future internet

MDPI

Article

A Review on Hot-IP Finding Methods and Its Application in Early DDoS Target Detection

Xuan Dau Hoang [1,*] and Hong Ky Pham [2]

[1] CyberSecurity Lab, Posts and Telecommunications Institute of Technology, Hanoi 100000, Vietnam
[2] VNPT Software, Hanoi 100000, Vietnam; phamkyit@gmail.com
* Correspondence: dauhx@ptit.edu.vn; Tel.: +84-904-534-390

Academic Editor: Jiankun Hu
Received: 18 September 2016; Accepted: 20 October 2016; Published: 25 October 2016

Abstract: On the high-speed connections of the Internet or computer networks, the IP (Internet Protocol) packet traffic passing through the network is extremely high, and that makes it difficult for network monitoring and attack detection applications. This paper reviews methods to find the high-occurrence-frequency elements in the data stream and applies the most efficient methods to find Hot-IPs that are high-frequency IP addresses of IP packets passing through the network. Fast finding of Hot-IPs in the IP packet stream can be effectively used in early detection of DDoS (Distributed Denial of Service) attack targets and spreading sources of network worms. Research results show that the Count-Min method gives the best overall performance for Hot-IP detection thanks to its low computational complexity, low space requirement and fast processing speed. We also propose an early detection model of DDoS attack targets based on Hot-IP finding, which can be deployed on the target network routers.

Keywords: DDoS attack detection; Count-Min; Count-Sketch; Group Testing

1. Introduction

In network monitoring and attack detection applications, the collection and processing of data packets passing through the network is a big challenge because the number of transferred packets is huge, especially on high-speed connections. In addition, apart from the processing of information items in packet headers, many applications also need to process the packet payloads. This causes significant delays in the processing of data packets and can result in network traffic congestion. Therefore, it is necessary to select suitable packet information items and appropriate processing methods to speed up the processing of data packets.

In an IP (Internet Protocol) packet header, the source IP and destination IP are important information for the transfer of the packet from the source host to the target host. In DDoS (Distributed Denial of Service) attacks, a huge number of packets of fake requests are sent to the target host to exhaust system resources or to flood the network connection. Since the destination IP of these packets is the attacked target host, it is possible to detect a DDoS attack in the early stage by monitoring high-frequency destination IPs in the target network router. Similarly, the monitoring can be done in ISP (Internet Service Provider) routers to track source IPs of hosts that originate a large number of packets. These IPs may be the addresses of hosts infected with network worms and these worms are scanning the network for the next target hosts. IP addresses with a high occurrence frequency in the IP packet stream are called Hot-IPs. Therefore, the problem of target detection of DDoS attacks or the detection of network worm sources can be solved by monitoring the IP packet stream transferred through the network to find Hot-IPs [1–3].

On a certain network connection, an IP packet stream is a sequence of IP packets, which can be represented as $S = \{a_1, a_2, \ldots, a_M\}$, where there are M packets with N unique IP addresses. Suppose f_i

to be the occurrence frequency of the packet with IP address s_i in S, and then we have $f_i = \{j \mid s_j = s_i\}$, where $1 \leq i \leq N$ and $1 \leq j \leq M$. We also have $f_1 + f_2 + \ldots + f_N = M$ and, given the occurrence frequency threshold φ, the Hot-IP = $\{s_i \mid f_i \geq \varphi\}$ [1,2].

The problem of finding Hot-IPs in the IP packet stream can be solved using algorithms for finding elements with a high occurrence frequency in the data stream. There are a number of such algorithms, including Majority, Frequent, Lossy Counting, Space Saving, Count-Sketch, Count-Min and Group Testing [4–12]. Section 2 of this paper will briefly discuss these algorithms.

Huynh et al. [1,2] proposed to use the non-adaptive Group Testing method for fast finding of Hot-IPs in the IP packet stream and to apply the results in the detection of DDoS attacks and network worm spreading sources. Since the computational complexity of the Group Testing method ($O(tN)$, where N is the number of unique IP addresses and t is the number of tests) is relatively high, it is not efficient for the processing of the IP packet stream in heavy traffic [1]. To solve this issue, they use the Reed Solomon code concatenation method to construct the d-disjunct matrix m [1,2]. Thus, the storage space of matrix m is significantly reduced and the computational complexity is also reduced, which is equivalent to the polynomial time.

In this paper, we evaluate methods to find high-occurrence-frequency elements in the data stream and apply them to find Hot-IPs in the IP packet stream passing through the network. Based on the computational and space complexity evaluation results of each method, the most efficient method is selected for use in the target detection model of DDoS attacks.

The rest of the paper is organized as follows: Section 2 reviews and compares methods for finding high-occurrence-frequency elements in the data stream and applies them in finding Hot-IPs in the IP packet stream. Section 3 presents the proposed target detection model of DDoS attacks and Section 4 is our conclusion.

2. Hot-IP Finding Methods

2.1. Methods for Finding High-Occurrence-Frequency Elements

2.1.1. Majority

The Majority algorithm was proposed by Boyer-Moore in 1982 [4]. A common element is defined as the element that has the number of occurrences greater than half of the total number of elements in the data stream. The algorithm can be summarized as follows: Given the data stream $P = \{a_1, a_2, \ldots, a_m\}$ and N is the number of unique elements. Assume f to be the frequency vector of N unique elements, $f = \{f_1, f_2, \ldots, f_N\}$ and $f_1 + f_2 +, \ldots + f_N = m$. A high-occurrence-frequency element j is determined by $j \mid f_j > m/2$. The Majority algorithm consists of two stages, including (1) finding elements with a high occurrence frequency if they exist, and (2) checking if these elements are really common elements based on their occurrence frequencies.

2.1.2. Frequent

The Frequent algorithm was proposed by Misra and Gries in 1982 [8]. The algorithm attempts to find all elements that have an occurrence frequency greater than m/k, where m is the total number of elements in the data stream and k is the number of high-occurrence-frequency elements to find. It uses k pairs of elements and counters to monitor these k elements. Initially the k pairs are empty. The main idea of the algorithm can be described as follows: for each element in the data stream, if the element exists in k pairs, increase the element's counter by 1. If the element does not exist in k pairs and the number of pairs is less than k, add the new element into the k pairs and assign 1 to its counter. Otherwise, decrease all counters and remove elements from k pairs if their counters are 0. At completion, the algorithm can find most k elements that have an occurrence frequency greater than m/k.

2.1.3. Lossy Counting

The Lossy Counting algorithm was proposed by Manku and Motwani in 2002 [7]. The algorithm uses a data structure that has three attributes, including the element, L (lower bound) and H (higher bound) to process the data stream. Assume that m is the total number of elements in the data stream and k is the number of high-occurrence-frequency elements to find. Set $P = H - L$. For each ith element in the data stream, if the element was stored in the data structures, set $L \leftarrow L + 1$; otherwise, create a structure for the element and set $L = 1$ and $P = i/k$. If $H < i/k$, the ith element is removed from data structures. At completion, the algorithm can find k elements that have an occurrence frequency greater than m/k.

2.1.4. Space Saving

The Space Saving algorithm was proposed by Metwally et al. in 2005 [11]. The algorithm uses a data structure of k pairs, in which each pair consists of an element and its counter. Initially, the first k unique elements are assigned to k pairs and their counters are assigned to their corresponding numbers of occurrences. For the next element from the data stream, if the element is found in k pairs, increase its counter by 1. Otherwise, replace the smallest counter element with the new element and increase its counter by 1. At completion, the algorithm can find k elements that have an occurrence frequency descending from high to low.

2.1.5. Count-Sketch

The Count-Sketch algorithm was proposed by Charikar et al. for the first time in 2002 and the improved version of the algorithm was introduced in 2004 [12]. Sketch is a terminology used to represent a data structure, or a linear mapping of the input frequency vector. Count-Sketch is used to find k elements that have a high occurrence frequency in the data stream. Sketch data structure is compact and the algorithm's processing speed is fast.

Count-Sketch uses two input variables, where ε is the error rate and δ is the error probability. To store counter variables in the processing, Count-Sketch uses an array sized $w \times d$, where d is the number of hash functions and w is the space size used by a hash function.

The Count-Sketch algorithm consists of three stages: Stage 1 is for variable initialization; Stage 2 is a loop to process the data stream; and Stage 3 is for processing of the output result. The details of this algorithm can be found in [12].

2.1.6. Count-Min

The Count-Min algorithm was proposed by Cormode and Muthukrishnan in 2005 [5] to replace other sketch-based methods, such as Count-Sketch. The aim of Count-Min is to provide a simple and compact sketch data structure. Count-Min has been widely used in many computing areas thanks to its simple sketch data structure. Count-Min's data structure uses the linear reference system with some random vectors. These vectors are defined by a simple hash function. The increase of the hash function's operating scope will increase the counting accuracy and reduce the error rate. Count-Min's data structure can be expanded or shrunk to fit each specific case without compromising the accuracy.

Count-Min's data structure is a matrix of simple counting variables. The matrix has the width w and depth d, including elements from $CM[1,1]$ to $CM[d,w]$. Similar to other sketch-based methods, Count-Min also uses ε as the error rate and δ as the error probability and the algorithm consists of three stages: Stage 1 is for variable initialization; Stage 2 is a loop to process the data stream; and Stage 3 is for the calculation of occurrence frequencies of elements in the data stream. The details of this algorithm can be found in [5].

2.1.7. Group Testing

The Group Testing method was proposed by R. Dorfman in 1943 [9] to detect who was infected with syphilis in the US Army in the Second World War with the minimum number of blood tests. He first divided each soldier's blood sample into many smaller samples, and then he formed groups of soldiers and mixed their blood samples for each group to make group samples. Then, he carried out a test for each group sample. If the group test result is negative, all samples in the group are negative. If the group test result is positive, at least one sample in the group is positive. Repeat the procedure on the samples of the positive group until the infected sample is found. Thus, the number of tests had been decreased significantly. The purpose of the Group Testing method is to determine a sub-set of d positive elements of a large set of N elements with the number of tests as small as possible [1].

Group Testing methods can be classified into two types: adaptive Group Testing and non-adaptive Group Testing. In adaptive Group Testing, the next test is created based on the result of the previous test. Therefore, tests of this method must be done in sequence. On the other hand, in non-adaptive Group Testing, tests are independent and therefore tests can be processed in parallel to speed up the process. In this research, we use the non-adaptive Group Testing method proposed by Kautz and Singleton [9]. Non-adaptive Group Testing requires that all tests must be designed in advance and then executed at the same time to produce the final result.

The problem of using the non-adaptive Group Testing method to find high-occurrence-frequency elements in a data stream can be stated as follows: Given the data stream a of M elements, in which there are N unique elements ($M >> N$), assume that there is a maximum of d high-occurrence-frequency elements; we need to design t tests for N elements. Construct the binary matrix m_{txN}, where the matrix's columns represent elements and the matrix's rows represent tests. If $m[i][j] = 1$, it means that element j belongs to test i and if $m[i][j] = 0$, it means that element j does not belong to test i [9]. The details of the non-adaptive Group Testing method can be found in [9].

We adopt the non-adaptive Group Testing method to find d high-occurrence-frequency elements used in [1,2,9], in which the d-disjunct matrix m_{txN} is built using the Reed Solomon code concatenation. The binary matrix m_{txN} is the d-disjunct matrix if and only if the union of any d columns does not contain any column in the matrix. The Reed Solomon code is the error-correcting code in the form of $[n, k]_q$, where q is the alphabet size, n is the block length and k is the message length. The relationship among the three parameters is $k < n \leq q$ and it is often that $n = q - 1$. The construction procedure of the d-disjunct matrix using the Reed Solomon code concatenation can be found in [1,2,9].

2.2. Application in Finding Hot-IPs

In this section, we evaluate the performance of the Count-Sketch, Count-Min and Group Testing methods on finding Hot-IPs in IP packet streams extracted from the UCLA (University of California, Los Angeles) data sets [13]. These methods have been widely used to find high-occurrence-frequency elements in the data stream because they have low computational complexity and low space requirements [1,2,4–7].

2.2.1. Experimental Data Sets and Parameters

The UCLA data sets [13] were collected from simulated DDoS attacks using DoS/DDoS attack tools. Raw data of collected packets were processed, whereby each packet is converted into a record. For TCP packets, each record consists of 11 fields, including *Packet_TIME*, *IP_from*, *IP_to*, *PORT_from*, *PORT_to*, *LENGTH*, *FLAG*, *SEQ_from*, *SEQ_to*, *ACK* and *WIN*. In the experiments in this section, the *IP_from* (source IP address) is used to find Hot-IPs.

The experimental data set extracted from the UCLA data sets [13] consists of 200,000 records (also contains M = 200,000 source IP addresses), in which there are N = 2214 unique IP addresses.

For the Count-Sketch and Count-Min methods, select $\varepsilon = 0.005$, $\delta = 0.00000001$ and the number of IPs with the highest occurrence frequency $k = 10$.

For the Group Testing method, select the maximum number of unique IPs L = 10,000, using the Reed Solomon code $RS[n = 15, k = 3]_{q=16}$ and the code concatenation to construct the d-disjunct matrix m_{txN} [5]. The d-disjunct value is computed as: $d = (n - 1)/(k - 1) = (15 - 1)/(3 - 1) = 7$.

We use a one-dimensional matrix C_N to be the counter of the occurrence frequency of IP addresses in the IP packet stream. An IP is considered a Hot-IP if $C[i] > L/(d + 1)$.

2.2.2. Experimental Results

Table 1 shows the result of finding 10 source IP addresses that have the highest occurrence frequency using the Count-Sketch and Count-Min methods. The IP occurrence frequencies found by both methods are almost the same. If we set the occurrence frequency threshold for a Hot-IP as 10,000, we get four Hot-IPs, which are at the top of the table (1.1.17.29, 1.1.6.6, 1.1.35.39 and 1.1.17.17).

Table 1. Result of finding 10 source IP addresses having the highest occurrence frequency using Count-Sketch and Count-Min.

No	IP Addresses	Frequency (Count-Sketch)	Frequency (Count-Min)
1	1.1.17.29	29,670	29,670
2	1.1.6.6	17,243	17,243
3	1.1.35.39	16,345	16,350
4	1.1.17.17	12,068	12,071
5	1.1.4.4	8996	8996
6	15.111.6.94	6573	6556
7	10.82.164.13	4295	4298
8	14.150.78.191	3895	3900
9	1.1.1.94	3727	3727
10	7.210.22.183	3432	3438

Table 2 shows the result of finding Hot-IPs using the Group Testing method. Since the purpose of the Group Testing method is to find d Hot-IPs, the IP occurrence frequencies are not the output parameters. It can be seen that the result of finding Hot-IPs using the Group Testing method is equivalent to the results of the Count-Sketch and Count-Min methods.

Table 2. Result of finding Hot-IPs using Group Testing.

No	Hot-IPs
1	1.1.17.29
2	1.1.6.6
3	1.1.35.39
4	1.1.17.17

2.3. Comparison of Hot-IP Finding Methods

As discussed in Section 2.2, the Count-Sketch, Count-Min and Group Testing methods are capable of finding Hot-IPs correctly in the IP packet stream. In this section, we compare these methods based on the following criteria: (1) algorithm complexity; (2) space requirements; and (3) processing time in finding Hot-IPs.

2.3.1. Complexity of Algorithms

Count-Sketch

The space required for this algorithm is $O(d \times w + 2d)$ because the counter matrix has the size of $d \times w$ and there are two hash functions for each row. Thus, the space required for the algorithm is $O(d \times w + 2d) = O(\ln\left(\frac{1}{\delta}\right) \times \frac{e}{\varepsilon^2})$. The processing time for each element is $O(d) = O(\ln\frac{1}{\delta})$

and the average processing time for d elements is linear; therefore, the processing time required is $O(d) = O(\ln\frac{1}{\delta})$ [12].

Count-Min

Count-Min uses a counter matrix sized $d \times w$ and d hash functions, thus the space required for the algorithm is $O(d \times w) = O(\ln\frac{1}{\delta} \times \frac{e}{\varepsilon})$. The processing time for each element is $O(d) = O(\ln\frac{1}{\delta})$ and the average processing time for d elements is linear; therefore, the processing time required is $O(d) = O(\ln\frac{1}{\delta})$ [6].

Group Testing

The polynomial space required for each counter is $O(\log N + \log M)$ bit, which means the space required for the t counter is $(O(\log N + \log M) \times t)$ bit [1]. Given $t = O(d^2 \log N)$ and $d = O(\log N)$, the space required is $O\left(\log^3 N\,(\log N + \log M)\right)$ [1]. The d-disjunct matrix created using a linear code concatenation (Reed-Solomon code, $RS[n, k]_q$) requires a polynomial time of $O(q^2 \times \text{poly}(\log q))$. Thus, the processing time required is $O(q^2 \times \text{polylog} q)$ [1]. The time required to find the Hot-IP is $\text{poly}(d) \times t\log^2 t + O(t^2)$ [1].

Table 3 presents the complexity of the Count-Sketch, Count-Min and Group Testing methods.

Table 3. Complexity comparison of Count-Sketch, Count-Min and Group Testing methods.

Method	Count-Sketch	Count-Min	Group Testing
Space requirements	$O(d \times w + 2d) = O(\ln\left(\frac{1}{\delta}\right) \times \frac{e}{\varepsilon^2})$	$O(d * w) = O(\ln\frac{1}{\delta} \times \frac{e}{\varepsilon})$	$O\left(\log^3 N\,(\log N + \log M)\right)$
Processing time	$O(d) = O(\ln\frac{1}{\delta})$	$O(d) = O(\ln\frac{1}{\delta})$	$O(q^2 \times \text{poly}(\log(t)))$
Estimated Hot-IP finding time	$O(d) = O(\ln\frac{1}{\delta})$	$O(d) = O(\ln\frac{1}{\delta})$	$\text{poly}(d) \times t \times \log^2 t + O(t^2)$

2.3.2. Comparison of Hot-IP Finding Time

We use data sets extracted from UCLA data sets [13] for our experiments. Our experimental data sets have the numbers of records (also the IP addresses) gradually increased to measure the processing time of the Count-Sketch, Count-Min and Group Testing methods. The sizes of the data sets are gradually increased from 10,000 IP addresses to a maximum of 1,000,000 IP addresses. Because of the differences in the input parameters and space required, we create three scenarios to compare these three methods on two pairs, including (1) Count-Sketch and Count-Min; and (2) Count-Min and Group Testing.

All experiments in this section are implemented in Java and run on a laptop machine with an Intel core i5-M460 2.53 GHz CPU, 8 GB RAM on 64-bit Microsoft Windows 8. For each dataset, five runs are executed and the average of processing time (excluding the time for I/O operations) is the final result.

1 Scenario 1: Compare Count-Sketch and Count-Min using the same initialization parameters $\varepsilon = 0.005$ and $\delta = 0.00000001$. The experiment result is shown in Table 5.

2 Scenario 2: Compare Count-Sketch and Count-Min using the same processing space. For Count-Sketch, select $\varepsilon = 0.01$ and $\delta = 0.00005$, and select $\varepsilon = 0.00009$ and $\delta = 0.00001$ for Count-Min. The experiment result is shown in Table 4.
It can be seen from Tables 4 and 5 that Count-Min is significantly faster than Count-Sketch in both cases of using the same initialization parameters or using the same processing space.

3 Scenario 3: Compare Count-Min and Group Testing using the same matrix size. For Count-Min, select $\varepsilon = 0.00045$ and $\delta = 0.00000001$, and we have the counter matrix CM[19,6041]. For Group Testing, select $RS[15,3]_{16}$ and d-disjunct as 6000 (corresponding to 6000 unique IPs), and we have d-disjunct matrix m[240,6000]. The experiment result is shown in Table 6.

Table 4. Time for finding Hot-IPs of Count-Sketch and Count-Min using the same processing space.

Number of IP in Stream	Count-Sketch (s)	Count-Min (s)
10,000	0.019	0.014
50,000	0.078	0.058
100,000	0.135	0.112
200,000	0.294	0.201
300,000	0.382	0.318
400,000	0.531	0.458
500,000	0.674	0.547
600,000	0.758	0.608
700,000	0.898	0.715
800,000	0.998	0.811
900,000	1.127	0.934
1,000,000	1.260	1.014

Table 5. Time for finding Hot-IPs of Count-Sketch and Count-Min using the same initialization parameters.

Number of IP in Stream	Count-Sketch (s)	Count-Min (s)
10,000	0.093	0.019
50,000	0.152	0.071
100,000	0.271	0.162
200,000	0.445	0.301
300,000	0.637	0.433
400,000	0.849	0.616
500,000	1.038	0.744
600,000	1.202	0.832
700,000	1.400	0.998
800,000	1.731	1.155
900,000	1.876	1.308
1,000,000	2.050	1.374

Table 6. Time for finding Hot-IPs of Count-Min and Group Testing using the same matrix size.

Number of IP in Stream	Count-Min (s)	Group Testing (s)
10,000	0.019	0.023
50,000	0.084	0.077
100,000	0.159	0.143
200,000	0.377	0.354
300,000	0.442	0.388
400,000	0.625	0.497
500,000	0.761	0.625
600,000	0.897	0.822
700,000	1.035	0.916
800,000	1.207	1.047
900,000	1.377	1.224
1,000,000	1.558	1.307

The experimental results given in Table 6 show that Group Testing is slightly faster than Count-Min in finding Hot-IPs in the IP packet stream. However, the space requirement of Group Testing is much higher than that of Count-Min, even though they use the same width matrices because Group Testing uses the d-disjunct matrix with a much larger number of tests t. Overall, Count-Min is more efficient than Group Testing because its space requirement is much smaller than that of Group Testing while its processing time is almost equivalent to that of Group Testing.

3. Proposed Target Detection Model of DDoS Attacks

DDoS (Distributed Denial of Service) is the type of attack that makes the computer or network system overload, and the system cannot provide the service, or has to stop working. While a DoS attack is usually originated from one source, or a small number of sources, a DDoS attack is originated from large number of sources distributed all over the Internet. In real DDoS attacks, network service servers are "flooded" by a huge amount of requests sent from controlled hosts (also called zombies or bots) distributed on the networks [14]. When the amount of requests is too large, the server is overloaded and fails to handle incoming requests. Consequently, legitimate users are not able to access the service provided by servers. Figure 1 illustrates a typical architecture of DDoS attacks.

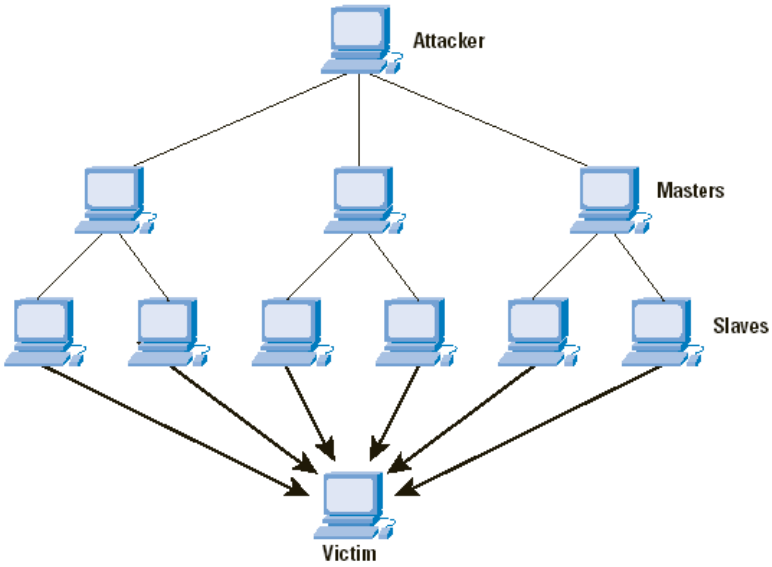

Figure 1. A typical architecture of DDoS attacks.

There have been a number of proposed measures to defend against DDoS attacks over the last decade. However, until now there has not been any solution capable of DDoS prevention comprehensively and effectively due to the complexity, scale and highly distributed nature of DDoS attacks [14–17]. These DDoS defense measures can be classified into three groups: (1) measures based on deployment location; (2) measures based on network protocols; and (3) measures based on the time of action. Group (1) consists of measures that are deployed at the sources or targets of DDoS attacks. On the other hand, group (2) includes measures that defend DDoS attacks at the IP, TCP/UDP (Transmission Control Protocol/User Datagram Protocol), or application layers. Based on the time of action, group 3 includes pre-attack, in-attack and post-attack measures [14–17].

Based on the analysis of DDoS architectures and the Hot-IP finding results presented in Section 2, we propose a DDoS target or victim detection model based on Hot-IP finding, as shown in Figure 2. The IP addresses of service servers are the destination IP addresses of IP packets sent to these servers. Under a DDoS attack, these IP addresses usually have an extremely high occurrence frequency. Therefore, if we deployed a Hot-IP/DDoS detector on the target network router, it is possible to detect the signals of a DDoS attack early. Our DDoS target detection model can be deployed at the target host, or would be best at the router of the target network.

In the proposed model, the Hot-IP/DDoS detector deployed on the target network router is responsible for capturing and processing all IP packets sent to service servers. The detector uses the sliding window method on the IP packet stream to find destination IPs that have high occurrence frequency. Count-Min is the method implemented for finding high-occurrence-frequency IPs in the detector. A threshold of occurrence frequency for Hot-IPs is determined in advance to identify the

possibility of a DDoS attack. The threshold is determined based on the type of network services and the user access patterns when the system is in normal operation. Initial experiment results on a simulated environment show that the detector is capable of quickly identifying Hot-IPs correctly and based on that, it can detect DDoS attacks on network service servers.

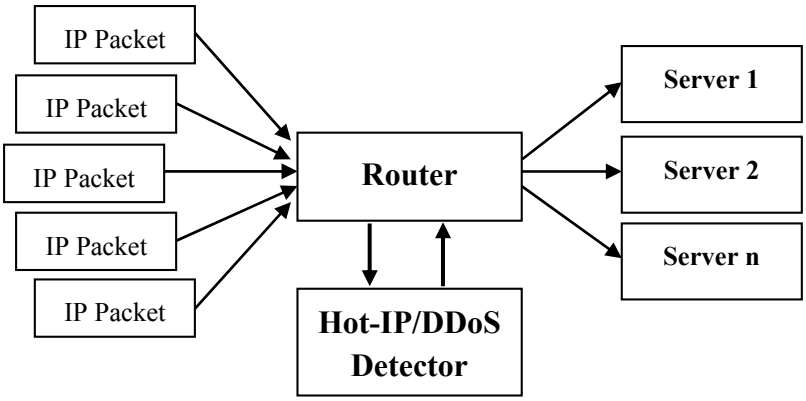

Figure 2. DDoS target detection model based on Hot-IP finding.

4. Conclusions

Finding Hot-IPs in the IP packet stream flowing through the network can be used to detect targets or victims of DDoS attacks, or spreading sources of network worms. It can also be used in applications to monitor activities of network elements. This paper reviewed methods to find high-occurrence-frequency elements in the data stream and applied them in finding Hot-IPs in the IP packet flow passing through the network. Research results shows that Count-Min gives the best overall performance thanks to its low computational and space complexity, and its fast processing speed. We also proposed a Hot-IP finding-based model for early target/victim detection of DDoS attacks, which can be deployed on the router of the target network.

This research can be extended in the following directions: (i) complete the Hot-IP–based detection module and deploy it in the real environment; and (ii) optimize the Hot-IP finding module using embedded processors to speed up the processing of IP packets to be able to monitor large bandwidth network connections.

Acknowledgments: This work has been supported by the CyberSecurity Lab, Posts and Telecommunications Institute of Technology, Hanoi, Vietnam.

Author Contributions: Xuan Dau Hoang raised the idea and initialized the project; Hong Ky Pham designed and carried out the experiments under the supervision of Xuan Dau Hoang; Both authors analyze the data and results; Xuan Dau Hoang wrote the paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Huynh, N.C.; Nguyen, D.T.; Tan, H. Finding Hot-IPs in network using Group Testing method—A review. In Proceedings of the 2012 International Conference on Green Technology and Sustainable Development (GTSD2012), Ho Chi Minh City, Vietnam, 29–30 September 2012.
2. Huynh, N.C.; Nguyen, D.T.; Tan, H. Fast detection of DDoS attacks using Non-Adaptive Group Testing. *Int. J. Netw. Secur. Appl.* **2013**, *5*. [CrossRef]
3. Simkhada, K.; Taleb, T.; Waizumi, Y.; Jamalipour, A.; Namoto, Y. Combating against internet worms in large-scale networks: An autonomic signature-based solution. *Secur. Commun. Netw.* **2009**, *2*, 11–78. [CrossRef]
4. Boyer, B.; Moore, J. *A Fast Majority Vote Algorithm*; Technical Report 35; Institute for Computer Science, University of Texas: Austin, TX, USA, 1982.

5. Cormode, G.; Muthukrishnan, S. An improved data stream summary: The count-min sketch and its applications. *J. Algorithms* **2005**, *55*, 58–75. [CrossRef]
6. Cormode, G.; Muthukrishnan, S. What's hot and what's not: Tracking most frequent items dynamically. *ACM Trans. Database Syst.* **2005**, *30*, 249–278. [CrossRef]
7. Manku, G.; Motwani, R. Approximate frequency counts over data streams. In Proceedings of the 28th International Conference on Very Large Databases, Hong Kong, China, 20–23 August 2002; pp. 246–357.
8. Misra, J.; Gries, D. Finding repeated elements. *Sci. Comput. Program.* **1982**, *2*, 143–152. [CrossRef]
9. Kautz, W.; Singleton, R. Nonrandom binary superimposed codes. *IEEE Trans. Inf. Theory* **1964**, *4*, 363–377. [CrossRef]
10. Fischer, M.; Salzberg, S. Finding a majority among n votes solution to problem. *J. Algorithms* **1982**, *3*, 376–379.
11. MetWally, A.; Agrawal, D.; Abbadi, A.E. Efficient computation of frequent and top-k elements in data streams. In *Proceedings of the 10th International Conference on Database Theory*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 398–412.
12. Charikar, M.; Chen, K.; Colton, M. Finding frequent items in data streams. *Theory Comput. Sci.* **2004**, *312*, 3–15. [CrossRef]
13. UCLA CSD Traffic Traces. Available online: http://www.lasr.cs.ucla.edu/ddos/traces/ (accessed on 12 June 2016).
14. Hoang, X.D. DDoS attack classification and defence measures (Part 1). *J. Inf. Commun.* **2014**, *483*, 37–40.
15. Zargar, S.T.; Joshi, J.; Tippe, D. A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. *IEEE Commun. Surv. Tutor.* **2013**. [CrossRef]
16. Hashmi, J.; Saxena, M.; Saini, R. Classification of DDoS Attacks and their Defense Techniques using Intrusion Prevention System. *Int. J. Comput. Sci. Commun. Netw.* **2012**, *5*, 607–614.
17. Alenezi, M. Methodologies for detecting DoS/DDoS attacks against network servers. In Proceedings of the Seventh International Conference on Systems and Networks Communications—ICSNC 2012, Lisbon, Portugal, 18–23 November 2012.