Fine-Grained DDoS Detection Scheme Based on Bidirectional Count Sketch

Haiqin Liu, Yan Sun, and Min Sik Kim
School of Electrical Engineering and Computer Science
Washington State University
Pullman, Washington 99164-2752, U.S.A.
Email: {hliu,ysun,msk}@eecs.wsu.edu

***Abstract*—Over the past decade, various intrusion detection and prevention systems have been proposed to detect DDoS attacks and mitigate the caused damage. However, many existing IDS systems still keep per-flow state to detect anomaly, and thus do not scale with link speeds in multi-gigabit networks. In this paper, we present a two-level approach for scalable and accurate DDoS attack detection by exploiting the asymmetry in the attack traffic. In the coarse level, we use a modified count-min sketch (MCS) for fast detection, and in the fine level, we propose a bidirectional count sketch (BCS) to achieve better accuracy. At both detection levels, sketch structures are utilized to ensure the scalability of our scheme. The main advantage of our approach is that it can track the victims of attacks without recording every IP address found in the traffic. Our scheme can save over 90% key storage. Such feature is significant for the detection in the high-speed environment. Experimental results using the real Internet traffic show that our approach is able to quickly detect anomaly events and track those victims with a high level of accuracy.**

I. Introduction

In recent years, distributed denial of service (DDoS) attacks have posed one of the most serious security threats to the Internet [1]. DDoS attacks can do a great damage to the network service. Over the past decades, many intrusion detection systems (IDSs) have been proposed to fight against DDoS attacks. However, those existing schemes usually present a tradeoff between scalability and accuracy. That is to say, finer grained traffic monitoring can ensure the accuracy of detection while does not scale well. For example, two most popular open-source IDSs, Snort and Bro [2], [3], keep per-flow state to detect anomalies. Since the volume of the Internet traffic doubles every year, how to monitor a large amount of traffic in real-time is becoming more crucial in anomaly detection. Although dimensionality reduction [4], [5] may be effective in dealing with such a large data, it usually requires complex operations, and thus is impractical in real-time detection.

Recently, a series of sketch-based approaches have been proposed for anomaly detection [6]–[9]. Sketch [10] is a data structure to store a summary of a large data set for space efficiency. Kompella et al. presented Partial Completion Filters (PCF) by utilizing multiple hash tables for scalable attack detection in high-speed networks [7]. However, it can only tell when an attack happens without providing any hint on where the anomaly occurs; the latter is critical in mitigating the attack at an early stage. Identifying victims is also useful in responding to attacks. For instance, an IDS can generate packet classification rules automatically based on the victim information, so as to minimize future damage. Salem et al. proposed a flooding attack detection method using a count-min sketch (CMS) with multi-channel nonparametric CUSUM (MNP-CUSUM) [8]. The CUSUM technique [11] can accumulate those small offsets during the process to amplify a varying statistical feature so as to improve the detection sensitivity. Although it can detect flooding events effectively, it suffers from the following shortcomings which render it still insufficient to detect general DDoS attacks. First of all, their scheme only takes the high frequency of packets in a flow as the evidence of an anomaly event. However, this feature of traffic alone is not enough to detect an anomaly. For example, a Flash Crowd event, which is caused by a large number of legitimate users simultaneously accessing the same server during historical events, can also result in an outburst of the traffic. Their scheme will lead to a large false positive in such a case. Moreover, their scheme suffers from the scalability problem. Because of the key recovery issue in the original sketch scheme they exploited, their method has to record every incoming destination IP (DIP) for the key recovery later, which makes it unscalable to a large amount of traffic due to the huge memory consumption. Finally, it applies a multi-channel CUSUM algorithm to every bucket in the sketch, which requires heavy computations in high speed neworks.

In this paper, we propose a two-level approach for DDoS detection. We are motivated by the fact that a typical DDoS attack traffic possesses three characteristics: high frequency of incoming packets, asymmetry in interaction patterns, and high diversity of source IP (SIP) addresses. Our modified count-min sketch (MCS), bidirectional count sketch (BCS), and distinct IP addresses estimator are designed precisely for detecting these three characteristics. Although sketch is also used in our work to achieve high scalability, the differences between the previous sketch-based detection approaches and ours lie in the following ways:

a) Memory consumption: Our scheme outperforms previous works in terms of the space requirement. By utilizing the two-level model, most of benign traffic information, which may be considered as a redundancy for IDS systems, does not need to be recorded in the system. Moreover, the traditional key recovery process in sketch [6] requires sketches to record

every input keys, which will consume much space, especially when a large number of DIPs are involved in high-speed networks. By taking advantage of the high diversity of source IP addresses feature, our approach can reveal the victim set without recording every destination IPs as previous works do.

b) Searching time: Our scheme also can achieve faster detection than previous sketch-based approaches in two aspects. Firstly, since most of traffic is benign, the adopted two-level scheme can greatly reduce the search space while the sketch adopted in [8] processes different kinds of traffic equally. Secondly, rather than applying CUSUM to multiple channels of each bucket in the high frequency anomaly detection phase, we utilize a light-weight exponentially weighted moving average (EWMA) technique to achieve the same goal while introducing much less computing overhead.

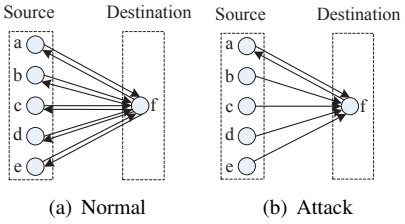
c) Accuracy: By taking the asymmetry feature into accounts, our approach can greatly reduce the false positives by distinguishing between Flash Crowds and DDoS attacks, and thus can improve the overall accuracy.

The remainder of this paper is organized as follows. Section II describes the data structures in our scheme and overviews the architecture. Section III presents the design and implementation in detail. Experimental results are presented in Section IV, and we conclude in Section V.

II. System Description

A. Asymmetry Feature of DDoS Attacks

Fig. 1 demonstrates the "asymmetry" in typical DDoS attacks for web services. We pay attention to the fundamental difference in flow patterns between DDoS attacks and normal traffic. In order to exhaust resources at the server side, an attacker (or more likely a large number of "puppets" in his or her botnet) tries to generate as many requests as possible. When the number of requests exceeds the capacity of the server, we observe fewer responses from the server than requests it receives. We notice that it is not always true that an IP address will serve as both source and destination of traffic, especially when the UDP or ICMP protocol is involved. Thus, in this paper we only refer to the TCP flooding attack by default. Let $N_{\text{forward}}(i)$ denote the number of flows from other nodes to a server i, and $N_{\text{backward}}(i)$ the number of those flows that originate from the server i. By "flow," we mean a group of packets with the same pair of source IP (SIP) and destination IP (DIP) addresses. We do not consider the port information in the flow because we only consider the node-level interactions in our scheme. We define the asymmetry index $AI(i) = |N_{\text{forward}}(i) - N_{\text{backward}}(i)|$ for the server i. We expect that the server under a DDoS attack will exhibit a higher AI value than the one experiencing no attack. For example, Fig. 1(a) shows normal interactions between clients and a server, and $AI(f)$ is 0. On the other hand, in an attack scenario depicted in Fig. 1(b), $AI(f)$ is 4.



(a) Normal (b) Attack

Fig. 1. Flow patterns of normal traffic and DDoS attacks

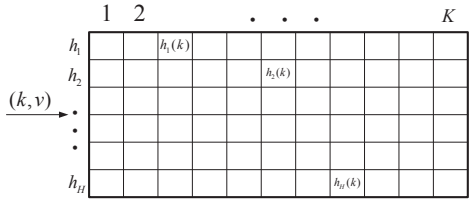


Fig. 2. Illustration of sketch data structure

B. Data Structure

K-ary sketch is a data structure to efficiently and accurately estimate the original signals by aggregating high dimensional data streams into fewer dimensions. As shown in Fig. 2, it consists of H hash tables of size K. A hash function for each row is selected independently and randomly from a set of hash functions. Each data item contains a key k_i and an associated value v_i. When a new item $s_i = (k_i, v_i)$ arrives, its value v_i is added to those buckets corresponding to the key k_i. The $CMS\_Query(key)$ function can return the minimum value among all the buckets corresponding to a specific key. In case of hash collisions, the colliding keys will be listed in the bucket for the key recovery purpose later. The key recovery process [6] can reveal those keys with high frequency in the sketch by looking into the intersection set of high value buckets across the whole sketch. The recovery process is crucial in tracking victims, which will greatly benefit in responding to attacks. Our proposed approach makes two important changes to this original sketch structure: modified count-min sketch (MCS) and bidirectional count sketch (BCS), which will be introduced in the Section II-C.

C. System Architecture

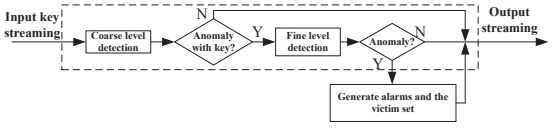


Fig. 3. High-level view of detection process

The overall framework of our detection system is shown in Fig. 3. During each detection period, the related information of every incoming packet is inserted into MCS for the coarse-level detection. We use DIP as the key, and the number of packets that are destined for that IP address as the associated value. MCS only maintains counters for input IP addresses. Compared with the original CMS structure, our MCS structure utilizes space more efficiently because no information on IP addresses themselves is stored in this structure. Besides, unlike

CMS, MCS does not rely on CUSUM; MCS is used only for coarse-grained filtering and a light-weight EWMA technique is applied to each bucket to determine whether to generate an alarm for this bucket or not. Whenever an incoming packet satisfies the condition that every bucket it hashed into has an alarm signal, it will trigger the second stage for finer-grained detection, where a new structure called BCS is used.

In BCS, those suspicious flows detected in the coarse-level detection in both directions are mapped into buckets. While the general sketch structure in which a value in each bucket can increase only, a bucket value in BCS may increase or decrease. We demonstrate how we apply BCS to exploiting the asymmetry of the attack traffic in Section III-B. Once an attack is detected, we use a light-weight distinct IP addresses estimator to pinpoint victims that have the most number of distinct sources, which is a strong indication of DDoS attacks.

III. Scheme Design and Implementation

In this section, we describe the detection processes of both coarse and fine level in details and then explain how the proposed distinct sources estimator works and why it can help to indicate DDoS attacks. We also analyze the space requirement of our scheme.

A. Coarse-level Detection

In our scheme, each bucket in the MCS contains four values $(v_t, v_{t-\Delta t}, v_{\text{backup}}, Flag)$. v_t is the number of packets that are accumulated from $t - \Delta t$ to t, $v_{t-\Delta t}$ is the previous value of v_t, and v_{backup} is the value of v_t right before the alarm occurs (or null if there has been no alarm). $Flag$ is set to 1 whenever the alarm condition is satisfied; otherwise it is set to 0. Here, the definition of alarm conditions depends on the practical deployment, and we will further explain it when we describe Algorithm 1 below. For each incoming record, we update the sketch with $(k_i, 1)$ where k_i is the DIP and 1 represents the number of this incoming record. In our MCS, rather than returning the minimum value of v_t as the original sketch does, the CMS_Query function in MCS returns the minimum value of $Flag$ among all the buckets corresponding to a specific DIP to indicate whether it is under flood attacks. As we can see, the sketch adopted here only requires $O(H \times K)$ cells, which is constant.

The main purpose of MCS is to detect items with abnormal frequency at the coarse level. It is the first stage in the system, which every packet must go through. When a new packet arrives, hash values of H hash functions are computed, and the corresponding buckets are updated; the value in each bucket is incremented by 1. This accumulation process repeats every Δt seconds. The alarm condition is tested for all $H \times K$ buckets periodically. If the alarm condition is satisfied, then the alarm flag associated with the bucket is set to 1. Whenever there is an alarm, the previous v value of the bucket is recorded in the v_{backup} for determining whether the raised alarm is terminated or not.

We use an EWMA technique to decide whether there is an anomaly in each bucket, as shown in Algorithm 1. For each

Algorithm 1: Adjustment procedure of $Flag$

```
for k = 1, h = 1 to K, H do
    if Flag = 0 then
        v_t ← (1 − α)v_{t−Δt} + αv_t ;
        if v_t ≥ (1 + θ)v_{t−Δt} then
            Flag ← 1 ;
            v_backup ← v_{t−Δt} ;
        end
    else
        v_t ← (1 − α)v_backup + αv_t ;
        if v_t < (1 + θ)v_backup then
            Flag ← 0 ;
        end
    end
end
```

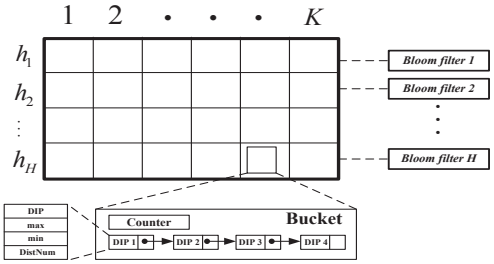


Fig. 4. Illustration of BCS data structure

bucket, if the bucket status is normal, then we estimate v_t with an EWMA parameter α. Whenever $v_t \geq (1 + \theta)\overline{v_{t-\Delta t}}$, which is considered as the satisfaction of the alarm condition, an alarm is raised. θ is the parameter that represents the percentage above the estimated value that can be considered to be an indication of anomalous pattern. The procedure is different after an alarm was raised. In order to estimate when the generated alarm should be terminated, we need to compare the current value with the specific value right before the time that the alarm happened. Such specific value is recorded in v_{backup} before the alarm is generated. Also, rather than using the previous value $\overline{v_{t-\Delta t}}$, we estimate the $\overline{v_t}$ by v_{backup} in order to eliminate the impact of the anomaly on the next following $\overline{v_t}$ series.

We can do the coarse-level detection by querying the minimal value of alarm flag for a specific key. If $CMS\_Query(key) = 1$, then there may be an anomaly associated with the key. However, the coarse-level detection would yield a certain number of false positives. There are two possible reasons for false positives. The first possibility is hash collisions, which can be reduced by carefully selecting hash functions or enlarging the size of the sketch. The second possibility is flash crowds. They can also yield many items with high frequencies in the sketch. Thus, we need to examine traffic further to separate these possibilities from true attacks, which is the goal of our next technique, BCS, which detects anomalies at the finer level.

Algorithm 2: BCS update procedure in the forward direction

```
for h = 1 to H do
    k ← BCS[h].hash(DIP) ;
    if DIP is not in BCS[h][k].list then
        insert DIP into BCS[h][k].list ;
        update BCS[h].BF by DIP|SIP ;
        BCS[h][k].counter ← BCS[h][k].counter + 1 ;
    else
        if DIP|SIP is not in BCS[h].BF then
            update BCS[h].BF by DIP|SIP ;
            BCS[h][k].counter ← BCS[h][k].counter + 1 ;
        end
    end
    DistinctSourcesEstimator(SIP) ;
end
```

B. Fine-level Detection

The objective of the fine-level detection is to find out those DIPs exhibiting high asymmetric communication patterns. A successful DDoS attack employs a large number of zombies to exhaust resources of the target side. However, they are usually unaware of the exact capacity of the server. Therefore, to guarantee to overwhelm the server, an attacker sends as much traffic as allowed, exceeding the server's capacity. This results in highly asymmetric communication patterns between clients and the server, as shown in Fig. 1(b). There are two reasons for such an asymmetry pattern. First, the capacity of the server, namely the victim, to respond is limited while the attacker can keep launching new connections. Second, the SIP addresses of attack traffic are forged, and thus the server has to abort communications. During some time interval ΔT, for a specific IP address, we can detect whether this address serves as both source and destination or not. If yes, then the corresponding flow (the source and destination IP pair containing this IP address) can be considered as a normal flow. Motivated by this observation, we propose the BCS structure to monitor such kind of anomaly with fine granularity.

During one time interval, we use DIP as the key in updating the BCS structure. An illustration of BCS sketch is shown in Fig. 4. All the keys with hash collisions will be stored as a list in the corresponding bucket. Rather than incrementing corresponding H counters by 1 every time a new packet arrives, the counters increase only when the DIP belongs to a new flow. For example, a flow (s_i, d_i), where s_i denotes the SIP address of node i and d_i is the DIP address of node i, will contribute to the corresponding H buckets only once during a single period. On the other hand, (s_j, d_i), which is another flow with the same destination d_i, will contribute another 1 to the buckets that the key d_i is hashed into. Since we do not need to record SIP addresses in the sketch, we employ H bloom filters (BF) with m bits and k_{bf} hash functions as an ancillary structure to estimate whether a specific flow that new packets belong to has been inserted to the BCS structure or not. Algorithm 2 presents the details of how BCS works on the forward direction of traffic. We use $|$ as the string concatenation operator.

Algorithm 3: BCS update procedure of the backward direction

```
for h = 1 to H do
    k ← BCS[h].hash(SIP) ;
    if SIP is in BCS[h][k].list then
        if SIP|DIP is in BCS[h].BF then
            BCS[h][k].counter ← BCS[h][k].counter − 1 ;
        end
    end
end
```

The procedure for the backward traffic is shown in Algorithm 3. Whenever we find a backward flow that can be paired with an existing forward flow in the BCS structure, the corresponding counter decreases. In this way, the counters with anomalous high values indicate an anomaly event caused by asymmetric communication patterns for a specific victim.

C. Distinct Sources Estimator

In order to avoid being detected, attackers may employ a large number of SIP addresses. In such cases, those DIPs that are associated with the largest distinct SIP addresses should be a good candidate for a victim under attack. Thus, how to find the number of distinct SIPs for a victim is crucial in the DDoS defense. Without recording the SIP addresses in the system, which requires too much memory, we need to find a way to estimate this number. For each DIP that is hashed into BCS, we pick a hash function $h\colon \mathcal{N} \rightarrow [0,1]$ which maps every number into $[0,1]$, and then we apply $h(\cdot)$ to all the SIP addresses that are associated with this DIP, and maintains the maximal value max and minimal value min, and then the number of distinct IP addresses, $DistNum$, can be estimated as $\frac{1}{2} \cdot \left(\frac{1}{min} + \frac{1}{1-max}\right)$. If the hash function that we choose is sufficiently random, then the above formula is a sufficiently good estimator for our purpose. In this way, each DIP which has been hashed into the BCS will be associated with a number: $DistNum$. For a specific DIP, this number $DistNum$ can be used as an indicator on how diverse the corresponding SIPs are.

D. Victims Identification

At the end of each time interval, for each row in the BCS, we compute the average counter value $\overline{C[h]}$ and the corresponding mean square deviation $D[h]$. For a specific bucket, whenever its counter value $BCS[h][k].counter$ satisfies the following condition, then it raises an alarm for an anomaly:

$$BCS[h][k].counter - \overline{C[h]} \geq \beta \cdot D[h] \tag{1}$$

where β is an adjustment factor that should be empirically determined. Then, we merge those DIPs that correspond to those anomalous buckets together, and sort them by their $DistNum$. In addition, we eliminate those DIPs that satisfy the condition $BCS\_Query(DIP) < TH_{counter}$ in the merged set, where BCS_Query is similar to the original CMS_Query, which

returns the minimum counter value through all the hashed buckets in the sketch BCS. $TH_{counter}$ is a threshold which can be empirically determined. Finally, those victims can be chosen from the merged DIP set by picking the top few DIPs with the largest $DistNum$ value. Or, we can set a threshold $TH_{DistNum}$ to select those victims that can satisfy $DistNum \geq TH_{DistNum}$ to process the selection of victims.

E. Space Requirements

The primary space consumption of the system is due to the two sketches (MCS and BCS) and H bloom filters that are employed at the finer detection stage. Let L_{mc} denote the length (in bytes) of each bucket in the sketch MCS and L_{bc} represent the same in the sketch BCS. Moreover, each bloom filter will occupy L_{bf} space. Suppose the MCS and BCS have the size $H_{mc} \cdot K_{mc}$ and $H_{bc} \cdot K_{bc}$, respectively, the total memory requirement will be:

$$H_{mc} \cdot K_{mc} \cdot L_{mc} + H_{bc} \cdot K_{bc} \cdot L_{bc} + H_{bc} \cdot L_{bf} \qquad (2)$$

According to our method proposed above, the length L_{bc} will not be a constant value because the length of the linked list varies. In the practical deployment, we are able to limit the maximal number of the nodes in the link list. For example, for a specific link list, we can only keep those top few DIPs that are associated with the highest $DistNum$ value in the list.

IV. Evaluation

We evaluate the performance of the proposed scheme via simulations. We use the trace data from AU [12] as the background traffic. It contains packet traces captured from the link connecting Auckland University and the Internet. This background traffic, which contains both forward and reverse directions, has an average rate of 523 packets per second. We consider the accuracy of victim identification and the amount of memory consumption as two main performance metrics. Unless otherwise noted, the default settings for the parameters we adopted in our experiment are $\Delta t = 5s$ for the periodical sketch construction, $H_{mc} = 32$, $K_{mc} = 1024$ for the sketch size of MCS, $H_{bc} = 128$, $K_{bc} = 5$ for the size of BCS, $L_{bf} = 10000Bytes$ for bloom filters, $\alpha = 0.4$, $\theta = 0.5$ for the coarse level detection, $\Delta T = 5s$, $\beta = 2$ and $TH_{counter} = 10$ for the fine-level detection.

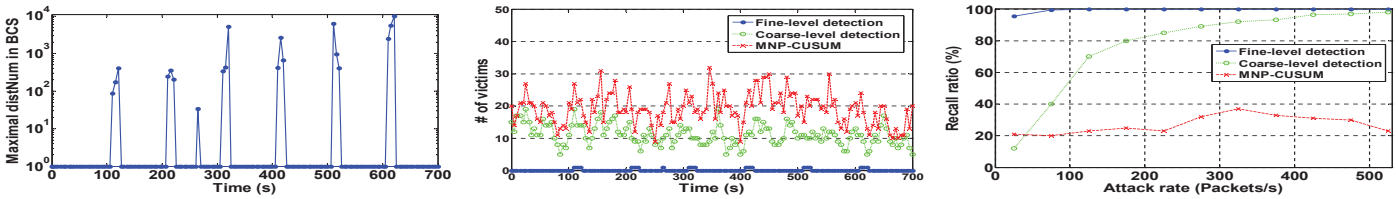
A. Detection Accuracy Evaluation

We generate the flooding traffic using attack tools we developed. The attack rates vary from 25 to 500 packets per second (25, 50, 75, 100, 200, and 500) and the duration of each the attack is 20 seconds. Those attacks are injected at the offset of every 100 seconds. Our goal is to try to gauge the detection sensitivity of our scheme under a large range of attack rates. Fig. 5(a) shows the maximal $DistNum$ series among all the detected victims in the sketch BCS as the time goes. The six spikes (excluding the smallest one) indicate all six DDoS attacks we injected. Even when the attack rate is as low as 25 packets per second, which happens at the offset of 100 seconds, our scheme is still able to identify such low rate attacks while maintaining high accuracy. The maximal $DistNum$ values well reflect the rates of corresponding attacks. After we manually inspected the background traffic, we found that the remaining spike with the lowest value in Fig. 5(a) represents a low rate flooding attack in the original trace. Fig. 5(b) demonstrates the number of victims that are identified by the coarse-level and fine-level detection. On average, the coarse-level detection identify 12 victims per interval. All of those victims experience high rates of requests, which may be caused by flash crowds or DDoS attacks. However, after we further filter those potential victims using the fine-level detection, at most one victim per interval remains, which is the actual attack contained in the traffic. Moreover, the average victim number detected by the MNP-CUSUM approach [8] is around 21, which is even higher than the coarse-level detection of our approach. This is because the original CUSUM technique does not take care of the quick termination after the alarm happens, which results in that too many buckets in the sketch remains high value for a long time. Therefore, it usually causes many false positives. After we modified the original CUSUM techniques by a method for quickly terminating alarm as proposed in [17], the average number is significantly reduced to around 12, which can be due to flash crowds.

We also measure the recall ratio under different attack rates. A recall ratio is the fraction of the true victims in the estimated victims returned by our scheme. The estimated victims identified by the coarse-level detection is the set of all DIPs which satisfy $CMS\_Query(DIP) = 1$. In Fig. 5(c), we can see that the recall ratio of the fine-level detection is very stable; nearly 100% of victims are accurately identified. Even when the attack rate is as low as 25 packets per second, the recall ratio is still over 95%. However, with the coarse-level detection only, the ratio is much lower. It requires more than 350 packets per second (about 66% of the background traffic rate) to achieve the ratio over 95%. Again, due to the alarm termination problem, the MNP-CUSUM technique performs poorly here. Its recall ratio is around 23% on average.

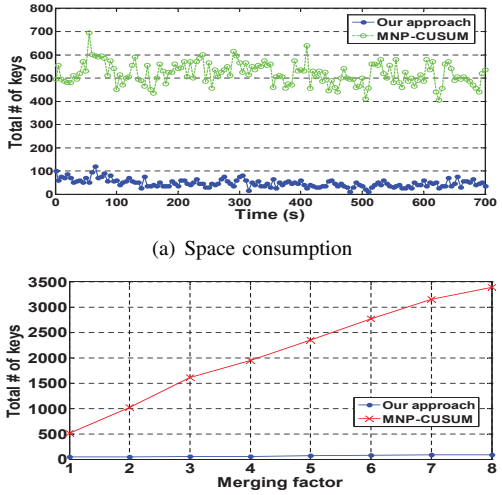
B. Space Consumption

We also sought to measure the memory consumption. Basically, the overall space consumption of sketch-based approaches consists of two different parts. The first part, which can be attributable to the sketch structure itself, takes constant size of small space while the other part, which serves for assisting functions such as the key storage, occupies dynamic size. Since the scalability performance of sketch-based approaches greatly depends on the dynamic part, we compare our approach against [8] by measuring the number of keys that should be stored. The results are shown in Fig. 6(a). During one interval, there are 47 keys that are needed to be stored in our scheme on average while the average number of the keys of MNP-CUSUM approach is around 519. Our approach can save up to 90% keys, which translates to less memory consumption and searching space, when comparing with the previous approach. In order to evaluate the storage



(a) Maximal DistNum value in BCS (b) # of detected victims (c) Recall ratio

Fig. 5. Evaluation of accuracy



(a) Space consumption

(b) Storage scalability

Fig. 6. Evaluation of space consumption

scalability, we shift time stamps of different periods of traces from AU and then merge them together in order to enhance the traffic intensity. We define "Merging factor" as the number of different periods, which can also reflect the intensity of the traffic. Then, we measure the required key storage over various approaches as shown in Fig. 6(b). Our method nearly keeps constant number of keys when the merging factor increases, while the MNP-CUSUM holds a linear-like trend in the same case. That is because our method only record those suspicious DIPs rather than storing every DIPs.

From the total number of keys in the sketches and the default parameter settings, the total memory consumption of our scheme can be estimated using the Eq. 2. The average memory cost is around 563.6 KB, which we consider can be easily accommodated in modern routers.

V. Conclusion

In this paper, we present a fine-grained DDoS detection scheme based on the BCS structure to counter the threat of DDoS attacks. Our approach employs a two-level model to reduce both the size of the search space and time, and further make identification of specific victims possible in the high-speed network environment. We adopt the MCS structure in coarse-level detection to achieve fast detection, and the BCS structure in the fine-level to further guarantee the accuracy. We believe that this approach can accurately identify victims of DDoS attacks with a low memory footprint and give a timely response. Experimental results show that our scheme outperforms previous sketch-based methods with respect to both storage scalability and detection accuracy. The output of this approach can be used to reinforce firewall and IDS systems in real-time.

References

[1] J. Mirkovic and P. Reiher, "A taxonomy of DDoS attack and DDoS defense mechanisms," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 39–53, Apr. 2004.

[2] M. Roesch, "Snort - lightweight intrusion detection for networks," in *Proceedings of the 13th USENIX Conference on System Administration*, Nov. 1999.

[3] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Computer Networks*, vol. 31, no. 23–24, pp. 2435–2463, Dec. 1999.

[4] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," in *Proceedings of ACM SIGCOMM*, Aug. 2004.

[5] ——, "Mining anomalies using traffic feature distributions," in *Proceedings of ACM SIGCOMM*, Aug. 2005.

[6] R. Schweller, Z. Li, Y. Chen, Y. Gao, A. Gupta, Y. Zhang, P. Dinda, M.-Y. Kao, and G. Memik, "Reverse hashing for high-speed network monitoring: Algorithms, evaluation, and applications," in *Proceedings of IEEE INFOCOM*, Apr. 2006.

[7] R. R. Kompella, S. Singh, and G. Varghese, "On scalable attack detection in the network," *IEEE/ACM Transactions on Networking*, vol. 15, no. 1, pp. 14–25, Feb. 2007.

[8] O. Salem, S. Vaton, and A. Gravey, "A scalable, efficient and informative approach for anomaly-based intrusion detection systems: theory and practice," *International Journal of Network Management*, vol. 20, pp. 271–293, Sep. 2010.

[9] S. Ganguly, M. Garofalakis, R. Rastogi, and K. Sabnani, "Streaming algorithms for robust, real-time detection of DDoS attacks," in *Proceedings of the 27th International Conference on Distributed Computing Systems*, Jun. 2007.

[10] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss, "Quicksand: Quick summary and analysis of network data," DIMACS, Tech. Rep. 2011-43, 2001.

[11] M. Basseville and I. V. Nikiforov, *Detection of Abrupt Changes: Theory and Application*. Prentice Hall, 1993.

[12] "Auckland-IV trace data," 2001, http://wand.cs.waikato.ac.nz/wand/wits/auck/4/.

[13] P. Hick, E. Aben, K. Claffy, and J. Polterock, "The CAIDA DDoS Attack 2007 Dataset," http://www.caida.org/data/passive/ddos-20070804_dataset.xml (accessed on 2010-02-28).

[14] S. Sarvotham, R. Riedi, , and R. Baraniuk, "Network traffic analysis and modeling at the connection level," in *Proceedings of Internet Measurement Workshop*, San Francisco, Nov. 2001.

[15] A. Kuzmanovic and E. W. Knightly, "Low-rate TCP-targeted denial of service attacks and counter strategies," *IEEE/ACM Transactions on Networking*, vol. 14, pp. 683–696, Aug. 2006.

[16] J. L. Carter and M. N. Wegman, "Universal classes of hash functions," *Journal of Computer and System Sciences*, vol. 18, no. 2, pp. 143–154, 1979.

[17] H. Liu and M. S. Kim, "Real-time detection of stealthy DDoS attacks using time-series decomposition," in *Proceedings of IEEE International Conference on Communications 2010*, May 2010.