A Lightweight Mechanism to Mitigate Application Layer DDoS Attacks

Jie Yu[1,2], Chengfang Fang[2], Liming Lu[2], and Zhoujun Li[3]

[1] Department of Computer Science, National University of Defense Technology, China
[2] Department of Computer Science, National University of Singapore, Singapore
[3] School of Computer Science and Engineering, Beihang University, China
yj@nudt.edu.cn, {c.fang,luliming}@comp.nus.edu.sg, lizj@buaa.edu.cn

Abstract. Application layer DDoS attacks, to which network layer solutions is not applicable as attackers are indistinguishable based on packets or protocols, prevent legitimate users from accessing services. In this paper, we propose Trust Management Helmet (*TMH*) as a partial solution to this problem, which is a lightweight mitigation mechanism that uses *trust* to differentiate legitimate users and attackers. Its key insight is that a server should give priority to protecting the connectivity of good users during application layer DDoS attacks, instead of identifying all the attack requests. The trust to clients is evaluated based on their visiting history, and used to schedule the service to their requests. We introduce *license*, for user identification (even beyond NATs) and storing the trust information at clients. The license is cryptographically secured against forgery or replay attacks. We realize this mitigation mechanism and implement it as a Java package and use it for simulation. Through simulation, we show that *TMH* is effective in mitigating session flooding attack: even with 20 times number of attackers, more than 99% of the sessions from legitimate users are accepted with *TMH*; whereas less than 18% are accepted without it.

Keywords: DDoS Attacks, Trust, Lightweight, Application layer.

1 Introduction

Distributed denial of service (DDoS) attack refers to the attempt to prevent a server from offering services to its legitimate users, typically by sending requests to exhaust the server's resources, e.g. bandwidth or processing power. DDoS attack, which makes a server suffer in having slow responses to clients or even refusing their accesses, may be exploited by one's business competitors expecting to gain an edge in the market or political enemies trying to stir chaos. Since more and more efficient DDoS defense mechanisms and tools are proposed and installed on routers and firewalls, the traditional network layer DDoS attacks, such as SYN flooding, ping of death, Smurf, etc, are much easier to be detected and defended against. Nowadays, they are giving way to sophisticated application layer attacks [15]. Application layer DDoS attack is a DDoS attack that sends out requests following the communication protocol and thus these requests are indistinguishable from legitimate requests in the network layer. Most application layer protocols, for example, HTTP1.0/1.1, FTP and SOAP, are built on TCP and they communicate with users using sessions which consist of one or many requests. An application layer DDoS attack may be of one or a combination of the following types[15,24]: (1) *session flooding attack* sends session connection requests at a rate higher than legitimate users; (2) *request flooding attack* sends sessions that contain more requests than normal sessions; and (3) *asymmetric attack* sends sessions with more high-workload requests. In this paper, we focus on how to mitigate the session flooding attack.

Constrained by the bandwidth and processing power, application layer servers will set a threshold for the maximum number of simultaneously connected sessions to guarantee the quality of services. Under session flooding attack, a defense mechanism is needed by the server to reject attackers and to allocate the available sessions to legitimate users. The fraction of the rejection of requests from legitimate users over the total number of requests from legitimate users is called the False Rejection Rate (FRR), similarly, False Acceptance Rate (FAR) can be defined. Although a DDoS defense mechanism should reduce both FRR and FAR, reducing FRR is more important for the sake of user experience. That is, a server would rather maximally accommodate the legitimate user sessions, even if a small number of attacker sessions are not detected. Furthermore, the defense mechanism must be lightweight, to prevent itself from being the target of DDoS attacks. It is also preferred that the defense mechanism is independent of the details of the services, as then it can be deployed at any server without modification.

In this paper, we propose a lightweight mechanism, named Trust Management Helmet (*TMH*), that uses trust management to mitigate session flooding DDoS attack. For every established connection it records four aspects of trust to the user: short-term trust, long-term trust, negative trust and misusing trust which are used to compute an overall trust that helps in determining whether to accept a client's next connection request. These values are stored as part of a license at clients and when a client revisits the server, he attaches his license to the session connection request. Based on the license, *TMH* computes the client's overall trust, updates his license, and decides whether to accept his request. The license is designed such that the server can easily identify the client and verify his associated trusts, but license forgery or replay is computationally infeasible. We also extend *TMH* to collaborative trust management among multiple servers. Our mechanism is independent from services deployed on servers and is portable[1]. We have implemented it as a Java package and it can run separately and then redirect scheduled requests to servers protected or be integrated with other open-source application layer servers after slight modification.

As far as we know, our work is the first in applying trust management to application layer DDoS defense. Trust of a client is built up through his visiting history, and used as the criteria in evaluating the likelihood of the client being

[1] The mechanism is called as a helmet for this reason as well as its lightweight.

legitimate or not. Most existing schemes use packet rate as the metric to identify attackers. It is potential that intelligent attackers can adjust their packet rate based on server's response to evade detection [7]. In contrast, clients' visiting histories are hard to be modified, because servers keep access logs, and the data used in trust evaluation are secured using cryptography. Hence using trust as the evaluation criteria will be more reliable in application layer DDoS attacks.

The organization of this paper is as follows. In Section 2, we discuss related work. We describe the legitimate user model and attacker model in Section 3. We then propose our *TMH* defense mechanism in Section 4 and in Section 5, we simulate and analyze it. Finally, we extend *TMH* to collaborative trust management in Section 6 and conclude in Section 7.

2 Background and Related Work

There are extensive works on defending against network layer DDoS attacks with different strategies and heuristics, for example, anomaly detection [11], ingress/egress filtering [20], IP trace back [9,18], ISP collaborative defense [3], etc. Recently, the more sophisticated application layer DDoS attack [15] is threatening the security of the Internet content providers, especially web servers. One critical application layer DDoS attack is the index reflection attack [8]. In this attack, attackers declare to be the victim and pretend to share lots of resources in peer-to-peer network, so as to fool a large number of peers into requesting download of resources from the victim. It has been verified on many P2P applications, such as Gnutella [2], Bittorrent, Overnet [13], FastTrack [8], ESM [13], etc. We also verified this attack on Kad [25], which is the first DHT implemented in real applications and has millions of simultaneous users as to date. E. Athanasopoulos *et al.* [2] found that attackers can construct HTTP packets by misusing Gnutella protocol and then build new HTTP connections with victim (web servers) and download high workload resources. In index reflection attack, attackers do not need to control any botnets and thus it is very easy to perform this attack. Since application layer DDoS attacks are non-intrusive and protocol-compliant, attackers are indistinguishable based on packets or protocols and thus these attacks cannot be defended using network layer solutions. Clearly, new defense mechanisms are required for application layer DDoS attacks.

M. Walfish *et al.* [21] proposed a speak-up method, which encourages clients to send more session connection requests. This method is based on the assumption that attackers are already using most of their upload bandwidth so that they cannot react to the encouragement. S. Ranjan *et al.* [15] proposed a counter-mechanism by building legitimate user model for each service and detecting suspicious requests based on the content of the requests. S. Khattab *et al.* [6] proposed living baiting for applications that can be decomposed into several virtual services. It leverages group-testing theory to detect attackers with small state overhead. J. Yu *et al.* [24] introduced a detection and offense mechanism to protect legitimate sessions, but it is too resource consuming to be implemented. M. Srivatsa *et al.* [17] performed admission control to limit the number of concurrent clients served by the online service. Admission control is based on port hiding that renders the online service invisible to unauthorized clients by hiding the port number on which the service accepts incoming requests. This mechanism need a challenge server which can be the new target of DDoS attack. Y. Xie *et al.* [22,23] proposed a anomaly detector based on hidden semi-Markov model to describe the dynamics of Access Matrix and to detect the attacks. The entropy of document popularity fitting to the model was used to detect the potential application-layer DDoS attacks.

Trust management has been well studied in distributed systems to ensure the fairness in resource sharing or to evaluate the reliability of a resource provider. It has many potential applications in P2P networks. Trust management often uses peers' records, such as their upload and download data amount, or peer reviews, to build up trust information [4,16]. *P2PRep* [4] provided a protocol on top of Gnutella to estimate the trustworthiness of a node. M. Srivatsa *et al.* [16] identified three vulnerabilities of decentralized reputation management and proposed *TrustGuard* that let reputation grow slowly but drop quickly. In this paper, we apply trust management to defend against application layer DDoS attacks.

3 Legitimate User and Attacker Model

In this section, we build the legitimate user model, and the attacker model with several attack strategies of different complexity. Firstly, we would like to make two assumptions about the server.

Assumption 1. Under session flooding attacks, the bottleneck of a server is the maximal number of simultaneous session connections, called as *MaxConnector*. It depends not only on the bandwidth of the server, but also on other resources of the server, e.g. CPU, memory, maximal database connections.

Assumption 2. Without attacks, the total number of session connections of the server should be much smaller than *MaxConnector*, e.g., smaller than 20% of *MaxConnector*, as a server would set the threshold much higher to tolerate the potential burst of requests, e,g., flash crowds on websites.

3.1 Legitimate User Model

In contrast to attackers, legitimate users are people who request services for their benefit from the content of the services. Therefore, the inter-arrival time of requests from a legitimate user would form a certain density distribution $density(t)$ [5]. With this insight, we build the user model in the following way:

1. Use traces of Internet accesses to build an initial model $density_0(t)$, where t is a inter-arrival time and $density(t)$ is the probability a legitimate user will revisit the service after t seconds. Many traces has been done by researchers, e.g. F. Douglis *et al.* [5] traced web users to investigate caching technique in World Wide Web, and M. Arlitt *et al.* [1] presents a workload characterization study for Internet Web servers. Six different data sets are traced in this study: three from academic (i.e., university) environments, two

from scientific research organizations, and one from a commercial Internet provider.

2. Rebuild user model $density_{i+1}(t)$ with the newly collected inter-arrival times of all legitimate users after *TMH* runs d days under model $density_i(t)$, where d is randomly chosen from $[d_{min}, d_{max}]$. Note that we build the new density distribution using the data of legitimate users, whose requests are accepted by *TMH*. It means that $density_{i+1}(t)$ is tightly derived from $density_i(t)$ and hence is difficult to be fooled by attackers.

As a practical legitimate user model, it should satisfy the following properties: firstly, it should converge fast to the users' accesses interval distribution; secondly, it should be dynamic as the distribution may change from time to time; and most importantly, it should be lightweight to be easily implemented and monitored in the defense mechanism. The user model we proposed in this section can satisfy the first two requirements as the density function is updated regularly; and it is lightweight as the update to density distribution is incremental and it does not try to capture the complicated reasons for the changes reflected.

In our implementation, we employ the traces collected at AT&T Labs-Research and Digital Equipment Corporation by F. Douglis *et al.* [5] to build $density_0(t)$. In this initial density distribution model, there are a number of peaks in the user request arrival intervals, with the most prominent ones corresponding to intervals of one minute, one hour and one day. The mean inter-arrival time was 25.4 hours with a median of 1.9 hours and a standard deviation of 49.6 hours.

3.2 Attacker Model

The goal of session flooding DDoS attack is to keep the number of simultaneous session connections of the server as large as possible to stop new connection requests from legitimate users being accepted. Therefore, an attacker may consider using the following strategies when he controls a lot of zombie machines or can misuse P2P network as an attack platform as introduced in section 2:

1. Send session connection requests at a fixed rate, without considering the response or the service ability of victim.
2. Send session connection requests at a random rate, without considering the response or the service ability of victim.
3. Send session connection requests at a random rate and consider the response or the service ability of victim by adjusting request rate according to the proportion of accepted session connection requests by the server.
4. First send session connection requests at a rate similar to legitimate users to gain trust from server, then start attacking with one of the above attacking strategies.

The tradeoff of these strategies is between cost and ability to avoid the detection. Strategy 1 and 2 are easy to implement, but they are also easier to be detected; strategy 3 and 4 are more complicated as they consider the server responses or modeling legitimate users. Strategy 4 requires long-term preparation of attackers in order to gain a high trust level. This strategy needs attackers being more "patient". In session flooding attacks, attackers cannot spoof their IPs or change them within a session, because a session is set up on TCP connection which requires a three-way handshake. Since attackers cannot hide themselves through modifying IPs, they would prefer using strategy 3 and 4 to mimic behavior of legitimate users, to evade detection. We will simulate each strategy in Section 5.

4 Mitigation by Trust Management

We have considered the following properties in designing our mitigation mechanism: (1) It should be deployed at the server for incentive and performance reasons [14]. (2) It should be lightweight, to reduce the processing delay and to avoid being a new target of attacks. (3) It should be easy to deploy and independent to the details of servers. The defense mechanism need not know what services the server runs or what configuration it uses. (4) It should be adaptive to the server's resource consumption and differentiate between concurrent requests.

To evaluate the visiting history of clients[2] effectively, we use *trust*. The client who behaves better in history will obtain higher degree of trust. Here we define several components of it before defining trust.

Definition 1. Short-term trust T_s, estimating the recent behavior of a client. It is used to identify those clients who send session connection requests at a high rate when the server is under session flooding attacks.

Definition 2. Long-term trust T_l, estimating the long-term behavior of a client. It is used to distinguish clients with normal visiting history and those with abnormal visiting history.

Definition 3. Negative trust T_n, cumulating the distrust to a client, each time the client's overall trust falls below the initial value T_0. It is used to penalize a client if he is less trustworthy than a new client.

Definition 4. Misusing trust T_m, cumulating the suspicious behavior of a client who misuses its cumulated reputation. It is used to prevent vibrational attacks by repeatedly cheating for high trust.

Definition 5. Trust T, representing the overall trustworthiness of a client, which takes into account all of his short-term trust, long-term trust, negative trust and misusing trust.

To reduce the processing overhead brought by *TMH*, a short-term blacklist should be implemented. The blacklist records the list of clients whose trust values are too low. When a client's trust T drops below some threshold, he is recorded

[2] Clients are used to represent both legitimate users and malicious attackers in this paper.

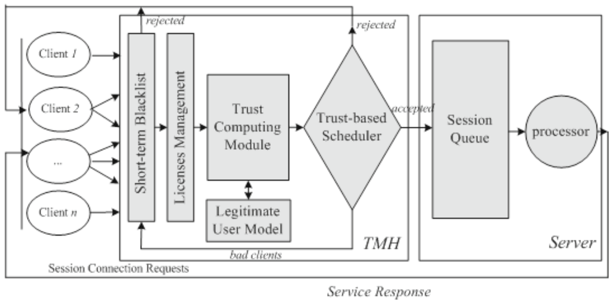


Fig. 1. The components of *TMH* and its communication with other modules

into the blacklist with an expiration time. He is then banned from accessing the server until his blacklist record expires.

The *TMH* mitigation mechanism is deployed at the server. A session connection request first reaches *TMH* and it checks whether the client is blacklisted; if not, it computes the trust to the client and use trust-based scheduling to schedule the connection request for the server. The architecture of *TMH* is shown in Fig.1. We will introduce the components of it in the rest of this section.

4.1 License Management

Because of mobile technology, users can make connections at different network segments. It is difficult to identify mobile users with dynamic IPs. Users connected from a proxy (e.g. HTTP proxy) are also difficult to be identified. Therefore, one way to mitigate session flooding attacks is to give priority in serving a subset of the good users, who we can identify and trust. The identification information and trust states can be stored at clients and verified by the server.

We call the information stored at clients as *license*. It contains the following: 64-bit identifer ID, IP address of client IP, the overall trust T to the client, negative trust T_n, misusing trust T_m, last access time LT, average access interval AT, the total number of accesses AN, and a keyed hash H of the concatenation of all the above, with a 128-bit server password SP as the key. SP is private to the server. Note that we identify a client by his public IP and the server assigned identifier. If IP address alone is used, clients behind NATs cannot be distinguished, because they share the same public IP address. Including the identifier ID enables uniquely identifying a client even if he is hidden behind NATs.

A license serves two functionality: for user identification and trust computation. The identification information, such as ID and IP, must be stored at the client license. The state variables for trust computation can be stored at the client or at the server. Each has its advantages and drawbacks. Keeping licenses at a server largely prevents attackers from tempering them, but it is a single point of data failure. Issuing licenses to users distributes the storage, making *TMH* more scalable in supporting clients, but the server needs to verify the authenticity of a license. It trades off between a server's storage and computation resources. We use client-based license for distributing the data and better scalability. The license can be dispensed to clients using cookies or by additional application layer protocols.

Client provides his license whenever he requests a connection. *TMH* verifies the license by first checking if the request originates from the IP address included in the license[3] and whether the last access time LT matches the server's log, then validating if the hash H agrees with the hash computed using the license and the server password SP. Connection request without a license will be treated as from new users and a new license will be issued if *TMH* decides to accept it. Note that an attacker can not change its IP address during a single session since a session must be set up according a full-TCP connection which needs three-handshake. Even in different sessions, an attacker is only able to change its IP address in a limited range, such as in a small network segment; otherwise, ISPs can not route handshake packets to the attacker.

Correctness. If the hash is one way and weak-collision resistant (e.g., MD5 or SHA-1), then a user is not able to create a valid license without the server password, except with negligible probability. The proof follows from the definition of weak-collision resistance and one way function.

Implementation. We mainly consider the protection of the most important application layer server on Internet, i.e. web server, which is also the most favourite target of known DDoS attacks. Cookies (RFC 2965, 2109) are small bits of textual information that a web server sends to a browser and that the browser returns unchanged when visiting the same web site or domain later. They are widespread used for convenient purposes, e.g. identifying a user during an e-commerce session, avoiding username and password, customizing a site, and so on. The default setting of most operation systems and browsers allow cookies. Hence, we employ them to keep the license information of the client. In our implantation, we use Java Servlet Cookie API to manage licenses. Table 1 shows the license set and get functions in our implantation of *TMH*. These functions are very lightweight and need little process power. Note that although each cookie is limited to 4KB, it is enough for us since each license needs only 544 bits.

4.2 Adaptive Trust Computing

The computation of trust $T^{'}$ employs T, T_n, T_m, LT, AT and AN in license, current time *now*, and *usedRate* (i.e., the percentage of connected sessions over *MaxConnector*) of the server. Based on Assumption 2 in Section 3, $usedRate \ll 1$ normally. As we explained, a server should give priority to protecting the connectivity of good users during session flooding attacks, instead of identifying all the attack requests. Since a higher trust value means a request is more likely to be accepted, it is desired to satisfy: $T_{legitimate\ user} > T_{new\ client} > T_{attacker}$.

[3] It is suggested that we only compare the class prefixes of IPs to support DHCP users.

Table 1. License management functions

```
void setLicense(String license, long maxAge){
  Cookie cookie = new Cookie("TMH.license", license);
  cookie.setMaxAge(maxAge);
  response.addCookie(cookie);
}
```

```
String getLicense(){
  Cookie[] cookies = request.getCookies();
  String license = ServletUtilities.getCookieValue(
                     cookies, "TMH.license", null);
  return license;
}
```

Short term trust is very important in distinguishing attackers, as almost all DDoS attacks are carried out in a relatively short period. For short-term trust, we consider both the interval of latest two accesses of the client and the current process ability of the server. Considering two different session connection requests with the same access interval at different arrival time, for the client when it arrives the server is relatively busy, it has a higher possibility to be an attacker and thus the short-term trust of it will be relatively lower. We give the formula of short-term trust as follows:

$$T_s^{'} = \frac{density(now - LT)}{e^{\alpha \times usedRate}} \tag{1}$$

where α is a weight factor deciding the influence of *usedRate*. It is a positive real number with default value 1 and can be modified by servers as needed. When $\alpha \approx 0$, the short-term trust mainly relies on the interval of the latest two accesses of the client.

Long term trust is the most important factor when a legitimate user builds up his credit. For long-term trust, the negative trust, average access interval and the total number of accesses should all be taken into account. They can represent the long-term behavior of a client. The formula of long-term trust is:

$$T_l^{'} = \frac{lg(AN) \times density(AT)}{e^{T_n}} \tag{2}$$

Using the short-term trust and long-term trust computed above and the misusing trust provided in license, we can then compute trust $T^{'}$ as follows:

$$T^{'} = min(2 \times \frac{\beta \times T_s^{'} + (1-\beta) \times T_l^{'}}{e^{T_m}},\ 1) \tag{3}$$

where $\beta \in [0, 1]$ with default value 0.5, it decides the weight of short-term trust and long term trust in the overall trust computation.

Negative trust is used to penalize users that have carried out attack, or carried out abnormal requests during periods that the server is busy. It cumulates the difference of trust T' to the initial value T_0 each time T' is smaller than T_0. The formula is as follows:

$$T_n^{'} = max(T_n + \gamma \times (T_0 - T^{'}),\ T_n) \tag{4}$$

Misusing Trust prevents vibrational attacks that repeatedly cheat for high trust by checking whether a user's trust is decreasing. It cumulates the difference in trust values if trust T' is smaller than the last time. The formula is as follows:

$$T_m^{'} = max(T_m + \gamma \times (T - T^{'}),\ T_m) \tag{5}$$

where $\gamma \in (0, 1]$, which is a weight factor deciding the degree of cumulation. It can be assigned by servers according to their demands with default value 1.

Recall that in above four formulas, T_n and T_m are the negative trust and misuse trust provided by the license respectively. For a client accessing the server for the first time, its initial value of the overall trust is 0.1, and its initial value of negative trust and misusing trust are both 0, i.e. $T_0 = 0.1$, $T_{n0} = T_{m0} = 0$.

Computation overhead. As can be seen from the formulas, the computation in updating a trust value is minimal. The major factor of computation overhead is in generating the cryptographic hash of a license. Yet each hash input is only 544 bits, and MD5 can compute more than 120,000 such hashes per second (measured in software using Java 5.0 and a PC with 2.13GHz CPU and 2GB memory). Even if using an off-the-shelf PC as a server, the server is capable of verifying more licenses than the normal network bandwidth can transmit. Besides, servers usually have more computational resources.

4.3 Trust-Based Scheduler

When a session connection request reaches *TMH*, it firstly validates the license the client provides. If passed, it will compute the client's new overall trust, negative trust and misusing trust and then update this information into the license. Afterwards, the scheduler in *TMH* decides whether to redirect it to the server based on the trust values.

TMH schedules session connection requests once every time slot. If the total number of the on-going sessions and the sessions waiting to be connected is not larger than the *MaxConnector* of the server, the scheduler will redirect all requests to the server. Otherwise, suppose there are N session connection requests waiting to be connected and the percentage of requests should be dropped is θ, we propose following the scheduling policies to drop suspicious requests:

1. *Foot-n*: sort all requests in current time slot by the clients' trusts in the decreasing order. For clients that have the same overall trust, sort them by their misusing trusts in the increasing order. We then drop the last $n = \theta \times N$ requests.

2. *Probability-n*: give each client i a probability p^i

$$p^i = min(\frac{T^i \times (1-\theta) \times N}{\sum_{j=1}^{N} T^j},\ 1) \tag{6}$$

to denote the probability at which his session connection request will be accepted. Thus we drop his request with probability $1 - p^i$.

4.4 Possible Attacks

We discuss some possible attacks to *TMH* in this subsection.

Index reflection attack. In Section 2, we described the index reflection attack. The peers manipulated into flooding session connection requests are either new users to the server or behave as attackers with strategy 4. During attack, *TMH* gives priority in serving the known users with high trusts. As the attacking peers frequently request for session connections, the trust of them drops till they are blacklisted. Thus they are penalized.

License forgery, replay and deletion attacks. Clients might not follow the protocol of *TMH* exactly, they might try to cheat about the license, for example, forging a new one, sharing a license, using an old license, or refuse to store a license.

As mentioned, the license is hashed with a server password, thus, it is computationally infeasible for a forgery to be valid. If attackers send random licenses to make *TMH* verify, *TMH* risks exhausting its computation power. However, the computation performed by *TMH* is lightweight, it can verify as many licenses in time as the network can transmit. And as the license stores the IP address, sharing a license is only possible for clients within the same subnet or NAT. Furthermore, since the last access time is included in the license, *THM* can detect if a client reuses an old license, by cross-checking the last access information the server logs, i.e. 64-bit identifer ID and last access time LT.

If an attacker discards his license of low trust to pretend to be a new user, he will still be assigned lower priority than the known users with high trusts. Additional efforts can be made to distinguish a benign new user and an apparent new user but who is actually a zombie attacker having discarded his license. For example, *TMH* can issue a request on the server's behalf, asking the new users to send their connection requests to another IP which is also under the server's control. If a user responds to the request and redirects his connections correctly, the user is not a zombie machine [19]. Graphical turing test [12] is another possible solution to tell apart zombies from benign new users.

5 Simulation

We implement *TMH* as a package, which consists of about 500 lines of Java source code. This package can run separately and then redirect scheduled requests to web servers or be integrated with other open source web servers after slight modification, such as Tomcat or JBoss. In this section, we present the simulation results to analyze the performance of *TMH* against different attack strategies and to compare the effect of different scheduling policies.

5.1 Simulation Setup

The simulation is set up in a local area network with 100Mbps links. We simulated 100 legitimate users, varying number of attackers and a server protected by *TMH*. Clients request the server for HTTP sessions. The server directly responds to them if they pass the verification and get scheduled by *TMH*.

Constrained by the server's memory and other resources, *MaxConnector* is set to 1000. That is, the server can serve maximally 1000 concurrent sessions; beyond that, the session requests will be dropped. In our simulation, legitimate users follow the model described in Section 3.1, we set d_{min}=15 and d_{max}=20; while attackers attack with different strategies described in Section 3.2. The life time of a session follows an exponential distribution with mean equals to 20 seconds.

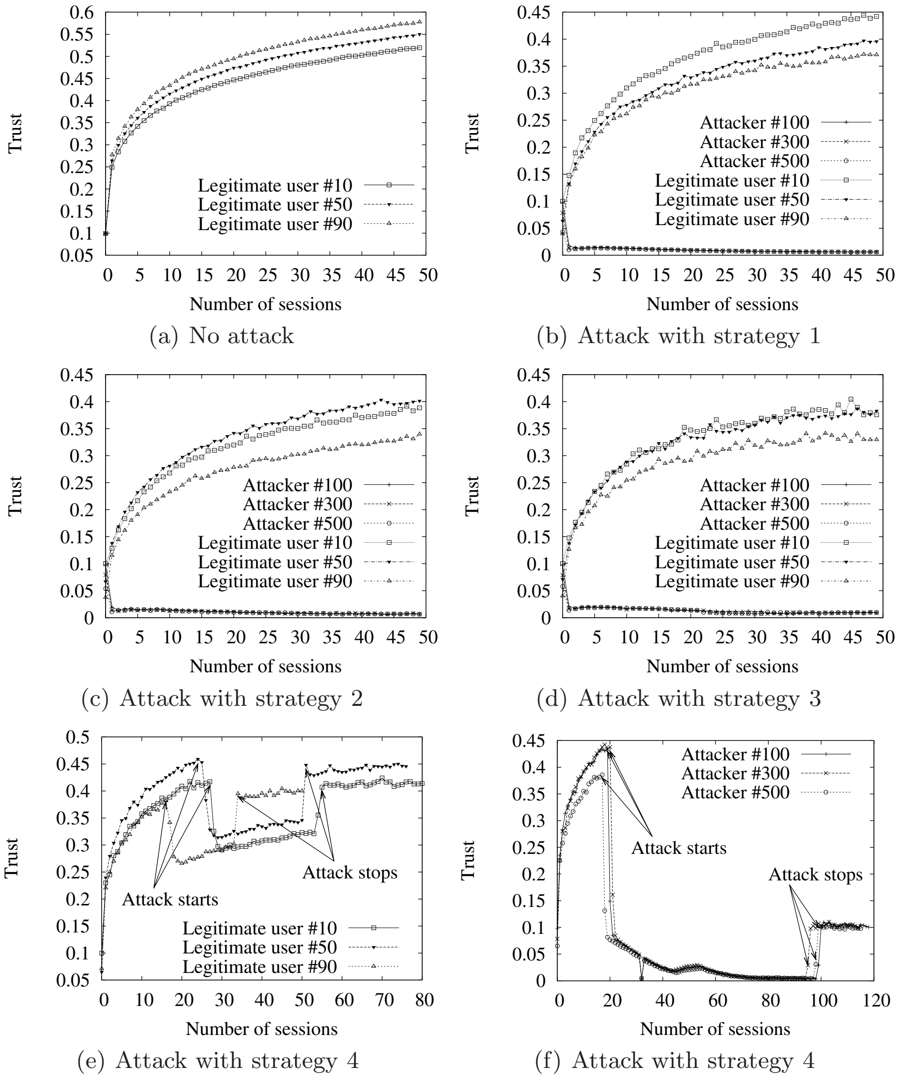
TMH uses default values of α, β and γ in the computation of trust. It issues license to new users with $density(now - LT)$ and $density(AT)$ set to be 0.1. After it verifies a license and updates the trust, it schedules the requests using the policies described in Section 4.3. For comparison, we also implemented two simple scheduling policies: (1) *Tail-n*: drop the $n = \theta \times N$ requests that arrive last in a time slot. (2) *Random-n*: randomly drop $n = \theta \times N$ requests in a time slot. A time slot is one second.

5.2 Results and Analysis

Fig.2 shows the change of overall trusts of legitimate users and attackers. Its result is obtained using *Probability-n* as the scheduling policy. For Fig.2(a), there are 100 legitimate users and no attacker; for Fig.2(b) to Fig.2(f), there are 500 attackers, besides the 100 legitimate users. All the users and attackers are started sequentially. In each simulation, the change of overall trusts of each legitimate user is very similar to each other. To illustrate this, we keep track of three representative users, that is, users started at the beginning, in the middle and at the end. Following the same argument, we select three attackers based on the starting sequence.

Fig.2(a) plots the trusts of three selected users when there is no attacker. All requests are accepted. It shows that the trusts of legitimate users quickly increase from 0.1 to 0.3 in the first few sessions. After 50 sessions, their trust values are over 0.5.

For Fig.2(b), attackers use strategy 1. They send session connection requests with a fixed rate at one request per 5 seconds. Fig.2(b) shows the trusts of legitimate users increase slower than in Fig.2(a). That is due to the high used rate of server's session connections under attacks. After 50 sessions, the trusts of legitimate users are around 0.4. However, the trusts of attackers decrease to around 0.01 in the first few sessions due to their high request rate, and they keep reducing slowly in the following sessions.



(a) No attack (b) Attack with strategy 1 (c) Attack with strategy 2 (d) Attack with strategy 3 (e) Attack with strategy 4 (f) Attack with strategy 4

Fig. 2. Global trusts over the number of sessions

For Fig.2(c), attackers use strategy 2. They send session connection requests with varying rate at one request in every 5 to 10 seconds uniformly. The randomness in attack rate causes the server to experience some burst of session requests. This decreases the misusing trust of legitimate users, which results in the fluctuation of their trust values, as shown in the figure. After 50 sessions, the trusts of the legitimate users are about 0.38.

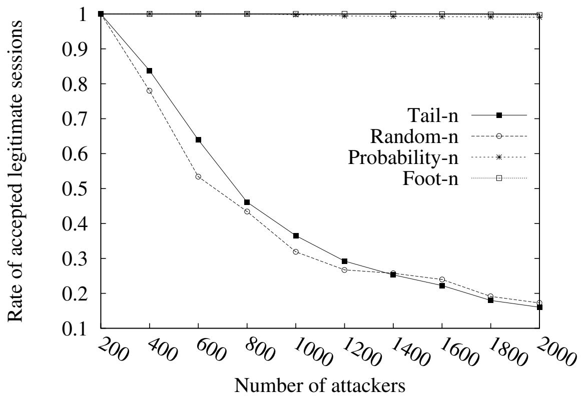


Fig. 3. The rate of accepted legitimate sessions over the number of attackers

For Fig.2(d), attackers use strategy 3. They adjust their sending rate between 5 to 10 seconds for a request, according to the rate of accepted requests by *TMH*. This attack strategy brings in more randomness to the used rate of server's session connections, and thus increasing the fluctuation of trusts of legitimate users. After 50 session requests, the trusts of legitimate users are about 0.35. Notice that the trust of attackers in Fig.2(b) is smaller than the trust of the same session in Fig.2(c) which is then smaller than the trust in Fig.2(d).

When attackers use strategy 4, the change of the overall trusts of three legitimate users are plotted in Fig.2(e) and that of three attackers are plotted in Fig.2(f). Attackers first send session connection requests complying with the legitimate user model; then attack using strategy 1 for a period; finally they follow the legitimate user model again. When attack starts or stops, the number of sessions requested by clients differs because the access interval in the legitimate user model is probabilistic. Fig.2(e) shows that the trusts of legitimate users decrease by 30% in three sessions after the attack starts. This is because of the sudden increase of used rate of server's session connections. However, they slowly increase even when the attack is carrying on and almost recover to about their original level after attack stops. Fig.2(f) shows that the trusts of attackers decrease to a much lower level and is at around 0.1 even they stop attacking. Note that we have removed some middle results under attacks in Fig.2(f) so that it is convenient to compare with Fig.2(e).

We compare different scheduling policies by plotting the acceptance rate of legitimate sessions, i.e. $1 - FRR$, in Fig.3. The number of legitimate users is 100 and the proportion of each kind of attackers who adopt one of the four attack strategies is 25%. We can see that under trust-based scheduling strategies, the acceptance rate of legitimate sessions keeps at a high level and is insensitive to the number of attackers. Even when the number of attackers is 2000, the acceptance rate of legitimate user sessions is still 99.1% and 99.7% with *Probability-n* and *Foot-n* scheduling policies respectively. However, it is only 16.0% using *Tail-n* policy and 17.2% using *Random-n* policy.

6 Collaborative Trust Management

Many services are related but provided by different servers, e.g., online auction and PayPal, or e-newspaper and its advertisements. The related servers probably share a large group of clients. For the common good or economic incentives, *TMH*s deployed at these servers can collaborate with one another by sharing trust information of clients. We say these *TMH*s form a collaboration group. The trust information shared is particularly useful in distinguishing legitimate users. The collaborating *TMH*s can take either or both actions below:

1. *Exchange blacklist*: a *TMH* exchanges its blacklist (including client ID and its expiration time) with other *TMH*s periodically. When a *TMH* receives a blacklist, it merges the received blacklist into its own.
2. *Exchange the trust values of clients*: a *TMH* sends its overall trust of clients to other *TMH*s periodically. A client may visit the same server multiple times within a period. Only the latest overall trust logged by *TMH* is exchanged. When client j requests a new session, *TMH* i uses following formula to recompute the overall trust T_G^j of the client, which considers both the local trust T^j computed by *TMH* i and the recommended trust T_r^j from other *TMH*s:
$$T_G^j = \tau \times T^j + (1 - \tau) \times \sum_{r \in I(i)} \frac{C_{ir} \times T_r^j}{\sum_{r \in I(i)} C_{ir}} \qquad (7)$$
where $I(i)$ is the collaboration group of *TMH* i, C_{ir} is the confidence level that *TMH* i has for the recommended trust from *TMH* r, and $\tau \in [0, 1]$ is the confidence level of *TMH* i itself, 1 means the most confident and 0 the least.

The collaboration of *TMH*s can reduce the false negatives of a single *TMH* and accelerate the identification of attackers.

7 Conclusion

To defend against application DDoS attacks is a pressing problem of the Internet. Motivated by the fact that it is more important for service provider to accommodate good users when there is a scarcity in resources, we present a lightweight mechanism *TMH* to mitigate session flooding attack using trust evaluated from users' visiting history. We verify its effectiveness with simulations under different attack strategies. Comparing to other defense mechanism, *TMH* is lightweight, independent to the service details, adaptive to the server's resource consumption and extendable to allow collaboration among servers. In future work we will investigate how to apply *TMH* into real-world applications and how to defend against other types of application layer DDoS attacks, including request flooding attack and asymmetric attack.

References

1. Arlitt, M., Williamson, C.: Web Server Workload Characterization: The Search for Invariants. In: Proceedings of ACM SIGMETRICS 1996 (1996)
2. Athanasopoulos, E., Anagnostakis, K., Markatos, E.: Misusing Unstructured P2P systems to Perform DoS Attacks: The Network That Never Forgets. In: Zhou, J., Yung, M., Bao, F. (eds.) ACNS 2006. LNCS, vol. 3989, pp. 130–145. Springer, Heidelberg (2006)
3. Chen, Y., Hwang, K., Ku, W.: Collaborative Detection of DDoS Attacks over Multiple Network Domains. IEEE Transations on Parallel and Distributed Systems (2007)
4. Cornelli, F., Damiani, E., Vimercati, S., Paraboschi, S., Samarati, P.: Choosing reputable servents in a p2p network. In: Proceedings of WWW 2002 (2002)
5. Douglis, F., Feldmannz, A., Krishnamurthy, B.: Rate of change and other metrics: a live study of the World Wide Web. In: Proceedings of USENIX Symposium on Internetworking Technologies and Systems (1997)
6. Khattab, S., Gobriel, S., Melhem, R., Mossäe, D.: Live Baiting for Service-level DoS Attackers. In: Proceedings of INFOCOM 2008 (2008)
7. Li, Q., Chang, E., Chan, M.: On the Effectiveness of DDoS Attacks on Statistical Filtering. In: Proceedings of INFOCOM 2005 (2005)
8. Liang, J., Naoumov, N., Ross, K.W.: The Index Poisoning Attack in P2P File Sharing Systems. In: Proceedings of INFOCOM 2006 (2006)
9. Lu, L., Chan, M., Chang, E.: Analysis of a General Probabilistic Packet Marking Model for IP traceback. In: Proceedings of ASIACCS 2008 (2008)
10. Mirkovic, J., Dietrich, S., Dittrich, D., Reiher, P.: Internet Denial of Service: Attack and Defense Mechanisms. Prentice Hall PTR, Englewood Cliffs (2004)
11. Mirkovic, J., Prier, G.: Attacking DDoS at the source. In: Proceedings of ICNP 2002 (2002)
12. Morein, W.G., Stavrou, A., Cook, D.L., Keromytis, A.D., Misra, V., Rubenstein, D.: Using graphical turing tests to counter automated DDoS attacks against web servers. In: Proceedings of ACM CCS 2003 (2003)
13. Naoumov, N., Ross, K.: Exploiting P2P Systems for DDoS Attacks. In: Proceedings of INFOSCALE 2006 (2006)
14. Natu, M., Mirkovic, J.: Fine-Grained Capabilities for Flooding DDoS Defense Using Client Reputations. In: Proceedings of LSAD 2007 (2007)
15. Ranjan, S., Swaminathan, R., Uysal, M., Knightly, E.: DDoS-Resilient Scheduling to Counter Application Layer Attacks under Imperfect Detection. In: Proceedings of INFOCOM 2006 (2006)
16. Srivatsa, M., Xiong, L., Liu, L.: TrustGuard: Countering Vulnerabilities in Reputation Management for Decentralized Overlay Networks. In: Proceedings of WWW 2005 (2005)
17. Srivatsa, M., Iyengar, A., Yin, J., Liu, L.: Mitigating application-level denial of service attacks on Web servers: A client-transparent approach. ACM Transactions on the Web (2008)
18. Stone, R.: CenterTrack: An IP Overlay Network for Tracking DoS Floods. In: Proceeding of 9th Usenix Security Symposium (2002)
19. Thing, V.L.L., Lee, H.C.J., Sloman, M.: Traffic Redirection Attack Protection System (TRAPS). In: Proceedings of IFIP SEC 2005 (2005)
20. Tupakula, U., Varadharajan, V.: A Practical Method to Counteract Denial of Service Attacks. In: Proceedings of ACSC 2003 (2003)

21. Walfish, M., Vutukuru, M., Balakrishnan, H., Karger, D., Shenker, S.: DDoS Defense by Offense. In: Proceedings of SIGCOMM 2006 (2006)
22. Xie, Y., Yu, S.: Monitoring the Application-Layer DDoS Attacks for Popular Websites. IEEE/ACM Transactions on Networking (2009)
23. Xie, Y., Yu, S.: A large-scale hidden semi-Markov model for anomaly detection on user browsing behaviors. IEEE/ACM Transactions on Networking (2009)
24. Yu, J., Li, Z., Chen, H., Chen, X.: A Detection and Offense Mechanism to Defend Against Application Layer DDoS Attacks. In: Proceedings of ICNS 2007 (2007)
25. Yu, J., Li, Z., Chen, X.: Misusing Kademlia protocol to perform DDoS attacks. In: Proceedings of ISPA 2008 (2008)