Filtering Spoofed IP Traffic Using Switching ASICs[*]

Jiasong Bai, Jun Bi, Menghao Zhang, Guanyu Li
Institute for Network Sciences and Cyberspace, Tsinghua University
Department of Computer Science and Technology, Tsinghua University
Beijing National Research Center for Information Science and Technology (BNRist)

CCS CONCEPTS

• **Security and privacy → Network security**; • **Networks → Network architectures**;

KEYWORDS

Programmable Switch, Spoofed IP Traffic, Hop-Count Filter

ACM Reference Format:
Jiasong Bai, Jun Bi, Menghao Zhang, Guanyu Li. 2018. Filtering Spoofed IP Traffic Using Switching ASICs. In *Posters-Demos '18: Posters and Demos, August 20–25, 2018, Budapest, Hungary*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3234200.3234205

1 INTRODUCTION

Spoofed IP traffic remains a significant threat to the Internet. Although an attacker can forge any field in the IP header, she/he cannot falsify the number of hops an IP packet takes to reach its destination. Based on this, some previous works propose hop-count based defense mechanisms to filter spoofed IP traffic (e.g., TCP, UDP, ICMP) with an IP-to-hop-count (IP2HC) mapping table [3, 5]. To guarantee the correctness, the table should only be updated by legitimate packets, i.e., TCP connections with *established* states. Since the monitoring of TCP establishment procedures can only be conducted in hosts under traditional networks, these defense mechanisms are *all* located in end systems. Therefore, spoofed packets cannot be filtered until they arrive at the targeted server, making end hosts still suffer.

The advent of the programmable data plane allows offloading intelligence into the network, providing opportunities to conduct the TCP connection monitoring in switches.

[*]Supported by National Key R&D Program of China (2017YFB0801701), the National Science Foundation of China (No.61472213) and CERNET Innovation Project (NGII20160123). Jun Bi is the corresponding author.

In this poster, we present NetHCF, an in-network spoofed traffic filtering system. By building the IP2HC table in key switches and filtering spoofed packets with switching ASICs, NetHCF can provide *bandwidth protection* for legitimate traffic and save *computation* and *storage* resources for end servers. Benefit from the high performance of the switching ASICs, NetHCF is able to examine every packet to update the IP2HC table without sampling, and perform spoofed packets detection and filtering at line rate.

Despite these great benefits, applying the traditional techniques of HCF into a single switch is non-trivial. Several challenges must be carefully addressed:

First, fitting the whole IP2HC table in limited SRAM and keeping the false positive rate acceptable. It is impossible to store the IP2HC table at the granularity of each IP address (2^{32} bytes (4GB)) in limited switch SRAM ((50~100MB) [4]), and a hash-based space compression has to be adopted. Given a specific storage size, the more mappings are added, the larger the false positive rate becomes as the probability of collision rises. To address this, we adopt multiple hash functions to map a source IP to multiple entries with a 32-bit bitmap and give the theoretical analysis accordingly (§2.1).

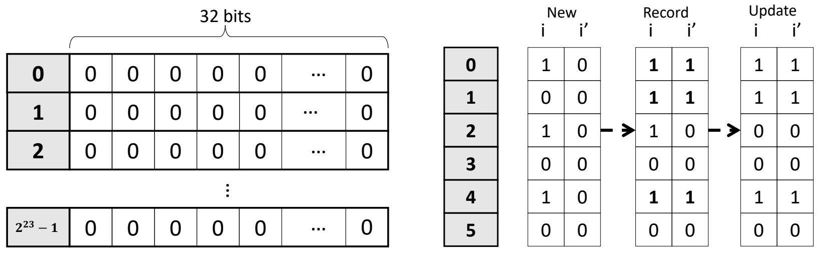
Second, recording limited mappings and updating the IP2HC table timely. With new mappings generated all the time, some mappings have to be removed timely, otherwise the false positive rate will increase. Typically we hope to remove mappings with a lower probability to be accessed in the following time. We design an efficient update procedure to remove the out-dated mappings in the IP2HC table (§2.2).

Besides, we clarify two running states of NetHCF to identify the attack scenarios (§2.3), and give a preliminary implementation and evaluation to demonstrate that NetHCF provides effective filtering with only minor overheads (§3).

2 NETHCF DESIGN

2.1 IP2HC Table

The IP2HC table is the core module of NetHCF. It compresses the address space from 2^{32} to 2^{23}, and each entry maintains a 32-bit bitmap to record all possible hop-count values. Although not recorded directly, the hop-count can be easily inferred by subtracting the final TTL from the initial TTL[1, 3]. Inspired by *bloom filter*, we use k hash functions to map an IP address to different entries. The i-th bit of the j-th entry is set to 1 when a legitimate packet with i hop-count



(a) Table Structure. (b) Update Procedure.

Figure 1: IP2HC Table.

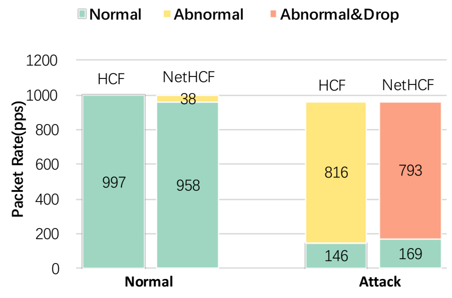


Figure 2: Bandwidth Protection.

gets hashed to j by one or more hash functions. When a TCP connection is established, it gets hashed by all k functions, and corresponding bits are set to 1. For a few packets with hc bigger than 32, they can be stored in the $hc\%32$ location. As depicted in Figure 1(a), a table takes 2^{28} bits or 32 MB, fitted for most commodity switches [4].

There will be false positives for IP2HC, when a forged packet with wrong hop-count happens to be hashed to bits already set as 1. Given an IP2HC table with m entries and k hash functions, after inserting n mappings ($n = \sum_{i=0}^{31} n_i$, n_i stands for number of packets with i hops), the probability that a specific bit i is still 0 is $p = (1 - \frac{1}{m})^{kn_i} \approx e^{-kn_i/m}$. The probability of a false positive is

$$f = (1-p)^k = (1-e^{-kn_i/m})^k = e^{kln(1-e^{-kn_i/m})} = e^{\frac{-m}{n_i}ln(x)ln(1-x)}$$

f gets minimized when

$$x = 1 - e^{-kn_i/m} = \frac{1}{2} \Rightarrow k = \frac{m}{n_i}ln2 \Rightarrow f = (1/2)^{\frac{m}{n_i}ln2}$$

Set k as 7, we have $f < 0.01$ for $n_i < 8.5 \times 10^5$. Considering n_i is the number of mappings sharing the same hc value i, we have $n = 32 * \overline{n_i}$. Given a m size table with k hash functions, NetHCF can work at a relatively low false positive rate f to keep less than n mappings.

2.2 Update IP2HC Table

Considering the access locality, the mappings which have not been accessed in the past period are less likely to be visited in the future. During an update, these mappings should be deleted first. We double the bitmap to catch the time-related information since it is missed in the IP2HC table. To help clarify the update procedure, we use a table with m equals 6, k equals 2 to give a brief overview in Figure 1(b), i stands for front i-th bits and i' stands for cache i-th bits used for matching.

The IP2HC table is updated periodically. At the beginning of each period, the cache bits are cleared (New stage). During the period (Record stage), a cache bit is set to 1 when (1) the front bit of this entry is accessed by a normal packet (2) a new IP with i hc is hashed to this entry. As shown in Figure 1(b), if a packet with i hops is hashed to 0 and 4 entry, the two front bits are accessed, then the cache bits of entry 0 and 4 are set to 1. Meantime, if a new source IP is hashed to 0 and 1 entry, both front and cache bits of these two entries are set to 1. At the end of each period (Update stage), values of cache bits are copied to front bits, then cache bits get cleared and continue to perform the above operations (New stage). The frequency of update should be set according to n value and the emerging speed of new source IPs.

2.3 Learning and Filtering State

There are two states of NetHCF according to whether to discard spoofed packets. In both states, NetHCF monitors handshake messages to maintain a TCPSession table [2]. For connections passing three handshakes, NetHCF removes them from the TCPSession table and updates IP2HC to record new mappings. Incoming packets are hashed by k functions to check whether they have correct hop-count values. For spoofed packets which fail the IP2HC check, NetHCF ignores them in learning state while discards them in filtering state.

To switch between two states, we count the number of spoofed packets in a short period to decide whether to filter the spoofed packets. When the number exceeds a high threshold T_{high}, NetHCF enters the filtering state and begins to drop the spoofed packets. And if the number goes lower than a low threshold T_{low}, NetHCF returns to learning state and passes all packets to the server.

3 EVALUATION AND FUTURE WORKS

We implement a prototype[1] of NetHCF in bmv2 in 400 lines of P4. As depicted in Figure 2, under normal circumstances, NetHCF mistakes a few normal packets as spoofed ones for the sake of hash collisions in the TCPSession table. Since NetHCF passes all packets in learning state, the communication will not get influenced. Under attack circumstances, NetHCF is able to recognize and drop most spoofed packets with 2.82% of false positive rate. Our prototype only implements one hash function for IP2HC, the false positive rate can be decreased by applying more hash functions as the analysis in Section 2.1. In future, we will implement all details and conduct more evaluations, and consider more complicated challenges such as network-wide coordination, TCP connections with asymmetric paths, etc.

[1]https://github.com/ZhangMenghao/Anti-spoof

REFERENCES

[1] Noah Davids. 2018. default TTL values. http://noahdavids.org/self_published/TTL_values.html. (2018).
[2] Cheng-Hung He and et al. 2018. A Zero Flow Entry Expiration Timeout P4 Switch. In *SOSR*. ACM, 19.
[3] Cheng Jin and et al. 2003. Hop-count filtering: an effective defense against spoofed DDoS traffic. In *CCS*. ACM, 30–41.
[4] Barefoot Networks. 2018. Tofino. https://barefootnetworks.com/products/brief-tofino/. (2018).
[5] Haining Wang, Cheng Jin, and Kang G Shin. 2007. Defense against spoofed IP traffic using hop-count filtering. *ToN* 15, 1 (2007), 40–53.