Hindawi Publishing Corporation
The Scientific World Journal
Volume 2014, Article ID 659894, 11 pages
http://dx.doi.org/10.1155/2014/659894

Research Article

Storage-Efficient 16-Bit Hybrid IP Traceback with Single Packet

Ming Hour Yang

Information and Computer Engineering, Chung Yuan Christian University, 200 Chung Pei Road, Chung Li City, Taoyuan County 32023, Taiwan

Correspondence should be addressed to Ming Hour Yang; mhyang@cycu.edu.tw

Received 12 May 2014; Revised 5 September 2014; Accepted 8 September 2014; Published 20 October 2014

Academic Editor: Marco Listanti

Since adversaries may spoof their source IPs in the attacks, traceback schemes have been proposed to identify the attack source. However, some of these schemes' storage requirements increase with packet numbers. Some even have false positives because they use an IP header's fragment offset for marking. Thus, we propose a 16-bit single packet hybrid IP traceback scheme that combines packet marking and packet logging with high accuracy and low storage requirement. The size of our log tables can be bounded by route numbers. We also set a threshold to determine whether an upstream interface number is stored in a log table or in a marking field, so as to balance the logging frequency and our computational loads. Because we store user interface information on small-degree routers, compared with current single packet traceback schemes, ours can have the lowest storage requirements. Besides, our traceback achieves zero false positive/negative rates and guarantees reassembly of fragmented packets at the destination.

1. Introduction

Recent years have seen the rapid growth of the Internet, and the widespread Internet services have become a part of our daily life. These services, however, are vulnerable to many potential threats. To name one out of many, a malicious user may launch distributed/denial of service (D/DoS) attacks to disrupt the Internet services. Judging from the number of attacking packets, a D/DoS attack can be categorized into flooding-based attack and software exploit attack [1]. In a flooding-based attack, the victim's resources can be exhausted by a huge amount of forged source packets. But in a software exploit attack, a villain needs to find the host's vulnerabilities and then uses only a few packets to launch attacks, for example, Teardrop attacks and LAND attacks [2]. Normally the source and destination IP addresses are stored in a packet's IP header to indicate its source and destination. In practice, however, most routers do not verify a packet's source IP. This is why attackers usually take this advantage and spoof their real address to evade tracking. This security issue has come to our attention and we find it urgent to propose an efficient traceback scheme tracking the real source of impersonation attacks.

Therefore, packet-marking schemes are proposed to trace the real source of flooding-based packets. They use the free fields of each packet's IP header to mark the packet's route and the routers along the route. As these packets are usually in a huge amount, these marking schemes are categorized as probabilistic packet marking (PPM) [3–9] and deterministic packet marking (DPM) [10–14]. The two methods are proposed to lower the routers' marking loads. While PPM is purely based on probability, DPM puts a single mark on inbound packets at the point of network ingress. However, both PPM and DPM require at least eight packets for path reconstruction [12], so they may not be able to trace the source of software exploit attacks, which can use only one packet to paralyze the system.

If we want to achieve single packet traceback, we have to use packet logging schemes [15–17] to log the packet's unchanged data on the routers. And the path reconstruction requires hop-by-hop queries of previous routers. For example, in Snoeren et al.'s SPIE [15] and Zhang and Guan's TOPO [16], routers that comply with these schemes have to use a Bloom filter [18] to log their packets' digests. But if the filter logs too many packets, there might be collision in their log tables and therefore they will have false positives during path reconstruction. In order to reduce the chance of collision and keep the false positive rate within 1%, SPIE has to back up and refresh its log tables when the accumulative packet size is larger than the table's size by 20% [15]. Likewise, TOPO

[16] uses each upstream router's identifier to decrease the chance of collision and false positives. However, like SPIE, TOPO still has to clear its logged data on the routers when the packet number is too large. If the log tables are refreshed, the traceback scheme is unable to reconstruct the attack route.

For these reasons, hybrid single packet traceback schemes have been proposed to combine packet marking and packet logging. These methods can achieve single packet tracking and have lower storage requirements and false positive rates. There are two types of these hybrid single packet traceback schemes: the first type hashes each packet's marking field (or some specific fields of a packet's header) to compute an index and modify the indexed value in the Bloom filter [19–23]. But the storage requirement on each router grows when the packet number increases. When it exceeds the router's quota, the logged data will be refreshed and the path reconstruction fails.

The other type encodes a packet's route as a mark and stores it in the packet's header. If the mark is larger than the size of a marking field, the packet's route is logged onto a router [24–26] to decrease each router's storage loads. These schemes decrease the false negative rate because the logged data in a router does not need to be refreshed. Lu et al. use multiprotocol label switching (MPLS) networks [25] to encode a router's upstream routes and the destination router's ID as a 29-bit mark. When a router receives the packet, it uses the packet's destination IP as an index to choose a log table to log this mark. Then the router writes its ID and the packet's upstream routes into the mark, so that the downstream routers can use the mark to trace the origin of the attack. However, in Lu et al.'s scheme, every router that the packet passes through has to log the packet's mark. Besides, the scheme does not have indexes for their log tables. It needs to do an exhaustive search during path reconstruction, so as to find the corresponding upstream interface number of the attack packet. Since the exhaustive search consumes lots of computation power of a router, it makes their traceback scheme not practical. Hence, M. H. Yang and M. C. Yang propose RIHT [24] to encode all the upstream routers' interface numbers as their log table's indexes. The routers do not need to search their log tables during path reconstruction. In their scheme, the maximum storage requirement for each router is only 320 KB, and their average storage requirement can remain low because they do one logging on every three routers.

Besides, RIHT's marking field requires 32 bits and consists of an IP header's ID, Fragment Flags, and Fragment Offset [24], whereas Lu et al. use ID and Fragment Offset as their 29-bit marking field. But in the two schemes, if a packet's size exceeds the maximum transmission unit (MTU), the packet will be fragmented and cannot be assembled at the destination. Thus Yang proposes a 16-bit traceback scheme (Hybrid Single-Packet IP Traceback with Low Storage and High Accuracy, HAHIT) [26] using only the ID field as their marking field. If a router receives a packet whose mark is larger than 65535, the router hashes the packet's destination IP and uses the hash value to assign a log table; it also hashes the packet's mark to compute an index value. According to the table number and the index value, the packet's route is logged on the router. Unlike RIHT [24] and Lu et al.'s scheme [25], Yang's scheme [26] prevents the fragmented packets from being dropped and guarantees precision in the traceback of an attack. Besides, because a router that supports IPsec may need to add ESP's header to each packet, it can increase a packet's length and the chance of fragmentation. According to John and Tafvelln's research [27], 63% fragmented packets are ESP packets. Hence, IPsec may not work because of the high chance of packet fragmentation and because of the difficulty in packet reassembly. Also, the values of Fragment Flag and Fragment Offset are used to show whether a packet is fragmented or not. If they are used as a marking field instead, the downstream router cannot tell if the received packet has been fragmented. Further, RIHT and Lu et al.'s method may have a packet-drop issue because, according to RFC 6274 [28], when the value of Fragment Offset is larger than a packet's maximum length, the packet will be dropped.

However, in Yang's 16-bit hybrid single IP traceback scheme [26], he uses the quadratic probing algorithm to search an available index for his log tables and to minimize the impact of collision. In quadratic probing, the load factor suggests the usage rate of each log table. RIHT defines its load factor according to the chance of their successful and unsuccessful searches, and it finds its unsuccessful search rate soars when each log table has used over half its slots. In order to balance the collision times and each table's usage rate, Yang sets his load factor as 0.5. However, the use of quadratic probing has caused half of his log tables to be unused and this results in a waste of space to the routers. To reduce the storage requirements for logging, we propose two schemes in our 16-bit hybrid traceback protocol to encode the upstream routers' interface numbers as an index of the log table's entry. A router will compare its degrees with a threshold to choose a coding scheme to calculate the mark. Therefore, we can decrease the storage requirements by reducing the logging frequency. Also, we propose a logging scheme to further reduce the storage requirements for logging. To write the packet's route into a log table, we search the first empty slot in the log table from the top to the bottom sequentially. The main contributions of our scheme are listed below and we aim to satisfy the first three so as to achieve the last two:

(i) Single IP traceback.

(ii) The storage requirements of logging are bounded by the number of upstream routes, and no duplicate route is logged.

(iii) The fragmented packets can be reassembled at the destination.

(iv) Reduction of the logging frequency.

(v) Decrease of the storage requirements of logging.

Our traceback scheme will be elaborated in the following sections. The simulation, analysis, and comparison of our scheme and other related hybrid traceback approaches are provided in Section 3. And a conclusion is drawn in Section 4.

Table 1: Our marking field in an IP header (the bold text).

Bit offset	0–3	4–7	8–15	16–18	19–31
0	Version	Header length	TOS	Total length	
32	**Identification field**			Flag	Fragment offset
64	TTL		Protocol	Header checksum	
96	Source address				
128	Destination address				
160	Options				
160 Or 196+	Payload (first 8 bytes)				

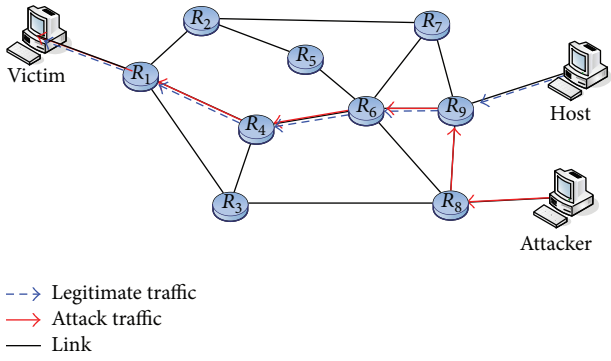


Figure 1: An example setup of our traceback scheme.

2. Single Packet IP Traceback Protocol

In order to prevent packet drop caused by fragmentation and high storage requirements, we propose a new marking scheme to further decrease the storage requirements for a router. As shown in Table 1, we use the 16-bit ID field as our marking field in our traceback scheme. While we keep low storage requirements, our storage can still be bounded by path numbers and the fragmented packets can be reassembled.

Figure 1 illustrates an example setup of our traceback scheme. A router can be connected to a local network or other routers. Here we assume there are y routers in the set R, that is, $R = \{R_1, R_2, \ldots, R_i, \ldots, R_y\}$, and each router complies with our protocol. A border router receives packets from its local network and sends the packets to the destination through a core router. Because packets come from different sources, a border router may also be a core router. For example, R_9 serves as a border router when it receives packets from Host. However, it becomes a core router when receiving packets from R_8. In the following discussion, we use $D(R_i)$ to indicate the degree of router R_i, that is, the number of routers adjacent to R_i. But the degree does not include the interface of a LAN. Besides, we require each router to maintain an interface table, in which UI_i denotes the upstream interface number of router R_i and $0 \le \mathrm{UI}_i \le D(R_i) - 1$.

In our protocol, any router R_i and its network topology has to follow the following assumptions:

(i) R_i is secure against attacks,

(ii) R_i maintains an interface table, in which the interface number ranges from 0 to $D(R_i) - 1$,

(iii) routers on an attack route, from the attack source to the victim, have stable interface tables and degrees during path reconstruction,

(iv) R_i knows whether a packet comes from a router or from a local network,

(v) this traceback scheme is viable on every router. If there are any routers unable to comply with this scheme, they can establish a tunnel to communicate with each other.

The notations used in our protocol are listed in Notations section.

Our traceback scheme consists of two stages: the first is marking/logging stage, and the second is path reconstruction. The steps of how we trace the origin of an attack will be elaborated in the following subsections.

2.1. Marking and Logging. In our marking scheme, we mark a router' interface numbers and store the mark in a packet's IP header. But an IP header has only limited space, so we combine logging with marking to log marks on the routers. During path reconstruction, each router can only track its upstream router's adjacent interface number.

When a packet enters a network from its host, every router that complies with our protocol has to mark its own route info on the passing packets and store the mark in each packet's marking field. Each router's route info consists of the interface number where the packet enters; its log table's information; and its degrees. The packets that a router receives can be categorized into two types. In the first type, when a border router receives a packet from its local network, it sets the packet's marking field as zero and forwards the packet to the next core router. Therefore, when adversaries send attack packets with a forged path in the marking field trying to confuse our tracking, we can still locate their origin correctly. Hence we can verify whether a router is the source router of an attack by checking if the marking field is zero. On the other hand, when a core router R_i receives a packet P_j from another router, R_i uses packet P_j's mark $P_j \cdot$ mark and the incoming interface UI_i and the degree $D(R_i)$ to compute a

Table 2: Example of any log table HT_k.

(a) If $D(R_i) \leq$ threshold

$[T_k^s, T_k^f)$ HT_k	
Index	Mark
0	Source router
⋮	⋮
l	$P_j \cdot mark$
⋮	⋮

(b) if $D(R_i) >$ threshold

$[T_k^s, T_k^f)$ HT_k		
Index	Mark	UI
0	Source router	
1	$P_g \cdot mark$	$P_g \cdot \mathrm{UI}_i$
⋮	⋮	⋮
l	$P_j \cdot mark$	$P_j \cdot \mathrm{UI}_i$

new marking field $\mathrm{mark}_{\mathrm{new}} = P_j \cdot \mathrm{mark} \times (D(R_i) + 1) + \mathrm{UI}_i + 1$. If $\mathrm{mark}_{\mathrm{new}}$ does not overflow, that is, ≤65535, the core router R_i overwrites $P_j \cdot$ mark with $\mathrm{mark}_{\mathrm{new}}$ and then forwards the packet to the next router. If $\mathrm{mark}_{\mathrm{new}}$ overflows, that is, >65535, the core router R_i has to log the mark into a log table and use the index value to calculate a new $\mathrm{mark}_{\mathrm{new}}$. However, such a marking and logging method may require more log tables on a router. According to CAIDA's skitter data [29], this method would exceed a log table's maximum entries [26].

As shown in Table 2(a), a router's log table HT_k consists of three parts: the top row is used to indicate the table's creation time T_k^s and fill-up time T_k^f; the left column indicates the index of each entry; the right column stores packets' marks. The marks include the routers' interface numbers and are passed to the next router with the packets. But a large degree $D(R_i)$ makes a large logged mark, which can cause high logging frequency and increase the storage requirements for its downstream routers. To lower the logging frequency caused by a large interface number UI_i, router R_i logs the packet mark and its interface number UI_i to reduce the mark size; see Table 2(b). However, if we insert the interface number into a logging table, it requires more storage for router R_i to store the table. To balance the storage requirements for router R_i and its downstream routers and to have lower average global storage requirements, we set a threshold for a router's degree so as to decide whether to write the interface number UI_i into a packet's header or into a log table. We suggest the value of threshold in Section 3.

Algorithm 1 shows the detailed steps of our marking and logging scheme. When a router receives a packet P_j and needs to log its mark, the router checks its degree $D(R_i)$ to decide whether or not to log the interface number UI_i; compare lines 29–33 in Algorithm 1. If its $D(R_i) <$ threshold, the router chooses the log table HT_k by hashing the packet's $P_j \cdot$ srcIP to calculate the table number $k = H_{\mathrm{table}}(P_j \cdot \mathrm{srcIP})$; compare line 6 in Algorithm 1. Next, the router checks if there is any logged mark in HT_k identical with $P_j \cdot$ mark, searching from entry index $l = 1$ in monotonic ascending order to the largest index value; compare lines 17–19 in Algorithm 1. If the router cannot find a matched mark in the table, it logs $P_j \cdot$ mark into the empty entry that has the smallest index value; compare line 26 in Algorithm 1. Because we have set a threshold value, when $D(R_i) <$ threshold, our protocol has to write the interface number UI_i into the packet's mark instead of having it logged. This part is the main difference between our marking/logging and HAHIT's [26], and it is aimed at preventing data collision and waste of table size in HAHIT.

Then, the router uses the entry's index l and the packet's incoming interface UI_i to compute a new mark $\mathrm{mark}_{\mathrm{new}} = ((\mathrm{UI}_i \ll l \cdot \mathrm{length}) + l) \times (D(R_i) + 1)$ and sends the new mark to the downstream router; compare line 32 in Algorithm 1. When $D(R_i) <$ threshold, the router searches the table, from $l = 1$ to the largest index value in monotonic ascending order, to check if there is a logged mark and a logged UI that are identical with $P_j \cdot$ mark and UI_i; compare lines 13–15 in Algorithm 1. If it cannot find a matched mark and UI, it logs $P_j \cdot$ mark and UI_i as a pair into the empty entry that has the smallest index value; compare lines 13–15 in Algorithm 1. Since UI_i is now logged, here we only use the index value l to compute a new mark $\mathrm{mark}_{\mathrm{new}} = l \times (D(R_i) + 1)$; compare line 30 in Algorithm 1. Next the router sends the new mark to the downstream router.

Since which table will be used to log a packet is determined by the hash value of the packet's source, packets that have the same source IP but come from different routes will be logged in the same table [26]. A large table leads to large index values and large marks, which will cause high logging frequency in the downstream routers. But a logging table with limited size will be filled up quickly if we use a hashed source IP to determine the table number. To prevent the problem of insufficient table entries, we create a new table when the table is full. We use each table's top row $[T_k^s, T_k^f)$ to indicate the table's creation time T_k^s and fill-up time T_k^f, in which $T_k^s < T_k^f$ and T_k^0 is the first table's creation time. Therefore, every log table's top row is initialized as $[T_k^0, T_k^\infty)$. The analysis of the threshold value, the log table's size and the log table's numbers, and how they affect the storage requirements for one single router or for all the routers will be provided in Section 3.

Figure 2(a) exemplifies our marking and logging scheme. In the figure, packets P_1, P_2, and P_3 come from different sources and their marks are logged on routers R_2 and R_3, whose threshold = 3. The grey cells in Figure 2 indicate the indexed entries of the log tables. When P_1 passes through R_1 en route for R_2, its mark is larger than 65535. The router uses P_1's source IP to calculate $k = H_{\mathrm{table}}(P_1 \cdot \mathrm{srcIP}) = 0$, so the mark will be logged into the table HT_0. Then it goes to the cell whose index value $l = 1$, that is, $HT_0^1 \cdot$ mark, and compares the stored mark with $P_1 \cdot$ mark. Because the router's degree is only 3, it searches the table from $l = 1$ to $l = 11$. As the router cannot find any mark that matches $P_1 \cdot$ mark, it logs $P_1 \cdot$ mark into HT_0^{11}.

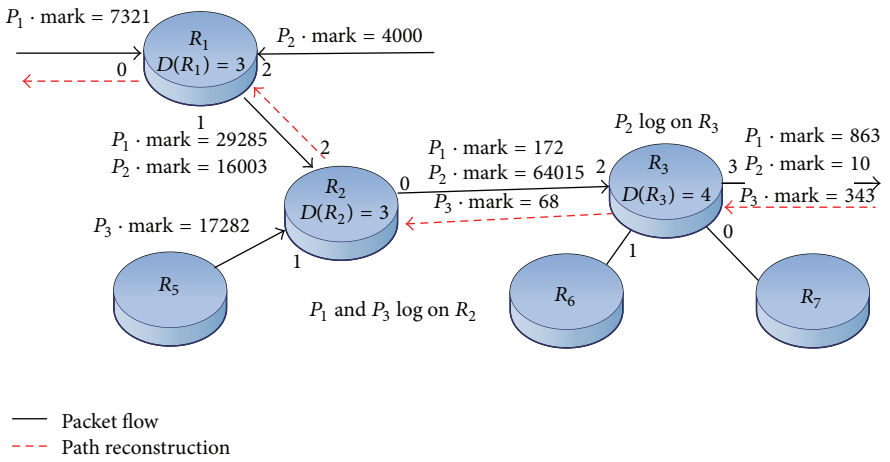
Input: P_j, UI_i
begin
(1) **if** P_j comes from LAN
(2) $P_j \cdot mark = 0$
(3) **else**
(4) $\mathrm{mark}_{\mathrm{new}} = P_j \cdot mark \times (D(R_i) + 1) + \mathrm{UI}_i + 1$
(5) **if** $\mathrm{mark}_{\mathrm{new}} > 65535$
(6) Get table number $k = HT_{\mathrm{table}}(P_j \cdot srcIP)$
(7) **if** HT_k is full
(8) ${}^{\infty}_{t}HT_k \Rightarrow {}^{t+1}_{t}HT_k$
(9) Create new log table ${}^{\infty}_{t+1}HT_k$
(10) **endif**
(11) $l = 1$
(12) **if** $(\mathbf{D(R_i)} > \mathbf{threshold})$
(13) **while not** $(HT^l_k == \varnothing$ or $HT^l_k == (P_j \cdot mark, \mathrm{UI}_i))$
(14) l++
(15) **endwhile**
(16) **else**
(17) **while not** $(HT^l_k == \varnothing$ or $HT^l_k == P_j \cdot mark)$
(18) l++
(19) **endwhile**
(20) **endif**
(21) **if** $HT^l_k == \varnothing$
(22) **if** $(\mathbf{D(R_i)} > \mathbf{threshold})$
(23) $HT^l_k \cdot \mathrm{mark} = P_j \cdot mark$
(24) $HT^l_k \cdot \mathrm{UI} = \mathrm{UI}_i$
(25) **else**
(26) $HT^l_k \cdot \mathrm{mark} = P_j \cdot mark$
(27) **endif**
(28) **endif**
(29) **if** $(\mathbf{D(R_i)} > \mathbf{threshold})$
(30) $\mathrm{mark}_{\mathrm{new}} = l \times (D(R_i) + 1)$
(31) **else**
(32) $\mathrm{mark}_{\mathrm{new}} = ((\mathrm{UI}_i \ll l \cdot length) + l) \times (D(R_i) + 1)$
(33) **endif**
(34) **endif**
(35) $P_j \cdot mark = \mathrm{mark}_{\mathrm{new}}$
(36) **endif**
(37) forward the packet to the next router
end

Algorithm 1: Our marking and logging scheme.

After packet P_2 passes through the routers R_1 and R_2, it enters R_3 and needs to be logged. The router uses P_2's source IP to compute $k = 3$ and therefore the mark will be logged into the table HT_3. However, as Figure 2(c) shows, after searching the table R_3 finds that $P_2 \cdot$mark and $HT^2_3 \cdot$mark are identical and the interface that P_2 enters is the same as $HT^2_3 \cdot \mathrm{UI}$. It means this route has been taken by other packets and it has been logged in the table. Since the mark that matches $P_2 \cdot \mathrm{mark}$ is logged in the cell HT^2_3, we use $l = 2$ and $D(R_3) = 4$ to compute a new mark $\mathrm{mark}_{\mathrm{new}} = 2 \times (4+1) = 10$.

When P_3 needs to be logged into R_2's HT_0 but HT_0 has reached its storage limit, the table's fill-up time will be changed to the present time T^1_0. A new log table will be created and its creation time will be set as T^1_0, denoted as ${}^{\infty}_{1}HT^1_0$. Last, $P_3 \cdot \mathrm{mark}$ will be logged into the new table's first entry ${}^{\infty}_{1}HT^1_0$.

2.2. Path Reconstruction. As shown in Algorithm 2, when a victim detects P_j as an attack packet at the time T_r, it sends P_j and T_r to the tracking server and requests the server to find the attack source. The server sends the packet's mark $P_j \cdot$mark, source IP $P_j \cdot \mathrm{srcIP}$, and the reception time T_r to the victim's upstream routers. When each router receives the request for path reconstruction, it uses $P_j \cdot$mark to compute UI_i; compare line 1 in Algorithm 2. If UI_i is not −1, it means the packet has never been logged on this router. Then the router uses the received mark to compute $\mathrm{mark}_{\mathrm{old}} = P_j \cdot \mathrm{mark}/(D(R_i)+1)$ and



(a) Network topology and packet flow

(T_0^0, T_0^1)	
R_2's HT_0	
Index	Mark
0	Source router
1	17952
2	25109
3	23428
4	27116
5	27718
6	20293
7	17203
8	18952
9	26227
10	20238
11	29285

(T_0^0, T_0^∞)	
R_2's HT_0	
Index	Mark
0	Source router
1	17282

(b) R_2's log table

(T_2^1, T_2^∞)		
R_3's HT_2		
Index	Mark	UI
0	Source router	
1	17034	2
2	28133	2

(T_3^1, T_3^∞)		
R_3's HT_3		
Index	Mark	UI
0	Source router	
1	25689	1
2	64015	2
3	30958	1
4	17094	0
5	26785	2
6	24187	2
7	17453	1

(c) R_3's log table

FIGURE 2: Example of our traceback.

sets $P_j \cdot \text{mark}$ as mark_{old}; compare lines 34-35 in Algorithm 2. Next, it sends the request to its upstream router that is adjacent to UI_i; compare line 35 in Algorithm 2. If a router's UI_i is –1, it means the packet has been logged on this router. If the router's degree does not exceed the threshold, the router divides $P_j \cdot \text{mark}$ by $(D(R_i) + 1)$ and obtains the index value l and UI_i; compare lines 6-7 in Algorithm 2. If $l \neq 0$, it uses $P_j \cdot \text{srcIP}$ to find which table $P_j \cdot \text{mark}$ is logged in. It computes $k = H_{\text{table}}(P_j \cdot \text{srcIP})$; compare line 10 in Algorithm 2. Because the current log table may not be the one that was used to log $P_j \cdot \text{mark}$, the router has to use T_r to find the log table whose time field matches T_r. After the table number and its time field are verified, the router follows the index value l to retrieve mark_{old} from table k; compare lines 16 and 26 (if table is full) in Algorithm 2. Because of our setup of a threshold value, when $D(R_i) <$ threshold, our path reconstruction has to take UI_i and l into consideration. And this part has made the major difference between our path reconstruction and HAHIT's [26]. If the router's degree $D(R_i)$ is larger than the threshold, it uses the mark $P_j \cdot \text{mark}$ to compute $l = P_j \cdot \text{mark}/(D(R_i) + 1)$; compare line 4 in Algorithm 2. Likewise, if $l \neq 0$, it computes the table number k (cf. line 10 in

input: $P_j \cdot mark$, $P_j \cdot srcIP$, T_r
begin
(1) $\text{UI}_i = P_j \cdot mark \,\%\, (D(R_i) + 1) - 1$
(2) **if** $\text{UI}_i = -1$
(3) **if** $(\mathbf{D(R_i)} > \mathbf{threshold})$
(4) $l = P_j \cdot mark/(D(R_i) + 1)$
(5) **else**
(6) $\text{UI}_i = (P_j \cdot mark/(D(R_i) + 1))/2^{l\text{-}length}$
(7) $l = (P_j \cdot mark/(D(R_i) + 1) - 1))\%2^{l\text{-}length}$
(8) **endif**
(9) **if not** $l = 0$
(10) Get table number $k = H_{\text{table}}(P_j \cdot srcIP)$
(11) **if** $T_t < T_r < T_\infty$
(12) **if** $(\mathbf{D(R_i)} > \mathbf{threshold})$
(13) $\text{UI}_i = HT_k^l \cdot \text{UI}$
(14) $\text{mark}_{\text{old}} = HT_k^l \cdot \text{mark}$
(15) **else**
(16) $\text{mark}_{\text{old}} = HT_k^l \cdot \text{mark}$
(17) **endif**
(18) **else**
(19) **while not** $(T_t < T_r < T_{t+1})$
(20) $t{-}{-}$
(21) **endwhile**
(22) **if** $(\mathbf{D(R_i)} > \mathbf{threshold})$
(23) $\text{UI}_i = {}_t^{t+1}HT_k^l \cdot \text{UI}$
(24) $\text{mark}_{\text{old}} = {}_t^{t+1}HT_k^l \cdot \text{mark}$
(25) **else**
(26) $\text{mark}_{\text{old}} = {}_t^{t+1}HT_k^l \cdot \text{mark}$
(27) **endif**
(28) **endif**
(29) send reconstruction request with mark_{old} and $P_j \cdot srcIP$ to upstream router by UI_i
(30) **else**
(31) this router is the nearest border router to the attacker
(32) **endif**
(33) **else**
(34) $\text{mark}_{\text{old}} = P_j \cdot mark/(D(R_i) + 1)$
(35) send reconstruction request with mark_{old} and $P_j \cdot srcIP$ to upstream router by UI_i
(36) **endif**
end

Algorithm 2: Our path reconstruction.

Algorithm 2) and checks the table's time field to find the table that logs the mark and the interface number. Last it follows l to retrieve mark_{old} and UI_i from table k; compare lines 13-14 and 23-24 (if table is full) in Algorithm 2. But if $l = 0$, it means this router is the source router; compare line 31 in Algorithm 2.

In Figure 2, we use dotted lines to indicate the path reconstruction of packet P_1. It shows R_3 receives the path reconstruction request in which $T_0 < T_r < T_1$. After R_2 receives $P_1 \cdot$mark, $P_1 \cdot$srcIP, and T_r from R_3, it uses $P_1 \cdot$mark to compute $\text{UI}_2 = -1$, which means the packet has been logged on R_2. Since the router's degree is three, the router divides $P_1 \cdot \text{mark}$ by $(D(R_2) + 1)$ and retrieves the pair of data $l = 11$ and $\text{UI}_2 = 2$. As $l \neq 0$, this router is not the source router. Then it uses $P_1 \cdot$srcIP to compute the log table's number $k = 0$. According to $T_0^0 < T_r < T_0^1$, the router retrieves the mark from ${}_0^1HT_0^{11}$ and replace $P_1 \cdot \text{mark}$ with the retrieved mark. Last, $P_1 \cdot$mark is sent to R_1 to continue the traceback until the attack source is identified.

3. Performance Analysis

In this section, we will introduce our simulation environment and how we determine log table size and the threshold.

3.1. Simulation Environment. To simulate the Internet topology, we use the skitter project topology distributed by CAIDA [29] as our sample data set of the Internet. The data set consists of paths to a specific host of the topology. We analyze CAIDA's skitter data and choose only 197,003 complete paths for our network topology. The analysis results are illustrated in Figure 3. Total number of its routers is 130,267; its average

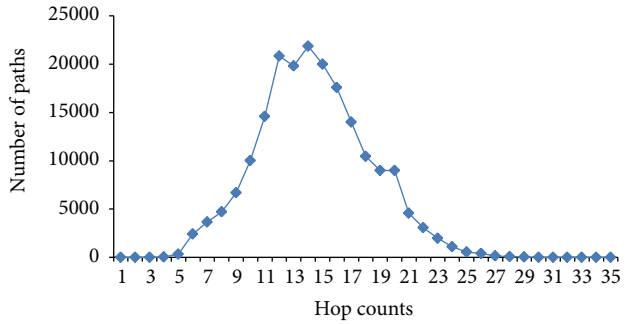


Figure 3: Distribution of path length.

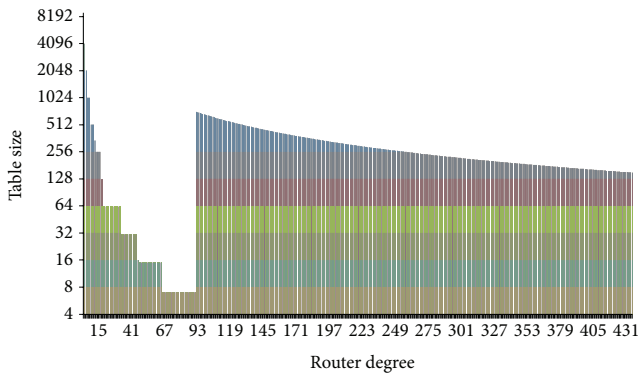


Figure 4: Relation between router degree and table size.

hop count of paths is 14.42; and its average upstream degree is 2.63.

3.2. Relation between Router Degree and Table Size. As shown in Figure 4, when a router's degrees are below 90, the table's maximum size decreases quickly with the increase of router degrees. It is because when the router's degrees are under the threshold our scheme marks the router's interface number UI_i into the fixed-size packet header. When UI_i's maximum number increases with the degree, the index value has to decrease. It means the maximum size of the table decreases too. When the degrees are over 90, UI_i has to be logged in the table and therefore the marking field allows a higher index value. This is why a log table's maximum size rises drastically when the router's degrees are larger than 90. For example, if a router's degrees are 66, the maximum size of its log tables is 7. The router can only accommodate log tables whose size ranges from 4 to 7. If the degrees are 91, the router allows log tables whose maximum size is 712. Then the table's maximum size decreases with the increase of degrees.

3.3. Relation among Threshold, Table Size, and Logging Times. Since the logging algorithm is determined by the threshold of a router's degree, we send 10 million packets to the network to find out the maximum storage requirement of our scheme. In the simulation, we send the packets to a randomly chosen path and count the logging times on the largest router in CAIDA's dataset, whose degree is 434. The result is shown in Figure 5. When the threshold is set as 10, the table has 8

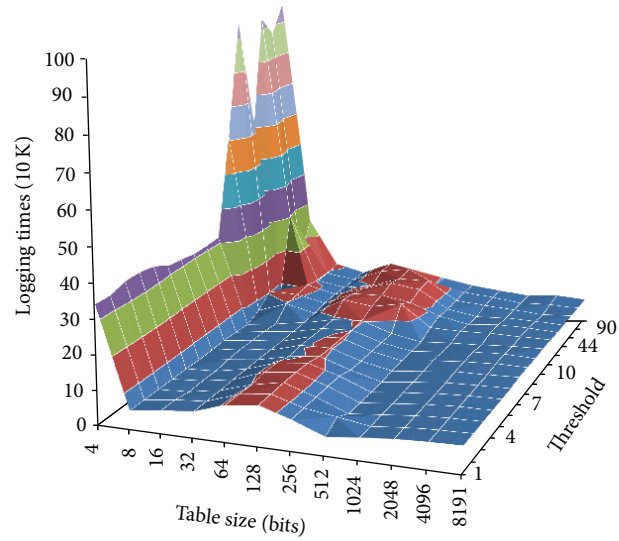


Figure 5: Relation among threshold, table size, and logging times.

entries (256 bits) and the router has the fewest logging times. Therefore, we suggest that routers set the table's maximum size as 256 bits and the threshold 10.

3.4. Storage Requirements. In this subsection, we compare the logging times and storage requirements of our scheme with those of other traceback schemes RIHT [24] and HAHIT [26]. Figure 6 shows the average logging times of our scheme and of RIHT and HAHIT when we send 10–40 million packets to the network. As depicted in the figure, compared with HAHIT our scheme requires fewer logging times and our logging times do not increase with the number of packets. Since the size of a marking field is fixed, a large index will leave a small space for the packet mark. And this can cause higher logging frequency. But our scheme requires an interface number to be logged if it exceeds the threshold value. In doing so, we can effectively lower the logging frequency. Furthermore, our logging frequency does not linearly increase with packet numbers because the index value of our scheme is bounded by the threshold. As for RIHT, it has lower logging frequency than our scheme because its marking field requires 32 bits and therefore has lower chance of overflow. But this advantage declines with the increase of hops between source and destination. Besides, RIHT has higher false positive/negative rates.

Figure 7 shows the average storage requirements on each router. While HAHIT requires 1500 KB and RIHT 320 KB in average for their logging, ours only needs 16 KB. Although our logging frequency is slightly higher than RIHT's, our scheme cuts its storage requirement by 95%. It is because our log tables allow more entries on the routers whose degrees are under the threshold value 10, and because we do not use fixed-size tables. Thus, we can avoid the paths that have been logged twice in the tables.

Figure 8 compares our scheme's maximum storage requirements with HAHIT's and RIHT's in different packet numbers. In these schemes, the maximum storage occurs

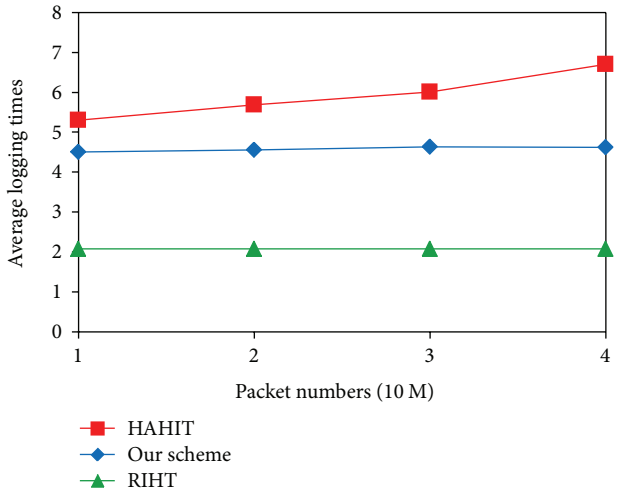


Figure 6: Comparison of logging times.

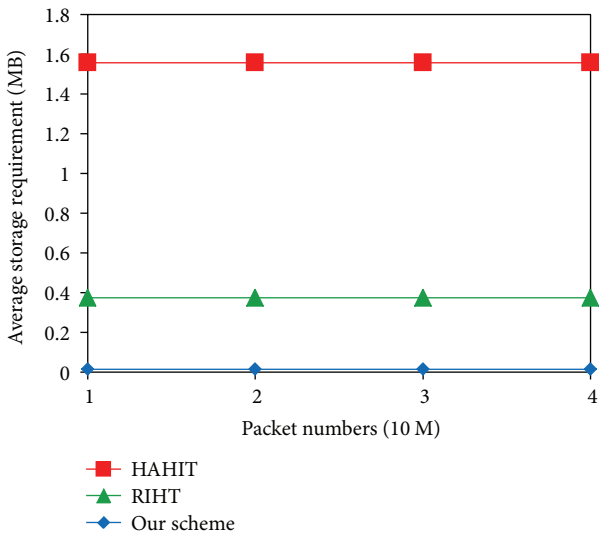


Figure 7: Comparison of average storage requirements.

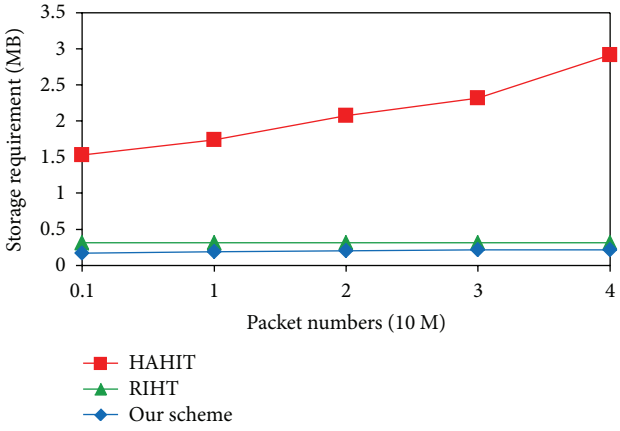


Figure 8: Comparison of the maximum storage requirements.

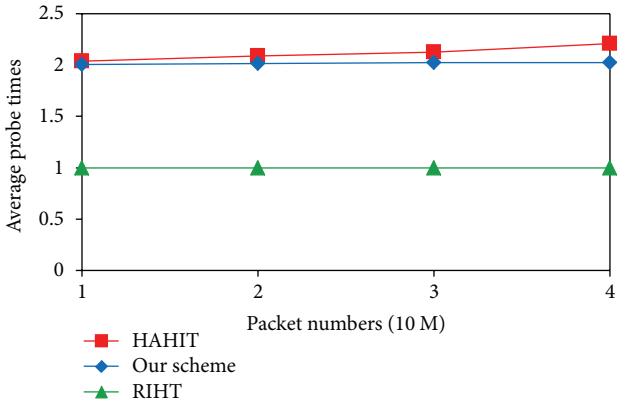


Figure 9: Average probe times in path reconstruction.

on the router that has the largest degrees because it will have the highest logging frequency. Figure 8 shows our storage requirements and RIHT's storage requirements do not linearly increase with packet numbers because they have constant logging frequency. The maximum storage requirement of our scheme is 220 KB, about 2/3 of RIHT's, that is, 320 KB, because our scheme has smaller index values.

3.5. Computational Loads. Since our scheme, HAHIT, and the RIHT use similar approaches to log packets, they have almost equal computation loads in logging. Thus, we analyze the computational loads of their path reconstruction only in this subsection. Besides, these three methods use packet marks to compute their log table's index value and then use the value to compute a new mark. Therefore, we analyze and compare the computation times required for each scheme to generate a valid index value. Figure 9 shows RIHT needs only one computation to find a logged path because it has just one table. But HAHIT and our scheme have to find the log table first and then the index value, hence two probes at least. The probe numbers will slightly increase if we take into account the probes of those filled-up tables. Figure 9 shows that compared with HAHIT, our average probe times are closer to 2. It is because the routers in our scheme with $D(R_i) > 10$ use larger log tables to lower the chance of tables being filled up.

3.6. False Positive/Negative Rates. When a router is mistaken as an attack router, we call it a "false positive." When we fail to trace back to an attacker, we call it a "false negative." Besides, a router's storage capacity is limited. If packet numbers exceed a router's storage limit, its log tables have to be refreshed. Then false negatives may occur in path reconstruction. In our simulation, each time we only choose one host from CAIDA's dataset to act as an attacker sending one packet, and then we repeat the process 10–40 million times, so as to try the false positive/negative rates in RIHT, HAHIT, and our scheme. Because our scheme, HAHIT, and RIHT have low storage requirements, routers can keep the path info for a long time and therefore do not need to refresh their log tables under flood attacks, hence 0 false negatives. RIHT, however,

Table 3: Comparison results.

	Marking size	Storage requirement	Major contributions
RIHT	32 bits	320 KB	RIHT bounds the storage requirements but may be prone to a fragmented traffic.
HAHIT	16 bits	1500 KB and above	HAHIT prevents fragmentation issues by changing the marking field from 32 bits to 16 bits, but its storage requirement is higher and its table search is inefficient.
Our scheme	16 bits	220 KB	Our scheme sets a threshold to determine whether to log UI or to mark UI in a packet, so as to solve the storage and fragmentation issues at the same time.

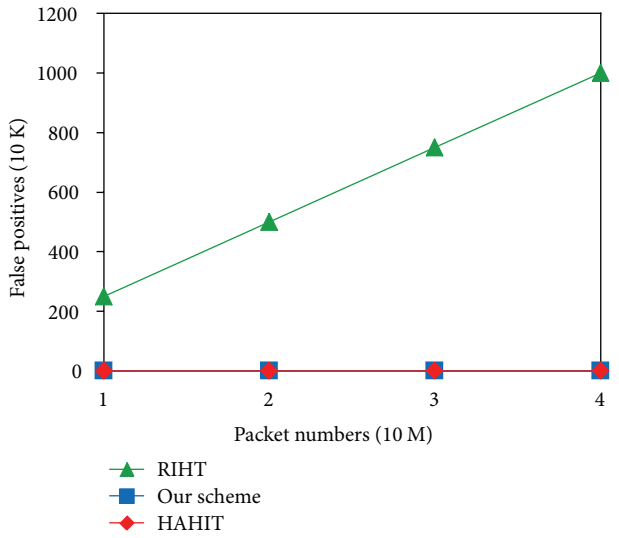


Figure 10: Comparison of false positives.

requires 32 bits for marking and consequently cannot make 0 false positives. Its false positive rates equal its fragmentation rates 0.25% [26]. Since our scheme and HAHIT use 16-bit marking fields, our ID fields can remain intact in packet fragmentation. Thus, both of the two schemes can make 0 false positives. As shown in Figure 10, RIHT's false positives rise with the increase of packet numbers, while ours and HAHIT's still remain 0.

To sum up, we list the comparison results in Table 3.

4. Conclusion

In this paper we propose a 16-bit single packet IP traceback scheme. Compared with current hybrid single packet traceback schemes, it has the lowest maximum storage requirement, which means the compulsory storage requirement for a router to support our hybrid single packet traceback. Besides, our scheme stores user interface information on small-degree routers, so that it can have small average storage requirements. Because the required storage for our routers' log tables is bounded by route numbers, it does not grow with the number of passing packets. Our experiment also shows that our scheme cuts RIHT's average storage requirement (320 KB) by 95%. Despite one more probe required for our path reconstruction, if compared with RIHT's, our traceback can achieve high accuracy (zero false positive/negative rates) because it complies with IPsec preventing marked packets from being dropped by routers during logging. Last, our scheme guarantees reassembly of fragmented packets at the destination.

Notations

R_i: $\{R_1, R_2, \ldots, R_i, \ldots, R_x\}$, routers in a network

$D(R_i)$: The degree of R_i, that is, the number of R_i's adjacent routers

P_j: A packet j

UI_i: The upstream interface number of router R_i

$P_j \cdot \mathrm{mark}$: Marking field of P_j

$P_j \cdot \mathrm{srcIP}$: P_j's source IP

m: A log table with m entries

$H_{\mathrm{table}}(\cdot)$: A hash function with hashed value ranging from 0 to $n-1$, where n denotes the number of log tables

$[T_k^s, T_k^f)$: T_k^s denotes log table k's created time; and T_k^f denotes k's full time, where s, $f = 0 \ldots, t, \ldots, \infty$. If $s = 0$, it means the first table of k. If $f = \infty$, it means the table has not been filled up

${}_t^{t+1}HT_k^l$: The log table k, in which l denotes the table's index value ranging from 0 to $m-1$. If $l = 0$, it refers to the source router. A packet's marking field is logged at $HT_k^l \cdot \mathrm{mark}$. UI_i is logged at $HT_k^l \cdot \mathrm{UI}$. ${}_t^{t+1}HT_k$ denotes table k's creation time is T_t and fill-up time is T_{t+1}. If the fill-up time is not specified, table k must be currently in use and is denoted as ${}_t^{\infty}HT_k$

%: Modulo operation.

Conflict of Interests

The author declares that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This research was supported by the Ministry of Science and Technology of Taiwan under Grant no. MOST103-2221-E-033-050.

References

[1] A. Hussain, J. Heidemann, and C. Papadopoulos, "A framework for classifying denial of service attacks," in *Proceedings of the Conference on Computer Communications ACM (SIGCOMM '03)*, pp. 99–110, Karlsruhe, Germany, August 2003.

[2] CERT, *IP Denial-of-Service Attacks*, Software Engineering Institute, Carnegie Mellon University, 1997, http://www.cert.org/historical/advisories/CA-1997-28.cfm.

[3] A. Yaar, A. Perrig, and D. Song, "FIT: fast Internet traceback," in *Proceedings of the IEEE INFOCOM 2005*, pp. 1395–1406, March 2005.

[4] D. X. Song and A. Perrig, "Advanced and authenticated marking schemes for IP traceback," in *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 2, pp. 878–886, April 2001.

[5] H. Tian, J. Bi, X. Jiang, and W. Zhang, "A probabilistic marking scheme for fast traceback," in *Proceedings of the 2nd International Conference on Evolving Internet (INTERNET '10)*, pp. 137–141, Valcencia, Spain, September 2010.

[6] J. Liu, Z.-J. Lee, and Y.-C. Chung, "Dynamic probabilistic packet marking for efficient IP traceback," *Computer Networks*, vol. 51, no. 3, pp. 866–882, 2007.

[7] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Network support for IP traceback," *IEEE/ACM Transactions on Networking*, vol. 9, no. 3, pp. 226–237, 2001.

[8] V. Paruchuri, A. Durresi, and S. Chellappan, "TTL based packet marking for IP traceback," in *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM '08)*, pp. 1–5, New Orleans, La, USA, December 2008.

[9] S. Saurabh and A. S. Sairam, "Linear and remainder packet marking for fast IP traceback," in *Proceedings of the 4th International Conference on Communication Systems and Networks (COMSNETS '12)*, Bengaluru, India, January 2012.

[10] A. Belenky and N. Ansari, "Accommodating fragmentation in deterministic packet marking for IP traceback," in *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM '03)*, pp. 1374–1378, December 2003.

[11] A. Belenky and N. Ansari, "IP traceback with deterministic packet marking," *IEEE Communications Letters*, vol. 7, no. 4, pp. 162–164, 2003.

[12] A. Belenky and N. Ansari, "Tracing multiple attackers with deterministic packet marking (DPM)," in *Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and signal Processing (PACRIM '03)*, vol. 1, pp. 49–52, August 2003.

[13] V. K. Soundar Rajam and S. Mercy Shalinie, "A novel traceback algorithm for DDoS attack with marking scheme for online system," in *Proceedings of the International Conference on Recent Trends in Information Technology (ICRTIT '12)*, pp. 407–412, April 2012.

[14] H. C. Tian, J. Bi, and P. Y. Xiao, "A flow-based traceback scheme on an AS-level overlay network," in *Proceedings of the 32nd International Conference on Distributed Computing Systems Workshops (ICDCSW '12)*, 2012.

[15] A. C. Snoeren, C. Partridge, L. A. Sanchez et al., "Single-packet IP traceback," *IEEE/ACM Transactions on Networking*, vol. 10, no. 6, pp. 721–734, 2002.

[16] L. Zhang and Y. Guan, "TOPO: a topology-aware single packet attack traceback scheme," in *Proceedings of the IEEE International Conference on Security and Privacy in Communication Networks (SecureComm '06)*, pp. 1–10, September 2006.

[17] E. Hilgenstieler, E. P. Duarte Jr., G. Mansfield-Keeni, and N. Shiratori, "Extensions to the source path isolation engine for precise and efficient log-based IP traceback," *Computers and Security*, vol. 29, no. 4, pp. 383–392, 2010.

[18] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[19] C. Gong and K. Sarac, "A more practical approach for single-packet IP traceback using packet logging and marking," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 10, pp. 1310–1324, 2008.

[20] K. H. Choi and H. K. Dai, "A marking scheme using huffman codes for IP traceback," in *Proceedings on the 7th International Symposium on Parallel Architectures, Algorithms and Networks (I-SPAN '04)*, pp. 421–428, May 2004.

[21] S. Malliga and A. Tamilarasi, "A hybrid scheme using packet marking and logging for IP traceback," *International Journal of Internet Protocol Technology*, vol. 5, no. 1-2, pp. 81–91, 2010.

[22] S. Malliga and A. Tamilarasi, "A proposal for new marking scheme with its performance evaluation for IP traceback," *WSEAS Transactions on Computer Research*, vol. 3, no. 4, pp. 259–272, 2008.

[23] Y. L. Wang, S. Su, Y. Yang, and J. Ren, "A more efficient hybrid approach for single-packet IP traceback," in *Proceedings of the 20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP '12)*, pp. 275–282, Garching, Germany, February 2012.

[24] M. H. Yang and M. C. Yang, "RIHT: a novel hybrid IP traceback scheme," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 2, pp. 789–797, 2012.

[25] N. Lu, Y. Wang, F. Yang, and M. Xu, "A novel approach for single-packet IP traceback based on routing path," in *Proceedings of the 20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP '12)*, pp. 253–260, February 2012.

[26] M. H. Yang, "Hybrid single-packet IP traceback with low storage and high accuracy," *The Scientific World Journal*, vol. 2014, Article ID 239280, 12 pages, 2014.

[27] W. John and S. Tafvelln, "Analysis of internet backbone traffic and header anomalies observed," in *Proceedings of the 7th ACM SIGCOMM Internet Measurement Conference (IMC '07)*, pp. 111–116, October 2007.

[28] "Security assessment of the internet protocol version 4," IETF RFC 6274, Internet Engineering Task Force (IETF), 2011.

[29] CAIDA, "CAIDA's skitter project," http://www.caida.org/tools/skitter/.