofing, the attacker changes the SIP addresses of the attack packets randomly and dynamically. In such a situation DDM can not filter the attack packets using its *black list*, as most of the SIPs appear for a very short duration. However, DDM can detect the presence of the attack by observing the growth of the number of such short lived unidirectional communications. DDM mitigates this situation by activating a filter, which allows packets only from known SIPs present in the *white list*. This remedy however, degrades the performance of DDM, as an unknown SIP always might not carry an attack packet. I.e., DDM drops all legitimate packets coming from one or more unknown SIPs during an attack.

2. *Selected spoofing:* In selected spoofing, the attacker spoofs the SIPs of the attack packets with the SIPs of one or more specific sources. For example, an attacker can spoof the attack packets with a set of selected SIPs which are know to the victim. In such a situation, DDM can not distinguish an attack packet from a benign packet if both the packets carry the same SIP.

To address these issues, in the next chapter, I discuss about techniques which can distinguish a spoofed packet from a benign packet carrying the same source IP in their IP headers.