Formal Specifications of Traceback Marking Protocols

Chun He

An Honors Thesis

The University of Texas at Austin
Department of Computer Sciences
Austin, Texas 78712

May, 2002

Supervisor: Dr. Mohamed. G. Gouda

Abstract

Denial-of-Service attacks and Distributed Denial-of-Service attacks are serious security problems over the Internet due to their nature. They are easy to implement, hard to prevent, and very difficult to trace. This paper describes Denial-of-Service attacks and Distributed Denial-of-Service attacks and presents various traceback that are proposed to identify the sources of theses attackers in the Internet. Finally it gives formal specifications for a class of these traceback protocols called marking protocols.

1. Introduction

1.1 Denial-of-service Attack

A "denial-of-service" attack (DoS for short) is characterized by an explicit attempt by attackers to prevent legitimate users of a service from using that service [1]. For example, a computer on the Internet can be victimized as follow, one or few attacker(s) can attack this computer by sending million meaningless messages to this victim contiguously, normally message has a spoofed source address to hide the identify of the attacker(s). The victim computer use resource to process the messages sent by the attacker(s), eventually it will discard these meaningless messages, but the victim computer already took a long period of time out to handle these messages. Eventually, the attack will block the victim computer from accepting the legitimate messages send by other computers. The examples of DoS attacks include,

- Attempts to "flood" a network with vacuous messages, thereby preventing legitimate network traffic
- Attempts to disrupt connections between two machines, thereby preventing access to a service
- Attempts to prevent a particular individual from accessing a service
- Attempts to disrupt service to a specific system or person

There are mainly three modes of the attacks,

- Attacking a victim computer by consuming resources of that computer
- Attacking a victim computer by destructing or altering of configuration information.
- Attacking a victim computer by physical destructing or altering of network components.

1.2 Distributed Denial-of-service Attack

An attack that is more malicious than a mere denial-of-service attack and a lot harder to deal with is called distributed denial-of-service attack, DDoS for short. A distributed denial-of-service attack is a denial-of-service launched from many sites to the victim computer at the same time. On the Internet, a distributed denial-of-service (DDoS) attack is one in which a multitude of compromised systems attack a single target, thereby causing denial-of-service for users of the targeted system. The flood of incoming messages to the target system essentially forces it to shut down, thereby denying any service at the system to legitimate users [11]. The attacker can use tools to develop and coordinate the attack easily. The attacker can even use random computers on the internet to perform the attack without the owners of these computers aware of the attack, like many “zombie” computers ganging up on one computer, directed by one “master”, which is controlled by the attacker.

1.3 History of Denial-of-service Attack

Because denial-of-service attacks are simple to implement and difficult to prevent, in the recent years the attacks have increased dramatically in frequency, severity and

sophistication. Traditional DoS attacks typically have generated a large amount of traffic to a victim computer and it is possible for a victim computer to detect such an attack in progress and defend itself. For example, year 1996, Hackers use "SYN floods" to take down several web servers by overwhelming and disabling them. The method was a simple form of denial-of-service attack, and its success increases the popularity of hacking [2].

Distributed denial-of-service attacks are a much more nefarious extension of DoS attacks because they are designed as a coordinated attack from many sources against one or more targets. Early February 2000, a teenager using very simple distributed denial-of-service tools [3] managed to bring down the high profile web sites of large companies like Yahoo, ebay, CNN, Buy.com, Amazon, ZDNet, E#Trade and Excite during a series of attacks [4]. Unfortunately, mechanisms for dealing with denial-of-service attack have not yet developed to counter DDoS attacks at the same level.

1.4 Ping Attack

Denial-of-service attacks come in a variety of forms and aim at a variety of services, TCP flood, UDP flood, ICMP echo request / reply, the most popular attack is the ping attack. Later in October 1996, a public newsgroup discussion started regarding a “fat ping” or “ping of death” DoS attack [12]. This particular ping DoS attack uses a ping packet of an abnormal size exceeding the TCP/IP specification to either cause a system crash or network programs to stop processing in the targeted computer. In an IP network, any computer can send any other computer a “ping” message and that computer replies by sending back a “pong” message to the source of the “ping” message [13]. An attacker can attack a computer *v* in a network using the following methods, an attacker inserts a “ping” message to the network, with the spoofed source *v* and whose ultimate destination is every computer on the network. Then every computer on the network will reply a “pong” message to this compute *v*, *v* will be flooded with all the “pong” messages and cause denial-of-service on *v*.

2. Approaches to Deal with DoS Attacks

In most DoS attacks, the attacking messages carry wrong information about the identity of their sources in order to protect the true source of these messages. Therefore, there are two approaches to deal with DoS attacks. The first approach is to detect the true source of the attacking messages by collecting additional information from the attacking messages. This approach is called IP traceback. The Second approach is to detect and discard all messages with wrong sources and discard them away as they approach the victim. The following are more detailed description and examples of each approach, see figure 1 about the relations of these protocols.

I. Detect the true sources of most messages by collection additional information, called IP traceback

There four types of IP traceback protocols in this approach.

1. Traceback Marking Protocol
   This protocol requires all routers in the network to add information in the messages that pass through them. Then the victim computer can figure out the route of attack from the information in the messages it received. We will discuss in great details about this protocol in later sections.

2. Traceback Logging Protocol
   This protocol doesn't require routers to mark any message, but require routers to remember for a short period of time, all the messages that pass through them. This approach suggested in [5] and [6] is to log packets at key routers and then use data mining techniques to determine the path that packets traveled. It can trace the attack long after the attack, however it requires enormous resource.

3. Link Testing Protocol
   It starts from the router closest to the victim and interactively tests its upstream links until it finds out which one is used to carry attackers' packets. This approach does not work with attacks detected after the fact,

attacks that occur intermittently or attacks that modulate their behavior in response to a traceback, since it assumes that an attack remains active until the completion of a trace. Refer to [14] and [15] for examples of this approach.

4. ICMP Traceback Protocol

   This approach is based on explicit router-generated ICMP traceback messages. The idea is for every router, with a low probability, to forward a copy of the contents into a special ICMP traceback message [7]. The approach could be quite effective, but an attacker can forge the ICMP traceback messages and send them to the victim.

II. Detect and discard fake messages and discard them as it travel over the network

The hop integrity protocol in [13] is an example of this approach. Two computers are called adjacent iff both computer are connected to the same subnetwork (so that they can send message directly to each other). A network is said to provide hop integrity iff the following two conditions hold for every pair of adjacent routers *p* and *q* in the network.

- Detection of Message Modification

  Whenever router *p* receives a message over the subnetwork connecting routers *p* and *q*, and determines that message has modified by an attacker after it was sent by *q* and before received by *p*.

- Detection of Message Replay

  Whenever router *p* receives a message over the subnetwork connecting routers *p* and *q*, and determines that message has not been modified, then *p* can determine correctly whether that message is another copy of a message that is received earlier by *p*.

The goal of the hop integrity protocol is to detect and discard the messages inserted by attacker, and report the physical location in the network where the attacker inserts the messages. This approach is effective and can identify the locations where the attackers break into the local network. However, this approach requires the participation of all routers in the Internet.

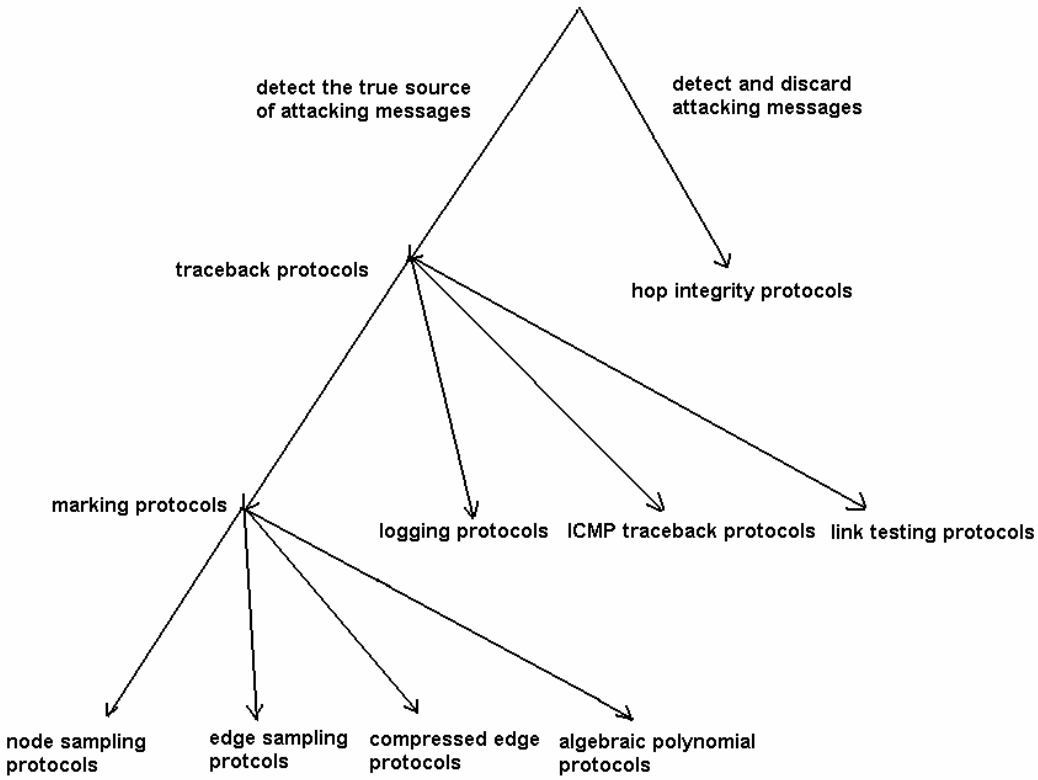

Figure 1. Protocols that deal with Dos attacks

In this paper we will describe four IP traceback protocols in great details. They are node sampling protocols, edge sampling protocols, compressed edge sampling protocols and algebraic polynomial protocols (see figure 1). The purpose of IP traceback protocols is to trace the source of attacks back toward their true origins, and ideally stopping an attack at its source. When a message passes through a router, that router might mark the

message in the IP header. The victim computer uses the information from the messages that have been marked to trace an attack back to its true origin. This approach has shown some potential advantages. It does not require interactive cooperation with Internet Server Providers (ISPs) and therefore avoids the high management overhead of input debugging. It does not require significant additional network traffic since it only modifies the IP field of the original messages. Moreover, just like logging, it can trace attacks long after the attacks have stopped. However, most of the IP traceback methods do not deal with DDoS attacks effectively. The effectiveness is reduced because the most significant method of traceback protocols to distinguish attack messages and regular messages is the large volumes of the attack messages, but in DDoS attacks, the attack messages came from many sources, therefore it loses the distinction between attack messages and regular messages. Therefore, in this paper, we are not going to deal with DDoS attacks, and just to describe them as they existed.

3. The Node Sampling Protocol

This protocol is suggested in [8] by *Savage*, *et al*. A node is a computer on the Internet; it can be a host or a router. In order to detect the true source of an attack, when messages travel through the computers on the network, each computer on the network marks messages with its IP address using a probability p. While a messages pass a computer, that computer chooses a random number x, if $x < p$ that computer adds its IP address to this message, otherwise it will simply pass this message to the next computer. Here we will assume these computers can aware whether they are under attack. If a computer finds out it is under attack it will start gathering the IP addresses from the messages and keeps a counter for each unique IP address, it will count how many messages it has received from each unique IP addresses. When the victim computer gathered enough information it will sort each IP address by its counter. The IP addresses of computers at a shorter distance on the path of attack will be marked more often than the computers at a greater distance. As a result of sorting the IP addresses' counter by descending order we can reveal all the computers on the path of attack in a way from the nearest computer to the farthest. In the following sections we will give the formal

specification of this protocol in Abstract Protocol (AP) notation and also talk about the advantage and disadvantage of this protocol.

First we will introduce some terms are used in the following specification. In this notation we will use the term "packet" for message. A computer sends out messages is a source of the messages, the computer where the message is delivered to is the destination. *p[i:0..n]* is an array of processes (computers) on the network, where *i* is in between 0 to *n* (0 to *n* are the indices of computers on the network). Here we use *i* as computer's IP address or its ID. In order to simplify this protocol, we will make no distinguish in between routers and hosts. Also we will use two constants *p* and *m*, which the probability for a message to get marked is *p/m*. There are three inputs for this protocol. *N* is a set, each element in set *N* is a neighbor computers of process *p[i]*. Array *rtb* is a routing table, which gives a destination process as index and returns the best next computer to route as the content of the array. *attack* here is a boolean to indicate whether the current process is in attack mode. The variables are *x*, *dst*, *node* and *count*. *x* is a variable in between 0 to *m*. *dst* and *node* are in between 0 to *n*. *count* is an array which takes process *i* as index and return how any times that process *i* has received with message reached to the destination computer. *j* is a parameter, which is in the set *N*. Each packet, *pkt,* needs to carry two fields, *node* (the IP Address marked by routers) and *dst* (destination computer's IP Address) where *dst* will not be change as the message travels in the network, but *node* might be overwritten by some routers.

The following is the AP notation of this protocol:

process p[i: 0..n]

const	p, m	{ probability to mark the packet is p/m }
inp	N:	**set** { g | g is a neighbor of p[i] },
	rtb:	**array**[0..n] **of** N
	attack:	**boolean**
var	x:	0..m,
	dst, node:	0..n,
	count:	**array**[0..n] **of integer** {init all 0's}

```
par     j:          N

begin
        true →
            dst := any;
            if i = dst →    skip
            □ i ≠ dst →     node := i;
                            send pkt(node, dst) to p[rtb[dst]]
            fi

        □ rcv pkt(node, dst) from p[j] →
            if dst ≠ i →    x := random;
                            if x < p → node := i
                            □ x ≥ p → skip
                            fi
                            send pkt(node, dst) to p[rtb[dst]]
            □ dst = i →
                    count[node] := count[node] + 1;
                    if  ~attack → skip
                    □ attack → { use "count" to detect location of attacker }
                    fi
            fi
end
```

This protocol includes two actions. Within process *i*, at any given time, first action generates a destination, *dst*, at random. Then it will check whether the *dst* it just generated equals to its own IP address, *i*. If it does, this action will do nothing else, otherwise it marks the *node* field with its own IP address, and generates a packet, *pkt*, with *node* and *dst*, then routes this packet to the next computer according to the routing table *rtb*. The second action receives a packet from *p[j]*. This action will first check whether itself, *i* is the ultimate destination. If it is not the destination it will randomly choose an *x* (where *x* is in between 0 to *m*), if $x<p$ it will overwrite field *node* in the packet with its own IP address, *i*; if $x \geq p$, it will *skip* which means do nothing. Finally *i* sends out the packet again to the next computer, *p[rtb[dst]]*. If it is the destination, the process increments the *count[node]* by 1, then check whether it is in *attack* mode. If it is not under attack, it does not do anything. If it is under attack, it use the information stored in *count* to detect the location of attacker by sorting the number of each *node* in array *count*.

This protocol is easy to implement in IP network because it only requires the addition of a write and checksum update to the forwarding path as additional operations for the routers. It needs to add a 32-bit field, *node*, to the IP header of each packet. It also need authentication between all routers and victim computer to make sure the victim computer can trust all the routers. If $p > 0.5$ then this protocol is robust against a single attacker since there is no way for an attacker to insert a "false" router into the paths by contributing more samples than the downstream router. However, there are two serious limitations of this approach. First, inferring the total router order from the distribution of samples is a slow process. Routers far away from the victims contribute few samples, it requires receiving a large amount of packets from the attackers to reconstruct the path, especially when p is larger than 0.5. For instance, if $h = 15$ and $p = 0.51$, the receiver must receive more than 42000 packets on average before it receive a single sample form the farthest router. To insure the order is correct with 95% certainty requires more than seven times of that number [8]. Second, if there are multiple attackers, the multiple routers may exist at the same distance and sampled with the same probability, so that we can not simply find the full attack path by sorting the marked IP addresses. Therefore, this technique is not robust against multiple attackers.

4. The Edge Sampling Protocol

This protocol is also presented by [8]. An alternative to the node sampling protocol is the edge sampling protocol. The edge sampling protocol explicitly marks the edges of the attack paths, instead of a single node. An edge represent one hop from one computer to another in the network, it includes a starting computer (here we will call it *start*) and ending computer (here we will call it *finish*, since end conflict with our keyword *end* in AP notation). For this protocol, we need to reserve two static address-sized fields in each message, *start* and *finish,* and also at least 5 bits for h (distance of the destination computer to the *finish* that carried by the message, we say 5-bit here since most of the messages travels in the Internet take no more than 20 hops) in the IP header of each message. This protocol requires computers on the network to add its IP address as *start* into a message as that message travels through the computers at a small probability, p.

When a computer receives a message, it chooses a random number *x*. If $x<p$, then it marks its IP address into the *start* field, marks the next computer's IP address, where this message will be routed to, into the *finish* field and assign *h* to be 0 (hop). If $x \geq p$, that computer will simply increment *h* by one hop. The destination computer will collect these edges once it discovers it is under attack, and will start using these edges to construct a tree with the destination computer as the root of the tree.

In the following section we will present the formal specification of the edge sampling protocol in AP notation. This protocol has the same *i, n, p[i:0..n], p, m, N, rtb, attack, x, dst* and *j* with the node sampling protocol. It has an additional input array *len* which takes any process as indices and returns the distance between *i* and that process, assume it is possible for us to find out this information. Also, it have variables *start* (the starting IP address of an edge), *finish* (the ending IP address of an edge), *h* (the distance from *finish* to the current process) where they are all in the range from 0 to *n.* It also has a two-dimensional array *edge* of integer takes *start* and *finish* as indices, *edge* starts with all 0's, adds one if there is a sampled edge between *start* and *finish*. For each packet, *pkt,* in this protocol, it requires to carry four fields, *start*, *finish*, *h* and *dst*.

The victim computer uses the edges sampled in these packets to create a graph, which leads back to the true source of the attack. Here we denote *T* be a tree with root *v*, each edge in *T* is a tuple of (*start*, *finish*, *h*). For each packet *pkt* from attacker, if $h = 0$ then insert edge (*start*, *v*, 0) into *T*, else insert edge (*start*, *finish*, *h*) into *T*. Note that any packets sent by the attacker will necessarily have a distance greater or equal to the length of than the routers on the path of attack. Therefore we can prevent a single attacker forge sampled edges by removing any edge which its *h* doesn't equal to distance from *end* to *v* in *T*. Finally, extract path by enumerating acyclic paths in *T*.

The following is the AP notation of this protocol:

process p[i:0..n]

const p, m

inp N: **set** { g | g is a neighbor of p[i] }

```
        rtb:                    array[0..n] of N
        attack:                 boolean
        len:                    array[0..n] of 0..n
                                { len[x]=shortest length between node x and node i }

var     x:                      0..m
        start, finish, dst, h:  0..n
        edge:                   array[0..n, 0..n] of integer    { init all 0's }

par     j:                      N

begin
        true →
                dst := any;
                if i = dst →    skip
                □ i ≠dst →      start, finish, h := i, rtb[dst], 0;
                                send pkt (start, finish, h, dst) to p[rtb[dst]]
                fi

        □ rcv pkt(start, finish, h, dst) from p[j] →
                if dst ≠ i →
                        x := random;
                        if x < p → start, finish, h := i, rtb[dst], 0
                        □ x ≥ p → h := h + 1
                        fi;
                        send pkt(start, finish , h, dst) to p[rtb[dst]]

                □ dst = i →
                        if len[finish] = h →
                                        edge[start, edge] := edge[start, finish] + 1

                        □ len[finish] ≠ h →
                                        { incorrect edge }
                                        skip
                        fi;

                        if  ~attack → skip
                        □  attack   → { use "edge" to detect location of attacker }
                        fi
                fi
end
```

This protocol includes two actions. Within process *i*, at any given time, first action generates a destination, *dst*, at random. Then it will check whether the *dst* it just generated equals to its own IP address, *i*. If it does, this action will do nothing else,

otherwise it marks the *start* field with its own IP address, *i*, marks the *finish* field with the next computer's IP address, *rtb[dst]*, where the message routes to next, and assigns *h* to 0. Then it generates a packet, *pkt*, with *start, finish, h* and *dst*, and routes this packet to the next computer according to the routing table *rtb*. The second action receives a packet from *p[j]*. This action will check whether *i* is the ultimate destination first. If it is not the destination, the computer *i* will choose a random *x*. If $x<p$, this computer marks the *start* field with its own IP address, *i*, marks the *finish* field with the next computer's IP address, *rtb[dst]*, where the message routes to next, and assigns *h* equals to 0. If $x \geq p$, it just increments *h* by one hop. Finally this computer will send out the packet to the next computer, *p[rtb[dst]]*. If *i* is the destination, the process start to store data in array *edge* using that *start* and *edge* it just received. It sets *edge[start, finish] = true*, , iff *h* equals to the distance from *i* to *finish*, since *len[finish]* does equal *h* is impossible to be the correct data. Then this action checks whether computer *i* is in *attack* mode. If *i* is not under attack, it does not do anything. If *i* is under attack, we detect the attacker by enumerating acyclic paths from array *edge* and *len.*

This protocol is better than the node sampling protocol in the sense it doesn't relay on sampling rank approach to distinguish "false" samples, we can choose an arbitrary values for the marking probability *p*. Now say the farthest router is *h* hops away, it is efficient to choose $p=1/d$. By reducing *p*, we will not need as many as packets needed by node sampling protocol to identify the path of attack. Edge sampling is also robust against DoS attack by single attacker, because it is impossible for any edge closer than the closest attacker to be spoofed, due to the distance determination we discussed in the previous section of how to construct the attack path. Conversely, this is also means in a distributed attack that is impossible to trust the content of any edge farther away from the closest attacker. Therefore, it is not robust against DDoS. Another significant limitation of this approach is that it requires 72 bits additional space in the IP header of a packet and therefore it is not backward compatible. It is expensive to append additional bit to IP header on the fly. An idea to solve the backward compatible problem base on this idea is to compress the data, which is we will next introduce Compressed Edge Sampling Protocol.

5. The Compressed Edge Sampling Protocol

The idea of this protocol is presented in [9] by *Song* and *Perrig* (it was called Advanced Marking Scheme I in [9]). To compress the size of extra field required in the IP header of the edge sampling protocol we introduce the compressed edge sampling protocol. Instead of keep three fields *start, finish* and *h,* now we only need to keep two fields, *cedge* (stands for compressed edge) and *h* in the IP header. This protocol requires computers on the network to add *cedge* into a message as that message travels through the computers at a probability, *p*. In this protocol we will use the operation of XOR, $\oplus$. After a computer receives a message, it chooses a random number *x*. If $x<p$, then it uses its IP address XOR with the next computer's IP address, where this message will be routed to, then marks the XOR result into the *cedge* field and assigns *h* to be 0 (hop). If $x \geq p$, at computer simply increments the *h* by one hop. As a requirement of this protocol we need to have an upstream router map available as a road-map to the victim computer to perform a breadth-first search from the victim computer. It was shown in [9] section 5.1, that is possible to construct such a map. Recall $b \oplus a \oplus b = a$, then the *cedge* receives at the destination computer can be used to decode the previous *cedge*, and so on, hop-by-hop until we reach the attacker. Then we can construct a tree in the similar matter with edge sampling protocol. In the original design of this protocol [9], *Song* and *Perrig* used a hash functions to apply on *cedge* in order to reduce the bit-size of *cedge*, but here for simplification we will omit the usage of hash function.

In the following section we will present the formal specification of this protocol in AP notation. This protocol has the same *i, n, p[i:0..n], p, m, N, rtb, attack, len, x, dst, edge* and *j* with the edge sampling protocol. Input array *connt* takes two processes as indices, it returns true if there is a direct connection in between these two processes. In addition, it has variables *cedge* (the XOR value of a starting computer's IP address and the next computer's, IP address, where the message will route to) as an integer, *h* (the distance from start computer of an end point of *cedge* to the current process), *t* and *z* variables in the range from 0 to *n.* For each packet, *pkt,* in this protocol, it requires to carry three field *cedge*, *h* and *dst*.

The following is the AP notation of this protocol:

```
process   p[i:0..n]

const     p, m

inp       N:          set { g | g is a neighbor of p[i] }
          rtb:        array[0..n] of N
          attack:     boolean
          len         array[0..n] of 0..n
                      { len[x]=shortest length between node x  and node i }

var       x:          0..m { compressed edge }
          cedge:      integer
          dst, h:     0..n
          edge:       array[0..n, 0..n] of integer     { init all 0's }
          u, v:       0..n
          z:          0..n + 1

par       j:          N

begin
          true →
               dst := any;
               if i = dst →   skip
               □ i ≠dst →     cedge, h := i ⊕ rtb[dst], 0;
                              send pkt (cedge, h , dst) to p[rtb[dst]]
               fi

          □ rcv pkt(cedge, h, dst) from p[j] →
               if dst ≠ i →
                         x := random;
                         if x < p → cedge, h := i ⊕ rtb[dst], 0
                         □ x ≥ p → h := h + 1
                         fi;
                         send pkt(cedge , h, dst) to p[rtb[dst]]

               □ dst = i →
                         CMPEDG (in cedge, h, out u, v);
                         edge[u,v] := edge[u,v] + 1;
                         if ~attack   → skip
                         □ attack → { use array "edge" to detect the location of attacker }
                         fi
               fi
end
```

CMPEDG (**in** cedge, h, **out** u, v) ::

For every edge (u, v) in the network
if len[u] = h $\wedge$ u $\oplus$ v = cegde $\rightarrow$ return u, v
□ len[u] $\neq$ h $\vee$ u $\oplus$ v $\neq$ cedge $\rightarrow$ { incorrect edge }
skip
fi

This protocol includes two actions. Within process *i*, at any given time, first action generates a destination, *dst*, at random. Then it will check whether the *dst* it just generated equals to its own IP address, *i*. If it does, this action will do nothing else, otherwise it marks the *cedge* field with the XOR value of its own IP address, *i*, and the next computer's IP address, *rtb[dst]*, where the message routes to next, and assigns *h* to 0. Then it generates a packet, *pkt*, with *cedge, h* and *dst*, and routes this packet to the next computer according to the routing table *rtb*. The second action receives a packet from *p[j]*. This action will check whether *i* is the ultimate destination first. If it is not the destination, computer *i* will choose a random *x*. If $x<p$, this computer marks the *cedge* field with the XOR value its own IP address, *i*, and the next computer's IP address, *rtb[dst]*, where the message routes to next, and assigns *h* to 0. If $x \geq p$, it just increments *h* by one hop. Finally this computer will send out the packet to the next computer, *p[rtb[dst]]*. If *i* is the destination, the process will run a function *CMPEDG*. *CMPEDG* takes input *cedge* and *h*, outputs start point *u*, end point *v* of all the edges that have *len[u]* = *h*. Then process *i* will increment *edge[u,v]* by 1 with the data outputs from *CMPEDG*. This process *i* will also check whether it is in *attack* mode. When the destination is not under attack, it simply does nothing more. If it is under attack, it uses array *edge* to detect location of attackers.

This protocol is more robust than the previous protocols, with not much complex computation, it need a lot less packets to reconstruct the attack path. It is also better in dealing with DDoS ([9] section 3.3). If we take the hash value of each edge, the hashed addresses will fit into the fragmentation field of the IP header, which is not been used very often, this solves the backward compatibility problem.

6. The Algebraic Polynomial Protocol

The algebraic polynomial protocol is suggested in [10] (see section 4.1, Fall Path Encoding). This protocol uses a scheme of algebraic approach for encoding traceback information. The basic idea is that for any polynomial *f(y)* of degree *h-1* in the prime field *GF(p)*, we can recover the coefficients of *f(y)* given the values of *f(y)* evaluated at *h* distinct points. Let the coefficients *of f(y)* denoted $A_1, A_2, \ldots A_h$, be the 32-bit IP addresses of routers on a path. We associate a random id y_j with the *j*th packet (we need to use different id for each packet). We can evaluate $f_i(y_j)$ as the packet travels along the path, accumulating the result of the computation in a running total along the way. At the first router along the path, let $f_1(y_j) = A_1$, each of the router A_i along the path calculates $f_i(y_j) = (f_{i-1}(y_j) * y_j + A_i) \bmod r$, where *r* is the smallest prime larger than $2^{32} - 1$, the *mod* operation ensures that the size of *f(y)* won't excess 32 bits. At the packet's destination *f(y)* will equal to $A_1 y^{(h-1)} + A_2 y^{(h-2)} + \ldots + A_{h-1} y + A_h \bmod r$. When enough packets from the same path reach the destination, then *f(y)* can be reconstructed by interpolation. The interpolation calculation might be a simple set of linear equations (see figure 2). To make this protocol easy to write, by Horner's rule, $(((A_1 y + A_2)y + A_3)y + A_4 = A_1 y^3 + A_2 y^2 + A_3 y + A_4$. A router doesn't need to know the total length of the path of its position in the path for this computation of *f(y)*.

$$\begin{bmatrix} 1 & y_1 & y_1^2 & \ldots & y_1^{h-1} \\ 1 & y_2 & y_2^2 & \ldots & y_2^{h-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & y_h & y_h^2 & \ldots & y_h^{h-1} \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_h \end{bmatrix} = \begin{bmatrix} f_h(y_1) \\ f_h(y_2) \\ \vdots \\ f_h(y_h) \end{bmatrix}$$

Figure 2. Path *A1, A2, …An* can be reconstructed by solving this matrix equation over *GF(p)*

In the following section we will present the formal specification of this protocol in AP notation. This protocol has the same *i, n, p[i:0..n], N, rtb, attack, x* and *dst* with the previous protocol discussed. Constant *r* is the smallest prime larger than 2^{32} -1. Variable *h* is the degree of each polynomial *f(y)*, *y* is packet id used to computer *f(y)* and *z* is *f(y)*. *bff* is a set to store triple of *(h, y, z)*. *txt* is the text part of the message carried by packet. *g*

is a parameter to represent all the processes in the network. Each packet requires to carry five fields *dst*, *txt*, *h*, *y* and *z*.

The following is the AP notation of this protocol:

```
process     p[i:0..n]

const       r:          integer {r is the smallest prime larger than 2^32 -1}

inp         N:          set {g | g is a neighbor of p[i]},
            rtb:        array[0..n] of N,
            attack:     boolean,

var         x:          0..m
            dst:        0..n,
            txt:        1..n,
            h:          1..n-1
            y:          0..r-1,
            z:          0..r-1,                 {init 0}
            bff:        set{ (h, y, z) }
            node:       0..n
            count:      array [0..n] of integer {init all 0's}

par         g:          N

begin
            true  → dst := any;
                    if  dst = i → skip
                    □   dst ≠ i → txt, h, y, z := any, 1, random, i;
                                  send pkt(dst, txt, h, y, z) to p[rtb[dst]]
                    fi

            □ rcv pkt(dst, txt, h, y, z) from p[g] →
                    if dst ≠ i →
                            x := random;
                            if x < p →
                                    h, y, z := 1, random, i;
                              □ x ≥ p →
                                    h, z := h+1, ((z*y)+i) mod r;
                              fi;
                              send packet(dst, txt, h, y, z) to p[rtb[dst]]

                    □ dst = i →
                              bff := bff ∪ { (h, y, z) }
                              CMPNOD(in bff, out node);
                              count[node]:=count[node]+1;
```

if ~attack → **skip**
□ attack → { use array "count" to detect location of attacker }
fi

fi

end

CMPNOD(**in** bff, **out** node)::

for every *h* triples of the form {*(h, y_1, z_1), (h, y_2, z_2), …, (h, y_h, z_h)}* in *bff*,
solve the h equations: $(A_1 + A_2y_1 + \ldots + A_h(y_1)^{h-1})$ ***mod*** $r = z_1$

$(A_1 + A_2y_2 + \ldots + A_h(y_2)^{h-1})$ ***mod*** $r = z_2$

$\vdots$

$(A_1 + A_2y_h + \ldots + A_h(y_h)^{h-1})$ ***mod*** $r = z_h$

If the computed $A_1, A_2, \ldots, A_h$ correspond to a path from node *i* to a node *u,* then return node *u.*

This protocol includes two actions. Within process *i*, at any given time, first action generates a destination, *dst*, at random. Then it will check whether the *dst* it just generated equals to its own IP address, *i*. If it does, this action will do nothing else, otherwise it assigns *txt* to *any*, *h* to 1, *y* to *random* and *z* to *i*, then it will generate packet with *dst, txt, h, y* and *z* and route it to the next computer according to the routing table. The second action receives a packet, first it checks whether the current process is the destination. If the current process is not the destination, the process assign *x* to a random integer. If $x<p$, this process assigns *h* to 1, *y* to *random* and *z* to *i*. If $x \geq p$, this process needs to recomputed *z* by assigning *z* to $(z * y + i) \bmod r$, and increment *h* by 1. Then resend the packet with new *z* and *h* to the next computer. If it is the destination, it inserts triple *(h, y, z)* in set *bff* and run function *CMPNOD(**in** bff, **out** node)*. Then it increments *count[node]* by 1 with the *node* return from *CMPNOD*. Also this process checks whether it is under attack mode. If this process is not under attack, then it simply does nothing else. If it is under attack, it uses array *count* to detect the location of attackers.

This protocol shows a good approach to the traceback marking protocol. It requires very low computation overhead on the routers but it is not easy to compute the attack path. But a draw back to this protocol is that adding additional field to store the polynomial value could be difficult. Also, it is impossible for a router to know whether it

is the first router on the attack path, therefore it would be easy for an attacker to forge that information.

5. Conclusion

In this paper, I have given some knowledge of Denial-of-Service Attacks and Distributed Denial-of-Service Attacks. I also have talked about the history of Denial-of-Service Attack and the various approaches to encounter with this attack. Finally, I have presented four traceback marking protocols in detail and provided formal specifications of each protocol. Although, each of these protocols still has limitations to deal with Distributed Denial-of-Service Attacks and also requires modification of the routers on the Internet, they have improved the effectiveness on dealing with Denial-of-Service Attacks.

6. Reference

[1] CERT® Coordination Center, "Denial of Service Attack", url: http://www.cert.org/tech_tips/denial_of_service.html#1.

[2] Macu Networks, "The Brief but Disturbing History of DDoS Attacks", url: http://www.mazunetworks.com/ddos_library/chronology_print.html.

[3] CNN.com, "Cyber-attacks batter Web heavyweights", February 9, 2000 url: http://www.cnn.com/2000/TECH/computing/02/09/cyber.attacks.01/index.html.

[4] Advanced Networking Management Lab (ANML) Distributed Denial of Service Attacks(DDoS) Resources, "DDoS History in Brief", url: http://www.anml.iu.edu/ddos/history.html.

[5] G. Sager, "Security Fun with OCxmon and cflowd", presented at the Internet 2 Working Group, Novmber1998.

[6] R. Stone, "CenterTrack: An IP overlay network for tracing DoS floods", in Proc, 2000 USENIX Security Symp., July 2000, pp.199-212.

[7] S. M. Bellovin, "ICMP traceback messages", Internet Draft: draft-bellovin-itrace-00.txt, 2000.

[8] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Network Support for IP Traceback", IEEE/ACM transactions on networking, vol. 9, No. 3, June 2001.

[9] D. Song and A. Perrig, “Advanced and authenticated marking schemes for IP Traceback”, in Proc. IEEE INFOCOM, vol. 2, April 2001, pp. 878-886.

[10] D. Dean, Matt Franklin, and Adam Stubblefield, “An Algebraic Approach to IP Traceback”,
url: http://www.isoc.org/isoc/conferences/ndss/01/2001/papers/dean01.pdf.

[11] searchSecurity.com, “distributed denial-of-service attack”,
url: http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci557336,00.html.

[12] Silicon Graphics Inc. Security Advisory, “TCP SYN and Ping Denial of Service Attacks”, CERT(sm) Advisory CA-96.21, CERT(sm) Advisory, CA-96.26, Silicon Graphics Advisory 19960901, December 24, 1996.

[13] M. G. Gouda, E. N. Elnozahy, C.-T. Huang, and T. M. McGuire, “Hop Integrity in Computer Networks”, Eighth International Conference on Network Protocols, November 2000,
url: http://www.research.ibm.com/arl/projects/papers/hopintegrity.pdf.

[14] J. Glave, “Smurfing cripples ISPs”, Wired Technology News, 1998,
url: http://www/wired.com/news/technology/story/9506.html.

[15] H. Burch and B. Cheswick, “Tracing Anonymous Packets to Their Approximate Source”, in Proc. 2000 USENIX LISA Conf. December 2000, pp319-327.