AEGIS: An Active-Network-Powered Defense Mechanism against DDoS Attacks

Eric Y. Chen

NTT Information Sharing Platform Laboratories
1-1 Hikarinooka, Yokosuka-shi, Kanagawa, 239-0847, Japan
eric@nttmhs.tnl.ntt.co.jp

Abstract. Distributed Denial of Service (DDoS) attacks are a pressing problem on the Internet as demonstrated by recent attacks on major e-commerce servers and ISPs. Since their threat lies in the inherited weaknesses of the TCP/IP, an effective solution to DDoS attacks must be formulated in conjunction with a new networking paradigm, such as Active Networks. In this paper, we introduce a conceptual framework called Aegis, which we propose as a defense mechanism against DDoS attacks. The core-enabling technology of this framework is the Active Network, which incorporates programmability into intermediate network nodes and allows end-users to customize the way network nodes handle data traffic. By introducing Aegis, we also wish to demonstrate some of the new possibilities that the Active Networks can offer.

1 Introduction

Since February 2000, when a number of major commercial web sites such as Yahoo, CNN.com, E*TRADE, eBay, Buy.com and ZDNet were attacked and rendered useless for a period of time by DDoS (Distributed Denial of Service) attacks, the word 'DDoS' has become part of the active vocabulary of most Internet users. The concept of DDoS attacks had been known for some time by networking experts, who had warned of their implied danger long before these incidents occurred. Until last year, however, most companies had treated DDoS as merely theoretical, and were willing to wait until something happened before taking any action. Today, DDoS is undoubtedly a pressing problem on the Internet, and its potential impact has been well demonstrated. Since its threat lies in the inherited weaknesses of the TCP/IP, an effective solution must be formulated in conjunction with a new networking paradigm, such as Active Networks.

In this paper, we will introduce a conceptual framework called Aegis, which we propose as a defense mechanism against DDoS attacks. This framework has been designed on top of the Active Networks technologies, which incorporate programmability into intermediate network nodes (routers or switches) and allow end-users to customize the way network nodes handle data traffic.

The remainder of this paper is organized as follows: Section 2 introduces some different types of DDoS attacks and existing countermeasures; Section 3 lists a number of our system requirements for the underlying platform that we intend to

I.W. Marshall, S. Nettles, and N. Wakamiya (Eds.): IWAN 2001, LNCS 2207, pp. 1-15, 2001.

exploit in defending against DDoS attacks; Section 4 explains Aegis, our proposed framework, in detail; Section 5 outlines our future work and identifies a number of known issues concerning Aegis; and Section 6 concludes this paper.

2 DDoS Attacks

2.1 Types of Attacks

A Distributed Denial of Service (DDoS) attack is, as its name applies, a distributed form of Denial of Service (DoS) attack. A DoS attack is characterized by the deliberate act of sending a flood of malicious traffic to a server, thus depriving it of its resources so that it becomes unavailable to other legitimate users. While there are many different tools available to launch DoS attacks, the attacks themselves can be divided into two basic types.

The first type of attack aims to crash the server OS or starve the server's system resources, such as its CPU utilization, file storage or memory, by exploiting flaws in the server software, security policy or TCP/IP implementation. Common examples of this type of attack include SYN Flood, IP Fragmentation Overlap, Windows NT Spool Leak and Buffer Overflow [1]. Prevention mechanisms against this type of attack, such as SYN cookie, have been well developed by various software vendors, and so system administrators can defend their sites by upgrading their server software or firewalls.

The second type of DoS attack is rather more primitive, but also far more insidious. This time the attackers do not care what software the victim is using. Instead, they simply try to consume all available network bandwidth of the target network by bombarding it with massive amounts of traffic. Well-known examples include Smurf, Fraggle, ICMP Flood and UDP Flood [1]. Most DDoS attack tools, such as TFN, TFN2K, and Stacheldraht, intensify these bandwidth-consuming DoS attacks by launching them from multiple sources [2]. We have been particularly interested in this second type of attack because to date no existing technologies have been able to effectively tackle this problem. Aegis, the system we are proposing, focuses on defending against this type of attack.

2.2 Existing Solutions and their Weaknesses

A complete countermeasure against bandwidth-consuming DDoS attacks involves five stages - prevention, detection, first response, traceback and second response. Most current work related to DDoS attacks attempts to address the problems occurring in these individual stages.

In the prevention stage, we want to stop DDoS attacks from being launched in the first place. This can be achieved by two means. One is to have all Internet users look for the presence of DDoS daemons in their own machines by using some scanning tools. Although this method prevents DDoS attackers to exploit unprotected hosts for malicious purposes, getting such large-scale cooperation from all Internet users is a huge bottleneck. Another prevention method is to use Cisco's Ingress Filter [3],

which is designed to be installed and configured in every edge router to verify the legitimacy of each outgoing packet's source address. Packets with IP prefixes not matching the edge router's network will not be allowed to go out. This technology prevents DDoS attackers from using forged source addresses. Ingress Filters have two major weaknesses: 1) at present not all edge routers are configured to enable this function. If the attacker's edge router does not have the Ingress Filter enabled, once packets with forged source addresses pass the edge router successfully, they are almost impossible to catch; 2) Ingress Filters do absolutely nothing to prevent non-spoofing flood. Unlike attacks that aim to consume server resources, bandwidth-consuming attacks are destructive even without spoofing source addresses.

Because the current technologies on preventing DDoS attacks are limited, detection mechanism against DDoS floods plays an important role. In the detection stage, we want to distinguish 1) a DDoS flood from a high peak of regular traffic, and 2) malicious packets from legitimate packets. This may be achieved by identifying a number of anomalies such as sudden bursts of traffic, oversized ICMP and UDP packets, or large number of packets with identical payloads. A number of NIDSs (Network Intrusion Detection System) have been developed to monitor abnormal traffic. One major weakness with most NIDSs is that they can cause “collateral damage”. In other words, legitimate packets may be mistakenly treated as malicious packets.

Upon the detection of a DDoS attack, the next thing we should do is to bring our server back to the Internet to allow access by legitimate users. We call this the "first response" stage. Currently, the only option at this stage is to contact the ISP and ask them to filter out the malicious traffic before it clogs our bandwidth. This solution has two drawbacks: 1) the process of getting help from an ISP can take hours or even days, allowing the attack to continue [4]; 2) in theory, the attacker can further consume all the ISP's bandwidth by increasing the size of flood.

After we have applied the first-aid patch, we want to find out where the attack is coming from. If the source addresses of the attack traffic are spoofed, we can attempt to trace them back to their origins by examining log in each router hop-by-hop. The weaknesses with this approach are: 1) it is a slow manual process; 2) this process can be easily thwarted if one or more routers along the path do not have the facility for identifying the upstream source. Recently, more promising traceback technologies using some packet-marking mechanisms have been proposed by [5] and [6].

During the last stage, the "second response" stage, there is very little that current technology can offer in the way of help. The hosts that are generating the flood are unlikely to be the attacker's own hosts, but hosts that have been hacked and exploited. Therefore, even if we obtain the IP address of each host participating in the attack, we are left with only two options: 1) to contact the owner of each host directly and inform him that his computer might have been hacked and is participating a DDoS attack; or 2) to contact the ISP of each host and ask them to block the flood at the uppermost stream possible. Since most DDoS attacks involve hundreds or thousands of hosts, neither option is practical.

The Aegis system is designed as an integrated solution to address issues pertaining to four stages: detection, first response, trace back and second response (Fig. 2). The Active Networks paradigm offers us a new degree of freedom to respond to malicious traffic automatically and effectively.

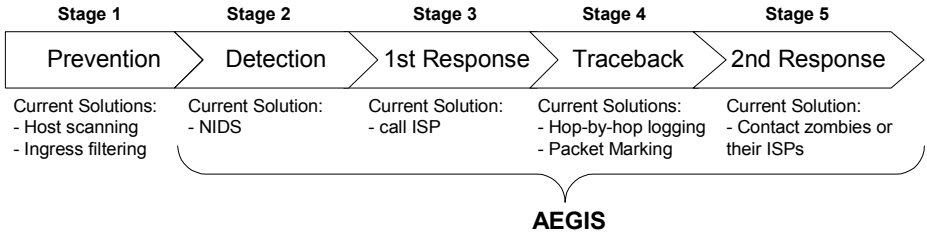

Fig. 1. Scope of Aegis.

3 Active Networks and Our System Requirements

We have chosen the Active Networks paradigm as the enabling technology for the Aegis system because of its potential and flexibility. Currently there are three major test beds available: ABone in the US [7], FAIN in Europe [8] and JGN in Japan [9]. Various EEs (Execution Environments) have been proposed and are often categorized into three groups [10]: Active Packets (such as Smart Packets [11] and Active IP Option [12]); Active Nodes (such as ANTS [13] and DAN [14]); and Hybrids (such as SwitchWare [15] and Netscript [16]). Although Aegis is not designed for any specific EE, our system requirements for the underlying routers include the following:

1. New functions or network services implemented in standardized modular form can be loaded into routers and executed at runtime. We call these modules active code. Active code can act autonomously and move among routers as mobile agents do among end systems.
2. Each piece of active code must have a specific owner and must be authorized to have full control over all packets associated with its owner. Let "I" denote the entire name space of global IP addresses and (s,d) denote a packet with source address s and destination address d. Every piece of active code residing in an active node belongs to a specific user, who owns a set of IP addresses, denoted as "O". Each piece of active code has access rights to packets $\{(s,d) \in [(OxI) \cup (IxO)] | s \neq d\}$ that are received by the active node. (i.e. IP packets in which the owner's IP addresses appear in the destination or source field). Library functions such as capturePacketBySrc(), capturePacketByDest(), capturePacketAssociatedWith() must be provided by the EE. Active code may then monitor, discard or modify these packets. This requirement has been proposed in [18].
3. Incoming packets with specified IP addresses can trigger active code. No proprietary headers, such as one proposed as ANEP [18], are required to distinguish the so-called "Active Traffic" from the "Passive Traffic". This requirement is crucial since most attackers would not send packets with these proprietary headers. In accordance to this requirement, a routing table in an active node would look like the one illustrated in Figure 2. An incoming packet is either dispatched to a specific active code residing in the same node, or routed to a neighboring node. In this routing table, entries associated with active code

would always precede ones for regular traffic, and would be applied only to incoming packets (not outgoing ones, so that the processed packets could be forwarded to the next hop without causing loops).

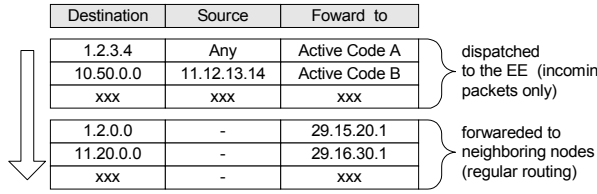

Fig. 2. Routing Table for Active Nodes.

4. Any given active node must have the knowledge of all neighboring active nodes. Note that a neighboring active node may not be the next-hop router, but it may be two or three routers away.
5. Each active node supports class-based queuing (CBQ) for outgoing packets.

4 Aegis

4.1 The Basic Concept

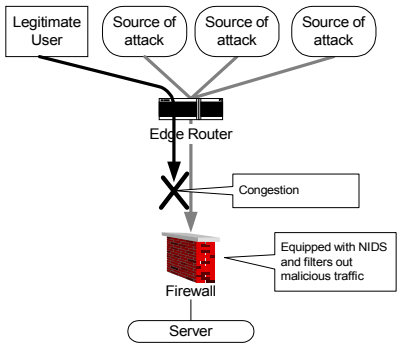

Fig. 3. Conventional Firewall.

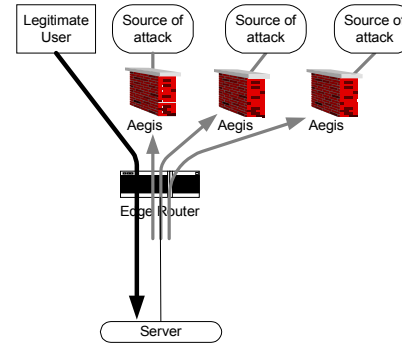

Fig. 4. Aegis.

Fig. 3 illustrates how a conventional stationary firewall would attempt to fend off a DDoS attack. Assuming that the attackers have been successfully distinguished from the legitimate users, the firewall may effectively free up the server's computing resources by blocking floods from the attackers. However, if the attackers aim to consume the victim's network bandwidth, congestion is likely to occur between the ISP edge router and the victim's firewall. Legitimate visitors will still have trouble accessing the server and, as a result, the attackers will have essentially achieved their goal.

By exploiting the programmability of Active Networks, however, the filtering process can be distributed and moved to optimal locations so that the unwanted

packets can be blocked more effectively while preserving as much bandwidth as possible. Fig. 3 illustrates this concept. Instead of attempting to block all unwanted traffic at one fixed location, Aegis filters packets in the upstream in a distributed manner. As a result, the damage caused by the attack is effectively distributed and congestion is far less likely to occur.

4.2 Modeling DDoS Attacks

For the purpose of describing our work, we have generalized bandwidth-consuming DDoS attacks into the following two models:

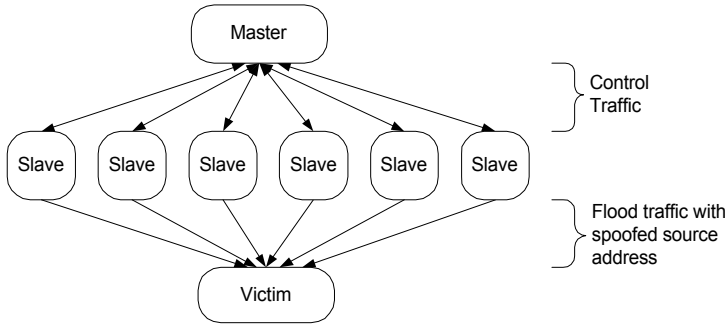

Fig. 5. Model A.

Model A: This model, illustrated in Fig. 5, includes DDoS attacks using ICMP Flood or UDP Flood. The "Master" represents the attacker's terminal, while the "Slaves" represent terminals being intruded upon and controlled by the attacker. DDoS software is often uploaded from the Master terminal to each compromised Slave terminal. Communication between the Master and the Slaves is called Control Traffic. In this model, each Slave terminal attempts to bombard the victim with massive amount of packets, sometimes with spoofed source address so that the victim is unable to determine the real source of the flood. Let μ denote the number of packets sent from a Slave per second, k the number of Slave terminals, s the size of each malicious packet in bytes and v the victim's bandwidth in Kbps. Note that while μ depends on the capability of each Slave, s is often fixed across different Slaves. The attacker can effectively knock the target victim off the Internet if

$$v < \frac{s}{8000}\sum_{i=1}^{k}\mu_i \qquad (1)$$

Now let us represent the Internet as an undirected graph INet=(V, E) where V is the set of network nodes and E is the set of physical links between the elements in V. Note that V includes both end systems and routers. Let $S \subset V$ denote the set of Slave terminals and $d \in V \backslash S$ denote the victim. Assuming that the routes are fixed, we can represent each attack path of Model A from a Slave to a victim as

$$A_i = \langle s_i, v_{i,1}, v_{i,2}, \ldots, v_{i,n_i}, d \rangle \qquad (2)$$

Each route is comprised of n_i routers and nodes $\{v_{i,1}, v_{i,2},\ldots, v_{i,ni}, d\}$. The terminal d receives packets with a source address $s_{i.}$, which may or may not be forged.

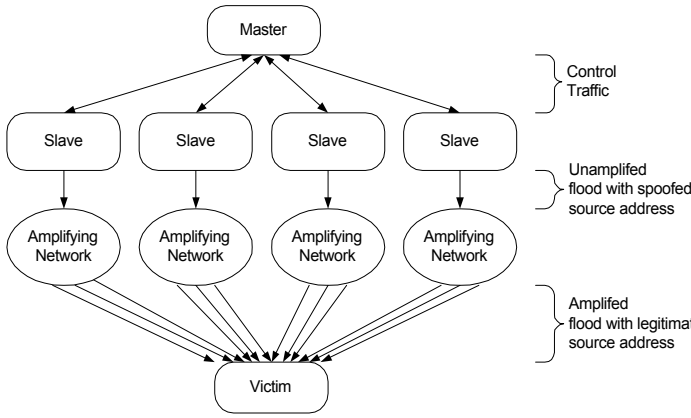

Fig. 6. Model B.

Model B: This model, illustrated in Fig. 6, includes DDoS attacks using Smurf or Fraggle. Model B is characterized by the use of amplifying networks to magnify the damage. The attacker commands each Slave to send spoofed ICMP Echo packets or UDP port-7 (echo) packets to the broadcast address of some amplifying network (shown in Fig. 7). The source address of each Echo request is forged with the victim's address, so that all the systems on the amplifying network will reply to the victim. The number of systems available to reply to Echo requests is often called the amplifying ratio (denoted as λ). Using this model, the attacker can effectively knock the target victim off the Internet if:

$$v < \frac{s}{8000}\sum_{i=1}^{k}\lambda_i\mu_i \tag{3}$$

In a similar manner, we can represent each attack path from a Slave to a victim in Model B as:

$$B_i = <s_i, v_{i,1}, v_{i,2},\ldots v_{i,mi-1}, \alpha_i, v_{i,mi+1}\ldots, v_{i,ni}, d> \tag{4}$$

The m_i^{th} router ($1 \le m_i \le n_i$) from the Slave terminal is denoted by i, which represents the edge router of an amplifying network. Each packet generated by the attacker in the subpath $< s_i, v_{i,1}, v_{i,2},\ldots v_{i,mi-1}, \alpha_i>$ carries a spoofed source address d, and the address of the victim, so that the subpath appears as $< d, v_{i,1}, v_{i,2},\ldots v_{i,mi-1}, \alpha_i>$. However, each packet originating from the amplifying network in the subpath $<\alpha_i, v_{i,mi+1}\ldots, v_{i,ni}, d>$ carries a legitimate source address due to the way in which Echo replies.

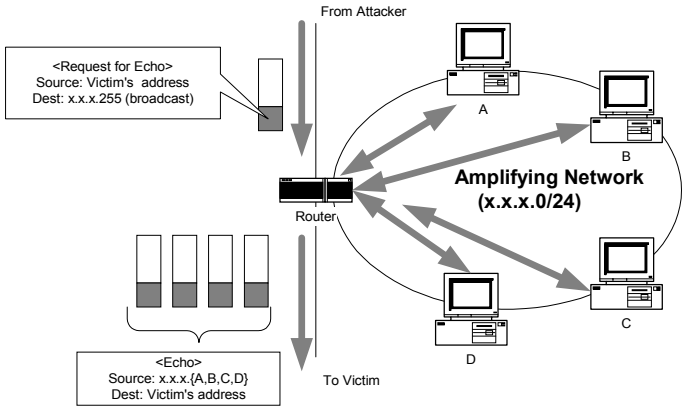

Fig. 7. Amplifying Network.

4.3 Components

The Aegis system consists of three core components: Commander, Shield and Probe. Fig. 8 depicts a possible deployment of the Aegis system on the Internet.

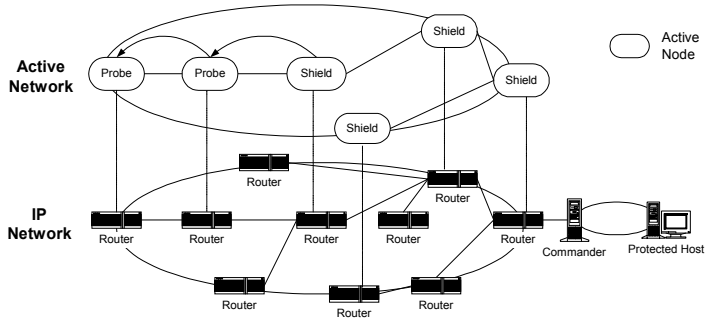

Fig. 8. Deployment of Aegis.

The Internet is given as INet=(V, E), and the Active Network that satisfies our system requirements is given as ANet=(V', E') in which V'⊂V and E'⊂E. Shield and Probe modules may reside in any node v∈ V'. Fig. 9 presents a logical view of the Aegis system. The role of each component is described in the following sections.

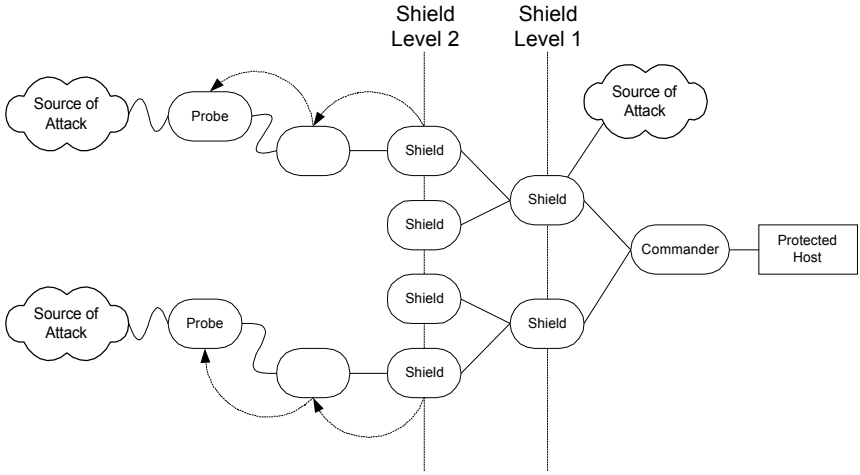

Fig. 9. Logical View of Aegis.

4.3.1 Shield

The Shields have been designed to act during the "first response" stage. The Shield modules are implemented as mobile code. They can be executed at runtime in the Execution Environments (EE) of Active Networks that satisfy the requirements listed in Section 3. Residing in an active node, each Shield module monitors the data packets flowing towards the protected host or network. This is possible through the third requirement in Section 3, which entitles these modules to full control over all packets associated with the owner of the module. Each Shield is equipped with a traffic classifier, which determines the probability that the inbound traffic is part of a DDoS flood. Packets with a high suspicion-level are pushed into low-priority output queues and vice versa. (Fig. 10) The goals of this design are twofold: 1) to increase the likelihood of normal traffic reaching the protected site under attack, while preventing suspicious traffic from doing so; and 2) to give more flexibility and precision to the DDoS detection algorithms. Determining whether unusual bandwidth consumption is caused by a DDoS attack or simply by a peak in regular traffic is not a trivial problem. Even if an attack is confirmed, separating the DDoS flood from the legitimate traffic cannot always be done with reasonable confidence. To minimize "collateral damage", the detection algorithm should not yield simple true-or-false binary output, but rather, a rating of the likelihood that certain traffic is engaged in an attack. Only packets that are extensively anomalous and suspicious should be discarded on the spot. The traffic classifier in each Shield performs two types of inspection:

Local Stateless Inspection: This type of inspection is applied to each inbound packet based on a number of static parameters. These parameters may include source/destination addresses, protocol values in IP headers, packet length, destination port numbers, the ICMP type and hashes of part of the payload. For example, if the host under protection is a web site and it is certain that the host would not send Ping

requests to any outside host, Shields should be configured to immediately discard all inbound ICMP packets with type 0 (echo reply) and all UDP packets with source port number 7. This policy is very effective in defending against Smurf and Fraggle attacks. Another example of anomaly that can be inspected statelessly is packet with identical source and destination IP addresses. Some attacks send their target hosts packets with address of these hosts as both the sources and destinations, causing these hosts to go into a loop. Shields should therefore discard all packets with such an anomaly.

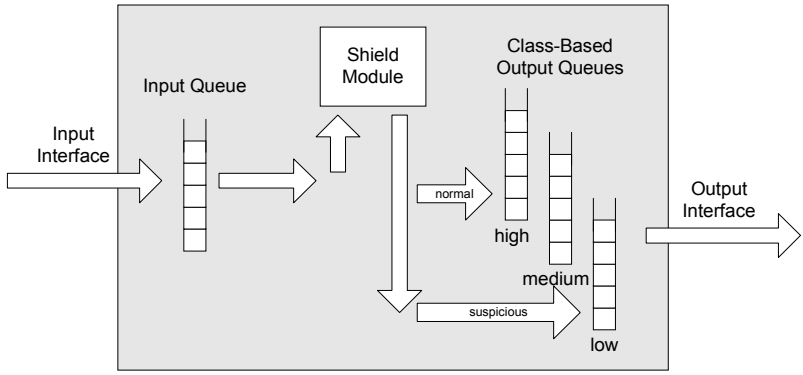

Fig. 10. Queuing Based on Suspicion Level.

Local Stateful Inspection: This type of inspection is regularly applied to the log file recorded by the Shield. The log file keeps count of certain parameters of traffic over a certain period of time in order to facilitate the detection of anomalies. For example, if there is a sudden influx of maximum-sized packets coming from the same sources over a period of 5 minutes or so, the Shield should raise the suspicion-level of packets coming from these sources and push them into the low-priority output queues. Another powerful inspection of this type has been suggested in [19], which defines packets to be malicious if they are destined for a host from which too few packets are coming back. We can set the Shields to spot disproportional packet rates.

Thus, by distributing Shield modules in the upper stream, we hope to achieve two goals: 1) to comfortably absorb most DDoS attacks and significantly decrease the likelihood of congestion; and 2) to allow sampling and inspection of network traffic at different locations.

4.3.2 Commander

The Commander acts as the central control center of the Aegis system and dispatches Shield modules outwards to the surrounding Active Nodes. The Commander performs the following operations:

Shield Level Configuration: Shield level is the depth that the Shields expand outward into the public network. The Shields are then constructed into a breadth-first tree. In Figure 7, the Shield level is set to 2, but it could be increased dynamically

when the protected site is under heavy attack. The Commander also has exclusive control over all dispatched Shield modules, and is able to terminate them at will.

Global Stateful Inspection: While the two types of inspection performed locally at each Shield can effectively block certain types of attacks, some anomalies can be detected only by analyzing an overall view of the network activity. In the initial stage of deploying Aegis, the Commander undergoes a learning stage by regularly taking a snapshot of incoming traffic and applying statistical algorithms to obtain a "fingerprint" of normal traffic patterns, which can vary from network to network depending on the purpose of the protected site. The Commander can than use this fingerprint to detect anomalies, such as a unusually high degree of randomness in the source addresses of inbound packets, which may suggest an attack with randomly spoofed traffic. Once an anomaly is detected, the Commander can issue new filtering rules to all Shields.

4.3.3 Probe

In order to minimize the damage caused by a DDoS attack, the Aegis system takes one step further and uses Probes, which are designed to act during the "traceback" and "second response" stages. If the Aegis system confirms a DDoS attack and decides to block the flood completely, it sends Probes towards the sources of the flood in order to achieve two goals: to block the flood at the uppermost stream possible, and to gather evidence of the attack for legal purposes. The Probes are also implemented as Active Network modules and behave like mobile agents that can move from hop to hop towards the flood sources. In the following section, we will develop a probing algorithm as we attempt to counter both Model A and Model B DDoS attacks.

Countering a Model-A DDoS Attack: The source addresses of packets received may or may not be spoofed. Because there is no way of telling if these addresses are legitimate or not, we can only assume initially that all of them are spoofed. We need some backtrack mechanism to identify a better position to block the malicious flood in the upstream. A number of research efforts have been dedicated to tackling this problem ([5] and [6]) using packet marking. While it is possible to integrate these proposals into Aegis, we have taken a different approach; one that utilizes the flexibility offered by Active Networks. Our probing algorithm is outlined in the following pseudo-code. The source address of the tracked traffic found by the countering Shield is stored in the variable *attSrc*. In accordance with the second requirement in Section 3, we can assume that each Active Node will provide us with a function that will allows us to capture packets associated with the owner of the module. (capturePacketByDest in line 3).

Starting from the countering Shield, Probes are continuously replicated and dispatched to neighboring nodes in the direction of the source of the attack. After arriving at a new node, each Probe performs one of the following sequences of actions:

1) if the traffic being tracked is found in the new node (line 3-4): block the traffic (line 5) → dispatch replicates to adjacent nodes except the predecessor (line 7,8) → self-destruct when the tracked traffic no longer comes (line 11, 12).

2) if the traffic being tracked is not found in the new node after a certain time: self-destruct (line 11, 12).

```
//Arriving at a new node u
1. explored←false
2. do
3.     p←capturePacketByDest(victim's IP)
4.     if p.src=attSrc then
5.         discard(p)
6.         if !explored
7.             for each v∈Adjacent[u]
8.                 if v∉predecessor[u] then
9.                     dispatch a replicate of Probe to v
10.    else releasePacket(p)
11.while (p≠NIL) and (!IdleTimeOut)
12.self-destruct
```

Fig. 11 illustrates the progress of this traceback mechanism on a portion of the network.

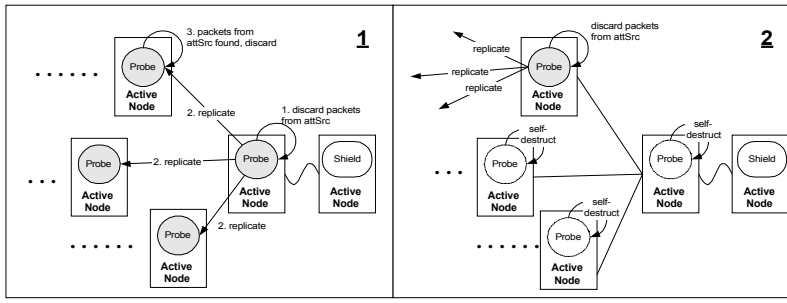

Fig. 11. Traceback.

Recall that in Model A, each attack path is denoted as $A_i = <s_i, v_{i,1}, v_{i,2},\ldots, v_{i,ni}, d>$. The goal of the above algorithm is to find $v_{i,min}$ such that $v_{i,min} \in$ Anet, and to block all unwanted traffic there. This has the effect of freeing routers $v_{i,min+1}, v_{i,min+2,\ldots}, v_{i,ni}$ from processing packets launched by the attacker. Once the probe reaches $v_{i,min}$, where no more neighboring active node can be found in the direction of flood source, it reports its location (IP address) to the Commander for the purpose of evidence gathering.

Countering a Model-B DDoS Attack: Recall that in a Model B DDoS attack, each attack path is denoted as $B_i = <s_i, v_{i,1}, v_{i,2},\ldots v_{i,mi-1}, \alpha_i, v_{i,mi+1}\ldots, v_{i,ni}, d>$. Each packet generated by the attacker in the subpath $<s_i, v_{i,1}, v_{i,2},\ldots v_{i,mi-1}, \alpha_i>$ carries a spoofed source address d, while each packet in the subpath $<\alpha_i, v_{i,mi+1}\ldots, v_{i,ni}, d>$ carries a legitimate source addresses of hosts in the amplifying network. Since the Shield would initially store in *attSrc* the source addresses of hosts residing in the amplifying network, the Probes can explore only as far as $v_{i,min}$ such that $v_{i,min} \in$ Anet and $m \leq min \leq n$, using the algorithm described above. If there exists $v_{i,k}$ such that $v_{i,k} \in$ Anet and $1 \leq k \leq m$, by moving our Probes into $v_{i,k}$, we can further reduce the unwanted traffic by up to a factor of the amplifying ratio. We now need to upgrade our algorithm so that the Probes would move beyond the amplifying network. This can be done by modifying line 3 and 4 in the original algorithm (see psuedo-code below). This time, the Probe examines all packets associated with the victim's IP (i.e. packets in which the victim's IP appears as the source or destination). In addition to the original condition in line 4, if the packet appears to have originated from the victim and to be destined for the broadcast address (such as 255) of the attSrc, the packet is immediately discarded. These modifications allow the Probes to penetrate further towards the Slave terminals.

```
//Arriving at a new node u

1. explored←false

2. do

3.     p←capturePacketAssociatedWith(victim's IP)

4.    if (p.src=attSrc) or ((p.src=victim's IP) and
      (p.dest=atSrc's broadcast)) then

5.          discard(p)

6.          if !explored

7.             for each v∈Adjacent[u]

8.                if v∉predecessor[u] then

9.                    dispatch a replicate of Probe to v
```

```
10.     else releasePacket(p)

11.while (p≠NIL) and (!IdleTimeOut)

12.self-destruct
```

5 Future Work

In our future work, we will attempt to tackle the following issues:

Dynamic Routing. In our framework, we have assumed that all attack paths are fixed; however, it is possible to experience route changes during an attack. We therefore need to consider such a scenario and incorporate it into Aegis.

Tracking Control Traffic. Currently Aegis is able to trace back up to the Slave terminals. In order to facilitate legal actions, it is necessary to go beyond the Slaves and trace the control traffic in order to find the Master machine. This is an enhancement we plan to work on in the future.

Finding/Designing a Suitable EE. We are looking for a suitable EE that meets the requirements listed in Section 3. If none of the existing ones satisfy our needs, we plan to design one by ourselves and implement a prototype.

6 Conclusion

In principle, the Aegis is a distributed firewall system designed to defend against the distributed nature of DDoS attacks. We have proposed an effective solution that cuts off unwanted traffic automatically at the upper stream, thus freeing the victim's network from massive bandwidth consumption. Although Aegis cannot be incorporated into the Internet in its present form to offer an immediate solution to DDoS attacks, we hope that we have at least demonstrated some of the new possibilities that Active Networks can offer.

Acknowledgements

The author wishes to thank Keiichi Okabe and Hitoshi Fuji, researchers of NTT Information Sharing Platform Laboratories, for valuable support.

References

1. Joel Scambray, Stuart McClure and George Kurtz: “Hacking Exposed: Network Security Secrets & Solutions, Second Edition”, ISBN: 0-07-212748-1, McGraw Hill, U.S.A., 2001.
2. David Dittrich: “Distributed Denial of Service (DDoS) Attacks/tools” http://staff.washington.edu/dittrich/misc/ddos/ .
3. P. Ferguson, D. Senie: “Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing” RFC2267, Jan 98.
4. S. Gibson: “The Strange Tale of the Distributed Denial of Service Attacks Against GRC.COM”, http://grc.com/dos/grcdos.htm, June 2, 2001.
5. Kihong Park and Heejo Lee: “On the Effectiveness of Probabilistic Packet Marking for IP Traceback under Denial of Service Attack”, Network Systems Lab and CERIAS, Purdue University.
6. Stefan Savage, David Wetherall, Anna Karlin and Tom Anderson: “Practical Network Support for IP Traceback”, Proceedings of the 2000 ACM SIGCOMM Conference, pp. 295-306, Stockholm, Sweden, August 2000.
7. Steve Berson (ISI): "A Gentle Introduction to the ABone", OPENSIG 2000 Workshop, Napa, CA, 11-12 October 2000.
8. FAIN project WWW Server – May 2000 http://face.ee.ucl.ac.uk/fain.
9. Japan Gigabit Network, http://www.jgn.tao.go.jp/gaiyou.html.
10. K. Psounis “Active Networks: Applications, Security, Safety, and Architectures”, IEEE Comsoc Surveys, 1st Quarter 1999.
11. Beverly Schwartz, Wenyi Zhou, and Alden W. Jackson: “Smart Packets for Active Networks, BBN Technologies, Jan. 1998.
12. David J. Wetherall and David L. Tennenhouse, “The ACTIVE_IP Option”, 7th ACM SIGOPS European Workshop.
13. D.J. Wetherall, J. V. Guttag, and D.L.Tennenhouse, “ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols”, Proceedings of IEEE OPENARCH’98, April 1998.
14. Dan Decasper and Bernhard Plattner: “DAN: Distributed Code Cashing for Active Networks”, Proceedings of IEEE INFOCOM’98, San Francisco, CA, April 1998.
15. C.A. Gunter, S.M. Nettles, and J.M. Smith, “The SwitchWare Active Network Architecture”, IEEE Network Magazine, May/June 1998, vol. 12, no. 3.
16. Y. Yemini and S. da Silva: “Towards Programmable Networks”, IFIP/IEEE International Workshop on Distributed Systems, Operations and Management, L'Aquila, Italy, October, 1996.
17. Eric Y. Chen and Hitoshi Fuji: “A Security Framework for Agent-Based Active Networks”, Proceedings of M2USIC 2000, Petaling Jaya, Malaysia, October 2000.
18. Active Network Encapsulation Protocol (ANEP) – July 1997 http://www.cis.upenn.edu/~switchware/ANEP/docs/ANEP.txt.
19. T.M. Gil and M. Poletto: “MULTOPS: A data-structure for bandwidth attack detection”, http://pdos.lcs.mit.edu/thomer/mit/multops_usenix2001.pdf.