

# Online Identification of Multi-Attribute High-Volume Traffic Aggregates Through Sampling

Yong Tang      Shigang Chen

Department of Computer & Information Science & Engineering

University of Florida, Gainesville, FL 32611, USA

{yt1, sgchen}@cise.ufl.edu

**Abstract**—We propose and implement a set of efficient online algorithms for a router to sample the passing packets and identify multi-attribute high-volume traffic aggregates. Besides the obvious applications in traffic engineering and measurement, we describe its application in defending against certain classes of DoS attacks. Our contribution includes a reservoir sampling algorithm that employs a biased sampling strategy favoring packets from high-volume aggregates. It identifies single-attribute aggregates. Another algorithm is designed to combine single-attribute aggregates into multi-attribute aggregates. We implement the algorithms on a Linux router and demonstrate that the router can effectively filter out malicious packets in stateful DoS attacks.

## I. INTRODUCTION

A traffic aggregate is a collection of packets sharing some common attributes, i.e., they have the same values in one or more header fields. For example, all packets to the same destination address form an aggregate. All SYN packets to the address/port of a server form a more specific aggregate. The traffic volume of this aggregate will surge when the server is under a SYN-flooding attack.

The function of identifying high-volume aggregates provides a powerful tool for network diagnosis and engineering. When a router is overloaded, instead of punishing all packets, the router can selectively drop more packets from the aggressive high-volume aggregates while providing better quality of service to other aggregates that behave normally. In another example, a global traffic assessment based on the traffic aggregates provides valuable information for traffic engineering. Identifying high-volume aggregates is also important in defending against DoS attacks, which is the focus of this paper. In many DoS attacks, the malicious traffic concentrates in certain high-volume aggregates, typically with specific destination addresses, ports, and flags in the packet headers. A router can filter out those high-volume aggregates to reduce the collateral damages on the same routing path. Some anti-DoS mechanisms [1], [2] impose restrictions on the source addresses that the attackers may forge in the malicious packets. In this case, a router can block out attack packets within multi-attribute high-volume aggregates with specific source/destination addresses to protect legitimate packets. In the following, we will describe the existing anti-DoS defense mechanisms, review the prior art in aggregate identification, and discuss our contributions.

The goal of a DoS attack is to exhaust the resources of a server and consequently inhibit it from performing the normal functions. The typical offense strategy is to flood the server with a stream of overwhelming packets. The existing defense mechanisms fall in two broad categories, *host-based* defense at the end systems and *router-based* defense at the routers.

**Host-based mechanisms.** Many anti-DoS defense mechanisms are designed to restrict the attackers from forging arbitrary source addresses. Cookies are typically used to force the attack sources to complete a cryptographic exchange before accessing any resources. SYN-cookie [1] relies on a stateless three-way handshake to defeat the SYN-flooding attack, which attempts to overflow the server's listen queue with excessive SYN packets. The client-puzzle approach [2] requires the clients to solve cryptographic puzzles before establishing the TCP connections. The attackers could be overwhelmed by the server since the more connections the attackers initiate, the more computation they need to perform. Cookies can also be used in conjunction with the http redirection messages to defend against SYN-flooding attacks [3] without the modification of the TCP protocol.

The cookie exchange approach fails under a stateful DoS attack with the attacker resides on the routing path between the server and the forged source addresses. The attacker keeps the state information about each forged connection request, sniffs the network traffic for the responding cookie from the server, and uses the intercepted cookie to complete the connection.

**Router-based mechanisms.** *Ingress filtering* [4] requires the edge routers of stub networks to inspect the outbound packets and drop packets whose source addresses do not belong to the stub networks. Therefore, an attacking host in a stub network can only use its real address or forge source addresses belonging to the same stub network. *SYN-dog* [5] identifies SYN-flooding sources by installing a software agent at the leaf routers of the stub networks. The agent identifies the attack source from its stub network based on the difference between outbound SYN packets and inbound SYN/ACK packets. Another router-based mechanism is called *route-based filtering* [6]. The router will drop the packet received from a link that is not on any routing path between the source and destination of the packet.

IP traceback has been studied extensively in recent years as well [7], [8]. Its purpose is to identify the origins of packets with forged source addresses. The aggregate-based congestion

control (ACC) was proposed to rate-limit attack traffic [9] and progressively push the limit to some neighbor routers to form a dynamic rate-limit tree.

The network-based data streaming methods inspect the passing packets, extract information by using limited storage, and allows queries for the estimated size (rate) [10] of each flow. To make the measurement, the flows have to be well defined in the first place, e.g., bundles of connections to different destinations. This paper solves a different problem. To begin with, the router is not configured with any specification about the aggregates (or flows) to be measured. Instead, it inspects the passing packets and dynamically form (or identify) the boundary of the current high-volume aggregates.

**Our contributions.** The packets from a DoS-attack form high-volume aggregates, which are often identified by the sets of destination address ranges, destination ports, flags, and source address ranges. For a DoS attack against a specific server, the destination of the high-volume aggregate will be a single address. For a DoS attack against an access link, the destination can be multiple address ranges. Many anti-DoS mechanisms restrict the scope of source-address spoofing but does not eliminate it. For example, we have discussed that stateful DoS attacks can forge source addresses behind the attackers even when cookies are used. Both ingress filtering and route-based filtering allow the forgery of source addresses in a narrowed space. Some DoS attacks are launched through zombies whose own addresses are used.

Our goal is for a gateway router to automatically identify the dimensions (attributes) of those high-volume aggregates and to make the range on each dimension as narrow as possible. To achieve this goal, we propose a set of efficient on-line algorithms employing a biased sampling strategy. We design a base algorithm that identifies a list of single-attribute high-volume aggregates by using limited, fixed memory space. The algorithm is expanded for multi-attribute high-volume aggregates. The most related work is MULTOPS [11] which uses a completely different technique to identify single-attribute aggregates restricted to the class boundaries (i.e., class A, B, or C networks). Our algorithms generate multi-attribute aggregates with a finer granularity, allowing arbitrary address ranges. The experiments based on a Linux implementation demonstrate the effectiveness of the algorithms.

## II. IDENTIFICATION OF HIGH-VOLUME AGGREGATES

This section presents the algorithms for identifying multi-attribute high-volume traffic aggregates. First we discuss how to take samples from the streaming traffic passing by a router. We present a naive random sampling algorithm and then describe our reservoir sampling algorithm to deal with single-attribute and multi-attribute aggregates.

### A. Naive Random Sampling

We present the naive random sampling to identify the high-volume aggregates from the incoming packets. Before we describe the algorithm, a special structure called *traffic map* is introduced. A traffic map take samples of the incoming

packets and record a selected attribute (e.g., source address) of the samples. Consider a traffic map  $M$  with size  $|M|$ . The following basic algorithm is used to maintain the randomness of the samples from the incoming packets.

---

#### Algorithm 1 NaiveRandomSampling( $M$ )

---

- 1: Insert the attribute values of the first  $|M|$  packets to  $M$ .
  - 2: **for**  $i = |M| + 1$  to  $\infty$  **do**
  - 3:   wait for a new packet  $e$  to appear
  - 4:   with probability  $|M|/i$ , use the attribute value of  $e$  to replace a randomly selected one in  $M$
  - 5: **end for**
- 

The benefit of the above algorithm is that at any time, the set  $M$  is a true, uniformly random sampling of the traffic. The biggest problem is that the number of attribute values (e.g., number of addresses) stored in the map is limited by  $|M|$ .

During the attack, not all packets are required to identify the high volume aggregates. Instead, we are more concerned with those traffic flows that have a high “concentration” over a limited range of the IP addresses and a high “volume” over a period of time. In other words, a DDoS attack traffic has the following characteristics:

**spatial locality** The high-volume traffic aggregates from the DDoS attacks will always have a higher density (number of packets per address/port) from limited address/port space than the normal, isotropic traffic from the entire Internet.

**temporal locality** The high-volume traffic aggregates from DDoS attacks are more likely to continue in the most recent period of time.

In the following subsections, we describe new algorithms that utilize these localities.

### B. Reservoir Sampling Algorithm and Identification of Single-Attribute Aggregates

The attacking traffic is often a collection of high-volume aggregates on limited address/port space. By identifying and controlling high-volume traffic aggregates at the congested routers, the networks behind those routers are relieved to some extent from the malicious traffic. In this section, we describe a reservoir sampling algorithm. The algorithm automatically converges a master traffic map to the address/port ranges of the high-volume aggregates based on the attribute statistics of the incoming packets. The attribute can be the source address, the destination address, the source port, or the destination port.

Two traffic maps are used. A *temporary traffic map* performs random sampling and record those addresses (or port numbers) that appear most frequently in recent period of time. An entry  $e$  in the temporary traffic map consists of an attribute value  $a$  (e.g., address) and a counter  $c$ , that is,  $e = (a, c)$ . When  $a$  is first sampled, the entry is created with  $c = 1$ . When a packet sampled later carries the attribute value  $a$ ,  $c$  is increased by one. The content of temporary traffic map is periodically merged into a *master traffic map*, which keeps those address/port **ranges** that have a high traffic concentration. Rather

than representing the attribute values as a collection of IP addresses (or port numbers), the master traffic map uses a set of prefixes<sup>1</sup> representing address (or port) ranges. An entry  $e$  in the master traffic map consists of a prefix  $p$  and a counter  $c$ , that is,  $e = (p, c)$ . We denote  $lcp(p_1, p_2)$  as the longest common prefixes of  $p_1$  and  $p_2$ .

Attribute values of randomly sampled packets are first placed in the temporary traffic map. After the temporary traffic map is full, the new attribute value that is sampled will replace the existing map entry that has the smallest counter (i.e., the least number of matching packets in the past). If multiple entries have the smallest counter value, the one whose counter has not been increased for the longest time is replaced.

The entries in the temporary traffic map will be merged to the master traffic map periodically. Let  $b$  be the size of the temporary traffic map  $B$  and  $\lambda$  be the size of the master traffic map  $M$ . An algorithm for merging  $B$  into  $M$  is given below. After inserting  $B$  into  $M$ , the algorithm repeatedly finds two entries that share the longest common prefix and replaces the two entries by the common prefix until the size of  $M$  returns to  $\lambda$ .

---

**Algorithm 2** UpdateMasterTrafficMap( $M, B$ )

---

```

1:  $M \leftarrow M \cup B$ 
2: for  $i = 1$  to  $b$  do
3:    $e_1 = (p_1, c_1), e_2 = (p_2, c_2) \leftarrow$  two entries in  $M$  that
     share the longest common prefix
4:    $M \leftarrow M \cup \{(lcp(p_1, p_2), (c_1 + c_2))\} - \{e_1, e_2\}$ 
5: end for

```

---

The time complexity of Line 3 is  $O(\lambda^2)$  and the time complexity of the whole algorithm is  $O(b\lambda^2)$ . In the following, we present a faster algorithm with time complexity  $O((\lambda + b) \log(\lambda + b))$ .

---

**Algorithm 3** UpdateMasterTrafficMap( $M, B$ )

---

```

1:  $M \leftarrow \text{sort}(M, B)$ 
2: compute common prefixes between adjacent entries in  $M'$ 
3:  $C \leftarrow$  the set of  $b$  longest common prefixes
4: for each  $p \in C$  do
5:   let  $e_1 = (p_1, c_1)$  and  $e_2 = (p_2, c_2)$  be the two entries
     sharing  $p$ 
6:   if  $p_1$  (or  $p_2$ ) is a prefix of  $p_2$  (or  $p_1$ ) then
7:     remove  $e_2$  (or  $e_1$ )
8:   else
9:     replace  $e_1$  and  $e_2$  by  $(p, c_1 + c_2)$  in  $M$ 
10:  end if
11: end for

```

---

During the sorting in Line 1, each prefix in  $M$  is treated as a 32-bit number with the appropriate number of trailing zeros. The time complexity of Line 1 is  $O((\lambda + b) \log(\lambda + b))$ . The complexity of Line 2 is  $O(\lambda + b)$ . Line 3 can be done by sorting

<sup>1</sup>An address prefix specifies the address space of a subnet. A port prefix can be defined similarly.

the common prefixes with a complexity of  $O((\lambda + b) \log(\lambda + b))$ . The complexity of Lines 5-10 is constant. Therefore, the total time complexity of the algorithm is  $O((\lambda + b) \log(\lambda + b))$ .

When defending against a DoS attack, a firewall may be configured to block all packets that match the prefixes in the master traffic map. One problem is that the master traffic map may also contain attribute values from normal packets that happen to be sampled by the temporary traffic map. Once those attribute values (e.g., source addresses of normal users) are inserted into the master traffic map, the subsequent packets carrying them will be blocked. To prevent normal traffic from being blocked indefinitely, if the number of packets blocked by an entry in the master traffic map is below a threshold for a period of time, the entry is removed from the map. With such a bias, the master traffic map is in favor of retaining those attribute values that belong to the high-volume aggregates of the attack traffic due to their higher traffic density.

*C. Identification of Multi-Attribute Aggregates*

One of the most important differences between our method and the ACC method is the ability to identify multi-attribute high-volume aggregates. Multi-attribute aggregates are defined by more than one attribute, comparing with destination address only in the ACC method. It is true that high-volume aggregates in many DoS attacks share the same destination address, but there are cases where aggregates are grouped by both source and destination addresses, and possibly port numbers as well. An example is the traffic aggregates in the stateful DDoS attack. Assume a cookie-based approach is enabled against source-address spoofing. In the stateful DDoS attack, a malicious host forges source addresses within the same LAN or downstream LANs. The cookie information can be obtained by sniffing the returning traffic from the server. In such an attack, the aggregates of offending traffic are decided by the destination IP address of the victim server and the source IP addresses of the sniffed LANs. Our goal is to identify multi-attribute high-volume aggregates that narrowly identify the attack traffic. Consequently, the normal, isotropic traffic is less likely to be effected if we want to restrict those high-volume aggregates.

The following algorithm iteratively merges  $k$  single-attribute traffic maps, which are individually identified by the reservoir sampling algorithm, to an output multi-attribute map  $M$ .

---

**Algorithm 4** MergeTrafficMap( $M, M_1, M_2, \dots, M_k$ )

---

```

1:  $M \leftarrow M_1$ 
2: for  $i = 2$  to  $k$  do
3:   merge  $M$  and  $M_i, M' \leftarrow M \times M_i$ 
4:   from  $n^2$  entries in  $M'$ , select  $n$  entries with the highest
     traffic density.
5:   entries in  $M$  is replaced with these  $n$  entries.
6: end for

```

---

$M_1, M_2, \dots,$  and  $M_k$  are the master traffic maps for attributes  $a_1, a_2, \dots,$  and  $a_k$ , respectively. Assume their sizes are all  $n$ . The algorithm first copies the first traffic map  $M_1$  to

the output map  $M$ . It then performs the cartesian product of  $M$  and  $M_2$  to form a new traffic map  $M'$  with  $n^2$  entries, each having two attributes. Among the  $n^2$  entries of  $M'$ , only the  $n$  most dominant entries, i.e., those entries with the highest *traffic density*, are selected to replace the entries in  $M$ . The above process is repeated by merging one map  $M_i$  to  $M$  at a time and each time increasing the number of attributes in  $M$  by one. How to calculate the *traffic density* will be explained in the following example.

Consider two attributes, the source IP address and the destination IP address.<sup>2</sup> Two traffic maps,  $M_{src}$  and  $M_{dest}$ , are built for the two attributes, respectively, by using the reservoir sampling algorithm. Suppose each map has two entries.  $M_{src}$  includes  $M_{src,1} = (128.100.100.0/24, 30)$  and  $M_{src,2} = (130.100.100.0/24, 20)$ .  $M_{dest}$  includes  $M_{dest,1} = (128.100.100.0/24, 40)$  and  $M_{dest,2} = (130.100.100.0/24, 10)$ . The first element in an entry is an address range, and the second element is a counter that records the number of samples that are taken so far belonging to this address range. The cartesian product of  $M_{src}$  and  $M_{dest}$  will form a new traffic map  $M'$  with four entries:  $(M_{src,1}, M_{dest,1})$ ,  $(M_{src,1}, M_{dest,2})$ ,  $(M_{src,2}, M_{dest,1})$ , and  $(M_{src,2}, M_{dest,2})$ .

The total number of samples in the above example is  $30 + 20 = 40 + 10 = 50$ . The percentage falling in  $(M_{src,1}, M_{dest,1})$  is estimated as  $\frac{30}{50} \cdot \frac{40}{50}$ . There are  $2^8 \cdot 2^8$  different address pairs in  $(M_{src,1}, M_{dest,1})$ . The traffic density per address pair in  $(M_{src,1}, M_{dest,1})$  can be measured by

$$50 \times \frac{30}{50} \cdot \frac{40}{50} \times \frac{1}{2^8 \cdot 2^8} = 24/2^{16}$$

The traffic densities for the other three entries in  $M'$  are  $6/2^{16}$ ,  $16/2^{16}$ , and  $4/2^{16}$ , respectively. Therefore, the two entries with the highest densities,  $(M_{src,1}, M_{dest,1})$  and  $(M_{src,2}, M_{dest,1})$ , are selected for the output traffic map  $M$ .

### III. IMPLEMENTATION

Our proposed prototype is implemented in a 1GHz Pentium III machine utilizing Netfilter in Linux Kernel 2.4.20 currently. Netfilter is a framework allowing packet manipulation inside the kernel. As is shown in Figure 1, our prototype together with the administratia utilities are implemented as several independent components for distributed runtime configurations: a kernel module registered to the hook points of the Netfilter, which performs monitoring of the network packets passing by, a user space Java GUI for configuration, and a character device driver that is responsible for logging and communicating with the GUI. Those network packets passing by will be checked and sampled by the monitoring kernel module. The master and temporary traffic map are established based on the attribute values of the sampled packets. To protect the servers against DDoS attacks, the system can be initiated automatically to drop packets from spaces in master traffic map whenever the network traffic exceeds the maximum allowed rate. It can also

<sup>2</sup>Although IP addresses are used as examples, the attributes are not limited to IP addresses. They can be port numbers as well.

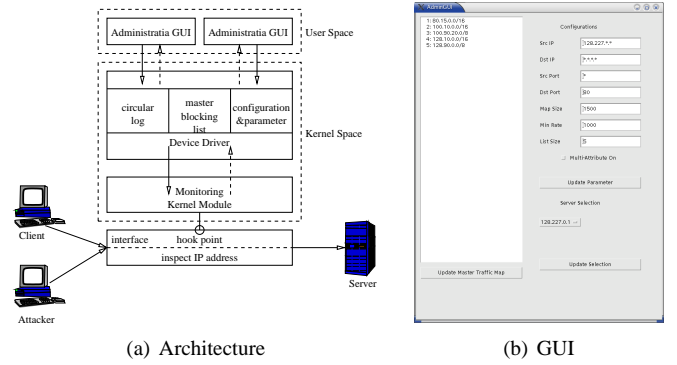


Fig. 1. System Architecture and Administratia GUI

be initiated manually through the Java GUI. A snapshot of the GUI is provided in Figure 1.b.

The proposed system in this paper is deployed on the routers independently at the firewall level. Though we did not discuss the cooperative deployment of the system, it can be more effective with the help of other routers across the whole Internet. One possible cooperative strategy is to work with the pushback mechanism where the downstream router will send a request to the upstream router. The upstream router will build its own traffic map and the restriction on traffic aggregates is more effective here since the total possible address (or port) space has been reduced to a fraction of the  $2^{32}$ . We will not discuss it in detail since it is out of our scope.

### IV. EXPERIMENT

The testbed architecture of our experiments is shown in Figure 1.a. There are two types of generated traffics. One is the normal, isotropic traffic generated by the client machine with randomly selected source addresses, the other is the attacking traffic generated by the attacker machine emulating stateful DDoS attacks. The source addresses of the attacking traffic in our experiments are selected by the following configurations. There is a total of 150 independent attacking sources. 50 of them come from Class B networks, 50 of them come from Class C networks, and the rest addresses come from different single sources. Reserved address spaces (e.g.,  $127.*.*$ ) are excluded. During the experiments, the size of the master traffic map and the temporary traffic map are set as 1500 and 1000 respectively. In our experiments, the normal traffic is generated with with randomly selected source addresses from the whole Internet with a cumulative rate of 40,000Hz and taking up to 80% of the router's capacity.

Different cumulative rates of the attacking traffic have been selected in our experiments to show their influence on our system. Figure 2.a are three typical results from our experiments. In the top plot, the the attacking traffic has a cumulative rate of 40,000Hz, which means 50% of the packets passing by belong to the attacking traffic. In the middle and bottom plots, the ratio has been adjusted to 75% and 90%, respectively. As is shown, most packets in normal traffic are continuously passing by our system without being inserted into the master traffic map, while more and more packets in

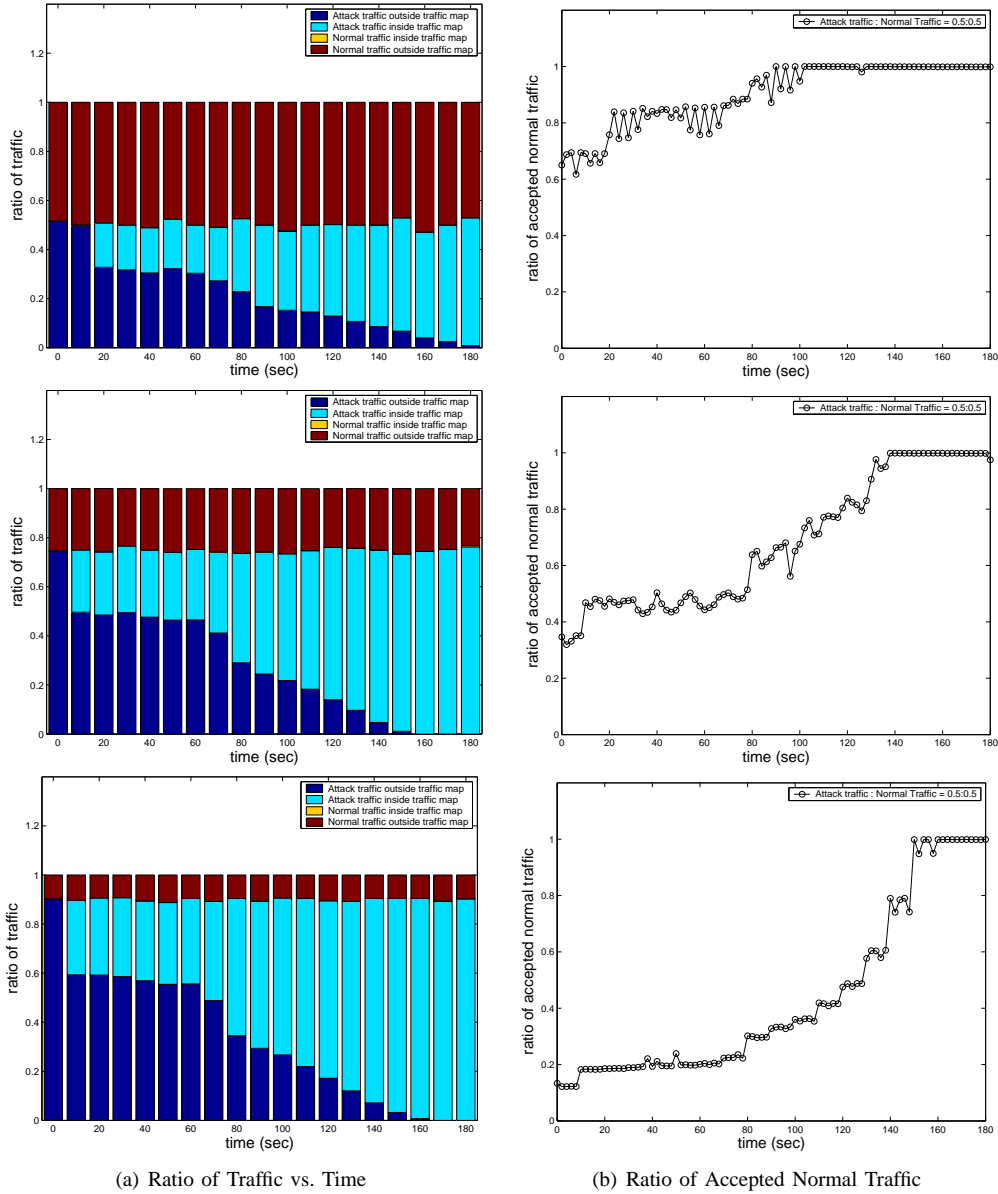


Fig. 2. Experimental Results

attacking traffic are identified and recorded, therefore can be dropped to protect the public server. The complete address space of the attacking traffic is able to be covered by the master traffic map in minutes during the experiments.

Figure 2.b are the results showing the ratio of the normal packets that successfully pass the router during the DDoS attacks. In our experiments, the system is triggered to perform the targeted filtering based on the master traffic map as long as the incoming traffic rate exceeds the capacity of the router and will continue until the rate drops back within the router's capacity. The results in these three plots demonstrate that our system is able to allow most of the normal packets passing by while blocking the majority of the attacking traffic in fairly small amount of time after DDoS happens. By comparing Figure 2.b with Figure 2.a, we conclude that, while increasing

the ratio of the attacking traffic facilitates the establishment of the master traffic map more accurately, the final accepted normal traffic has been decreased.

## V. CONCLUSION

We propose a new system implemented on the router to protect public servers from DDoS attacks by identification and restriction of the high-volume multi-attribute aggregates of the packets passing by. The goal is achieved by utilizing the spatial and temporal localities of the traffic aggregates in DDoS attacks. In the proposed algorithm, only those packets with the most interests are recorded in order to reduce the overhead. Our experiments demonstrate the effectiveness of the proposed system. The system may also be implemented with other approaches, e.g., the pushback mechanism, to

protect the public servers from DDoS attacks more effectively.

#### REFERENCES

- [1] D. J. Bernstein. SYN cookies. [syncookies.html](http://cr.yp.to/syncookies.html). [Online]. Available: <http://cr.yp.to/syncookies.html>
- [2] A. Juels and J. Brainard, "Client puzzles: A cryptographic countermeasure against connection depletion attacks," in *Proc. of NDSS '1999*, Feb. 1999.
- [3] J. Xu and W. Lee, "Sustaining availability of web services under distributed denial of service attacks," *IEEE Trans. Comput.*, vol. 52, no. 2, pp. 195–208, 2003.
- [4] P. Ferguson and D. Senie, "Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing," RFC 2827, May 2000.
- [5] H. Wang, D. Zhang, and K.G.Shin, "SYN-dog: Sniffing SYN flooding sources," in *Proc. of IEEE ICDCS '2002*, July 2002, pp. 421–428.
- [6] K. Park and H. Lee, "On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets," in *Proc. of ACM SIGCOMM '2001*. ACM Press, Aug. 2001, pp. 15–26.
- [7] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Practical network support for IP traceback," in *Proc. of ACM SIGCOMM '2000*. ACM Press, Aug. 2000, pp. 295–306.
- [8] D. X. Song and A. Perrig, "Advanced and authenticated marking schemes for IP traceback," in *Proc. of IEEE INFOCOM '2001*, Apr. 2001, pp. 878–886.
- [9] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker, "Controlling high bandwidth aggregates in the network," *SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 3, pp. 62–73, 2002.
- [10] A. Kumar, J. Xu, J. Wang, O. Spatschek, and L. Li, "Space-code bloom filter for efficient per-flow traffic measurement," in *Proc. of IEEE INFOCOM '2004*, Mar. 2004, pp. 1762–1773.
- [11] T. M. Gil and M. Poletto, "MULTOPS: A data-structure for bandwidth attack detection," in *Proc. of USENIX Security '2001*, Aug. 2001.