

**A NEW MARKING SCHEME USING HUFFMAN CODES**

**FOR IP TRACEBACK**

By

**KYU HYONG CHOI**

**Bachelor of Engineering**

**Chonbuk National University**

**Chonju, Korea**

**1988**

Submitted to the Faculty of the  
Graduate College of the  
Oklahoma State University  
In partial fulfillment of  
the requirements for  
the Degree of  
**MASTER OF SCIENCE**  
May, 2003

*Oklahoma State University Library*

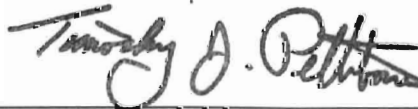
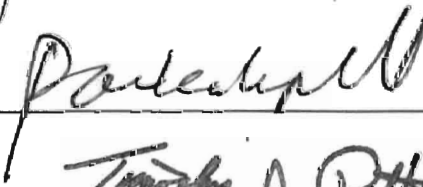
A NEW MARKING SCHEME USING HUFFMAN CODES

FOR IP TRACEBACK

Thesis Approved:



Thesis Advisor



Dean of the Graduate College

## ACKNOWLEDGMENTS

I truly would like to express my appreciation to my advisor, Dr. H. K. Dai, for his intelligent supervision, great guidance, wonderful inspiration and kind friendship. My sincere appreciation extends to my other committee members, Dr. J. P. Chandler and Dr. N. Park for their excellent supervision and guidance. I also would like to thank all my computer science department colleagues for their kind co-operations.

Special thanks go to my parents in Korea for their sacrifice not to give up enthusiasm to educate their children. I also would like to express my debt to the Korean government for the support for my study in the United States. I especially would like to thank to my wife for her great assistance and patience during my study.

Finally, I would like to thank the Computer Science Department for its qualified and advanced education.

## TABLE OF CONTENTS

Chapter	Page
I . INTRODUCTION.....	1
II . LITERATURE REVIEW .....	4
2.1 ICMP Traceback Messages.....	4
2.2 Link Testing .....	4
2.3 Marking.....	5
2.4 Logging.....	7
III . A NEW MARKING SCHEME USING HUFFMAN CODES.....	9
3.1 Encoding of Marking Field .....	12
3.2 Storing Marking Fields at Intermediate Routers.....	14
3.3 Traceback Procedure .....	16
3.4 Representations of Link Numbers .....	18
3.5 Organization of Link Tables .....	20
3.6 Encoding of Marking Field in the IP header.....	21
3.7 The Message Digest Algorithm.....	21
3.8 Compromised Intermediate Routers.....	24
IV . SIMULATION .....	25
4.1 Average Lengths of Huffman Codes and Link Sequences .....	26
4.2 Memory Requirement for Routers to Store Marking Fields.....	27
4.3 Comparison with Other Methods .....	40
V . PRACTICAL ISSUES FOR THE NEW SCHEME.....	42
VI . CONCLUSION AND FUTURE WORK .....	44
REFERENCES .....	46
APPENDIX (Source Codes of Simulation Program).....	48



## LIST OF FIGURES

Figure	Page
Attacking path .....	10
Link tables for routers.....	11
Encoding of the marking field.....	12
Marking procedure at a router with a packet P.....	13
An example of marking.....	15
Traceback procedure at a victim v with a packet P.....	17
An example of distribution and corresponding codes with degree 5 .....	18
The Huffman tree for unequal distribution.....	19
The Huffman tree for equal distribution.....	19
Structure of link tables .....	20
The fields of an IP packet (adopted from [1]).....	23
Average length of Huffman codewords (table).....	26
Average length of Huffman codewords with degrees 2,3,4,5, and 6 (graph).....	30
Average length of sequence of link codes with unequal distribution .....	31
Average length of sequence of link codes with equal distribution .....	32
Average count of savings of 32-bit marking field with unequal distribution .....	33
Average count of savings of 32-bit marking field with equal distribution .....	34
Average count of savings of 16-bit marking field with unequal distribution .....	35
Average count of savings of 16-bit marking field with equal distribution .....	36
Hop distribution (adopted from [10]) .....	37
Average count of savings of 32-bit marking field with average degrees 3 and 4.....	38
Average count of savings of 16-bit marking field with average degrees 3 and 4.....	39

## NOMENCLATURE

IP	Internet Protocol
IP Traceback	Tracing an IP packet back to its source
ICMP	Internet Control Message Protocol
DOS	Denial of service
DDOS	Distributed denial of service
ISP	Internet service provider
XOR	Bitwise Exclusive OR operation
MD	Message digest
Degree	Number of adjacent routers

## INTRODUCTION

The Internet has become indispensable and plays an important role in our life today, in many things we depend on the Internet. While the Internet gives us easy access to almost all open information, convenience and promptness, we are exposed to the various problems of security, usually called cyber crimes.

One of the problems is denial of service (DoS) attack. DoS attacks are different from system penetration attacks to steal information or destroy system in that DoS attacks consume the resources of target host or network by flooding with an amount of anonymous packets, thereby preventing legitimate access to the target host or network, resulting in loss of transactions with clients and credibility. Many well known Web sites like CNN, eBay, Yahoo!, and Amazon have suffered from distributed denial of service (DDoS) attacks. In DDoS attacks, the attacker uses a number of compromised hosts residing on different networks to intensify the flooding and make it hard to detect the attacker. Several automated DDoS attack tools such as Stacheldraht and TFN have been developed and (D)DoS attacks have become more prevalent recently due to the relative ease of acquiring and executing such attacking tools and their near untraceability to the attacker. In (D)DoS attacks attackers hide their real identity by forging the source Internet Protocol (IP) addresses of attacking packets, and they generate the addresses randomly.

If we had a mandatory mechanism to prevent the use of incorrect source address, then we would never hear about (D)DoS attack. Unfortunately we do not have any and the anonymous nature of the IP protocol makes it difficult to identify the true source of an IP packet if the source uses fake address. Many routers use ingress filtering [4] to limit source address of IP packets coming from the stub network to addresses belonging to that network. Each router is configured to block packets that arrive with illegitimate source addresses. This technique is most feasible in customer networks or at the border of Internet service providers (ISPs) where address ownership is relatively unambiguous and traffic load is low. However, some existing services such as network address translators (NATs), mobil IP and unidirectional link technology for hybrid satellite architectures depend on source address spoofing. Therefore it is difficult to prevent (D)DoS attacks fundamentally but there are two ways to deal with (D)DoS attacks, the one is to detecting and discarding attacking packets on the way to their destinations, and the other is IP traceback to find the real source of attacking packets and then possibly make the attacker (criminal person) responsible.

Even though it is not easy to detect the actual attacker (host and person), if a victim could find the path of attacking packets in real-time, it would be much easier to quickly stop the attack, and the possession of capability to trace back would somewhat deter attackers from launching (D)DoS attacks. The problem of traceback of spoofed packets has become a topic as a measure against (D)DoS attacks, and it will remain a topic in the Internet world. We believe that it is worthy of study of the methods of IP traceback for the analysis of packet routing.

Chapter II provides brief explanation of existing techniques and chapter III proposes a new technique that uses Huffman codes to mark packets with router's information as packets traverse routers during the journeys to reach their destinations. In Chapters IV and V simulation results and practical issues are presented.

## RELEVANT WORK

Various ideas for IP traceback have been proposed and some of them are practically used to determine the path that attacking packets pass through. We can basically categorize them into link testing, marking, logging, and Internet Control Message Protocol (ICMP) traceback messages

### 1. ICMP Traceback Messages

When a router forwards packets it samples packets with a low probability, and if a packet is sampled, the router creates ICMP traceback message with the content of the sampled packet and router's information, and send the messages to the destination of the sampled packet. With enough traceback messages from enough routers along the path, the attacking path can be determined.

### 2. Link Testing

The link testing tests all possible upstream routers of a router that is already known to be carrying the attacking packets to find out from which one the packets are coming in. For example, in Figure 1 at router m the only possible upstream is 1 (k), and at router k the possible upstream links are 1 (i), 3 (n), 4 (l) and 5 (o). After link 1(i) has been determined by link testing at router k, the upstream links 1(g) and 3(j) of router i will be tested. We can test a possible upstream link by dropping all packets addressed to the victim for a second or so and seeing if

there is a break in incoming packets at the victim, or by flooding a link [3] with a large burst of traffic, the packets traveling across the flooded link will have an increased probability of being dropped, and then observing the changes in the rate of attacking packets. Another test is by using the feature called input debugging which make it possible to filter particular packets on some egress port and determine which ingress port they arrived on. The victim creates a signature consisting of common features contained in all attacking packets and sends a query including the signature to each router along the path as routers are determined hop by hop from the closest router to the victim. Drawbacks are that tests can only be done during an ongoing attack, and cooperation and attention of intervening routers' operators are required. Furthermore, the execution of a link testing could be another DoS attack toward the link that is being tested.

### 3. Marking

In this method routers append or add their IP address information to the packets, and the victim can construct the attacking path by examining the added address information. The naive marking is Node Append in which every router append its IP address to all packets it forwards consequently increasing packet size

#### 3.1 Node Sampling

To reduce the size of packet and the overhead of the Node Append, each router samples packets with some probability to mark the packets with its address. The address written by a router may be overwritten by another router that a packet passes through later. As the distance between a router and the victim increases, the victim needs to receive more packets to receive a

packet marked by the router.

### 3.2 Edge Sampling [5]

In this sampling, a packet has three marking fields: start, end and distance. Start and end will denote an edge between two adjacent routers on the path, and distance will show the distance from the edge to the victim. If a router decides to mark a packet according to the some probability, then it marks the start field with its IP address, and set the distance to zero. If it decides not to mark the start field and distance is zero, then it marks the end field with its IP address, thus representing an edge between itself and its upstream router. Whenever the router does not mark the start field it increments the distance. This method is very robust to multiple attacks but requires additional space in the packet header compared to Node sampling.

### 3.3 XOR-Encoding of Edge Sampling [5]

To reduce the space required by edge sampling that requires three marking fields, packets have two marking fields called edge (start) instead of two (start and end), and a distance field. The field edge will be the result of operation XOR of two adjacent routers addresses. Marking decision procedure is same as edge sampling except when router marks the end field it does operation XOR with its IP address and the value of edge (start) field and then write the result to the edge field.

### 3.4 Advanced Markings of XOR Encoding [2]

As an advanced marking scheme of sampling, this scheme uses Identification field of an IP packet for marking. The field is divided into a 5-bit distance field and an 11-bit edge field, and



instead of using IP addresses themselves it uses the outputs of hash functions with IP addresses as inputs. Two different hash functions are used for start and end IP address.

Another advanced marking scheme is using two sets of hash functions, each set is for start and end IP address respectively, and there are three fields (flag identification, distance and edge) for marking. A hash function requires two parameters (IP address and flag identification) as inputs. The purpose of flag identification is to specify which hash function to use in a hash set, and the meaning of other fields are same as in XOR-encoding. When an end router calculates hash value it must use the same flag identification as the preceding (start) router used.

#### 4. Logging

In this method routers stores information, for some period of time because of the limit of space, about all packets they forward. When a victim traces back a packet, the victim checks all stored information in all adjacent routers to determine which one forwarded the packet, after the one has been determined then checks again all adjacent upstream routers to the determined one.

##### 4.1 Hash based IP Traceback [1]

In this approach they devised a system called Source Path Isolation Engine (SPIE) consisting of Data Generation Agents (DGAs), SPIE Collection and Reduction Agents (SCARs) and a SPIE Traceback Manager (STM) for a network which consists of many sub-networks and routers. Each router has a DGA associated with it. The DGA produces a message digest of each packet as it departs the router, and stores the digests for some period of time, and the

digests can be transferred to a SCAR for a long term storage and analysis. SCARs are responsible for a particular region of network, serving as data concentration points for several routers. The STM controls the whole SPIE system and is the interface to the intrusion detection system or other entity requesting a traceback.

A traceback begins when a traceback request arrives at STM, STM sends a query consisting of packet, egress point (the last router packet passed), and time of receipt to a SCAR responsible for the victim's region of the network. The SCAR responds with a partial attack path and the packet as it entered the region (it may have been transformed, possibly multiple times, within the region), and a node at the edge of the SCAR's region. STM sends a query to another SCAR that is abutting that edge node. This process continues until all branches of attack path terminate, either at a source within a region managed by a SCAR, or at the edge of the entire SPIE system.

When a SCAR determines a router that forwarded a packet, the SCAR searches the digests stored in its DGA for the digest of the packet and if the digest is found in a DGA associated with a router, the packet is assumed to have passed through the router. This approach requires huge space to store digests of all packets and the management of whole SPIE system is complex.

### NEW MARKING SCHEME USING HUFFMAN CODES

This new idea utilizes the following facts. First, routers are able to know which physical network interface port packets arrive on, this ability is used in ingress filtering and input debugging of routers. Second, each router is connected with not so many adjacent routers, in a router-level Internet map the average degree (the number of neighboring routers of a router) is 3.15 [8].

There are two differences in the new method from other marking methods. Firstly when a router marks a packet with address information, the information is not of the router that is marking but of a router which sent the packet to the current router, and secondly it uses a special table called link table, which shows all the links between the router and its adjacent routers. The router appends to the marking field a Huffman codeword representing the link number of the link (router) through which the packet arrived.

When the marking field of a packet becomes short of space left to append the corresponding Huffman codeword for the link number, the router stores the content of the marking field with a message digest of the packet into the router's local memory, and then clears the field and appends the codeword. The stored link sequence can be retrieved via the message digest of the packet from the intermediate router during an IP traceback procedure.

Figure 1 illustrates an attacking scenario, in which the attacking path is [a,e,g,i,k,m] and the sequence of link numbers is [1,3,2,1,1,1], and corresponding sequence of Huffman codewords is [0,11,10,0,00,0] according to the link tables in Table 1. Decoding process can be optimized by appending Huffman codes in reverse order, therefore the actual sequence of Huffman codewords is [0,11,01,0,00,0].

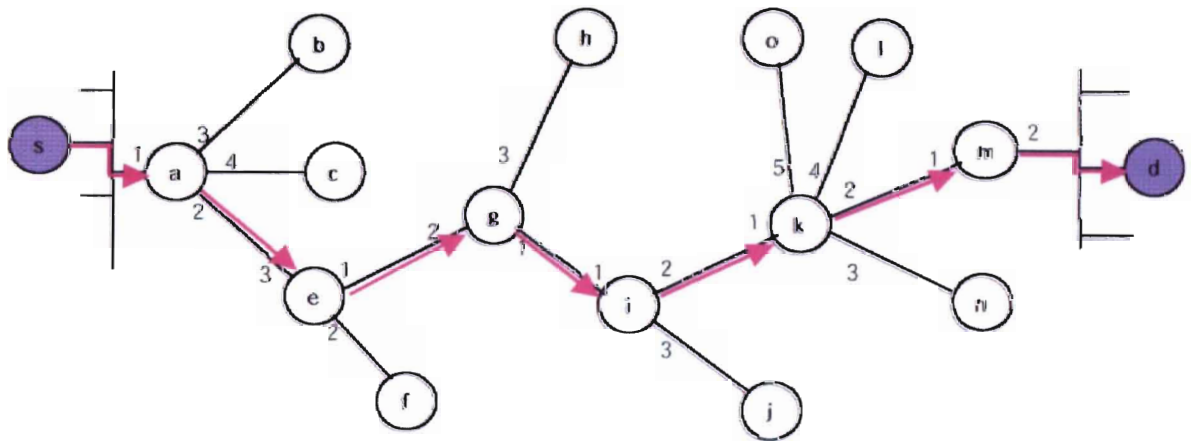


Figure 1. Attacking path: (s).a.e.g.i.k.m.(d) (s: source host (attacker), d: destination host (victim)).

Link #	Router	Link Code
1	a:local	0
2	e	10
3	b	110
4	c	111

Link table  
for router a

Link #	Router	Link Code
1	g	0
2	f	10
3	a	11

Link table  
for router e

Link #	Router	Link Code
1	i	0
2	e	10
3	h	11

Link table  
for router g

Link #	Router	Link Code
1	g	0
2	k	10
3	i	11

Link table  
for router i

Link #	Router	Link Code
1	i	00
2	m	01
3	n	10
4	l	110
5	o	111

Link table  
for router k

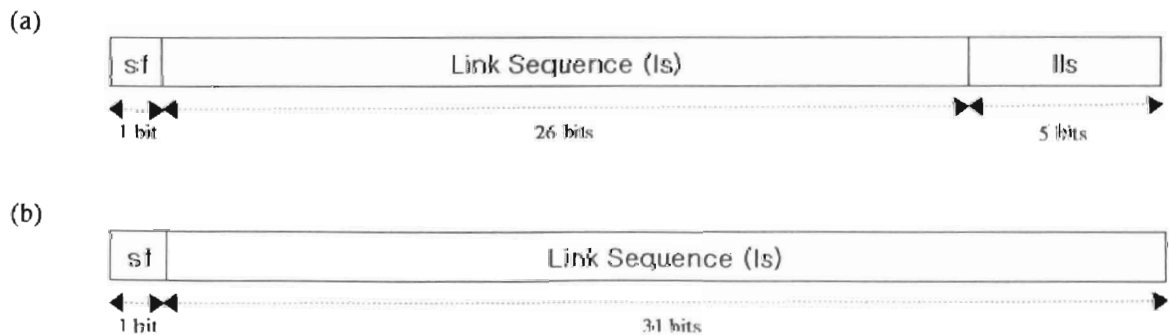
Link #	Router	Link Code
1	k	0
2	m:local	1

Link table  
for router m

Table 1. Link tables for routers a, e, g, i, k and m of the attacking path of Figure 1. Link codes are of variable length, and a, b, c, ..., n are abbreviations for IP addresses of the routers.

## 1. Encoding of Marking Field

Figure 2 shows encodings of 32-bit marking field, in format (a) marking field is divided into a 1-bit saved flag (*sf*), a 26-bit link sequence (*ls*) and a 5-bit length of link sequence (*lls*), in format (b) it is into a 1-bit saved flag (*sf*) and a 31-bit link sequence (*ls*). To reduce the possibility that the marking field has to be stored at intermediate routers' local memory, it is required to assign a longer field to the link sequence. So instead of using *lls* to specify the length of bitstring (sequence of link codes) in the field *ls*, we use bit 1 as a delimiter with leading 0s to designate the start position of the valid bitstring. When a packet passes through a router, *ls* is augmented with a codeword that represents a link through which the packet came in. Before appending the reversal of the codeword at the right end of *ls*, the router checks if there is enough bit-space left in *ls* to append the codeword by counting the leading 0s before the delimiter in *ls*.



*sf* : Saved flag    *lls* : Length of link sequence

Figure 2. Encoding of the marking field. Format (a) uses *lls* to specify the length of bitstring in *ls*, while format (b) uses a delimiter bit 1 at the leftmost of the valid sequence of link codes in *ls*.

### Marking procedure at a router:

Determine a link that packet **P** came from, and a Huffman codeword representing the link by consulting the link table.

```
if (sf == 1 and packet P(old_P) transformed into a different packet (new_P))
    then store MD(new_P):MD(old_P)(sf,ls) //store at local memory
        ls = 0x01 //clear ls by setting with 000...01

if (space_left < length(codeword)) //if not enough space left in link sequence(ls)
    then store MD(P):( sf,ls) //MD(p): message digest of packet p
        sf = 1, ls = 0x01 //marking field saved, ls cleared

Append codeword to ls // append codeword to the link sequence(ls)
```

Figure 3. Marking procedure at a router with a packet P.

## 2. Storing Marking Fields at Intermediate Routers

Because of the limited space of  $ls$  in the marking field we may not be able to store the complete link sequence of a path. After a router has determined that the bit-space left in the  $ls$  is not enough for appending a codeword, the router stores the contents of the marking field in its local memory, which is indexed by the message digest of the packet, denoted by  $MD(\text{packet})$ . After saving,  $ls$  will be cleared by setting with  $0x01$  having only delimiter bit 1 at the rightmost bit, and  $sf$  is set to 1 indicating that the marking field is stored.

### Possible packet transformations

If a packet undergoes a transformation after the marking has been saved (when  $sf$  is 1), then a router can not retrieve the stored marking field unless it knows the message digest of the packet before the transformation. Therefore when  $sf$  is 1 and a transformation happens, a router should store a pair of digests of old and new packets  $MD(\text{new Packet}):MD(\text{old Packet})$  along with the marking field, and clear  $ls$  by setting with  $0x01$ , but  $sf$  remains 1.



**Example of marking procedure:**

Router	In-Link #	Link Code	sf	Is	Saving of Marking Field
			0	0000001	
1	1	0	0	000001 <u>0</u>	
2	3	11	0	00010 <u>11</u>	
3	2	10	0	01011 <u>01</u>	
4	1	0	0	10110 <u>10</u>	
5	1	00	1	00001 <u>00</u>	MD(P <sub>4</sub> ):01011010
6	3	010	1	00010 <u>10</u>	MD(P <sub>6</sub> ):MD(P <sub>5</sub> )10000100
7	4	001	1	1010 <u>100</u>	
8	1	0	1	00000 <u>10</u>	MD(P <sub>7</sub> ):11010100

Assumption : Length of Marking Field is 8 bits, 1 bit for *sf*, 7 bits for *Is*  
 MD(P<sub>k</sub>) = Message digest of packet P at router k before marking

Figure 4. An example of marking: The first bit 1 in *Is* is a delimiter that indicates the start of sequence of the reversals of link codes. At router 5, 6 and 8 the marking field is saved. At router 6 *sf* is 1 and the packet transforms into a different new packet, therefore router 6 saves the marking field with a pair of digests of old and new packets, and then marks the marking field. Link codes are appended at the end of *Is* in reverse bit-order.

### 3. Traceback Procedure

Starting from a router that is directly connected with a victim, the victim can traceback a packet by decoding the link sequence ( $ls$ ) in the marking field of the packet. When decoding a codeword the victim consults the link table of current router to find the upstream router that forwarded the packet to the current router. After a codeword has been decoded,  $ls$  will be right-shifted times of the length of the decoded cordword. When  $ls$  become 1 (only with a delimiter at the rightmost bit) and  $sf$  is 1, the stored marking field should be retrieved via the message digest of the packet. Now the upstream router becomes current router and the traceback continues until  $ls$  becomes 1 and  $sf$  becomes 0.

### Traceback procedure at a victim with a packet P:

Starting at the closest router (current router) that the victim is connected directly with

```
While (1)
{
    Print current router

    Construct Huffman tree with the link table of current router

    Decode one Huffman codeword from the right end of ls by using the tree

    Find the router that the decoded codeword represents

    if (ls == 0x01 and sf == 1) // marking field is stored at current router's memory
        then retrieve MD(p):MD(pre_p)(sf,ls) or MD(p):( sf,ls)
            reset sf, ls with retrieved values

    if (ls == 0x01 and sf == 0)
        then break //stop traceback, no more link sequence to decode

    Set current router with the found router
}
```

Figure 5. Traceback procedure at a victim with a packet P.

#### 4. Representation of Links

To reduce the length of the marking field in the IP packet header and the times link sequence has to be stored in intermediate routers due to the lack of space left in the marking field during the marking procedure, we use Huffman codes, which is widely used to compress data by assigning shorter codewords to higher-frequency characters and longer codewords to lower-frequency characters, to represent the link numbers. For a router, each link between itself and one of its adjacent routers has a relative number (frequency) of packets coming into the router through the link, and using the frequencies of packets we can assign a Huffman codeword to each link. Table 2 shows an example where the number of links is five and the average number of bits to represent a link with unequal distribution is 2.04 while fixed-length representation requires 3 bits. Figures 6 and 7 illustrate two Huffman trees each with equal and unequal distribution of packets among 5 links of a router of Table 2.

Link Number	1	2	3	4	5
Unequal Distribution	45	34	10	8	3
Equal Distribution	255	255	255	255	255
Fixed-Length Codes	000	001	010	011	100
Huffman Codes for Unequal Distribution	1	00	011	0100	0101
Huffman Codes for Equal Distribution	110	111	00	01	10

Table 2. An example of distribution and corresponding codes for links with degree 5.

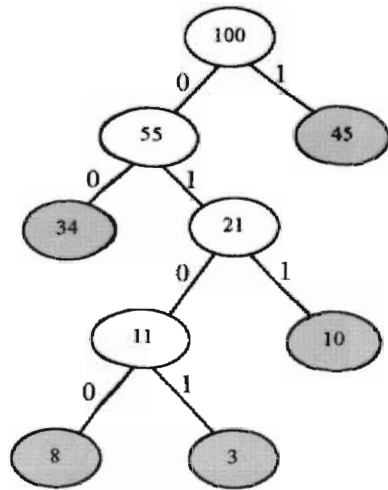


Figure 6. The Huffman tree for the unequal distribution of Table 2. Frequencies of packets arriving through each link are respectively 45, 34, 10, 8 and 3. The compression rate over the fixed-length representation is 61.2%  $((45 * 1\text{bit} + 34 * 2\text{bits} + 10 * 3\text{bits} + 8 * 4\text{bits} + 3 * 4\text{bits}) / (45 + 34 + 10 + 8 + 3) * 3\text{bits})$ .

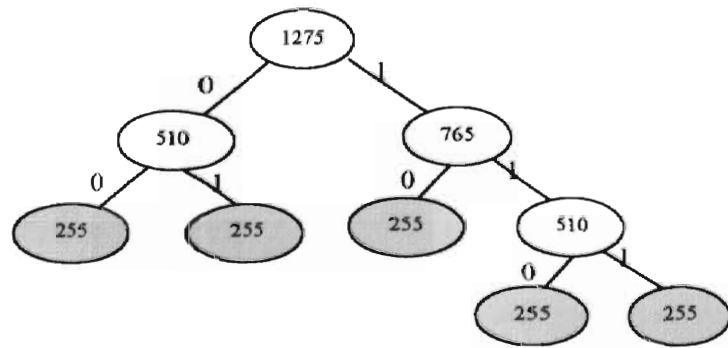


Figure 7. The Huffman tree for the equal distribution of Table 2. Frequencies of packets arriving through each link are all the same with 255. The compression rate over the fixed-length representation is 80%  $((20 * 2\text{bits} + 20 * 2\text{bits} + 20 * 2\text{bits} + 20 * 3\text{bits} + 20 * 3\text{bits}) / (45 + 34 + 10 + 8 + 3) * 3\text{bits})$ .

## 5. Organization of Link Tables

The link table of a router is a file that is supposed to be accessed by a victim to decode a Huffman codeword to find an upstream router on the attacking path. All routers must have agreed structure for their link tables.

Figure 8 shows a possible structure of a link table. In the structure, number of links is the number of adjacent routers directly connected with a router and the frequency of each link is the relative number of packets coming through the link. The number of links and frequencies of each link are represented by one byte for each, and IP addresses of routers are 4-bytes long.

Number of links (k)	Frequency of link 1	Frequency of link 2	...	Frequency of link k	IP address of link 1	IP address of link 2	...	IP address of link k
---------------------	---------------------	---------------------	-----	---------------------	----------------------	----------------------	-----	----------------------

Figure 8. Structure of link table.

## 6. Encoding of Marking Field in the IP Header

Some fields of IP header must be used as the marking field. The Option field of IP packet looks most adequate but in [5] Sabage uses the Identification field of IP header to store path information on account of that less than 0.25% of packets undergo fragmentation [7]. If the IP Identification field is used for marking then the original function (reassembling fragmented packets by inspecting the Identification field of packets) of the field will be impeded. And using the Option field is not supported practically because the Option field has rarely been used in reality and most of routers that are running currently in the Internet cannot handle the Option field. Even though they could handle the Option field there remain still other problems like increasing possibility of fragmentation of packets due to increased size by using the Option field, because basically the Option field does not have an assigned fixed length space in IP header as its name means literally. Therefore this study does not propose a certain field to use for the marking field.

## 7. The Message Digest Algorithm

When routers store a marking field it will index the marking field with a message digest of the packet. If we choose a message digest algorithm with longer output (64-bits or 128-bits) then routers need to have more memory space to store the digest with along a marking field. Using one of existing digest algorithms with adequate output length is the easiest way. There are many message digest algorithms and if we adopt MD5 [11] then we can use only 32 bits of 124 output bits, for example by selecting every forth bit of the output.

As explained later in the memory requirement section of the results of simulation, at a high-end router with capacity of 1 Tera bit/sec with 1-minute period of keeping of marking fields,

the number of 32-bit marking fields that has to be stored is 1200 Mega. Therefore with 32-bit marking field, the probability that a message digest collides with a stored one is  $1.2/4$  because there are 4 Giga digests with 32 bits long.

#### **Fields of IP packets to be used as input of the message digest algorithm**

When routers compute a digest  $MD(P)$  of a packet  $P$ , the input of the digest algorithm is not the whole packet, only some parts of the packet are used to reduce the processing time and because some fields of IP header changes as the packet passes through routers. In [1] as shown in Figure 9 TTL, ToS, Options, and Checksum of IP header will be masked out before digesting, and the first 8 bytes of the payload are used as input. But for the new marking scheme, in addition to the fields masked out in Figure 9, fields that are used for marking field are masked out too before digesting.



Version	Header Length	Type of Service	Total Length		
Identification			D	M	Fragment Offset
			F	F	
TTL	Protocol		Checksum		
Source Address					
Destination Address					
Options					
Payload ( First 8 bytes )					

Figure 9. The fields of an IP packet. Fields in gray are masked out before digesting, including the type of service, time to live (TTL), and IP options fields (adopted from [1]).

### **8. Compromised Intermediate Routers**

In a (D)DoS, there are possibly compromised intermediate routers on the path of attacking packets, and they could mess up or carefully manipulate the marking field of a packet. However compromised routers cannot affect the marking after them, and the victim can traceback correctly at least up to a compromised router that is the closest to the victim on the path.

## SIMULATION

Simulation has been done to see whether new idea works correctly and analyze mainly memory requirement of the new idea. To imitate a packet flow in the Internet, first a packet, not a real IP packet but a data structure having a marking field, is created, then this packet traverses a certain number of routers (hops). Before the packet reaches a router the router is created by assigning an IP address, a link table including degree (number of links), IP addresses of neighboring routers that links connect with the router, and distribution of packets (frequencies) among the links, then Huffman codes for the links are constructed by creating a Huffman tree using the distribution, and one of the links is chosen randomly assuming that the packet comes in through the chosen link. Finally it marks the packet with a Huffman codeword representing the chosen link.

When a packet reaches its destination, IP traceback of this packet may be accomplished from the last router. During a traceback, at each router, a Huffman tree is created with packet distribution, and one codeword is decoded from the marking field of the packet. With the decoded information the next upstream router's IP address is found consulting the link table of the current router. The traceback continues at the found router until there is no link sequence left in the marking field.

For the analysis of memory requirement of routers, all information about created routers and

Huffman codes for the links, such as distance, degree, length of Huffman codes, and length of complete link sequence were collected.

### 1. Average Lengths of Huffman Codes and Link Sequences

The average length of codewords increases in proportion to the average degree. Table 3 and Figure 10 shows the average length of codewords for degrees from 2 to 6. For average degree 3, the average length is 1.44 and 1.56 bits for unequal distribution and equal distribution respectively, and for average degree 4 it is 1.77 and 1.95 bits. In equal distribution all links of a router were given 255 the same frequency of incoming packets, and in unequal distribution the frequency of incoming packets through each link is differently given in the range of 1~255 by random.

Average Degree		2	3	4	5	6
Average Length of Codewords	Equal Distribution	1.00	1.56	1.95	2.23	2.47
	Unequal Distribution	1.00	1.44	1.77	2.03	2.26

Table 3. Average length of Huffman codewords with average degrees 2, 3, 4, 5, and 6.

Figures 11 and 12 show the average length of complete link sequence with unequal and equal packet distribution among links. As like average length of Huffman codes, the average length of link sequence increases in proportion to the average degree and the distance (hop). The

average length of link sequence with average degree 3, distance 16 and unequal packet distribution is 23.11 bits, and 24.95 bits with equal distribution.

## 2. Memory Requirement for Routers to Store Marking Fields

Memory requirement was analyzed for each 32-bit and 16-bit long marking field with equal incoming packet distribution among links because we cannot ensure that the routers' link tables are optimally tuned with the actual packet distribution. But the simulation has been done with each equal and unequal distribution. In equal distribution all the links were given the same frequency 255 of incoming packets, and in unequal distribution the links were given different frequencies in the range of 1~ 255 randomly.

### 32-bit marking field

Almost all paths are less than 32 hops and the average length of path (number of hops) is around 16 [9], and the average degree (average number of neighbors of a router) is slightly larger than 3 [8]. Therefore using 32 bits for marking field, with distance 16 hops and average number of links 3, the average length of complete sequence of link codes is 23.11 bits in Figure 11 and the probability that the marking field has to be stored is 0.002 in Figure 13 with unequal packet distribution among links. But with equal distribution where all the frequencies are same with 255, the average length of sequence of link codes is 24.95 bits in Figure 12 that is a little larger than that with unequal distribution, while the probability of saving of the marking field is 0.001 in Figure 14 because the length of Huffman codes is a little longer with  $\lceil \log_2 n \rceil$  bits or  $\lceil \log_2 n \rceil - 1$  bits for  $n$  links with same frequencies (255) of packets among  $n$  links while the

average length of Huffman codes with unequal distribution is always less than with  $\lceil \log_2 n \rceil$  bits.

In Figure 12 with average degree 3 at distance 21, the average length of complete sequence of link codes exceeds 30 bits that is the length of  $ls$  in the marking field and consequently the link sequence has to be stored. Therefore the probability that the marking field of a packet with average distance 16 and average degree 3 is stored at least once on the way to its destination is about 12/32 (distance 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31 and 32 out of 32 distances) that is 0.375 (The area in Figure 18 is about 1/3). And the probability that a router on the path of 16 routers stores the marking field approximates to  $0.375/16$ , which means that 2.34% of marking fields are stored at a router.

With average degree 4, the probability that marking field is stored at least once is  $16/32 = 0.5$  (distance 17, 18, ..., 32 out of 32 distances) because at distance 17 the average link sequence exceeds 30bits and the area in Figure 18 is about 1/2. The probability of saving of marking field at a router out of average 16 routers is  $0.5/16 = 3.1\%$ .

Since the actual average degree in the Internet is between 3 and 4, the percentage of packets whose marking field is stored at a router is inferred less than 3%. And furthermore, as the distances of paths are distributed around 16 as shown in Figure 17 if we apply different weights according to their distribution the actual percentage will drop to less than 2%.

A high-end core router with capacity of 1 Tera bps that is 1 Giga packets/sec with assumption

that average packet size is 1 Kbits will store 20M marking fields per second. The memory required for the router is 160MB/sec ( $20M * 64\text{bits} / 8\text{bits}$ ), which is 0.128 % of router's capacity for keeping of marking fields for a second, and 7.68% for a minute. For a low-end router with capacity 1Gbps the requirement is 9.6 MB/min.

### 16-bit marking field

Figures 15 and 16 show the average count of savings of 16-bit marking field with each unequal and equal distribution, and Figure 18 shows only with degrees 3 and 4 with equal distribution of incoming packets among links. The average count of savings of marking field of a packet is about 2 according to Figure 19 with degree between 3 and 4. And even if we apply hop distribution (Figure 15) to Figure 18, the actual average count of savings will be some 2. The average count of savings of a packet at a router of 16 routers is  $2/16$ , which is 0.125 meaning that marking fields of 12.5% of packets that a router forwards are stored at this router. The memory requirement for a router with capacity of 1 Giga bps is 750KB/sec ( $0.125M * (32+16)\text{bits} / 8\text{bits}$ ) which is 0.6% of router's capacity for a second keeping and 36% for a minute.

### Transformations

Transformation did not affect on the average length of codewords and the memory requirements due to the percentage of packets that undergo transformation is generally low, in the simulation the percentage were 0.1, 0.5, 1, 2 and 3%.

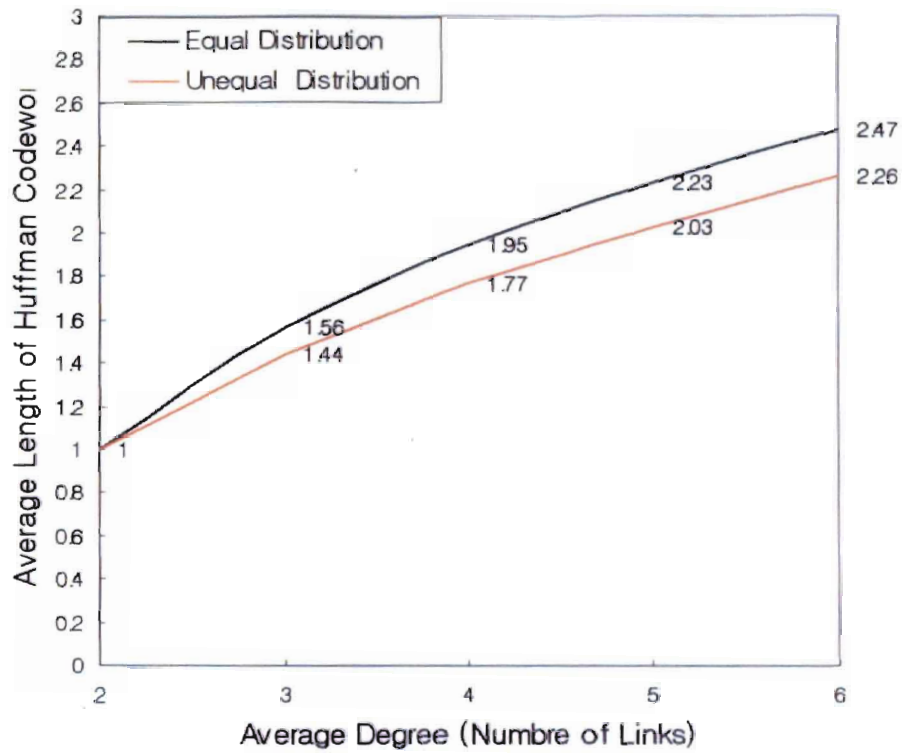


Figure 10. Average length of Huffman codewords for average degrees 2, 3, 4, 5 and 6 with one percentage of packet transformation.



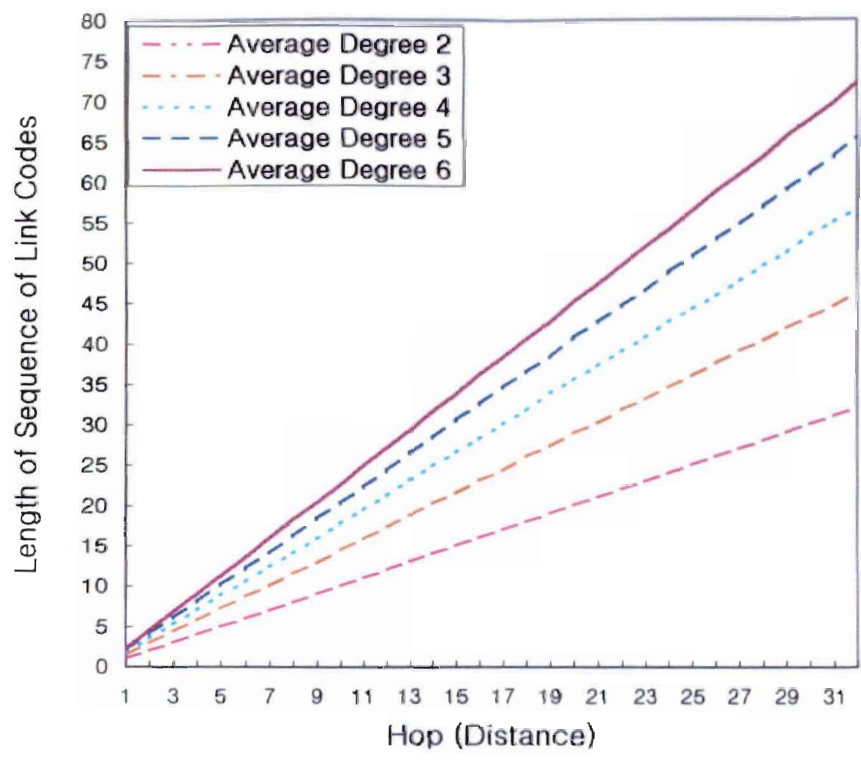


Figure 11. Average length of sequence of link codes with unequal distribution of packets among links. Links were given different frequencies in the range of 1~ 255 randomly and one percentage of packets transformed.

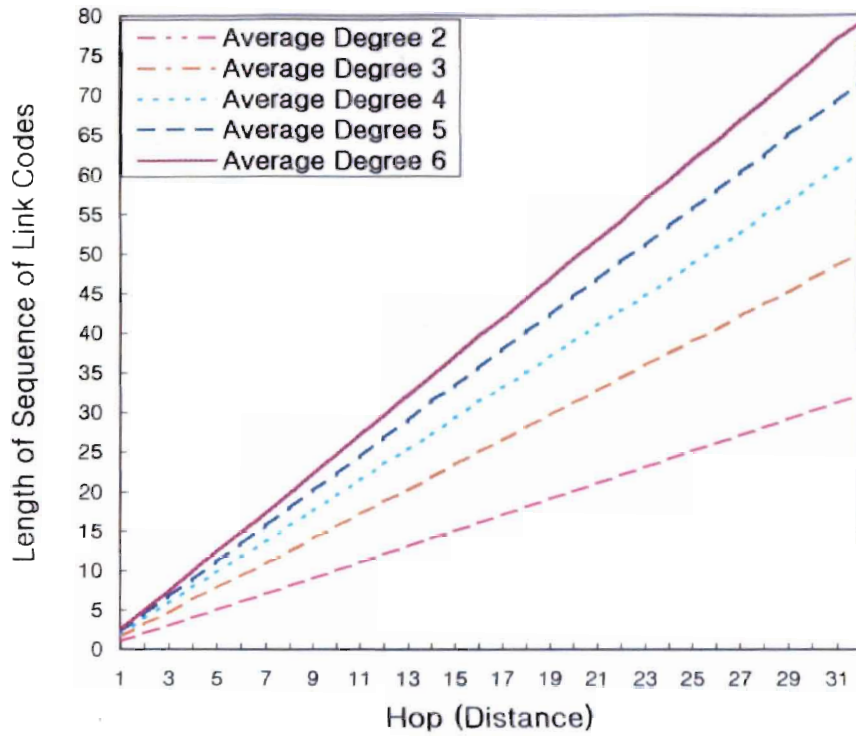


Figure 12. Average length of sequence of link codes with equal distribution of packets among links. Every link was given 255 the same frequency of packets, and one percentage of packets transformed.

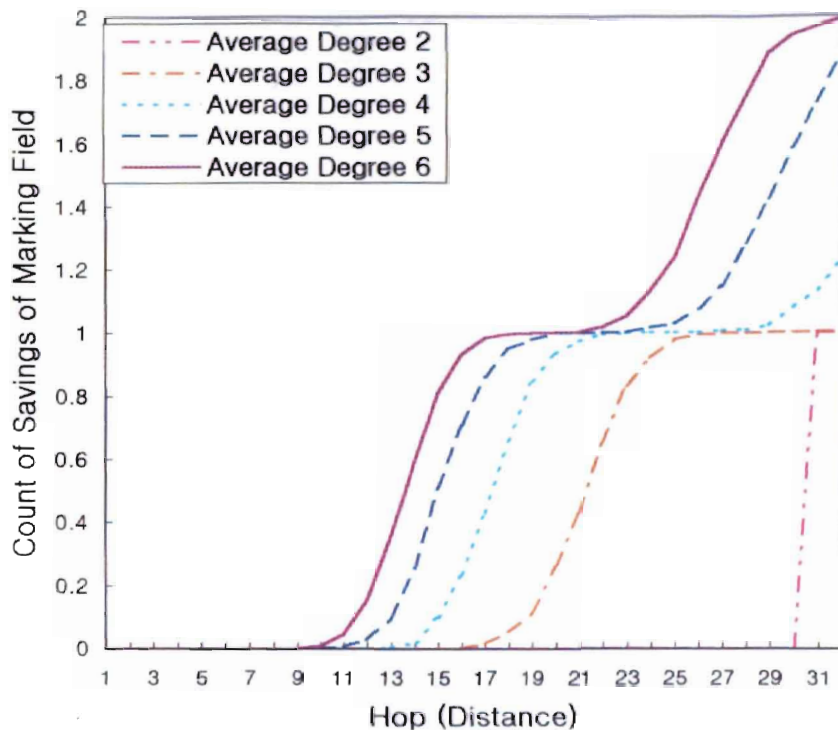


Figure 13. Average count of savings of 32-bit marking field of a packet during its travel with unequal distribution of packets among the links. Links were given different frequencies in the range of 1~ 255 randomly and one percentage of packets transformed.

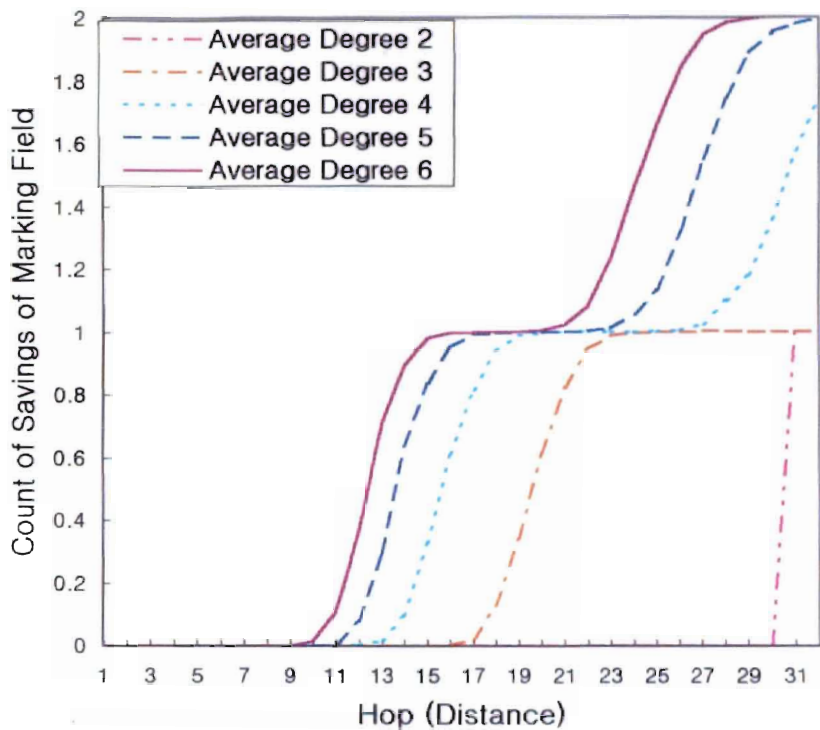


Figure 14. Average count of savings of 32-bit marking field with equal distribution of packets among the links. Every link was given same frequency of packets with 255, and one percentage of packets transformed.

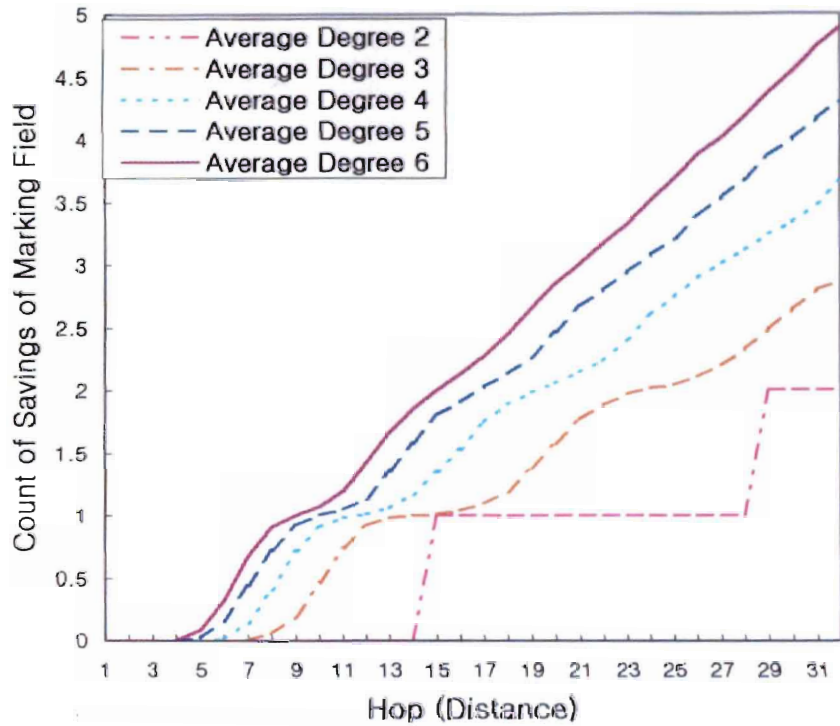


Figure 15. Average count of savings of 16-bit marking field of a packet with unequal distribution of packets among the links during its travel. Links were given different frequencies in the range of 1~ 255 randomly and one percentage of packets transformed.

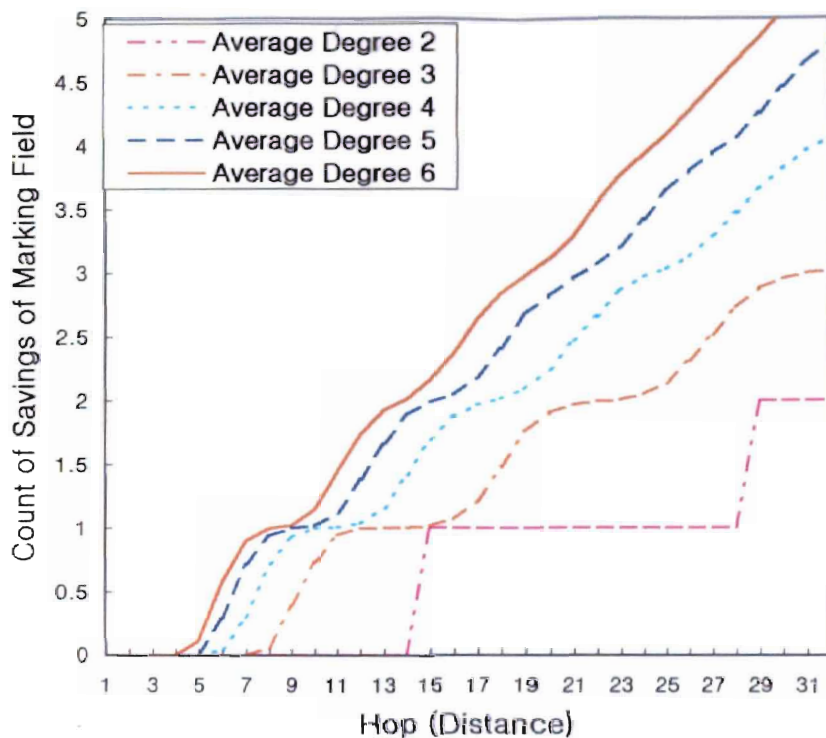


Figure 16. Average count of savings of 16-bit marking field of a packet with equal distribution of packets among the links during its travel. Every link was given same frequency of packets with 255, and one percentage of packets transformed.

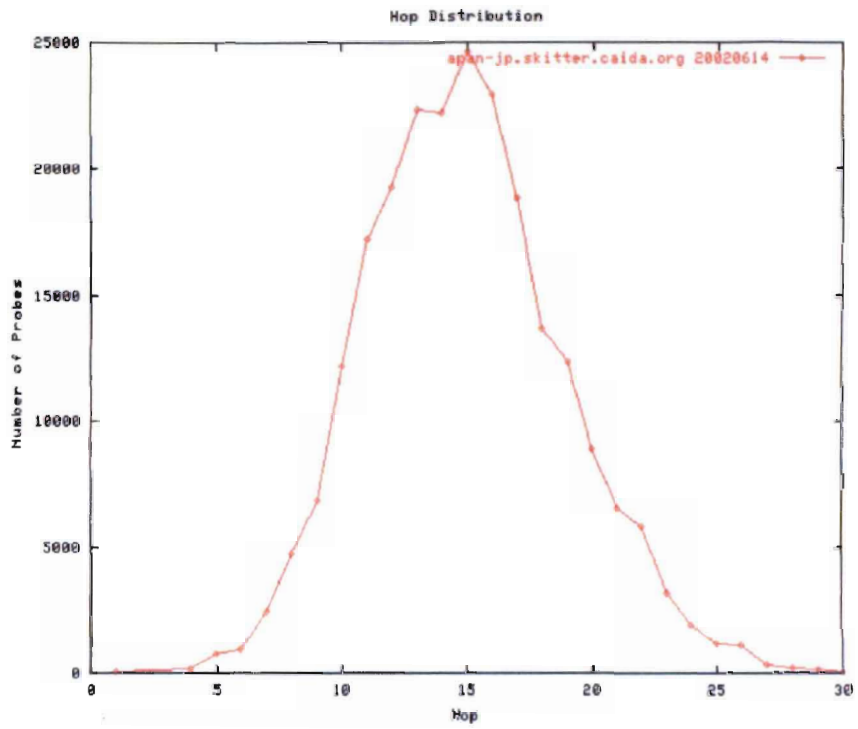


Figure 17. Hop distribution (adopted from [10]).

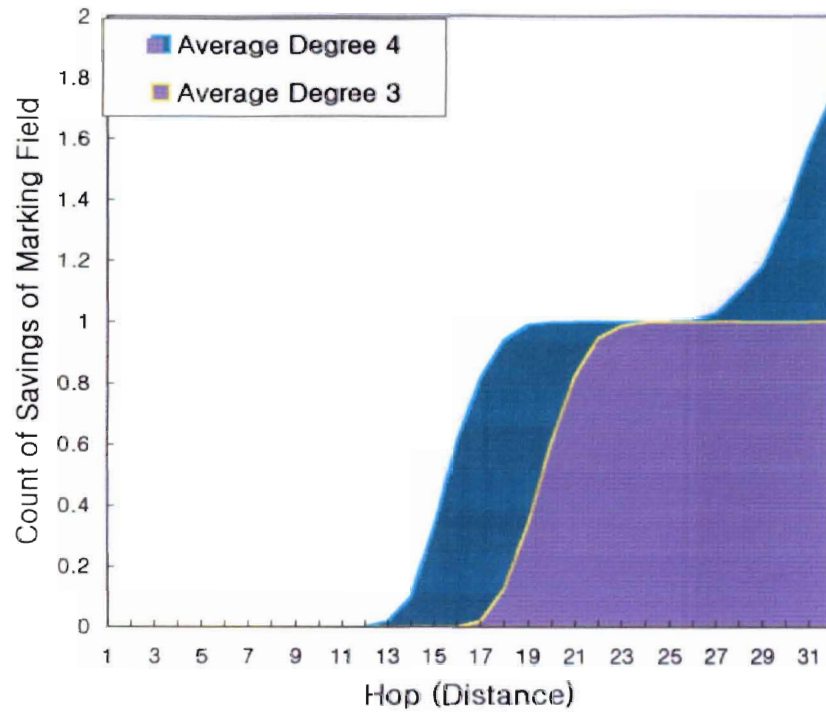


Figure 18. Average count of savings of 32-bit marking field of a packet with average degrees 3 and 4, with one percentage of packet transform and equal packet distribution among links.



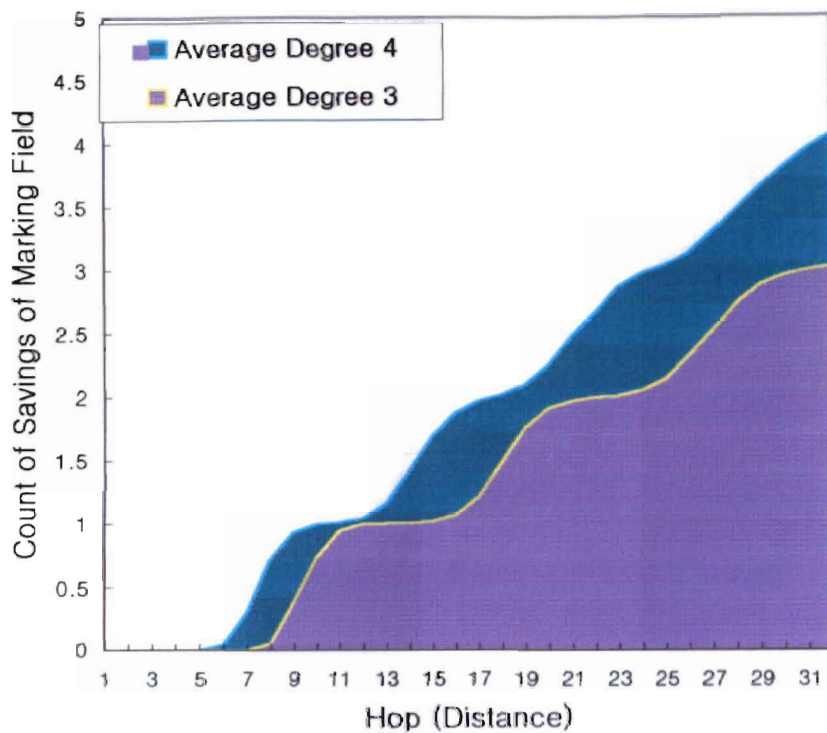


Figure 19. Average count of savings of 16-bit marking field of a packet with average degrees 3 and 4 , with one percentage of packet transform and equal packet distribution among links.

### 3. Comparison with Other Methods

- ✓ Can trace both during an ongoing attack or postmortem

Like other marking methods this new method allows a victim traceback a packet both during an attack and after an attack has been completed, provided that the link sequence still remains in intermediate routers in case that intermediate routers stored the sequence.

- ✓ Can construct a path of any packet correctly

With this method we can construct a path of any packet regardless of whether it is an attacking packet or not, meaning that the method does not require amount of packets to construct a path, only one packet is enough.

- ✓ Can construct all paths of DDoS attacks correctly

Packets of different attack paths will have different link sequences and each sequence can be decoded into a different attacking path.

- ✓ Requires less computation to traceback

Compared to probabilistic markings or hash based logging, the new method can easily construct a path of a packet provided that it can access link tables of intermediate routers.

- ✓ Requires smaller amount of space than other loggings

This method requires about a third of amount of space required in hash based logging to store marking fields along with message digests in intermediate routers.

- ✓ Requires local memory to store marking fields

It is essential for routers to have enough memory to store marking fields even though the memory requirement is less than that of other method. The requirement increases in proportion to the period of keeping of marking fields

- ✓ Adds overhead of marking to routers

It is a load for routers to maintain a link table and that the table must be correct and well optimized, and it is an issue how to enforce or impose an obligation of keeping and managing the table to all routers.

- ✓ Vulnerable to 1-bit error

The new idea requires all routers a packet pass through to mark, and if one of internal router does not mark or if there is at least a 1-bit error in the marking field of a packet then the traceback of the packet will fail. It is a characteristic of a variable length codes like Huffman codes that if one of bit is inverted or missing by error then correct decoding (expanding) of the encoded (compressed) bit string is impossible from the codeword including the bit error.

## PRACTICAL ISSUES FOR THE NEW SCHEME

### Management of link tables

Each router must maintain a correct link table and provides victims with the table when asked for access. Link tables should be optimized as well as possible to reflect the correct distribution of packets coming into the routers from its adjacent routers.

It will be an issue how to enforce or impose an obligation of keeping and managing tables to all routers. We may authorize a certain system or an organization to collect all the link tables and manage them: checking correctness and controlling access to the tables. This collection of tables will be a whole router-level Internet map. If the configuration of links of a router changes then the link table should be updated promptly and previous link table must be preserved for some period of time such that a victim can access the previous table and decode a link code marked by the router. Each previous table must be annotated with starting and ending dates and times.

### Local memories of routers

Many current routers are not equipped with a hard disk and do not have enough main memory to store marking field for a certain period of time, that is to say one minute or so.

### **The ability to know on what link packets arrive**

Every router can satisfy the assumption that is capable to know from which link packets arrive on. But the function to figure out from which neighbor sent a packet to it must be done automatically upon arrival of the packet, and the packet must be tagged with the link information until it is marked by the embedded program of a router.

### **Packet transformations**

A packet can be transformed more than once during its journey by internal routers. To trace a packet that was transformed from another packet back up to routers before the transformation, the marking field of old packet must be copied into new packets. Some protocols of transformation like ICMP copy the contents of the IP header of previous packet into the data field of new packets.

## CONCLUSIONS AND FUTURE WORKS

It is difficult to trace a packet back to its source with current IP version 4.0. From the beginning the Internet was not designed and implemented with tracebacks in mind, needless to say when people started building the Internet they had not imagined situations where tracebacks are needed.

Current IP header is not appropriate for marking, using either the Identification field or the Option field of IP header has its own limitation. Therefore this study does not suggest specific fields in IP header to use for marking, but suggests and analyzes a new marking technique with two different sizes of marking field, 16-bits and 32-bits.

The new idea proposed in this study requires routers to have enough memory space regardless of whether it is a hard disk or a main memory to store marking fields for a certain period of time in accordance with the amount of traffic. However most of routers have been doing their jobs without a local hard disk or even with a small main memory, so they have to be equipped with a secondary memory to store marking fields. In hash based logging [1] they attach a DGA (Data Generation Agent) to a router to store information of packets the router forwards.

The scheme presented in this study is to mark every packet at routers so that every packet will have information about intermediate routers between source (attacker) and destination (victim).

It may be worth thinking over whether it is necessary to generate IP traceback information for all packets regardless of whether it is a marking or a logging. In probabilistic markings routers do not mark all packets but sample packets to mark because packets cannot keep all the router's IP information due to the limited space of the marking field in IP header. To lessen the marking load of routers and to decrease the size of the marking field in IP header probabilistic marking can be applied to the new scheme.

Moreover it is necessary to deliberately select fields of IP header to use as an input of the message digest so that routers do not need to store  $MD(\text{oldP}):MD(\text{newP})(sf,ls)$  in case that a transformation does not change the fields of IP header that are used as inputs of the message digest. For instance the Identification field is used in fragmentation transformation, and if routers do not use the Identification field and data portion as inputs when calculating the digest to store a marking field because of lacking of space in the link sequence field ( $ls$ ), routers do not need to store  $MD(\text{oldP}):MD(\text{newP})(sf,ls)$  in case of fragmentations.

## REFERENCES

- [1] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer. Hash-based IP Traceback. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 3-14. Association for Computer Machinery, 2001.
- [2] D. X. Song and A. Perrig. Advanced and authenticated marking schemes for IP traceback. In *Proceedings of Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 878-886. IEEE, 2001.
- [3] H. Burch and B. Cheswick. Tracing anonymous packets to their approximate source. In *Proceedings of the 2000 System Administration Conference (LISA 2000)*, pages 319-327. Advanced Computing Systems Association, Dec 2000.
- [4] P. Fergusson and D. Seine. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source address Spoofing. *RFC2827*. Internet Engineering Task Force (IETF), May 2000.
- [5] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical network support for IP traceback. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 30(4):295-306. Association for Computer Machinery, August 2000.
- [6] S. Bellovin, M. Leech, and T. Taylor. ICMP Traceback Messages. *RFC2026*. Internet Engineering Task Force (IETF), March 2000, Expires April 2002.
- [7] I. Stoica and H. Zhang. Providing guaranteed services without per flow management. in *Proceedings of ACM SIGCOMM Special Interest Group on Data Communications '99*, pages 81-94. Association for Computer Machinery, Aug 1999.
- [8] C. R. Palmer, G. Siganos and M. Faloutsos. The Connectivity and Fault-Tolerance of the Internet Topology. *Workshop on Network-Related Data Management (NRDM 2001)*, In cooperation with ACM SIGMOD/PODS Association for Computing Machinery Special Interest Group on Management of Data/Principles of Database Systems, 2001 Santa Barbara, May 2001



- [9] W. Theilmann and K. Rothermel. Dynamic distance maps of the Internet. . In *Proceedings of Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 275-284. IEEE, 2000..
- [10] K. C. Claffy and D. McRobb. Measurement and vi-sualizatiion of Internet connectivity and peformance. <http://www.caida.org/tools/measurement/skitter>
- [11] R. L. Rivest. The MD5 message digest algorithm. *RFC 1321*. Internet Activities Board, Internet Privercy Task Force. April 1992.

## APPENDIX

### Source Codes of Simulation Program (32-bit Marking Field Version)

```
//trace.h

struct packet
{
    unsigned int mf32; //(MSB is sf(saved flag), rest 31-bits are for ls(link sequence))

    //for analysis
    int saved_times; //not a part of packet, only to count the times mf is saved
    int ls_length; //not a part of packet, only to find the length of complete link sequence
};

//for Huffman tree
struct hf_node
{
    unsigned int frq;
    int left; //index of left child node
    int right;
};

//for huffman codes
struct hf_code
{
    unsigned short bits; //codeword (....xxxxxxxx), x is 1 or 0
    int length; //number of x's in codeword
};

struct router
{
    unsigned int IPAddr;

    //local memory
    unsigned int stored_mf32;

    //link table
    int links; //number of links(adjacent routers)
    unsigned int frq[29]; //only first links-frq are valid
    unsigned int l_IPAddr[29]; //only first links-IPaddresses are valid //max num of links = 29
    struct hf_code code[29]; //only first links-codewords are valid for links-links
};

void Marking(int r, struct packet &p, int in_link, int tf);
void Traceback(int router, struct packet &p);

int CreateHuffmanTree(int router);
void AssignHuffmanCode2Link(int router, unsigned codeword, int length, int node);
int CountAvailOFLS( unsigned int ls);
void AppendHFcode2LS(unsigned int &ls, int router, int in_link);
```

```

// trace.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "trace.h"
#include <stdlib.h>
#include <time.h>
#include <string.h>

struct router R[36]; //max 36 routers
struct hf_node Node[58]; //max 29 links(terminal nodes) --> max 57 nodes + 1
unsigned int Path[36]; //will be filled by Traceback(),(list of IP addresses of routers on the path of a packet)
int Path_length; //number of routers on the path, set by Traceback()

#define TFs 5
int TF_ratio[TFs] = {1,5,10,20,30}; // 0.1, 0.5, 1,2,3% each

#define MaxDistance 36 //distance is 1 ~ 36
#define MaxAvgLinks 6 //average number of links is 2 ~ 6

#define MarkingField16 16 //marking field is 16 bits long
#define MarkingField32 32 //marking field is 32 bits long

int MarkingField = MarkingField32;

int DoTrace = 0; //if DoTrace=1 then afer a packet has reached its destination, the destination will do the TraceBack of the packet
//for analysis
//number of packets to test
//length of link sequence of Trys-packets
unsigned int LLS[2][TFs][MaxDistance][MaxAvgLinks-1][Trys];
//LLS[0] is with uniform distribution( same frequencies of packets from each links of a router)
//LLS[1] is with not uniform // ( different // )

//Times the marking field has to be saved during the journey of a packet toward its destination
unsigned int SavedTimes[2][TFs][MaxDistance][MaxAvgLinks-1][Trys];

//average of 1000 packets
float AvgLLS[2][TFs][MaxDistance][MaxAvgLinks-1];
float AvgSavedTimes[2][TFs][MaxDistance][MaxAvgLinks-1];

//average length of Huffman codeword
float AvgLCW[2][TFs][MaxAvgLinks-1];

int main(int argc, char* argv[])
{
    int distance, avg_links;
    struct packet p;

    FILE *outf;
    outf = fopen("result_32.txt", "ab");
    char msg[120];

    sprintf(msg, "Frq_Type TF(%%) Distance Avg_links Length_of_LS Times_Saved\n");
    fwrite(msg, strlen(msg), 1, outf);

```

```

/* Seed the random-number generator with current time so that
 * the numbers will be different every time we run.
 */
srand((unsigned)time( NULL ) );

for(int frq_type=0; frq_type <2; frq_type++)
{
    for(int t=0; t<TFs; t++) //TF_ratio[TFs] = {1,5,10,20,30} : 0.1, 0.5, 1, 2, 3% each
    {
        for(avg_links = 2; avg_links <= MaxAvgLinks/*12*/; avg_links++)
        {
            printf(" t=%d,avg_links=%d\n",t,avg_links);
            for (distance = 1; distance <= MaxDistance/*36*/; distance ++)
            {
                for(int n=0; n<Trys; n++) //try Trys-times
                {
                    p.mf32 = 0x00000001; //sf = 0; ls = NULL; //initialize packet as it is departing
                    the source host
                    p.saved_times = 0;
                    p.ls_length = 0; //length of complete link sequence includeing stored link
                    sequences

                    //packet p traverses routers
                    int cur_router; //index of current router
                    for(int r=0; r<distance; r++)
                    {
                        cur_router = r;

                        ////////////////////////////////////////////////////////////////////
                        //create a router and a link table for it
                        ////////////////////////////////////////////////////////////////////

                        R[r].stored_mf32 = 0x00000000;

                        int links = ( rand() % (2*avg_links-3)) + 2;

                        R[r].links = links;

                        for(int i=0; i<links;i++)
                        {
                            R[r].l_IPaddr[i] = rand() * rand(); //4-byte IP address of adjacent router
                            printf("IP:%08x ", R[r].l_IPaddr[i]);

                            if(frq_type==0)
                                R[r].frq[i] = (rand() % 255) + 1; // 1 ~ 255
                            else
                                R[r].frq[i] = 255; //same frequency for all links

                            //
                            printf("frq:%d ", R[r].frq[i]);
                            //
                        }

                        //choose a in_link from links, assuming that the packet p came from this
                        in_link
                        //to simulate an biased distribution(frq_type=0), a packet will be selected from
                        total packet distribution of current router
                        int in_link;

                        if(frq_type==0)
                        {
                            int total=0;

```

```

int total_distribution[255 * 32];
for(i=0; i<links; i++)
    for(int j=0; j < R[r].frq[i]; j++)
        total_distribution[total++] = i;

//
    in_link = total_distribution[ rand()*rand() % total ];
    printf("\nin_link=%d", in_link);
}
else in_link = rand() % links;

//IP address of proceeding router is the in_link-th router of current router
if(r!=0)
{
    R[r-1].IPAddr = R[r].l_IPAddr[in_link];
}

//create Huffman tree to assign codewords to links
int root = CreateHuffmanTree(r); //root = index of root node

AssignHuffmanCode2Link(r, 0/*codeword*/,0/*length of codeword in bit*/,
root);

//////////////////////////////////////
//Transformation
//////////////////////////////////////
int tf = 0;
if( rand()% (1000*TF_ratio[t]) ==0)//TF_ratio[0] = 1 =>0.1 % of packets
undergo a transformation
    tf = 1; //transformed

//////////////////////////////////////
//marking packet p at router R[r]
//////////////////////////////////////
Marking(r, p, in_link, tf);

} //for each router packet p traverses
R[r-1].IPAddr = rand()*rand(); //IP address of the last router

//////////////////////////////////////
//for analysis
//////////////////////////////////////
LLS[frq_type][t][distance-1][avg_links-2][n] = p.ls_length;
SavedTimes[frq_type][t][distance-1][avg_links-2][n] = p.saved_times;

//////////////////////////////////////
// packet p has traversed distance-routers
// IP traceback
//////////////////////////////////////
if(DoTrace)
{
    Path_length = 0;
    Traceback(cur_router, p);
}

```

```

        printf("Path=");
        for(int i=Path_length-1; i>=0 ; i--)
        {
            printf("%08x:", Path[i]);
        }
        printf("Path_length=%d,saved=%d\n",Path_length,p.saved_times,p.ls_length);
    }

    Path_length = 0;
    } //if(DoTrace)
} //for 1000 times

////////////////////////////////////
//for analysis
////////////////////////////////////
//calculate the averages of lls and times
for(n=0; n<Trys; n++)
{
    AvgLLS[frq_type][t][distance-1][avg_links-2] += LLS[frq_type][t][distance-1][avg_links-2][n];
    AvgSavedTimes[frq_type][t][distance-1][avg_links-2] += SavedTimes[frq_type][t][distance-1][avg_links-2][n];
}

    AvgLLS[frq_type][t][distance-1][avg_links-2] /=Trys;
    AvgSavedTimes[frq_type][t][distance-1][avg_links-2] /=Trys;

    AvgLCW[frq_type][t][avg_links-2] += AvgLLS[frq_type][t][distance-1][avg_links-2];

    char msg[120];
    sprintf(msg, "Frq_Type   TF(%%)   Distance   Avg_links   Length_of_LS\n");
    Times_Saved\n";
    sprintf(msg,
"%3d   %3.1f   %3d   %6.2f   %5.3f\n",
frq_type, (float)TF_ratio[t]/10, distance, avg_links,
AvgLLS[frq_type][t][distance-1][avg_links-2],
AvgSavedTimes[frq_type][t][distance-1][avg_links-2]);

    fwrite(msg, strlen(msg), 1, outf);
    //////////////////////////////////////

} //for distance

} //for average links

//for analysis
for(avg_links = 2; avg_links <= MaxAvgLinks/*12*/; avg_links++)
{
    AvgLCW[frq_type][t][avg_links-2] /= (37*18); //average length of a codeword

    sprintf(msg, "%3d   %3.1f   xxx   %d   %5.2f AvgLeng of\n",
a Codeword\n",
frq_type, (float)TF_ratio[t]/10, avg_links, AvgLCW[frq_type][t][avg_links-2]);

    fwrite(msg, strlen(msg), 1, outf);
}
}

```

```

    } //for tf_ratio
} //for frq_type

fclose(outf);

return 0;
}

/*****
// Marking() marks a packet p at a router with link information
*****/
void Marking(int router, struct packet &p, int in_link, int tf)
{
    unsigned int ls, sf;

    sf = p.mf32 & 0x80000000;
    ls = p.mf32 & 0x7fffffff;

    //check if packet p has transformed into a new packet and sf is set to 1
    if( sf && tf) //marking field had been stored before a transformation happened
    {
        //store marking field at router r
        R[router].stored_mf32 = p.mf32;
        p.saved_times++;

        //reset marking field of packet p
        ls = 0x00000001; //ls =NULL(only has the delimiter bit 1 in the right most position)
    }

    //check if enough space left in marking field
    int left_bits = CountAvailOfLS(ls);

    if(left_bits < R[router].code[in_link].length) //lack of space in marking field
    {
        //store marking field at router r
        R[router].stored_mf32 = ls|sf;

        p.saved_times++;

        //reset marking field of packet p
        sf = 0x80000000; // sf =1
        ls = 0x00000001; // ls =NULL(only has the delimiter bit 1 in the right most position)
    }

    //append a huffman codeword for in_link to the marking field
    AppendHFcode2LS(ls, router, in_link);

    p.mf32 = ls|sf;
    p.ls_length +=R[router].code[in_link].length;
}

/*****
CountAvailOfLS() counts the available number of bits in ls
*****/
int CountAvailOfLS( unsigned int ls)
{
    int left =0;

```

```

unsigned int mask = 0x40000000; //01000000 00000000 00000000 00000000

while(1)
{
    if(ls & mask) break;

    mask >>= 1;
    left++;
}
return left;
}

/*****
//huffmans codeword will be appended to the right end of marking field,
//the code is appended in reverse bit order
//Eg. 1100100101 --> 00...1xxx...xxxxx1010010011
*****/
void AppendHFcode2LS(unsigned int &ls, int router, int in_link)
{
    unsigned int mask = 0x00000001;
    unsigned int code;
    int length;

    code = R[router].code[in_link].bits;
    length = R[router].code[in_link].length;

    //append a code in reversed bit-order to the end of ls
    for(int i=0;i<length;i++)
    {
        ls<<=1;
        if(code & mask) ls |= 0x00000001;
        mask<<=1;
    }
}

/*****
// Traceback() starts IP traceback of packet p from a router and recursively calls Traceback()
*****/
void Traceback(int router, struct packet &p)
{
    int i;
    unsigned int ls;
    unsigned int sf;

    ls = p.mf32 & 0x7fffffff;
    sf = p.mf32 & 0x80000000;

    Path[Path_length++] = R[router].IPAddr; //current router's IP address

    //construct a huffman tree using distribution (frequencies of packets among links)
    int node = CreateHuffmanTree(router); //node = index of root node

    //decode one codeword in marking field(ls)

```



```

while(node >= R[router].links) // if the node is a terminal node, then one codeword from ls
decoded
{
    if(ls & 0x00000001) // bit = 1
    {
        node = Node[node].right;
    }
    else
    {
        node = Node[node].left;
    }

    ls>>=1;
    //if( CountAvailOfLS(mf) ) break;
}

//here, node(index of Node[]) represents the link number
//now find the router that is connected by the link with current router
for( i=0; i<router; i++)
{
    if( R[i].IPAddr == R[router].l_IPAddr[node] ) break;
}

//in this program, i is router-1,
//a router R[i] is created after router R[i-1] by the order packet p traverses
//so above for statement is not necessary and below can be just "Traceback(router-1, p);
//router R[i] is the upstream router of the current router R[router]

if(router!=0 && i == router) {printf("proceeding router Not found(error in ls data"); }

p.mf32 = ls | sf;

//if no link codes are in ls and sf = 1 then retrieve the stored making field
if( CountAvailOfLS(ls)==30 && sf )
    p.mf32 = R[router].stored_mf32;

else if(CountAvailOfLS(ls)==30 && !sf) //all link sequence has been decoded
    return; //no more ls left to be decoded

Traceback(i, p);
}

```

/\*\*\*\*\*  
 CreateHuffmanTree() makes the Huffman tree from the frequency information and return  
 the index of root\_node  
 \*\*\*\*\*/

```

int CreateHuffmanTree(int router) {

    int i, min1, min2, next_free_node, links;

    links = R[router].links;

    for(i=0; i< links; i++)
    {
        Node[i].frq = R[router].frq[i];
    }
}

```

```

next_free_node = links; // here, i==links
Node[57].frq = 0xffffffff; //actual frq of links is between 1 ~ 255
while(1) {
    min1=min2 = 57;

    for(i=0; i<next_free_node; i++) { //look over to find 2 min counts
        if(Node[i].frq !=0) { //frq 0 means this node is already added to tree
            if(Node[i].frq < Node[min1].frq) {
                min2 = min1; //Node[min1].frq < Node[min2].frq
                min1 = i;
            }
            else if(Node[i].frq < Node[min2].frq) min2 = i;
        }
    }

    if(min2 == 57) break; //root node is created and no more node left to be added

    //new node with weight of min1's + min2's
    Node[next_free_node].frq = Node[min1].frq + Node[min2].frq;
    Node[min1].frq = Node[min2].frq = 0; //2 nodes are added to the tree

    Node[next_free_node].left = min1;
    Node[next_free_node].right = min2;

    next_free_node++;
}

return --next_free_node; //index of root node
}

/*****
Assign Huffman Code 2 Link() finds and assigns Huffman codes to links (terminal nodes)
*****/
void Assign Huffman Code 2 Link(int router, unsigned codeword, int length, int node) {

    int links = R[router].links;

    if(node < links) { //node is a terminal node( codeword is completed )
        R[router].code[node].bits = codeword;
        R[router].code[node].length = length;
        return;
    }
    codeword <<=1; //bitstring value
    length++; //length of bitstring value, each time it descends, length increases by 1
    Assign Huffman Code 2 Link(router, codeword, length, Node[node].left);
    Assign Huffman Code 2 Link(router, codeword | 1, length, Node[node].right); //code: ...xxx =
length3, value xxx, right most x added last
}

```

7

## VITA

Kyu Hyong Choi

Candidate for the Degree of

Master of Science

Thesis: A NEW MARKING SCHEME USING HUFFMAN CODES FOR  
IP TRACEBACK

Major Field: Computer Science

Biographical:

Education: Graduated from Yungseng High School, Chonju, Korea in 1984; Received Bachelor of Engineering Degree in Computer Engineering from Chonbuk National University, Chonju, Korea in February 1988; Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University, Stillwater, Oklahoma in May 2003.

Experience: Completed Military Service as a Technical Instructor, Korean Air Force, August 1989 to July 1992; Employed by Korean Government, Records and Archives Service, as a computer programmer and system administrator, Teajon, Korea, August 1992 to present.

Professional Membership: The Korean Computer Scientists and Engineers Association in America (KOCSEA).