

A Detection and Offense Mechanism to Defend Against Application Layer DDoS Attacks

Jie Yu¹, Zhoujun Li²,

¹School of Computer
National University of Defense Technology
Changsha, China

Huowang Chen¹, Xiaoming Chen²

²School of Science & Engineering
Beihang University
Beijing, China

Abstract—Application layer DDoS attacks, which are legitimate in packets and protocols, gradually become a pressing problem for commerce, politics and military. We build an attack model and characterize layer-7 attacks into three classes: session flooding attacks, request flooding attacks and asymmetric attacks. We proposed a mechanism named as *DOW* (Defense and Offense Wall), which defends against layer-7 attacks using combination of detection technology and currency technology. An anomaly detection method based on K-means clustering is introduced to detect and filter request flooding attacks and asymmetric attacks. To defend against session-flooding attacks, we propose an encouragement model that uses client's session rate as currency. Detection model drops suspicious sessions, while currency model encourages more legitimate sessions. By collaboration of these two models, normal clients could gain higher service rate and lower delay of response time.

Keywords- DDoS Attacks; Detection; K-means; Currency; Encouragement

I. INTRODUCTION

While Internet has brought great convenience and efficiency to our lives, most of Internet users have been exasperated by computer viruses and network attacks. Denial of Service (DoS) attack is different in goal, form, and effect than most of the attacks that are launched at networks and computers [1]. In DDoS (Distributed DoS) attacks, breaking into a large number of computers and gaining malicious control of them is just the first step. The attacker then moves on to the DoS attack itself, which has a different goal—to prevent victim machines or networks from offering service to their legitimate users.

Most of past works have focused on detecting and defending DDoS attack on network layer [2, 3, 4, 5], which mostly detect DDoS by distinguishing the normal traffic and malicious traffic. There have been several researchful and commercial products of these defense methods up to now [1], such as SNORT [9], SPADE, MIDNS [10], Mazu Enforcer, Peak-flow, et al. However, these methods will lose effectiveness in the face of lay-7 attacks which make use of the asymmetric computation between client and server, as they are proper-looking requests from the protocol and traffic. For example, they can make queries of search engines, and issuing computationally expensive requests (e.g., database queries or transactions) [8]. This attack will delay response time of normal user, even refuse her access to service, which may be used by a com-

mercial competitor wishing to gain an edge in the market or political enemy trying to bring economy losing or panic. E-government, E-commerce et al. are the primary targets of lay-7 attack.

Users access application layer service in the form of session which is consisted of one or more requests. Similar to [6], we characterize layer-7 attacks into three classes: (1) *session flooding attacks* that send sessions at rates higher than normal users; (2) *request flooding attacks* that send requests at rates higher than normal sessions; and (3) *asymmetric attacks* that send high-workload request types in sessions. Attackers may use one or any combination of them.

Sun Tzu¹ said: security against defeat implies defensive tactics; ability to defeat the enemy means taking the offensive. Similarly, we think the best security strategy against DDoS attacks is to defend as well as offend. Offense helps defense, while defense improves offense. In our opinion, offending is to defending better.

In this paper, we first build an application layer threat model and then propose a mechanism to defend against the layer-7 DDoS attacks, which combines detecting and blocking method with currency method. Anomaly detection model uses session history to detect behavior of each session and encouragement model encourages clients whose sessions are dropped by randomly dropping model and refused by anomaly detection model to send more session connection requests again. We use client's session connection request ability which is denoted by its session rate as currency.

The organization of this paper is following. In Section 2, we describe the threat model based on the three types of layer-7 attacks. We design defense and offense mechanism in Section 3 and in Section 4, we further discuss this mechanism considering some important factors. Finally, we discuss related work in Section 5 and conclude in Section 6.

II. THREAT MODEL

The goal of an attacker is to overwhelm one or more server resources so that the normal clients experience higher delays or lower service rates thereby reducing or eliminating the capacity of the servers to its intended clients [6]. In this Section, we first

¹ One of the most famous strategists in China, whose magnum opus is *The Art of War*.

describe several assumptions, then abstract application model of layer-7, and finally introduce our attack model with simple analysis.

A. Assumptions

Before describing our attack model, we make a series of assumptions on the attackers' and servers' capability, which mostly accord with actual network environment.

Assumption 1: Attackers attack server through a series of sessions.

A session has four basic steps: (1) set up connection; (2) send request messages; (3) send response messages; (4) close connection. As all accesses to layer-7's service (e.g., HTTP, FTP, and TELNET) need the first and last steps and the difference of them is times of the second and third steps, we describe one such process as a session with one or more requests.

Assumption 2: Attackers could use a spoof IP address.

For datagram protocols without three-way handshakes (e.g., DNS-over-UDP), spoofing is trivial, and even for protocols with three-way handshakes, spoof is possible. Although using a spoof IP address might can't set up a connection correctly in a multi-request session, attackers can use different IP addresses in different sessions.

Assumption 3: Under DDoS attacks, the server is at least capable of rejecting the incoming packets and sending reply messages to the origins of these packets.

In other words, suppose that the attacker has commanded a very large number N of zombie computers distributed across a wide-range of geographical areas, the server has sufficient capacity to support a number of concurrent client sessions much larger than N and if each of these N session was a normal session, the server could serve the sessions within acceptable response times.

Assumption 4: Servers can describe each request's workload by a positive real number and attackers can test and find request with the highest workload by accessing to the public application interface.

For a server which has R request types, each request type i has a positive real number $Workload[i]$, which describes the interval it needs to complete the request. The higher $Workload[i]$ is, the longer time it's needed to deal with request i .

Assumption 5: A server is able to serve a single request at one time. It is easy to extend our model to concurrent requests service, which we will discuss later.

Assumption 6: Each request type needs the same resource of client as others, in despite of the workload of server it consumed.

Assumption 7: The service of the server consumes only the renewable resources, such as CPU cycles and bandwidth of network.

The last assumption has also been discussed in [7].

B. Application Model

Based on the above assumptions, we can abstract the server and client (normal or malicious) as follow:

- A **server** has a session-queue and a processor. Each unit of session-queue is a request-queue with length Lr , The session-queue has Ls units and the processor randomly fetches a request from the head of one request-queue immediately after it completes a request (based on assumption 5). Thus, we can describe a session-queue using a matrix $Request[Ls][Lr]$, the row of which denotes a session. When the session-queue is overloaded, it randomly drops excess sessions; also as the request-queue.
- A **client** can be denoted as a tuple $C = \langle S_{max}, R_{max} \rangle$, where S_{max} is maximal session rate of the client and R_{max} is maximal request rate of the client. A client can send sessions and requests at any rates under these limits. These two factors lie on both the resource of client machine and the network bandwidth it connects to server.

C. Attack Model

Each attacker sends traffic from many compromised hosts, and this traffic obeys all protocols, so the server has no easy way to tell from a single request that it was issued with ill intent [8]. We say a server is under a layer-7 attack when a surge in a resource usage is accompanied by reduction in service rate and increase in response time while no DDoS attack is detected at lower layers.

We characterize layer-7 attack model into three parts: session flooding attacks, request flooding attacks and asymmetric attacks.

- **Session Flooding Attack:** Each zombie computer sends sessions at a higher rate than normal users and the rate may change randomly.
- **Request Flooding Attack:** Each attack session sends requests at a higher rate than normal sessions and the rate may change randomly.
- **Asymmetric Attack:** Each attack session sends a higher proportion of requests, each of which has a higher value of $Workload[i]$.

As a server serves requests orderly per session, the attacker may choose to send more sessions with high workload requests. A cormorant attacker may send sessions and requests at the maximal rate and each of the requests is a highest workload request type, while a smart attacker may do this changeably to evade the detection and defense mechanism.

Suppose that normal clients send S_n session connection requests ($S_n < Ls$) and malicious clients send S_m session connection requests ($S_n + S_m > Ls$) in T seconds (no session is finished in this short interval), then only a fraction $Ls / (S_n + S_m)$ of the normal clients' sessions will be served. Assuming $S_m \gg Ls$ (because of the session flooding attack), this fraction is very small.

For all sessions in the session-queue, suppose that normal workload (the aggregate workload of all requests of normal sessions) is T_n seconds and malicious workload is T_m seconds, then the average delay of all normal requests is $(T_n+T_m)/2$ seconds. Because of the request flooding attack and asymmetric attack, T_m should be great longer than T_n and the average delay will be much long.

Above all, we can see that under the application layer attack, a few sessions of normal clients will be served; furthermore, these few sessions have to suffer much high delay.

III. DEFENSE AND OFFENSE MECHANISM

Our goal to defend and offend DDoS attacks is to 1) maximize the normal users' service rates, which denote their benefits from the server; and 2) minimize the normal users' delay for a service request. Both of these sub goals can be achieved by relatively reducing attack session rate, attack request rate and fraction of high-workload request.

This paper achieves the goal by combining a detecting and blocking method with a currency method. The detection method is to reduce the attack request rate and the fraction of high-workload request using abnormal session detection, while the currency method is to increase the normal session rate by encouragement, where the server causes a client, resources permitting, to automatically send a higher volume of traffic, which relatively reduces the attack session rate.

This mechanism, which we call as *DOW* (Defense and Offense Wall), is shown in Figure 1. The randomly dropping model drops sessions by the probability of overloaded session connection requests. If our mechanism was able to process new connection at all, the probability is zero. Anomaly detection model that uses the session history to detect every session's behavior will be described in Section 3.1. Encouragement model which encourages client whose sessions are dropped by randomly dropping model and refused by anomaly detection model to send more session connection requests again will be discussed in Section 3.2.

A. Anomaly Detection Model

Current DDoS detection methods can be classified into two strategies: misuse-based and anomaly-based. Misuse-based methods which rely on labeled data to train, generally can not detect new attacks, while anomaly-based methods is able to detect many different types of new attack. Anomaly detection detects anomalies in the data (i.e. data instances in the data that deviate from normal or regular ones) [12].

The anomaly packets detection mechanism employs a number of search algorithms [4], such as statistical analysis, deviation analysis, rule induction, neural abduction, making associations, correlations, and clustering [5, 11, 12, 13]. Most of these methods could be used to detect abnormal session behavior, too. In [6], Ranjan et al. design a suspicion assignment mechanism using deviation analysis, which build profiles for normal client behavior with respect to session inter-arrival times, request inter-arrival times, and session workload profile; they obtain the distributions of session inter-arrival and request

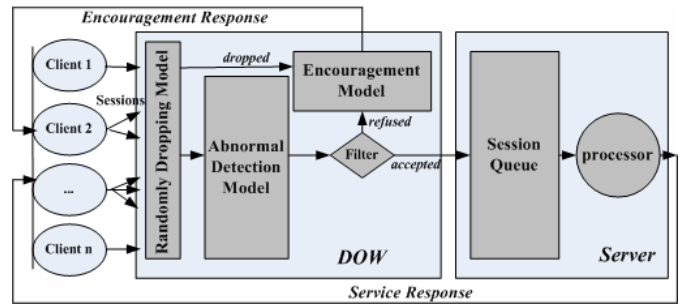


Figure 1. Defense and Offense Mechanism: *DOW*

inter-arrival by evaluating their empirical distribution respectively.

In our opinion, for an application layer service, there are several types of normal users whose sessions have different character. For example, we can classify normal users into three types: browsing users, shopping users and ordering users in an e-commerce application. We describe each type of normal users as session type that can be characterized by a set of request type and request interval: $\bigcup_{i=1}^N \{RT_i, RI_i\}$, where RT_i denotes the request type of i -th request, RI_i denotes the request interval between i -th request and $(i-1)$ -th request (especially, $RI_i = 0$ as it is the first request of a session).

Recall that in assumption 4, servers can describe each request type by its workload. However, it's not an easy work for server to describe its session type which characterizes the normal user behavior, due to the complexity and inconstancy of human behavior. We use a clustering method to obtain dynamic session types. Compared with classifications method that need both normal and attack data to train, clustering method just need normal data, which is easy to collect abundantly in real application.

Our anomaly detection model is described as a three-phase work. First, we build normal client behavior profiles using K-means method; second, we detect attacks by a cluster distance based method; finally, suspicious sessions are dropped by a filter based on the trust value of each session.

1. Training

Clustering is a well known and studied problem. Basic methods for clustering include the Linkage based [14] and K-means [15] techniques. Linkage based technique is much quick and useful for data with high dimensionality, while K-means method generally produces a more accurate clustering than linkage based methods. Because our training data just includes instances with 3-dimension and we'd build normal client behavior profiles offline, we use K-means as our clustering method.

- *Data Collection*

We start to collect data in the anomaly detection model when server is under normal state. When i -th request of a session comes, we will calculate the mean request workload MRW_i and mean request interval MRI_i of total i requests of this session immediately. Each instance is described as a vector: $\langle i, MRW_i, MRI_i \rangle$.

- *Normalization*

For the initial dataset we get, a different measurement of mean request interval may cause the clustering results differently, such as millisecond, second or minute; and it doesn't take into account the distribution that the input dataset is taken from. We'd normalize every element of instance vector as follow:

- 1) For element e , calculate the mean of all its values in dataset: $m_e = \sum_{i=1}^N x_{ie} / N$, where N is the number of instances.
- 2) Calculate the mean absolute deviation:
$$s_e = \sum_{i=1}^N |m_e - x_{ie}| / N.$$
- 3) Calculate normalized value, $z_{ie} = (x_{ie} - m_e) / s_e$.

We use the mean absolute deviation instead of normal distribution deviation, because it can reduce the influence of noise instances without square operation. Actually, this is a transformation of an instance from its own space to our standardized space, based on statistical information retrieved from the training set.

- *Clustering*

To create clusters from input normalized data instances, we use a standard K-means method which makes several passes through the training data and on each pass shifts cluster centers to the mean of the data points assigned to that cluster. It then re-assigns data points to the nearest prototype, and continues iterating in this manner until no significant changes in cluster center positions occur. A more detailed description of this algorithm can be found in [15].

A cluster can be denoted as a tuple $S = \langle X_m, r, p \rangle$, where X_m is mean vector of cluster S , r is radius of the cluster, and p is the proportion of sessions in the cluster to N total sessions.

Training process is done offline to avoid reducing the performance of our mechanism, and periodically, in order to keep newest update of normal user behaviors.

2. Detecting

Once the clusters are created from a training set, our mechanism is ready to perform detection of attacks. We describe a session's suspicion by a trust value t which is a real number between 0 and 1. When a new request of a session comes, we collect and normalize its signature vector as above. Given a normalized instance s , detection proceeds as follows:

- 1) Find a cluster $S_i = \langle X_{mi}, r_i, p_i \rangle$, which is closest to s and calculate their distance d .
- 2) If $d \leq r_i$, we just assign $t_s = 1$; otherwise, $t_s = \min(1, (r_i \times p_i \times L_c) / d)$, where L_c is the number of clusters.

We consider a session is normal when its signature is located in a normal cluster; otherwise, its trust value is inverse proportion to the distance to the nearest cluster S_i , and direct proportion to S_i 's session proportion and the number of clusters. Lower t_s is, more suspicious instance s is. The trust value will be used in next filter component.

3. Filtering

Under DDoS attack, incoming session rate of server is much higher than normal and most malicious sessions last longer than normal. The randomly dropping model drops sessions by the probability of overloaded session connection requests, and the session-queue must be full. To keep session-queue usable for new sessions (both normal and malicious), we should periodically drop some sessions in the queue. This component is to decide which sessions should be dropped.

As shown in figure 1, there is a filter in our *DOW*, which drops sessions based on the trust value of each session in session-queue. Suppose that we need drop α proportion of sessions in session-queue, follow strategies are proposed.

- *Foot-n*: All trust values of sessions are sorted in decrease order, we just drop last $n = \alpha \times L_s$ of this sequence.
- *Probability-n*: Given each session i one probability $p_i = t_i \times (1 - \alpha) \times L_s / \sum_{i=1}^{L_s} t_i$, and p_i denotes the probability it will be accepted. Then this session is dropped by the probability $1 - p_i$.

All dropped sessions are inputted to our encouragement model of *DOW*, which will be introduced in next Section.

B. Encouragement Model

In our application model (see Section 2.1), a server serves requests in a session from the head of the request-queue, thus, a later reached request has to wait until all requests before it have been served completely (Actually, most web servers e.g. Apache are implemented like this [6]). So, eager attackers may choose to send more sessions to retain high attack impact. In this Section, we introduce a currency method to defend against this type of attack.

In currency methods, an attacked server offers a client service only after it pays in some currency [8]. A usually used currency is puzzle [7], while we propose a method that calls currency as client's session connection request ability which is denoted by the session rate that lies on both the resource of client and the network bandwidth it connects to server. This method is based on the reality that malicious clients usually use most of their session rate to attack the server, while normal clients typically spend only a small fraction of it. In other words, our encouragement model is proposed to defend against session flooding attack.

When under layer-7 attack, because the session-queue is limited, randomly dropping model will drop session connection requests randomly to keep the queue full. Also, anomaly detection model starts to detect abnormal sessions and drop them through anomaly filter. **Our encouragement model, that uses both of above dropped sessions as inputs, immediately asks each input's client to retry using the same session.** To implement this, we just need to set the retry wait time as zero. In order to send repeated retries in pipeline way, we have to modify packets sending scheme to a congestion-controlled stream, which is discussed in [8].

In order to explain the effect of encouragement model, we consider the ideal scenario and just deal with inputs from

randomly dropping model. As the example in Section 2.3, we suppose that each client is encouraged to send N times retry connections. Malicious clients are already in the maximal session rate and normal clients don't use up session rate even increase to N times. Then, we can get that the fraction of the normal clients' sessions that will be served by server will increase from $Ls/(Sn+Sm)$ to $\min(1, Ls/(Sn+Sm/N))$.

Assuming that a session can be served in 10 seconds and the server's session-queue length is 1000 (thus its process rate is 100 sessions per second). One normal client's session rate is 0.1 session per second, and then a server can serve 1000 normal clients. Usually, the number of normal clients is much less than server's maximal service ability, otherwise, the owner of the server will try to increase the server's ability to make it greater than the usual consumption (by increasing server resources or/and deploying more servers). We assume that the number of normal clients $Nn=100$. When the number of malicious clients $Nm = 200$ and all malicious clients use the maximal session rate (2 sessions per second), Figure 2(a) describe the ideal fraction of the normal clients' sessions be served as the retry times N increases. The increase of ideal fraction is almost linearly with retry time N until $N = 4.4$. Figure 2(b) shows the maximal fraction of the normal clients' sessions be served as the number of malicious clients increases. The maximal fraction keeps 1 until number of malicious clients increase to 900 and reduces to 0.24 when $Nm = 4000$.

Another advantage of this model is it offers chances for legitimate sessions that are dropped by anomaly detection model to get service eventually. Some legitimate sessions may have lower trust values sometimes due to usual user's unusual behavior, thus they might be dropped by the filter of anomaly detection model. This advantage could help to reduce the false-positive rate of our mechanism.

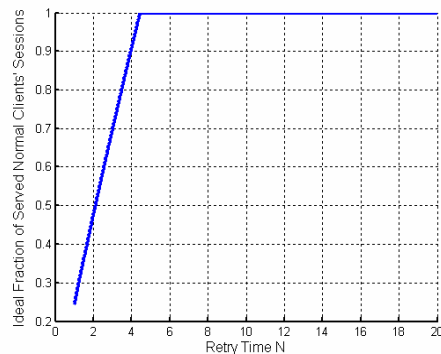
One of the most pivotal problems is network bandwidth. We consider follow facts: 1) most real DDoS attacks last very short interval; 2) most real DDoS attacks occur in a small range, we have not yet heard of a large scale of companies or governments being attacked simultaneously; 3) As technique develops rapidly, bandwidth will increase accordingly while the cost will reduce. Above all, we believe that network bandwidth is not a great obstacle to our encouragement model, especially for large applications which have enough bandwidth and other resources.

The effect of encouragement model should also be limited by the performance of client machines, the inclination of client users, the service rate of encouragement model, and so on. We will discuss some of them in Section 4.

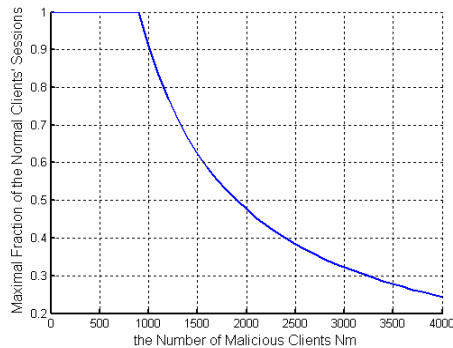
IV. MODEL ANALYSIS AND FURTHER DISCUSSION

Our *DOW* mechanism uses the combination of detection technology and currency technology, which has never been proposed as we know. Detection model drops suspicious sessions, while currency model encourages more legitimate sessions. By collaboration of these two models, normal clients could gain higher service rate and lower delay of response time.

In application model, we abstract server and client to simple prototypes based on our assumptions. We will discuss some extension of our mechanism in this Section.



(a) $Nn = 100, Nm = 200, N$ increases



(b) $Nn = 100, N = 20, Nm$ increases

Figure 2. Ideal Effect of Encouragement Model

1. Time Out

In anomaly detection model, a new trust value is calculated for a session when a new request of it comes. This might be a vulnerability used by attackers, as they may send requests normally at the first M requests, and then stop to send request or close session. In this situation, an element of session-queue will be hold for a long time.

Actually, we can set a timer for each session according to its former trust value. For example, $Timeout_{i+1} = t_i \times M \times RI_i$, where t_i is the trust value when i -th request comes, M is an adjusting factor greater than 1, and RI_i is the request interval between i -th request and $(i-1)$ -th request. If the $(i+1)$ -th request doesn't come in $Timeout_{i+1}$, the filter of anomaly detection model will drop it directly.

2. Process Schedule

We consider each session left in session-queue equally and processor of server randomly fetches request from them. Each session has an equal probability to be selected. We can apply some schedule strategies based on session's trust value reversing with filter strategies proposed in Section 3.1.

- *Top-one*: All trust values of sessions in session-queue are sorted in decrease order, we just select first one of this sequence.
- *Probability-one*: Given each session i one probability $p_i = t_i / \sum_{i=1}^{Ls} t_i$, then p_i is the probability it will be selected.

3. Concurrent Servers

In application model (Section 2.2), we simplify that processor of server disposes request one by one, while in real server, it processes request by time slice. We don't think this will change our model as the average time of each session keeps the same. In a large application e.g. E-commerce, there are a cluster of servers, which are scheduled by the *Load Balancers* above them. We'd like to consider all the servers as one super server, and apply our mechanism all the same. The difference is just average workload of each request.

4. Clients Performance

Actually, clients are very different from each other, including client machines, such as CPU, memory, disk et. al, and the network it connects to Internet, such as bandwidth. Session rate of clients is limited by both of them. For clients whose memories are key resource (means that their other resources are sufficient), fewer memories result in lower session rates and request rates, and eventually lower service rate.

Another factor affects client performance is the inclination of client user, which decides the proportion of resource she will use to launch this session request. Sometimes client user is doing other business and could not use up all her resource; while sometimes she would like to abandon this session and request it from another server (such as browsing another website) if there are replaceable servers, or even she just abandon it as it's a unimportant session. In these situations, maximal session rate and maximal request rate of client will also be limited.

Due to these limitations, the real effect of encouragement model would be lower than ideal effect. However, it indeed increases service rates and reduces response delays as normal sessions' probability of being served increases and proportion of malicious sessions reduces.

V. RELATED WORK

In this section, we will introduce some important previously proposed DDoS attack detection and defense mechanisms, and compare our work with [6].

DDoS threat can be countered at different locations in the network [1]. Each attack stream flows out of a zombie machine; through a server or router into the Internet; across one or more core Internet routers; into the router, server, or firewall machine that controls access to the target machine's network; and finally to the target itself. Defense mechanisms can be placed at some or all of these locations [2, 3, 4, 12, 13].

A source-end method is proposed in [17], it compares input and output network volume with normal volume model of each client to detect attack behavior. Perhaps the most standard approach against denial of service attacks at source end is the use of Ingress/Egress filtering techniques [18], where ISPs filter out packets with illegitimate source address, based on the ingress link by which the packet enters the network. Udaya K.T. et al propose a Controller-Agent model that would greatly minimize DDoS attacks on Internet. With a new packet marking technique and agent design, their scheme is able to

identify the approximate source of attack with a single packet even in case of attack with spoofed source addresses.

Trace Back Approach [19] is a usual method locating in the middle. An overlay network is created by deploying special tracking routers which links all the edge routers to a central tracking router. During an attack, the traffic to the victim is routed through the overlay network by dynamic routing. The largest challenge for source-end and middle-end defense methods is the incentive of users and ISPs to deploy it.

Source-end methods including our *DOW* mechanism are popular in research, as the incentive to deploy it is obvious. Most anomaly detection methods are source-end [3, 4, 5]. A speak-up method which encourages clients to send more requests is introduced in [8]; they use bandwidth as currency and do not consider different type of attacks in application layer. In [6], S. Ranjan et al. propose a counter-mechanism that consists of a suspicion assignment mechanism and a DDoS-resilient scheduler, *DDoS Shield*.

Compared with [6], our mechanism has great difference, such as: 1) we deal with each whole application layer service as a session, while they just consider session-oriented connections to server, like HTTP/1.1 session; 2) we defend session flooding attack by offense using session rate as currency, while they treat all income sessions as the same and use the deviation analysis; 3) we can train and detect automatically using a self-adapted process with high accuracy, while they obtain the distributions of session inter-arrival and request inter-arrival by evaluating their empirical distribution respectively; 4) our mechanism offers chances for dropped (both by randomly dropping model and anomaly detection model) legitimate sessions to get service eventually through encouragement model, however, they just drop them.

VI. CONCLUSION

In this paper, we first abstract the application model and attack model of layer-7, based on a series of assumptions we make. Application layer DDoS attacks are more subtle than network layer attacks; we characterize them into three classes: session flooding attacks, request flooding attacks and asymmetric attacks. Then, a mechanism named as *DOW* (Defense and Offense Wall), which defends against layer-7 attacks using combination of detection technology and currency technology is proposed. An anomaly detection method based on K-means clustering is introduced to detect and filter request flooding attacks and asymmetric attacks. To defend against session-flooding attacks, we proposed an encouragement model that using client's session rate as currency. Our further work is to implement our mechanism and deploy it in real network.

REFERENCES

- [1] J. Mirkovic, S. Dietrich, D. Dittrich, P. Reiher, "Internet Denial of Service: Attack and Defense Mechanisms", Prentice Hall PTR, 2004
- [2] Harjit S., Steven F. et al, "Investigating and Evaluating Behavioral Profiling and Intrusion Detection Using Data Mining", MMM-ACNS 2001, LNCS 2052, pp. 153-158, 2001
- [3] Abdelsayed S., Glimsholt D., Leckie C., Ryan S., Shami S., "An efficient filter for denial-of- service bandwidth attacks", In: Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM'03), San Francisco, USA, 2003, 3:1353-1357

- [4] Brugger S. T, "Data mining methods for network intrusion detection", Technique Report, UC davis, 2004
- [5] Chan E., Chan H., Chan K., Chan V., Chanson S. et al, "IDR: An intrusion detection router for defending against distributed denial-of-service (DDoS) attacks", In: Proceedings of the 7th International Symposium on Parallel Architectures, Algorithms and Networks 2004 (ISPAN'04), Hong Kong, 2004, 581~586
- [6] S. Ranjan, R. Swaminathan, M. Uysal, E. Knightly, "DDoS-Resilient Scheduling to Counter Application Layer Attacks under Imperfect Detection", INFOCOM'06, 2006
- [7] Lv J., Wang Y., Qiu W, "A Game-based Mechanism to defend Against DDoS Attacks", CISC'06, 2006
- [8] Michael Walfish, Mythili Vutukuru, et al. "DDoS Defense by Offense", SIGCOMM'06, 301~312, 2006
- [9] Snort: The Open Source Network Intrusion Detection System, <http://www.snort.org/>
- [10] Ertoz L. , Eilertson E. , Lazarevic A. , Tan P. et al. "Detection and Summarization of Novel Network Attacks Using Data Mining", Technical Report, 2003
- [11] Bass, T, "Intrusion detection systems and multi-sensor data fusion", Communications of the ACM 43(4), 99-105, 2000
- [12] Portnoy L., Eskin E., Stolfo S. J. "Intrusion detection with unlabeled data using clustering", In: Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA2001), Philadelphia, PA, 2001
- [13] Mohiuddin S., Hershkop S., Bhan R., Stof S. "Defending against a large scale denial-of-service attack", In: Proceedings of the IEEE. Workshop on Information Assurance and Security, 2001
- [14] H.H. Bock. "Automatic Classification", Vandenhoeck and Ruprecht, 1974
- [15] K. Fukunaga. "Introduction to Statistical Pattern Recognition", Second Edition, Academic Press, Boston, MA, 1990
- [16] Jiawei Han and Micheline Kamber, "Data Mining: Concepts and Techniques", Morgan Kaufmann Publishers Inc. 2001
- [17] Mirkovic J., Prier G. , "Attacking DDoS at the source", In: Proceedings of the 10th IEEE international Conference on Network Protocols, 312-321, 2002
- [18] Udaya Kiran Tupakula, Vijay Varadharajan. "A Practical Method to Counteract Denial of Service Attacks", ACSC'03, 2003
- [19] Robert Stone, "CenterTrack: An IP Overlay Network for Tracking DoS Floods", In proceeding of 9th Usenix Security Symposium, 2002