Mitigating HTTP GET Flooding Attacks through Modified NetFPGA Reference Router

Jinghe Jin
Dept. of Information and Communication Engineering
Yeungnam University, Republic of Korea
jinjinghe@ynu.ac.kr

Nazarov Nodir
Dept. of Information and Communication Engineering
Yeungnam University, Republic of Korea
nazarov.nodir@gmail.com

Chaetae Im
IT Infrastructure Protection Division, KISA, Republic of Korea
chtim@kisa.or.kr

Seung Yeob Nam
Dept. of Information and Communication Engineering
Yeungnam University, Republic of Korea
synam@ynu.ac.kr

ABSTRACT
Distributed denial of service (DDoS) attacks are a grave threat to Internet services. These days it is getting more difficult to discriminate legitimate traffic of normal users from DoS attack traffic after emergence of application-level DoS attacks, because the bots performing application-level DoS attacks tend to send seemingly normal traffic. We propose a hardware-based HTTP GET flooding detection and defense system, which can protect a given web server farm by filtering out malicious HTTP requests based on the difference of the behavior between normal browsers and bots. The objective of the proposed DDoS defense system is to provide continued service to legitimate clients even when the normal or attack HTTP traffic arrives at the rate of up to Gbps. We implement the system by modifying the Verilog gateware of the NetFPGA Platform to filter HTTP GET packets, extract and count the requested Uniform Resource Locators (URLs) efficiently using hash tables. The blacklist of attackers' IP addresses is managed and displayed through the corresponding application. We evaluate the performance of the proposed defense system in terms of the throughput and CPU utilization of the defense system and the victim through experiment on a test bed.

Categories and Subject Descriptors

B.6.1 [Logic Design]: Design Styles—sequential circuits, parallel circuits; C.2.0 [Computer-Communication Networks]: General—security and protection; C.2.5 [Computer-Communication Networks]: Local and Wide-Area Networks—ethernet, highspeed, internet

General Terms

Measurement, Design, Security.

This work was supported by the IT R&D program of MKE/KEIT. [2009-S-038-01, The Development of Anti-DDoS Technology].
1-st Asia NetFPGA Developers Workshop, June 13–14, 2010, Daejeon, Korea.

Keywords
DDoS, DoS, NetFPGA, HTTP GET flooding.

1. INTRODUCTION
A denial of service (DoS) attack is an attempt to make a computer resource unavailable to normal users. These days the power of a DoS attack is amplified through botnets [1] launching distributed denial of service (DDoS) attacks by incorporating over thousands of "zombies", i.e. computers taken over through a worm or other automated methods. Although a lot of defense mechanisms have been proposed to counter DDoS attacks [2], it still remains as a difficult problem especially because the attackers tend to mimic normal traffic. For example, a new type of DDoS attack called HTTP flooding attack has emerged recently. In case of HTTP flooding attack, the infected hosts create many threads to send a large amount of requests to the victim's website to disable it [3]. Since these requests have legitimate contents and are sent via normal TCP connections, the server usually serves them as normal requests, and exhausts its resource finally. The attack launched by the worm Mydoom [4] in 2004 is an example of HTTP flooding. Recently, there have been intensive DDoS attacks against major government, organization, news media, and financial company websites in South Korea and US around July 7, 2009 [5]. According to analysis of Cisco Korea, HTTP GET flooding rate was not less than 20 packets per second for each zombie machine [6].

A lot of defense mechanisms have been proposed to counter DDoS attacks [2], and they can be classified into two categories: software-based DDoS defense system and hardware-based DDoS defense system. Software-based DDoS defense systems usually measure flow information with a large storage space and enough memory. However, these kinds of DDoS defense systems may not accommodate gigabit rate traffic especially under the gigabit rate of HTTP flooding, because of the limitations on the kernel buffer or CPU overloading. One of the major advantages of hardware-based DDoS defense systems is that they can process packets at a higher speed than the software-based ones. Since the inter-arrival time can be very short at a high link rate, fast memory such as SRAM is usually required to process those packets arriving at a high speed. But, the size of high speed memory is usually very limited. Since the proposed HTTP GET flooding defense system is implemented on a NetFPGA [7] which also has a very limited

memory size of 4.5MB, we attempt to alleviate the load on the memory by refreshing the memory periodically.

Among the well-known hardware-based DDoS defense systems there are Cisco Guard XT series [8] and RioRey RX series [9] systems. Although these two hardware-based DDoS defense systems can defeat the application-level DDoS attacks such as HTTP GET flooding, they might yield high false positives because of the per-source IP counting[10][11], which means that the number of incoming packets are counted for each IP address. When a normal user accesses a web page by typing the URL manually or following an existing link, the corresponding base html file comes first as a response. Then, the browser on the user's machine usually generates subsequent HTTP request packets to collect other objects, e.g. images, Java applets, or video clips, referenced in the base html file. Occasionally, even a single link click can induce up to hundreds of HTTP request packets destined to different URLs for the embedded objects. But, if all those requests are destined to the same IP address, that normal user might be detected as an attacker by the conventional DDoS defense systems that are based on per-IP counting.

In order to reduce such false positives, we propose a per-URL counting mechanism based on the difference of the behavior between legitimate users and bots. As mentioned above, the browser on the machine of a legitimate user might generate a rather large number of HTTP request packets even with a small number of user actions. But, those request packets are usually spread out over many different URLs because they are generated to collect different objects of a given web page. On the other hand, bots, e.g. NetBot [12], BlackEnergy [13], usually send many HTTP request messages only to a selected target URL frequently. Since the proposed scheme counts the number of HTTP request packets for each URL in a given time interval, it can clearly detect the HTTP GET flooding attack, identify which URL is under attack, and identify who is the attacker while reducing false positives. We implement the per-URL counting scheme on the NetFPGA reference router. NetFPGA is a PCI card that contains a large Xilinx FPGA, 4 Gigabit Ethernet ports, Static RAM (SRAM), Double-Date Rate (DDR2) Dynamic RAM (DRAM) [7].

We evaluate the performance of the proposed defense system in terms of the throughput and CPU utilization of the defense system and the victim through experiment on a test bed.

2. Proposed DDoS detection and defense mechanism

In order to investigate the difference of the behavior between legitimate users and bots, we made a web site which has 10 objects with different URLs and attacked the web site by using the attack tool called NetBot [12], and compared the behavior of a bot with that of a normal user. Figure 1 shows the test result.

From Figure 1, we can easily observe that there is a significant difference between a legitimate user and a bot. Legitimate users send an HTTP request packet for the main web page, and then, the browser generates multiple additional request packets for the referenced objects, usually images. Bots send many HTTP requests only to the target URL differently from the browser.

We also investigated how high HTTP request rate can be achieved by a legitimate user through a simple test. Figure 2 shows the test results.

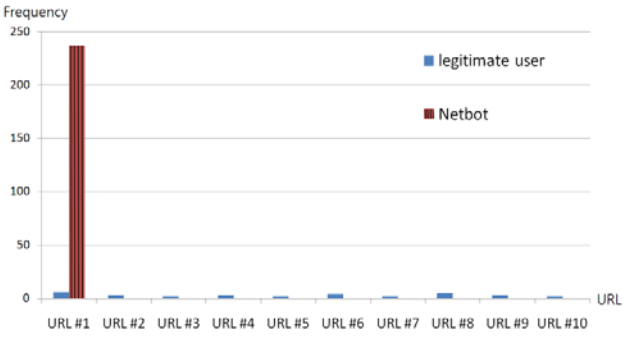

Figure 1. Behavior difference between legitimate users and bots

When the key F5 (web page refresh button) is pressed repeatedly, the HTTP request rate reaches only up to 3 times per second. Since the HTTP request rate is usually not less than 20 packets per second when a zombie machine attacks a web server [6], the HTTP GET flooding attack by a bot can be detected if we count the number of accesses to URLs for each source IP address, and compare it with a pre-determined threshold, e.g. 15.

When we press and hold the key F5, we find that the HTTP request rate reaches up to 30 times per second. But, this is not the behavior expected from the normal users, and thus, we will consider this high HTTP request rate as malicious behavior. We set the threshold for attacker detection to 15 in our system.

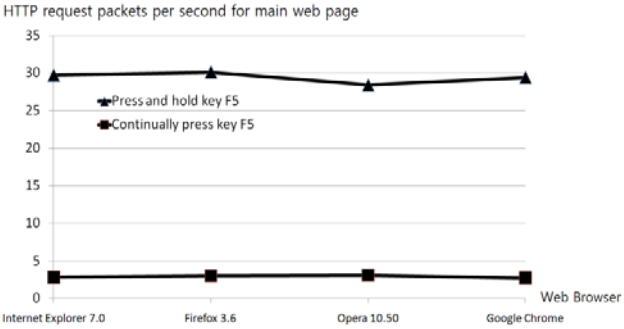

Figure 2. Rate of HTTP request packets generated by human users

Based on above observation, we propose a hardware-based HTTP GET flooding defense system and implement it on the NetFPGA platform. The hardware component is an extended NetFPGA IPv4 reference router that is composed of three parts: HTTP GET filter, URL Extractor, and a hash table-based URL counter. The HTTP GET filter parses HTTP GET packets, and URL Extractor extracts URL information from the HTTP GET packets. For each source IP, the number of accesses to each URL is managed in a hash table. If the number of HTTP request packets from a source IP to a specific URL exceeds a pre-defined threshold, the defense system drops all the packets from the detected source IP and notifies the host of the detected IP so that the corresponding application program can display the blacklist of attackers' IP addresses.

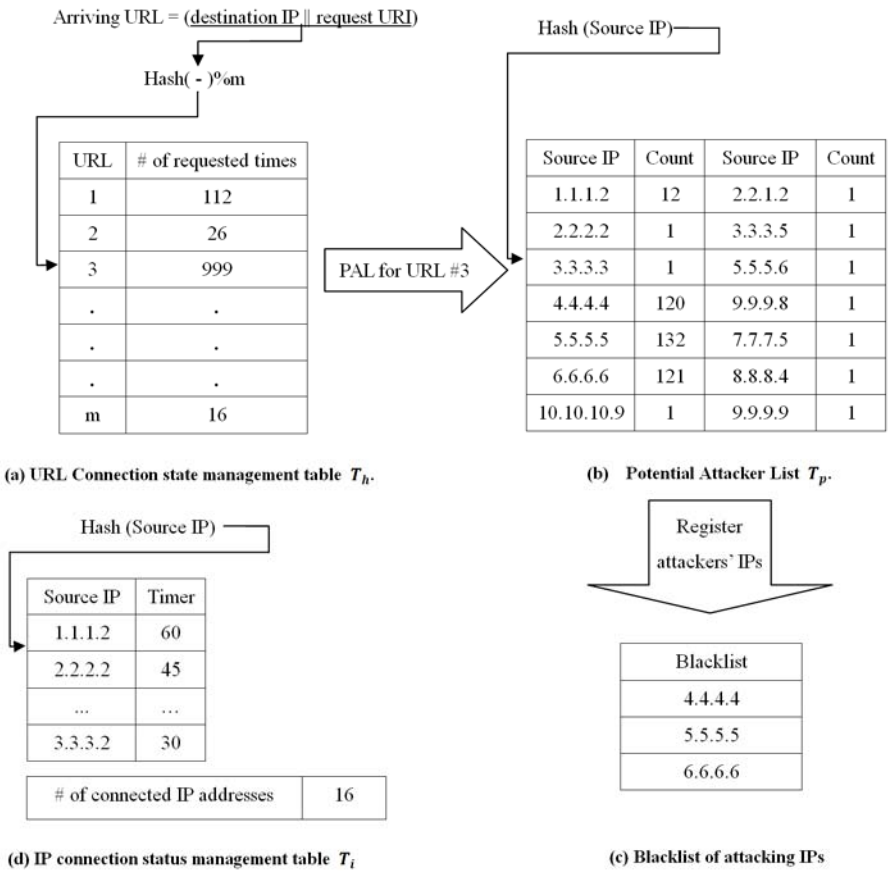

Figure 3. Detection of HTTP GET flooding attacker based on per-URL counting

The proposed per-URL counting mechanism consists of three stages as shown in Figure 3. In the first stage of proposed defense mechanism, we count the number of accesses for each URL by using a hash table to check whether any URL is under attack. In the second stage, we manage a hash table named Potential Attacker List (PAL) to find out the real attacker by counting the number of HTTP request packets from each source IP to the victim URL which is selected in the first stage. In the third stage, the IP address of the real attacker detected in the second stage is registered in the blacklist and any packets from the attacker will be dropped.

In the first stage, the source IP, destination IP and Uniform Resource Identifier (URI) information of arriving HTTP request packets are extracted. In the proposed scheme, the URL is considered as the combination of the destination IP and URI. We manage the number of request packets for each URL in a given interval in the URL connection state management table T_h as shown in Figure 3(a). C_{r_i} denotes the number of the request packets for URL i, C_s denotes the number of IP addresses connected to the server and I_t denotes the measurement time interval. According to the observation result on bot and legitimate user behavior, we set the threshold N_{th}^1 to detect the victim URL as $N_{th}^1 = \alpha \times I_t \times C_s$ ($\alpha = 15$). If $C_{r_i} > N_{th}^1$, then we consider that URL i is under the HTTP GET flooding attack. The number of currently connected IP addresses can be tracked through a separate table T_i of source IP address and timer pair, as shown in Figure 3(d), by refreshing the timer whenever a new packet from the corresponding IP address is observed and deleting the source IP address when the timer expires. The default timer value needs to be determined considering the average flow life time.

If the victim URL is detected in the first stage, then we will create PAL T_p for the victim URL to find the real attacker in the second stage. As shown in Figure 3(b), PAL counts the number of HTTP request packets from each source IP to the victim URL in a given time interval I_t. The threshold N_{th}^2 for the second stage is set as $N_{th}^2 = \alpha \times I_t$. Let S_i denotes the count value of i-th source IP in the table T_p. If $S_i > N_{th}^2$, then the corresponding IP address is detected as an attacker's IP.

In the third stage, the IP address detected in the second stage is registered in the blacklist as shown in Figure 3(c) and all the packets from the registered IP addresses are discarded.

3. Implementation

Our hardware-based HTTP GET flooding defense system is implemented on the NetFPGA platform. It has two main components: measurement sub-system and blacklist display application. The measurement component is implemented on

NetFPGA IPv4 reference router and it consists three parts: HTTP GET filter, URL Extractor, and a hash table-based URL counter. The blacklist display application shows the blacklist of attackers' IP addresses. Figure 4 shows the system diagram.

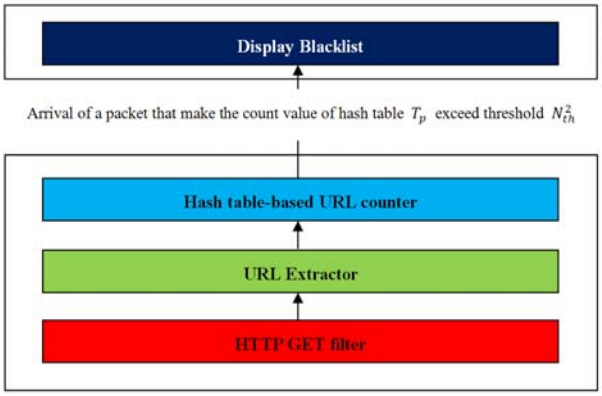

Figure 4. System Diagram

We modify the Output Port Lookup module of the reference router by adding a new sub module called *http_ddos_defense* as shown in Figure 5. The *output port lookup.v* file has been modified to include the definition of the *http_ddos_defense* and its wire connects to the *preprocess_control* and *op_lut_process_sm* sub modules.

The *http_ddos_defense* is a new preprocess block that identifies the HTTP GET packets, extracts URL information from those packets and manages the hash table of the URL counters. The HTTP protocol uses the GET method to send URL requests to web servers. The difference between GET request packets and other http packets is that GET request packets contain the "GET" string at the beginning of the TCP payload. Michael Ciesla *et al.* [14] have already implemented HTTP GET filter to indentify the HTTP GET packets. In order to identify the HTTP GET packets, we first inspect four header fields (refer to Figure 6). First, we check whether the packet is large enough to contain the "GET" string. Second, we need to make sure the transport layer protocol is TCP, because HTTP goes on top of TCP. Third, we check whether the destination port number is 80, a well-known port number for web server. Forth, the TCP header length is checked because it can be different depending on the operating systems. For example, Linux TCP headers include a 12-byte Option field that Windows does not use. Consequently, this changes the location of the "GET" string, and thus, an extra state must be maintained to track whether the current packet is from Windows or Unix OS. As a fifth step, we check whether the "GET" string is included at the beginning of the TCP payload. In the remaining steps, we extract the URL information. After identifying HTTP GET request packets, as a sixth step we register the source IP address in a hash table T_d , which will be explained in more detail later, in order to track the connection status of each source IP address to the URLs of the protected servers. According to observation result on the distribution of URL length, 99.4% URLs are shorter than 150 bytes [15][16]. Thus, in the seventh step we find the end of URI from *in_data*, 64-bit unit of header and payload of an incoming packet, by searching the pattern of '0x20' that indicates the end of URI in *in_data*, which has a fixed width of 64 bits in NetFPGA platform, in parallel. We repeat the seventh step for the 64 bits of the subsequent *in_data* until the end of URI is found. If the URL length exceeds 150 bytes, we just extract and store the prefix of the length of 150 bytes. The initial five steps are already implemented [14], and we added the remaining two steps to extract URL information.

The previous seven steps designed to identify GET packets and extract URL information is implemented by the state machine shown in Figure 7. In order to check the previously mentioned protocol header fields and the existence of the "GET" string at the beginning of the TCP payload, the *http_ddos_defense* sub-module carries out seven stages of elimination process on the left side of Figure 7 to identify a GET packet, and the state machine continuously carries out at least one or at most 19 more stages to extract URI information. In the state of 'URI_X_*i*', the pattern '0x20' is searched for the *i*-th 64 bit unit of TCP payload, where $1 \leq i \leq 19$. Since *i* can reach up to 19, 19 URI extraction stages can cover the URI length of up to 152 byte, i.e. 99.4% URIs according to [15][16]. If the current packet is a non-HTTP GET packet or URI extraction finishes for a HTTP GET packet, the state should be changed to the *WAIT_IP_PROTO_LEN* state waiting for the new packet on the data bus. The *preprocess_control* signals the *http_ddos_defense* when this data is on the bus, and the elimination process is started. If any of the checks fail, the state machine resets to the *WAIT_IP_PROTO_LEN* state, and waits for a new packet.

WORD_7 is an idle state added to avoid unnecessary processing on the 8-th word in Figure 6 without degrading the performance of pipelining between the stages in Figure 7.

As a first prototype, we implement a simplified version of the system described in Section 2. We only manage the per-URL HTTP packet counting table T_d instead of T_h and T_p of Figure 3 as shown in Figure 8. Although T_d is similar to the potential attacker list T_p of Figure 3, the difference is that T_d is shared among different URLs because there is no T_h in the simplified version. The hash table T_d counts the number of HTTP request packets for each pair of a source IP address and a URL measured in time interval I_t. T_d is implemented as a register array managed in the *http_ddos_defense* sub module. *COUNT(u)* counts the number of accesses to the URL *u*. If a packet with a source IP *s*, destination IP *d* and URL *u* increases the value of *COUNT(u)* over the threshold N_{th}^2, then *s* is considered as an attacker's IP and any subsequent packet from the IP address *s* will be dropped by the defense system. In order to alleviate the effect of collision on the hash table T_d, we put two counters at each row to accommodate two different IP addresses mapping to the same row. When a third IP address arrives at a row with no empty space, if there is an IP address that has a counter value of one, then the new IP address replaces the entries of the old IP address with a small counter value. If there is no IP address which has a counter value of one, then the replacement does not occur. The whole hash table T_d is cleared at the interval of I_t to avoid overloading on it.

We can identify the target URL by extracting the URL information from the detected packet. The detected packet signals *op_lut_process_sm* to drop the packet and a duplicated copy of the detected packet is sent to the host system so that the corresponding application can display the list of the detected IP addresses.

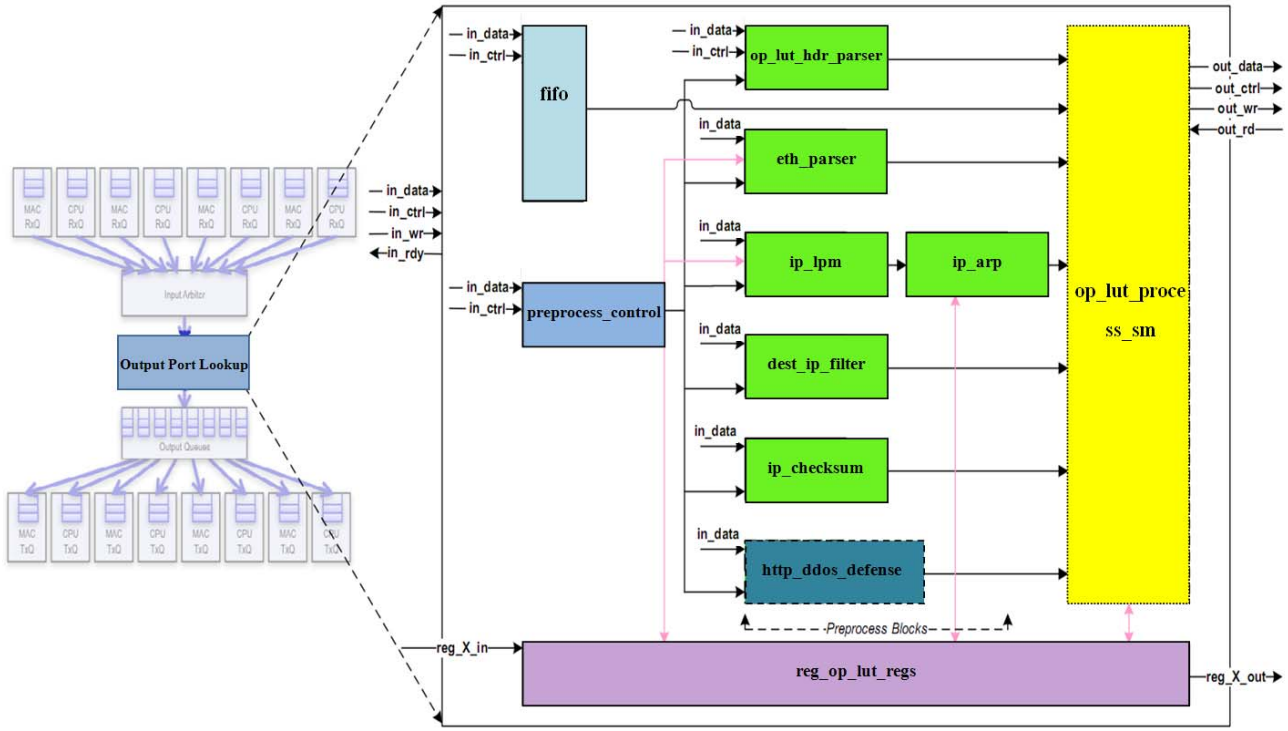

Figure 5. Modification of *op_lut_process_sm* through addition of *http_ddos_defense* sub module

	User Data Path (in_data) Register Bits			
Words	63:48	47:32	31:16	15:0
1	eth da			eth sa
2	eth sa		type	ver, ihl,tos
3	total length	identification	flags, foff	ttl, proto
4	checksum	src ip		dst ip
5	dst ip	src port	dst port	sequence
6	sequence	ack		doff, flags
7	win size	checksum	urgent pointer	options
8	options			
9			**HTTP "GET"**	

Figure 6. NetFPGA word alignment for Unix GET packets

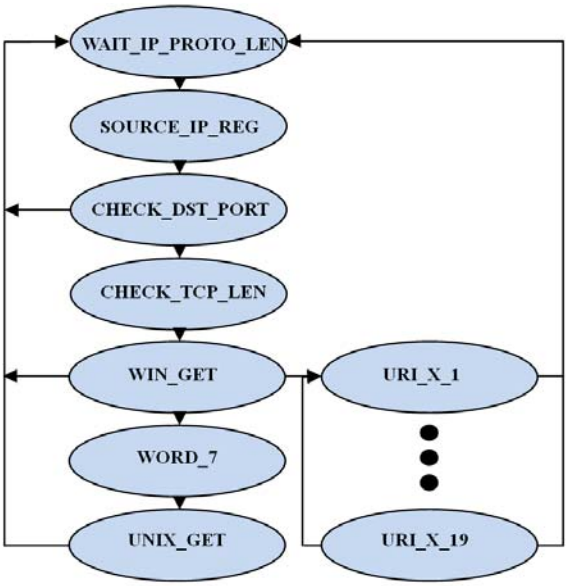

Figure 7. State transition diagram for the mechanism to identify GET Packets and extract URI information

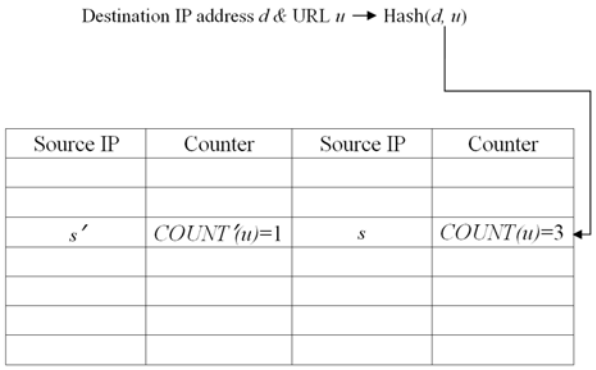

Figure 8. Per-URL HTTP packet counting table T_d

4. Numerical Result

We evaluate the performance of the proposed scheme through experiment on a test bed. To evaluate the performance of the proposed defense system, we measure CPU utilization, false positives and false negatives of the defense system, and also measure the CPU utilization of the victim servers and the utilization of the link connecting the defense system to the servers. We use three kinds of DDoS attack tools: Netbot_Attacker VIP 5.5, BlackEnergy and DoSHTTP 2.5. Each of these attack tools can generate 100Mbps HTTP GET packet stream. Figure 9 shows the network topology of the test bed. There are five PCs accessing five web servers. Each PC can send HTTP GET flooding traffic at the rate of up to 100Mbps. Thus, the maximum aggregate flooding rate is 500Mbps. When only a part of these PCs attack the web server, the remaining PCs become legitimate user machines. Figure 10 compares the CPU utilization of NetFPGA reference_router, the proposed DDoS defense system and Snort (Software IDS tool) [17] under HTTP GET flooding attack. We find that the software-based IDS tool Snort consumes the CPU resource intensively (up to 100%). The CPU utilization of the reference router is not that high since it does not perform DoS detection or defense functionality, but only performs statistics management. In case of our proposed defense system, both HTTP GET packet detection and URL extraction are done in the hardware, and only the packets that induces threshold crossing in T_d are sent to the application which displays the blacklist. Since the application on the host processes less number of packets, the CPU utilization can be reduced. Figure 11 compares the CPU utilization of the victim servers protected by the proposed defense system with that of the victims under no defense mechanism. Figure 12 shows the utilization of the link between NetFPGA node and Switch2 on the topology for the two cases with and without the defense mechanism. From Figure 11 and Figure 12, we find that the proposed defense system protect the given web servers efficiently by maintaining the load on the victim server and the network link low through filtering of attack traffic. Because the DDoS attack tools have a much higher HTTP GET request sending rate, i.e. 120 packets/sec, than the threshold (15 packets/sec), both the false positive and false negative probabilities are measured to be 0. If we let R_{T_d} denote the number of rows in T_d, then to total required memory size is $16 \times R_{T_d}$ bytes. In the considered scenario, we set the parameters as R_{T_d} = 10000, N_{th}^2 = 15, I_t = 1. Thus, the required memory, i.e. SRAM, size is about 160KB.

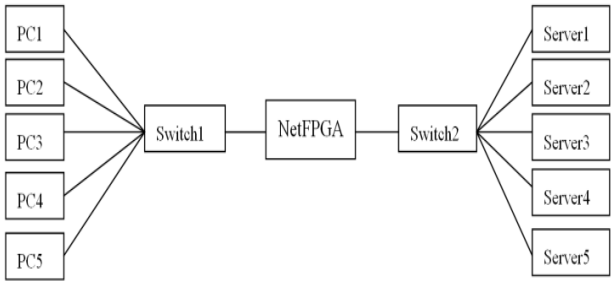

Figure 9. Network Topology

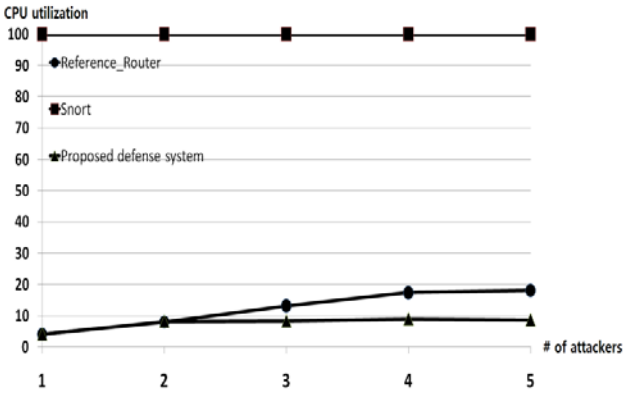

Figure 10. CPU utilization of the NetFPGA reference router

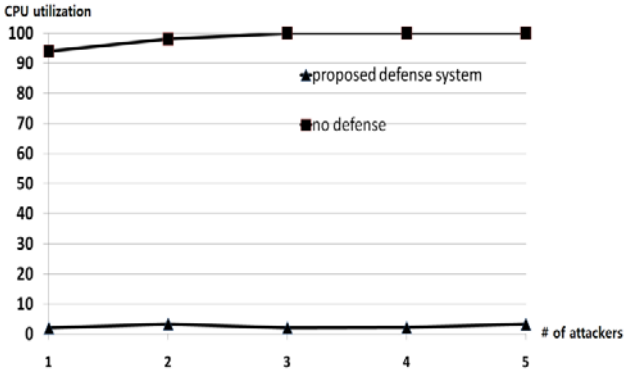

Figure 11. CPU utilization of the victim server

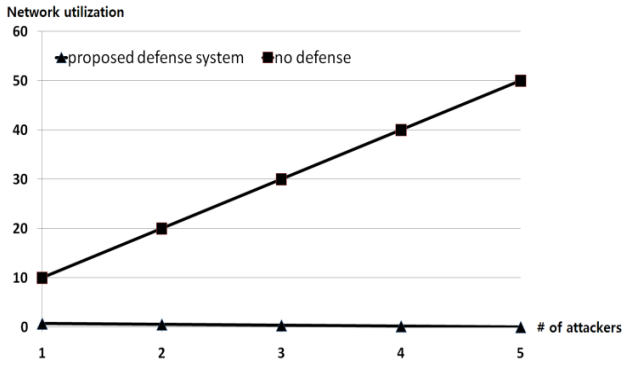

Figure 12. Network utilization of the link between NetFPGA node and Switch2

5. Conclusion

We proposed a hardware-based HTTP GET flooding defense system which protects a given web server farm by discriminating normal users from bots based on the difference of the behavior between browsers and bots. The experiment results show that the proposed defense system can protect web servers efficiently with a reduced load on the CPU of the defense system host under sub-gigabit rate of HTTP GET flooding attacks.

6. REFERENCES

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

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

[3] S. Byers, A. D. Robin and D. Kormann. Defending Against an Internet-Based Attack on the Physical World. *ACM Transactions on Internet Technology*, 4(3): 239-254, August 2004.

[4] Mydoom. http://en.wikipedia.org/wiki/Mydoom

[5] Sionics and Kaientt. 7.7 DDoS: Unknown Secrets and Botnet Counter-Attack. *Power of Community*, 2009.

[6] 7.7 DDoS analysis by Cisco Systems Korea. http://www.cisco.com/web/KR/learning/events/down/July_DDoS_Webseminar.pdf

[7] NetFPGA. http://netfpga.org/foswiki/bin/view/NetFPGA/OneGig/Guide#Walkthrough_the_Reference_Designs

[8] Cisco Guard. http://www.cisco.com/en/US/products/ps5888/index.html

[9] RioRey™ RX Series. http://www.riorey.com/rx-series/

[10] RioRey. Application note – New approach to application level DDoS. *Technical Documents*, July 2009.

[11] Cisco systems. Defeating DDoS Attacks. *White paper*, 2004.

[12] K. S. Han and E. G. Im. A Study on the Analysis of Netbot and Design of Detection Framework. In *Proc. Of Joint Workshop on Information Security*, Aug. 2009.

[13] J. Nazario. BlackEnergy DDoS Bot Analysis. *Arbor Networks*, 2007.

[14] M. Ciesla, V. Sivaraman and A. Seneviratne. URL Extraction on the NetFPGA Reference Router. *In Proc. of NetFPGA Developers Workshop*, 2009.

[15] T. Masao, O. Keizo, A Akiko, I. Haruko, M. Kengo, K. Shin, Y. Hayato and H. Junya. Building a Terabyte-scale Web Data Collection "NW1000G-04" in the NTCIR-5 WEB Task. *NII Technical Report*, 2006.

[16] D. Gomes and M. J. Silva. On URL and content persistence. *Technical Report*, Universidade de Lisboa, 2005.

[17] Snort. http://www.snort.org/