Defending HTTP Web Servers against DDoS Attacks through Busy Period-based Attack Flow Detection

Seung Yeob Nam[1] and Sirojiddin Djuraev[1]
[1] Department of Information and Communication Engineering, Yeungnam University
712-749, Gyeongsan-si, Gyeongsangbuk-do, Republic of Korea
[e-mail: synam@ynu.ac.kr, sirojiddin1987@gmail.com]
*Corresponding author: Seung Yeob Nam

Received February 6, 2014; revised April 9, 2014; revised May 12, 2014; accepted June 18, 2014 ; published July 29, 2014

Abstract

We propose a new Distributed Denial of Service (DDoS) defense mechanism that protects http web servers from application-level DDoS attacks based on the two methodologies: whitelist-based admission control and busy period-based attack flow detection. The attack flow detection mechanism detects attach flows based on the symptom or stress at the server, since it is getting more difficult to identify bad flows only based on the incoming traffic patterns. 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 also need to protect the servers from a sudden surge of attack flows even before the malicious flows are identified by the attack flow detection mechanism. Thus, we use whitelist-based admission control mechanism additionally to control the load on the servers. We evaluate the performance of the proposed scheme via simulation and experiment. The simulation results show that our defense system can mitigate DDoS attacks effectively even under a large number of attack flows, on the order of thousands, and the experiment results show that our defense system deployed on a linux machine is sufficiently lightweight to handle packets arriving at a rate close to the link rate.

Keywords: denial-of-service (DoS) attacks, application layer DoS attack, admission control, busy period, attack flow detection, Bloom filter

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2013R1A1A2012006).

http://dx.doi.org/10.3837/tiis.2014.07.018

1. 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, 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, blackenergy, and slowloris [6][7][8] 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.

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 2. In Section 3, we describe the outline of the proposed two-stage DDoS defense mechanism. In Section 4, we investigate whitelist-based admission control scheme, as the first stage of the proposed defense mechanism. In Section 5, 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 6, the performance of the proposed DDoS defense mechanism is evaluated by OPNET simulation and experiment. Finally, conclusions are presented in Section 7.

2. Related Work

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. In this paper, we focus only on the first type of attack.

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). Several attempts have been made to discriminate attack flows from normal flows based on the anomaly-based detection concept by building a legitimate network traffic model in terms of characteristics of arriving traffic such traffic rate correlation or flow correlation [13][14][15]. 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. [16] 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. [17] 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. In the congestion control step, the server monitors the behavior of each flow to give a higher priority to well-behaving flows. 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 the servers while running on a machine physically separated from the servers.

Ranjan et al. [18] 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. It is very important to set up a reliable normal model for this scheme. However, the normal model construction is usually difficult, and normal model might be susceptible to pollution by the attackers. 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.

3. Outline of the Proposed Defense Mechanism

In this paper, we propose a mechanism that can protect a given web server farm from application-level DoS attacks, especially, the attacks targeting the resources, including CPU, sockets, or memory, of the web server.

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 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 5.

This attack flow detection mechanism is insufficient to protect the servers from the sudden large scale DDoS attacks, since it might take tens of seconds to detect malicious flows based on the client-induced busy period. We use whitelist-based admission control, as an additional measure 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 the packet, if the source IP address is not registered in the whitelist. Thus, this idea constitutes the first stage of the two-stage defense mechanism. The attack flow detection algorithm is applied to the packet afterwards, in the second stage.

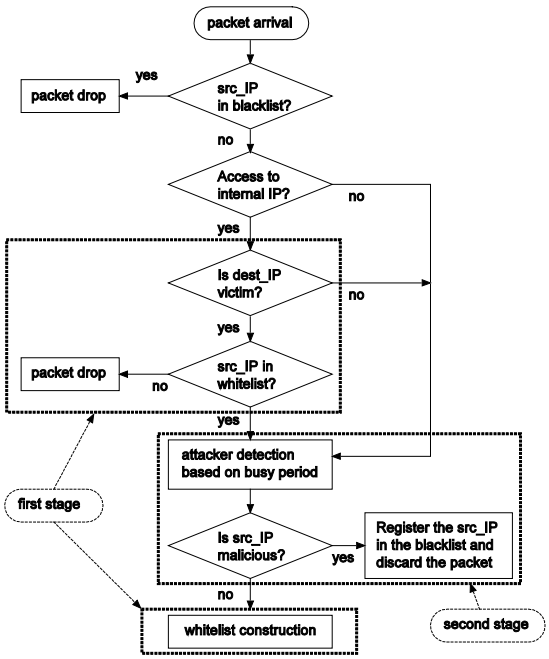


Fig. 1. Outline of the proposed 2-stage DdoS defense mechanism

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 dropped, 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 another threshold, then the server is declared as a victim, and only the IP addresses in the whitelist are allowed to access the server.

We need to note that the whitelist-based admission control scheme is activated only when the DoS attack is detected based on the load on the server. Thus, whitelist-based packet filtering is not used in a normal situation. On the other hand, the busy period-based attack flow detection scheme always works and protects the server with the blacklist.

4. 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.

Whitelist or IP access history-based DDoS defense approach has been investigated by other researchers. According to Jung et al. [19], when the number of clients increases during a DDoS attack, most of them are from new IP addresses that have not been seen before. Based on this observation, Peng et al. [20] tried to manage the normal IP addresses in a whitelist using access history for each IP address, e.g. using the condition on the access days or the number of packets per IP address. 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 Nam et al. [21] to reduce the memory requirement of the system compared to Peng et al.'s scheme. In [21], the whitelist is constructed during a rather short time interval to reduce the possibility of whitelist poisoning. However, even that approach did not resolve the whitelist poisoning issues clearly. Our proposed defense scheme can cope with this whitelist poisoning problem better, since the attack flow detection mechanism can identify the malicious flows even within 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 our 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 offered load. If an internal IP address is declared as a victim, then the whitelist will be used to perform admission control, as shown in **Fig. 1**. We now describe more details on the first and second substages.

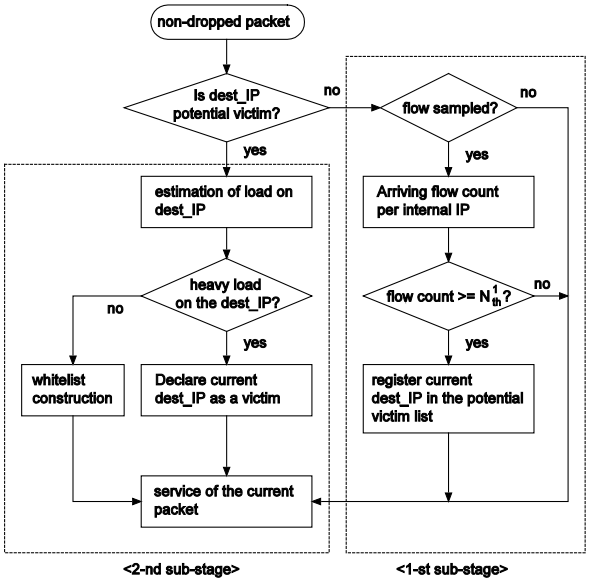


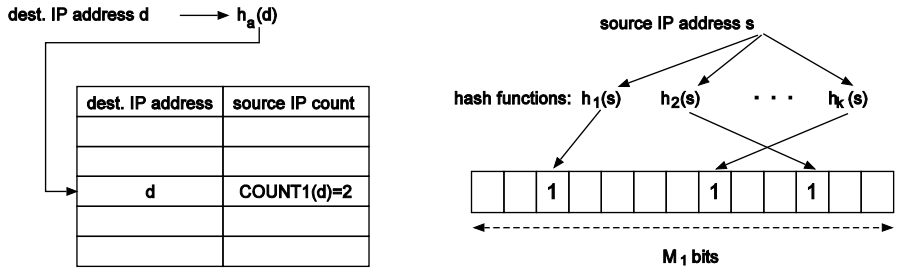
Fig. 2.. Flowchart describing the whitelist construction process

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.

In the first substage, a flow is defined by a pair of source and destination IP addresses, i.e. (SrcIP, DstIP), and we sample flows with the sampling probability p_s [21]. In Fig. 3, connection status table T_1 counts the number of distinct IP addresses accessing a specific internal IP only for the sampled flows, in a given time interval I_1. *COUNT1(d)* counts the number of sampled IP addresses accessing an internal address *d*. Each sampled client 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_{th}^{1}, then the IP address *d* is considered as a potential victim and the whitelist is constructed for *d* in the second substage. Since *COUNT1(d)* counts only sampled source addresses accessing *d*, the total number of IP addresses that accessed *d* is N_{th}^{1} / p_s on average when *COUNT1(d)* = N_{th}^{1}. The value of N_{th}^{1} is set to 3, which is the same as the one used in [21].

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, but new flows are dropped to prevent the failure of the protected server. 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.



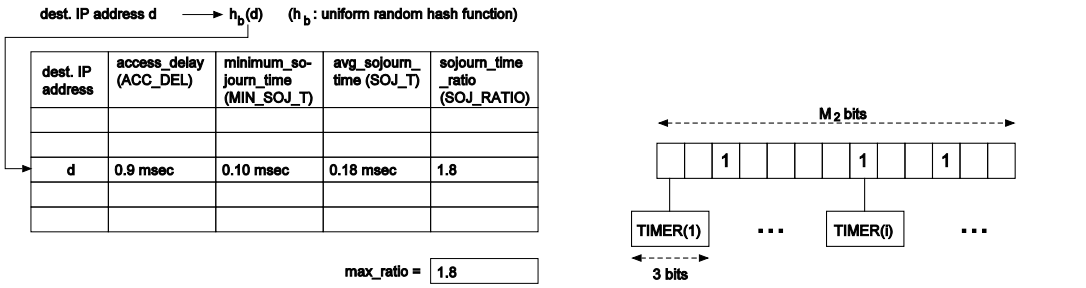
(a) Connection status table (T_1) (b) Bloom Filter (B_1) for source address tracking

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

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. $\rho \approx 0$, $W \approx 1/\mu$. 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 the access delay can be estimated using the technique discussed in Subsection IV.A of [24]. 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 Bloom filter B_2. Whenever a new IP address is registered in B_2, the timers corresponding to the hash positions of the IP address are set to R. The timers are refreshed on each subsequent packet arrival from that IP address, and all timers are decreased by one at the interval of I_T. If any timer reaches zero for a given IP address, then the connection from that IP address is considered to be inactive. 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].

5. Busy Period-based Attack Flow Detection Mechanism

In this section, we discuss a new DoS attack flow detection mechanism. We use a new metric termed client-induced server busy period (CSBP) to detect low-rate application-level DoS attack flows.

We explain the concept of CSBP using an example interaction between a client and a server in **Fig. 5**. 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. 5**, 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. 5**, 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. 5**. 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 estimate the access delay by the mechanism described in Subsection IV.A of [24] and use that value, the effects are likely to be limited, since the CSBP lengths are usually much longer than the access delay.

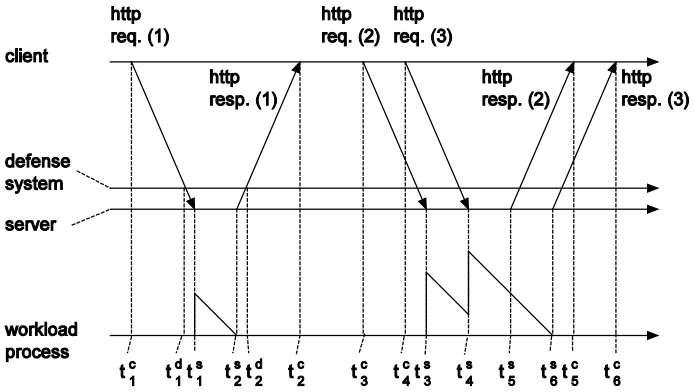


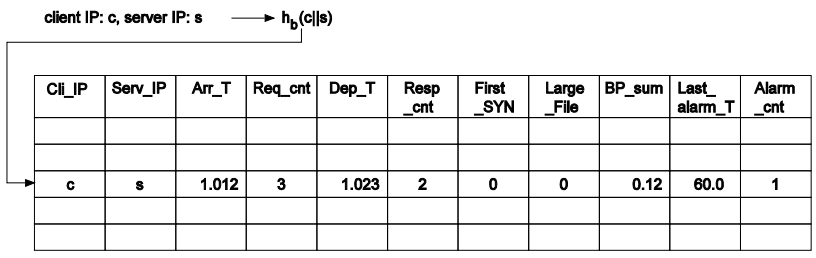
Fig. 5. Detection of client-induced busy period based on transit packet monitoring time

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 needs to maintain per-flow information, as shown in **Fig. 6**. The *Arr_T* field tracks the starting time of each busy period. When a new TCP connection is set up, *Arr_T* is set to 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 *Arr_T* field from the second busy period of the selected TCP session. The *Req_cnt* field counts the number of http request packets observed in the current busy period. The *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 *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 *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 *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 *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 CSBP ratio is calculated as *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 *Alarm_cnt* field counts the number of consecutive alarms. The *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 *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 *Alarm_cnt* will be incremented by one, since the alarms are consecutive.



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. 6. Per-flow busy period management table

The algorithms in **Figs. 7** and **8** 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. 7** is applied to the packets coming from the clients, and the algorithm in **Fig. 8** 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.

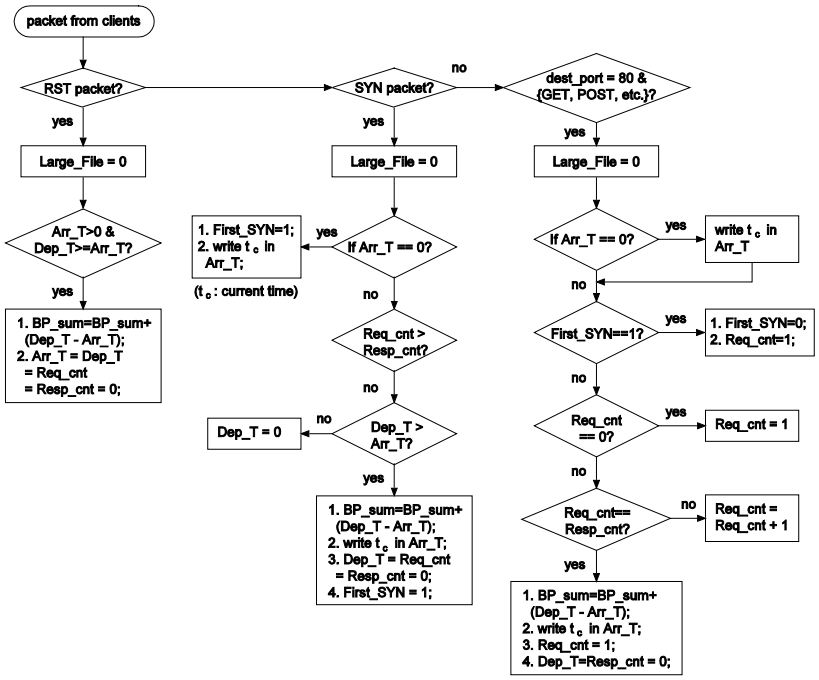


Fig. 7. Busy period detection algorithm applied to the packets coming from clients

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 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, *Resp_cnt*, by 1, only if *Resp_cnt* < *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, *Resp_cnt*, by 1 to minimize CSBP ratio estimation error due to early closing of a busy period, only when *Resp_cnt* = *Req_cnt*.

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. 8**. 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. 8**. The value of L_{th} is fixed to 50,000 Bytes after extensive experiments.

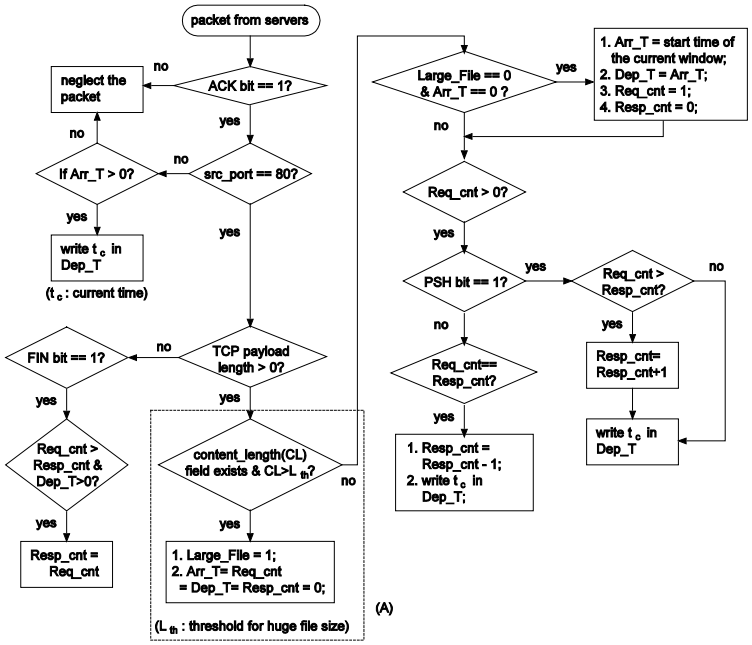


Fig. 8. Busy period detection algorithm applied to the packets outgoing from servers

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. 7**.

6. Numerical Results

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

6.1 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. 7** and **8** of Section 5. Then, we investigate the efficacy of the busy period-based attack flow detection mechanism.

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

- web browsing to cnn web server (normal traffic pattern)
- huge file downloading (normal traffic pattern)
- Blackenergy 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 study. For the attack traffic patterns, we generated http get flooding attacks using Blackenergy attack tool, 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. 9** 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. 9** 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.

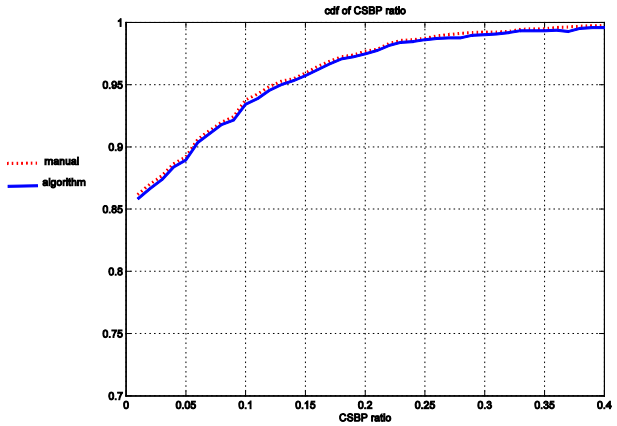


Fig. 9. Accuracy of busy period detection algorithms described in **Figs. 7** and **8**

Fig. 10 shows the estimated CSBP ratio over time, especially for the web browsing traffic pattern. The measurement time window length is 30 seconds. There are several points in each measurement window, as the graphs for multiple web servers related with CNN are superposed in one figure. From the figure, we can observe that the CSBP ratios stay below 0.2 most of the time. We find that CSBP ratios tend to exceed 0.2 as the window length gets smaller. Thus, the measurement window size is fixed to 30 seconds, and the threshold for the CSBP ratio, ζ, is fixed to 0.2 hereafter. 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. 11 shows the busy period detection results for the remaining two types of traffic patterns. For the web browsing case in **Fig. 10**, 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. In **Fig. 11**, the CSBP threshold is rarely violated for the other normal traffic pattern corresponding to the huge file downloading case. Conversely, the attack traffic pattern violates the threshold for the CSBP ratio almost all the time. Thus, the attack nodes will be blacklisted when the number of the CSBP threshold violations reaches N_a, i.e. 3. We find that the false positives are zero for the normal traffic patterns in **Figs. 10** and **11(a)**, and the false negatives are zero for the malicious traffic pattern in **Fig. 11(b)**.

When we do not include the exception rule of (A) in the busy period detection algorithm of **Fig. 8**, we find that the CSBP ratios are kept high, close to 100%, for the huge file downloading case. However, when the exception rule of (A) is included, the CSBP ratios remain low for that case, as shown in **Fig. 11(a)**, even when such a behavior occurs in the middle of normal web browsing events.

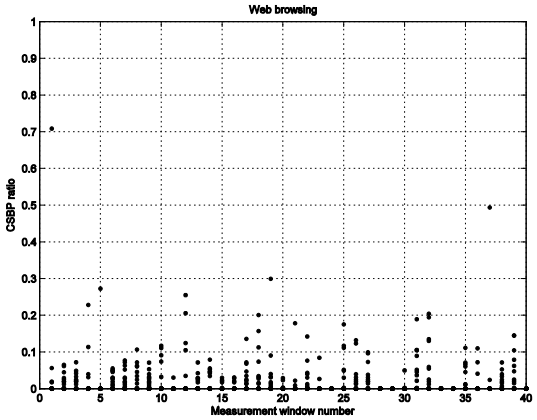
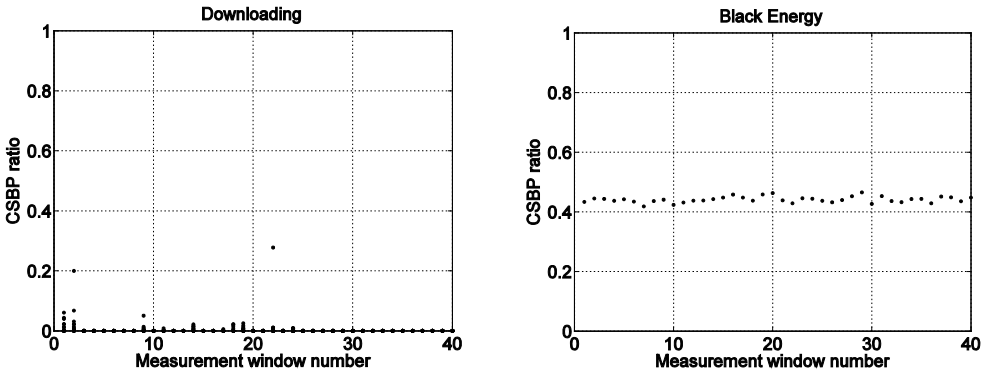


Fig. 10. CSBP estimation result over time obtained from the web browsing traffic pattern (Measurement time window (T) = 30 sec)



(a) Huge file downloading (b) Blackenergy attcker

Fig. 11. CSBP estimation results for different types of traffic patterns (T = 30 sec)

6.2 Evaluation of the proposed DDoS defense mechanism via simulation and experiment

In this subsection, we evaluate the performance of the proposed DoS defense mechanism via

OPNET simulation and experiment. We investigate the efficacy of the proposed defense mechanism in protecting web servers from low-rate but resource-consuming attacks, by measuring the server response time with and without the defense mechanism.

Fig. 12 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 indistinguishable 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 [18]. 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.

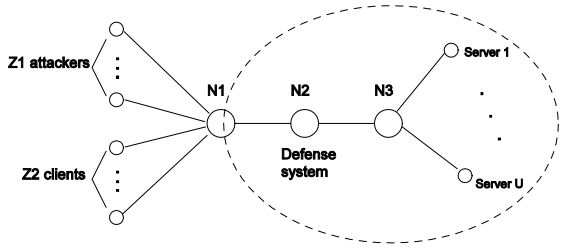


Fig. 12. Network topology for simulation

Fig. 13 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 scheme* means the full version of our proposed mechanism, even including the malicious flow detection stage. The *Peng's scheme* means the IP history-based DDoS defense mechanism based on [20] and [25]. The DDoS attack starts from 1000 sec. The Peng's scheme detects DDoS attack based on the sudden increase of the number of IP addresses, and we find that the attack detection simply based on IP address numbers does not work effectively.

In the Peng's scheme, the attack detection decision is made only at the interval of 10 sec, the default interval duration [25]. On the other hand, early detection is possible in both the *first stage only* and the *2-stage schemes* since the detection decision is made on each packet arrival. Thus, the *first stage only* and *2-stage schemes* perform better than Peng's scheme in terms of the server response time.

However, the server response time still increases over time when the first stage only scheme is used. 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 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.

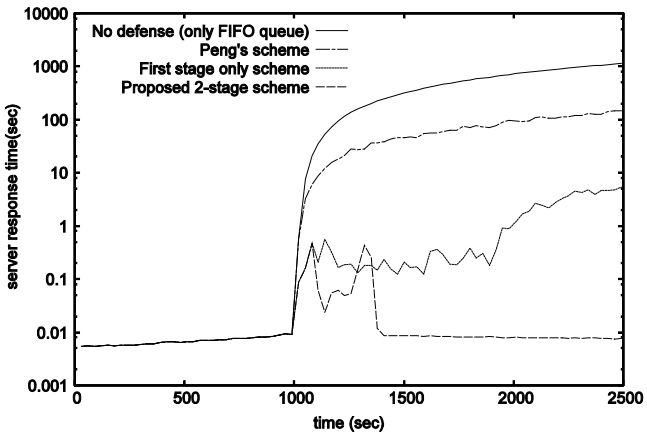


Fig. 13. Comparison of server response time obtained under the proposed defense mechanism to that obtained under other or no defense mechanism

We also tested the performance of the proposed defense system through experiment. The defense system is implemented as a combination of the packet monitoring application and the whitelist and blacklist deployed in the Linux kernel firewall. Our packet monitoring application detects the IP addresses to register in the whitelist or blacklist, and registers the detected IP addresses using *iptables* command [26]. **Fig. 14** shows the detailed commands that we used to add an IP address to the blacklist or whitelist.

```
# iptables  -A FORWARD -s [source IP address] -j DROP
```

(a) Iptables command for adding an IP address to the blacklist

```
# iptables  -A FORWARD -s [source IP address] -j ACCEPT
```

(b) Iptables command for adding an IP address to the whitelist

Fig. 14. Iptables command for adding an IP address to the blacklist or whitelist

The topology is the same as the one in **Fig. 12**. We put only one server machine with a 1.86GHz dual-core CPU, and Z1 and Z2 are set to 7 and 1, respectively. Our defense system is deployed on Node N2, which is a linux machine in our experiment, in **Fig. 12**. Node N2 also works as a router, and we configured the routing table of the linux system statically using "route add" command so that the server can exchange packets with the single normal user and seven attackers. The Blackenergy attack tool has been used to launch http get flooding attacks from the start of the experiment for the attack scenarios.

Table 1 shows the experiment results. When DoS attack is applied, the response time increases by a factor of more than 10 compared to the normal situation. On the other hand, when the defense system is running, all the attackers are detected by the attack flow detection mechanism of Section 5 after 90 seconds. Thus, the server response time resumes to the normal level after 90 seconds in Table 1. This experiment also shows that our defense system running on a linux machine is sufficiently lightweight to handle packets arriving at a rate close to the link rate.

Table 1. Average server response time with or without the proposed defense system

Scenarios	Measurement interval	
	0 ~ 90 sec	90 ~ 180 sec
Normal condition (no attack)	2.62 msec	1.88 msec
DoS attack without defense system	46.2 msec	41.3 msec
DoS attack with defense system	41.1 msec	2.24 msec

6.3 Memory Requirement

The number of zombie machines actively participating in a DoS attack is usually less than 60,000, according to recent studies [18][27][28]. High speed memory, such as SRAM is usually required to process packets arriving at the speed of 1 Gbps or more [29]. However, the size of SRAM is still very limited [29]. 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 substage
- Bloom Filter B_1 for the first substage
- Connection status table T_2 for the second substage
- Whitelist B_2 implemented as a Bloom Filter
- per-flow busy period management table (**Fig. 6**)
- 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. 6** 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.

7. Conclusion

In this paper, we investigated a new two-stage mechanism that can protect web servers from low rate resource-consuming DoS attacks. The proposed mechanism is based on 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 results show that our defense system can mitigate DDoS attacks effectively even under a large number of attack flows, on the order of thousands, and the experiment results show that our defense system deployed on a linux machine is sufficiently lightweight to handle packets arriving at a rate close to the link rate. Thus, performance evaluation 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)*, December 10-14, 2007. Article (CrossRef Link)

[2] T. Peng, C. Leckie, K. Ramamohanarao, "Survey of Network-Based Defense Mechanisms Countering the DoS and DDoS Problems," *ACM Computing Surveys*, vol. 39, no. 1, pp. 1-42, April, 2007. Article (CrossRef Link)

[3] S. Kandula, D. Katabi, M. Jacob, A. W. Berger, "Botz-4-sale: surviving organized DDoS attacks that mimic flash crowds," in *Proc. of Symposium on Networked Systems Design & Implementation (NSDI)*, May 2-4, 2005. Article (CrossRef Link)

[4] C. Estan, G. Varghese, "New Directions in Traffic Measurement and Accounting," in *Proc. of ACM SIGCOMM*, August 19-23, 2002. Article (CrossRef Link)

[5] R.R. Kompella, S. Singh, G. Varghese, "On Scalable Attack Detection in the Network," in *Proc. of ACM Internet Measurement Conference (IMC)*, October 25-27, 2004. Article (CrossRef Link)

[6] Jose Nazario, BlackEnergy DDoS Bot Analysis, Technical report, Arbor Networks, October 2, 2007.

[7] Z. Zhu, G. Lu, Y. Chen, Z. J. Fu, P. Roberts, K. Han, "Botnet Research Survey," in *Proc. of IEEE International Computer Software and Applications Conference (COMPSAC)*, pp. 967-972, July 28-August 1, 2008. Article (CrossRef Link)

[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*, vol. 34, no. 2, pp. 39-53, April, 2004. Article (CrossRef Link)

[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, August 25-29, 2003.

Article (CrossRef Link)
[11] G.Macia-Fernandez, J.E.Diaz-Verdejo, P.Garcia-Teodoro, "Evaluation of a low-rate DoS attack against application servers," *Computers & Security*, vol. 27, no. 7-8, pp. 335-354, December, 2008. Article (CrossRef Link)
[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 (ICNP)*, pp. 196-205, October 5-8, 2004. Article (CrossRef Link)
[13] T. Thapngam, S. Yu, W. Zhou, G. Beliakov, "Discriminating DDoS attack traffic flash crowd through packet arrival patterns," in *Proc. of 1th International Workshop on Security in Computers, Networking and Communications*, April 10-15, 2011. Article (CrossRef Link)
[14] S. Yu, W. Zhou, W. Jia, S. Guo, Y. Xiang, F. Tang, "Discriminating DDoS attacks from flash crowds using flow correlation coefficient," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 6, pp. 1073-1080, June 2012. Article (CrossRef Link)
[15] Z. Tan, A. Jamdagni, X. He, P. Nanda, R. P. Liu, "A system for Denial-of-Service attack detection based on multivariate correlation analysis," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 2, pp. 447-456, February 2014. Article (CrossRef Link)
[16] G. Macia-Fernandez, R. A. Rodriguez-Gomez, J. E. Diaz-Verdejo, "Defense techniques for low-rate DoS attacks against application servers," *Computer Networks*, vol. 54, no. 15, pp. 2711-2727, October 28, 2010. Article (CrossRef Link)
[17] 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*, vol. 2, no. 3, pp. 15:1-15:49, July 2008. Article (CrossRef Link)
[18] 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*, vol. 17, no. 1, pp. 26-39, February, 2009. Article (CrossRef Link)
[19] 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 7-11, 2002. Article (CrossRef Link)
[20] T. Peng, C. Leckie, K. Ramamohanarao, "Protection from Distributed Denial of Service Attack Using History-based IP Filtering," in *Proc. of IEEE International Conference on Communications (ICC)*, pp. 482-486, May 11-15, 2003. Article (CrossRef Link)
[21] S. Y. Nam, T. Lee, "Memory-Efficient IP Filtering for Countering DDoS Attacks," in *Proc. of APNOMS*, September 23-25, 2009. Article (CrossRef Link)
[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, February, 1998.
[23] H. Takagi, Queueing analysis - volume 1: vacation and priority systems, Part 1, North-Holland, 1991.
[24] S. Y. Nam, N. Nazarov, and T. Lee, Defending HTTP Web Servers against DDoS Attacks through Admission Control and Attack Flow Detection, Technical Report, Yeungnam University, March 8, 2012.
[25] 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 9-14, 2004. Article (CrossRef Link)
[26] Red Hat, Inc., 42.9 IPTables, http:// www.centos.org/docs/5/html/Deployment_Guide-en-US/ch-iptables.html
[27] Honeynet Project and Research Alliance, Know Your Enemy: Tracking Botnets, http://www.honeynet.org
[28] Linda Dailey Paulson, Hackers strengthen malicious botnets by shrinking them, http://csdl2.computer.org/comp/mags/co/2006/04/r4017.pdf
[29] N. Weaver, S. Staniford, V. Paxson, "Very fast containment of scanning worms," in *Proc. of the 13th Usenix Security Conference*, pp. 29-44, August 9-13, 2004. Article (CrossRef Link)

Seung Yeob Nam received his BS, MS, and PhD in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejon, Korea, in 1997, 1999, and 2004, respectively. From 2004 to 2006, he was a postdoctoral research fellow at CyLab at Carnegie Mellon University, supported by both CyLab and the Postdoctoral Fellowship Program of the Korea Science & Engineering Foundation (KOSEF). In March 2007, he joined the Department of Information & Communication Engineering, Yeungnam University, Gyeongsan, Korea, where he is currently an associate professor. His research interests include network security, network monitoring, network architecture, wireless networks, etc.

Sirojiddin Djuraev received the BS degrees in Tashkent University of Information Technology (TUIT), Tashkent, Uzbekistan, in 2009. He graduated TUIT with an honor diploma. From 2009 to 2011, he worked as a programmer and a network administrator in Computer center in Kashkadarya, Uzbekistan. In 2011, he joined Computer Network Security Lab of Yeungnam University, where he is currently working toward the Ph.D. degree with a scholarship. His research interests include Network Security, VANET, Embedded Systems and VoIP.