An Algebraic Approach to IP Traceback

Drew Dean
Xerox PARC
ddean@parc.xerox.com

Matt Franklin*
U.C. Davis
franklin@cs.ucdavis.edu

Adam Stubblefield†
Rice University
astubble@rice.edu

Abstract

We present a new solution to the problem of determining the path a packet traversed over the Internet (called the traceback problem*) during a denial of service attack. Previous solutions to this problem have suffered from combinatorial explosion, and are unable to scale to realistically sized networks. This paper reframes the traceback problem as a polynomial reconstruction problem and uses techniques from algebraic coding theory to provide robust methods of transmission and reconstruction. We also present an implementation of one promising parameterization that is efficient, backwards compatible, and incrementally deployable.*

1. Introduction

A denial of service attack is designed to prevent legitimate access to a resource. In the context of the Internet, an attacker can "flood" a victim's connection with random packets to prevent legitimate packets from getting through. These Internet denial of service attacks have become more prevalent recently due to their near untraceability and relative ease of execution [8]. Also, the availability of tools such as Stacheldraht [10] and TFN [11] greatly simplify the task of coordinating hundreds or even thousands of compromised hosts to attack a single target.

These attacks are so difficult to trace because the only hint a victim has as to the source of a given packet is the source address, which can be easily forged[1]. Also, many attacks are launched from compromised systems so finding the source of the attacker's packets may not lead to the attacker. Disregarding the problem of finding the person responsible for the attack, if a victim was able to determine the path of the attacking packets in near real-time, it would be much easier to quickly stop the attack. Even finding out partial path information would be useful because attacks could be throttled at far routers.

This paper presents a new scheme for providing this traceback data by having routers embed information randomly into packets. This is similar to the technique used by Savage, *et al* [19], with the major difference being that our schemes are based on algebraic techniques. This has the advantage of providing a scheme that offers more flexibility in design and more powerful techniques that can be used to filter out attacker generated noise and separate multiple paths. Our schemes share similar backwards compatibility and incremental deployment properties to the previous work.

More specifically, our scheme encodes path information as points on polynomials. We then use algebraic methods due to Guruswami and Sudan [13] for reconstructing these polynomials at the victim. This appears to be a powerful new approach to the IP traceback problem. We predict that our basic framework will lead to useful variations and alternatives in the near future.

The rest of the paper is organized as follows: Section 2 discusses related work, Section 3 contains an overview of the problem and our assumptions, Section 4 presents our approach for algebraically coding paths, Section 5 discusses the issue of encoding this data in IP packets, Section 6 contains an analysis of our proposed scheme, Section 7 discusses future work, and Section 8 concludes.

2. Related Work

The idea of randomly encoding traceback data in IP packets was first presented by Savage, *et al* [19]. They proposed a scheme in which adjacent routers would randomly insert adjacent edge information into the ID field of packets. Their key insight was that traceback data could be spread across multiple packets because a large number of packets was expected. They also include a distance field which allows a victim to determine the distance that a particular edge is from the host. This prevents spoofing of edges from closer than the nearest attacker. The biggest disadvantages of this scheme is the combinatorial

*Work done while employed at Xerox PARC

†Work done during a summer internship at Xerox PARC

[1] Ingress filtering is helping to mitigate this problem by preventing a packet from leaving a border network without a source address from the border network [12]. Attackers have gotten around this by choosing legitimate border network addresses at random.

explosion during the edge identification step and the few feasible parameterizations. The work of Song and Perrig provides a more in depth analysis of the faults of this scheme [21].

There have been two other notable proposals for IP traceback since the original proposal. Bellovin has proposed having routers create additional ICMP packets with traceback information at random and a public key infrastructure to verify the source of these packets [5]. Song and Perrig have an improved packet marking scheme that copes with multiple attackers [21]. Unfortunately, this scheme requires that all victims have a current map of all upstream routers to all attackers (although Song and Perrig describe how such maps can be maintained). Additionally, it is not incrementally deployable as it requires all routers on the attack path to participate (although Song and Perrig note that it also suffices for the upstream map to indicate which routers are participating).

We refer the reader to Savage's paper for a discussion of other methods to detect and prevent IP spoofing and denial of service attacks.

The algebraic techniques we apply were originally developed for the fields of coding theory [13] and machine learning [3]. For an overview of algebraic coding theory, we refer the reader to the survey by Sudan [23] or the book by Berlekamp [7].

3. Overview

This paper addresses what Savage, *et al* call the *approximate traceback problem*. That is, we would like to recover all paths from attacker to victim, but we will allow for paths to have invalid prefixes. For example, for the network shown in Figure 1, the true path from the attacker A_1 to the victim V is $R_4R_2R_1$. We will allow our technique to also produce paths of the form $R_2R_6R_4R_2R_1$ because the true path is a suffix of the recovered path.

Our family of algebraic schemes was motivated by many of the same assumptions as used in previous work with two notable additions (numbers 8 and 9).

1. Attackers are able to send any packet
2. Multiple attackers can act together
3. Attackers are aware of the traceback scheme
4. Attackers must send at least thousands of packets
5. Routes between hosts are in general stable, but packets can be reordered or lost
6. Routers can not do much per-packet computation
7. Routers are not compromised, but not all routers have to participate
8. It is difficult to change the marking algorithm used by routers
9. It is easy to change the reconstruction algorithm used by victims

We will focus discussion here on these last two assumptions. The reasoning behind the others is well covered by Savage's paper [19]. Changing the algorithm used by routers to mark packets would require a hardware change in deployed routers. This presents severe problems in terms of cost, deployability, and access as a router would need to be taken offline while a new piece of hardware was inserted. On the other hand, the reconstruction algorithm will almost certainly be implemented in software, which is (relatively) easily modified. Also, because the reconstructor only needs to be running during an attack, taking it offline for upgrades is not detrimental.

These last two assumptions motivate us to look for a scheme which has acceptable performance at the present as well as an ability to improve in the future with only changes in the reconstruction step. We therefore have chosen an algebraic approach rooted in coding theory, namely that of polynomial evaluation. Over the past few years, techniques have repeatedly become more powerful in this field and we have no reason to suspect this will change in the near future. Current techniques already allow us to separate multiple paths and filter out noise with acceptable bounding conditions [3, 23, 13].

4. Algebraic Coding of Paths

We will now present a series of schemes that use an algebraic approach for encoding traceback information. All of these schemes are based on the principal of reconstructing a polynomial in a prime field. The basic idea is that for any polynomial $f(x)$ of degree d in the prime field $GF(p)$, we can recover $f(x)$ given $f(x)$ evaluated at $(d+1)$ unique points. Let $A_1, A_2, \ldots, A_n$ be the 32-bit IP addresses of the routers on path P. Let $f_P(x) = A_1x^{n-1} + A_2x^{n-2} + \ldots + A_{n-1}x + A_n$. We associate a packet id x_j with the jth packet. We then somehow evaluate $f_P(x_j)$ as the packet travels along the path, accumulating the result of the computation in a running total along the way. When enough packets from the same path reach the destination, then f_P can be reconstructed by interpolation. The interpolation calculation might be a simple set of linear equations, if all of the packets received at the destination traveled the same path. Otherwise, we will need to employ more sophisticated interpolation strategies that succeed even in the presence of incorrect data or data from multiple paths [6, 24, 13].

A naive way to evaluate $f_P(w)$ would be to have the jth router add A_jw^{n-j} into an accumulator that kept the running total. Unfortunately, this would require that each

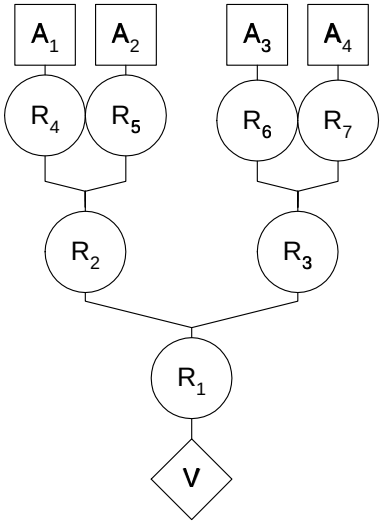

Figure 1. Our example network.

router know its position in the path and the total length of the path. We could eliminate the need for each router to know the total length of the path (while still requiring each router to know its position in the path) by reordering the coefficients of f_P: $A_1 + A_2 w + A_3 w^2 + \ldots + A_n w^{n-1}$. However, we can do even better by sticking with our original ordering, and using an alternative means of computing the polynomial. Specifically, to compute $f_P(w)$, each router A_j multiplies the amount in the accumulator by w, adds A_j, and returns the result to the accumulator, and passes the packet on to the next router in the path (Horner's rule [14]). For example, $((((0 \cdot w) + A_1)w + A_2)w + A_3)w + A_4 = A_1 w^3 + A_2 w^2 + A_3 w + A_4$. Notice that the router doesn't need to know the total length of the path or its position in the path for this computation of f_P.

We will use this polynomial evaluation trick for all of our algebraic schemes. What will vary is (a) whether we use polynomials that capture the entire path or just a fragment of the path, and (b) whether every router will participate deterministically or non-deterministically to outwit a malicious attacker.

4.1. Full Path Encoding

The simplest scheme that uses this algebraic technique encodes an entire path. At the beginning of a path, let $FullPath_{0,j} = 0$. Each router i on the path calculates $FullPath_{i,j} = (FullPath_{i-1,j} \cdot x_j + A_i) \bmod p$ where x_j is a random value passed in each packet, A_i is the router's IP address and p is the smallest prime larger than $2^{32} - 1$. The value $FullPath_{i,j}$ is then be passed in the packet, along with x_j, to the next router. At the packet's destination $FullPath$ will equal $(A_n x^{n-1} + A_{n-1} x^{n-2} + \cdots + A_2 x + A_1) \bmod p$, which can be reconstructed by solving the following matrix equation over $GF(p)$:

$$\begin{pmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \ldots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \ldots & x_n^{n-1} \end{pmatrix} \begin{pmatrix} A_1 \\ A_2 \\ \vdots \\ A_n \end{pmatrix} = \begin{pmatrix} FullPath_{n,1} \\ FullPath_{n,2} \\ \vdots \\ FullPath_{n,3} \end{pmatrix}$$

As long as all of the x_i's are distinct, the matrix is a Vandermonde matrix (and thus has full rank) and is solvable in $O(n^2)$ field operations [17].

Assuming that we get a unique x_j in each packet, we can recover a path of length d with only d packets. The downside, however, is that this scheme would require $\log_2(p) + \lceil \log_2(d) \rceil$ bits per packet (the first term is the encoding of the running $FullPath$ and the second term is the encoding of the x_j's). Even for modest maximum path lengths of 16, the space required (36 bits) far exceeds the number of bits available to us in an IP header.

We can trade off bits-needed for packets-needed by splitting a router's IP address into c chunks and adding $\lceil \log_2(c) \rceil$ bits to indicate which chunk was represented in a given packet. We could also reduce the order of the field, p, to the smallest prime larger than $2^{\lceil 32/c \rceil}$. If we chose to split the 32 bit IP address into 4 chunks we would need $\log_2(257) + \log_2(16) + \log_2(4) = 15$ bits per packet and $4d$ packets. While this is an improvement, a better technique would be to have each router add all of its chunks into each packet. So, instead of spreading its c chunks among c packets by adding one coefficient to the polynomial, each router would add c coefficients to the polynomial in each packet. That is, each router would update $FullPath$ c times, substituting each chunk of their IP address in order. The destination could then trivially reconstruct the IP addresses by interpolating to

recover $\tilde{f}_P(x) = A_{1,1} + A_{1,2}x + \ldots + A_{1,k}x^{k-1} + A_{2,1}x^k + \ldots + A_{n,k}x^{nk-1}$, where $A_{j,1}, A_{j,2}, \ldots, A_{j,k}$ are the successive chunks of A_j. For $c = 4$, this technique requires only $\log_2(257) + \log_2(16) = 13$ bits and $4d$ packets. This second technique is thus clearly better. In Section 6 we use a slightly different chunking strategy based on the Chinese Remainder Theorem [2].

4.2. Random Full Path Encoding

The astute reader has probably noticed a serious flaw in the above schemes; we require $FullPath_{0,j} = 0$. This implies that there is some way for a router to know that it is the "first" participating router on a particular path. In the current Internet architecture there is no reliable way for a router to have this information. We must therefore extend our scheme to mitigate this problem.

In our revised scheme a router first flips a weighted coin. If it came up tails the router would assume it was not the first router and simply follow the $FullPath$ algorithm presented above, adding its IP address (or IP address chunk) data. On the other hand, if the coin came up heads, the router would assume it was the first router and randomly choose a x_j to use for the path. We will refer to this state as "marking mode."

At the destination, we would receive a number of different polynomials, all representing suffixes of the full path. In our example network, packets from A_1 could contain $R_4R_2R_1$, R_2R_1, or R_1. From now on, we'll refer to each of these path suffixes as "virtual paths", because algebraically they are indistinguishable from full paths. Discriminating these virtual paths would be a real nightmare if we were not able to leverage the power of our algebraic approach. It turns out that recovering these paths is the well studied problem of reconstructing mixed algebraic functions [3]. We can therefore simply appeal to the current best algorithm for solving this problem. In the future, as better algorithms are available they could be implemented at the destination *without changing anything on the routers*.

The algorithm we will use in our analysis is due to Guruswami and Sudan [13]. If we have N total packets, it allows us to recover all virtual paths of length d for which we have at least $\sqrt{N(d-1)}$ packets. For example, if we assume that we analyze $10,000$ packets at a time and want to recover all virtual paths of length 17 or less, we would need to ensure that we receive 400 packets from each virtual path. Generally, we would need to expect packets from a router at distance d with a probability of no less than $\sqrt{N(d-1)}/N$. Since the probability of getting a packet from a router Δ hops away is $p(1-p)^{\Delta-1}$, where p is the probability that a router is in marking mode, we come up with the inequality $p(1-p)^{\Delta-1} > \sqrt{N(d-1)}/N$. We would like to recover paths of length d so this becomes $p(1-p)^{d-1} > \sqrt{N(d-1)}/N$. Unfortunately, this inequality has only negative and imaginary solutions for any of the values of N and d that interest us.

To remedy this problem, we change our marking strategy slightly. Whenever a router receives a packet, it still flips a weighted coin. But now, instead of simply going into marking mode for one packet when the coin comes up heads, the router will stay in marking mode for the next τ packets it receives. The router should do this coin flip for each pair of interfaces and not as a global state. Our goal now is not to recover all virtual paths in one run, but instead to recover only a few paths per run. To accomplish this we should choose $\tau = \sqrt{Nd} + \varepsilon$ where ε is a factor designed to allow small overlaps in routers on the same path both being in marking mode. Our tests have shown that ε can be small compared to $\sqrt{Nd}$.

To analyze this scheme we simulated thousands of runs of $10,000$ packets each through a paths of length 48, which we feel is a reasonable upper bound on expected path lengths. The results of these tests show that the optimum choice for p in this scenario is around 10^{-5}. Even with an "optimum" probability, we find that we must receive more than $100,000$ packets in order to reconstruct even more moderate length 35 virtual paths.

An even bigger problem than the number of packets needed to reconstruct these paths is that attackers can cause more false paths than true paths to be received at the victim. This is due to the fact the our choice of a small p creates large number of packets in which no router on the packet's path is in marking mode. The attacker can thus insert any path information he wishes into such packets. Because the attacker can generally find out the path to his victim (using *traceroute*, for example) he can compute $FullPath_{0,j} = (FakePath_j/x_j^n - A_n x_j^{n-1} - \ldots - A_0) \bmod p$. This choice will cause the victim to receive $FullPath_j = FakePath_j$. When trying to reconstruct paths, the victim will have no indication as to which paths are real and which paths are faked. Two solutions to this problem are to increase p or to store a hop count in the packet that each participating router would increment. Increasing the probability makes it even harder to receive long paths, so we do not think that is a viable option. Adding a hop count would prevent an attacker from forging virtual paths that are closer than its actual distance from the victim but would require $\lceil \log_2(d) \rceil$ more bits in the packet. While either of these solutions may be appropriate in some situations, we feel that the scheme presented in the next section is a better alternative.

4.3. Random Partial Path Encoding

Our final scheme is a further generalization of the random full path encoding method. We add another parameter, ℓ, that represents the maximum length of an encoded

Bits per packet	Polynomial Degree $\lceil 32/c \rceil (\ell+1)-1$	Bits for Accumulator $\lceil 32/c \rceil$	Bits for Randomness $\log_2(\lceil 32/c \rceil (\ell+1))$
19	3	16	2
15	5	11	3
12	7	8	3
11	11	6	4
10	13	5	4
9	15	4	4

Table 1. Parameterizations of Random Partial Path Encoding (all assume $\ell = 1$)

path. The value of ℓ is set by the marking router and decremented by each participating router who adds in their IP information. When the value reaches 0, no more routers add in their information. For example, in the full path encoding scheme $\ell = \infty$, while $\ell = 1$ would represent encoding of edges between routers.

The purpose for this change is to decrease the maximum d used in the reconstruction bound ($\sqrt{N\ell}$ for $0 < \ell < \infty$) in order to reduce the number of packets needed out of a given set or packets to recover a route. Of course we do not get anything for free; this adds $\lceil \log_2(\ell+1) \rceil$ bits to the packets. On the other hand, we now have $p(1-p)^{x-1} > \sqrt{N(\lceil 32/c \rceil (\ell+1)-1)}/N$ which *does* have solutions that are interesting to us. Table 1 shows some of these interesting combinations.

Of course, if ℓ is less than the true path length, then reconstruction finds arbitrary subsequences of the path (not just suffixes as in Full Path encoding). The reconstructor still has some work to do to combine these subsequences properly. Thus reconstruction in this scheme has an algebraic step followed by a combinatorial step.

In section 6, we will be looking at the parameterization where $\ell = 1$ and $d = 5$. This encodes edges between adjacent participating routers at a cost of 15 bits per packet. In the next section, we will discuss where to fit the 15 bits of information in an IP packet.

5. Encoding Path Data

We now need a way to store our traceback data in IP packets. We will try to maximize the number of bits available to us while preserving (for the most part) backwards compatibility.

5.1. IP options

An IP option seems like the most reasonable alternative for storing our path information. Unfortunately, most current routers are unable to handle packets with options in hardware [4]. Even if future routers had this ability, there are a number of problems associated with this approach as presented by Savage, *et al* [19]. For all of these reasons we have concluded that storing data in an IP option is not feasible.

5.2. Additional Packets

Instead of trying to add our path data to the existing IP packets, we could instead send the data out of band using a new protocol that would encapsulate our data. While this may have limited uses for special cases (such as dealing with IP fragments), a general solution based on inserting additional packets requires a means of authenticating these packets. This is because, presumably, the number of inserted packets is many orders of magnitude less than the number of packets inserted by the attacker. Thus, because we assume that an attacker can insert any packet into the network, the victim can be deluged with fake traceback packets, preventing any information to be gained from the legitimate packets.

5.3. The IP Header

Our last source of bits is the IP header. There are several fields in the header that may be exploited for bits, with varying tradeoffs. As shown in Figure 2, we have found 25 bits that might possibly be used, although we think that a subset of these bits would better meet our goal of preserving backwards compatibility.

5.3.1. The TOS Field

The type of service field is an 8 bit field in the IP header that is currently used to allow hosts a way to give hints to routers as to what kind of route is important for particular packets (maximized throughput or minimized delay, for example) [1]. This field has been little used in the past, and, in some limited experiments, we have found that setting this field arbitrarily makes no measurable difference in packet delivery. There is a proposed Internet standard [15] that would change the TOS field to a "differentiated services field." Even the proposed DS field has two unused bits, however, there are already other proposed uses for these bits (e.g. [18]).

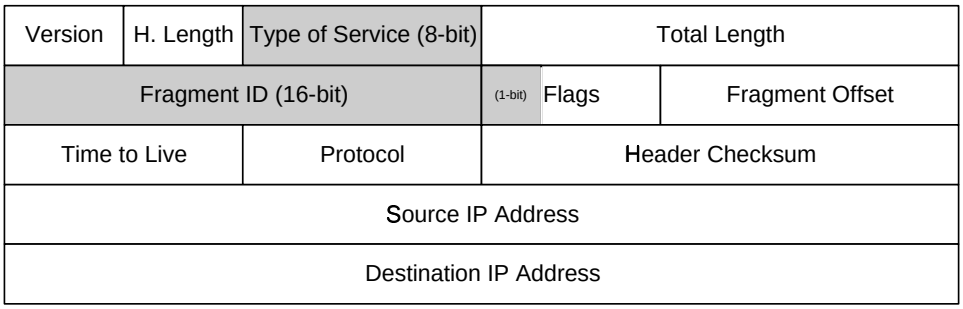

Figure 2. The IP Header. Darkened areas represent underutilized bits.

5.3.2. The ID Field

The ID field is a 16 bit field used by IP to permit reconstruction of fragments. Naive tampering with this field breaks fragment reassembly. Since less than 0.25% of all Internet traffic is fragments [22], we think that overloading this field is appropriate. A more in-depth discussion of the issues related to its overloading can be found in Savage's work [19].

5.3.3. The Unused Fragment Flag

There is an unused bit in the fragment flags field that current Internet standards require to be zero. We have found that setting this bit to one has no effect on current implementations, with the exception that when receiving the packet, some systems will think it is a fragment. The packet is still successfully delivered however, because it looks to those systems as though it is fragment 1 of 1.

Our Selection

As shown in Figure 3, we chose to use 15 bits out of the ID field. Since we needed more than 9 bits, we had to use at least part of the ID field and using only part of the ID field and part of another field would not have provided us with any benefits.

5.4. IPv6

Since IPv6 does not have nearly as many backwards compatibility issues as IPv4, the logical place to put traceback information is a hop-by-hop option in the IPv6 header [9]. However, schemes such as those presented here are still valuable because they use a fixed number of bits per packet thereby avoiding the generation of fragments.

6. Analysis

A major advantage of our approach is the amount of flexibility available in choosing a scheme. There is a rich space of algebraic alternatives to Savage's design. We have chosen a particular parameterization to implement for the purpose of analysis, but we note that our choice is certainly not the only practical alternative and under different assumptions and design criteria would not be the ideal choice.

We will use 15 bits out of the ID field of the IP header to store our data. As mentioned above, this choice breaks IP fragmentation, but due to the prevalence of MTU path discovery and the decline of fragmentation in general we feel this is an acceptable tradeoff. A proposed work-around to allow fragmentation by using additional packets has also been proposed [19].

As shown in Figure 3, 11 bits are used as an accumulator, 3 bits are used as random data, and one bit is used for signaling. This means that all arithmetic in the accumulator will be done in $GF(2039)$ (2039 is the largest prime $< 2^{11}$). The signaling bit will allow a router to tell the next router that it should add its values into the accumulator. That router will also reset the signaling bit. This corresponds to random partial path encoding with $\ell = 1$.

Each router must precompute three 11 bit chunks based on its 32 bit IP address, Z. Let $z_1 = Z \bmod 2027$, $z_2 = Z \bmod 2029$, and $z_3 = Z \bmod 2039$. Since 2027, 2029, and 2039 are all prime and $2027 \cdot 2029 \cdot 2039 > 2^{32}$, we will be able to reconstruct the value of Z by invoking the Chinese Remainder Theorem [2].

With a probability of $1/25$, a router will set the 3 random bits (let's call this value x_i), set the accumulator to $z_3x_i^2 + z_2x_i + z_1 \bmod 2039$, and set the signal bit. The rest of the time it will check to see if the signal bit is set. If so, it will incorporate its values (y_1, y_2, and y_3) using Horner's rule and clear the signal bit, thus completing the calculation of $z_3x_i^5 + z_2x_i^4 + z_1x_i^3 + y_3x_i^2 + y_2x_i + y_1 \bmod 2039$. This procedure is presented in pseudocode in Figure 4.

6.1. Packets Needed

The receiver sees the values of evaluated degree 5 polynomials; $z_3x_i^5 + z_2x_i^4 + z_1x_i^3 + y_3x_i^2 + y_2x_i + y_1$, for example. Our goal is to recover the IP addresses of the routers

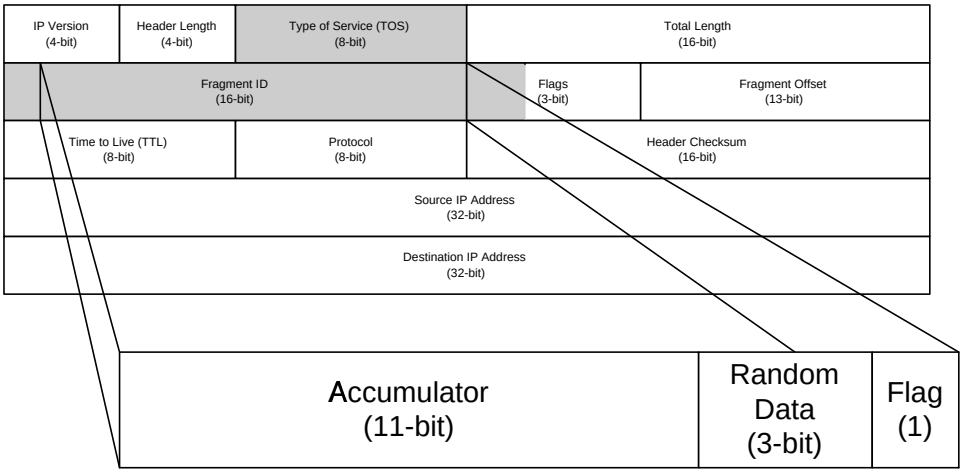

Figure 3. We choose to use 15 bits from the IP header

(Z and Y in our example) from this data. Using the method of [13], we will need to have $(1/25)(1-1/25)^{d-1} > \sqrt{5N}/N$ to recover edges distance d away from us if we analyze N packets at a time. Therefore, N must be greater than $5/((1/25)(1-1/25)^{d-1})^2$ for us to expect to get edges distance d away. Figure 5 shows the number of packets needed for different values of d.

In our simulations, we were able to recover paths of length 25 over 98% of the time by analyzing 20,000 packets at once, which agrees with our analytic result. In recent denial of service attacks, Yahoo reported receiving over 1 gigabyte of data per second[2]. Even if every packet was of the largest possible size, Yahoo would have received more than enough packets in under 2 seconds. We realize that most sites do not have the bandwidth of Yahoo, but we still think that most sites would be able to recover interesting paths in far less than a minute. We also note that our scheme will be able to take advantage of any new algorithm for decoding Reed-Solomon codes to improve these results without any router modifications. If more than one path is present in the data, the Guruswami-Sudan algorithm might not find all the paths from a single sample of 20,000 packets. Repeating the reconstruction on different samples might be needed. Trying to find all paths from a single sample would require an increase in the sample size that was quadratic in the number of paths.

6.2. Router Performance

At the baseline, this scheme is already rather efficient for routers, requiring only normal ALU operations, compares, and a random number generator. We can, however, use some precomputation to improve this situation considerably.

We only need to have our degree 5 polynomial evaluated at 6 points in order to recover it, so we will treat the random value 6 as 0 and 7 as 1. This should not cause us any trouble as long as all routers agree on the change, because the coupon collector's problem tells us that we would expect to get all 6 values in far fewer packets than are required by our multiple path reconstructor. Even the smallest routers should be able to precompute and store the 6 possible values that would need to be inserted when they are in marking mode (these values require only 12 bytes of storage). If we are storing these values already, we should also include an extra multiplication by the random value because that is the first thing that the next router would have to compute. At the victim we would, of course, have to divide by the random value for all packets that still have their signal bit set. This reduces the work needed at the second router to, at most, 2 random number generations, 2 compares, 2 shifts, 5 adds and a reduction modulo 2039. It is worth noting that this could easily be accomplished using combinational logic in an ASIC or custom chip. For larger routers it would probably make sense to precompute a lookup table with all possible second hop values.

We implemented this scheme under FreeBSD 4.0 on a Pentium II running at 333 MHz. Using RC4 [20] as the random number generator, the scheme executed in less than 50 clocks per packet. When routing packets across a 100 Mbit/sec Ethernet, there was no measurable difference in throughput between the modified and unmodified kernels (more than 95 Mbit/sec worth of packets were routed in both cases).

6.3. Reconstruction Performance

The reconstruction algorithm due to Guruswami and Sudan [13] can be implemented in a number of ways.

[2] `http://abcnews.go.com/sections/tech/DailyNews/yahoo000209.html`

At each router:
let Z = the router's IP address
let $z_1 = Z \bmod 2027$
let $z_2 = Z \bmod 2029$
let $z_3 = Z \bmod 2039$
foreach packet p
  let r be a random number from $[0..1)$
  if $r < (1/25)$ then
    let x be a random integer from $[0..7)$
    set $p.accumulator = (z_3x^2 + z_2x + z_1) \bmod 2039$
    set $p.flag = 1$
    set $p.random = x$
  else
    if $p.flag$ then
      set $p.accumulator = (p.accumulator \cdot p.x) + z_3$
      set $p.accumulator = (p.accumulator \cdot p.x) + z_2$
      set $p.accumulator = (p.accumulator \cdot p.x) + z_1$
      set $p.accumulator = p.accumulator \bmod 2039$
      set $p.flag = 0$

Figure 4. Marking algorithm executed by each router

The most straightforward implementation would take time $O(n^{15})$ to recover all edges for which we received at least $\sqrt{5n}$ out of n packets. However, this drops to $O(n^3)$ time by requiring only slightly more packets: $\sqrt{5n(1+\delta)}$ out of n, for any $\delta \geq 1$. By scaling δ appropriately, this allows us to trade off computation time (and memory) for accuracy. A recent algorithmic breakthrough by Olshevsky and Shokrollahi would reduce our reconstruction time even further, to $O(n^{2.5})$ [16]. Moreover, this new algorithm is highly parallelizable (to up to $O(n)$ processors), which suggests that distributing the reconstruction task might speed things up even more.

These reconstruction times compare quite favorably in the multiple attacker scenario to the $O(m^8)$ time required by Savage [19], where m is the number of routers at a given distance from the victim.

6.4. Resistance to Attack

While this metric is the most important in evaluating a traceback scheme it is also the most difficult to analyze. Our scheme seems to be resistant to all of the same attacks as the scheme proposed by Savage, *et al* and even with current algorithms for filtering mixed data can deal with multiple attackers more robustly. One major difference between these schemes is our decision not to include an explicit hop count which allows Savage's scheme to deny an attacker the ability to insert packets closer than his distance to the victim. We would note that this only prevents the insertion of edges closer than the *closest attacker*. An attack on this would be to have multiple attackers at different distances and use the close attackers to "hide" the routes of packets from those attackers farther away. Our scheme also suffers from this problem, but not as severely because of the built in noise filtering of the Guruswami-Sudan multiple path reconstruction algorithm. We think that simply by comparing the frequency at which an edge is marked to the expected marking probability of the edge, we can detect false edges closer than the attacker, so long as the nearest attacker is at least a few hops away. This technique has worked well in our simulations.

Our schemes could also make use of the HMAC techniques discussed by Song and Perrig to ensure that edges are not faked, but this would require us to either use additional space in the packets to store the hash or lose our incremental deployment properties [21]. If we decided to make one of these tradeoffs, our scheme should be comparably secure against multiple attackers.

7. Future Work

One important open problem is to find better variations of our Random Full Path tracing schemes. Perhaps an approach based on algebraic geometric codes [13] would be successful. We have been unable to find a variation that immediately improves on combinatorial approaches in all situations, but it seems intuitively plausible that one should exist. More generally, it would be interesting to more carefully explore resource and security tradeoffs for more of the many parameterizations of our methods.

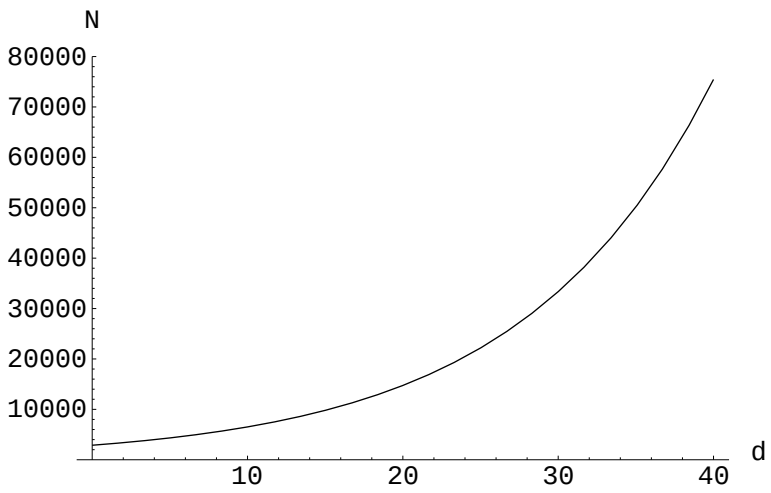

Figure 5. Number of packets needed to recover different length paths

8. Conclusions

We have presented a new algebraic approach for providing traceback information in IP packets. Our approach is based on mathematical techniques that were first developed for problems related to error correcting codes and machine learning. Our best scheme has improved robustness over previous combinatorial approaches, both for noise elimination and multiple-path reconstruction. Another key advantage of our schemes is that they will automatically benefit from any improvement in the underlying mathematical techniques, for which progress has been steady in recent years.

Acknowledgments

We would like to thank David Goldberg and Dan Boneh for helpful discussions. We would also like to thank Dawn Song, Adrian Perrig, and the anonymous referees for helpful comments on an earlier version of this paper.

References

[1] P. Almquist. Type of service in the internet protocol suite. RFC 1349, July 1992.

[2] J. A. Anderson and J. M. Bell. *Number Theory with Applications*. Prentice Hall, 1996.

[3] S. Ar, R. J. Lipton, R. Rubinfeld, and M. Sudan. Reconstructing algebraic functions from mixed data. In *33rd Annual Symposium on Foundations of Computer Science*, pages 503–512, Pittsburgh, Pennsylvania, 24–27 Oct. 1992. IEEE.

[4] S. M. Bellovin. Personal Communications, May 2000.

[5] S. M. Bellovin. ICMP traceback messages. `http://search.ietf.org/internet-drafts/draft-bellovin-itrace-00.txt`, Mar. 2000.

[6] E. Berlekamp and L. Welch. Error correction of algebraic block codes. United States Patent 4,490,811, Dec. 86.

[7] E. R. Berlekamp. *Algebraic Coding Theory*. Aegean Park Press, 1984.

[8] CERT coordination center denial of service attacks. `http://www.cert.org/tech_tips/denial_of_service.html`, Feb. 1999.

[9] S. Deering and R. Hinden. Internet protocol, version 6 (IPv6) specification. RFC 2474, Dec. 1995.

[10] D. Dittrich. The "stacheldraht" distributed denial of service attack tool. `http://staff.washington.edu/dittrich/misc/stacheldraht.analysis.txt`, Dec. 1999.

[11] D. Dittrich. The "Tribe Flood Network" distributed denial of service attack tool. `http://staff.washington.edu/dittrich/misc/tfn.analysis`, Oct. 1999.

[12] P. Ferguson and D. Senie. Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing. RFC 2267, Jan. 1998.

[13] V. Guruswami and M. Sudan. Improved decoding of Reed-Solomon and algebraic-geometric codes. *IEEE Transactions on Information Theory*, 45:1757–1767, 1999.

[14] D. E. Knuth. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*. Addison-Wesley, 1998.

[15] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the Differentiated Services field (DS field) in the IPv4 and IPv6 headers. RFC 2474, Dec. 1998.

[16] V. Olshevsky and M. A. Shokrollahi. A displacement approach to efficient decoding of algebraic-geometric codes. In *31st Annual ACM Symposium on Theory of Computation*, pages 235–244, Atlanta, Georgia, May 1999. ACM.

[17] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in FORTRAN: The Art of Scientific Computing*. Cambridge University Press, 1992.

[18] K. Ramakrishnan and S. Floyd. A proposal to add Explicit Congestion Notification (ECN) to IP. RFC 2481, Jan. 1999.

[19] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical network support for IP traceback. In *2000 ACM SIGCOMM Conference*, Aug. 2000.

[20] B. Schneier. *Applied Cryptography, Second Edition*. John Wiley and Sons, 1996.

[21] D. Song and A. Perrig. Advanced and authenticated marking schemes for IP traceback. Technical Report UCB/CSD-00-1107, University of California, Berkeley, June 2000.

[22] I. Stoica and H. Zhang. Providing guaranteed services without per flow management. In *ACM SIGCOMM '99*, pages 81–94, Cambridge, MA, 1999.

[23] M. Sudan. Algorithmic issues in coding theory. In *17th Conference on Foundations of Software Technology and Theoretical Computer Science*, Kharagpur, India, 1997.

[24] M. Sudan. Decoding of Reed Solomon codes beyond the error-correction bound. *Journal of Complexity*, 13(1):180–193, Mar. 1997.