Off The Wall: Lightweight Distributed Filtering to Mitigate Distributed Denial of Service Attacks

Zhang Fu, Marina Papatriantafilou
Chalmers University of Technology, 42196 Gothenburg Sweden. Email: {zhafu,ptrianta}@chalmers.se

Abstract—**Distributed Denial of Service (DDoS) attacks are hard to deal with, due to the fact that it is difficult to distinguish legitimate traffic from malicious traffic, especially since the latter is from distributed sources. To accurately filter malicious traffic one needs (strong but costly) packet authentication primitives which increase the design complexity and typically affect throughput. It is a challenge to keep a balance between throughput and security/protection of the network core and end resources. In this paper, we propose SIEVE, a lightweight distributed filtering protocol/method. Depending on the attacker's ability, SIEVE can provide a standalone filter for moderate adversary models and a complementary filter which can enhance the performance of strong and more complex methods for stronger adversary models.**

I. Overview

Mitigating DDoS attacks is difficult, since it is always a two-fold project: On one hand, the illegitimate traffic should be filtered as much as possible, while on the other hand, the network performance for the legitimate traffic should be degraded as little as possible. To have good filtering effectiveness, it is always necessary that the legitimate traffic is distinguishable from the malicious traffic. Usually, this can be achieved by letting legitimate clients share some secret knowledge with the filter, which cannot be revealed to the attacker. Thus legitimate packets can have valid properties (e.g. message authentication codes, MACs) that enable them to pass through the filter. Based on this filtering paradigm, three challenges follow as consequences:

a) Connection setup challenge: Legitimate clients need to request the filtering knowledge before sending packets to the server. The attacker can flood massive such requests to prevent legitimate requests from reaching the server. Granting the local knowledge of the server to the filtering entities (e.g. routers) may not be applicable due to privacy issues. So there is a need for a way for legitimate requests to have chances to be forwarded to the server when competing with massive spurious requests.

b) Efficient authentication challenge: It is quite common that unforgeable authentication tokens (e.g. MACs) are used for packets filtering [18], [22], [17], which requires the filtering entities to share the same secrets (e.g. keys) with the legitimate clients and execute some hash function to verify each packet. Such authentication procedure definitely undermines the throughput of filtering entities (e.g. routers). The attacker can simply flood packets with arbitrary MACs in order to keep the checking node busy with verifying packets, which may form a Denial of Service attack. Furthermore, using authentication tokens may require new field(s) in the packet header. This increases the design complexity and typically harms the bandwidth utilization.

c) Deployment challenge: Due to the distributed nature of DDoS attacks, routers' support to filter massive malicious packets seems inevitable. Routers can filter packets based on the paths through which packets are forwarded [16], [13], they can also filter packets according to message authentication codes (MAC) [22]. However, since there is a need for global deployment and changes to current routing infrastructure, such solutions are not straightforward to apply.

The research leading to these results has received funding from the Swedish Civil Contingencies Agency (MSB) and from European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement No.257007.

Motives and our approach: By studying the communication architecture of the modern botnets [9], it is observed that the attacker usually needs some time to configure the bots with new attack commands. We reflected on this and on earlier research where it was shown that smart updating of secret knowledge enables to both allow legitimate entities to use resources and leave non-legitimate ones "in the dark". The latter has been suggested and worked out in detail for application-level mitigation [3]. Inspired from this reflection, we suggest SIEVE: a new lightweight method that enables DDoS mitigation at the network level through overlay networks. Our goal is to provide a simple and efficient, yet effective enough mechanism to mitigate DDoS attacks. In particular, we address the above challenges in the following way:

a) Protecting connection establishment: In SIEVE, legitimate clients need to acquire from the server some secret knowledge to pass through the filter. We provide a simple solution to address the connection setup challenge. Roughly speaking, in our solution, network hosts are partitioned into domains. Each domain has its quota for sending connection requests to the server. Even if an attacker can compromise many sending sources in some domains, the connection requests from the uninfected domains will not be affected. In each infected domain, the solution provides guaranteed probability for a legitimate request to be served.

b) Lightweight authenticator: SIEVE uses IP addresses as the authenticators. According to its address, a legitimate client will send its packets to different filtering points in different time periods. Since it takes time for the attacker to find out the filtering rules and to configure an attack using e.g. a botnet, if the filtering rules can change with appropriate timing, the attacker can be kept in the "dark" and cannot launch flooding attacks effectively. Using IP addresses as the authenticators needs neither extra fields in the packet header nor extra processing time for each incoming packet. This implies saving of many orders of magnitude of CPU cycles compared with MAC authentication.

c) Overlay-based deployment: SIEVE uses overlays to deploy the distributed filter and expand the receiving and filtering capacity of the protected victim. Overlay nodes act as the access points of the potential victim. Any packet that goes to the victim should be checked and forwarded by the overlay nodes. Note that several solutions were proposed [11], [2], [18], [19], [5] to use overlay networks for mitigating DDoS attacks, since overlay networks are easy to deploy and do not require global changes of current routing protocols. However, SIEVE differs from the previous work in the way of addressing the connection establishment and authentication/throughput challenges.

The proposed solution also achieves the following:

d) Adaptive cost: We provide analysis of the filtering effectiveness depending on the *forwarding bandwidth* of each overlay node, sending rate of the legitimate packets, flooding rate from the attacker. According to our analysis, the potential mitigation service provider can offer different cost options of different filtering qualities to its customers.

II. Problem and System Model

The network is comprised of: *servers*: network entities who provide some services (e.g. remote storage services) to their clients; *clients*, end-hosts who have valid identities to get service from the servers; *overlay nodes*, network hosts or routers who forward packets from clients to the servers; *compromised hosts*, network hosts that are controlled by the adversary. There are N overlay nodes, we use n_i, $i \in \{1, 2, \cdots, N\}$ to denote overlay node i. The word "node" refers to an overlay node, unless specified otherwise. The deviation of the clock values of any pair of different network entities is bounded by a small range, ϵ time units. The method can be generalized as in [8] to work in the presence of clock drifts.

The problem addressed here involves the adversary (attacker) whose goal is to decrease the proportion of the legitimate traffic received by a server as much as possible by flooding massive packets to the server. From the protection view, the proposed solution tries to use overlay nodes to filter malicious traffic before it reaches the server. Each such overlay node has limited bandwidth to forward traffic to the server. Following the link congestion model [4], we assume that the proportion of legitimate traffic that can pass through node n_i is on average $\frac{c}{a+b}$, where c the forwarding bandwidth of node n_i, a is the total legitimate traffic rate to n_i and c is the rate of the malicious traffic that can pass the filter of n_i. We also assume that the incoming bandwidth of each overlay node is well provisioned and hard to be attacked (e.g. the overlay nodes are deployed in the core network.)

The adversary controls a number of compromised hosts. The aggregated attacking rate is bounded by $\mathcal{M}$. The adversary cannot predict any future filter rules based on the information of historical filter rules. The adversary may find out the addresses that each node accepts in the current time period by randomly eavesdropping some legitimate traffic. However, it takes some time (usually several seconds are needed) for it to send new attack command and aggregate enough attacking traffic to flood the overlay nodes. This is known as the *ramp-up* behavior of multi-source flooding attacks [10]. The ramp-up behavior may be caused by the hierarchical communication pattern between the adversary and the compromised hosts [14]. Therefore, we assume that the adversary needs time to adjust commands and aggregate traffic which matches the newly updated filtering rules; the rate of such aggregated attacking traffic is negligible before the filter rules are updated. Based on the above, we consider the adversary is able to launch *blind attacks*, following the literature [3], [8], i.e. the adversary can flood packets to arbitrary overlay nodes using arbitrary source addresses in the packets.

For the targeted services that SIEVE aims to protect, we assume that there exist application-level credentials (e.g. passwords or public-private keys) that the clients can use to authenticate themselves to the server, such as the ones used in remote storage services. We assume that legitimate clients are honest and cannot be compromised. However, note that SIEVE can be combined with strong and more complex authentication methods to deal with compromised clients (see Section IV).

III. SIEVE Overview

For simplicity of the presentation, we describe SIEVE from the perspective of one server. The method can protect multiple servers. In SIEVE, there are three basic components: *redirectors*, *overlay nodes*, and the *protected server*. All the packets to the server should pass through the distributed filter constructed by N overlay nodes. To support the updating of filter rules, time is divided into periods. We use τ_i, $i \in \mathbb{N}$ to denote time period i. Set $\mathcal{S}$ is the set of addresses of legitimate clients. In period τ_i, the set of addresses from which overlay node n_j accepts packets is denoted as $s_{i,j}$, $j \in \{1, 2, \cdots, N\}$. We say in τ_i, node n_j serves address set $s_{i,j}$ and we have $\forall i : \bigcup_{1 \le j \le N} s_{i,j} = \mathcal{S}$. Which node serves which address set is based on a pseudo-random function possessed by the server and kept secret to the adversary. A client needs to set a connection with the server to get the information of the correct overlay node that accepts its address in each time period.

Initially a client has no idea about the addresses of the overlay nodes. To get such information it first sends its connection request to one *redirector*. In the Internet infrastructure, the redirectors can be DNS servers. The addresses of the overlay nodes should be registered in the redirectors. The redirector will forward the request to one of the overlay nodes e.g. the one nearest to the client. Then it is the task of the overlay nodes to forward the request to the server.

Upon receiving a connection request from one overlay node, the server verifies the application-related credential included in the request. This procedure depends on the service being protected, and is up to the server. The server admits the connection request by issuing the information of overlay nodes to which the client should send packets in each time period. Initially, the server only grants the client overlay nodes information for several time periods, and after that the server informs the client about the updates through overlay nodes until the session of the client terminates.

To avoid the possibility for the adversary to bypass the overlay nodes and flood malicious packets directly to the server, SIEVE adopts a method also employed in OverDose [17] to isolate the server by placing it on a private network that the adversary cannot reach through IP routing infrastructure. The ISP of the server can configure tunnels between overlay nodes and the server by e.g. Multiprotocol Label Switching (MPLS).

IV. Design Details

Bootstrapping connection setup: In SIEVE, the network is divided into N subnets, where N is the number of overlay nodes. Each subnet is called a *domain*. Each overlay node always forwards requests from a specific domain. In particular, in each time period, overlay node n_i only forwards w_i requests from domain d_i, and $\sum_i w_i = W$, where W is the number of requests that the server can handle in a time period. We say the *quota* for domain d_i is w_i. The allocation of the quota for a domain depends on the proportion of the legitimate requests from that domain when there is no attack. To establish a connection to the server, a client should first send a request (the *original request*) to the overlay node which is responsible for its domain. When the overlay node receives the original request it sends back a response to the client with a cookie, which is a key-hashed value using a period key and a nonce which are generated by the overlay node itself. The cookie can be expressed in the following way:

$$Cookie = H_{\mathcal{K}}\left[srcIP||nonce||\tau_i\right]$$

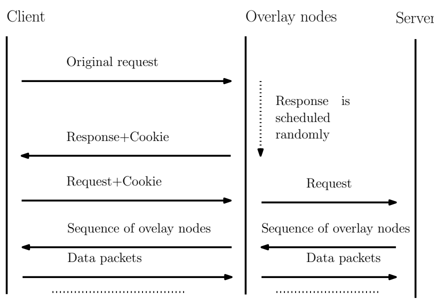

Fig. 1. Cookies and random scheduling are used to protect the connection setup phase. The client sends data packets through overlay nodes according to the sequence of node-addresses granted by the server.

where $\mathcal{K}$ denotes the period key, $srcIP$ is the client's address; τ_i denotes period, and $||$ denotes concatenation; $\mathcal{K}$ and the nonce will be changed periodically. When a client gets the response, it sends back the request with the cookie. When the overlay node receives the request with a valid cookie, it checks whether the quota of the current period is exhausted, and whether the same request has already been forwarded in the current period (one request received by the server is enough for the client to setup a connection). If everything is OK, the request is forwarded to the server and the quota is decreased; otherwise the request is dropped. To bound the overhead of keeping state of requests per period by overlay nodes, we propose the use of Bloom Filters [15].

Using cookies can prevent the adversary from consuming the quota by sending requests with spoofed addresses, since it will not receive the corresponding cookies. However, the adversary may compromise many hosts in a domain and let them keep sending requests with their real addresses, so that at the beginning of every period they can acquire a big proportion (could be all) of the quota, leaving only a little chance for the legitimate requests to get forwarded by the overlay node. To address the problem, we propose *random scheduling*. Instead of sending the response immediately after receiving an original request, the overlay node first randomly chooses a time point within the current period. If the time point is smaller than or equal to the current time, the response is sent immediately. Otherwise, the response is inserted into a *sending queue* and will be sent at the corresponding time point. The time sequence diagram of the random scheduling is shown in Figure 1. Algorithm 1 shows the pseudo-code for an overlay node to protect the connection setup.

Lightweight traffic filtering: In SIEVE, the client-address set $\mathcal{S}$ (could be e.g. the whole IP space or a subnet) is partitioned into N subsets (which change regularly and are not the same as the domains mentioned above) and each overlay node serves one subset in each time period. To make the partition hard to guess, in each time period, the server randomly chooses $\log_2 N$ bits in the binary expression of the addresses, and $\mathcal{S}$ is partitioned by the patterns of those $\log_2 N$ bits. For example, suppose $\mathcal{S}$ contains all the addresses of IPv4, and $N = 4$, then the server will randomly choose $\log_2 N = 2$ bits from the 32 bits. By choosing the 1^{st} and 6^{th} bits, the whole address set is partitioned into 4 subsets which have the following patterns:

0****0**.********.********.********
0****1**.********.********.********
1****0**.********.********.********
1****1**.********.********.********

An overlay node filters packets according to the bit-pattern of the address subset it serves. Note that $\mathcal{S}$ does not necessarily contain the whole IP address space, it can e.g. belong to a subnet, depending on the service offered by the server.

Algorithm 1: Algorithm for an overlay node to forward requests from a specific domain.

```
// When time period $p_i$ starts, $i \in \mathbb{N}$
$Pnumber$++; $quota \leftarrow w$;
clear the Bloom filter;
$t_s \leftarrow$ beginning time of $p_i$; $t_e \leftarrow$ ending time of $p_i$;
$nonce \leftarrow$ new random nonce; $\mathcal{K} \leftarrow$ new key;
// When receive an original request $req$
if $req$ is not from the domain then drop $req$;
else
    if sendingQueue.hasResponse($req$) then drop $req$;
    else
        $response \leftarrow req||H_{\mathcal{K}}(srcIP||nonce||Pnumber)$;
        random choose $t \in (t_s, t_e)$;
        if $t <= currenttime$ then send $response$;
        else sendingQueue.insert($response$,$t$);
    end
end
// When receive a request $req$ with cookie
if $quota > 0$ & cookie is valid & $req$ is not in the Bloom filter then
    add $req$ into Bloom filter;
    forward $req$; $quota - -$;
end
// Sending responses in the sendingQueue
while true do
    $res \leftarrow$ sendingQueue.nextResponse;
    if $res.sendingtime <= currenttime$ then send $res$;
end
```

The server generates the partition bits used for each period according to some pseudo-random function. In each time period, the server informs the overlay nodes which bits are used in the address partition, and which node serves which subset of $\mathcal{S}$. Note that such information can also be given for a bunch of periods, for cost amortization. Pseudo-code for the server to generate this information is shown in Algorithm 2.

Algorithm 2: Algorithm for the server to generate partition bits for Φ periods.

```
// Generate partition bits for $\Phi$ periods
$Pnumber \leftarrow$ current period number;
$List_{bits}$ ; $List_{nodes}$ ;    /* Each item in $List_{bits}$ is a
set of bits used for one period. Each item in
$List_{nodes}$ is an address of an overlay node. */
for $i \leftarrow 0$ to $\Phi - 1$ do
    $List_{bits}[i] = \mathcal{F}_{rand}(seed, Pnumber)$ ; /* $\mathcal{F}_{rand}$ is the
    pseudo random function for generating $\log_2 N$
    indexes indicating address bits. */
    $Pnumber$++;
end
Randomly reorder the items in $List_{nodes}$;
$subset = 0$;
for $node \in List_{nodes}$ do
    keep information of $(node, subset)$;
    inform $\langle node, List_{bits}, subset, \Phi \rangle$;
    $subset$ ++;
end
```

Cost of packet authentication: Filtering packets according to source addresses is quite easy and does not need extra CPU time especially, when routers are used as the overlay nodes, since anyway they will check the addresses for routing packets. Usually checking some bits in the address field only costs 2 to 3 CPU cycles per packet. The cost for message authentication codes (MACs) depends on the hash algorithms used in the method and the packet sizes. According to the simulation results in [21], taking

UMAC-64 as example (since it is advocated for balancing the security and performance trade-off), it needs 1433 CPU cycles per packet with size of 1024 bytes, which is around 600 times the cost of the method in SIEVE. If SHA-1 is used as the hash algorithm to produce MACs, then it costs 24166 CPU cycles per packet with size of 1024 bytes, which is around 10000 times the cost of the method in SIEVE.

Setting forwarding bandwidth: The forwarding bandwidth from each overlay node to the server can be set using probability estimations. Since the bits for partitioning $\mathcal{S}$ are generated randomly, on average each client has the same probability to belong to each address subset. This implies that the expected number of clients that each overlay node serves is the same. Let X denote the number of the clients served by an overlay node, μ be the expectation of X. By applying Chernoff Bounds, we have $Pr\left(X \leq (1+\delta)\mu\right) > 1 - \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mu}$, where $\delta > 0$. Suppose there are 10000 clients and 10 overlay nodes, then each node is expected to serve $\mu = 1000$ clients. Suppose each client sends traffic of rate $100Kbps$, then the total rate of legitimate traffic goes through each overlay node is expected to be $100Mbps$. If we set $\delta = 0.1$, then the probability that one node serves less than $(1+\delta)\mu = 1100$ clients is more than 0.9922. This implies that the total rate of legitimate traffic that goes through each overlay node can hardly exceed $110Mbps$. So by allocating the forwarding bandwidth of an overlay node to $110Mbps$, it is safe to support the legitimate traffic that goes through that node.

Using SIEVE as a complementary filter: SIEVE offers a standalone and lightweight anti-DDoS method under a moderate adversary model. However, when the adversary is more powerful (e.g. it can quickly "replay" legitimate packets snooped from the network or it can compromise legitimate clients), using MACs (message authentication codes) for checking the validity of packets and keeping states for legitimate flows may be indeed necessary. To balance the trade-off between cost and protection, SIEVE can be used as a complementary filter to enhance the performance of stronger and more complex methods to fight against stronger adversary. For example when MACs are used in the filtering mechanism in SIEVE, the overlay nodes can generate the symmetric key shared with a client (for generating and verifying the MACs of the client's packets). The key can be piggybacked (encrypted with the client's public key) on the connection reply message to the client. Note that overlay nodes check the MACs of packets only when necessary. So when overlay nodes observe abnormal big volumes of traffic that can pass the lightweight filter, the MAC validation procedure can be invoked and the clients will be informed to include MACs in their packets. In that case, overlay nodes will first check the source address of each incoming packet and then the MAC. In this way, the lightweight filter is the first level filter which can efficiently reduce the amount of packets that the second level filter needs to check.

V. Analysis

We analyze the effectiveness of the protection of connection setup and the lightweight filtering through a series of lemmas. Due to space limitations, all of the proofs and comments are omitted and the reader is redirected to [7].

Lemma 1. *Within a time period, a compromised host has to send the original request at the beginning of the period so that the expectation of the time that it gets the response is minimized.*

Lemma 2. *Consider a domain whose quota is w per period. There are ζ compromised hosts in the domain, and the arrival events of original legitimate requests form a Poisson process with rate λ, SIEVE guarantees that the proportion of legitimate requests forwarded by the overlay node is at least $\frac{w-\zeta\frac{\sqrt{4\lambda w+\zeta^2}-\zeta}{2\lambda}}{\lambda}$.*

Lemma 3. *The expectation of the waiting time for a specific response to be sent back to the client is $\frac{T}{6}$, where T is the length of a time period.*

Lemma 4. *Regarding the legitimate traffic served by node n_i, if L_i is the rate of this traffic, then the expected proportion of this traffic that can pass through n_i is at least $\frac{C_i}{L_i+\frac{m_i}{N}}$, where C_i is forwarding bandwidth from overlay node n_i to the server, m_i is the traffic rate that the adversary floods to node n_i.*

Lemma 5. *The proportion of total legitimate traffic that can pass through the distributed filter is*

$$\mathscr{P} \geq \sum_{i|n_i \notin \mathcal{N}_{\mathcal{A}}} \frac{L_i}{\mathcal{L}} + \sum_{i|n_i \in \mathcal{N}_{\mathcal{A}}} \frac{C_i}{L_i + \frac{\mathcal{M}}{N|\mathcal{N}_{\mathcal{A}}|}} \cdot \frac{L_i}{\mathcal{L}},$$

where $\mathcal{N}_{\mathcal{A}}=\{n_i|m_i > 0, i \in \{1,2,\cdots,N\}\}$, $\sum_{i=1}^{N} L_i = \mathcal{L}$, $\sum_{i=1}^{N} m_i = \mathcal{M}$.

Lemma 6. *The proportion of legitimate traffic that can pass through the distributed filter is at least $\frac{\mathcal{C}}{\mathcal{L}+\frac{\mathcal{M}}{N}}$*

Corollary 1. *Given the total attacking rate $\mathcal{M}$, to achieve a specific filtering quality that the proportion of the legitimate traffic received by the server is at least $\mathscr{P}$, there should be $\left\lceil \frac{\mathcal{M}}{\frac{\mathcal{C}}{\mathscr{P}}-\mathcal{L}} \right\rceil$ overlay nodes deployed in SIEVE, where $\mathcal{C}$ is the receiving capacity of the server and $\mathcal{L}$ is the total rate of the legitimate traffic.*

Corollary 1 actually provides a guideline for balancing the trade off between cost and protection (i.e. the filer effectiveness). The analysis shows that SIEVE can filter out malicious traffic by the order of $\Omega(N)$, meaning that at most $\frac{1}{N}$ of the malicious traffic can really compete for the server's bandwidth with the legitimate traffic. So according to different protection demands, the protection provider can decide the number of nodes used in the protection. E.g. consider a server whose incoming bandwidth is $100Mbps$, wants to be protected. Assume that the total rate of the legitimate traffic to that server is $50Mbps$. To withstand $10Gbps$ attacking traffic and guarantee that the server receives at least 90% of the legitimate traffic, 164 overlay nodes are needed to be deployed in SIEVE, which is a reasonable number for a protection provider, such as Akamai [1], which has several thousands overlay nodes.

Induced latency by routing via overlay nodes: Routing packets through overlay nodes may induce extra latencies, since the overlay nodes may not reside in the optimal path between the hosts and the server. To have an exact estimation of the induced latency is rather complex, involving several parameters, such as network topology and the network conditions. We provide an illustrative example to study how the number of hops affects the latency. We compare the shortest path length via overlay nodes with the original shortest path length between a host and the server in power-law network topologies motivated by earlier results [6]. Figure 2 shows the corresponding ratio. It is claimed that Phalanx [5] can control the induced latencies to 20% by using iPlane [12] to choose overlay nodes for the legitimate clients. iPlane can be definitely used also by SIEVE for the same purpose, since it can provide accurate predictions of Internet path performance.

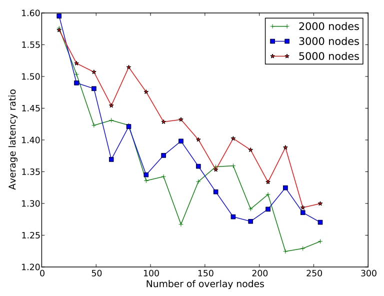

Fig. 2. Induced latency by routing through overlay. The network graph has power-law topology with $\beta = 2$ (β is the exponent in the power-law model), and the total number of nodes in the graph ranges in (2000,3000,5000).

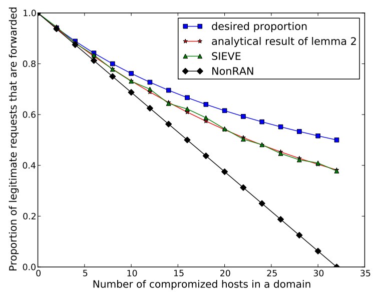

Fig. 3. The proportion of legitimate requests that can be forwarded by the overlay node. The quota for the domain is 32, which is equal to the number of legitimate hosts. The number of compromised hosts ranges from 0 to 32.

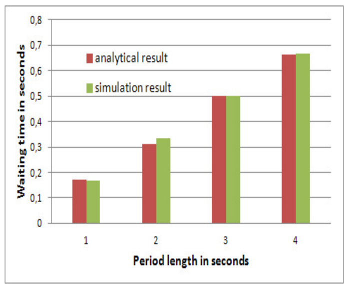

Fig. 4. The average waiting time for a response of a legitimate request to be sent, compared with the result in lemma 3.

VI. Simulation Study

We study SIEVE's performance in two aspects: protecting connection setup and lightweight traffic filtering. In particular, we measure the proportion of legitimate requests that can be forwarded, the latency induced by the random scheduling and the proportion of legitimate traffic that can be received by the server. In order to validate our analysis results, we associate each simulation result with the corresponding lemma in Section V. All the simulations are performed in Ns2 [20]; and for each simulation scenario, we run the simulation for 10 times and take the average.

Connection setup simulation: We create a network with 128 nodes for hosts, and assume that they are in the same domain. The quota for the domain is 32 per period. We vary the number of compromised hosts from 0 to 32. From the rest of the hosts that are not compromised by the adversary, we select 32 hosts to act as legitimate clients. In each period, the time for a legitimate client to send a request is uniformly distributed among the period, and each legitimate client only sends one request per period. To model a worst case scenario, the compromised hosts send their requests at the beginning of each time period, following the conclusion of lemma 1. We measure the proportion of legitimate requests forwarded by the overlay node, and compare it with the *desired proportion*, which is defined as $\frac{\text{quota}}{\text{number of hosts sending requests}}$. The desired proportion is the optimal fairness (i.e. *per-host fairness*) that can be achieved when the malicious requests cannot be distinguished. In order to see the benefit from random scheduling, we also measure the performance of SIEVE without random scheduling, called NonRAN. In NonRAN, the overlay node immediately sends a response with a cookie when it receives a request. Figure 3 shows the simulation results and the comparisons. We can see that the curve follows closely the analytical result of lemma 2. Random scheduling used in SIEVE may cause the clients to wait for the responses for some time. Lemma 3 shows that the expected induced latency is bounded by $\frac{T}{6}$. Figure 4 shows the simulation result which conforms well to the analytical result.

Traffic filtering simulation: We measure the percentage of legitimate traffic and also the percentage of attack traffic that can pass through the filtering of SIEVE. We keep the previous configuration, which has 128 client nodes. The total incoming bandwidth of the server is $32Mbps$. Each of the client sends UDP packets with constant rate $0.25Mbps$, so the total legitimate traffic rate is $32Mbps$. The adversary's strength is simulated through the total attacking rate which ranges from $96Mbps$ to $256Mbps$, i.e. 3 to 8 times the legitimate traffic rate. To measure the filtering performance of SIEVE, we measure the percentage of legitimate packets that can be received by the server. We set the length of time period to 1 second. The simulation time consists of 10 periods. We choose the number of overlay nodes to be 32 and 64.

When there are 32 overlay nodes used in SIEVE, the address space of the 128 legitimate clients is divided into 32 subsets, each has 4 legitimate clients. Thus each overlay node serves 4 clients with $1Mbps$ of the total legitimate traffic rate. To perform stress test on SIEVE, we let each overlay node have a link to the server with bandwidth $1Mbps$. So the forwarding bandwidth of each overlay node is just enough for forwarding the legitimate traffic sent to it. To associate with the analysis of lemma 4 and lemma 6, we vary the number of overlay nodes that the adversary attacks. During the attack, the adversary floods packets with its total attacking rate. Each malicious packet is randomly assigned an address of a legitimate client, and the packet is sent randomly to one of the overlay nodes that the adversary wants to attack. Figure 5 shows the percentage of legitimate packets received by the server when there are 32 overlay nodes in SIEVE. It is observed that the measured percentage decreases when the adversary attacks more overlay nodes. When the adversary attacks all the overlay nodes, the measured percentage reaches its minimum value. For the analytical study of this observation, we refer to the proof of lemma 6 [7].

By repeating the simulation with a larger number of overlay nodes (64 nodes) and the same client set, we observe better filter effectiveness. Figure 6 shows the corresponding result. Figure 7 shows the bandwidth utilization of the server's incoming link. We can see that SIEVE saves most of the server's bandwidth (above 90% when SIEVE has 64 nodes) for legitimate traffic. To contrast this with the situation that the server has no protection, note that in the latter case only little (around 10% when attacking rate is $256Mbps$) of the bandwidth can be used by legitimate traffic. Figure 8 shows the fraction of the attack traffic passing through the filter.

VII. Related work

Since we use an overlay to build a lightweight distributed filter, we mainly focus on solutions that also adopt overlays to deal with DDoS attacks.

SOS [11] is the first solution that uses an overlay network to mitigate DDoS attacks. SOS and its generalized version Mayday [2] focus on protecting the traffic of *confirmed users*,

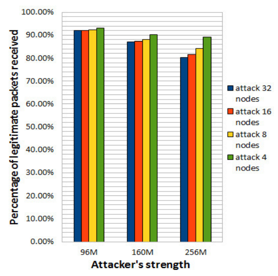

Fig. 5. The percentage of legitimate packets received by the server, when there are 32 overlay nodes.

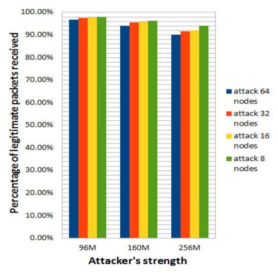

Fig. 6. The percentage of legitimate packets received by the server, when there are 64 overlay nodes.

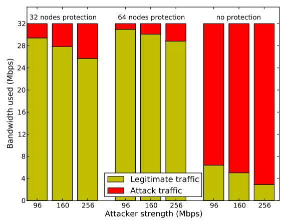

Fig. 7. Bandwidth utilization of the server's incoming link.

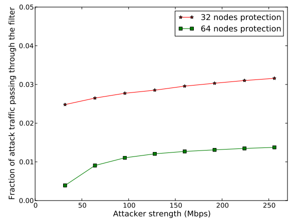

Fig. 8. Fraction of the attack traffic that can pass through the lightweight filter

which by assumption have prior permissions to communicate with the server. The issues of securing connection setup phase addressed in SIEVE are not considered in the work. MOVE [19] and Multipath-Overlay [18] provide MAC-based mechanisms for the overlay nodes to filter out illegitimate packets. These two solutions also suggest to use Graphic Turing Test (GTT) for filtering out requests from remotely controlled zombie machines. However, GTT needs humans involved and is not transparent to users. OverDose [17] protects the connection setup phase by using crypto-puzzles. Each request packet should contain a correct solution of a puzzle generated with the current puzzle seed (which is changed periodically), and packets with correct solutions of puzzles in higher levels have priority for being forwarded. However, in OverDose there is no mechanism to prevent a host from reusing solutions of the same puzzle. Furthermore, the adversary can always flood spurious request packets with puzzle level higher than the legitimate requests, in order to keep the overlay node busy with checking puzzle solutions with that level. So legitimate clients always have to solve puzzle of level as high as possible, which costs more computation resources and induces high latency. As shown in the analysis and simulation sections, the induced latency by SIEVE is bounded.

Like the method in SIEVE, Phalanx [5] also tries to provide a system that is easy to deploy, yet is powerful enough to mitigate DDoS attacks. Phalanx uses puzzle-based solution as OverDose to protect connection setup phase. Instead of using overlay nodes to filter malicious packets, Phalanx uses overlay nodes as temporary *mailboxes* of the server. Legitimate clients send different packets to different overlay nodes according to a pseudo-random pattern. In order to get legitimate packets, the server has to know which client sends which packet to which overlay node, and has to send requests to the correct overlay nodes as soon as possible to get the packets before the buffer of the overlay node becomes overloaded. With big number of clients, the overhead can be too high for the server to perform those operations for each packet from each client.

VIII. Conclusion

In this paper, we proposed SIEVE, which uses overlay nodes to form a lightweight distributed filtering system against DDoS attacks. SIEVE uses resource isolation and randomization to address the connection setup challenge, and provides guaranteed chance for the legitimate clients to set connections. SIEVE uses source addresses as lightweight authenticators to filter malicious traffic; this saves several orders of magnitude of computing power compared with common message authentication algorithms. By both analytical and simulation study, we showed that SIEVE can efficiently filter out attack traffic and save server's and router's bandwidth for legitimate traffic.

References

[1] Akamai Technology. http://www.akamai.com/, 2011.

[2] David G. Andersen. Mayday: distributed filtering for internet services. In *USITS'03*, pages 3–3, 2003.

[3] G. Badishi, A. Herzberg, and I. Keidar. Keeping denial-of-service attackers in the dark. *IEEE Trans. Dependable Secur. Comput.*, 4(3):191–204, 2007.

[4] Jean-Yves Le Boudec. Rate adaptation, congestion control and fairness: A tutorial, EPFL, December 2000.

[5] C. Dixon, T. Anderson, and A. Krishnamurthy. Phalanx: withstanding multimillion-node botnets. In *NSDI'08*, pages 45–58, 2008.

[6] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM '99*, pages 251–262, 1999.

[7] Z. Fu and M. Papatriantafilou. Off the wall: Lightweight distributed filtering to mitigate distributed denial of service attacks. *Technical report 2011-20, Chalmers University of Technology*, pages http://www.cse.chalmers.se/∼zhafu/Tech–2011–20.pdf, 2011.

[8] Z. Fu, M. Papatriantafilou, and P. Tsigas. Mitigating distributed denial of service attacks in multiparty applications in the presence of clock drifts. In *SRDS'08*, 2008.

[9] G. Gu, R. Perdisci, J. Zhang, and W. Lee. Botminer: clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *CCS'08*, pages 139–154, 2008.

[10] A. Hussain, John S. Heidemann, and C. Papadopoulos. A framework for classifying denial of service attacks. In *SIGCOMM*, pages 99–110, 2003.

[11] A. D. Keromytis, V. Misra, and D. Rubenstein. SOS: secure overlay services. *SIGCOMM Comput. Commun. Rev.*, 32(4):61–72, 2002.

[12] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iplane: an information plane for distributed services. In *OSDI '06*, pages 367–380, 2006.

[13] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. *SIGCOMM Comput. Commun. Rev.*, 32(3):62–73, 2002.

[14] Jelena Mirkovic and Peter Reiher. A taxonomy of DDoS attack and DDoS defense mechanisms. *SIGCOMM Comput. Commun. Rev.*, 34(2):39–53, 2004.

[15] Andrei Broder I Michael Mitzenmacher. Network applications of bloom filters: A survey. In *Internet Mathematics*, pages 636–646, 2002.

[16] Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson. Practical network support for ip traceback. *SIGCOMM Comput. Commun. Rev.*, 30(4):295–306, 2000.

[17] Elaine Shi, Ion Stoica, David Andersen, and Adrian Perrig. Overdose: A generic ddos protection service using an overlay network. *Technical report, Carnegie Mellon University, CMU-CS-06-114*, 2006.

[18] A. Stavrou and A. D. Keromytis. Countering dos attacks with stateless multipath overlays. In *CCS'05*, pages 249–259, 2005.

[19] Angelos Stavrou, Angelos D. Keromytis, Jason Nieh, Vishal Misra, and Dan Rubenstein. MOVE: An end-to-end solution to network denial of service. In *NDSS'05*, pages 81–96, 2005.

[20] The Network Simulator. http://isi.edu/nsnam/ns/, 2011.

[21] UMAC Performance. http://fastcrypto.org/umac/2004/perf04.html, 2004.

[22] Xiaowei Yang, David Wetherall, and Thomas Anderson. A DoS-limiting network architecture. In *SIGCOMM '05*, pages 241–252, 2005.