Noname manuscript No.
(will be inserted by the editor)

ConnectionScore: A Statistical Technique to Resist Application-layer DDoS Attacks

Hakem Beitollahi · Geert Deconinck

Received: date / Accepted: date

Abstract In an application-layer distributed denial of service (DDoS) attack, zombie machines send a large number of legitimate requests to the victim server. Since these requests have legitimate formats and are sent through normal TCP connections, intrusion detection systems (IDS) cannot detect them. In these attacks, an adversary does not saturate the bandwidth of the victim server through inbound traffic, but through outbound traffic. The next aim of the adversary is to consume and exhaust computational resources (e.g., CPU cycles), memory resources, TCP/IP stack, resources of input/output devices, etc. This paper proposes a novel scheme which is called ConnectionScore to resist such DDoS attacks. During the attack time, any connection is scored based on history and statistical analysis which has been done during the normal condition. The bottleneck resources are retaken from those connections which take lower scores. Our analysis shows that connections established by the adversary give low scores. In fact, the ConnectionScore technique can estimate legitimacy of connections with high probability. The rate of suspicious connections being dropped is adjusted based on the current level of overload of the server and a threshold-level of free resources. To evaluate the performance of the scheme, we perform experiments in the Emulab environment using real traceroute data of the ClarkNet WWW server[1].

Keywords Distributed Denial of Service Attacks · Connection Score · Application-layer DDoS · HTTP flood

H. Beitollahi
Soran University, Kurdistan region, Iraq
E-mail: Hakem.Beitollahi@soran.edu.iq

G. Deconinck
University of Leuven, Electrical Engineering Department, Kasteelpark Arenberg 10, Leuven, Belgium
E-mail: Geert.Deconinck@esat.kuleuven.be

[1] http://ita.ee.lbl.gov/html/contrib/ClarkNet-HTTP.html

1 Introduction

In the literature on DDoS attacks, an adversary normally uses IP spoofing to prevent disclosing location of its zombie machines (aka bots) [16,20]. Also in the literature of DDoS attacks, it has been stated that the goal of the adversary is to overwhelm the bottleneck resources of the victim server by flooding of **bogus** packets [16,20,9]. For instance, in a bandwidth attack, the adversary tries to saturate the bandwidth through a flood of **inbound** traffic; i.e., sending a flood of bogus large packets toward the server. In a SYN flood attack, the goal of the adversary is to exhaust the TCP/IP stack through sending a large number of bogus SYN requests without taking the third step of the 3-way TCP handshaking.

In contrast, this paper focuses on a more difficult DDoS problem in which an adversary attempts to overwhelm the server through zombie machines via **legitimate** requests. In this attack type which is called application-layer DDoS attack, any zombie machine has to establish a TCP connection with the victim server, which requires a genuine IP address; otherwise, the TCP connection cannot be established. Although, in this attack model, the IP addresses of zombie machines cannot be spoofed, the adversary does not worry about disclosing the IP addresses of zombie machines. The fact is that today social sites such as Facebook, Twitter, Yahoo messenger, etc., and software tools such as VoIP tools, etc., have provided many zombie machines for the adversary. Most users of these sites and software tools have little knowledge of computer and network security and more interestingly, most of these users leave their computer system online for long times. On the other hand, today, improvements of technology have provided high speed (high bandwidth) Internet for such users. All of the above evidences show that power of attackers to perform DDoS attacks, where the adversary does not need to hide the location of its zombie machines, is rapidly expanding.

In application-layer DDoS attacks, requests of zombie machines cannot be distinguished from requests of normal users since both of them have a legitimate format and are sent via normal TCP connections. Consequently, Intrusion Detection Systems (IDS) fail to detect these. In these attacks, an adversary does not overwhelm the bandwidth of a victim server through flooding the server with inbound traffic, but through saturating the outbound traffic. In other words, the adversary sets the zombie machines to legitimately and frequently download files from the victim server and consequently, overwhelms the bandwidth of the victim through outbound traffic. The next aim of application-layer DDoS attacks is that the adversary attempts to exhaust any limited resources of the victim server such as TCP buffers, CPU cycles, memory, input/output device resources, etc.

HTTP flood [27] is one of the most popular application-layer DDoS attacks. The World Wide Web (WWW) is one of the most popular applications on the Internet. WWW applications generally use the Hypertext Transfer Protocol (HTTP) over TCP port number 80. Most firewalls on the Internet leave this port open to allow HTTP traffic to pass. In the HTTP flood attack, the adversary sets the zombie machines to bombard the victim server with HTTP requests. To saturate the bandwidth of the victim server, the adversary can set the zombie machines to request large pages. The victim then has to read the page from the hard disk, store it in its memory, load it into packets and then send the packets to the zombie machines. As can be seen, a simple HTTP flood attack can kill four birds with one stone: (1) it can overwhelm the bandwidth through outbound traffic, (2) it can exhaust memory, (3) it can exhaust CPU cycles and (4) it can exhaust input/output device resources. In the case of a large number of connections it can even exhaust the TCP/IP stack. Due to its attractiveness, HTTP floods have become a common feature in most botnet software programs [13]. FTP flood is another example of such attack.

Previous well-known DDoS countermeasures cannot tackle this type of attacks (see next section). CAPTCHA puzzles [17,21] have been proposed against these attacks. However, CAPTCHA puzzles suffer from some challenges (see next section). This paper proposes a novel, cheap and systematic technique against these attacks which is called ConnectionScore technique. Our goal is to design a technique that tackles application-layer DDoS attacks without using CAPTCHA puzzles or with a minimum level of CAPTCHA puzzles.

The ConnectionScore technique proposes that during normal conditions, any server can measure various statistical attributes for its users and their traffic. The statistical attributes represent the behavior and the characteristic of normal users. A server can keep the statistical attributes as a reference profile. Now, when an attack occurs against the server, the server assigns scores to the connections based on the reference profile. It retakes bottleneck resources from those connections which have low scores. The key point is that the connections which have been established by the attackers get low scores because they cannot have statistical attributes of the normal users. The reason is that first, only the server knows what the statistical attributes of its users are and second even if attackers attempt to have some attributes close to attributes of normal users, they cannot launch an effective attack against the server. Dropping of suspicious connections is adjusted based on the current level of the overload of the server.

To evaluate the performance of the scheme, we focus only on a HTTP flood attack though our scheme can be used for other application-layer DDoS attacks. We perform experiments in the Emulab environment using real logged data of the ClarkNet WWW server as a case study. Our experiments show that the ConnectionScore scheme can precisely detect malicious connections and retake the bottleneck resources from them.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 describes the ConnectionScore scheme. Section 4 presents and analyzes statistical attributes for the case study. Section 5 shows the experimental results and finally Section 6 concludes the paper.

2 Related work

Both academic and industrial researchers have proposed various defense techniques against network-layer DDoS attacks. However, those techniques cannot be used as a remedy against application-layer DDoS attacks. The techniques proposed for SYN flood attacks [5,10] cannot be used as a remedy for the application-layer DDoS attacks because all attackers successfully complete the 3-way TCP handshaking and establish TCP connections. The cooperative techniques [8,29,9] which have been proposed against TCP flood, UDP flood and ICMP flood cannot be used as a countermeasure against application-layer DDoS attacks as cooperative techniques prevent saturation of the victim server's bandwidth through controlling and rate-limiting the inbound traffic. In our attack model, however, attackers attempt to overwhelm the server's bandwidth with outbound traffic. The techniques which have been proposed for IP traceback [23,1] are useless for this attack type as all attackers use their real IP address to attack the server. Consequently there is no need for IP traceback techniques. In this paper, we do not review countermeasure techniques against network-layer DDoS attacks. A useful survey that collects network-layer DDoS countermeasures is [4]. Although research on countermeasures against application-layer DDoS attacks is young, some interesting techniques have been proposed during recent years. This paper reviews some of them.

The most promising technique against application-layer DDoS attacks is CAPTCHA puzzles. A CAPTCHA puzzle [21] is a type of challenge-response test used in computing as an attempt to ensure that the response is generated by a human not by a machine. However, the CAPTCHA solution has three major challenges. (1) Patience of the users: several reports [6,11] show that these tests annoy the users and they are not user-friendly. Since many users have little patience to solve a CAPTCHA test and wait for response, a site that uses CAPTCHA may drive away legitimate users. (2) Breaking techniques: today, several image recognition techniques have been proposed to break CAPTCHAs [18]. (3) Labor attack: some reports [2,26] indicate that there are free or cheap 3rd party human labor to break CAPTCHAs.

Gavrilis et al. [12] proposed "Decoy Hyperlink" to detect zombie machines. The decoys are hyperlinks without semantic information and are invisible to the human users, acting like traps for DDoS attacks because a human user would never follow them. A zombie machine is detected when such hyperlink is followed. The authors assume that an attacker scans all hyperlinks in a page and follows all hyperlinks, but if an attacker only follows a fraction of the hyperlinks of a page, then it is very likely that the decoy hyperlinks are not selected. Moreover, as an attacker can solve CAPTCHAs using pattern recognition techniques, he can recognize decoy hyperlinks by designing similar tools. When decoy hyperlinks are invisible for human users, the attacker can simply design a tool and detect such hyperlinks.

Yatagai et al. [27] proposed two simple ideas: a) when there are attacks from compromised clients with the same virus or bot, the server can observe the same browsing order of pages continually at the server, 2) attackers browse a web page for a shorter time than normal users; thereby if a user browses a web-page shorter than a threshold time, it is considered as malicious. The first idea does not work as the attacker can set zombie machines to send requests for random pages. The second idea also does not work as the attacker can browse a web-page for a longer time and simply pass the threshold.

Thapngam et al. [25] classified attack rates into two categories: predictable rates and non-predictable rates. Predictable rates include constant rate, monotonically increasing rate and periodical rate. However, non-predictable rates have no classification. The author then proposes a Pearson correlation coefficient theorem to detect predictable rates for all three classes. However, they have no solution when attackers send requests at random and unpredictable rates.

Xie and Yu [28] proposed a hidden semi-Markov model for anomaly detection of browsing behavior. The authors assume that normal users always access pages sequentially based on hyperlinks organization, while attackers do not follow this organization and access random pages directly using their URL. Then authors recognize attackers through the entropy test. Even if, the first assumption is true, the second assumption which is the base of the algorithm is not always true. We note that an attacker can easily design a tool and ask zombie machines to follow pages based on the hyperlink organization. In this case, the entropy value of attackers and normal users locate in the same range and the server cannot detect zombie machines. The second challenge of this method is its algorithmic complexity.

Oikonomou and Mirkovic [19] proposed a countermeasure based on human behavior modeling which recognizes DDoS botnet machines from human users. The basic of the technique is based on three aspects: request dynamics, request semantics and ability to process visual cues. According to their technique, they should build a possibility graph for a website. The major challenge is that for most large websites with too many dynamic pages and objects inside, finishing construction of the graph is hardly done or never done.

Paul Barford et al. [3] used wavelets to distinguish the legitimate requests from malicious DoS requests. However, the method is a post-mortem technology and cannot stop the attack on-line.

Ranjan et al. [22] proposed a counter-mechanism that consists of a suspicion assignment mechanism and a DDoS-resilient scheduler, DDoS Shield. The suspicion mechanism assigns a value to each client session in proportion to its deviation from legitimate behavior according to three parameters: session arrivals, session request arrivals and session workload profiles. Then the DDoS shield decides when and where a session is serviced. First, we believe those three parameters cannot determine deviation from legitimate behavior accurately and second, the DDoS shield cannot actively block the malicious traffic.

3 Description of the ConnectionScore scheme

The basic idea of the ConnectionScore scheme is as follows. A server can measure various statistical attributes for its users and their traffic during the normal condition when there is no attack against it. Undoubtedly, the measured statistical attributes represent behavior and characteristics of the normal users of the server. These attributes are site-dependent which means that an outside attacker cannot be aware of such statistical attributes. The server can consider the measured statistical attributes as a reference profile and use it during the attack time as a judgment reference point. When an application-layer DDoS attack occurs against the server, the server assigns a score to connections based on the reference profile. As can be seen below, the connections which get lower scores are more probable to be the connections which have been established by the attackers. In fact, the ConnectionScore scheme predicts legitimacy of connections with high probability. In the next step, in a feedback-control process, the server retakes bottleneck resources from the connections which have lower scores until its current load reaches below a threshold.

3.1 Attributes

A server can consider various attributes for users and their traffic. In this paper, we introduce some of them, though other attributes can be discussed.

Request rate

The request rate represents the number of requests that a user sends to the server in a specific interval time. The server measures request rate for different random users during different random times a day, different days, different weeks, etc. Then the server can find the cumulative distribution function (CDF) for request rate of normal users. Also, it can measure the average amount and standard deviation of this attribute.

Download rate

The download rate represents the number of bytes that a user downloads from the server during a specific interval time. Similarly, the server can measure download rate by choosing different random users during different times a day, different days, different weeks, etc. Hence, the server discovers CDF for download rate of its normal user and meanwhile it calculates the average and standard deviation for this attribute.

Uptime

The uptime represents a time that a user starts communication with the server until he terminates the communication with the server. The server randomly selects users during different times (days, weeks, etc.) and calculates their uptime. Similarly, the server can obtain CDF, average and standard deviation for uptime of normal users.

Downtime

The downtime represents the interval time from the time that a specific user disconnects from the server until the time he connects again to the server. Although the downtime attribute cannot be measured for all users as some users may either never connect again to the server or connect to the server too late, surely some users reconnect every few days or several times a day. In such case, the server can measure the downtime for such users and also discover CDF, average and standard deviation for this attribute[2].

Browsing behavior

The browsing behavior of users of a web-site depends on two main factors: a) the structure of the website and b) the behavior of users. Normally, any web-server consists of many web-pages that have been organized hierarchically through hyperlinks. In fact, a typical webpage contains a number of hyperlinks to point to other pages. The behavior of users indicates that which pages are more favorite for users (page popularity). Which fraction of hyperlinks on a typical page is clicked by normal users? We will also discuss other behaviors below. In fact, we discuss this attribute via several sub-attributes.

Sequential-hyperlink pages

The structure of most websites is hierarchical, based on hyperlinks such that a user cannot access to a particular page directly unless he has accessed some sequence of pages prior to that particular page. However, it is possible that a user directly accesses such pages by typing their URL in the browser. During normal condition, a server can predict the percentage of such accesses through its history. Then in the attack time, if direct access rate of a user to pages (not through hyperlinks) passes a threshold, the user gets a negative score.

Repetitive pages

In a random flood attack, an attacker may request several times the same page. When the number of requests for a same page through a same user passes a threshold, the user is more probable to be a malicious user.

Out-of-time pages

Out-of-time pages refer to the old pages that are rarely requested. For instance, pages that have been uploaded 24 hours ago may have no requests or few requests. A server can estimate a threshold time for the out-of-time attribute such that if during the attack a user sends request for the pages that have been uploaded before that time, it gets a negative score.

[2] An ISP normally changes the IP address of users every two or three days. As a result, a server can measure downtime of users who connect to the server again in less than two or three days.

Page classification based on type

In several servers, pages can be virtually or logically be classified based on their types. For instance, a news web agency can classify its pages based on the type of the news: politics, economics, culture, sport, etc. Then the server can measure the request rate of each user for each category. The collected data of different users of different times assists the server to depict the CDF for each category. Similarly, the CDF helps the server to consider a threshold rate for each category such that during the attack if a user has a request rate larger than the threshold rate of a category, he gets a negative score.

Page access rate and page popularity

Surely, every page has a different access rate. Some pages are frequently requested by the users and some have very few requests. The result of web mining [14] shows that in most of cases, about 10% of the pages of a website draw 90% of the access. This attribute can significantly help the server to detect malicious connections as attackers are not aware of the popularity of pages and thereby access the pages randomly. The server can measure the access rate for any page during a time interval. Then the server can measure the popularity for each page as the following formula:

$$p_i^t = \frac{a_i^t}{\sum_{j=1}^{N} a_j^t} \tag{1}$$

where p_i^t shows the popularity of page i during time interval t; N is the number of pages and a_i^t is the access rate for page i during the time interval t.

The key point is that the server classifies pages based on their popularity into five major categories: very low, low, medium, high and very high popular pages. Then, it classifies any of the above major categories into some smaller classes such that pages which have similar popularity locate in one class. Then, for each class, the server depicts the CDF of the percentage of requests of users for pages of each class. Next, the server determines a threshold rate for each class based on the CDF. For instance, the server can consider a threshold rate for a CDF of 95% in a class; while it can consider the threshold rate for a CDF of 90% in another class. During the attack time, if the percentage of requests of a user exceeds the threshold rate for a class, he gets a negative score as explained below.

Hyperlink fraction click

Suppose a page has k hyperlinks. Which fraction of hyperlinks of a page is clicked by a user? During normal condition, a server can extract for each page a fraction of hyperlinks on which a user clicks with a given probability. Then the server can define a threshold for each page based on its observation from normal users such that the higher deviation from the threshold, the higher the probability is to be a malicious user.

Hyperlink depth

Let us explain this attribute with this question: how many consecutive interlinked pages a user requests? The depth (D) a user proceeds in hyperlink pages is an effective attribute that a server can extract for its normal users.

Similarly, a server can compute the CDF, average and standard deviation for this attribute.

Source IP address distribution

In most cases, source IP distribution of legitimate users is different from source IP distribution of attackers. While, distribution of source IP addresses of legitimate users is more uniformly scattered across the Internet; the distribution of source IP addresses of attackers is more cumulated in some places. This is because an adversary can catch several zombie machines in the same LAN or same area. For instance, in a university or in a company, most users rely on central firewall that has been installed on the gateway (they, themselves, do not care about installing firewalls or regularly updating the tool); hence, if an adversary could break the firewall's rules, he can capture several zombie machines from the same LAN. Figure 1 shows an example of this attribute where the left image shows source IP address distribution of a server just prior an DDoS attack (the right image). In fact, in some attacks, we can see the creation of several clusters of source IP addresses; while before the attack there were no such clusters. The IP addresses within the range of clusters are more suspicious to be the IP address of zombie machines.

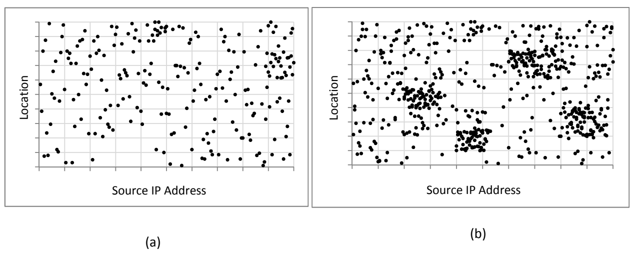


Fig. 1 Source IP address distribution: a) just before attack, b) after attack

Vice versa, in some particular cases, most users of a web-site are clustered in few particular places in different times a day. For instance, most users of a news web-agency that broadcasts news in the Persian language can be clustered in a few places especially in the countries which speak Persian language. During normal conditions (different days, weeks and months), the administrator of such web-sites can extract the map which shows cluster places of its normal users for different times a day. As it is more probable that the zombie machines are clustered in the places rather than the cluster places of normal users, the administrator can distinguish clusters of source IP addresses of its normal users from clusters of source IP addresses of zombie machines.

However, we should have in mind that the source IP distribution cannot always help, as the source IP distribution of zombie machines in some attacks may be uniformly distributed across the Internet or in the latter case zombie machines be in the same clusters of the normal users.

Arrival distribution rate of users

In any server, the arrival rate of users is different during different times a day. A server can measure arrival rate of users for different times a day, during different days a week, month, etc. Hence, a server can predict the arrival rate for different times for future days (except flash crowds that we discuss in Section 3.6). Normally, arrival rate of users follows the Poisson distribution. The Poisson distribution theory indicates that if the expected arrival rate in an interval is λ, the probability that k users connect the server during that interval is:

$$f(k;\lambda) = \frac{\lambda^k e^{-\lambda}}{k!} \,. \tag{2}$$

For example if any second, two users connect the system ($\lambda = 2$), then the probability that 20 users connect to the server during a second is 5.8×10^{-14}. In a DDoS attack, a large number of attackers simultaneously or in a short time connect the server. According to the above probability, we can predict that most of the users who connect to the system during that particular time are attacker's machines. The interesting point is that the server can predict percentage of false positive (falsely detect a legitimate user as a attacker's machine) for that particular time as it knows the normal arrival rate for each time.

Well-known confirmed users

In several sites, some users often and regularly visit the sites, for instance, every day, every few days or even several times a day. A server can precisely detect its well-known users. These users always appear in the same place with the same IP address. Even, a server can extract the interests of such users (what kind of files, pages they always refer). However, this is only possible for those users that use static IP addresses. The users which their system use dynamic IP address cannot be well-known for a server, as the corresponding ISPs regularly (every few days) change their IP addresses[3]. During the attack time, a server can consider positive credits for such users.

A server must precisely identify its well-known users; otherwise an adversary can cheat the server with such a rule.

3.2 Analysis of attributes

Let us discuss the abilities of attributes and show whether attackers can bypass attributes or not. We are aware that an attacker can easily set the attack tool such that it does not request for repetitive and out-of-time pages (though the attacker really does not know what the threshold time is for the out-of-time pages). The attack tool also can be set in such a way that it always follows

[3] We note that in most of ISPs a user can have a static IP address by paying some additional money.

the rule of sequential-hyperlink pages and does not request for pages directly. In fact, an attacker can easily thwart these three attributes. However, we still consider them for calculating the scores because if an attacker forgets to set his attack tool correctly, then these attributes make sense.

An attacker also can thwart some other attributes, but he encounters some difficulties and limitations. For instance, an attacker can send request rate in the order of legitimate users or he can have a download rate in the order of legitimate users as well. But, the problem is that in such cases, the attacker should have numerous zombie machines to be able to set up an effective attack. We call such attack a meek attack and below, we discuss it in detail.

The attribute of "source IP address distribution" is somewhat in contradiction with attributes of "request rate" and "download rate". If an attacker tries to avoid creation of clusters of source IP addresses, then he should use smaller number of zombie machines. In this case, the attacker can choose a set of zombie machines from the pool of zombie machines that he has in such a way that IP address of zombie machines are evenly distributed across the globe. But, the point is that in this case, the attacker should use a high rate for request rate and download rate to have an effective attack. On the other hand, if the attacker wishes that all zombie machine have send/receive rates in the order of legitimate users, then he should use numerous zombie machines for an effective attack (meek attack). In this case, creation of clusters of source IP addresses is inevitable!

Tackling the attribute of "arrival rate distribution" is not easy for the attacker. For instance, suppose in a meek attack, the attacker uses 20 000 zombie machines to bring down a server. If he organizes the attack in such a way that every second, 10 zombie machines establish connections, then about 34 minutes are required for completing the attack scenario. Moreover, the attribute of "uptime" is in contradiction with this scheme because those zombie machines which have established connection earlier encounter the limitation of "uptime" and get negative scores of uptime. In fact, the attacker is forced to activate a large fraction of zombie machines in a short time and consequently, they get a negative score of this attribute.

To bypass attributes of "hyperlink fraction click" and "hyperlink depth", an attacker encounters a serious challenge because these two attributes are in contradiction. First, we note that any page has a particular threshold rate for "hyperlink fraction click". For instance, while this threshold for a page may be 40%, for another page, this threshold may be 3%. Second, an attacker is not aware of these threshold rates. However, an attacker may try to click small fraction of hyperlinks of each page to guarantee that he remains below the threshold rate for each page. In this case, first, an attacker does not know which particular rate to select and does not know if the selected rate is much larger or smaller than the threshold rate of different pages. The second issue which is much important is that if an attacker chooses a small fraction rate for clicks on each page, then he should proceed in depth. We note, in this case, the attacker get negative score of attributes of "hyperlink depth". In fact, the attacker should find a semantic relation between these two attributes which it seems unlikely because the attacker does not have access to statistics of these two attributes.

We believe an attacker cannot bypass the attributes of "uptime", "downtime","page classification based on type" and "page popularity" because these attributes are completely site-dependent and thus more difficult for an outsider attacker to collect such information. Below, in Section 3.7 we discuss the reaction of the attacker against page popularity in depth.

3.3 Computing scores

In this section we formulate the notion of score for each attribute and subsequently for each connection. An established connection c has a set of attributes A_i^c, where A_1^c could be the request rate, A_2^c, the download rate, etc. Let $S(A_i^c)$ be the score of connection c associated with attribute A_i. We then calculate the total score for connection c as the sum of scores of all attributes:

$$S(c) = \sum_{i=1}^{n} S(A_i^c)\,, \tag{3}$$

where n is the number of attributes. Now, let us explain how the score of a connection is calculated for an attribute such as A_i. The score for attributes of "request rate", "download rate", "uptime", "page popularity", "out-of-time pages", "repetitive pages", "sequential-interlink pages", "page classification based on type", "hyperlink fraction click", "hyperlink depth" and "arrival distribution rate" is calculated as follows. Suppose $y = f_i(x)$, where f_i shows cumulative distribution function (CDF) for the attribute A_i in the reference profile. For instance, when the value of attribute is x_1, the CDF for this value is $y_1 = f_i(x_1)$. Assume that the control unit has determined a pair of (x_b, y_b) as a reference baseline; so, $y_b = f_i(x_b)$. In the next section, we explain how the control unit through a feedback-control process selects the appropriate baseline point. Assume that the value of the connection c for the attribute A_i is shown by x_i^c. If $x_i^c \le x_b$, then the score associated with this attribute for the connection c would be zero; otherwise, the score is calculated as the following formula:

$$S(A_i^c) = -1 \times k^{div(\frac{x_i^c - x_b}{\Delta x})} \times \frac{x_i^c - x_b}{\Delta x}\,, \tag{4}$$

where k is a geometric constant value (e.g., 1.2), $div(m/n)$ shows quotient m per n and Δx is a constant scale factor. As can be seen, the higher the deviation from the base-line value (i.e., $x_i^c - x_b$), the lower the score which is decreased in a semi-geometric progression. For instance, assume that the baseline point is $(0.3, 0.7)$ for the attribute of request rate; $0.7 = f_i(0.3)$ which it means that 70% of users have equal or less than one request per each three seconds. Now, suppose that the value of connections c_1 and c_2 for the attribute of request rate are 0.4 and 0.75, respectively (i.e., $x_i^{c1} = 0.4$ and $x_i^{c2} = 0.75$).

Assume that k and Δx are 1.2 and 0.1, respectively. In this case, the score of request rate for connections c_1 and c_2 would be -2 and -9.33, respectively. Any server can select k and Δx appropriately based on the volume of attack rate.

The score for the attribute of "downtime" is calculated as for the above attributes, but in the reverse direction; i.e., 0 if $x_i^c \geq x_b$ and $-1 \times k^{div(\frac{x_i^c - x_b}{\Delta x})} \times \frac{x_i^c - x_b}{\Delta x}$ if $x_i^c < x_b$.

The score for attributes of "source IP address distribution" is a constant value: if a connection has been established from locations where we guess are the location of attackers, the connection gets a constant score, for example -1[4]; otherwise the connection gets zero.

3.4 Control unit

The goal of the control unit is to prevent exhaustion of bottleneck resources and, as a result, prevent that the server goes down during the attack. To achieve this goal, the control unit defines two thresholds for each bottleneck resource and defines three strategic states: red, yellow and green. For each bottleneck resources, we define threshold1 and threshold2 as 90% and 60% of the total capacity of the bottleneck resource, respectively[5]. When the load of a bottleneck resource exceeds **threshold1**, we say so the situation of the bottleneck resource is in the red state. Whenever, traffic is controlled and the load of the bottleneck resource returns below **threshold1**, but still is above **threshold2**, we say so the situation of the bottleneck resource is in the yellow state and finally, when the load of the bottleneck resource returns below **threshold2**, we say so the situation of the bottleneck resource is in the green (normal) state.

The control unit periodically checks status of the bottleneck resources of the server (e.g., bandwidth, TCP/IP stack, CPU cycles, memory, etc.). Whenever the load of one of the bottleneck resources of the server exceeds threshold1, the server goes to the freeze mode and the control procedure is started. As long as the server is in the freeze mode, it does not accept new connections. The next task of the red state is that the control unit assigns scores to the established connections and then drops suspicious connections until the load of bottleneck resource(s) has(ve) returned below threshold1. When the state of the system exits the red state, the server exits the freeze mode and accepts new connections. After exiting from the red state, if the server locates in the yellow state, although the server exits the freeze mode, the control unit still calculates score for the established connections. In this state, if a request for a new connection (i.e., SYN packet) arrives, the control unit first checks whether

[4] The value of constant score can be discussed and it is possible that a server could extract a suitable value by its experience. However, we choose -1 for constant score in this paper.

[5] These thresholds can be varied from a server to another server. They can be chosen precisely for any server based on the experience that the server gets during several days (the rate of thresholds for this work have been selected based on our case study).

the source IP address of the SYN packet is in the blacklist or not (see below). If it is in the blacklist, then its previous score is summed up with the score of the "downtime" attribute. If it is not in the blacklist, only score of "downtime" attribute is calculated for it. Then the score of the new connection is compared with the score of established connections; if the lowest score of the system is lower than the score of the new connection, the connection with the lowest score is dropped and the new connection is appropriately established.

The control unit always follows the following rules:

Rule 1: the connections which get zero score are not dropped. In other words, only connections with negative scores are candidates for being dropped.

Rule 2: if the score of a connection is equal or greater than a threshold (e.g., -10), the control unit should send a CAPTCHA test for the user. If the user solves the test the connection is not dropped; otherwise, the connection is dropped. Let us call this threshold the "drop threshold".

Rule 3: if the score of a connection is smaller than the drop threshold (e.g., -10), the connection is candidate for dropping without testing CAPTCHA puzzle with it.

The control procedure is as follows.

1. The control unit initializes the baseline point for each attribute to a specific value. The decision about the initial value of baseline points can be made in the pre-attack stage. Normally, initial values are selected such that minimum amount of false positive (FP) occurs; thereby they will be initialized to the maximum amount such that FP at the beginning is zero. The points where FP is close to zero are the points where CDF is close to one because when CDF is one it means that 100% of normal users are covered.
2. The control unit monitors all established connections for duration of so called "score interval" (e.g., one minute) and then computes the score for each established connection based on the baseline point for that interval.
3. The control unit starts dropping connections from the connection which have the lowest score. It continues dropping connections until either no connections with negative scores remain (considering the rules) or the load of bottleneck resource(s) returns below threshold1.
4. If no bottleneck resource is in the red state, the server exits the freeze mode. If the state of at least one bottleneck resource is in the yellow state, the control unit still calculates scores for the connections and waits for new connections.
5. If the state of all bottleneck resources returns to the green state, the control procedure is terminated.
6. If at least one of the bottleneck resources is in the red state and there are no connections with negative scores smaller than drop threshold (e.g., -10), the control unit changes the baseline point appropriately. For all attributes except attributes of "downtime" and "source IP address distribution", the baseline point is changed as follows: suppose in the CDF function of an attribute, (x_b, y_b) and (x'_b, y'_b) shows the current and next baseline points, respectively. For the next base-line point, the control unit decreases y_b by

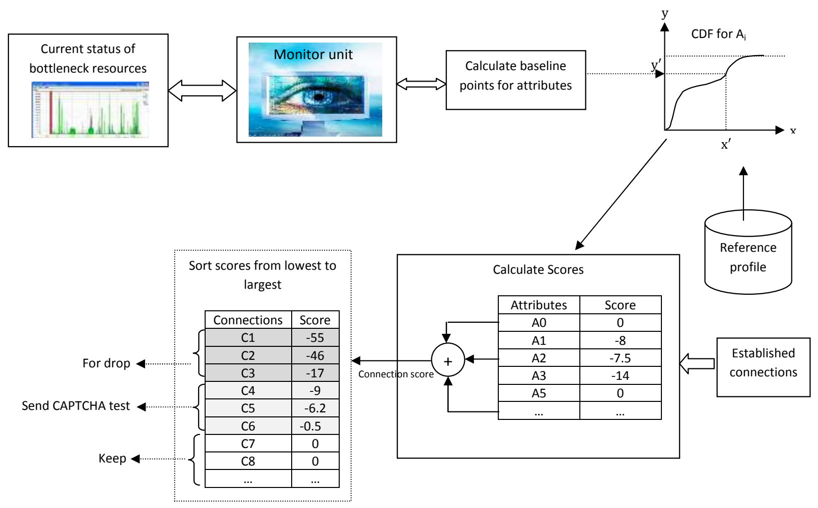
a Δy. So, we have

$$y'_b = y_b - \Delta y \Rightarrow x'_b = f^{-1}(y_b - \Delta y) \tag{5}$$

The amended baseline point, i.e. the next base-line point would be $(f^{-1}(y_b - \Delta y), y_b - \Delta y)$. The amount[6] of Δy is considered appropriately, for instance, 0.1 or 0.05. For the attribute of "downtime" the calculation is similar, but the direction is opposite. In other words, for this attribute, we have $(x'_b, y'_b) = (f^{-1}(y_b + \Delta y), y_b + \Delta y)$. For the attribute of "source IP address distribution", there is no baseline point and the scores are calculated as before.

7. The algorithm returns to step 2. This loop is continued until the condition of step 5 is succeeded.

Figure 2 illustrates the control unit. Below, we address some additional issues of the control unit.



Attributes	Score
A0	0
A1	-8
A2	-7.5
A3	-14
A5	0
...	...

Connections	Score
C1	-55
C2	-46
C3	-17
C4	-9
C5	-6.2
C6	-0.5
C7	0
C8	0
...	...

Fig. 2 The control unit

– It is not required that at each iteration of the loop, baseline points for all attributes are changed. Sometimes, the control unit may only change baseline points for some attributes and not for all. For instance, in an iteration of the loop, the control unit may not change the baseline point for attributes of "uptime" and "downtime".

[6] The amount of Δy (0.1, 0.05, 0.02, etc.) is chosen based on the response time within which the system shall be stabilized (the attack is controlled and the load of bottleneck resource returns below threshold 2). If a big value is chosen for Δy (e.g., 0.15, or 0.1), the system is stabilized faster, but the percentage of false positives is increased. In contrast, if a small value is selected for Δy (e.g., 0.02, or 0.05), the system is stabilized slower, but the percentage of false positive would be low.

– When a connection is dropped, the IP address of the connection within its score is recorded in a list which is called blacklist.
– When some connections have the same scores, we can randomly select which one is dropped first. However, we can also consider some preferences, for instance, the score of request rate and download rate is more significant than score of other attributes for dropping. Other preferences can be discussed.
– A server can select a suitable "drop threshold" before the attack time based on the maximum absolute value of negative scores that legitimate connections may get.
– We note that according to rule 2, if a user has negative score larger than drop threshold (-10) and could solve the CAPTCHA test is not dropped. The question is whether we should worry about the attackers who have score larger than drop threshold and could break the CAPTCHA tests (solve these tests)? The answer is we are not worried about such cases because these attackers get negative score lower than the drop threshold in the next "score intervals" due to their behavior and thereby without CAPTCHA tests are dropped.

3.5 Management of the bottleneck resource(s)

When the state of a bottleneck resource reaches threshold1, the victim server goes to the freeze mode. In this case, we expect all established connections get services without performance degradation. But, as the state of the bottleneck resource(s) is in the threshold level, if established malicious connections request further services, the bottleneck resource(s) is exhausted. Unfortunately, exhaustion of the bottleneck resource(s) during the attack increases uptime of legitimate users abnormally. In fact, due to the limitation of the bottleneck resource(s), legitimate users should stay more on-line to get services. On the other hand, exhaustion of the bottleneck resource(s) may cause disconnecting of several legitimate users. This causes some legitimate users have an abnormal downtime attribute as well. Abnormal uptimes and downtimes lead to abnormal increase in the percentage of false positives.

In order to handle the above problem, we should manage the bottleneck resources carefully such that the established legitimate connections could get services without performance degradation. Hence, we avoid abnormal uptimes and downtimes. The max-min fairness method is a simple and effective method to manage a bottleneck resource during the attack time. The max-min fairness is the problem of dividing a scarce resource among a set of users, each of whom has an equal right to the resource, but some of whom intrinsically demand fewer resources than others. How, then, should we divide the resource? Intuitively, a fair share allocates a user with a "small" demand what it wants, and evenly distributes unused resources to the "big" users. Consider a set of users 1, ..., n that have resource demands x_1, x_2, ..., x_n. Without loss of generality, order the users demands so that $x_1 \leq x_2 \leq ... \leq x_n$. Let the capacity of the bottleneck resource is C. Then, we initially give C/n of the resource to the user with the smallest demand, x_1. This may be more than what user 1 wants, perhaps, so we can continue the process. The process ends when each user gets no more than what it asks for, and, if its demand was not satisfied, no less than what any other user with a higher index got. Such resource allocation is called a max-min fair allocation, because it maximizes the minimum share of a user whose demand is not fully satisfied. Combination of the freeze mode and the max-min fairness algorithm guarantee that established legitimate connections get services without performance degradation and consequently, abnormal uptimes and downtimes are not concern issues.

3.6 Flash crowd versus application-layer DDoS attacks

Flash crowd is the sudden increase of workload in a server when many legitimate users simultaneously (or in a short time) connect to the server. Flash crowds are quite similar with application-layer DDoS attacks in terms of network anomaly and traffic phenomenon. In spite of these similarities, there are several significant differences between them such that an application-layer DDoS attack can easily be recognized from a flash crowd. The differences are as follows: (1) Users of flash crowd have same attributes of normal users; while attributes of attackers significantly differ from attributes of normal users. (2) A flash crowd happens when a server provides a new and attractive event (e.g., a breaking news story) for users. For instance, on September 11, 2001, a flash crowd happened for the CNN website. As another example, the vote for the host of the 2008 Olympic Games caused a flash crowd. In fact, a server can predict the happening of a flash crowd, while a DDoS attack may happen anytime unpredictably. (3) The duration of flash crowds is short. In fact, users of flash crowds have shorter uptime than normal users during normal times. The reason is that users of flash crowds leave the server upon acquiring their interest object from the server. For instance, when a user downloaded the page related to the breaking news story and read it, he leaves the server immediately. However, in DDoS attacks, attackers stay on-line for long time and uninterruptedly consume resources of the server. (4) Users of flash crowds normally have the same request, for instance request to download the page related to the breaking news. While in a DDoS attack, attackers have different and various requests. Table 1 summarizes the differences between flash crowds and application-layer DDoS attacks.

Several techniques have been invented for handling flash crowds [7,30,24]. However, as the attributes of users in a flash crowed completely differ from the attributes of users of an application-layer DDoS attack, those techniques cannot handle the problem of application-layer DDoS attacks.

Table 1 Summarizing differences between flash crowds and application-layer DDoS attacks

Index	Item	Flash crowds	Application-layer DDoS attacks
1	Attributes of legitimate users	Yes	No
2	Uptime	Short	Long
3	Target requests	Identical	Random
4	Time of occurrence	When a new and attractive event happens	Anytime
5	Scattering	Typically, uniformly distributed across the world	Typically, cumulated in some locations

3.7 Page popularity and the reaction of attackers

For measuring popularity of pages[7], a server selects random users at random times and then for a specific duration (e.g., two minutes) measures page popularity. The selected users must have the following conditions.

- The selected user should have a normal and common uptime.
- According to the attribute of "downtime", a reappeared user is not selected and if such user had already been selected for measuring page popularity, the associate measuring regarding that user is undone.
- The selected user should not have requested repetitive pages.

For classification, we act as follow. The pages that are never appeared for measurement or their popularity is below a specific threshold (e.g., Thd_1) are considered for class 1 (very low popular pages). The pages that their popularity is between Thd_1 and Thd_2 are considered for class 2. Similarly, the pages that their popularity is between Thd_{i-1} and Thd_i are considered for class i. The popularity of the pages of class i is greater than the popularity of the pages of class j when $i > j$. The values of Thd_1, Thd_2, Thd_3, etc., are determined based on the server's experience from several days. These threshold rates are fixed and do not change.

Now, the question is whether an attacker can infect page popularity and falsely increase popularity for certain pages. In this case, the attacker may escape from getting negative score for the attribute of "page popularity". For answer to this question, let us first explain two types of http flood attack.

A common attack: this attack is the most common attack that attackers use. In this attack, a zombie machine sends requests to the server at a high rate. The request rate of zombie machines is higher than the request rate of normal users.

[7] Some websites show the most popular and most recently read pages to the public. Such websites cannot use the ConnectionScore technique for handling application-layer DDoS attacks because one of the most important attribute is revealed for the attackers. We hope the websites that want to use the ConnectionScore technique do not show such information to the public.

A meek attack: this attack is a more sophisticated attack where zombie machines send requests to the server at the rate of normal users and also downloads from the server at the rate of normal users.

For a common attack, an attacker needs only a small number of zombie machines; while for a meek attack, the attacker needs numerous zombie machines. As we show in Section 5, a common attack can be treated using attributes such as "request rate" and "download rate"; thereby there is no need for attribute "page popularity" for handling this attack. But, the attribute of "page popularity" is an effective remedy for a meek attack. Now, let us explain whether in a meek attack, an attacker can infect the page popularity before the attack time.

When the attack is started, the attribute of "downtime" assigns negative scores to the connections which appeared before the attack. Consequently, if an attacker, before the attack time, uses a set of zombie machines to change the page popularity, he cannot use them for the attack; so, the attacker misses a fraction of its zombie machines for the attack. Another point is that the server selects random users at random times for measuring page popularity, and moreover, the selected users must have the above mentioned conditions. These will increase difficulties for an attacker to change page popularity. To change page popularity in a somewhat range, an attacker must continually activate new zombie machines several hours before the attack time. As, the attacker needs numerous zombie machines to handle a meek attack, it is unlikely that an attacker has enough zombie machines for both the meek attack and page popularity infection. However, suppose that an attacker has some extra zombie machines further than the zombie machines that he needs for the meek attack. For instance, suppose that an attacker has enough extra zombie machines such that continually activate new zombie machines to connect to the server in the order of 10% of the normal arrival rate from several hours prior the attack time. For instance, if 200 normal users connect to the server every minute, we assume that an attacker activates 20 zombie machines at every minute several hours before the attack to infect the page popularity.

An attacker does not know (1) which zombie machines are selected for sampling, (2) at which time the sampling is taken (for instance if a zombie machine has been selected, the attacker does not know the machine is in the first minute of its uptime, the second minute of its uptime, etc.) and (3) what the range of threshold is for a specific class. Hence, the attacker must select limited number of random pages and ask zombie machines of every round to send requests for them. As mentioned above, 90% of requests of normal users are for only 10% of pages. So, about 90% of pages of a web-site belong to low popular classes. Hence, it is very likely that the pages which the attacker selects for popularity changing are selected from this 90 percent.

In fact, the attacker may can falsely generate popularity for only limited number of random pages of low popular classes and force the server to classify them for higher popular classes. From the experience that a server has from several days, it knows the estimated relative population of each class. When an attacker infects the page popularity, the server can observe the population of some classes has been changed. For instance, it observes that the population of a low popular class decreases, while the population of a high popular class increases because the attacker could falsely increase popularity for some pages of a low popular class and transfers those pages to a higher popular class. As discussed above, the threshold rates of classes are fixed. In this case, transferring some pages from a lower popular class to a higher popular class may increase the percentage of false positive. For instance, suppose, before the infection, the population of class i and j ($i > j$) is 150 and 1000 pages, respectively, and the threshold rate for these two classes is 3% and 20%, respectively. Due to the infection of an attacker, 80 pages of class j move to class i (see Figure 3). The probability of false positive is increased because the pages of class j for which normal users may send requests, now have been moved to class i and as the threshold rate of class i is the same as before, the percentage of false positive increases. To handle this challenge, we increase the threshold rate for class i proportional to the number of pages that has been moved to it and the population of the source class. For example in our example, we change the threshold rate for class i from 3% to 11% (0.03 + 80/1000).

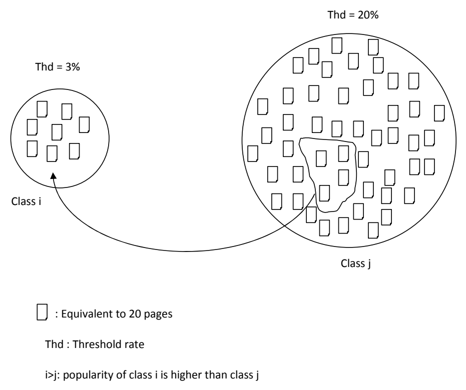

Thd = 20%
Thd = 3%
Class i
Class j
▯ : Equivalent to 20 pages
Thd : Threshold rate
i>j: popularity of class i is higher than class j

Fig. 3 The attacker can transfer some pages from a low popularity class to a higher popularity class by infecting the page popularity.

Now, the question is if the attacker sets zombie machines to only send requests for these specific pages (in our example, these 80 pages), then whether zombie machines can escape from negative score of the attribute of "page popularity". The answer is "no"; they cannot escape because the attacker does not know the threshold rate of classes and he does not know in which class or classes these specific pages have been located. So, if a zombie machine sends 100% of its requests for these specific pages, surely it exceeds the threshold rate of a class and gets negative score. However, if zombie machines only send requests to a specific set of pages, we can handle the problem with the following further solutions.

Solution 1: It is important to know that by using the attribute of "arrival rate", a victim server can detect that an attacker is infecting the page popularity. Secondly, as 90% of pages are low popular, the victim server knows that the attacker selects 90% of pages for generating falsely high popular pages from these low popular pages. Now, we can use these falsely high popular pages as a decoy to detect attackers. For instance, suppose an attacker could generate falsely high popularity for 80 pages and zombie machines send most of their requests for these pages. Also assume that there are 10000 zombie machines during the attack time. When the server sees that 10000 machines send request for 80 specific pages, it can understand that these 80 pages are false negative high popular pages and the requesters for those pages are attackers.

Solution 2: when the server sees that a large number of users send requests for specific number of pages, the server can use the technique of "multicasting" for replying to these users instead of a unicast reply to each user. For instance, suppose 10000 users send requests for 80 pages during a short time, i.e., every page has request from 125 users. Suppose that the size of each page is 100 kB equivalent to 70 packets (size of each packet is 1500 bytes). With unicast replying, the server should send out 700000 packets to these 10000 users; while with multicasting replying, the server needs to send out 8000 packets (note that in the multicasting packets 500 bytes is used for inserting IP address of 125 users). In fact, the bandwidth required for multicasting technique is 87.5 times less than the bandwidth required for unicasting technique.

Solution 3:, we can handle the attack through the techniques which have been proposed for flash crowd. In fact, when request rate for particular pages is increased, the problem can be viewed as a flash crowd problem because in a flash crowd, most users send requests for a specific set of pages. The idea is that during the attack time, the server copies a version of the pages which have high request onto spare servers. Then, when it receives a request for these pages, it redirects the request for the corresponding spare server. The corresponding spare server sends the page for the user using the source IP address of the main server (i.e., it inserts the IP address of the main server in the source IP address of the packets). This idea is well-known as dynamic resource allocation which is used for handling flash crowds [7,30]. Multiple spare servers are free and idle servers that can be allocated dynamically to any application when the need arises [7,30]. We assume enough spare servers are available. It is worth noting that this is an evident assumption for several flash crowd countermeasures [7, 30]. A combination of this technique and "multicasting" technique can be considered also as a solution.

In this case, the attacker has two ways: a) he can still ask zombie machines to send requests for falsely high popular pages, b) he can ask zombie machines to send requests for other pages. In the former one, all or a large fraction of attackers' requests is redirected (and distributed) to multiple spare servers, or is answered by multicasting or is handled by combination of multicasting and dynamic resource allocation. Consequently, the load on the victim server falls below the threshold and the server exits the freeze mode and accepts new connections. In fact, DoS attack is controlled. Furthermore, if even zombie machines could escape from getting negative score of attribute of "page popularity", when the timeout of the attribute of "uptime" arrived, they get negative score of this attribute and are dropped. In the latter case, malicious connections get negative score of the attribute of "page popularity" with high probability as they send requests for random pages. We note that malicious connections may get negative score of other attributes such as "hyper fraction click", "hyper depth", "source IP distribution", etc. Consequently, we expect after elapsing few "score intervals", most malicious connections are dropped.

In summary, we indicate that (1) in a meek attack, the attacker has negligible chance to infect page popularity, as the attacker needs numerous number of zombie machines for the attack; (2) even if the attacker has some extra zombie machines further than the number of zombie machines that he needs for the meek attack, he can change the popularity of only few pages as the server utilizes some rules for measuring page popularity; (3) even the attacker changes popularity for some pages, he cannot escape from getting negative score of the attribute of "page popularity" as he does not know the threshold rate of classes; (4) if zombie machines send requests for only specific set of pages, the problem can be handled using "multicasting technique", "dynamic resource allocation techniques" or a combination of both.

4 Analyzing attributes for a case study

This section studies and analyzes the nature of the distribution of the mentioned attributes for a real case-study in the Internet: real-life traces collected from the traffic archive of ClarkNet WWW server. The traces contain two week's worth of all HTTP requests to this web server. Traces are available on-line[8].

Request rate

Figure 4.a shows the cumulative distribution function for request rate where Y-axis shows CDF and X-axis shows request rate per second. As can be seen more than 50% of users have request rate per second less than 0.066. It means that more than 50% of users have sent a request to the server every 15 seconds. Moreover, more than 80% of users have request rate per second less than 0.25. As the figure shows about 10% of users have sent more than one request every two seconds. Our analysis also shows that the average, the standard deviation, the minimum rate and the maximum rate for this attribute are 0.18, 0.37, 0.0026 and 0.8, respectively. For this case study, we suggest $(x_b, y_b) = (0.5, 0.9)$ as an initial base-line point.

Download rate

Figure 4.b shows CDF for download rate. As the figure shows, more than 50% of normal users have a download rate of less than 1000 bytes per second. More

[8] http://ita.ee.lbl.gov/html/contrib/ClarkNet-HTTP.html

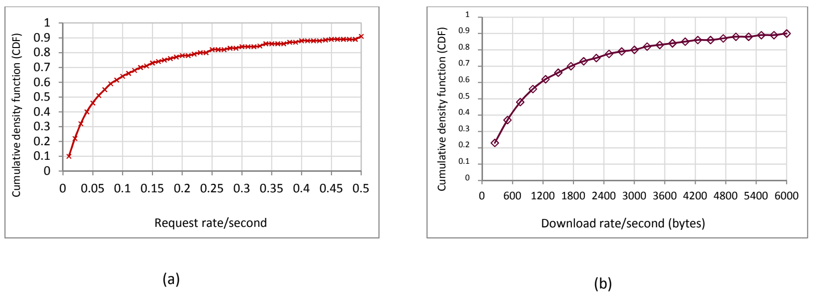


Fig. 4 Cumulative distribution function: a) request rate/second, b) download rate/second

than 80% of users have a download rate less than 3000 bytes per second. Moreover, only 10% of users have download rate more than 6000 bytes per second. The average, the standard deviation, the minimum rate and the maximum rate for this attribute are 2283, 3870, 63 and 15611 bytes per second, respectively. We suggest $(x_b, y_b) = (6000, 0.9)$ as an initial base-line point of this attribute for this case-study.

Uptime

Figure 5.a depicts CDF for uptime of normal users where Y-axis shows CDF and X-axis shows uptime in the range of minutes. As can be seen about 54% of users have stayed online for less than 1.5 minutes. About 80% of users have an uptime less than 5 minutes and only 4% of users have an uptime more than 16 minutes. Our measurements show that normal users of this WWW server have the following average, standard deviation, minimum rate and maximum rate for uptime: 3.45, 6, 0.14 and 80 minutes, respectively. The point of $(10, 0.9)$ can be considered as an initial base-line point for this attribute in this case-study.

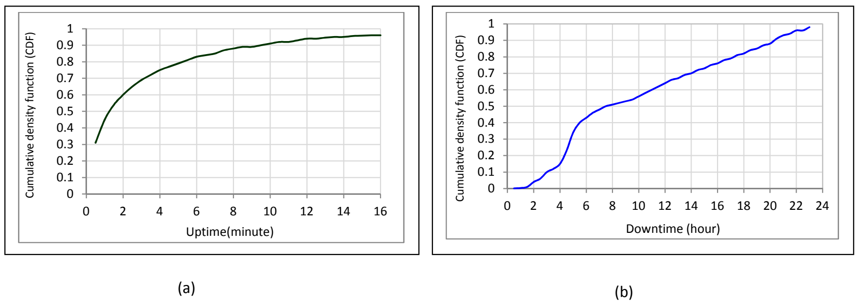


Fig. 5 Cumulative distribution function: a) Uptime, b) Downtime

Downtime

Figure 5.b illustrates CDF for downtime of those normal users that more than once in two consecutive days have established a connection with the server. As can be seen more than 50% of those normal users have a downtime more than 8 hours and more than 80% have a downtime more than 4 hours. Our analysis shows that only 0.3% of users have downtime less than one hour. The average, the standard deviation, the minimum value and the maximum value for this attribute are 12.34, 8.43, 0.25 and 23 hours, respectively. The point of $(3, 0.1)$ can be considered as an initial base-line point for this attribute.

Browsing behavior

Page popularity

Figure 6 shows distribution of page popularity for day 3 between 11:00 to 14:00 o'clock. As discussed above, we divide pages based on their popularity into five major categories: very low, low, medium, high and very high popular pages. Next, for more accuracy, any of the above major categories may divided into some smaller classes. Figure 6 shows that pages of class 1 compose very low popular class; pages of classes 2 and 3 compose low popular class; pages of classes 4 to 12 compose medium popular class and finally pages of classes 13 to 17 and 18 to 19 compose high and very high popular classes, respectively. In Figure 6, popularity of class i is more than class j when $i > j$.

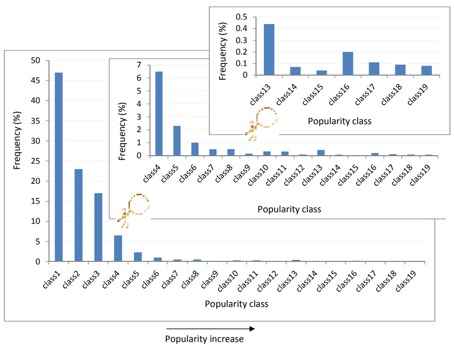


Fig. 6 Frequency rate of page popularity (different classes)

Figure 7.a shows the threshold rate (percentage) for different classes based on CDF of 90% to 95% (depending on the class) for four different days. In other words, Figure 7.a represents that for example the percentage of total requests of 95% of users for a specific class is below the determined threshold rate for that class. As can be seen, the threshold rate of a specific class is in a similar range for different days. So, we can determine an upper bound and fixed threshold rates for classes independent from days.

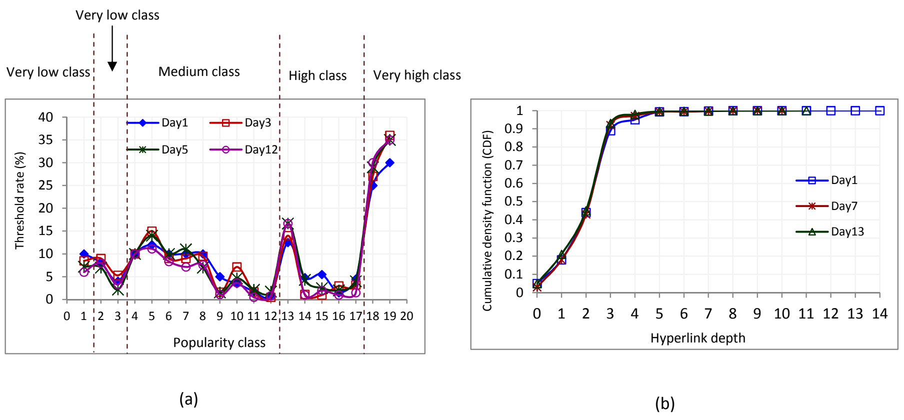


Fig. 7 a) Threshold rate for popularity classes, b) Cumulative density function for hyperlink depth

Hyperlink depth (D)

Figure 7.b shows CDF for the attribute “hyperlink depth”. As can be seen the hyperlink depth of only 10% of users is more than 3. Moreover, less than 1% of users proceed in the depth of the web-site more than 5. We can consider the point of (4, 0.95) as an initial base-line point.

Source IP address distribution

Figure 1.a shows source IP distribution for duration of 5 minutes of day 6. As can be seen, users are scattered uniformly across the Internet.

Arrival distribution rate of users

Figure 8.a and 8.b shows arrival distribution rate for day 1 and day 12, respectively. As can be seen arrival rate is different for different times a day. In both days, the arrival rate after mid-night between 01:00 to 07:00 is lower than other parts of day. The average arrival rate in this period of day is about 8 users per minute. The arrival rate between 11:00 to 16:00 is in the maximum state. The average arrival rate for this period of the day is about 27 users per minute. As can be seen in the maximum case less than 50 users have connected to the server during a minute; in other words, less than one user per second.

This attribute indicates that if at a second, several users (e.g., 200 users) suddenly connect to the server and server goes to warning state, most of those users belong to an adversary and false positive is about only one legitimate user. Such argument can be considered for duration of one minute and also false positive for that duration also can be estimated.

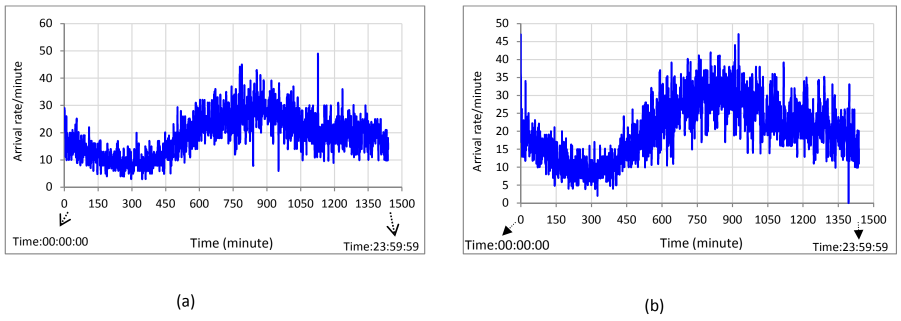


Fig. 8 Arrival distribution rate: a) day 1, b) day 12

5 Experimental results

To evaluate the effectiveness of the ConnectionScore scheme, we set up some experiments on the Emulab environment. In these experiments, we simulate day 6 of the Clarknet www server. We generate 3559 pages with different sizes according to distribution of file sizes of day 6 before 14:00 o'clock and then upload them in the server. A machine with 1.7 GHz, Pentium IV and 512 MB RAM plays the role of the Clarknet www server. In the experiments, legitimate clients follow all attributes of the real users of the Clarknet www server. We assume that the attack is started at time 14:03. We set the bandwidth of the server to 10^6 bytes per second.

Performance metrics: we are interest to see 1) score distribution of connections, 2) how fast the server can recover from the attack, 3) percentage of false positive and false negative, 4) percentage of candidate connections for false positive and 5) percentage of the legitimate connections that get negative scores (R_L) and also percentage of the malicious connections that do not get negative scores (R_A).

Parameters of the experiments: we set "drop threshold" to -10 and "score interval" to 1 minute. In these experiments, we assume that attackers are clever enough to avoid sending requests for repetitive pages and out-of-time pages. We also assume that attackers follow sequential-hyperlink pages correctly. In fact, we do not consider the above attributes in calculating the scores.

We evaluate the scheme against two types of attacks: 1) common attacks and 2) meek attack. For the common attack, we follow a real scenario to model this attack which has been represented in [27,15]. In this scenario, attackers use three viruses programs: Netsky.Q, Trojan_Sientok and BlueCode.Worm to send HTTP GET requests to the server. These virus programs send the requests in about 300 milliseconds, 250 milliseconds and 137 milliseconds, respectively. In this experiment, 150 machines play the role of attackers. For the meek attack, 600 attackers create TCP connection and follow attributes of request rates and download rates of legitimate clients. In other words, request rates and download rates of attackers are in the order of legitimate users.

Figure 9.a and 9.b show the score distribution for established connections when the server is under the common attack and the meek attack, respectively. Red bars show the score of malicious connections and green bars show the score of legitimate connections. As can be seen, about 25% of legitimate connections get negative score, but with small absolute values (maximum 11). In the common attack, the almost malicious connections get high negative scores (i.e., negative scores with high absolute values). The reason is that users' high request rate leads to high negative values in most of attributes. We observe that the largest and the lowest negative score in the common attack are -7485.65 and $-2.60E+06$, respectively. In the meek attack, all malicious connections do not get negative scores and moreover, the absolute value of negative scores is much lower than the common attack. We observe that the largest and the lowest negative score in the meek attack are 0 and -2585.3, respectively.

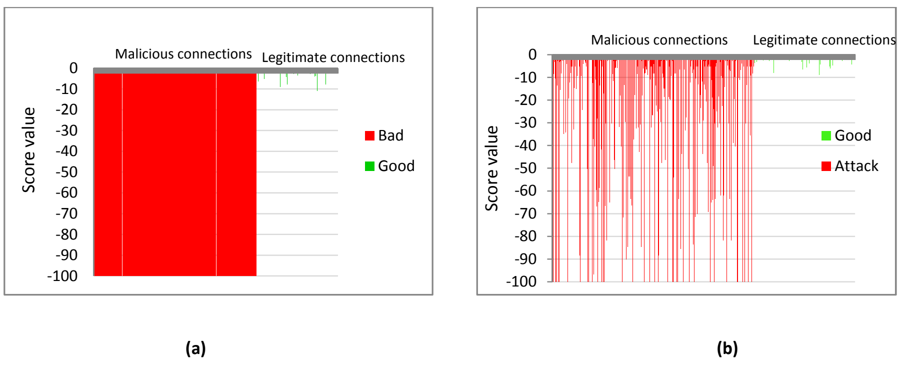


Fig. 9 Score distribution: a) common attack and b) meek attack

Figure 10 shows the bandwidth occupied by legitimate traffic and attack traffic during the meek attack (for the common attack we have the similar situation). At time 14:03, the attack is started and the server goes to the freeze mode. One minute (score interval) after starting the attack, the control unit drops enough number of connections with negative scores such that the bandwidth rate drops below threshold1. At this time (14:04) the server exits the freeze mode and accepts the new connections. The control unit replaces the established connections with the lowest score by new connections until the bandwidth drops below threshold2. Then the duty of the control unit

Table 2 Performance metrics (FP, FN, PFP, R_L and R_A) for both common and meek attack

	FP	FN	PFP	R_L	R_A
Common	0	0	2%	24%	0
Meek	0	6.3%	2%	24%	6.3%

terminates. During 14:03 and 14:04, the bandwidth rate of good traffic slowly decreases because some legitimate users normally leave the system (remember uptime).

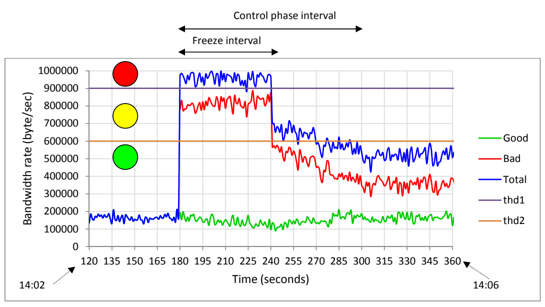


Fig. 10 Bandwidth rate occupied by good traffic and attack traffic

To calculate the percentage of false positive (FP), false negative (FN), the percentage of legitimate connections for possible false positive (PFP), R_L and R_A, we repeat the experiments 30 times during different times of a day. Table 2 summarizes the results. As can be seen, there is no false positive in both common and meek attack. On average 6.3% of malicious connections do not get negative score in the meek attack; while all malicious connections get negative score in the common attack. So, the percentage of false negative and R_A is 6.3%. On average 24% of legitimate connections get negative score. Finally, on average, 2% of legitimate connections get score lower than -10 (the drop threshold); thereby, 2% of legitimate connections could be candidate for possible drop without asking them to solve the CAPTCHA tests.

6 Conclusion

This paper proposes a new countermeasure against application-layer DDoS attacks that is called ConnectionScore technique. In this scheme, connections get score based on their behavior. With a high probability, the connections which get lower scores are malicious connections; thereby the server retakes bottleneck resources from them. The ConnectionScore technique is evaluated using Emulab testbed for two types of attacks: common and meek attacks. The results indicate that the technique can handle both common and meek attacks effectively. Although the results show that 24% of legitimate connections get negative scores, the percentage of false positive for both common and meek attacks is zero. Our results show that all malicious connections get negative scores with high absolute value in common attacks while in meek attacks, on average 6.3% of malicious connections do not get negative scores. We believe that the administrators of web-sites do not need to annoy users by forcing them to solve CAPTCHA tests. The ConnectionScore scheme can be effectively considered as an alternative technique to handle application-layer DDoS attacks. Other efficient attributes for calculating scores can be investigated in future work.

References

1. Adler, M.: Tradeoffs in Probabilistic Packet Marking for IP Traceback. Journal of the ACM **52**(2), 217–244 (2005)
2. Athanasopoulos, E., Antonatos, S.: Enhanced CAPTCHAs: Using Animation to Tell Humans and Computers Apart. In: Proceedings of Communications and Multimedia Security, pp. 97–108 (2006)
3. barford, P., Kline, J., Plonka, D., Ron, A.: A Signal Analysis of Network Traffic Anomalies. In: Proceedings of ACM SIGCOMM Internet Measurement Workshop, pp. 71–82. New York, USA (2002)
4. Beitollahi, H., Deconinck, G.: Analyzing Well-Known Countermeasures against Distributed Denial of Service Attacks. Elsevier Journal of Computer Communications **35**(11), 13121332 (2012)
5. Bernstein, D.J.: SYN Cookies. http://cr.yp.to/syncookies.html. Visited September 2011
6. Caum, L.O.: Why is CAPTCHA so annoying? http://lorenzocaum.com/blog/why-is-captcha-so-fing-annoying/ (2011)
7. Chandra, A., Gong, W., Shenoy, P.: Dynamic Resource Allocation for Shared Data Centers Using Online Measurements. In: Proceedings of Eleventh International Workshop on Quality of Service (2003)
8. Chen, R., Park, J.: A Divide-and-Conquer Strategy for Thwarting Distributed Denial-of-Service Attacks. IEEE Transactions on Parallel and Distributed Systems **18**(5), 577–588 (2007)
9. Chu-Hsing, L., Hung-Yan, L., Tang-Wei, W., Ying-Hsuan, C., Chien-Hua, H.: Preserving quality of service for normal users against DDoS attacks by using Double Check Priority Queues. Springer Journal of Ambient Intelligence and Humanized Computing (JAIHC) **4**(2), 275–282 (2013)
10. Fallah, M.: A Puzzle-Based Defense Strategy Against Flooding Attacks Using Game Theory. IEEE Transactions on Dependable and Secure Computing **7**(1), 5 – 19 (2010)
11. Fraser, J.: Why you should never use a CAPTCHA. online aspect, http://www.blogopreneur.com/2007/04/02/captchas-are-annoying/ (2010)
12. Gavrilis, D., Chatzis, I., Dermatas, E.: Flash Crowd Detection Using Decoy Hyperlinks. In: Proceedings of IEEE International Conference on Networking, Sensing and Control (2007)
13. The Honeynet Project, http://old.honeynet.org/papers/kye.html: Know Your Enemy: Tracking Botnets (2007)
14. Kantardzic, M.: Data Mining: Concepts, Models, Methods, and Algorithms, second edition edn. IEEE press (2002)
15. Lu, W., Yu, S.: An HTTP Flooding Detection Method Based on Browser Behavior. In: Proceedings of the International Conference on Computational Intelligence and Security, pp. 1151 – 1154 (2006)
16. Mirkovic, J., Reiher, P.: A Taxonomy of DDoS Attack and DDoS Defense Mechanisms. ACM SIGCOMM Computer Communications Review **34**(2), 39–54 (2004)
17. Morein, W.G., Stavrou, A., Cook, D.L., Keromytis, A.D., Misra, V., Rubensteiny, D.: Using Graphic Turing Tests To Counter Automated DDoS Attacks Against Web Servers.

In: Proceedings of the 10th ACM conference on Computer and communications security. Washington, DC, USA (2003)

18. Mori, G., Malik, J.: Recognizing Objects in Adversarial Clutter: Breaking a Visual CAPTCHA. In: Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition. Madison, Wisconsin (2003)
19. Oikonomou, G., Mirkovic, J.: Modeling Human Behaviour for Defense against Flash-Crowd Attacks. In: Proceedings of IEEE International Conference on Communications, pp. 1–6. Newark, USA (2009)
20. Peng, T., Leckie, C., K.Ramamohanarao: Survey of Network-Based Defense Mechanisms Countering the DoS and DDoS Problems. ACM Computing Surveys **39**(1) (2007)
21. Podevin, J.F.: Telling Humans and Computers Apart Automatically. Communications of the ACM **47**(2), 57–60 (2004)
22. Ranjan, S., Swaminathan, R., Uysal, M., Knightly, E.: DDoS-Resilient Scheduling to Counter Application Layer Attacks under Imperfect Detection. In: Proceedings of the IEEE Computer and Communications Societies, pp. 1–13. Barcelona, Spain (2006)
23. Savage, S., Wetherall, D., Karlin, A.R., Anderson, T.E.: Network Support for IP Traceback. IEEE/ACM Transactions on Networking **9**(3), 226–237 (2001)
24. Stavrou, A., Rubenstein, D., Sahu, S.: A Lightweight, Robust P2P System to Handle Flash Crowds. IEEE Journal on Selected Areas in Communications **22**(1), 6–17 (2004)
25. Thapngam, T., Yu, S., Zhou, W., Beliakov, G.: Discriminating DDoS Attack Traffic from Flash Crowd through Packet Arrival Patterns. In: Proceedings of 2011 IEEE Conference on Computer Communications Workshops, pp. 969–974 (2011)
26. Truong, H.D., Turner, C.F., Zou, C.C.: iCAPTCHA: The Next Generation of CAPTCHA Designed to Defend Against 3rd Party Human Attacks. In: Proceedings of IEEE International Conference on Communications. Kyoto, Japan (2011)
27. T.Yatagai, Isohara, T., Sasase, I.: Detection of HTTP-GET flood Attack Based on Analysis of Page Access Behavior. In: Proceedings of IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, pp. 232–235 (2007)
28. Xie, Y., Yu, S.: A Large-Scale Hidden Semi-Markov Model for Anomaly Detection on User Browsing Behaviors. IEEE/ACM Transactions on Networking **17**(1), 54–65 (2009)
29. Yau, D.Y., Lui, J.C.S., Liang, F., Yam, Y.: Defending against Distributed Denial-of-Service Attacks with Max-Min Fair Server-Centric Router Throttles. IEEE/ACM Transactions on Networking **13**(1), 29–42 (2005)
30. Zhou, W., Wang, D.: A Dynamic-Resource-Allocation based Flash Crowd Mitigation Algorithm for Video-on-Demand Network. In: Proceedings of the 3rd IEEE International Conference on Computer Science and Information Technology, pp. 388–392 (2010)