Defending HTTP Web Servers against DDoS Attacks through Admission Control and Attack Flow Detection

Seung Yeob Nam* , Nazarov Nodir*, and Taijin Lee†

*Department of Information and Communication Engineering, Yeungnam University, Gyeongsan, Republic of Korea

†Convergence R&D Team, Internet Convergence Division, KISA, Seoul, Republic of Korea

Yeungnam University
Technical Report YU-CNS-12-01
March 8, 2012

Abstract

We propose a two-stage Distributed Denial of Service (DDoS) defense mechanism, which can protect specific web servers from application-level DDoS attacks. In the first stage, we protect the server by limiting the number of machines accessing the server and serving existing flows with a higher priority than new flows when there are DDoS attacks on the server. Thus, it is important to manage the whitelist to discriminate accepted flows from newly arriving flows under DDoS attacks. In the second stage, as a post-processing step, we detect malicious flows, luckily included in the whitelist, by measuring the stress that each host offers to the server. The stress is measured by the time interval during which a given client makes the server busy, referred to as a client-induced server busy period (CSBP). We attempt to detect various types of application-level DDoS attack patterns, even including low-rate attacks, such as slowloris. We evaluate the performance of the proposed scheme via experiments based on captured packet traces and simulation.

Index Terms

denial-of-service attacks, application layer DoS attack, whitelist, admission control, attack flow detection

I. Introduction

DoS attack is a malicious attempt to disrupt the service provided by networks or servers. The power of a DoS attack is amplified by incorporating over thousands of zombie machines through botnets [1] and mounting a distributed denial-of-service (DDoS) attack. Although many defense mechanisms have been proposed to counter DDoS attacks [2], this remains a difficult issue, especially because the attack traffic tends to mimic normal traffic recently [3].

If a small number of machines are participating in a DoS attack to a selected server, the IP addresses of those attack machines might be detected using the approaches of [4, 5] without managing per-flow states. However, if the number of machines participating in a DoS attack increases, then each attack node needs not send attack traffic at a high rate, since the aggregate rate of attack traffic from many bot nodes can be sufficiently high to cause critical damage to the target node. This kind of low rate DoS attacks may not be easily detected by conventional metrics of per-flow traffic rate or SYN packet rates, since such low rate attack traffic is not much different from the traffic of normal users in terms of those metrics. Thus, the decrease of attack traffic rate due to the large population of attack machines recruited through a botnet is becoming a challenge to DDoS defense.

There is another factor that makes it more difficult to discriminate attack traffic of bots from the traffic of normal users. If DoS attack is launched at the application layer, then the attack can be effective with a small number of packets. For example, some specially crafted http request packets might induce an extensive database search, inject, or modify the data in the database disabling the target server ultimately. Netbot and blackenergy [6, 7] are well-known tools that can launch network/transport layer DoS attacks as well as application layer DoS attacks such as http get flooding attack and CC attack. Another tool, slowloris [8], sends partial http request packets with

incomplete http header to exhaust some resources, especially sockets, of a remote server so that no other normal users can access the server. In this case, the attacking node just sends subsequent headers at regular intervals to keep the sockets from closing. Since the packet interval can be up to 300 seconds, depending on the Timeout value of the Apache server, the average attack traffic rate can be maintained low. Thus, there are many types of application-level DoS attacks that are effective even with a small number of packets.

These low rate application-level attacks may not be detected by conventional DoS detection mechanisms based on SYN packet rate or traffic rate. Thus, a new approach is investigated to detect these application-level DoS attacks, especially targeting http web servers, in this paper. Recently emerging application-level DoS attacks may not be distinguished from normal user traffic. However, the intention of the attacking machines differs from that of normal users. Although normal users just want to get the information in which they are interested, malicious machines attempt to burden the target server as much as possible. Thus, we attempt to discriminate the attack flows from normal user flows based on the time interval during which each client makes the server busy. Since this step requires at least tens of seconds, this attack flow detection mechanism may be insufficient to protect a given web server in real time. Thus, we use an additional step of whitelist-based admission control to protect the given web server or server farm in real time.

The remainder of this paper is organized as follows. We first discuss related work in Section II. In Section III, we describe the outline of the proposed two-stage DDoS defense mechanism. In Section IV, we investigate whitelist-based admission control scheme, as the first stage of the proposed defense mechanism. It is important to control the number of entries in the whitelist to prevent the failure of the web server, and thus, we investigate the way to control the whitelist size by measuring the load on the server indirectly through the metric of average response time. In Section V, we propose a new attack flow detection mechanism based on the the time interval during which a given host makes the server busy. In Section VI, the performance of the proposed DDoS defense mechanism is evaluated by experiment and OPNET simulation. Finally, conclusions are presented in Section VII.

II. Related Works

Mirkovic et al. [9] has classified DoS attacks into two categories. The first type is the flooding attack, which targets overwhelming the resource of the victims, by sending a sufficiently large amount of traffic to the victims. The second type is the vulnerability attack, which takes advantage of vulnerability in the victim and sends specially crafted messages to the victim to disable it. In this paper, we focus only on the first type of attack. Peng et al. [2] also surveyed many types of DoS attack methods and classified DoS defense mechanisms into several categories: attack prevention, attack detection, attack source identification, and attack reaction. However, these survey papers did not discuss the defense against low rate DoS attacks or application-level DoS attacks in detail.

Several types of low-rate DoS attacks have been reported recently. One example is Shrew attack against TCP [10]. The attacker sends bursts of packets to create packet losses in a link and increments the retransmission timeout for certain TCP flows. The bursts are sent only around the expiration times of these flows to reduce the overall throughput. Another example is low-rate DoS attacks against application servers [11].

Regarding the defense against these low-rate DoS attacks, Sun et al. [12] reports that the ON/OFF traffic pattern of the Shrew attack can be detected using the autocorrelation of the traffic rate signal and dynamic time warping (DTW). Other researchers try to detect the attack, by analyzing the frequency information by spectral analysis [13] or by considering the correlation between the ON/OFF traffic rate signal and the round trip time of the affected flows [14]. However, since the attack traffic itself is generated by the attacker, there is a possibility that the attackers evade these traffic signature-based detection mechanisms by changing the traffic pattern. Thus, we attempt to detect attack flows based on the symptoms appearing at the server, rather than based on the incoming traffic pattern.

Some researchers have investigated the defense mechanism for the attacks against application servers. Macia-Fernandez et al. [15] investigated an approach to change the behavior of the server to lower the efficiency of the low-rate DoS attacks by making the instant when the server resource is available less predictable. This mechanism needs to be deployed on the server to be effective. However, it may not be easy to modify the internal system of many servers in an environment where many heterogeneous types of servers coexist.

Srivasta et al. [16] suggested a mechanism based on admission control and congestion control. In the admission control step, the client is required to solve a computational puzzle that is implemented through javascript. If the client node provides the correct solution to the puzzle, then the server provides the client with the authentication

code information, which is transformed into a private port number. In the congestion control step, the server monitors the behavior of each flow to give a higher priority to well-behaving flows. When the behavior is monitored, the response time for each request packet is also considered. The packet response time is related to the metric of the busy period considered in this paper, but they are different, as described in the subsequent sections. In addition, the congestion control functions are performed in the server-side kernel or firewall. However, since these defense functions can be a burden to the server itself, we consider the defense mechanism that can protect a single server or server farm while running on a machine physically separated from the servers.

Ranjan et al. [17] tried to provide DDoS resilience to web servers by allocating suspicion measure to each session and scheduling the requests of each session based on the suspicion measure. In order to quantify attack suspicion, Ranjan considers the session inter-arrival distribution, request inter-arrival distribution, and session workload profile. Among them, the session workload profile means the request type distribution of the packets belonging to a session. Each request type is identified by their average CPU utilization, which is also closely related with the response time of the server. However, the response time is different from the metric of the busy period discussed in this paper. Since the suspicion measure tries to capture the deviation of a session behavior from the normal model, it is very important to set up a reliable normal model. However, the normal model construction is usually difficult, since the ideal model can differ depending on the server, or it can change over time. Furthermore, there is a possibility that the normal model is polluted by the attackers working on a long time scale. On the other hand, our proposed mechanism requires only one threshold for the busy period ratio, and thus, it is less vulnerable to the normal model pollution problem. Another difference is that Ranjan's mechanism does not consider large scale attacks that involve a large number of attack sessions, but our proposed scheme can cope with such a large scale attack, because our mechanism registers malicious flows in a blacklist and drops the packets from the blacklisted IP addresses, instead of allowing them with a lower priority. Even when the attacker employs a botnet to send a large number of completely normal sessions, our proposed mechanism can protect the server by the whitelist-based admission control scheme in the first stage.

III. Outline of the Proposed Defense Mechanism

In this paper, we propose a mechanism that can protect a given web server farm from various types of application-level DoS attacks. In more detail, we attempt to mitigate the following well-known types of application-level DoS attacks and a limited set of lower-layer DoS attacks:

- SYN flooding attack
- application-level attacks targeting the CPU resource of the web server such as http get flooding or CC attacks
- application-level attacks targeting the other resources such as sockets or memory of the web server, e.g. slowloris

Although the detailed packet formats or the packet exchange patterns of these DoS attacks may differ, they usually have the same goal of disabling the server itself or hindering connection of normal users to the target server. As discussed in Section II, the DoS attack methodologies can be classified into two categories [9]: flooding attack and vulnerability attack. The second type of attack might be detected based on the known signatures. However, if the first type of attack is mounted on the application layer, especially through http packets, it is possible to offer a sufficiently high burden to degrade the performance of the web server, even with normal http request packets. For example, some specific query message can make the server busy for a while, by inducing heavy CPU processing on the database. Thus, the first type of attack cannot be detected based on the signature-based approaches.

One objective of this paper is to detect the first type of attack patterns based on the behavioral difference between the attack machines and normal users. The traffic rate of the source node including the SYN packet rate and the http request rate may not be effective any longer in discriminating the normal flows from the DoS attack flows, since the DoS attack can be effective, even with a low traffic rate after the emergence of low-rate attack tools, such as slowloris. Instead, we focus on the symptoms at the server, rather than the attack traffic pattern itself. Since almost all the DoS attack tools intend to disable the server or degrade the performance of the server by offering excessive work to the server or holding the limited resource of the server, we attempt to detect the malicious node based on the amount of work given by each source node. We use the concept of client-induced busy period to measure the amount of work given by each node. This will be investigated in more detail in Section V.

This attack flow detection mechanism is insufficient to protect the server from the sudden large scale DDoS attacks, since it takes tens of seconds to detect malicious flows based on the client-induced busy period. We

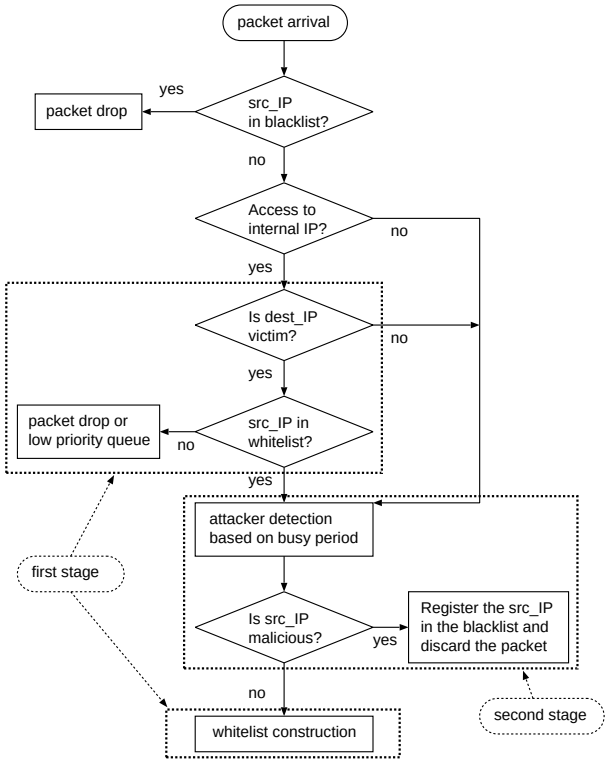


Fig. 1. Outline of the proposed 2-stage DDoS defense mechanism

use whitelist-based admission control, as an additional step to protect the server in real-time, even before most malicious flows are identified. Thus, the proposed DoS defense mechanism consists of two defense methodologies: busy period-based attack flow detection scheme and whitelist-based admission control scheme, as shown in Fig. 1. According to the flowchart in Fig. 1, when a packet arrives, if that packet is destined to a victim node that is under DoS attack, then the whitelist-based admission control policy is applied to the packet. The policy is simply to accept the packet, if the source IP address is registered in the whitelist already, but to drop or serve the packet with a lower priority, if the source IP address is not registered in the whitelist. Thus, this step becomes the first stage of the two-stage defense mechanism. The attack flow detection algorithm is applied to the packet afterwards, as a second step.

When the defense system monitors a packet destined to or outgoing from the protected servers, it first checks if the packet is coming from the blacklisted IP addresses. In that case, the packet is dropped promptly. If the non-dropped packet is destined to the victim inside the protected subnet, then we check if that packet is coming from the whitelisted IP addresses. When the packet is destined to a victim, if that packet is not from the IP addresses in the whitelist, then that packet is either dropped or served with a lower priority, as described before. All the non-dropped packets are inspected with the attack flow detection mechanism. If the source IP address is regarded as behaving maliciously, then that IP address is registered in the blacklist, and the packet is dropped. If the packet passes this stage, then the load on each internal IP address is monitored. If the number of nodes accessing a specific server exceeds some pre-defined threshold, then the whitelist is constructed for that server. If the load on the server exceeds a pre-specified threshold, then the server is declared as a victim, and only the IP addresses in the whitelist are allowed to access the server.

IV. Whitelist-based Admission Control

As we explained with Fig. 1, the first stage, i.e. the whitelist-based admission control stage, consists of two phases. The first is packet filtering based on the whitelist, especially for the victim servers. The second is the whitelist construction phase for the potential victim nodes. We discuss the second phase of the first stage, i.e. the last block in Fig. 1, in more detail in this section, since the first phase has been described in the previous section.

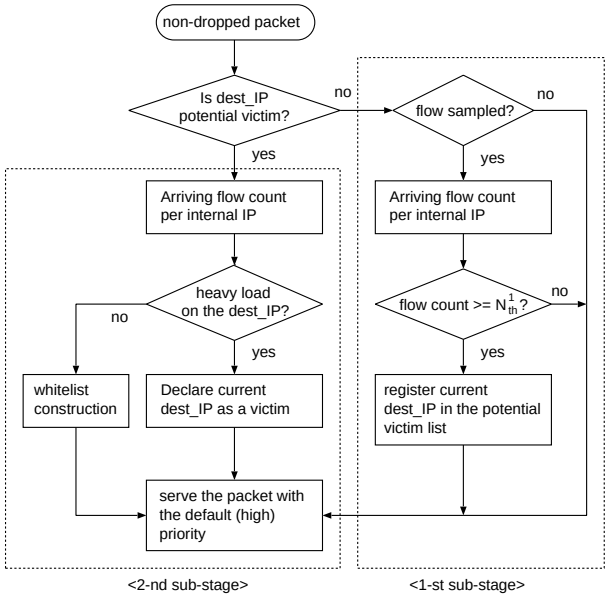


Fig. 2. Flowchart describing the whitelist construction process

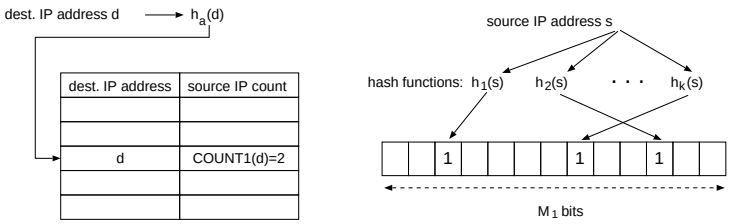
Whitelist or IP access history-based DDoS defense approach has been investigated by other researchers. Especially, Peng *et al.* [18, 19] tried to manage the list of normal IP addresses, based on the following observation. According to Jung et al. [20], when the number of clients increases during a DDoS attack, most of them are from new IP addresses that have not been seen before. Peng *et al.* used the following rules to accept only normal flows into the whitelist. The first rule is to accept a source IP address, if that IP address has accessed the server more than the minimum threshold days. The second rule is to accept an IP address, if the number of packets received from that IP address exceeds some threshold. However, even the malicious nodes can easily satisfy these conditions, if the threshold values are disclosed. Thus, the whitelist can be easily poisoned and the efficacy is likely to be limited in the presence of real DDoS attacks. Another whitelist-based DDoS defense mechanism has been proposed by us [21] to reduce the memory requirement of the system compared to Peng *et al.*'s scheme. In the previous version [21], the whitelist is constructed in a rather short time interval to reduce the possibility of whitelist poisoning. However, even that approach cannot be considered as resolving the whitelist poisoning issues clearly. The currently proposed DoS defense mechanism can cope with this whitelist poisoning problem much better, since the attack flow detection mechanism can sift out the malicious flows from the whitelist.

We use a modified version of the whitelist proposed in [21] for our whitelist-based admission control mechanism. However, we investigate one important issue on the whitelist that was not resolved in [21]. Although it is very important to control the size of the whitelist so that the servers in the protected region do not crash, the issue of determining the whitelist size was not discussed in [21]. This issue will be discussed in more detail after explaining the whitelist used in the proposed mechanism.

Fig. 2 shows the procedures performed in the last block of Fig. 1. The whitelist construction phase consists of two substages. In the first substage, we detect potential victims. If the number of external machines accessing an internal IP address is larger than or equal to the threshold N_{th}^1, then that internal IP address is considered as a potential victim. In order to reduce the storage space and packet processing overhead, packets are sampled and only sampled packets are inspected at this substage. The IP addresses of the potential victims are managed in the connection status table T_2. This will be explained shortly.

In the second substage, the whitelist is constructed for the potential victim nodes, and the victims are detected based on the load for each server. If an internal IP address is declared as a victim, then the whitelist will be used to perform admission control, as shown in the box corresponding to the first stage in Fig. 1. We now describe more details on the first and second substages.

In the first substage, we select internal IP addresses accessed from multiple external IP addresses by sampling. The internal IP addresses selected in the first substage are called potential victims, and the whitelist is constructed only for those selected potential victims in the second substage to reduce the memory size required for the whitelist.



(a) Connection status table (T_1) (b) Bloom Filter (B_1) for source address tracking

Fig. 3. The structure of tables managed in the first substage (h_a is a uniform random hash function.)

In the first substage, we sample flows in the following way. Let p_s denote the sampling probability, and let h_p be a uniform random hash function that maps the source and destination IP address pairs, (SrcIP, DstIP), to a real number in the range of $[0, 1)$. When s and d are the source and destination IP addresses of an arriving packet, if $h_p(s, d) < p_s$, then the IP address pair (s, d) is sampled.

In the first substage, we manage one connection status table T_1 and one Bloom filter B_1, as shown in Fig. 3. The hash table T_1 counts the number of distinct IP addresses accessing a specific internal IP in a given time interval I_1. $COUNT1(d)$ counts the number of sampled IP addresses accessing an internal address d. Each sampled source IP address is registered in the Bloom filter B_1, which is M_1-bit long and has k hash functions, as shown in Fig. 3(b). We use Bloom filters [22] to reduce the memory size and B_1 is shared among different destination IP addresses in order to raise efficiency of the limited memory space.

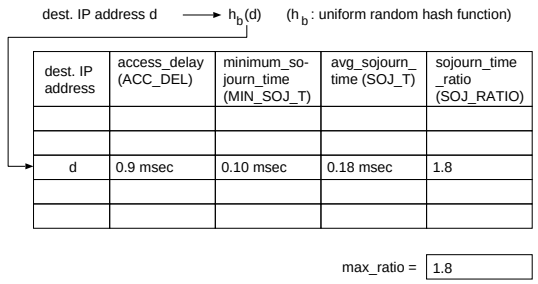
If the value of $COUNT1(d)$ reaches a pre-specified threshold N^1_{th}, then the IP address d is considered as a potential victim and the whitelist is constructed for d in the second substage. $COUNT1(d)$ counts only sampled source addresses accessing d. Thus, we need to note that the total number of IP addresses that accessed d is N^1_{th}/p_s on average when $COUNT1(d) = N^1_{th}$.

In the second substage, the whitelist is constructed only for the internal IP addresses selected in the first substage. After constructing the whitelist, if the load on the internal server is considered high, then existing flows, i.e. packet streams from the IP addresses registered in the whitelist, are served with a high priority, while new flows are either served with a lower priority or dropped depending on the policy to prevent the failure of the protected server. In this paper, packet discard is the default policy for lower priority flows. For the second substage, it is important to establish the criterion to determine if the server is under a heavy load or not. This issue is discussed hereafter.

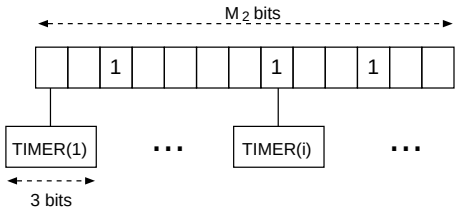
We model each web server as a queueing system to investigate the issue of estimating the load on the servers. Since the defense system may not be able to see all the internal behavior of many web servers in the protected network, we investigate how to estimate the load on each web server based on the packets exchanged between the internal server and the external clients. In more detail, we first estimate the packet sojourn time in the server, i.e. the time interval from the arrival time of the http request message to the time when the http response message departs from the web server. Since the packet sojourn time is dependent on the traffic load on the system, we attempt to estimate the load based on the change of sojourn time. Realistic modeling of web server can be a complex problem. Since the detailed server modeling is not a goal of this paper, we investigate the problem in a simplified environment of an $M/M/1$ queueing system to suggest a simple guideline for this issue. The average sojourn time (W) for an $M/M/1$ queueing system is well-known to be [23]

$$W = \frac{1}{\mu(1-\rho)}, \tag{1}$$

where $1/\mu$ is the average service time of the server, and ρ is the offered load to the queueing system. From (1), we can easily find when the offered load is very low, i.e. $\rho \approx 0$, $W \approx 1/\mu$. That is, if the load is very low, then the arriving packets are likely to be served directly without queueing delay. If $\rho = 0.8$, then $W = 5/\mu$. Thus, if the load is 80%, then the sojourn time or the server response time increases up to five fold compared to the case of a negligibly small load. Using this analysis as a guideline, if the ratio of the average sojourn time in the current time window to the minimum average sojourn time is less than five, then the server is not considered to be under attack, and new source IP addresses are accepted into the whitelist. However, if the ratio exceeds five, then the server is regarded as a victim, and no more new IP addresses are accepted into the whitelist.



(a) Connection status table (T_2)



(b) Whitelist implemented as a Bloom Filter (B_2)

Fig. 4. The structure of tables managed in the second substage

In the second substage, we manage one connection status table T_2 and one Bloom filter B_2 as shown in Fig. 4. The hash table T_2 tracks the load on each internal node selected in the first substage via the metric of sojourn time. The field of ACC_DEL(d) measures the access delay between the defense system and the server with the IP address d. The access delay is used to estimate the sojourn times from the packet monitoring times at the defense system, and this will be investigated in more detail in the next subsection. The field of MIN_SOJ_T(d) retains the minimum value of the average sojourn time in server d. The field of SOJ_T(d) has the average sojourn time in the current window. SOJ_RATIO(d) is equal to the ratio of SOJ_T(d) to MIN_SOJ_T(d).

The Bloom filter B_2 is the whitelist that manages the list of normal IP addresses accessing the potential victim nodes, and the whitelist B_2 is shared by the all potential victims in the protected subnet. We calculate the maximum value of the sojourn time ratios for all the selected servers, since the whitelist B_2 is shared among different servers. If the maximum sojourn time ratio exceeds five, then no more client IP addresses are accepted into the whitelist. If the maximum sojourn time ratio falls below five, then new IP addresses can be accepted into the whitelist again.

In B_2, a timer is allocated for each bit, as shown in Fig. 4(b). $TIMER(i)$ represents the timer allocated for the i-th bit in the M_2-bit Bloom filter B_2. In our scheme, if there has been no packet between an IP address pair over some pre-specified time interval, then the connection between the IP address pair is considered to be disconnected. Thus, the timer in the Bloom filter B_2 is used to monitor the connectivity between internal and external IP address pairs. We use similar values for the parameters related to B_2, i.e. the number of hash functions used in the Bloom filter k_2, the default value of the timer R, and the timer update interval I_T, as the ones used in [21]: $k_2 = 7$, $R = 7$, and $I_T = 30$secs. Then, if there is no packet exchange between a node pair over 180 seconds, the connection between the node pair will be regarded as being disconnected [21].

A. Estimation of Access Delay between The Defense System and The Server

As explained above, the load on the server is inferred based on the server response time or the packet sojourn time in the server. If we know the time when the request packet arrives at the server and the time when the response packet departs from the server, then the server response time or the packet sojourn time can be calculated as the difference of these time values. Although we assume that the defense system is located near the protected server, it may not be connected to the server directly as shown in Fig. 5. Then, the packet sojourn time needs to estimated from the packet monitoring times at the defense system.

Let us consider the topology of Fig. 5, where the defense system is separated from the web server by several intermediate nodes. In this case, the intermediate nodes $R_i (i = 1, \ldots, h-1)$ can be either a router or an

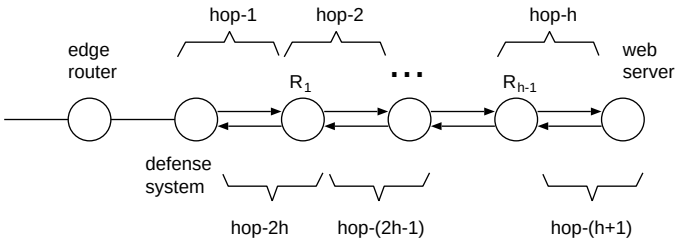


Fig. 5. Example topology describing the interconnection between the defense system and the protected web server

Ethernet switch. Let us consider an example case where the defense system observes a http request packet and the corresponding http response packet at times t_1 and t_2 ($t_2 > t_1$), respectively. Let us assume that those packets arrived at and departed from the server at times t'_1 ($t'_1 > t_1$) and t'_2 ($t'_2 < t_2$), respectively. Then, $t'_2 - t'_1$ is the packet sojourn time (or the server response time). In this case, $t'_1 - t_1$ is referred to as the forward access delay from the defense system to the server, and $t_2 - t'_2$ is referred to as the backward access delay. The summation of these two terms, i.e. $(t'_1 - t_1) + (t_2 - t'_2)$, is referred to as the access delay d_{access} between the defense system and the web server. If the traffic load is not high inside the subnet, then this access delay is likely to remain a constant. Since t_1 and t_2 can be measured at the defense system, if we know the access delay in advance, then we can estimate the packet sojourn time from the difference between $(t_2 - t_1)$ and d_{access}, i.e. $(t_2 - t_1) - d_{\text{access}}$. Thus, if we know the access delay between the defense system and the server, then the packet sojourn time can be easily estimated from the measurement results at the defense node. We now introduce a packet pair-based access delay estimation mechanism.

In Fig. 5, there are $h - 1$ intermediate nodes between the defense system and the web server, and we can find $2h$ hops on the round-trip path between the defense system and the web server. Especially, the $(h + 1)$-th hop corresponds to the web server. Let C_i, s_i, p_i, g_i, and q_i denote the link rate, the service time, the processing delay, the propagation delay, and the queueing delay at the i-th hop, respectively. We assume that all the link rates, i.e. C_i's, are identical to C: this is a reasonable assumption, if the defense system belongs to the same subnet as the web server. To simplify the problem, we assume that the http request packet size is fixed to L_f. The http response messages are usually divided into multiple TCP segments, since the message size can be larger than the maximum transmission unit (MTU) size. In this case, we only consider the time interval from the request arrival time to the time when the first segment of the response message departs from the server as the sojourn time or the server response time, since the effect of the traffic load can be reflected in the queueing delay component. Thus, we assume that the first segment of the http response message has a fixed size of L_b, the MTU size of the underlying transport technology. Under these assumptions, s_i can be summarized as

$$s_i = \begin{cases} L_f/C, & i \le h, \\ L_b/C, & i > h. \end{cases} \tag{2}$$

We assume that this component is fixed and negligibly small, since the processing time at the intermediate nodes, such as routers or Ethernet switches, including the functions such as table lookup or longest prefix matching, is usually minimized for the high performance. However, the processing time at the server, p_{h+1}, can differ depending on the types of request. Thus, p_{h+1} is not assumed to be fixed. The propagation delay g_i is usually assumed to be fixed, and the queueing delay q_i is usually a variable component. Thus, the round-trip delay D_{rt} can be expressed as

$$\begin{aligned} D_{rt} &= \sum_{i=1}^{2h} (s_i + p_i + g_i + q_i) \\ &= \sum_{i=1}^{2h} g_i + \sum_{i=1}^{h} \frac{L_f}{C} + \sum_{i=h+1}^{2h} \frac{L_b}{C} + \sum_{1 \le i \le 2h, i \ne h+1} p_i + p_{h+1} + \sum_{i=1}^{2h} q_i. \end{aligned} \tag{3}$$

The first four summation terms in the above relation can be considered constant under the above assumptions. If we let D_f denote the summation of those terms, then (3) can be simplified to

$$D_{rt} = D_f + p_{h+1} + Q,$$

where $Q = \sum_{i=1}^{2h} q_i$. The access delay d_{access} can be expressed as

$$d_{\text{access}} = D_f - \frac{L_b}{C} + \sum_{1 \le i \le 2h, i \ne h+1}^{2h} q_i. \tag{4}$$

Then, the minimum access delay d^m_{access} is obtained when the queueing delay components, $q'_i s$, are zero, and can be expressed as

$$d^m_{\text{access}} = D_f - \frac{L_b}{C}. \tag{5}$$

We assume that the access delay d_{access} is usually close to the minimum access delay d^m_{access} since the number of hops between the defense system and the web server is small, and the queueing delays at the intermediate switch or routers can be small due to the high speed of the LAN. In order to estimate the minimum access delay d^m_{access} in (5), we send two back-to-back http request packets to the default URL of the server. If we assume that there was no cross traffic, then the first http request packet does not experience queueing delay at any node, but, the second packet experiences the queueing delay of $p_{h+1}(1) - L_f/C$ at the server node, while the first packet is served. $p_{h+1}(j)$ is the processing delay experienced by the j-th test packet at the $(h+1)$-th node, and we assumed that the request packet processing time at the server is longer than the packet transmission time at the ingress link of the server since the request packets are usually processed at the application layer. If we let $D_{rt}(j)$ denote the round-trip delay experienced by the j-th test packet, then we obtain

$$\begin{aligned} D_{rt}(1) &= D_f + p_{h+1}(1), \\ D_{rt}(2) &= D_f + p_{h+1}(2) + p_{h+1}(1) - L_f/C. \end{aligned} \tag{6}$$

Then, we can derive the estimator $\hat{d}_{\text{access}}$ for the minimum access delay as

$$\hat{d}_{\text{access}} = 2D_{rt}(1) - D_{rt}(2) - \frac{L_f + L_b}{C}. \tag{7}$$

We use $\hat{d}_{\text{access}}$ to estimate the access delay since we assume that d_{access} is close to d^m_{access}. From (5), (6), and (7), we can calculate the relative error of the estimator (e) as

$$e = \frac{E\left[|\hat{d}_{\text{access}} - d^m_{\text{access}}|\right]}{E\left[|d^m_{\text{access}}|\right]} = \frac{E\left[|p_{h+1}(1) - p_{h+1}(2)|\right]}{D_f - \frac{L_b}{C}}.$$

If $p_{h+1}(1)$ is always the same as $p_{h+1}(2)$ for two back-to-back http request packets, then the error could be zero in the above equation. In [11], the processing time for identical requests is modeled as a random variable with a normal distribution. In order to find the relation between $p_{h+1}(1)$ and $p_{h+1}(2)$, we implemented a tool that sends two back-to-back http or ICMP packets to the target node. We found that if two packets are separated by much more than the transmission time, then the processing time between them can be very different. However, when two packets are sent to the server back-to-back, the two round-trip delays were very close to each other. Thus, we can indirectly infer that $p_{h+1}(1)$ is usually close to $p_{h+1}(2)$, especially when these two packets are sent back-to-back.

We analyze the error of the estimator in (7) in more detail through simulation in Section VI. Since the two relationships in (6) are obtained under the assumption that the first test packet of a packet pair does not experience any queueing delay, we need to send a sufficiently large number of packet pairs from the defense system to the server, and we should use only the packet pairs whose first test packet experiences the minimal round-trip delay when we estimate the access delay.

V. Busy Period-based Attack Flow Detection Mechanism

In this section, we discuss a new DoS attack flow detection mechanism. Recently, the DoS attack traffic rate has been gradually lowered since a huge number of machines can be easily recruited through a botnet and application layer legitimate packets can induce higher resource utilization for the target server than lower layer packets. Thus, a traffic rate itself, including SYN packet rates or http request packet rates, may not be effective in discriminating the normal user behavior from the malicious bot behavior in the future. We use a new metric termed client-induced server busy period (CSBP) to detect low-rate application-level DoS attack flows, as well as well-known DoS attack patterns, including SYN flooding attacks.

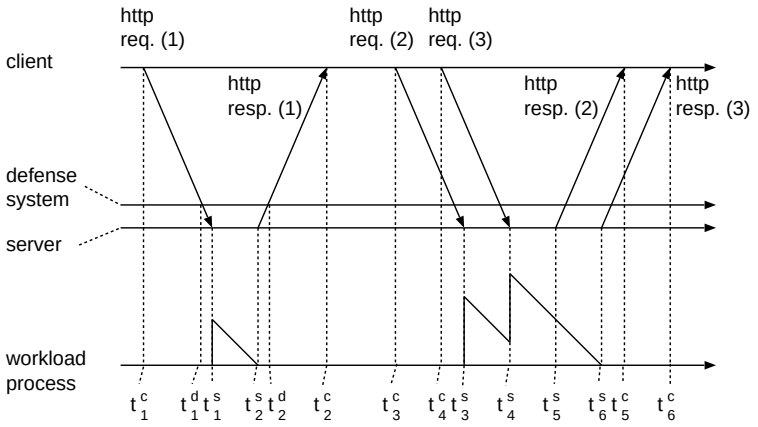


Fig. 6. Detection of client-induced busy period based on transit packet monitoring time

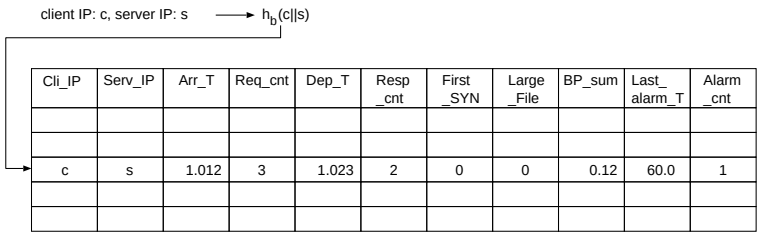
We explain the concept of CSBP using an example interaction between a client and a server in Fig. 6. t_i^c represents the time of i-th packet departure or arrival event at the client node. t_i^d and t_i^s denote the i-th packet arrival or departure time at the defense system and the server, respectively. The lower part of the figure shows a workload (or unfinished work) process at the web server, where the workload $U(t)$ at time t is defined as the time required to complete the service of all messages present at time t [23], assuming that the web server can be modeled as a queueing system. In Fig. 6, the server was busy serving the three http request packets from the selected client in two time intervals, $[t_1^s, t_2^s]$ and $[t_3^s, t_6^s]$. These time intervals become the CSBP between the selected client and the server. Since the malicious node participating in a DoS attack tends to make the server as busy as possible even with a small number of packets, we use the ratio of CSBP to the measurement interval, called CSBP ratio, to discriminate the normal clients from the malicious bots.

If the defense system is physically separated from the web server, then the defense system cannot monitor the internal status of the server, especially the resource utilization, directly. However, as shown in Fig. 6, the http response packet can be sent back to the client, only when the required task is completed by the server. Thus, we track the server busy period using the http response departure times and the http request arrival times at the server. As an example, the length of the first busy period can be calculated as $t_2^s - t_1^s$ from the http request packet arrival time t_1^s and the http response packet departure time t_2^s in Fig. 6. However, since the defense system does not know the packet departure and arrival times at the server, the defense system estimates the busy period length, based on its own packet monitoring time, assuming that the access delay is negligibly small between the defense system and the server. Although the busy period (CSBP) length can be estimated more accurately if we consider the access delay estimated by the mechanism described in subsection IV-A, the effects are likely to be limited, since the CSBP lengths are usually much longer than the access delay.

The busy period can be easily tracked, if the http response message is not segmented. Let us assume that the defense system monitored N http request packets and N http response packets. Let a_i denote the time when the defense system monitors the i-th http request packet, and let d_i denote the time when the i-th http response packet is monitored. If the i-th response packet is monitored earlier than the $(i+1)$-th request packet, then there will be N disjoint busy periods of $[a_i, d_i](i = 1, \ldots, N)$. When $k \geq 2$, if $d_{j+m} \geq a_{j+m+1}$ for all $m \in [0, k-2]$, then we have an extended busy period of $[a_j, d_{j+k-1}]$ for k consecutive http request packets.

Even when a single http response message is divided into multiple TCP segments, if we can identify the last segment of each response message, and d_i is redefined as the monitoring time of the last segment of the i-th http response message, then the busy period can be accurately tracked by the above approach. However, it is not possible to know if the current segment is the last segment of a long message solely based on the TCP header. It may be possible to identify the last segment of a http response message, if we use both of the TCP header and the http header information. However, the more http information we use, the higher the complexity of the defense system will be. Thus, we investigate a scheme that can estimate the busy period, i.e. CSBP, with an acceptably small error, while minimizing the use of the http header information.

Since the CSBP is managed for each pair of client and server IP addresses, the proposed defense mechanism



Cli_IP	Serv_IP	Arr_T	Req_cnt	Dep_T	Resp _cnt	First _SYN	Large _File	BP_sum	Last_ alarm_T	Alarm _cnt
c	s	1.012	3	1.023	2	0	0	0.12	60.0	1

Fig. 7. per-flow busy period management table

needs to maintain per-flow information, as shown in Fig. 7. The $\mathrm{Arr\_T}$ field tracks the starting time of each busy period. When a new TCP connection is set up, $\mathrm{Arr\_T}$ retains the time when the first SYN packet is monitored for the first busy period to accommodate the case where the server resource is exhausted by SYN flooding attacks. In case of persistent http mode where multiple http requests are sent during one TCP session, the http request arrival time is written into the $\mathrm{Arr\_T}$ field from the second busy period of the selected TCP session. The $\mathrm{Req\_cnt}$ field counts the number of http request packets observed in the current busy period. The $\mathrm{Dept\_T}$ field tracks the potential finish time of the current busy period based on the monitoring time of the http response packets from the server. The $\mathrm{Resp\_cnt}$ field estimates the number of the http response messages in the current busy period. The number of http response messages is estimated, since it is difficult to identify the last segment of each http response message without complete reassembly of the message. This field is used to track the end of each busy period. The $\mathrm{First\_SYN}$ field is set to one, when a new SYN packet starting a new busy period is monitored. This field is used to include the TCP connection set-up time as part of the first busy period of that TCP session. The default value of the $\mathrm{Large\_File}$ field is zero, and it is set to 1 only when the value of *content length* field in the header of the http response packet is larger than a threshold L_{th}. This field is used to prevent false positives for the special cases of streaming or huge file downloading, as described below. The $\mathrm{BP\_sum}$ field accumulates all the busy period lengths detected in the current measurement interval. If T is the length of the measurement interval, then the value of CSBP ratio is calculated as $\mathrm{BP\_sum}/T$, at the end of each measurement time interval. If the CSBP ratio exceeds a predefined threshold ζ, then one alarm is raised for the client. If the number of consecutive alarms is larger than or equal to a predefined threshold N_a, then the corresponding client IP address is considered malicious and is registered in the blacklist. The $\mathrm{Alarm\_cnt}$ field counts the number of consecutive alarms. The $\mathrm{Last\_alarm\_T}$ field retains the last alarm time (in seconds) for the client IP, and this field is used to check the consecutiveness of the alarms. For example, when a new alarm is raised for a specific IP address, if the last alarm occurred more than the measurement time interval ago, then $\mathrm{Alarm\_cnt}$ will be set to 1, since the alarms are not consecutive. On the other hand, if the last alarm occurred just the measurement time interval ago, then $\mathrm{Alarm\_cnt}$ will be incremented by one, since the alarms are consecutive.

The algorithms in Figs. 8 and 9 comprise the proposed busy period estimation algorithm. In more detail, the goal of these algorithms is to estimate the CSBP ratio as accurately as possible at the end of each measurement time window. The algorithm in Fig. 8 is applied to the packets coming from the clients, and the algorithm in Fig. 9 is applied to the packets outgoing from the internal server. If the http response messages are not divided into multiple segments, then we can easily identify the last packet closing the current busy period by comparing the number of http request packets to that of http response packets. For example, if the number of http request packets is n and the number of http response packets is $n-1$, this means that the current busy period is not closed, since one http response packet has not yet arrived. If a new http response packet arrives under that condition, then all the http response packets have arrived, and the busy period must be closed, since we assumed that the http response messages are not segmented. This is the basic idea of the proposed busy period detection algorithm. Even though the http response messages are segmented, this algorithm can accurately detect the busy period, if we can identify the last segment of each http response message and the http response packet count is increased only at the arrival of the last segment of each http response message.

However, it is not easy to identify the last segment of each http response message accurately if we do not reassemble the message on each segment arrival. On the other hand, we observed that the PSH bit in TCP header is always set to 1, if that packet is the last segment of any long message. However, we also found that the PSH bit is set to 1 in the middle segments occasionally, but the frequency of those events was not very high. Thus, we

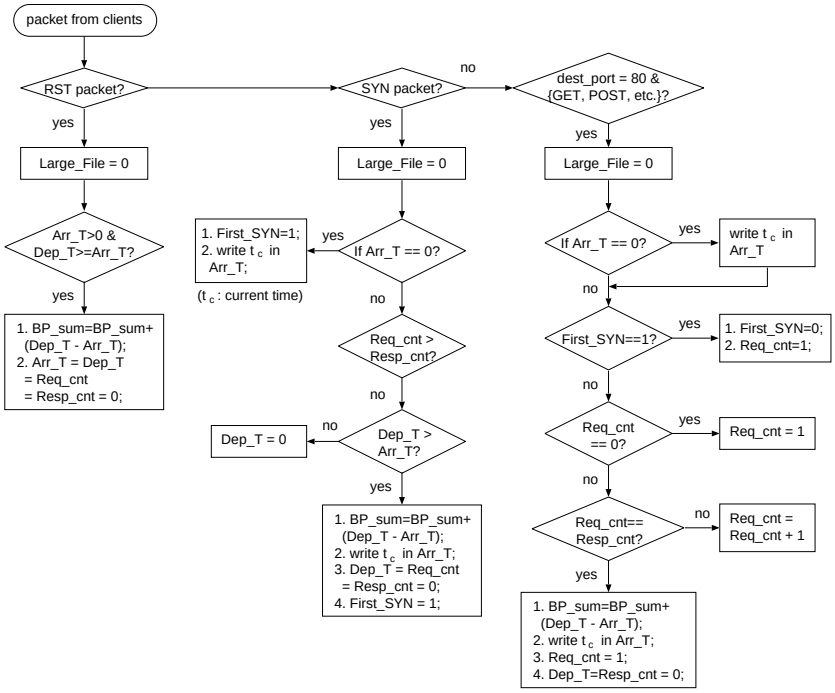


Fig. 8. Busy period detection algorithm applied to the packets coming from clients

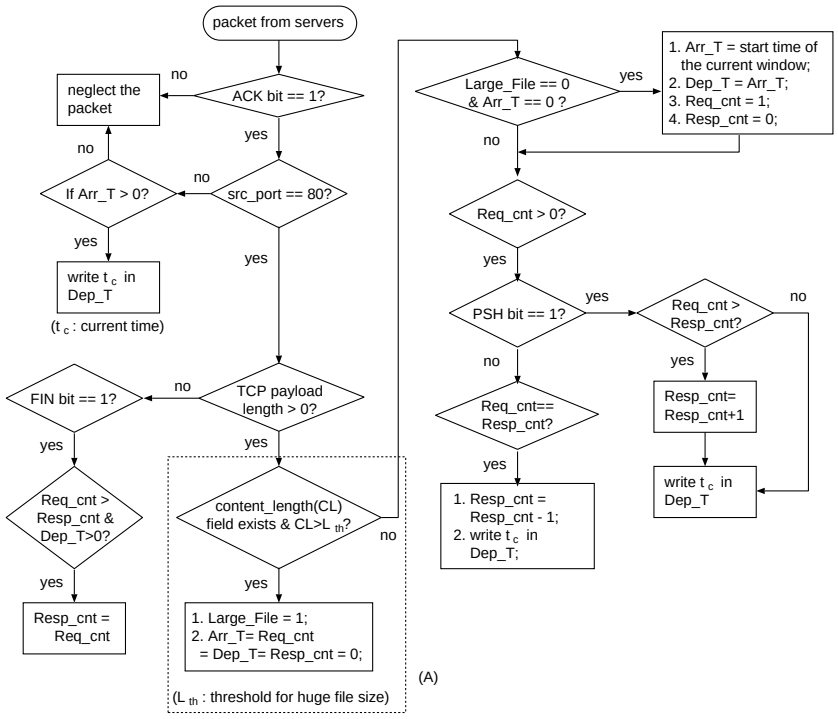


Fig. 9. Busy period detection algorithm applied to the packets outgoing from servers

use the following approach to estimate the last segment of each http response message. When we observe a TCP segment from the server whose source port number is 80 and PSH bit is set to 1, we increase the response count, $\mathrm{Resp\_cnt}$, by 1, only if $\mathrm{Resp\_cnt} < \mathrm{Req\_cnt}$. If the currently received TCP segment is not the last segment, then that packet is usually accompanied by another TCP segment whose PSH bit is not set. Thus, if we observe such a TCP segment, whose PSH bit is not set subsequently, then we decrease the response count, $\mathrm{Resp\_cnt}$, by 1 to avoid erroneous closing of a busy period, only when $\mathrm{Resp\_cnt} = \mathrm{Req\_cnt}$.

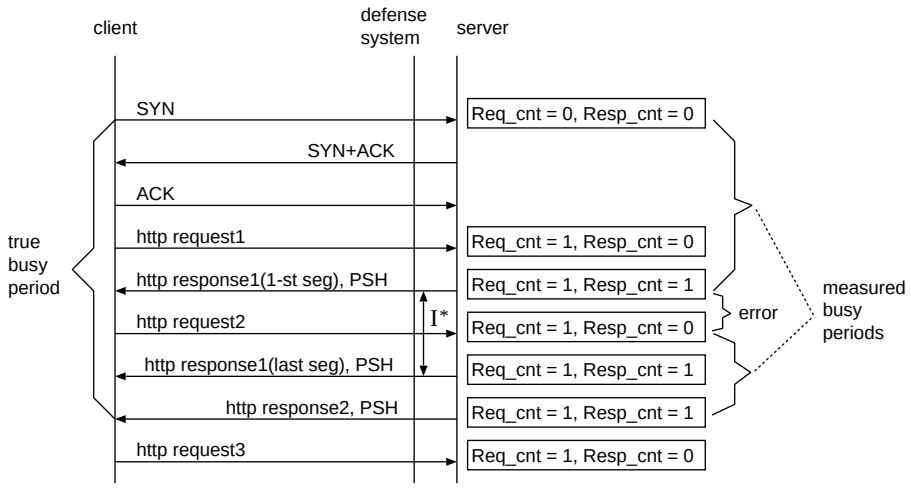


Fig. 10. Scenario illustrating the error of the proposed busy period detection algorithm

The CSBP ratio might be underestimated by the errors described in the scenario of Fig. 10, especially when a new http request gets in between two consecutive http response segments, e.g. between the first and second segments in the figure. However, even in this case, the magnitude of the error component should be shorter than the length of the interval between two consecutive http response segments, i.e. I^* in Fig. 10. Since this interval is usually very short and the PSH bit is rarely set in the middle segments, we expect that the effect of this error will be limited. The accuracy of the proposed algorithm is investigated in more detail through experiment in the next section.

The currently described version of the algorithm has one important limitation. When a client accesses a server for streaming services or it downloads a huge file from the server, a long busy period might be observed while the streaming service is provided or the huge file is downloaded. Thus, false positives can occur for the clients using those services. We treat those cases as exceptions, since the streaming service and huge file downloading are not malicious behavior, as shown in the box denoted by (A) in Fig. 9. Those special services inducing a high resource utilization over an extended period can be identified by the *content length* field in the http header of the first http response segment, since this field represents the total size of the file that will be delivered subsequently through the requested service. Thus, in the final version of the algorithm, if the *content length* value exceeds a threshold L_{th}, then the busy period corresponding to that specific service is not reflected in the calculation of the CSBP ratio, as shown in box (A) of Fig. 9. The value of L_{th} is fixed to 50,000 Bytes after extensive experiments.

The interleaving of any other types of http attack packets with the http request packets inducing the huge file downloading cannot be used to mask the underlying attacks, since any subsequent http request packet will be detected as a start of a new busy period, as shown in Fig. 8. If many malicious nodes send http request packets corresponding to the streaming service or huge file downloading simultaneously, then the failure of the server can be prevented by the whitelist-based admission control scheme in the first stage, since the whitelist is frozen when the load on the server reaches a predefined threshold as shown in Figs. 1 and 2.

VI. Numerical Results

In this section, we evaluate the performance of the proposed DDoS defense mechanism via simulation and experiment.

A. Evaluation of access delay estimation mechanism

In the proposed scheme, we detect the attack on the server based on the average sojourn time at the server, that is, the server response time. If the defense system is physically separated from the server, then the defense system cannot monitor the server response time directly from the request arrival and the response departure times. In order to estimate the packet sojourn time more accurately in this case, we proposed an access delay estimation mechanism in subsection IV-A. The accuracy of the proposed access delay estimation scheme is analyzed via OPNET simulation in this subsection.

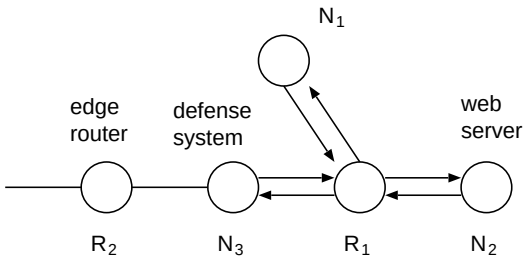


Fig. 11. Network topology to evaluate the access delay estimation scheme

Fig. 11 shows the network topology to evaluate the access delay estimation mechanism. The defense node N_3 is connected to the internal web server N_2 via LAN switch R_1. All the internal traffic destined to the web server is modeled as being generated by a single node N_1, and externally generated traffic can reach the web server only via the edge router R_2 and the defense system N_3. Thus, the defense system is only two hops away from the server in this topology.

All the link rates are fixed to 100 Mbps. To simplify the problem, we assume that each http request packet of the size of 500 Bytes induces only one http response packet of the size of 1500 Bytes. We assume that the server processing time is exponentially distributed with an average value of 1 msec. The cross traffic is generated by Node N_1, and the cross traffic is modeled as Interrupted Poisson Process (IPP). The average on and off period lengths are 0.1 and 0.9 seconds, respectively. The average traffic rate during on period is 2 Mbps. Since the cross traffic is generated by multiplexing 10 IPP streams of the given parameters, the load on the web server becomes 50 $\%$. Since the transmission times of the http request packets and the http response packets are 40 usec and 120 usec, respectively, the server processing stage becomes the major bottleneck in the round-trip path between the defense system and the server.

Fig. 12 shows the range of the estimated access delay, i.e. $[\mu - \sigma, \mu + \sigma]$, where μ and σ are the mean and the standard deviation of the estimated access delays obtained after 10 experiments, under the given environment. We find that the estimator for the access delay, described in subsection IV-A, converges to the real access delay very quickly, as the number of probing packet pairs increases.

It is not easy to evaluate the performance of the proposed access delay estimation mechanism in real networks, since it is difficult to measure the exact value of each delay component, e.g. the propagation delay, on each hop. We implemented a tool to estimate the access delay, and ran it to estimate the access delay from a local machine to an internal web server inside the campus network. Although we cannot verify the correctness of the estimated access delay, we find a similar tendency of convergence to some specific value. However, more packet pairs, around a few hundred, were required to get a converged value of access delay compared to the simulation case, since some simplifying assumptions may not be valid in real networks. For example, the minimum delay may not be obtained with a small number of packet pairs.

B. Evaluation of the busy period-based attack flow detection mechanism

In this subsection, we evaluate the accuracy of busy period estimation algorithm described in Figs. 8 and 9 of Section V. Then, we investigate the efficacy of the busy period-based attack flow detection mechanism.

We use the following three types of normal traffic patterns and three types of malicious traffic patterns to evaluate our proposed scheme:

- web browsing to cnn web server (normal traffic pattern)
- huge file downloading (normal traffic pattern)
- video streaming (normal traffic pattern)
- Netbot attacker (attack traffic pattern)
- Blackenergy attacker (attack traffic pattern)
- Slowloris attacker (attack traffic pattern)

We captured 20 minute long packet traces for each traffic pattern. We captured the packets that corresponded to normal behavior, while consenting volunteers were accessing a specific external web server, i.e. cnn.com in this

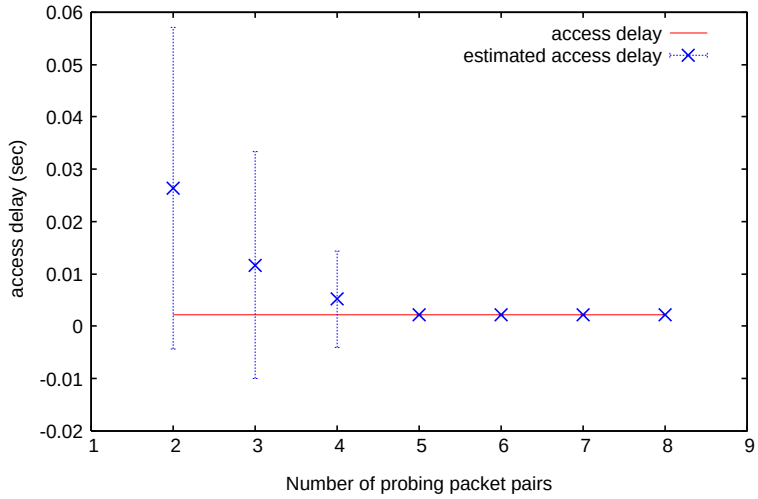


Fig. 12. Access delay estimation results

study. When we explained the proposed busy period-based attack flow detection mechanism in the previous sections, we assumed that the protected server resides in the same subnet as the defense system or the monitoring point. However, the normal traffic patterns are obtained between the local client nodes and the external web server, and thus, the access delay may not be negligibly small. If we do not get rid of the access delay component from the measurement results, then the measured busy period becomes an upper bound of the true busy period at the external server. In order to see whether the defense (or monitoring) system can discriminate normal flows from bad flows even in this worse case, we did not get rid of the access delay components. The attack traffic patterns are captured from the corresponding tools deployed on the local machines, as opposed to the capturing of the normal traffic patterns. We generated http get flooding attacks and CC attacks using Netbot attacker and Blackenergy, and used the traffic trace of as low a traffic rate as possible for the evaluation.

We investigate the accuracy of the proposed busy period detection algorithm with the normal web browsing traffic pattern. Fig. 13 compares the cumulative distribution function (cdf) of the busy period estimated by the proposed busy period detection algorithm to that of a manually measured busy period. Manual detection means that the busy period is determined based on the information about the correct last segment of each http response message. The well-known packet capture tool wireshark usually provides such information. We can observe that the cdf of the estimated busy period agrees well with that of the manually measured busy period. Although Fig. 13 is obtained from the web browsing traffic pattern, we find a similar tendency for other traffic patterns, that is, the proposed busy period detection algorithm closely estimates the true busy period.

Fig. 14 shows the estimated CSBP ratio over time, especially for the web browsing traffic pattern. The measurement time window length is 10 seconds. There are several points in each measurement window, as the graphs for multiple web servers related with CNN are superposed in one figure. We can observe that the CSBP ratios are sometimes high, even over 0.5. We find that some normal http request messages make the server busy over 5 seconds during the 10 sec measurement window, although the frequency of these events is rather low. In addition, we find that most of the CSBP sample points over 0.5 are due to the following strange behavior. Fig. 15 explains the strange behavior. In most of the cases where the CSBP ratio is over 0.5, the client has made TCP connection by completing the three-way handshaking process, but it did not do anything afterwards until the connection is terminated by the server with the error message of “HTTP/1.0 408 Request Time-out”. It is fair to raise an alarm if this kind of behavior occurs frequently, since this form of behavior can be exploited as another type of DDoS attack targeting the exhaustion of the server sockets. The client machine might have been infected by a virus or a worm non-detectable by the up-to-date anti-virus solution. However, this behavior did not occur frequently during our measurement period.

When we increased the measurement window size to 30 seconds, we found that the average level of CSBP ratios reduced significantly for the normal web browsing traffic pattern, as shown in Fig. 16. Thus, we fix the measurement window size to 30 seconds hereafter. From the figure, we can observe that the CSBP ratios stay below 0.2 most of

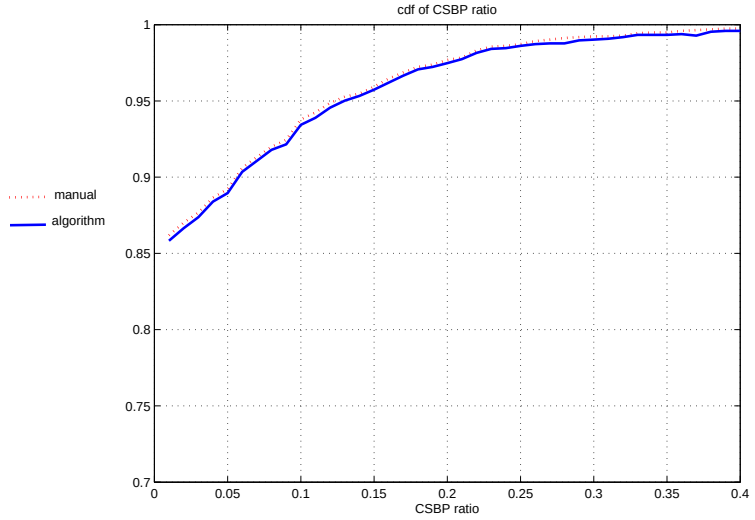


Fig. 13. Accuracy of busy period detection algorithms described in Figs. 8 and 9

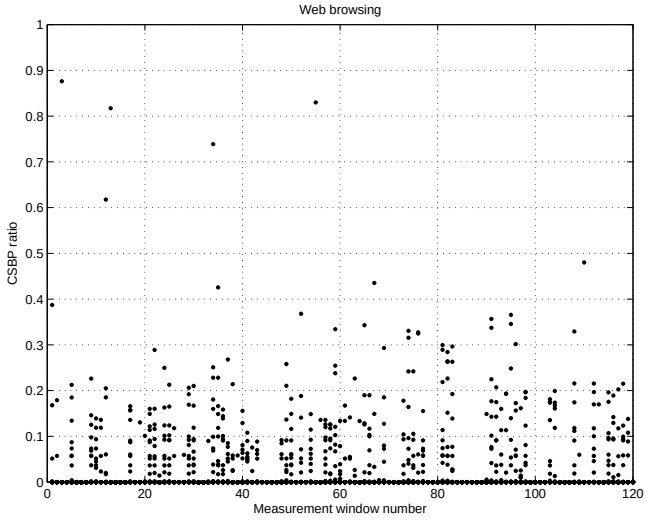


Fig. 14. CSBP estimation result over time obtained from the web browsing traffic pattern (Measurement time window (T) = 10 sec)

the time, and most of the points above the CSBP ratio of 0.2 are due to the strange behavior described in Fig. 15. Thus, we fix the threshold for the CSBP ratio, ζ, to 0.2, and the client IP address is registered in the blacklist, only when that IP address exceeds the threshold for the CSBP ratio, ζ, three times consecutively, to lower the false positives. That is, N_a is set to 3.

Fig. 17 shows the busy period detection results for various types of traffic patterns. In case of web browsing, some points exceed the CSBP threshold of 0.2. However, there is no case where the CSBP ratios exceed the threshold ζ three times consecutively. For the remaining two types of normal traffic patterns, the CSBP threshold is rarely

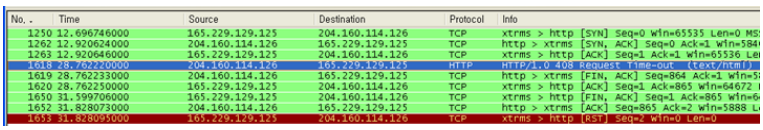


Fig. 15. Strange behavior of the client during the web browsing

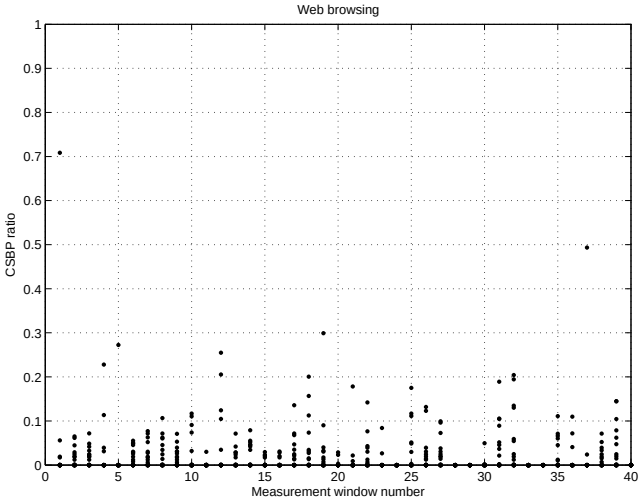
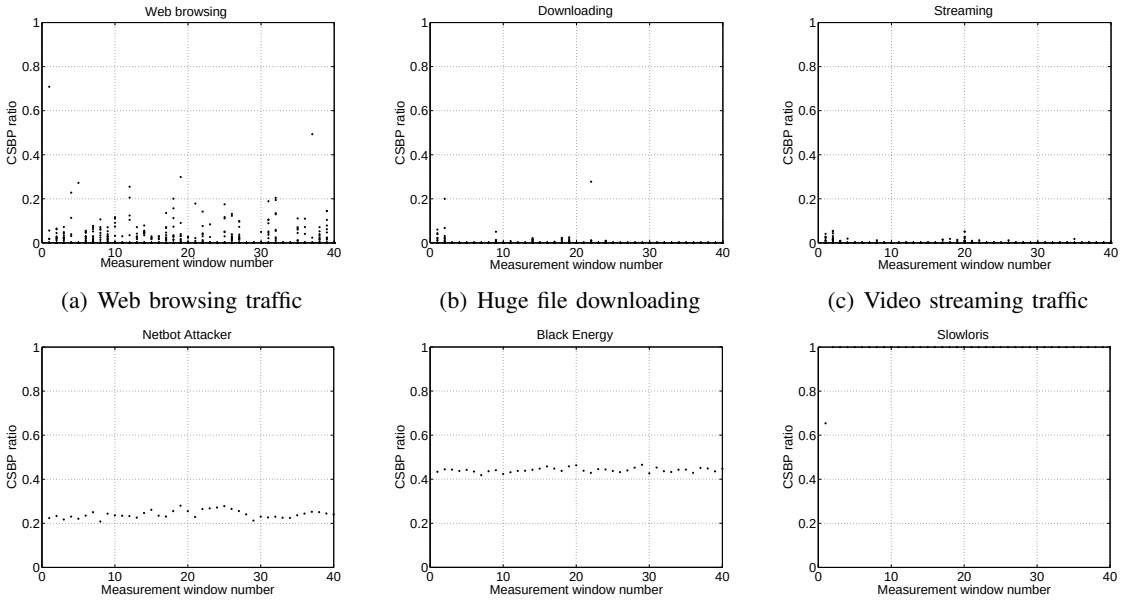


Fig. 16. CSBP estimation result over time obtained from the web browsing traffic pattern (Measurement time window (T) = 30 sec)



(a) Web browsing traffic (b) Huge file downloading (c) Video streaming traffic

(d) Netbot attacker (e) Blackenergy attacker (f) Slowloris attacker

Fig. 17. CSBP estimation results for various types of traffic patterns (Measurement time window (T) = 30 sec)

violated. Conversely, all three types of attack traffic patterns violate the threshold for the CSBP ratio almost all the time. Thus, all the attack nodes will be blacklisted when the number of the CSBP threshold violations reaches N_a, i.e. 3.

When the streaming service is provided by the server, the server usually works continuously for a long time. However, this service itself is not malicious behavior. When we do not include the exception rule of (A) in the busy period detection algorithm of Fig. 9, we find that the CSBP ratios are kept high, close to 100 $\%$, for both the streaming service and huge file downloading cases. However, when the exception rule of (A) is included, the CSBP ratios remains low for both cases, as shown in Fig. 17, even when those services or behaviors occur in the middle of normal web browsing events.

C. Evaluation of the proposed DDoS defense mechanism via simulation

In this subsection, we evaluate the performance of the proposed DoS defense mechanism via OPNET simulation. We investigate the efficacy of the proposed defense mechanism in protecting web servers from low-rate but resource-

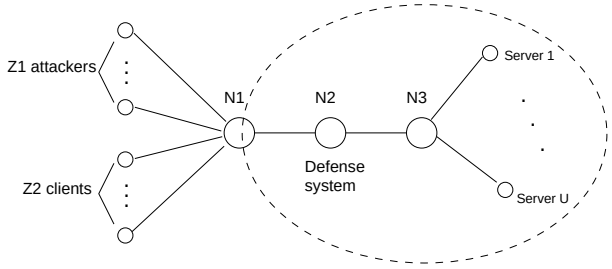


Fig. 18. Network topology for simulation

consuming attacks, by measuring the server response time with and without the defense mechanism.

Fig. 18 shows the network topology for simulation. $N1$ is the edge router located at the boundary of the subnet, where the protected set of servers exists. $N2$ is the node where the defense system is deployed. $N3$ can be a router, a LAN switch, or a load balancer. In our simulation model, the attackers and the normal users are uniformly distributed to U servers, and thus, $N3$ simply performs the forwarding function based on the destination address. $Z1$ and $Z2$ represent the number of the attackers and the normal clients, respectively. All the link rates are fixed to 100 Mbps, and the propagation delay on each hop is fixed to 0.25 msec.

We use the same traffic model for both normal flows and attack flows to investigate the scheme when the attack traffic pattern is not distinguished from the normal traffic pattern. Each normal client or attack node makes only one TCP connection to an internal server, and sends http request packets in a persistent mode without closing the TCP connection. In order to consider the worst case scenario, we let each newly established session persist until the end of the simulation. Session arrival is modeled as a Poisson process, as in [17]. Normal sessions arrive from the beginning of the simulation with an average inter-arrival time of 1 sec. Attack flows arrive after 1000 sec from the start of the simulation, with an average inter-arrival time of 0.5 sec. The http request packets are sent to the server with an exponentially distributed inter-arrival time within each session. The average inter-arrival time of request packets is set to 1 sec for both normal flows and malicious flows. However, we assume that the request packets of the attackers require more processing time at the server. Thus, the processing time of request packets from normal nodes is modeled by an exponential distribution with an average of 5 msec, and that of request packets from malicious nodes is exponentially distributed with an average of 250 msec. The simulation time is 2500 sec, and $Z1$ and $Z2$ are set to 2500 and 3000, respectively.

Fig. 19 shows the simulation result. The server response time is measured at the defense system. The result corresponding to *No defense* is obtained without deploying any defense mechanism at $N2$. Thus, $N2$ works just as a FIFO queue in this case. The result corresponding to *First stage only* is obtained by deploying only the first stage of our proposed mechanism in $N2$. The first stage only scheme is very close to the history-based IP filtering mechanism discussed in [21]. However, the first stage only mechanism is a more enhanced version, since the victim detection rule is discussed in more detail based on the packet sojourn time in the server. The *2-stage mechanism* means the full version of our proposed mechanism, even including the malicious flow detection stage. The DDoS attack starts from 1000 sec. We find that the server response time increases to an unacceptably large value, when there is no defense mechanism. When the first stage only mechanism is deployed, the server response time is reduced significantly compared to the case of no defense. However, it still increases over time. When the first stage only mechanism is used, the average sojourn time increases over 5 times the minimum sojourn time, in about tens of seconds after the arrival of malicious flows. Thus, no more good or bad flows are accepted afterwards, since the ratio of the average sojourn time to the minimum sojourn time remains over 5. However, some attacker nodes that are luckily registered in the whitelist at the initial stage of the DDoS attack, and these nodes offer persistently high load to degrade the performance of the servers. On the other hand, when the full version of the proposed mechanism is deployed in $N2$, the server response time is rather high during a limited period of $[1000, 1400]$, but it is reduced to the normal level afterwards. Differing from the first stage only mechanism, the 2-stage mechanism purifies the whitelist by evicting the malicious flows based on the CSBP ratio. In the simulation, all the malicious flows are blacklisted before 1400 sec has elapsed, and thus, the normal level of server response time resumes for the existing normal flows.

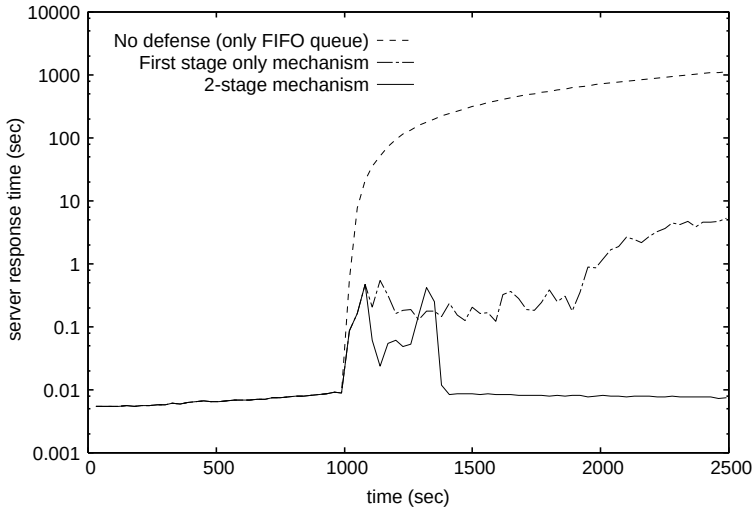


Fig. 19. Comparison of server response time obtained under the proposed defense mechanism to that obtained under other or no defense mechanism

D. Memory Requirement

The number of zombie machines actively participating in a DoS attack is usually less than 60,000, according to recent studies [17, 24–26]. High speed memory, such as SRAM is usually required to process packets arriving at the speed of 1 Gbps or more [27]. However, the size of SRAM is still very limited [27]. Thus, we briefly check whether it is possible to implement our proposed system to be capable of handling 60,000 malicious nodes with a 4.5MByte SRAM.

In our proposed mechanism, the below six tables need to be retained in SRAM.

- connection status table T_1 of the first stage
- Bloom Filter B_1 for the first stage
- Connection status table T_2 for the first stage
- Whitelist B_2 implemented as a Bloom Filter
- per-flow busy period management table (Fig. 7)
- Blacklist implemented as a Bloom Filter

The sizes of T_1, B_1 and T_2 are usually much smaller than for other tables. Thus, we focus on the sizes of the remaining three tables: one whitelist, one blacklist and one per-flow table. The blacklist should be sufficiently large to accommodate at least 60,000 malicious IP addresses. If we intend to accommodate N_b IP addresses in the blacklist, while maintaining a collision probability of less than 1%, then the blacklist size M_3 can be determined by the analysis result of [21] as

$$M_3 = 4 \times \lceil 10.1 \times N_b \rceil \text{ bits.}$$

Similarly, if we want to manage up to N_w normal flows in the whitelist, the whitelist size M_2 is determined as

$$M_2 = 4 \times \lceil 10.1 \times N_w \rceil \text{ bits.}$$

Since the size of each row of the per-flow busy period management table in Fig. 7 is 292 bits, if the number of rows is R_{BP}, then the total size of the three major tables M_T is given as

$$M_T = \lceil 40.4 \times N_b \rceil + \lceil 40.4 \times N_w \rceil + 292 R_{BP} \text{ bits.}$$

If we select the parameters as $N_b = N_w = 220,000$, and $R_{BP} = 60,000$, then $M_T = 4.412$ MBytes. Thus, our proposed system can be implemented with a 4.5MByte SRAM. If we consider the case where the normal user traffic is distributed to many different server farms depending on the user location or client IP address through the content distribution networks, our proposed mechanism can be practical in real applications.

VII. Conclusions and future work

In this paper, we investigated a new two-stage mechanism that can protect web servers from low rate resource-consuming DoS attacks. The propose mechanism consists of two key ideas. The first one is a whitelist-based admission control scheme in the first stage, which protects the servers from a sudden surge of attack flows. We also investigated the condition to detect the victim servers and freeze the whitelist based on the server response time in detail. The second key idea is to detect attack flows based on the concept of a busy period defined for each pair of client and server IP addresses in the second stage. The simulation and experiment results show that the whitelist-based admission control scheme protects the server at the initial stage of DDoS attack, and the busy period-based attack flow detection mechanism distinguishes attack flows from normal flows and effectively filters the IP addresses of the attackers from the whitelist based on the CSBP ratio. Although we focused on protecting http-based web servers in this paper, the proposed approach will be extended to other types of web servers in future study.

References

[1] D. Dagon, G. Gu, C. P. Lee, W. Lee, A Taxonomy of Botnet Structures, in Proc. of Annual Computer Security Applications Conference (ACSAC), Dec. 2007.

[2] T. Peng, C. Leckie, K. Ramamohanarao, Survey of Network-Based Defense Mechanisms Countering the DoS and DDoS Problems, ACM Computing Surveys 39 (1) (2007) 1-42.

[3] S. Kandula, D. Katabi, M. Jacob, A. W. Berger, Botz-4-sale: surviving organized DDoS attacks that mimic flash crowds, in Proc. of NSDI, Boston, MA, 2005.

[4] C. Estan, G. Varghese, New Directions in Traffic Measurement and Accounting, in Proc. of ACM SIGCOMM, Aug. 2002.

[5] R.R. Kompella, S. Singh, G. Varghese, On Scalable Attack Detection in the Network, in Proc. of ACM Internet Measurement Conference (IMC), Oct. 2004.

[6] Jose Nazario, BlackEnergy DDoS Bot Analysis, Technical report, Arbor Networks, Oct. 2007.

[7] Z. Zhu, G. Lu, Y. Chen, Z. J. Fu, P. Roberts, K. Han, Botnet Research Survey, in Proc. of IEEE COMPSAC, pp. 967-972, 2008.

[8] ha.cker.org security lab, Slowloris HTTP DoS, http://ha.ckers.org/slowloris/

[9] J. Mirkovic, P. Reiher, A taxonomy of DDoS attack and DDoS defense mechanisms, SIGCOMM Computer Communication Review 34 (2) (2004) 39-53.

[10] A. Kuzmanovic, E. Knightly, Low-rate TCP-targeted denial of service attacks (the shrew vs. the mice and elephants, in Proc. of ACM SIGCOMM, pp. 75-86, 2003.

[11] G.Macia-Fernandez, J.E.Diaz-Verdejo, P.Garcia-Teodoro, Evaulation of a low-rate DoS attack against application servers, Comput. Security 27 (7) (2009) 335-354.

[12] H. Sun, J. Lui, D. Yau, Defending against low-rate TCP attacks: dynamic detection and protection, in Proc. of 12th IEEE International Conference on Network Protocols (ICNP04), pp. 196-205, 2004.

[13] W. Wei, Y. Dong, D. Lu, G. Jin, H. Lao, A novel mechanism to defend against low-rate denial-of-service attacks, Lecture Notes Comput. Sci. 3975, pp. 261-271, 2006.

[14] A. Shevtekar, K. Anantharam, N. Ansari, Low rate TCP denial-of-service attack detection at edge routers, IEEE Commun. Lett. 9 (4) (2005) 363-365.

[15] G. Macia-Fernandez, R. A. Rodriguez-Gomez, J. E. Diaz-Verdejo, Defense techniques for low-rate DoS attacks against application servers, Computer Networks 54 (15) (2010) 2711-2727.

[16] M Srivatsa, A. Iyengar, J. Yin, Mitigating application-level denial of service attacks on web servers: a client-transparent approach, ACM Transactions on the Web 2 (3) (2008) 15:1-15:49.

[17] S. Ranjan, R. Swaminathan, M. Uysal, A. Nucci, E. Knightly, DDoS-Shield: DDoS-resilient scheduling to counter application layer attacks, IEEE/ACM Transactions on networking 17 (1) (2009) 26-39.

[18] T. Peng, C. Leckie, K. Ramamohanarao, Protecting from Distributed Denial of Service Attack Using History-based IP Filtering, in Proc. of IEEE International Conference on Communications (ICC), pp. 482-486, May 2003.

[19] T. Peng, C. Leckie, K. Ramamohanarao, Proactively Detecting Distributed Denial of Service Attacks Using Source IP Address Monitoring, in Proc. of Networking Conference, pp. 771-782, May 2004.

[20] J. Jung, B. Krishnamurthy, M. Rabinovich, Flash Crowds and Denial of Service Attacks: Characterization and Implication for CDNs and Web Sites, in Proc. of World Wide Web (WWW) Conference, May 2002.

[21] S. Y. Nam, T. Lee, Memory-Efficient IP Filtering for Countering DDoS Attacks, in Proc. of APNOMS, Sep. 2009.

[22] L. Fan, P. Cao, J. Almeida, A.Z. Broder, Summary cache: a scalable wide-area web cache sharing protocol, Technical Report 1361, Univ. of Wisconsin-Madison, Feb. 1998.

[23] H. Takagi, Queueing analysis - volume 1: vacation and priority systems, Part 1, North-Holland, 1991.

[24] P.I.T.A. Committee, Cyber Security: A Crisis of Prioritization, http://www.nitrd.gov/pitac/reports/20050301_cybersecurity/cybersecurity.pdf

[25] Honeynet Project and Research Alliance, Know Your Enemy: Tracking Botnets, http://www.honeynet.org

[26] Linda Dailey Paulson, Hackers strengthen malicious botnets by shrinking them, http://csdl2.computer.org/comp/mags/co/2006/04/r4017.pdf.

[27] N. Weaver, S. Staniford, V. Paxson, Very fast containment of scanning worms, in: Proceedings of the 13th Usenix Security Conference, 2004.