Practical Network Support for IP Traceback

Stefan Savage, David Wetherall, Anna Karlin and Tom Anderson
Department of Computer Science and Engineering
University of Washington, Seattle

Technical Report UW-CSE-00-02-01

Abstract

This paper describes a technique for tracing anonymous attacks in the Internet back to their source. This work is motivated by the increased frequency and sophistication of denial-of-service attacks and by the difficulty in tracing packets with incorrect, or "spoofed", source addresses. In this paper we describe a general purpose traceback mechanism based on probabilistic packet marking in the network. Our approach allows a victim to identify the network path(s) traversed by an attacker without requiring interactive operational support from Internet Service Providers (ISPs). Moreover, this traceback can be performed "post-mortem" – after an attack has completed. We present an implementation of this technology that is incrementally deployable, (mostly) backwards compatible and can be efficiently implemented using conventional technology.

This paper represents work-in-progress and should be reviewed as such. We believe there are further improvements to our algorithms, alternative approaches for backwards-compatible data encoding, and other disciplines for use. However, given the attention focused by current events, we believe now is an ideal time to introduce our ideas and start a discussion about their strengths and weaknesses, and the potential for deployment in future generations of network equipment. Please send any feedback or questions to savage@cs.washington.edu.

Practical Network Support for IP Traceback

Stefan Savage, David Wetherall, Anna Karlin and Tom Anderson
Department of Computer Science and Engineering
University of Washington, Seattle

Abstract

This paper describes a technique for tracing anonymous attacks in the Internet back to their source. This work is motivated by the increased frequency and sophistication of denial-of-service attacks and by the difficulty in tracing packets with incorrect, or "spoofed", source addresses. In this paper we describe a general purpose traceback mechanism based on probabilistic packet marking in the network. Our approach allows a victim to identify the network path(s) traversed by an attacker without requiring interactive operational support from Internet Service Providers (ISPs). Moreover, this traceback can be performed "post-mortem" – after an attack has completed. We present an implementation of this technology that is incrementally deployable, (mostly) backwards compatible and can be efficiently implemented using conventional technology.

1 Introduction

Denial-of-service attacks consume the resources of a remote host or network, thereby denying or degrading service to legitimate users. Such attacks are among the hardest security problems to address because they are simple to implement, difficult to prevent, and very difficult to trace. In the last several years, Internet denial-of-service attacks have increased in frequency, severity and sophistication. Howard reports that between the years of 1989 and 1995, the number of such attacks reported to the Computer Emergency Response Team (CERT) increased by 50 percent per year [How98]. More recently, a 1999 CSI/FBI survey reports that 32 percent of respondents detected denial-of-service attacks directed against them [CF99]. Even more worrying, recent reports indicate that attackers have developed tools to coordinate distributed attacks from many separate sites [CERT00].

Unfortunately, mechanisms for dealing with denial-of-service have not advanced at the same pace. Most work in this area has focused on *tolerating* attacks by mitigating their effects on the victim [SP99, BDM99, KS99, Mea99, Cis97]. This approach can provide an effective stop-gap measure, but does not eliminate the problem nor does it discourage attackers. The other option, and the focus of this paper, is to trace attacks back to their origin so they can be eliminated at the source.

Determining the source of an attack, which we call the *traceback problem*, is surprisingly difficult due to the stateless nature of Internet routing. Attackers routinely disguise their location using incorrect, or "spoofed", IP source addresses. As these packets traverse the Internet their true origin is lost and a victim is left with little useful information. While there are several ad hoc traceback techniques in use, they all have significant drawbacks that limit their practical utility in the current Internet.

In this paper we present a new approach to the traceback problem that addresses the needs of both victims and network operators. Our solution is to probabilistically mark packets with partial path information as they arrive at routers. This approach exploits the observation that attacks generally comprise large numbers of packets. While each marked packet represents only a "sample" of the path it has traversed, a victim can reconstruct the complete path after receiving a modest number of such packets. This approach allows victims to identify the approximate source of an attack without requiring the assistance of outside network operators. Moreover, this determination can be made even after an attack has completed.

A key practical deployment issue with any modification of Internet routers is to ensure that the mechanisms are efficiently implementable, may be incrementally deployed, and are backwards compatible with the existing infrastructure. We describe a traceback algorithm that adds little or no overhead to the router's critical forwarding path and may be incrementally deployed to allow traceback within the subset of routers supporting our scheme. Further, we demonstrate that we can encode the necessary path information in a way the peacefully co-exists with existing routers, host systems and more than 99% of today's traffic.

The rest of this paper is organized as follows: In Section 2, we describe related work concerning IP spoofing and solutions to the traceback problem. Section 3 outlines our basic approach and section 4 characterizes several abstract algorithms for implementing it. In Section 5 we detail a concrete encoding strategy for our algorithm that can be implemented within the current Internet environment. We also present experimental results demonstrating the effectiveness of our solution. In section 6 we discuss limitations of our proposal and extensions to address some of them. Finally, we summarize our findings in Section 7.

2 Related work

It has been long understood that the IP protocol permits anonymous attacks. In his 1985 paper on TCP/IP weaknesses, Morris writes:

> "The weakness in this scheme [the Internet Protocol] is that the source host itself fills in the IP source host id, and there is no provision in ... TCP/IP to discover the true origin of a packet." [Mor85]

In addition to denial-of-service attacks, IP spoofing can be used in conjunction with other vulnerabilities to implement anonymous one-way TCP channels and covert port scanning [Mor85, Bel89, HB96, VCIV99].

There have been several efforts to reduce the anonymity afforded by IP spoofing. Table 1 provides a subjective characterization of each of these approaches in terms of management cost,

	Management overhead	Network overhead	Router overhead	Distributed capability	Post-mortem capability
Ingress filtering	Moderate	Low	Moderate	N/A	N/A
Link testing					
Input debugging	High	Low	High	Good	Poor
Controlled flooding	Low	High	Low	Poor	Poor
Logging	High	Low	High	Excellent	Excellent
Marking	Low	Low	Low	Excellent	Excellent

Table 1: Qualitative comparison of existing schemes for combating anonymous attacks and the probabilistic marking approach we propose.

additional network load, overhead on the router, the ability to trace multiple simultaneous attacks, and the ability trace attacks after they have completed. We also characterize our proposed traceback scheme according to the same criteria. In the remainder of this section we describe each previous approach in more detail.

2.1 Ingress filtering

Obviously, the best way to address the problem of anonymous attacks is to eliminate the ability to forge source addresses. One such approach, frequently called *ingress filtering*, is to configure routers to block packets that arrive with illegitimate source addresses [FS98]. This requires a router with sufficient power to examine the source address of every packet and sufficient knowledge to distinguish between legitimate and illegitimate addresses. Consequently, ingress filtering is most feasible in customer networks or at the border of Internet Service Providers (ISP) where address ownership is relatively unambiguous and traffic load is low. As traffic is aggregated from multiple ISPs into transit networks, there is no longer enough information to unambiguously determine if a packet arriving on a particular interface has a “legal” source address. Moreover, on such high speed links the overhead of comparing every packet to a filter list becomes prohibitive.

The principal problem with ingress filtering is that its effectiveness depends on widespread, if not universal, deployment. Unfortunately, a significant fraction of ISPs, perhaps the majority, do not implement this service – either because they are uninformed or have been discouraged by the administrative burden[1], potential router overhead and complications with services like Mobile IP [Per96]. A secondary problem is that even if ingress filtering were universally deployed at the customer-to-ISP level, attackers could still forge addresses from the hundreds or thousands of hosts within a valid customer network [CERT00]. Unless these problems are resolved, there is still a significant need for traceback technologies.

2.2 Link testing

Most existing traceback techniques start from the router closest to the victim and interactively test its upstream links until they determine which one is used to carry the attacker's traffic. Ideally, this procedure is repeated recursively on the upstream router until the source is reached. This technique assumes that an attack remains active until the completion of a trace and is therefore inappropriate for attacks that are detected after the fact, attacks that occur intermittently, or attacks that modulate their behavior in response to a traceback. Below we describe two varieties of link testing schemes, *input debugging* and *controlled flooding*.

[1]Some modern routers ease the administrative burden of ingress filtering by providing functionality to automatically check source addresses against the destination-based routing tables (e.g. `ip verify unicast reverse-path` on Cisco's IOS). This approach is only valid if the route to and from the customer is symmetric – generally at the border of single-homed stub networks.

2.2.1 Input debugging

Many routers include a feature called *input debugging*, that allows an operator to filter particular packets on some egress port and determine which ingress port they arrived on. This capability is used to implement a trace as follows: First, the victim must recognize that it is being attacked and develop an *attack signature* that describes a common feature contained in all the attack packets. The victim communicates this signature to a network operator, frequently via telephone, who then installs a corresponding input debugging filter on the victim's upstream egress port. This filter reveals the associated input port, and hence which upstream router originated the traffic. The process is then repeated recursively on the upstream router, until the originating site is reached or the trace leaves the ISP's border (and hence its administrative control over the routers). In the later case, the upstream ISP must be contacted and the procedure repeats itself. While such tracing is frequently performed manually, several ISPs have developed tools to automatically trace attacks across their own networks [Sto99].

The most obvious problem with the input debugging approach, even with automated tools, is its considerable management overhead. Communicating and coordinating with network operators at multiple ISPs requires the time, attention and commitment of both the victim and the remote personnel – many of whom have no direct economic incentive to provide aid. If the appropriate network operators are not available, if they are unwilling to assist, or if they do not have the appropriate technical skills and capabilities, then a traceback may be slow or impossible to complete [Gla98].

2.2.2 Controlled flooding

Burch and Cheswick have developed a link testing traceback technique that does not require any support from network operators [BC99]. We call this technique *controlled flooding* because it tests links by flooding them with large bursts of traffic and observing how this perturbs traffic from the attacker. Using a pregenerated “map” of Internet topology, the victim coerces selected hosts along the upstream route into iteratively flooding each incoming link on the router closest to the victim. Since router buffers are shared, packets traveling across the loaded link – including any sent by the attacker – have an increased probability of being dropped. By observing changes in the rate of packets received from the attacker, the victim can therefore infer which link they arrived from. As with other link testing schemes, the basic procedure is then applied recursively on the next upstream router until the source is reached.

While the scheme is both ingenuous and pragmatic, in has several drawbacks and limitations. Most problematic among these is that controlled flooding is itself a denial-of-service attack – exploiting vulnerabilities in unsuspecting hosts to achieve its ends. This drawback alone makes it unsuitable for routine use. Also, controlled flooding requires the victim to have a good topological map of large sections of the Internet in addition to an associated list

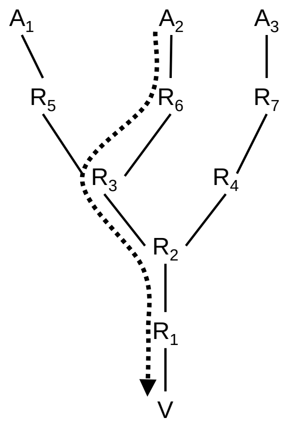

Figure 1: Network as seen from the victim of an attack, V. Routers are represented by R_i, and potential attackers by A_i. The dotted line represents a particular *attack path* between an attacker and the victim.

of "willing" flooding hosts. As Burch and Cheswick note, controlled flooding is also poorly suited for tracing distributed denial-of-service attacks because the link-testing mechanism is inherently noisy and it can be difficult to discern the set of paths being exploited when multiple upstream links are contributing to the attack. Finally, like all link-testing schemes, controlled flooding is only effective at tracing an on-going attack and cannot be used "post-mortem".

2.3 Logging

An approach suggested in [Sag98] and [Sto99] is to log packets at key routers and then use data mining techniques to determine the path that the packets traversed. This scheme has the useful property that it can trace an attack long after the attack has completed. However, it also has obvious drawbacks, including enormous resource requirements and a large scale inter-provider database integration problem. We are unaware of any commercial organizations that have developed a fully operational traceback approach based on logging.

3 Overview

Burch and Cheswick mention the possibility of tracing attacks by "marking" packets, either probabilistically or deterministically, with the addresses of the routers they traverse [BC99]. The victim uses the information in the marked packets to trace an attack back to its source. This approach has not been previously explored in any depth, but has many potential advantages. It does not require interactive cooperation with ISPs and therefore avoids the high management overhead of input debugging. Unlike controlled flooding, it does not require significant additional network traffic and can potentially be used to track multiple attacks. Moreover, like logging, packet marking can be used to trace attacks "post-mortem" – long after the attack has stopped. Finally, we have found that marking algorithms can be implemented without incurring any significant overhead on network routers. The remainder of this paper focuses on fully exploring and characterizing this approach.

3.1 Definitions

Figure 1 depicts the network as seen from a victim V. For the purposes of this paper, V may be a single host under attack, or a network border device such as a firewall or intrusion detection system that represents many such hosts. Every potential *attack origin* A_i is a leaf in a tree rooted at V and every router R_i is an internal node along a path between some A_i and V. The *attack path* from A_i is the unique ordered list of routers between A_i and V. For instance, if an attack originates from A_2 then to reach V it must first traverse the path R_6, R_3, R_2, and R_1 – as shown by the dotted line in Figure 1.

The *exact traceback* problem is to determine the attack path and the associated attack origin for each attacker. However, solving this problem is complicated by several practical limitations. The exact attack origin may never be revealed (even MAC source addresses may be spoofed) and a wily attacker may send false signals to "invent" additional routers in the traceback path. We address these issues in section 6, but for now we restrict our discussion to solving a more limited problem. We define the *approximate traceback* problem as finding a candidate attack path for each attacker that contains the true attack path as a suffix. We call this the *valid suffix* of the candidate path. For example, (R_5, R_6, R_3, R_2, R_1) is a valid approximate solution to Figure 1 because it contains the true attack path as a suffix. We say a solution to this problem is *robust* if an attacker cannot prevent the victim from discovering candidate paths containing the valid suffix.

All marking algorithms have two components: a *marking procedure* executed by routers in the network and a *path reconstruction procedure* implemented by the victim. A router "marks" one or more packets by augmenting them with additional information about the path they are traveling. The victim attempts to reconstruct the attack path using only the information in these marked packets. The *convergence time* of an algorithm is the number of packets that the victim must observe to reconstruct the attack path.

3.2 Basic assumptions

The design space of possible marking algorithms is large, and to place our work in context we identify the assumptions that motivate and constrain our design:

- an attacker may generate any packet,
- multiple attackers may conspire,
- attackers may be aware they are being traced,
- packets may be lost or reordered,
- attackers send numerous packets,
- the route between attacker and victim is fairly stable,
- routers are both CPU and memory limited, and
- routers are not widely compromised.

The first four assumptions represent conservative assessments of the abilities of the modern attackers and limitations of the network. Designing a traceback system for the Internet environment is extremely challenging because there is very little that can be trusted. In particular, the attacker's ability to create arbitrary packets significantly constrains potential solutions. When a router receives a packet, it has no way to tell whether that packet has been marked by an upstream router or if the attacker simply has forged this information. In fact, the only invariant that we can depend on is that a packet from the attacker must traverse all of the routers between it and the victim.

The remaining assumptions reflect the basis for our design and deserve additional discussion. First, denial-of-service attacks are only effective so long as they occupy the resources of the victim. Consequently, most attacks are comprised of thousands or millions of packets. Our approach relies on this property because we mark each packet with only a small piece of path state and the victim must observe many such packets to reconstruct the complete path back the the attacker. If many attacks emerge that require only a single packet to disable a host (e.g. ping-of-death [CERT96]), then this assumption may not hold (although we note that even these attacks require multiple packets to *keep* a machine down).

Second, measurement evidence suggests that while Internet routes do change, it is extremely rare for packets to follow many different paths over the short time-scales of a traceback operation (seconds in our system) [Pax97]. This assumption greatly simplifies the role of the victim, since it can therefore limit its consideration to a single primary path for each attacker. If the Internet evolves to allow significant degrees of multi-path routing then this assumption may not hold.

Third, while there have been considerable improvements in router implementation technology, link speeds have also increased dramatically. Consequently, we assert that any viable implementation must have low per-packet overhead and must not require per-flow state. Significantly simpler schemes than ours can be implemented if we assume that routers are not resource constrained.

Finally, since a compromised router can effectively eliminate any information provided by upstream routers, it is effectively indistinguishable from an attacker. In such circumstances, the security violation at the router must be addressed first, before any further traceback is attempted. In normal circumstances, we believe this is an acceptable design point. However, if non-malicious, but information hiding, routing infrastructures become popular, such as described in [GS99, RSG98], then this issue may need to be revisited.

4 Basic marking algorithms

In this section we describe a series of marking algorithms – starting from the most simple and advancing in complexity. Each algorithm attempts to solve the approximate traceback problem in a manner consistent with our assumptions.

4.1 Node append

The simplest marking algorithm – conceptually similar to the IP Record Route option [Pos81] – is to append each node's address to the end of the packet as it travels through the network from attacker to victim (see Figure 2). Consequently, every packet received by the victim arrives with a complete ordered list of the routers it traversed – a built-in attack path.

The node append algorithm is both robust and extremely quick to converge (a single packet), however it has several serious limitations. Principal among these is the infeasibly high router overhead incurred by appending data to packets in flight. Moreover, since the length of the path is not known *a priori*, it is impossible to ensure that there is sufficient unused space in the packet for the complete list. This can lead to unnecessary fragmentation and bad interactions with services such as MTU discovery [MD90]. This problem cannot be solved by reserving "enough" space, as the attacker can completely fill any such space with false, or misleading, path information.

```
Marking procedure at router $R$:
    for each packet $w$, append $R$ to $w$

Path reconstruction procedure at victim $v$:
    for any packet $w$ from attacker
        extract path $(R_i..R_j)$ from the suffix of $w$
```

Figure 2: Node append algorithm.

```
Marking procedure at router $R$:
    for each packet $w$
        let $x$ be a random number from [0..1)
        if $x < p$ then,
            write $R$ into $w$.node

Path reconstruction procedure at victim $v$:
    let $NodeTbl$ be a table of tuples (node,count)
    for each packet $w$ from attacker
        $z$ := lookup $w$.node in $NodeTbl$
        if $z$ != NIL then
            increment $z$.count
        else
            insert tuple ($w$.node,1) in $NodeTbl$
    sort $NodeTbl$ by count
    extract path $(R_i..R_j)$ from ordered node fields in $NodeTbl$
```

Figure 3: Node sampling algorithm.

4.2 Node sampling

To reduce both the router overhead and the per-packet space requirement, we can sample the path one node at a time instead of recording the entire path. A single static "node" field is reserved in the packet header – large enough to hold a single router address (i.e. 32 bits for IPv4). Upon receiving a packet, each router chooses to write its address in the node field with some probability p. After enough packets have been sent, the victim will have received at least one sample for every router in the attack path. As stated in section 3, we assume that the attacker sends enough packets and the route is stable enough that this sampling can converge.

Although it might seem impossible to reconstruct an ordered path given only an unordered collection of node samples, it turns out that with a sufficient number of trials, the order can be deduced from the relative number of samples per node. Since routers are arranged serially, the probability that a packet will be marked by a router and then left unmolested by all downstream routers is a strictly decreasing function of the distance to the victim. If we constrain p to be identical at each router, then the probability of receiving a marked packet from a router d hops away is $p(1-p)^{d-1}$. Since this function is monotonic in the distance from the victim, ranking each router by the number of samples it contributes will tend to produce the accurate attack path. The full algorithm is shown in Figure 3.

Putting aside for the moment the difficulty in changing the IP header to add a 32-bit node field, this algorithm is efficient to implement because it only requires the addition of a write and checksum update to the forwarding path. Current high-speed routers already must perform these operations efficiently to update the *time-to-live* field on each hop. Moreover, if $p > 0.5$ then this algorithm is robust against a single attacker because there is no way for an attacker to insert a “false” router into the path's valid suffix by contributing more samples than a downstream router, nor to reorder valid routers in the path by contributing more samples than the difference between any two downstream routers.

However, there are also two serious limitations. First, inferring the total router order from the distribution of samples is a slow process. Routers far away from the victim contribute relatively few samples (especially since p must be large) and random variability can easily lead to misordering unless a very large number of samples are observed. For instance, if $d = 15$ and $p = 0.51$, the receiver must receive more than 42,000 packets on average before it receives a *single* sample from the furthest router. To guarantee that the order is correct with 95% certainty requires more than seven times that number.

Second, if there are multiple attackers then multiple routers may exist at the same distance – and hence be sampled with the sample probability. Therefore, this technique is not robust against multiple attackers.

4.3 Edge sampling

A straightforward solution to these problems is to explicitly encode *edges* in the attack path rather than simply individual nodes. To do this, we would need to reserve *two* static address-sized fields, *start* and *end*, in each packet to represent the routers at each end of a link, as well as an additional small field to represent the distance of an edge sample from the victim.

When a router decides to mark a packet, it writes its own address into the start field and writes a zero into the distance field. Otherwise, if the distance field is already zero this indicates that the packet was marked by the previous router. In this case, the router writes its own address into the end field – thereby representing the edge between itself and the previous router. Finally, if the router doesn't mark the packet then it always increments the distance field. This somewhat baroque signaling mechanism allows edge sampling to be incrementally deployed – edges are constructed only between participating routers.

The mandatory increment is necessary to avoid spoofing by an attacker. When the packet arrives at the victim its distance field represents the number of hops traversed since the edge it contains was sampled.[2] Any packets written by the attacker will necessarily have a distance greater or equal to the length of the true attack path. Consequently, since we no longer use the sampling rank approach to distinguish “false” samples, we are free to use arbitrary values for the marking probability p.

The victim uses the edges sampled in these packets to create a graph (much as in Figure 1) leading back to the source, or sources, of attack. The full algorithm is described in Figure 4. Because the probability of receiving a sample is geometrically smaller the further away it is from the victim, the time for this algorithm to converge is dominated by the the time to receive a sample from the furthest router, $\frac{1}{p(1-p)^{d-1}}$ in expectation, for a router d hops away. However, there is a small probability that we will receive a sample from the furthest router, but not from some nearer router. We can bound this effect to a factor of $ln(d)$ by the following argument: We conservatively assume that samples from all of the d routers appear with the same likelihood as the furthest router. Since these probabilities are disjoint, the probability that a given packet will deliver a sample from some router is at least $dp(1-p)^{d-1}$. Finally, as per the well-known *coupon collector* problem, the number of trials required to select one of each of d equi-probable items is $d(ln(d) + O(1))$ [Fel66]. Therefore, the number of packets, X, required for the victim to reconstruct a path of length d has the following bounded expectation:

$$E(X) < \frac{ln(d)}{p(1-p)^{d-1}}$$

For example, if $p = \frac{1}{10}$, and the attack path has a length of 10, then a victim can typically reconstruct this path after receiving 75 packets from the attacker. While this choice of $p = \frac{1}{d}$ is optimal, the convergence time is not overly sensitive to this parameter for the path lengths that occur in the Internet. So long as $p \leq \frac{1}{d}$, the results are generally within a small constant of optimal. In the rest of this paper we will use $p = \frac{1}{25}$ since few paths exceed this length [CC97, TR00, CAIDA00]. For comparison, the previous example converges with only 108 packets using $p = \frac{1}{25}$.

This same algorithm can efficiently discern multiple attacks because attackers from different sources produce disjoint edges in the tree structure used during reconstruction. The number of packets needed to reconstruct each path is independent, so the number of packets needed to reconstruct all paths is a linear function of the number of attackers. Finally, edge sampling is also robust (it is impossible for any edge closer than the closest attacker to be spoofed, due to the robust distance determination).

Of course, a significant practical limitation of this approach is that it requires additional space in the IP packet header and therefore is not backwards compatible. In the next section we discuss a modified version of edge-sampling that addresses this problem, albeit at some cost in performance and robustness.

```
Marking procedure at router $R$:
   for each packet $w$
      let $x$ be a random number from [0..1)
      if $x < p$ then
         write $R$ into $w$.start and 0 into $w$.distance
      else
         if $w$.distance = 0 then
            write $R$ into $w$.end
         increment $w$.distance

Path reconstruction procedure at victim $v$:
   let $G$ be a tree with root $v$
   let edges in $G$ be tuples (start,end,distance)
   for each packet $w$ from attacker
      if $w$.distance = 0 then
         insert edge ($w$.start,$v$,0) into $G$
      else
         insert edge ($w$.start,$w$.end,$w$.distance) into $G$
   remove any edge ($x$,$y$,$d$) with $d \neq$ distance from $x$ to $v$ in $G$
   extract path ($R_i..R_j$) by enumerating acyclic paths in $G$
```

Figure 4: Edge sampling algorithm.

5 Encoding issues

The edge sampling algorithm requires 72 bits of space in every IP packet (two 32-bit IP addresses and 8 bits for distance to represent the theoretical maximum number of hops allowed using IP). It would be possible to directly encode these values into an MPLS label stack [RRT$^+$98], to enable traceback within a single homogeneous ISP network. However, our focus is on a heterogeneous environment based purely on IP datagrams. One universal approach

[2]It is important that distance field is updated using a saturating addition. If the distance field were allowed to wrap, then the attacker could spoof edges close to the victim by sending packets with a distance value close to the maximum.

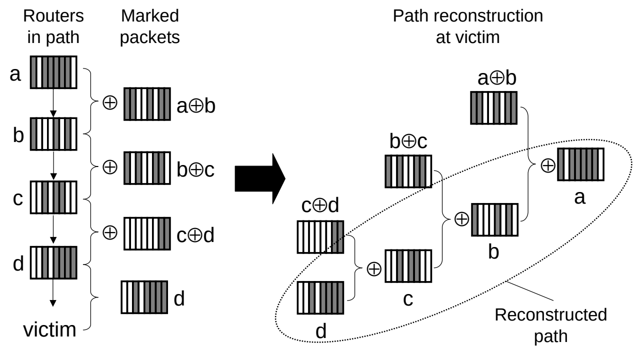

Figure 5: Edge data can be communicated in half the space by sending the XOR of the two nodes (i.e. router IP addresses) making up an edge, rather than sending each node separately. Over time the victim receives the messages d, $c \oplus d$, $b \oplus c$, and $a \oplus b$. By XORing these messages together the original path can be reconstructed.

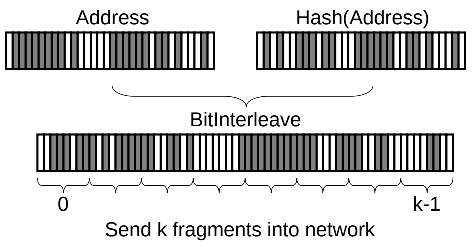

Figure 6: Each router calculates a uniform hash of its IP address once, at startup, using a well-known function. This hash is interleaved with the original IP address (the original address on odd bits, the hash on even bits). The resulting quantity is then broken into k fragments, which the router selects among randomly when marking a packet. Although it is not shown, each of these fragments is further labeled with its offset. The next downstream router users this offset to select the appropriate fragment to XOR – thereby encoding part of an edge.

is to store the edge sample data in an IP option, but this is a poor choice for the same reasons that the node append algorithm is infeasible – appending additional data to a packet in flight is expensive and may lead to fragmentation. We could also send this data out-of-band – in a separate packet – but this would add both router and network overhead plus the complexity of a new and incompatible protocol.

Instead, we have developed a modified version of edge sampling that dramatically reduces the space requirement in return for a modest increase in convergence time. Following an analysis of our algorithm we explore the practical implementation issues and discuss one concrete encoding of this scheme based on overloading the 16-bit IP *identification* field used for fragmentation. We stress that our solution reflects only one design point among many potential implementation tradeoffs for this class of algorithm and *does not* necessarily reflect an optimal balance among them.

5.1 Compressed edge fragment sampling

We use three techniques to reduce per-packet storage requirements while preserving robustness. First, we encode each edge in half the space by representing it as the *exclusive-or* (XOR) of the two

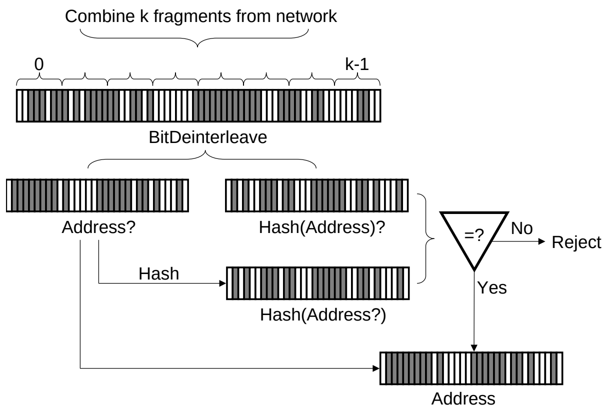

Figure 7: When reconstructing a candidate edge, the victim combines k fragments to produce a bit string. By de-interleaving this string, the address portion and the hash portion are extracted. We recalculate the hash over this address portion using the same hash function used by the router. If the resulting hash is the same as the hash portion extracted, then the address is accepted as valid. This procedure protects against accidentally combining fragments of different edges.

IP addresses making up the edge, as depicted in Figure 5. When some router decides to mark a packet it writes its address, a, into the packet. The following router, b, notices that the distance field is 0 and (assuming it does not mark the packet itself) reads a from the packet, XORs this value with its own address and writes the resulting value, $a \oplus b$, into the packet. We call the resulting value the *edge-id* for the edge between a and b. The edge-ids in the packets received by the victim always contain the XOR of two adjacent routers, except for samples from routers one hop away from the victim, which arrive unmodified. Since $b \oplus a \oplus b = a$, marked packets from the final router can be used to decode the previous edge id, and so on, hop-by-hop until we reach the first router.

Our second modification further reduces our per-packet space requirements by subdividing each edge-id into k smaller non-overlapping fragments. When a router decides to mark a packet, it selects one of these fragments at random and stores it in the packet. We use a few additional bits ($log_2 k$) to store the offset of this fragment within the original address – this is necessary to ensure that both fragments making up an edge-id are taken from the same offset. If enough packets are sent by the attacker, the victim will eventually receive all fragments from all edge-ids.

Finally, unlike full IP addresses, edge-id fragments are not unique and multiple fragments from different edge-ids may have the same value. If there are multiple attackers, a victim may receive multiple edge fragments with the same offset and distance. To reduce the probability that we accidentally reconstruct a “false” edge-id by combining fragments from different paths, we add a simple error detection code to our algorithm. We *increase* the size of each router address, and hence each edge-id, by bit-interleaving its IP address with a random hash of itself (depicted in Figure 6). As described earlier, this value is split into fragments, each fragment is selected randomly and stored with an offset, and downstream routers use XOR to combine fragments at the same offset to make up edge-id fragments. The victim constructs *candidate edge-ids* by combining all combinations of fragments at each distance with disjoint offset values. As shown in Figure 7, a candidate edge-id is only accepted if the hash portion matches the data portion for each of its two nodes. By making the hash sufficiently large the prob-

```
*Marking procedure at router $R$:*
    let $R'$ = BitIntereave($R$, Hash($R$))
    let $k$ be the number of non-overlapping fragments in $R'$
    for each packet $w$
        let $x$ be a random number from [0..1)
        if $x < p$ then
            let $o$ be a random integer from [0..$k-1$]
            let $f$ be the fragment of $R'$ at offset $o$
            write $f$ into $w$.frag
            write 0 into $w$.distance
            write $o$ into $w$.offset
        else
            if $w$.distance = 0 then
                let $f$ be the fragment of $R'$ at offset $w$.offset
                write $f \oplus w$.frag into $w$.frag
            increment $w$.distance

*Path reconstruction procedure at victim $v$:*
    let $FragTbl$ be a table of tuples (frag,offset,distance)
    let $G$ be a tree with root $v$
    let edges in $G$ be tuples (start,end,distance)
    let $maxd$ := 0
    let $last$ := $v$
    for each packet $w$ from attacker
        $FragTbl$.Insert($w$.frag,$w$.offset,$w$.distance)
        if $w$.distance $>$ $maxd$ then
            $maxd$ := $w$.distance
    for $d$ := 0 to $maxd$
        for all ordered combinations of fragments at distance $d$
            construct edge $z$
            if $d \neq 0$ then
                $z$ := $z \oplus last$
            if Hash(EvenBits($z$)) = OddBits($z$) then
                insert edge ($z$,EvenBits($z$),$d$) into $G$
                $last$ := EvenBits($z$);
    remove any edge ($x$,$y$,$d$) with $d \neq$ distance from $x$ to $v$ in $G$
    extract path ($R_i$..$R_j$) by enumerating acyclic paths in $G$
```

Figure 8: Compressed edge fragment sampling algorithm.

ability of a collision can be made extremely small. We provide a describe the full procedure in Figure 8.

The expected number of packets for this algorithm to converge is similar to the edge sampling approach, except now we need k fragments for each edge-id, rather than just one, a total of kd fragments. If we again assume conservatively that each of these fragments is delivered equi-probably with probability $p(1-p)^{d-1}$, the expected number of packets required for path reconstruction is bounded by:

$$E(X) < \frac{k \cdot ln(kd)}{p(1-p)^{d-1}}$$

For example, if there are 8 fragments per edge-id, an attacker is 10 hops away, and $p = \frac{1}{25}$, then a victim can reconstruct the full path after receiving slightly less than 1,300 packets on average. For a stronger guarantee we can conservatively approximate the number of packets required to ensure that a path can be reconstructed with probability $1 - \frac{1}{c}$ as:

$$\frac{k \cdot ln(kdc)}{p(1-p)^{d-1}}$$

packets. To completely reconstruct the previous path with 95% certainty should require no more than 2150 packets. Many denial-

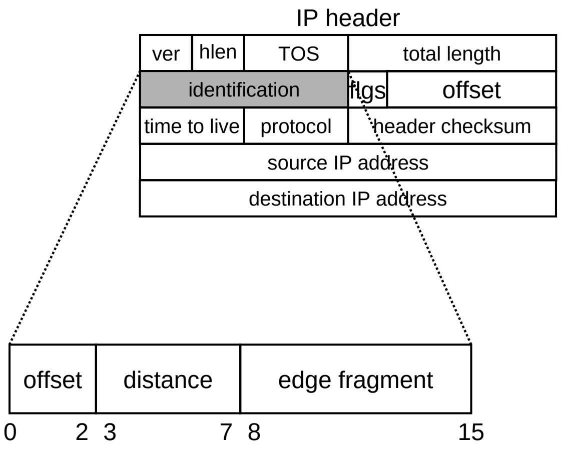

Figure 9: Encoding edge fragments into the IP identification field.

of-service attacks send this many packets in a few seconds.

Finally, we explore the robustness of this algorithm with respect to multiple attackers. For a random hash of length h, the probability of accepting an arbitrarily constructed candidate edge-id is $\frac{1}{2^h}$. In the event that there are m attackers, then at any particular distance d, in the worst case there may be up to m distinct routers.[3] Consequently the probability that any edge-id at distance d is accepted incorrectly is at most:

$$1 - (1 - \frac{1}{2^h})^{m^k}$$

since there are m^k possible combinations of fragments in the worst case. For $h = 32$ and $k = 4$ this means that 100 distinct routers at the same distance (i.e. disjoint attack paths) will be resolved with no errors with a probability of better than 97%. For $h = 32$ and $k = 8$, (the values we use for our implementation) the same certainty can only be provided for 10 distinct routers. However, even in the unlikely event of a corruption at distance d, the probability of propagating this error further is extremely small because the resulting edge-id, when XORed with the previous edge-id, must again produce a correct hash.

The most significant drawback to this scheme is the large number of combinations that must be considered as the multiple attack paths diverge. While these combinations can be computed off-line, for large values of k and m even this can become intractable. Consequently, there is a design tension in the size of k – per-packet space overhead is reduced by a larger k, while computational overhead and robustness benefits from a smaller k.

5.2 IP header encoding

To allow for practical deployment requires that we "overload" existing header fields in a manner that will have minimal impact on existing users. This is a difficult task, especially given that even after prodigious effort we require 16 bits of space. Nonetheless, we believe it possible to obtain this space by overloading the 16-bit IP identification field. This field is currently used to differentiate IP fragments that belong to different packets. We describe our proposed encoding below, and then discuss the issues of backwards-

[3] In practice, the number of distinct routers is likely to be smaller for the portion of the path closest to the receiver, since many attackers will still share significant portions of their attack path with one another.

compatibility that it raises. However, we note that because the issue of backwards-compatible encoding is largely separate from our traceback algorithms, we could adopt any reasonable encoding that comes to light.

Figure 9 depicts our choice for partitioning the identification field: 3 offset bits to represent 8 possible fragments, 5 bits to represent the distance, and 8 bits for the edge fragment. We use a 32-bit hash, which doubles the size of each router address to 64 bits. This implies that 8 separate fragments are needed to represent each edge – each fragment indicated by a unique offset value. Finally, 5 bits is sufficient to represent 32 hops, which is more than almost all Internet paths [CC97, TR00, CAIDA00].[4]

The observant reader will note that this layout is chosen to allow the highest performance implementation of our algorithm, which already had a low per-packet router overhead. In the common case, the only modification to the packet is to increment its distance field. Because of its alignment within the packet, this increment precisely offsets the required decrement of the time-to-live field implemented by each router [Bak95]. Consequently, the header checksum *does not need to be altered at all* and the header manipulation overhead could be even lower than in conventional routers – simply an addition to the distance field, a decrement to the ttl field, and a comparison to check if either has overflowed. In the worst case, our algorithm must read the IP identification field, lookup an edge fragment and XOR it, and fold the write-back into the existing checksum update procedure (a few ALU operations). This overhead is minimal in a software implementation, and easily parallelizable in dedicated hardware.

Reuse of the IP identification field must address issues of backwards-compatibility for IP fragment traffic. Fortunately, recent measurements suggest that that less than 0.25% of packets are fragmented [SZ99, Cla00]. Moreover, it has long been understood that network-layer fragmentation is detrimental to end-to-end performance [KM87] so modern network stacks implement automatic MTU discovery to prevent fragmentation regardless of the underlying media [MD90]. Consequently, we believe that our encoding will inter-operate seamlessly with existing protocol implementations in the vast majority of cases.

However, there is a small but real fraction of legitimate traffic that is fragmented, and we wish to ensure that it is not affected by our modifications to the extent that this is possible. Normally if a packet is fragmented, its identification field is copied to each fragment so the receiver can faithfully reassemble the fragments into the original packet. Our marking procedure can violate this property in one of two ways: by writing different values into the identification fields of fragments from the same datagram or by writing the same values into the identification fields of fragments from different datagrams. These two problems present different challenges and have different solutions.

First, a datagram may be fragmented *upstream* from a marking router. If the fragment is subsequently marked and future fragments from the same datagram are not marked consistently then reassembly may fail or data may be corrupted. While the simplest solution to this problem is to simply not mark fragments, an adversary would quickly learn to evade traceback by exploiting this limitation. In fact, some current denial-of-service attacks already use IP fragments to exploit errors in host IP reassembly functions [CERT97]. Instead, we propose an alternative marking mechanism for fragments. We use a separate marking probability, q, for fragments. When we decide to mark a fragment, we prepend an ICMP "echo reply" header, along with the *full* edge data – truncating the tail of the packet. The packet is consequently "lost" from the standpoint of the receiver, but the edge information is delivered in a way that does not impact legacy hosts. Because we can use the full edge sampling algorithm, q can be more than an order of magnitude smaller than p and yet achieve the same convergence time. This solution increases the loss rate of fragmented flows somewhat (more substantially for longer paths) but preserves the integrity of the data in these flows.

[4]It is also reasonable to turn off marking on any routers that cannot be directly connected to an attacking host (e.g. core routers). This both reduces the convergence time, and increases the "reach" of the distance field.

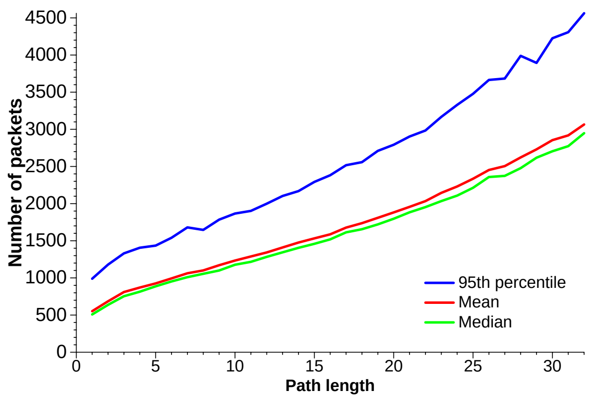

Figure 10: Experimental results for number of packets needed to reconstruct paths of varying lengths. The marking probability, p, is set to $\frac{1}{25}$. Each path length result represents the results of 1,000 independent simulation runs.

A more insidious problem is presented by fragmentation that occurs *downstream* from a marking router. If a marked packet is fragmented, but one of the fragments is lost, then the remaining fragments may linger in the victim's reassembly buffer for an extended period [Bra89]. Future packets marked by the same router can have the same IP identification value and consequently may be incorrectly reassembled with the previous fragments. One possibility is to leave this problem to be dealt with by higher layer checksums. However, not all higher layer protocols employ checksums, and in any case it is dangerous to rely on such checksums because they are typically designed only for low residual error rates. The safest solution we are currently aware of is to set the *Don't Fragment* flag on every marked packet. This will degrade communication between hosts not using MTU path discovery in the rare case that fragmentation is needed, but it will never lead to data corruption.

5.3 Experience

We have implemented the marking and reconstruction portions of our algorithm and have tested it using a simulator that creates random paths and originates attacks. In Figure 10 we graph the mean, median and 95th percentile for the number of packets required to reconstruct paths of varying lengths over 1,000 random test runs for each length value. We assume a marking probability of $\frac{1}{25}$. Note that while the convergence time is theoretically exponential in the path length, all three lines appear linear due to the finite path length and appropriate choice of marking probability.

We see that most paths can be resolved with between one and two thousand packets, and even the longest paths can be resolved with a very high likelihood within four thousand packets. To put these numbers in context, most flooding-style denial of service attacks send many hundreds or thousands of packets each second. The analytic bounds we described earlier are conservative, but in our experience they are no more than 30% higher than our experimental results.

6 Limitations and future work

There are still a number of limitations and loose ends in our approach. We discuss the most important of these here:

- finding the valid suffix in a path,
- approaches for determining the attack origin,
- general limitations of traceback.

6.1 Suffix validation

Some number of the packets sent by the attacker are unmarked by intervening routers. The victim cannot differentiate between these packets and genuine marked packets. Therefore an attacker could insert "fake" edges by carefully manipulating the identification fields in the packets it sends. While the distance field prevents an attacker from spoofing edges between it and the victim – what we call the *valid suffix* – nothing prevents the attacker from spoofing extra edges past the end of the true attack path.

There are several ways to identify the valid suffix within a path generated by the reconstruction procedure. With minimal knowledge of Internet topology one can differentiate between routers that belong to transit networks (e.g. ISPs) and those which belong to stub networks (e.g. enterprise networks). Generally speaking, a valid path will never enter a stub network and then continue into a transit network. Moreover, simple testing tools such as `traceroute` should enable a victim to determine if two networks do, in fact, connect. More advanced network maps [CB00, GT00] can resolve this issue in an increasing number of cases.

A more general mechanism is to provide each router with a "secret" that is sent along with each marked packet (perhaps in the single unallocated bit in the IP flags field). When the victim wants to validate a router in the path, it contacts the associated network (possibly out of band, via telephone or e-mail) and obtains the secret used by the router at the time of the attack. To guard against replay, the secret can be time-varying and hashed with the packet contents. Since the attacker will not know the router's secret, it will not be able to include the proper bit in its forged edge-id fragments. By eliminating edge-ids for which the secret in their constituent fragments can not be validated, we can prune a candidate attack path to only include the valid suffix.

6.2 Attack origin detection

Our algorithm determines the approximate origin of an attacker – in particular, the traceback-capable router closest to the attacker. However, this does not reveal the actual host originating the attack. Since hosts can forge both their IP source address and MAC address the origin of a packet may never be explicitly visible. On shared media such as FDDI rings, this problem can only be solved by explicit testing. However, on point-to-point media, the input port a packet arrives on is frequently enough to determine its true origin. On other media, there may be a MAC address, cell number, channel, or other hint that would help to locate the attack origin. In principle, our algorithm could be modified to report this information by occasionally marking packets with a special edge-id representing a link between the router and the input port on which the packet arrived (or other "hint" information). We have not explored the design of such a feature in any depth.

6.3 Finding attackers

While IP-level traceback is an important part of the solution for stopping denial-of-service attacks, it is by no means a complete solution. In particular, traceback is only effective at finding the source of an *attack*, not necessarily the source of an *attacker*. Stopping an attack may be sufficient to eliminate an immediate problem, but long term disincentives may require a legal remedy and therefore the means to determine an attacker's identity. However, attackers can hide their true identities by "laundering" attacks through third parties, either indirectly (e.g. the smurf attack [CERT98]) or directly via compromised "stepping stone" machines. While there is on-going work on following attackers through intermediate hosts [ZP99, SCH95], there are still significant challenges in developing a generally applicable and universally deployable solution to this problem. Finally, determining an attacker's origin machine may not provide sufficient forensic evidence, by itself, to determine their identity. Even with perfect traceback support, unambiguously identifying a sufficiently skilled and paranoid attacker is likely to require cooperation from law enforcement and telecommunications organizations.

7 Conclusion

In this paper we have argued that denial-of-service attacks motivate the development of improved traceback capabilities and we have explored traceback algorithms based on packet marking in the network. We have shown that this class of algorithm, best embodied in *edge sampling*, can enable efficient and robust multi-party traceback that can be incrementally deployed and efficiently implemented. As well, we have developed variant algorithms that sacrifice convergence time and robustness for reduced per-packet space requirements. Finally, we have suggested one potential deployment strategy using such an algorithm based on overloading existing IP header fields. We have demonstrated that this implementation is capable of fully tracing most attacks after they have sent only a few thousand packets.

References

[Bak95] Fred Baker. Requirements for IP Version 4 Routers. RFC 1812, June 1995.

[BC99] Hal Burch and Bill Cheswick. Tracing Anonymous Packets to Their Approximate Source. Unpublished paper, December 1999.

[BDM99] Gaurav Banga, Peter Druschel, and Jeffrey Mogul. Resource Containers: A New Facility for Resource Management in Server Systems. In *Proceedings of the 1999 USENIX/ACM Symposium on Operating System Design and Implementation*, pages 45–58, February 1999.

[Bel89] Steven M. Bellovin. Security Problems in the TCP/IP Protocol Suite. *ACM Computer Communications Review*, 19(2):32–48, April 1989.

[Bra89] R. Braden. Requirements for Internet Hosts – Communication Layers. RFC 1122, October 1989.

[CAIDA00] Cooperative Association for Internet Data Analysis. Skitter Analysis. `http://www.caida.org/Tools/Skitter/Summary/`, 2000.

[CB00] Bill Cheswick and Hal Burch. Internet Mapping Project. `http://cm.bell-labs.com/who/ches/map/index.html`, 2000.

[CC97] Robert L. Carter and Mark E. Crovella. Dynamic Server Selection Using Dynamic Path Characterization in Wide-Area Networks. In *Proceedings of the 1997 IEEE INFOCOM Conference*, Kobe, Japan, April 1997.

[CERT96] Computer Emergency Response Team. CERT Advisory CA-96.26 Denial-of-Service Attack via pings. `http://www.cert.org/advisories/CA-96.26.ping.html`, December 1996.

[CERT97] Computer Emergency Response Team. CERT Advisory CA-97.28 IP Denial-of-Service Attacks. `http://www.cert.org/advisories/CA-97.28.smurf.html`, December 1997.

[CERT98] Computer Emergency Response Team. CERT Advisory CA-98.01 ”smurf” IP Denial-of-Service Attacks. `http://www.cert.org/advisories/CA-98.01.smurf.html`, January 1998.

[CERT00] Computer Emergency Response Team. CERT Advisory CA-2000-01 Denial-of-Service Developments. `http://www.cert.org/advisories/CA-2000-01.html`, January 2000.

[CF99] Computer Security Institute and Federal Bureau of Investigation. 1999 CSI/FBI Computer Crime and Security Survey. Computer Security Institute publication, March 1999.

[Cis97] Cisco Systems. Configuring TCP Intercept (Prevent Denial-of-Service Attacks). Cisco IOS Documentation, December 1997.

[Cla00] K. Claffy. Sampled Measurements from June 1999 to December 1999 at the AMES Inter-exchange Point. Personal Communication, January 2000.

[Fel66] William Feller. *An Introduction to Probability Theory and Its Applications (2nd edition)*, volume 1. Wiley and Sons, 1966.

[FS98] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing. RFC 2267, January 1998.

[Gla98] James Glave. Smurfing Cripples ISPs. Wired Technolgy News: (`http://www.wired.com/news/news/technology/story/9506.html`), January 1998.

[GS99] Ian Goldberg and Adam Shostack. Freedom Network 1.0 Architecture and Protocols. Zero-Knowledge Systems White Paper, November 1999.

[GT00] Ramesh Govindan and Hongsuda Tangmunarunkit. Heuristics for Internet Map Discovery. In *Proceedings of the 2000 IEEE INFOCOM Conference*, Tel Aviv, Israel, March 2000.

[HB96] L. Todd Heberlein and Matt Bishop. Attack Class: Address Spoofing. In *1996 National Information Systems Security Conference*, pages 371–378, Baltimore, MD, October 1996.

[How98] John D. Howard. *An Analysis of Security Incidents on the Internet*. PhD thesis, Carnegie Mellon University, August 1998.

[KM87] Chrisopher Kent and Jeffrey Mogul. Fragmentation Considered Harmful. In *Proceedings of the 1987 ACM SIGCOMM Conference*, pages 390–401, Stowe, VT, August 1987.

[KS99] Phil Karn and William Simpson. Photuris: Session-Key Management Protocol. RFC 2522, March 1999.

[MD90] Jefferey Mogul and Steve Deering. Path MTU Discovery. RFC 1191, November 1990.

[Mea99] Catherine Meadows. A Formal Framework and Evaluation Method for Network Denial of Service. In *Proceedings of the 1999 IEEE Computer Security Foundations Workshop*, Mordano, Italy, June 1999.

[Mor85] Robert T. Morris. A Weakness in the 4.2BSD Unix TCP/IP Software. Technical Report Computer Science #117, AT&T Bell Labs, February 1985.

[Pax97] Vern Paxson. End-to-End Routing Behavior in the Internet. *IEEE/ACM Transactions on Networking*, 5(5):601–615, October 1997.

[Per96] C. Perkins. IP Mobility Support. RFC 2002, October 1996.

[Pos81] Jon Postel. Internet Protocol. RFC 791, September 1981.

[RRT$^+$98] Eric C. Rosen, Yakov Rekhter, Daniel Tappan, Dino Farinacci, Guy Fedorkow, Tony Li, and Alex Conta. MPLS Label Stack Encoding. Internet Draft: draft-ietf-mpls-label-encaps-07.txt (expires March 2000), September 1998.

[RSG98] Michael G. Reed, Paul F. Syverson, and David M. Goldschlag. Anonymous Connections and Onion Routing. *IEEE Journal on Selected Areas in Communications*, 16(4):482–494, May 1998.

[Sag98] Glenn Sager. Security Fun with OCxmon and cflowd. Presentation at the Internet 2 Working Group, November 1998.

[SCH95] Stuart Staniford-Chen and L. Todd Heberlein. Holding Intruders Accountable on the Internet. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy*, pages 39–49, Oakland, CA, May 1995.

[SP99] Oliver Spatscheck and Larry Peterson. Defending Against Denial of Service Attacks in Scout. In *Proceedings of the 1999 USENIX/ACM Symposium on Operating System Design and Implementation*, pages 59–72, February 1999.

[Sto99] Robert Stone. CenterTrack: An IP Overlay Network for Tracking DoS Floods. Unpublished paper, October 1999.

[SZ99] Ion Stoica and Hui Zhang. Providing Guaranteed Services Without Per Flow Management. In *Proceedings of the 1999 ACM SIGCOMM Conference*, pages 81–94, Boston, MA, August 1999.

[TR00] Wolfgang Theilmann and Kurt Rothermel. Dynamic Distance Maps of the Internet. In *Proceedings of the 2000 IEEE INFOCOM Conference*, Tel Aviv, Israel, March 2000.

[VCIV99] M. Vivo, E. Carrasco, G. Isern, and G. O. Vivo. A review of port scanning techniques. *ACM Computer Communications Review*, 29(2):41–48, April 1999.

[ZP99] Yin Zhang and Vern Paxson. Stepping Stone Detection. Presentation in the New Research Supplement to the 1999 ACM SIGCOMM Conference, August 1999.