Single-Packet IP Traceback

Alex C. Snoeren, *Student Member, IEEE*, Craig Partridge, *Fellow, IEEE*, Luis A. Sanchez, Christine E. Jones, Fabrice Tchakountio, *Member, IEEE*, Beverly Schwartz, Stephen T. Kent, and W. Timothy Strayer, *Senior Member, IEEE*

Abstract— **The design of the IP protocol makes it difficult to reliably identify the originator of an IP packet. Even in the absence of any deliberate attempt to disguise a packet's origin, wide-spread packet forwarding techniques such as NAT and encapsulation may obscure the packet's true source. Techniques have been developed to determine the source of large packet flows, but, to date, no system has been presented to track individual packets in an efficient, scalable fashion. We present a hash-based technique for IP traceback that generates audit trails for traffic within the network, and can trace the origin of a single IP packet delivered by the network in the recent past. We demonstrate that the system is effective, space-efficient (requiring approximately 0.5% of the link capacity per unit time in storage), and implementable in current or next-generation routing hardware. We present both analytic and simulation results showing the system's effectiveness.**

I. INTRODUCTION

TODAY'S Internet infrastructure is extremely vulnerable to motivated and well-equipped attackers. Tools are readily available, from covertly exchanged exploit programs to publicly released vulnerability assessment software, to degrade performance or even disable vital network services. The consequences are serious and, increasingly, financially disastrous. While distributed denial-of-service attacks, typically conducted by flooding network links with large amounts of traffic, are the most widely reported, there are other forms of network attacks, many of which require significantly smaller packet flows. In fact, there are a number of widely-deployed operating systems and routers that can be disabled by a single well-targeted packet (e.g., the Teardrop attack crashes versions of Microsoft Windows with one packet [1]). To institute accountability for these attacks, the source of individual packets must be identified.

Unfortunately, the anonymous nature of the IP protocol makes it difficult to accurately identify the true source of an IP datagram if the source wishes to conceal it. The network routing infrastructure is stateless and based largely on destination addresses; no entity in an IP network is officially responsible for ensuring the source address is correct. Many routers employ a technique called *ingress filtering* [2] to limit source addresses of IP datagrams from a stub network to addresses belonging to that network, but not all routers have the resources necessary to examine the source address of each incoming packet, and ingress filtering provides no protection on transit networks. Furthermore, spoofed source addresses are legitimately used by network address translators (NATs), Mobile IP, and various unidirectional link technologies such as hybrid satellite architectures.

Accordingly, a well-placed attacker can generate offending IP packets that appear to have originated from almost anywhere. While techniques such as ingress filtering, which suppresses packets arriving from a given network with source addresses that do not properly belong to that network, increase the difficulty of mounting an attack, transit networks are dependent upon their peers to perform the appropriate filtering. This interdependence is clearly unacceptable from a liability perspective; each motivated network must be able to secure itself independently.

Systems that can reliably trace individual packets back to their sources are a first and important step in making attackers (or, at least, the systems they use) accountable. There are a number of significant challenges in the construction of such a tracing system including determining which packets to trace, maintaining privacy (a tracing system should not adversely impact the privacy of legitimate users), and minimizing cost (both in router time spent tracking rather than forwarding packets, and in storage used to keep information).

We have developed a *Source Path Isolation Engine* (SPIE) to enable IP *traceback*, the ability to identify the source of a particular IP packet given a copy of the packet to be traced, its destination, and an approximate time of receipt. Historically, tracing individual packets has required prohibitive amounts of memory; one of SPIE's key innovations is to reduce the memory requirement (down to 0.5% of link bandwidth per unit time) through the use of Bloom filters [3]. By storing only packet digests, and not the packets themselves, SPIE also does not increase a network's vulnerability to eavesdropping. SPIE therefore allows routers to efficiently determine if they forwarded a particular packet within a specified time interval while maintaining the privacy of unrelated traffic.

The rest of this paper examines SPIE in detail. We begin by defining the problem of IP traceback in section II, and articulate the desired features of a traceback system. We survey previous work in section III, relating their feature sets against our requirements. Section IV describes the digesting process in detail. Section V presents an overview of the SPIE architecture, while section VI offers a practical implementation of the concepts. Section VII provides both analytic and simulation results evaluating SPIE's traceback success rates. We discuss the issues involved in deploying SPIE in section VIII before concluding in section IX with a brief look at future work.

II. IP TRACEBACK

The concept of IP traceback is not yet well defined. In an attempt to clarify the context in which SPIE was developed, this

This work was sponsored by the Defense Advanced Research Projects Agency (DARPA) under contract No. N66001-00-C-8038. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied.

A. C. Snoeren is with the MIT Laboratory for Computer Science, Cambridge, MA 02139 USA and with BBN Technologies, Cambridge, MA 02138 USA (e-mail: snoeren@lcs.mit.edu). L. Sanchez is with Megisto Systems, Inc. (e-mail: lsanchez@megisto.com). The remaining authors are with BBN Technologies, Cambridge, MA 02138 USA (e-mail: {craig, cej, ftchakou, bschwart, kent, strayer}@bbn.com).

A preliminary version of this paper was presented at ACM SIGCOMM '01 in San Diego, CA, August 2001.

section presents a detailed and formal definition of traceback. We hope that presenting a strawman definition of traceback will also help the community better evaluate different traceback schemes.

In order to remain consistent with the terminology in the literature, we will consider a packet of interest to be nefarious, and term it an *attack packet*; similarly, the destination of the packet is a *victim*. We note, however, that there are many reasons to trace the source of a packet; many packets of interest are sent with no ill intent whatsoever.

A. Assumptions

There are several important assumptions that a traceback system should make about a network and the traffic it carries:

- Packets may be addressed to more than one physical host
- Duplicate packets may exist in the network
- Routers may be subverted, but not often
- Attackers are aware they are being traced
- The routing behavior of the network may be unstable
- The packet size should not grow as a result of tracing
- End hosts may be resource constrained
- Traceback is an infrequent operation

The first two assumptions are simply characteristics of the Internet Protocol. IP packets may contain a multicast or broadcast address as their destination, causing the routing infrastructure to duplicate them internally. An attacker can also inject multiple identical packets itself, possibly at multiple locations. A tracing system must be prepared for a situation where there are multiple sources of the same (identical) packet, or a single source of multiple (also typically identical) packets.

The next two assumptions speak to the capabilities of the attacker(s). An attacker may gain access to routers along (or adjacent to) the path from attacker to victim by a variety of means. Further, a sophisticated attacker is aware of the characteristics of the network, including the possibility that the network is capable of tracing an attack. The traceback system must not be confounded by a motivated attacker who subverts a router with the intent to subvert the tracing system.

The instability of Internet routing is well known [4] and its implications for tracing are important. Two packets sent by the same host to the same destination may traverse wildly different paths. As a result, any system that seeks to determine origins using multi-packet analysis techniques must be prepared to make sense of divergent path information.

The assumption that the packet size should not grow is probably the most controversial. There are a number of protocols today that cause the packet size to grow, for example technologies that rely on packet encapsulation, such as IPsec and mobile IP. However, increasing the packet size causes MTU problems and increases overhead sharply (each byte of additional overhead reduces system bandwidth by about 1%, given the average packet size of about 128 bytes). A recent study by the Cooperative Association for Internet Data Analysis (CAIDA) [5] found that packet encapsulation (and the resulting growth in packet size) is the single largest cause of fragmentation on the Internet. It follows that an efficient traceback system should not cause packet size to grow.

We assume that an end host, and in particular the victim of an attack, may be resource-poor and unable to maintain substantial additional administrative state regarding the routing state or the packets it has previously received. This assumption comes from the observed rise in special purpose devices such as microscopes, cameras, and printers that are attached to the Internet yet have few internal resources other than those devoted to performing their primary task.

The final assumption that traceback queries are infrequent has important design implications. It implies queries can be handled by a router's control path, and need not be dealt with on the forwarding path at line speed. While there may be auditing tasks associated with packet forwarding to support traceback that must be executed while forwarding, the processing of the audit trails is infrequent with respect to their generation.

B. The goal

Ideally, a traceback system should be able to identify the source of any piece of data sent across the network. In an IP framework, the packet is the smallest atomic unit of data. Any smaller division of data (a byte, for instance) is contained within a unique packet. Hence an optimal IP traceback system would precisely identify the source of an arbitrary IP packet. Any larger data unit or stream can be isolated by searching for any particular packet containing data within the stream.[1]

As with any auditing system, a traceback system can only be effective in networks in which it has been deployed. Hence we consider the source of a packet to be one of:

- The ingress point to the traceback-enabled network
- The actual host or network of origin
- One or more compromised routers within the enabled network

If one assumes that any router along the path may be co-opted to assist in concealing a packet's source, it becomes obvious that one must attempt to discern not only the packet's source, but its entire path through the network. Because subverted routers can fabricate trace information, the path can only be guaranteed to be accurate on the portion from the victim to the a source or subverted router, whichever comes first. While subverted routers may attempt to conceal their identity by appending additional sources further up-stream, the subverted routers themselves must still appear as a node in the trace. We consider subverted routers that attempt to conceal the true source of a packet as co-conspirator, and therefore attack sources themselves.

Hence, we are interested in constructing an *attack path*, where the path consists of each router traversed by the packet on its journey from source to the victim. Each node in an attack path either forwarded the packet or lies upstream of a subverted router that did. Further, since multiple, indistinguishable packets may be injected into the network from different sources in the general case, a traceback system should construct an *attack graph* composed of the attack paths for every instance of the attack packet that arrived at the victim.

If routers are subverted, they may provide mis-information to the traceback system, causing the attack graph to contain false

[1] Indeed, we would argue that it is desirable to trace the individual packets within a stream because the individual packets may have originated at different sites (meeting only at the victim) and are likely to have followed different paths through the network.

positives; that is, the attack graph may implicate sources that did not actually emit the packet. We argue these false positives are unavoidable consequence of admitting the possibility of subverted routers. An ideal traceback system, however, produces no false *negatives* while attempting to minimize false positives; it must never exonerate an attacker by not including the attacker in the attack graph.

Further, when a traceback system is deployed, it must not reduce the privacy of IP communications. In particular, entities not involved in the generation, forwarding, or receipt of the original packet should not be able to gain access to packet contents by either utilizing or as part of participating in the IP traceback system. An ideal IP traceback system must not expand the eavesdropping capabilities of a malicious party.

C. Transformations

It is important to note that packets may be modified during the forwarding process. In addition to the standard decrementing of the time to live (TTL) field and checksum recomputation, IP packets may be further transformed by intermediate routers. Packet *transformation* may be the result of valid processing, router error, or malicious intent. A traceback system need not concern itself with packet transformations resulting from error or malicious behavior. Packets resulting from such transformations only need be traced to the point of transformation, as the transforming node either needs to be fixed or can be considered a co-conspirator (source). A complete traceback system should trace packets through valid transformations back to the source of the original packet.

Valid packet transformations are defined as a change of packet state that allows for or enhances network data delivery. Transformations occur due to such reasons as hardware needs, network management, protocol requirements, and source request. Based on the transform produced, packet transformations are categorized as follows:

1. *Packet Encapsulation*: A new packet is generated in which the original packet is encapsulated as the payload (e.g., IPsec). The new packet is forwarded to an intermediate destination for de-encapsulation. Also known as *tunneling*.
2. *Packet Generation*: One or more packets are generated as a direct result of an action by the router on the original packet (e.g., an ICMP Echo Reply sent in response to an ICMP Echo Request, or packet duplication in IP Multicast). The new packets are forwarded and processed independent of the original packet. (A large number of *reflector* attacks utilize such transforms to hide their source [6].)

Common packet transformations include those performed by RFC 1812-compliant routers [7] such as packet fragmentation, IP option processing, ICMP processing, and packet duplication. Network address translation (NAT) and both IP-in-IP and IPsec tunneling are also notable forms of packet transformation. Many of these transformations result in a loss of the original packet state due to the stateless nature of IP networks.

A recent CAIDA study of wide-area traffic patterns found that less than 3% of IP traffic underwent common transformation and IP tunneling [8]. While this study did not encompass all forms of transformation (NAT processing being a notable omission), it seems safe to assume that packet transformations account for a relatively small fraction of the overall IP traffic traversing the Internet today. However, attackers may transmit packets engineered to experience transformation. The ability to trace packets that undergo transformation is, therefore, an essential feature of any viable traceback system.

III. Related Work

There are two approaches to the problem of determining the route of a packet flow: one can audit the flow as it traverses the network, or one can attempt to infer the route based upon its impact on the state of the network. Both approaches become increasingly difficult as the size of the flow decreases, but the latter becomes infeasible when flow sizes approach a single packet because small flows generally have no measurable impact on the network state.

Route inference was pioneered by Burch and Cheswick [9] who considered the problem of large packet flows and proposed a novel technique that systematically floods candidate network links. By watching for variations in the received packet flow due to the restricted link bandwidth, they are able to infer the flow's route. This technique requires considerable knowledge of network topology and the ability to generate large packet floods on arbitrary network links.

One can categorize auditing techniques into two classes according to the way in which they balance resource requirements across the network components. Some techniques require resources at both the end host and the routing infrastructure, others require resources only within the network itself. Of those that require only infrastructure support, some add packet processing to the forwarding engine of the routers while others offload the computation to the control path of the routers.

A. End-host storage

Some auditing approaches attempt to distribute the burden by storing state and performing computation at the end hosts rather than in the network. Routers notify the packet destination of their presence on the route. Because IP packets cannot grow arbitrarily large, schemes have been developed to reduce the amount of space required to send such information. Recently proposed techniques by Savage *et al.* [10] and Bellovin [11] explore in-band and out-of-band signaling, respectively.

Because of the high overhead involved, neither Savage nor Bellovin attempt to provide audit information for every packet. Instead, each employs probabilistic methods that allow sufficiently large packet flows to be traced. By providing partial information on a subset of packets in a flow, auditing routers enable an end host to reconstruct the entire path traversed by the packet flow after receiving a sufficient number of packets belonging to the flow.

The two schemes diverge in the methods used to communicate the information to the end host. Savage *et al.* employ a packet marking scheme that encodes the information in rarely-used fields within the IP header itself. This approach has been extended by Song and Perrig to improve the reconstruction of paths and authenticate the encodings [12]. In order to avoid the backwards compatibility issues and increased computation required by the sophisticated encoding schemes employed in the packet marking schemes, Bellovin's scheme (and later "intentional" extension [13]) simply sends the audit information in an ICMP message.

B. *Infrastructure approaches*

End-host schemes require the end hosts to log meta data in case an incoming packet proves to be offensive. Alternatively, the network itself can be charged with maintaining the audit trails.

The obvious approach to auditing packet flow is simply to log packets at various points throughout the network and then use appropriate extraction techniques to discover the packet's path through the network. Logging requires no computation on the router's fast path and, thus, can be implemented efficiently in today's router architecture. Sager suggests such a monitoring approach [14]. However, the effectiveness of the logs is limited by the amount of space available to store them. Given today's link speeds, packet logs quickly grow to intractable sizes, even over relatively short time frames. An OC-192 link is capable of transferring 1.25GB per second. If one allows 60 seconds to conduct a query, a router with 16 links would require 1.2TB of high-speed storage.

These requirements can be reduced by sampling techniques similar to those of the end-host schemes, but down-sampling reduces the probability of detecting small flows and does not alleviate the security issues raised by storing complete packets in the router. The ability of an attacker to break into a router and capture terrabytes of actual traffic has severe privacy implications.

Alternatively, routers can be tasked to perform more sophisticated auditing in real time, extracting a smaller amount of information as packets are forwarded. Many currently available routers support *input debugging*, a feature that identifies on which incoming port a particular outgoing packet (or set of packets) of interest arrived. Since no history is stored, however, this process must be activated before an attack packet passes by. Furthermore, due to the high overhead of this operation on many popular router architectures, activating it may have adverse effects on the traffic currently being serviced by the router.

C. *Specialized routing*

One of the main problems with the link testing or logging methods above is the large amount of repetition required. A trace is conducted in a hop-by-hop fashion, querying each router along the way. Once the incoming link or links have been identified, the process must be repeated at the upstream router.

Several techniques have been developed to streamline and automate this process. Some ISPs have developed their own ad hoc mechanisms for automatically conducting input debugging across their networks. Schnackenberg *et al.* [15] propose a more general Intruder Detection and Isolation Protocol (IDIP) to facilitate interaction between routers involved in a traceback effort. IDIP does not specify how participating entities should track packet traffic; it simply requires that they be able to determine whether or not they have seen a component of an attack matching a certain description. Even with automated tools, however, each router in the ISP must support input debugging or logging which are not common in today's high-speed routers for reasons discussed above.

In order to avoid this requirement, Stone [16] suggests constructing an overlay network connecting all the edge routers of an ISP. By using a deliberately simple topology of specialized routers, suspicious flows can be dynamically rerouted across the special tracking network for analysis. This approach has two major shortcomings. First, the attack must be sufficiently long-lived to allow the ISP to effect the rerouting before the relevant flow terminates. Second, the routing change is perceptible by the attacker, and an especially motivated attacker may be able to escape detection by taking appropriate action. While techniques exist to hide precisely what changed about the route, changes in layer-three topology are hard to mask.

IV. PACKET DIGESTING

SPIE, the Source Path Isolation Engine, uses auditing techniques to support the traceback of individual packets while reducing the storage requirements by several orders of magnitude over current log-based techniques [14]. Traffic auditing is accomplished by computing and storing packet digests rather than storing the packets themselves. In addition to reducing storage requirements, storing packet digests instead of the actual packet contents preserves traffic confidentiality by preventing SPIE from being used as a tool for eavesdropping.

A. *Digest input*

The packet content used as input to the digesting function must uniquely represent an IP packet and enable the identification of the packet across hops in the forwarding path. At the same time, it is desirable to limit the size of the digest input both for performance and for reasons discussed below (c.f. section V-C). Duffield and Grossglauser encountered similar requirements while sampling a subset of forwarded packets in an attempt to measure traffic flows [17]. We use a similar approach, masking variant packet content and selecting an appropriate-length prefix of the packet to use as input to the digesting function. Our choice of invariant fields and prefix length is slightly different, however.[2]

Figure 1 shows an IP packet and the fields included by the SPIE digesting function. SPIE computes digests over the invariant portion of the IP header and the first 8 bytes of the payload. Frequently modified header fields are masked prior to digesting. Note that beyond the obvious fields (TTL, TOS, and checksum), certain IP options cause routers to rewrite the option field at various intervals. To ensure a packet appears identical at all steps along its route, SPIE masks or compensates for these fields when computing the packet digests. It is important to note that the invariant IP fields used for SPIE digesting may occasionally be modified by a packet transform (c.f. section V-C).

Our research indicates that the first 24 *invariant* bytes of a packet (20-byte IP header with 4 bytes masked out plus the first 8 bytes of payload) are sufficient to differentiate almost all non-

[2]Because we sample a smaller portion of the packet (28 vs. 40 bytes), we include fields like header length and protocol that Duffield and Grossglauser eschewed due to their lower entropy.

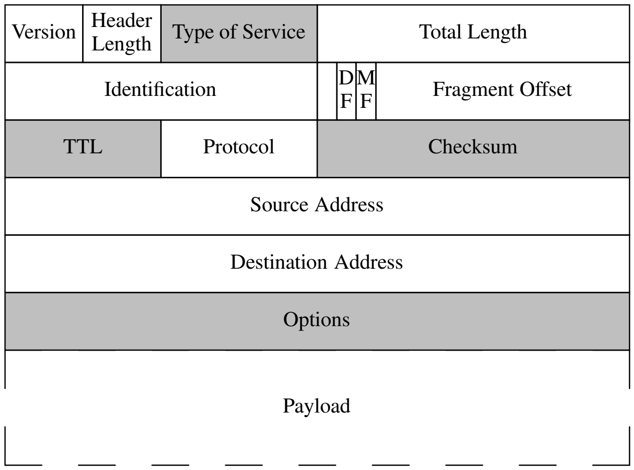

Fig. 1. The fields of an IP packet. Fields in gray are masked out before digesting, including the Type of Service, Time to Live (TTL), IP checksum, and IP options fields.

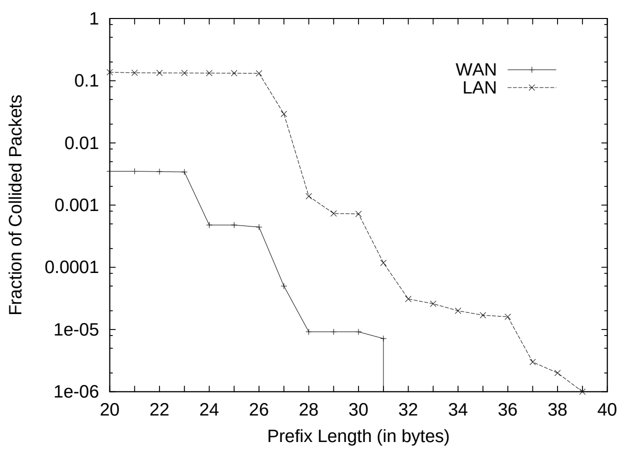

Fig. 2. The fraction of packets that collide (with ToS, TTL, and checksum fields masked out) as a function of prefix length. The WAN trace represents 985,150 packets (with 5,801 duplicates removed) between 6,031 host pairs collected on July 20, 2000 at the University of Florida OC-3 gateway. The LAN trace consists of one million packets (317 duplicates removed) between 2,879 host pairs observed on an Ethernet segment at the MIT Lab for Computer Science.

identical packets. Figure 2 presents the rate of packet collisions for an increasing prefix length for two representative traces: a WAN trace from an OC-3 gateway router, and a LAN trace from an active 100Mb Ethernet segment. (Results were similar for traces across a number of sites.) Two unique packets which are identical up to the specified prefix length are termed a collision. A 28-byte prefix (only 24 non-masked bytes) results in a collision rate of approximately 0.00092% in the wide area and 0.139% on the LAN.

Unlike similar results reported by Duffield and Grossglauser [17, fig. 4], our results include only unique packets; exact duplicates were removed from the packet trace. Close inspection of packets in the wide area with identical prefixes indicates that packets with matching prefix lengths of 22 and 23 bytes are ICMP Time Exceeded error packets with the IP identification field set to zero. Similarly, packets with matching prefixes between 24 and 31 bytes in length are TCP packets with IP identifications also set to zero which are first differentiated by

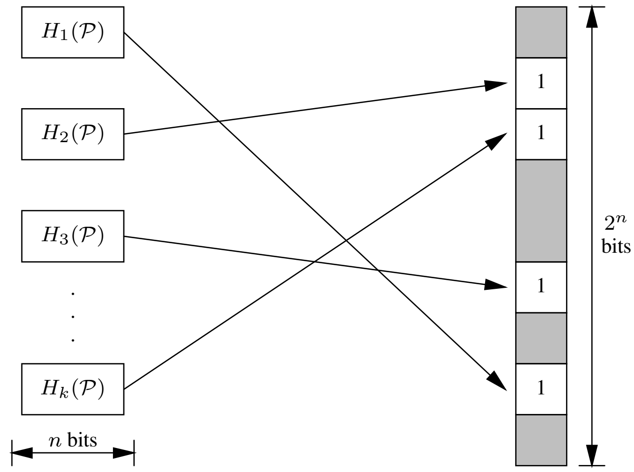

Fig. 3. For each packet received, SPIE computes k independent n-bit digests, and sets the corresponding bits in the 2^n-bit digest table.

the TCP sequence number or acknowledgment fields.[3]

The markedly higher collision rate in the local area is due to the lack of address and traffic diversity. This expected result does not significantly impact SPIE's performance, however. LANs are likely to exist at only two points in an attack graph: immediately surrounding the victim and the attacker(s). False positives on the victim's local network can be easily eliminated from the attack graph—they likely share the same gateway router in any event. False positives at the source are unlikely if the attacker is using spoofed source addresses, as this provides the missing diversity in attack traffic, and remain in the immediate vicinity of the true attacker by definition. Hence, for the purposes of SPIE, IP packets are effectively distinguished by the first 24 invariant bytes of the packet.

B. *Bloom filters*

Constructing a digest table containing packet digests corresponding to the traffic forwarded by a router for a given time interval is a challenging task. A naive technique that simply stored the digests themselves would require massive amounts of storage. Instead, SPIE implements digest tables using space-efficient data structures known as Bloom filters [3]. A Bloom filter computes k distinct packet digests for each packet using independent uniform hash functions, and uses the n-bit results to index into a 2^n-sized bit array. The array is initialized to all zeros, and bits are set to one as packets are received. Figure 3 depicts a Bloom filter with k hash functions.

Membership tests can be conducted simply by computing the k digests on the packet in question and checking the indicated bit positions. If any one of them is zero, the packet was not stored in the table. If, however, all the bits are one, it is highly likely the packet was stored. It is possible that some set of other insertions caused all the bits to be set, creating a *false positive*, but the rate of such false positives can be controlled by only allowing an individual Bloom filter to store a limited number of digests [18]. Saturated filters can be swapped out for a new, empty filter, and archived for later querying.

[3] Further investigation indicates a number of current operating systems, including recent versions of Linux, frequently set the IP ID to zero.

C. Hash functions

SPIE places three major restrictions on the family of hash functions, $\mathcal{F}$, used as digesting functions in its Bloom filters. First, each member function must distribute a highly correlated set of input values (IP packet prefixes), $\mathcal{P}$, as uniformly as possible over the hash's result value space. That is, for a hash function $H : \mathcal{P} \rightarrow 2^m$ in $\mathcal{F}$, and distinct packets $x \neq y \in \mathcal{P}$, $\Pr[H(x) = H(y)] = 2^{-m}$. This is a standard property of good hash functions.

SPIE further requires that the event that two packets collide in one hash function ($H(x) = H(y)$ for some H) be independent of collision events in any other functions ($H'(x) = H'(y), H' \neq H$). Intuitively, this implies false positives at one router are independent of false positives at neighboring routers. Formally, for any function $H \in \mathcal{F}$ chosen at random independently of the input packets x and y, $\Pr[H(x) = H(y)] = 2^{-m}$ with high probability. Such hash families, called *universal hash families*, were first defined by Carter and Wegman [19] and can be implemented in a variety of fashions [20], [21], [22].

Finally, member functions must be straightforward to compute at high link speeds. This requirement is not impractical because SPIE hash functions do not require any cryptographic "hardness" properties. That is, it does not have to be difficult to generate a valid input packet given a particular hash value. Being able to create a packet with a particular hash value enables three classes of attacks, each of which is fairly benign. One attack would ensure that all attack packets have the same fingerprint in the Bloom filter at some router (which is very difficult since there are multiple, independent hashes at each router), but this achievement is of little use, as the packet fingerprints would be distinct at neighboring routers (due to the independent hash functions at each router). Another attack is to ensure all attack packets have different fingerprints, but that is the common case already. The third, and most difficult attack, is to create an attack packet with the same fingerprint as another, non-attack packet. In general, this attack simply adds one additional false-positive node (where the two packets are indistinguishable) to the attack graph of both packets.

V. Source Path Isolation Engine

SPIE-enhanced routers maintain a cache of packet digests for recently forwarded traffic. If a packet is determined to be offensive by some intrusion detection system (or judged interesting by some other metric), a query is dispatched to SPIE which in turn queries routers for packet digests of the relevant time periods. The results of this query are used in a simulated reverse-path flooding algorithm to build an attack graph that indicates the packet's source(s).

A. Architecture

The tasks of packet auditing, query processing, and attack graph generation are dispersed among separate components in the SPIE system. Figure 4 shows the three major architectural components of the SPIE system. Each SPIE-enhanced router has a Data Generation Agent (DGA) associated with it. Depending upon the type of router in question, the DGA can be implemented and deployed as a software agent, an interface card plug to the switching background bus, or a separate auxiliary box connected to the router through some auxiliary interface.

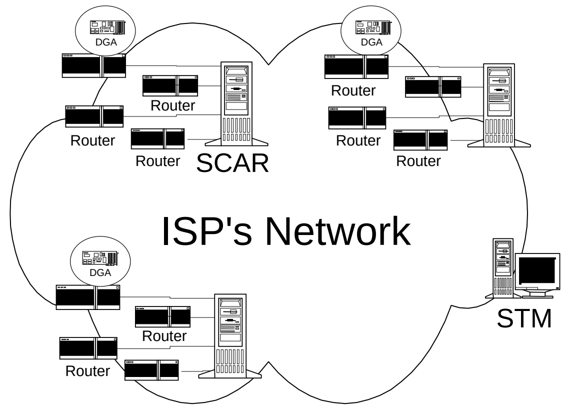

Fig. 4. The SPIE network infrastructure, consisting of Data Generation Agents (DGAs), SPIE Collection and Reduction Agents (SCARs), and a SPIE Traceback Manager (STM).

The DGA produces packet digests of each packet as it departs the router, and stores the digests in time-stamped digest tables. The tables are paged every so often, and represent the set of traffic forwarded by the router for a particular interval of time. Each table is annotated with the time interval and the set of hash functions used to compute the packet digests over that interval. The digest tables are stored locally at the DGA for some period of time, depending on the resource constraints of the router.

SCARs are responsible for a particular region of the network, serving as data concentration points for several routers and facilitating traceback of any packets that traverse the region. Due to the complex topologies of today's ISPs, there will typically be several SCARs distributed over an entire network. Upon request, each SCAR produces an attack graph for its particular region. The attack graphs from each SCAR are grafted together to form a complete attack graph by the SPIE Traceback Manager (STM).

The STM controls the whole SPIE system. The STM is the interface to the intrusion detection system or other entity requesting a packet trace. When a request is presented to the STM, it verifies the authenticity of the request, dispatches the request to the appropriate SCARs, gathers the resulting attack graphs, and assembles them into a complete attack graph. Upon completion of the traceback process, the STM replies to the intrusion detection system with the final attack graph.

B. Traceback processing

Before the traceback process can begin, an attack packet must be identified. Most likely, an intrusion detection system will determine that an exceptional event has occurred and provide the STM with a packet, $\mathcal{P}$, victim, V, and time of attack, T. SPIE places two constraints on the intrusion detection system: the victim must be expressed in terms of the last-hop router, not the end host itself, and the attack packet must be identified in a timely fashion. The first requirement provides the query process with a starting point; the latter stems from the fact that traceback must be initiated before the appropriate digest tables are overwritten by the DGAs. This time constraint is directly related to the amount of resources dedicated to the storage of traffic digests. (We discuss timing and resource tradeoffs in section VII).

Upon receipt of traceback request, the STM cryptographically verifies its authenticity and integrity. Any entity wishing to employ SPIE to perform a traceback operation must be properly authorized in order to prevent denial-of-service attacks. Upon successful verification, the STM dispatches the query to the relevant SCARs for processing. Beginning at the SCAR responsible for the victim's region of the network, the STM sends a query message containing $\mathcal{P}$, V, and T as provided by the intrusion detection system. The SCAR polls its DGAs and responds with a partial attack graph, the time T' the packet entered the region, and the entering packet itself $\mathcal{P}'$ (it may have been transformed, possibly multiple times, within the region).

The attack graph either terminates within the region managed by the SCAR, in which case a source has been identified, or it contains nodes at the edge of the SCAR's network region. In the latter case the STM sends a new query for the transformed packet $\mathcal{P}'$ to the SCAR for the abutting network region. This query uses the border router between the two network regions as its victim, V', and T' as the time of attack. This process continues until all branches of the attack graph terminate, either at a source within the network, or at the edge of the SPIE system. The STM then constructs a composite attack graph which it returns to the intrusion detection system.

C. Transformation processing

IP packets may undergo valid transformation while traversing the network, and SPIE must be capable of tracing through such transformations. In particular, SPIE must be able to reconstruct the original packet from the transformed packet. Unfortunately, many transformations are not invertible without additional information due to the stateless nature of IP networks. Consequently, SPIE must record sufficient packet data at the time of transformation to allow the original packet to be reconstructed.

The packet data chosen as input to the digesting function determines the set of packet transformations SPIE must handle—SPIE need only consider transformations that modify fields used as input to the digest function. SPIE computes digests over the IP header and the first eight bytes of the packet payload but masks out (or omits in the case of IP options) several frequently updated fields before digesting, as shown in figure 1 of section IV. Masking hides most hop-by-hop transformations from the digesting function, but forces SPIE to explicitly handle each of the following transformations: fragmentation, network address translation (NAT), ICMP messages, IP-in-IP tunneling, and IP security (IPsec).

Recording the information necessary to reconstruct the original packet from a transformed packet requires additional resources. Fortunately for SPIE, the circumstances that cause a packet to undergo a transformation will generally take that packet off of the fast path of the router and put it onto the control path, relaxing the timing requirements. The router's memory constraints remain unchanged, however; hence, transformation information must be stored in a scalable and space-efficient manner.

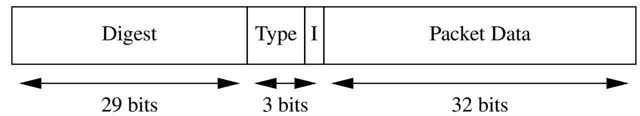

Fig. 5. A Transform Lookup Table (TLT) stores sufficient information to invert packet transformations at SPIE routers. The table is indexed by packet digest, specifies the type of transformation, and stores any irrecoverable packet data.

C.1 Transform lookup table

Along with each packet digest table collected at a DGA, SPIE maintains a corresponding transform table for the same interval of time called a *transform lookup table*, or TLT. Each entry in the TLT contains three fields, shown in figure 5. The first field stores a digest of the transformed packet. The second field specifies the type of transformation—three bits are sufficient to uniquely identify the transformation type among those supported by SPIE. The last field contains a variable amount of packet data the length of which depends upon the type of transformation being recorded.

For space efficiency, the data field is limited to 32 bits. Some transformations, such as network address translation, may require more space. These transformations utilize a level of indirection—one bit of the transformation type field is reserved as an *indirect* flag. If the indirect, or I, flag is set, the third field of the TLT is treated as a pointer to an external data structure which contains the information necessary to reconstruct the packet.

The indirect flag can also be used for flow caching. In many cases (e.g., tunneling or NAT), packets undergoing a particular transformation are related. In such cases, it is possible to reduce the storage requirements by suppressing duplicate packet data, instead referencing a single copy of the required data that can be used to reconstruct any packet in the flow. Such a scheme requires, however, that the SPIE-enabled router itself be capable of flow caching, or at least identification, so that the packets within the flow can be correlated and stored appropriately.

In order to preserve alignment, it is likely efficient implementations would store only 29 bits of the packet digest resulting in 64-bit wide TLT entries. This width implies eight distinct packet digests will map to the same TLT entry. The relative rarity of packet transformations [8], the sparsity of the digest table, and the uniformity of the digesting function combine to make collisions extremely rare in practice. Assuming a digest table capacity of roughly 3.2Mpkts (16Mb SRAM, see section VII-B) and a transformation rate of 3%, the expected collision rate is approximately 1:5333 packets. Even if a collision occurs, it simply results in an additional possible transformation of the queried packet. Each transformation is computed (including the null transformation) and traceback continues. Incorrectly transformed packets likely will not exist at neighboring routers and, thus, will not contribute any false nodes to the attack graph.

C.2 Special-purpose gateways

Some classes of packet transformations, notably NAT and tunneling, are often performed on a large fraction of packets passing through a particular gateway. The transform lookup table would quickly grow to an unmanageable size in such instances; hence, SPIE considers the security gateway or NAT functionality of routers as a separate entity. Standard routing transformations are handled as above, but special purpose gateway transformations require a different approach to transformation handling. Transformations in these types of gateways are generally computed in a stateful way (usually based on a static rule set); hence, they can be inverted in a similar fashion. While the details are implementation-specific, inverting such transformations is straightforward; we do not consider it here.

C.3 Sample transformations

A good example of transformation is packet fragmentation. To avoid needing to store any of the packet payload, SPIE supports inversion of only the first packet fragment, i.e., only the first fragment may be traced back beyond the point of fragmentation. The remaining fragments may be traced to the point of fragmentation, but no further. Note that for most fragment-based attacks [1], the attacker inserts fragments directly into the network (i.e., the attacker is the point of fragmentation) so the traceback is complete. (If only a subset of the fragments is received by the victim the packet cannot be reassembled; hence, the only viable attack is a denial-of-service attack on the victim's reassembly engine. But, if the fragmentation occurs within the network itself, an attacker cannot control which fragments are received by the victim so the victim will eventually receive a first fragment to use in traceback.) Packet data to be recorded includes the total length, fragment offset, and more fragments (MF) field. Since properly-behaving IP routers cannot create fragments with less than 8 bytes of payload information [23], when given the first fragment, SPIE is always able to invert fragmentation and reconstruct the header and at least 64 bits of payload of the pre-fragmented packet which is sufficient to continue traceback.

Observe that SPIE never needs to record any packet payload information. ICMP transformations can be inverted because ICMP error messages always include at least the first 64 bits of the offending packet [24]. Careful readers may be concerned that encapsulation cannot be inverted if the encapsulated packet is subsequently fragmented and the fragments containing the encapsulated IP header and first 64 bits of payload are not available. While this is strictly true, such transformations need to be inverted only in extreme cases as it takes a very sophisticated attacker to cause a packet to be first encapsulated, then fragmented, and then ensure fragment loss. If all the fragments are received, the original header can be extracted from the reassembled payload. It seems quite difficult for an attacker to ensure that packet fragments are lost. It can cause packet loss by flooding the link, but to do so requires sending such a large number of packets that it is very likely that all the fragments for at least one packet will be successfully received by the de-encapsulator for use in traceback.

D. Graph construction

Each SCAR constructs a subgraph using topology information about its particular region of the network. After querying each of the DGAs in its region, a SCAR simulates reverse-path flooding by examining the results in the order they would be queried if an actual reverse path flood was conducted on the topology that existed at the time the packet was forwarded. (The topology information itself is collected and stored independently at each DGA along with the digest tables, and returned to the SCAR as part of the query response.) Figure 6 shows how reverse-path flooding would discover the attack path from V to A, querying routers $R_8, R_9, R_7, R_4, S_5, R_5, R_3$, and R_2 along the way. It is important to note that the routers need not actually be queried sequentially—the SCAR proactively queries each DGA and caches the results locally.

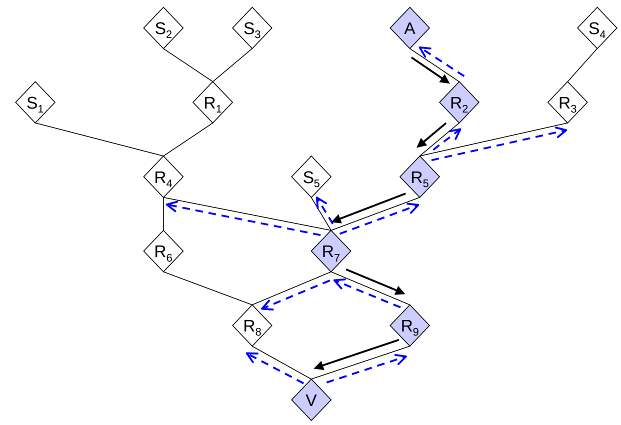

Fig. 6. Reverse path flooding, starting at the victim's router, V, and proceeding backwards toward the attacker, A. Solid arrows represent the attack path; dashed arrows are SPIE queries. Queries are dropped by routers that did not forward the packet in question.

In order to respond to a SCAR's query, a DGA computes the appropriate set of digests and consults the digest table for the indicated time period. If an entry exists for the packet in question, the router is considered to have forwarded the packet. If, however, the digest is not found in the indicated table, it may be necessary to search the digest table corresponding to the immediately preceding time period. Depending on the link latency between routers, DGAs may need to search multiple digest tables in order to assure they have examined an appropriate time frame (which is determined by the link latency and maximum queuing delay at that router). Once a digest is located, the packet arrival time is always considered to be the latest possible time in the interval. This ensures the packet must have been seen at an earlier time at adjacent routers.

Along with the digest tables, each DGA also consults its TLTs for the same time intervals. If the packet was transformed, the DGA informs the SCAR, which then reissues queries to the other DGAs in the region containing the transformed packet and an updated arrival time. If the packet is not found in any of the digest tables or TLTs for the relevant time period, a negative result is returned by the DGA, and the SCAR considers that particular branch of the search tree to be terminated.

The result of this procedure is a connected graph containing the set of nodes believed to have forwarded the packet toward the victim. Assuming correct operation of the routers, this graph is guaranteed to be a superset of the actual attack graph. But due to digest collisions, there may be nodes in the attack graph that are not in the actual attack graph. We call these nodes *false positives* and base the success of SPIE on its ability to limit the number of false positives contained in a returned attack graph.

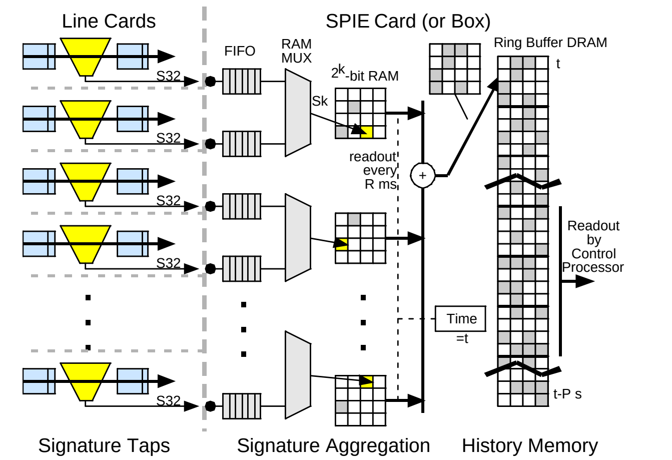

Fig. 7. A sample SPIE DGA hardware implementation for high-speed routers.

VI. Practical Implementation

For our PC-based SPIE prototype, we simulate a universal hash family using MD5 [25]. A random member is defined by selecting a random input vector to prepend to each packet. The properties of MD5 ensure that the digests of identical packets with different input vectors are independent. The 128-bit output of MD5 is then considered as four independent 32-bit digests which can support Bloom filters of dimension up to four. Router implementations requiring higher performance are likely to prefer other universal hash families specifically tailored to hardware implementation [22]. A simple family amenable to fast hardware implementation could be constructed by computing a CRC modulo a random member of the set of indivisible polynomials over Z_{2^k}.

In order to ensure hash independence, each router periodically generates a set of k independent input vectors and uses them to select k digest functions needed for the Bloom filter from the family of universal hashes. These input vectors are computed using a pseudo-random number generator which is independently seeded at each router. For increased robustness against adversarial traffic, the input vectors are changed each time the digest table is paged, resulting in independence not only across routers but also across time periods.

The size of the digest bit vector, or *digest table*, varies with the total traffic capacity of the router; faster routers need larger vectors for the same time period. Similarly, the optimum number of hash functions varies with the size of the bit vector. Routers with tight memory constraints can compute additional digest functions and provide the same false-positive rates as those who compute fewer digests but provide a larger bit vector.

Figure 7 depicts a possible implementation of a SPIE Data Generation Agent in hardware for use on high-speed routers. A full discussion of the details of the architecture and an analysis of its performance were presented previously [26]. Briefly, each interface card in the router is outfitted with an Interface Tap which computes multiple independent digests of each packet as it is forwarded. These digests are passed to a separate SPIE processor (implemented either in a line card form factor or as an external unit) which stores them as described above in digest tables for specific time periods.

As time passes, the forwarded traffic will begin to fill the digest tables and they must be paged out before they become oversaturated, resulting in unacceptable false-positive rates. The tables are stored in a history buffer implemented as a large ring buffer. Digest tables can then be queried or archived by a separate control processor while they are stored in the ring buffer.

VII. Analysis

There are several tradeoffs involved when determining the optimum amount of resources to dedicate to SPIE on an individual router or the network as a whole. SPIE's resource requirements can be expressed in terms of two quantities: the number of packet digest functions used by the Bloom filter, and the amount of memory used to store packet digests. Similarly, SPIE's performance can be characterized in two orthogonal dimensions. The first is the length of time for which packet digests are kept. Queries can only be issued while the digests are cached; unless archived to some external storage device within a reasonable amount of time, the DGAs will discard the digest tables in order to make room for more recent ones. The second is the accuracy of the candidate attack graphs which can be measured in the number of false positives in the graph returned by SPIE.

Both of these metrics can be controlled by adjusting operational parameters. In particular, the more memory available for storing packet digests, the longer the time queries can be issued. Similarly, digest tables with lower false-positive rates yield more accurate attack graphs. Hence, we wish to characterize the performance of SPIE in terms of the amount of available memory and digest table performance.

A. *False positives*

We first relate the rate of false positives in an attack graph to the rate of false positives in an individual digest table. This relationship depends on the actual network topology and traffic being forwarded at the time. We can, however, make some simplifying assumptions in order to derive an upper bound on the number of false positives as a function of digest table performance.

A.1 Analytic bounds

Suppose, for example, each router whose neighbors have degree at most d ensures its digest tables have a false-positive rate of at most $P = p/d$, where $0 \leq p/d \leq 1$ (p is an arbitrary tuning factor). It is easy to show that for any true attack graph G with n nodes, the attack graph returned by SPIE will have at most $np/(1-p)$ extra nodes in expectation. In other words, an average traceback will result in an attack graph with no more than $np/(1-p)$ false positives. We say "no more than" because the digest tables will typically not be at full capacity when queried, resulting in a lower false-positive rate than predicted.

The false-positive rate of a digest table varies over time, depending on the traffic load at the router and the amount of time since it was paged. Similarly, if the tables are paged on a strict schedule based on maximum link capacity, and the actual traffic load is less, digest tables will never reach their rated capacity. Hence, the analytic result is a worst case bound since the digest table performs strictly better while it is only partially full. It represents the expected number of false positives returned if the query was conducted at the worst possible moment, i.e., when all digest tables were at maximum capacity. Furthermore, our analysis assumes the set of neighbors at each node is disjoint which is not true in real networks. It seems reasonable to expect, therefore, that the false-positive rate over real topologies with actual utilization rates would be substantially lower.

For the purposes of this discussion, we arbitrarily select a false-positive rate of $n/7$, resulting in no more than 5 additional nodes in expectation for a path length of over 35 nodes (approaching the diameter of the Internet) according to our theoretical model. Using the bound above, $p = 1/8$ is then a reasonable starting point and we turn to considering its effectiveness in practice.

A.2 Simulation results

In order to relate false-positive rate to digest table performance in real topologies, we have run extensive simulations using the actual network topology of a national tier-one ISP made up of roughly 70 backbone routers with links ranging from T-1 to OC-3. We obtained a topology snapshot and average link utilization data for the ISP's network backbone for a week-long period toward the end of 2000, sampled using periodic SNMP queries, and averaged over the week.

We simulated an attack by randomly selecting a source and victim, and sending 1000 attack packets at a constant rate between them. Each packet is recorded by every intermediate router along the path from source to destination. A traceback is then simulated starting at the victim router and (hopefully) proceeding toward the source. Uniformly distributed background traffic is simulated by selecting a fixed maximum false-positive rate, P, for the digest table at each off-path router. (Real background traffic is not uniform, which would result in slight dependencies in the false-positive rates between routers, but we believe that this represents a reasonable starting point.) In order to accurately model performance with real traffic loads, the effective false-positive rate is scaled by the observed traffic load at each router.

For clarity, we consider a non-transformed packet with only one source and one destination. Preliminary experiments with multiple sources (as might be expected in a distributed denial-of-service (DDoS) attack) indicate false positives scale linearly with respect to the size of the attack graph, which is the union of the attack paths for each copy of the packet. We do not, however, consider this case in the experiments presented here. (A DDoS attack sending identical packets from multiple sources only aids SPIE in its task. A wise attacker would instead send *distinct* packets from each source, forcing the victim to trace each packet individually.)

In order to validate our analytic bound, we have plotted the expected number of false positives as a function of attack path length and digest table performance, $np/(1-p)$ as computed above, and show that in comparison to the results of three simulations on our ISP backbone topology in figure 8. In the first simulation, we set the maximum digest table false-positive probability to $P = p/d$, as prescribed above. This setting results false-positive rate significantly lower than the analytic bound.

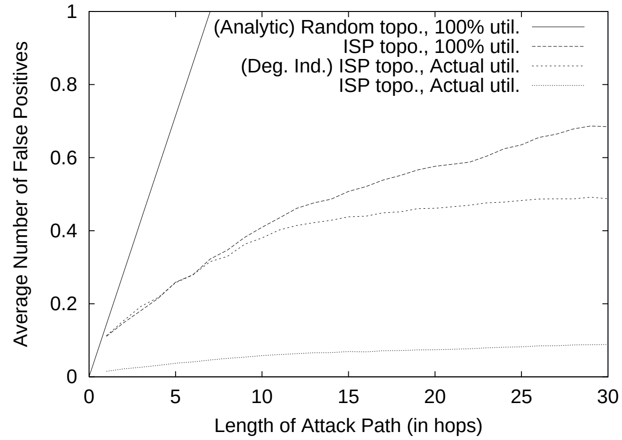

Fig. 8. The number of false positives in a SPIE-generated attack graph as a function of the attack path length, for $p = 1/8$. The analytic bound assuming random topology and 100% link utilization is plotted against three simulation results, two with false-positive rates conditioned on router degree, one without. For the two degree-dependent runs, one considered observed link utilization, while the other assumed full utilization. Each simulation represents the average of 5000 runs using topology and utilization data from a national tier-one ISP.

A significant portion of the disparity results from the relatively low link utilizations maintained by operational backbones (77% of the links in our data set had utilization rates of less than 25%), as can be seen by comparing the results to a second simulation on the ISP topology assuming full link utilization. There remains, however, a considerable gap between the analytic bound and simulated performance in network backbones.

The non-linearity of the simulation results indicates there is a strong damping factor due to the topological structure of the network. Intuitively, routers with many neighbors are found at the core of the network (or at peering points), and routers with fewer neighbors are found toward the edge of the network. This suggests false positives induced by core routers may quickly die out as the attack graph proceeds toward less well-connected routers at the edge.

To examine the dependence upon vertex degree, we conducted a third simulation in the ISP topology. This time, we removed the false-positive rate's dependence upon the degree of the router's neighbors, setting the digest table performance to simply $P = p$ (and returning to actual utilization data). While there is a marked increase in the number of false positives, it remains well below the analytic bound. This somewhat surprising result indicates that despite the analytic bound's dependence on router degree, the hierarchical structure of ISP backbones may permit a relaxation of the coupling, allowing the false positive rate of the digest tables, P, to be set independently of the degree, d, resulting in significant space savings.

B. *Time and memory utilization*

The amount of time during which queries can be supported is directly dependent on the amount of memory dedicated to SPIE. The appropriate amount of time varies depending upon the responsiveness of the method used to identify attack packets. For the purposes of discussion, however, we will assume one minute is a reasonable amount of time in which to identify an attack packet and initiate a traceback. As discussed in section V-A, once the appropriate digest tables have been queried by the SCARs the actual traceback process can be delayed arbitrarily.

B.1 Memory size

Given a particular length of time, the amount of memory required varies linearly with the total link capacity at the router and can be dramatically affected by the dimension of the Bloom filter in use. Bloom filters are typically described in terms of the number of digesting functions. The effective false-positive rate for a Bloom filter that uses k digest functions to store n packets in m bits of memory can be expressed as

$$P = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k. \qquad (1)$$

The performance of a Bloom filter can be quantified in terms of its memory efficiency factor (n/m) and false-positive rate P. For example, a Bloom filter with memory efficiency of 0.2 would need $5n$ bits in order to store n packets while delivering its expected false-positive rate. By solving equation 1 for (n/m) and differentiating with respect to k, it is easy to check that optimal memory efficiency is reached when $k = \log(1/P)$. That is, a Bloom filter with either $\lceil -\log P \rceil$ or $\lfloor -\log P \rfloor$ hash functions has the maximum memory efficiency for a given false-positive rate P. The memory requirement of such a table can easily be determined by substituting P back into equation 1 (observe $P^{(1/k)} = 1/2$):

$$m = -n \cdot \log(1/P)/\ln(1/2) \approx 1.44n \cdot \log(1/P). \qquad (2)$$

Tables providing the effective false-positive rates for various memory efficiencies and digesting functions are readily available [18]. For the purposes of discussion, we will consider using a Bloom filter with three digesting functions $(k = 3)$ and a memory efficiency factor (n/m) of 0.2. Such a filter provides an effective false-positive rate of $P = 0.092$ when full.

While this is well below the value of $1/8$ or 0.125 used in our degree-independent simulations, it is high if digest tables are calibrated with respect to router degree. Luckily, by increasing the number of digesting functions, Bloom filters are able to achieve significantly lower false-positive rates with slight decreases in memory efficiency. For instance, a false-positive rate of $P = 0.00314$, which corresponds to our degree-dependent simulation, $P = p/d$, with $p = 1/8$ for routers with as many as 40 neighbors, can be achieved using 8 digesting functions, with a memory efficiency factor of only 0.083—slightly less than half what we suggest.

SPIE's memory needs are determined by the number of packets processed. Hence, we consider an average-sized packet of approximately 1000 bits[4], and describe link speeds in terms of packets per second. We combine this with the Bloom filter efficiency factor of 0.2 from above to compute a rule of thumb: SPIE requires roughly 0.5% of the total link capacity in digest table storage. For a typical low-end router with four OC-3 links, this results in roughly 23MB of storage. On the very high end, a core router with 32 OC-192 links has a maximum capacity of about 320Mpkts/sec which would require roughly 1.6Gb/sec of digest table memory or 12GB for one minute's worth of storage. In practice, however, the size of a digest table will be limited by the type of memory required.

[4]This may in fact be a significant under-estimate. Recent studies have found the mean packet size has grown to over 400 bytes in many instances [8], [27]. The corresponding decrease in packet arrival rate eases the load on SPIE's digest tables.

B.2 Access rates

Size is not the only memory consideration, however—access times turn out to be equally important. Packets must be recorded in the digest table at a rate commensurate with their arrival. Even given an optimistic DRAM cycle time of 50ns per read-modify-write cycle, routers processing more than 20Mpkts/sec (roughly 2 OC-192 links, or 8 OC-48s) require an SRAM digest table. Current technology places pragmatic limits on SRAM size when operating at very high access rates. The increased power consumption, heat, and cost make it impractical to spread digest tables across more than a few SRAM chips. Hence, an entire minute's worth of traffic can only be stored in one digest table at low link speeds. Higher speed routers must page digest tables to SDRAM in order to store a minute's worth of digests as described in section VI. Given the unavoidable need for a two-tier digest architecture, the best choice of digest table size is likely dictated by pragmatic concerns, and using a single 16Mb SRAM avoids the timing problems inherent in grouping chips into one memory bank.

One way to decrease the update rate is to maintain separate digest tables for each input port. Unfortunately, since the input and output ports for an arbitrary packet are uncorrelated in general, this can complicate the query process. It may be especially problematic if the digest tables are not time synchronized across ports. In certain situations, however, the ability to isolate a specific input port may provide an additional benefit of reducing the number of upstream neighbors that need to be queried. Unfortunately, the ring and bus topologies common at many peering points force routers to have many neighbors on the same input port. The benefits of input port isolation are significantly reduced in such configurations, and are likely not worth the additional complexity.

In some border cases, it may be more practical to use a larger amount of slower memory and reduce the number of memory accesses required per packet, allowing DRAM to be used instead of SRAM, for example. This is especially true when considering cached-based memory architectures where access locality becomes an issue. In such cases, packet digests could be recorded in a hash table of b-bit values and collisions managed with open-addressed linear probing. If this table is never allowed to fill up, then it admits only false positives, and no false negatives, just like a Bloom filter. The false-positive rate of such a data structure is given by [28]

$$P = 1 - e^{-n/2^b}. \qquad (3)$$

Consider constructing a hash table intended to record n packet digests using $1.44n$ b-bit entries, requiring $m' = 1.44nb$ bits. Such a table is less than 70% full, hence, each packet insertion takes only $\sim$2 memory accesses in expectation [28, sec. 6.4, table 4]. Solving equation 3 for b, and substituting into the above equation, we see the memory required for a particular false-positive rate P while storing n packets is given by

$$m' = 1.44n \cdot \log(-n/\ln(1-P)).$$

When P is much smaller than 1, $\ln(1-P)$ is approximated by $-P$. Hence,

$$m' = 1.44n \cdot \log(n/P). \tag{4}$$

Combining equations 2 and 4, the additional cost of using a hash table instead of a Bloom filter, in terms of increased memory consumption, is a factor of (for small values of P)

$$m'/m = 1 + \log n/\log(1/P) = 1 + \log_{1/P}(n). \tag{5}$$

For slower routers with many neighbors (and therefore small P), the decrease in number and improved locality of memory accesses may outweigh the additional storage requirements of a hash table.

C. Timing uncertainties

For routers with a single OC-192 link, a 16Mb SRAM would hold roughly 10ms of traffic data; hence, the history buffer would store 6,000 individual digest tables. This observation gives rise to another important issue: imperfect timing may cause SPIE to need to examine multiple packet digests at a particular router. The more digests that must be considered, the greater the chance of false positives, so it is advantageous to make the digest tables as large as possible (within practical hardware limits). For reasonable link speeds, the memory access time becomes slow enough that SDRAM can be used which, using current technology, would allow 256Mb digest tables, with a capacity of roughly 50Mpkts.

It may be the case that the approximate packet service time cannot be confined to an interval covered by one digest table. In that case, we expect the false-positive rate to increase linearly with the number of digest tables examined. For high-speed routers, it is especially important to maintain precise timing synchronization between adjacent routers. We have not yet examined the impact of typical NTP clock skew on SPIE's performance, but believe synchronization can be maintained to within a small number of digesting intervals, not significantly impacting our false-positive rate.

VIII. Discussion

We believe there are three main areas that affect the practicality of SPIE. We examine several issues relating to deployment, vulnerability, and transform handling below.

A. Deployment

SPIE's usefulness increases greatly with widespread deployment because SPIE can only construct an attack graph for that portion of the packet's path within the SPIE domain. Within a particular ISP, however, it is likely that DGAs need not be deployed at every router. If a particular region of the network can be identified as transit-only, meaning no traffic originates within the region, and further, that no transforms are computed in the region, then the region need only be instrumented at the edges. Since all packets leaving the region are guaranteed to have entered the region, a traceback can consider the entire region as a single router without any loss of precision or reliability. When considering the network topology, the SCAR could simply collapse all the region's edge routers into one virtual router, and consider the virtual router's neighbors to be the set of all routers bordering the region.

Between ISPs, however, the situation is significantly more complicated. It is likely that independent ISPs may lack sufficient levels of technical or political cooperation to unite their SPIE infrastructures. Hence, regardless of the degree of deployment within adjacent ISPs, many ISPs will prefer to have their own STM responsible for all queries within their network. In such a case, one ISP's STM must be granted the authority to issue queries to adjacent ISPs' STMs in order to complete the traceback.

B. Vulnerabilities

SPIE's vulnerabilities can be divided into three distinct classes; we discuss each separately below.

B.1 Denial of service

Traceback operations will often be requested when the network is unstable (likely due to the attack that triggered the traceback); SPIE communications must succeed in a timely fashion even in the face of network congestion and instability. If SPIE traffic is not properly insulated from normal network traffic, SPIE may be unable to complete a traceback during periods of network congestion or routing failures. The best solution is to provide SPIE with an out-of-band channel, possibly through either physically or logically separate (e.g., ATM VCs) links. Even without private channels, it is still possible to ensure successful transmission by granting sufficient priority and configuring static routes for SPIE traffic.

B.2 Flow amplification

SPIE is designed to trace any distinct IP packet to its source(s). It does not, however, concern itself with the multiplicity of any particular packet. It is possible to exploit this fact to launch an "amplification" denial-of-service attack that SPIE alone is not able to isolate. Specifically, a router or host cannot surreptitiously insert a new, distinct packet into a SPIE-enabled network. It may, however, duplicate packets already in the network without detection, effectively amplifying the size of a traffic flow. In particular, a router $\mathcal{R}$ on the path between two hosts A and B may duplicate all packets going from A to B in an attempt to overwhelm downstream resources, including any routers and network links on the path from $\mathcal{R}$ to B, and even B itself.

The usefulness of such an attack is limited by the requirement that $\mathcal{R}$ lie on the path between A and B. Furthermore, duplicate packets are only undetectable if they fall within the same digest table page. Duplicate packets inserted significantly after the original packet will likely fall into a later digest table page on some downstream router, and therefore be detected as a distinct, later packet. Similarly, large numbers of duplicate packets would become apparent even to extremely simplistic network monitoring tools. Hence, an attacker likely can only increase the size of an individual flow by a small factor.

A naive attacker might attempt to increase the attack's effectiveness by amplifying a large number of flows destined to the same destination. This serves only to help isolate the attacker's location, however. If packets from several of the amplified flows are traced using SPIE, and their attack paths compared, the attacker must lie on the shared portion of the paths. As the number of flows amplified by the attacker grows, the portion of the path shared by all attack paths will converge to the path between the attacker and the destination, effectively identifying the rogue source $\mathcal{R}$.

B.3 Information leakage

In the normal course of operation, SPIE requires a querying intrusion detection system to submit the packet it wishes to trace. This obviously provides information to the entity administering SPIE about traffic a particular party finds interesting. In some rare cases, a querying party may not wish to leak such information by exposing the content of the packet, yet still wish to employ SPIE. In such a case, it might be possible to support call-backs from SCARs which would provide the querying intrusion detection system with the applicable digesting function and transformation information and ask it to do actual digesting. This is an expensive operation, but the existence of such a case implies the querying intrusion detection system has grave cause for concern in the first place and is likely willing to dedicate a great deal of resources to the traceback.

C. Transformations

Finally, transformations raise several additional issues, some related to performance, others to policy. In particular, assuming that packet transformations represent a small percentage of the overall IP traffic traversing a router, an efficient SPIE implementation can easily handle the resource requirements of logging transformation information. Attackers, though, may view packet transformations as a method of denial-of-service attack on SPIE. The number of transformations that are recorded during a given time interval is bounded by the rate at which the router is able to process the packet transformations. Therefore, SPIE aims to handle packet transformations at a rate equal or greater than the router. As a result, the router rather than SPIE is the bottleneck in processing packet transformations. This task is made easier when one realizes that the vast majority of transformations occur only at low-to-medium speed routers. Sophisticated transformations such as tunneling, NAT, and the like are typically done at customer premises equipment. Further, many ISPs turn off standard transformation handing, often even ICMP processing, at their core routers.

IX. CONCLUSION & FUTURE WORK

Developing a traceback system that can trace a single packet has long been viewed as impractical due to the tremendous storage requirements of saving packet data and the increased eavesdropping risks the packet logs posed. We believe that SPIE's key contribution is to demonstrate that single packet tracing is feasible. SPIE has low storage requirements and does not aid in eavesdropping. Furthermore, SPIE is a complete, practical system. It deals with the complex problem of transformations and can be implemented in high-speed routers (often a problem for proposed tracing schemes).

The most pressing challenges for SPIE are increasing the window of time in which a packet may be successfully traced and reducing the amount of information that must be stored for transformation handling. One possible way to extend the length of time queries can be conducted without linearly increasing the memory requirements is by relaxing the set of packets that can be traced. In particular, SPIE can support traceback of large packet flows for longer periods of time in a fashion similar to probabilistic marking schemes—rather than discard packet digests as they expire, discard them probabilistically as they age. For large packet flows, odds are quite high some constituent packet will remain traceable for longer periods of time.

ACKNOWLEDGEMENTS

Geva Patz assisted with the UNIX DGA prototype, and Alex Colvin helped design the SPIE messaging protocols. Chuck Blake pointed out the advantages of linear probing at high speeds. SPIE's vulnerability to traffic amplification was first noted by Dina Katabi. We thank Hari Balakrishnan and the anonymous reviewers for helpful feedback on earlier drafts.

REFERENCES

[1] Microsoft Corporation, "Stop 0A in tcpip.sys when receiving out of band (OOB) data," `http://support.microsoft.com/support/kb/articles/Q143/4/78.asp`.
[2] Paul Ferguson and Daniel Senie, "Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing," RFC 2267, IETF, Jan. 1998.
[3] Burton H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of ACM*, vol. 13, no. 7, pp. 422–426, July 1970.
[4] Vern Paxson, "End-to-end Internet path dynamics," *ACM Trans. on Networking*, vol. 7, no. 3, pp. 277–292, 1999.
[5] Colleen Shannon, David Moore, and K. Claffy, "Characteristics of fragmented IP traffic on Internet links," in *RIPE Workshop on Passive and Active Measurements*, Apr. 2001.
[6] Vern Paxson, "An analysis of using reflectors for distributed denial-of-service attacks," *ACM Comp. Comm. Review*, vol. 31, no. 3, 2001.
[7] Fred Baker, "Requirements for IP version 4 routers," RFC 1812, IETF, June 1995.
[8] Sean McCreary and K. Claffy, "Trends in wide area IP traffic patterns: A view from Ames Internet exchange," in *ITC Specialist Seminar on IP Traffic Modeling, Measurement and Management*, 2000.
[9] Hal Burch and Bill Cheswick, "Tracing anonymous packets to their approximate source," in *Proc. USENIX LISA '00*, Dec. 2000.
[10] Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson, "Network support for IP traceback," *ACM/IEEE Trans. on Networking*, vol. 9, no. 3, pp. 226–239, June 2001.
[11] Steven M. Bellovin, Marcus Leech, and Tom Taylor, "ICMP traceback messages," Internet Draft, IETF, Oct. 2001, `draft-ietf-itrace-01.txt` (work in progress).
[12] Dawn Xiaodong Song and Adrian Perrig, "Advanced and authenticated marking schemes for IP traceback," in *Proc. IEEE Infocom '01*, Apr. 2001.
[13] Allison Mankin, Dan Massey, Chien-Long Wu, S. Felix Wu, and Lixia Zhang, "On design and evaluation of "intention-driven" ICMP traceback," in *Proc. IEEE International Conference on Computer Communications and Networks*, Oct. 2001.
[14] Glenn Sager, "Security fun with OCxmon and cflowd," Internet 2 Working Group Meeting, Nov. 1998, `http://www.caida.org/projects/NGI/content/security/1198`.
[15] Dan Schnackenberg, Kelly Djahandari, and Dan Sterne, "Infrastructure for intrusion detection and response," in *Proc. First DARPA Information Survivability Conference and Exposition*, Jan. 2000.

[16] Robert Stone, "CenterTrack: An IP overlay network for tracking DoS floods," in *Proc. USENIX Security Symposium '00*, Aug. 2000.

[17] N. G. Duffield and M. Grossglauser, "Trajectory sampling for direct traffic observation," in *Proc. ACM SIGCOMM '00*, Aug. 2000, pp. 271–282.

[18] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *ACM/IEEE Trans. on Networking*, vol. 8, no. 3, pp. 281–293, 2000.

[19] L. Carter and M. Wegman, "Universal classes of hash functions," *Journal of Computer and System Sciences*, pp. 143–154, 1979.

[20] J. Black, S. Halevi, J. Krawczyk, T. Krovetz, and P. Rogaway, "UMAC: fast and secure message authentication," in *Proc. Advances in Cryptology — CRYPTO '99*, Aug. 1999, pp. 216–233.

[21] S. Halevi and H. Krawczyk, "MMH: Software message authentication in the Gbit/second rates," in *Proc. 4th Workshop on Fast Software Encryption*, 1997, pp. 172–189.

[22] H. Krawczyk, "LFSR-Based hashing and authentication," in *Proc. Advances in Cryptology — CRYPTO '94*, Aug. 1994, pp. 129–139.

[23] John Postel, "Internet Protocol," RFC 791, IETF, Sept. 1981.

[24] John Postel, "Internet Control Message Protocol," RFC 792, IETF, Sept. 1981.

[25] Ron Rivest, "The MD5 message-digest algorithm," RFC 1321, IETF, Apr. 1992.

[26] Luis A. Sanchez, Walter C. Milliken, Alex C. Snoeren, Fabrice Tchakountio, Christine E. Jones, Stephen T. Kent, Craig Partridge, and W. Timothy Strayer, "Hardware support for a hash-based IP traceback," in *Proc. Second DARPA Information Survivability Conference and Exposition*, June 2001, vol. 2, pp. 146–152.

[27] Chuck Fraleigh, Christophe Diot, Bryan Lyles, Sue Moon, Philippe Owezarski, Dina Papagiannaki, and Fouad Tobagi, "Design and deployment of a passive monitoring infrastructure," in *RIPE Workshop on Passive and Active Measurements*, Apr. 2001.

[28] Donald E. Knuth, *The Art of Computer Programming*, vol. 3, Addison-Wesley, Reading, Massachusetts, 2nd edition, 1998.

Alex C. Snoeren (S'00/ACM S'99) received the B.S. degrees in computer science and applied mathematics from the Georgia Institute of Technology, Atlanta, GA, in 1996 and 1997, respectively, and the M.S. degree in computer science in 1997. He is currently pursuing the Ph.D. degree in computer science at the Massachusetts Institute of Technology, Cambridge, MA. He is also a Scientist with the Internetworking Research Department, BBN Technologies, Cambridge, MA. His research interests include networking, distributed systems, and mobile computing.

Craig Partridge (M'88—SM'91—F'99/ACM) received the A.B., S.M., and Ph.D. degrees from Harvard University, Cambridge, MA. He is a Chief Scientist with BBN Technologies, Cambridge, MA, where he does research on various aspects of internetworking. He is chair of ACM SIGCOMM and the former editor in chief of IEEE NETWORK MAGAZINE and *ACM Computer Communication Review*. He is co-consulting editor of the Addison-Wesley Professional Computing Series.

Luis A. Sanchez received the B.S. degree in electrical engineering from the University of Puerto Rico, Mayaguez campus, in 1989 and the M.S. degree in electrical engineering from Boston University in 1994. He was the lead architect of Site Patrol v3.0, BBN Planet's first managed VPN service, before moving from BBN Planet to BBN Technologies. He left BBN in 2001 to join Megisto Systems as their Director of Security and Policy architecture. He co-chairs the IPsec Policy Working Group of the Internet Engineering Task Force.

Christine E. Jones received the B.S. degree in computer science from Whitworth College, Spokane, WA in 1998, and the M.S. degree in computer science from Washington State University, Pullman, WA in 2000. She joined BBN Technologies, Cambridge, MA in 2000 as a Scientist in the Internetworking Research Department. Her research interests include network security and wireless networks.

Fabrice Tchakountio (M'00) received the B.S degree in computer science from the Swiss Federal Institute of Technology, Lausanne, Switzerland in 1997 and the M.S degree in computer science from Eurecom Institute, Sophia Antipolis, France in 1998. He is a Scientist at BBN Technologies, Cambridge, MA, where he has been involved in the design and implementation of clustering algorithms for very large networks and predictive techniques for highly mobile network systems.

Beverly Schwartz (ACM'00) received the B.S. degree in electrical engineering from Tufts University, Somerville, MA, in 1985, and the S.M. degree in computer science from Harvard University, Cambridge, MA, in 1989. She is a Scientist at BBN Technologies, Cambridge, MA, in the Internetworking Research Department where she works on software solutions for IP traceback.

Stephen T. Kent (ACM) received the B.S. degree in mathematics from Loyola University, New Orleans, and the S.M., E.E., and Ph.D. degrees in computer science from the Massachusetts Institute of Technology. An ACM Fellow, he is the Chief Scientist for Information Security for BBN Technologies, Cambridge, MA, where he has engaged in information security R&D for over 20 years. His most recent work focuses on public-key certification infrastructures for government and commercial applications, security for Internet routing, and high assurance cryptographic modules. He has served on the Internet Architecture Board and chaired the Privacy and Security Research Group of the IRTF, both for over a decade. He currently co-chairs the Public Key Infrastructure Working Group of the IETF and is a member of the editorial board of the *Journal of Computer Security*.

W. Timothy Strayer (S'88—M'91—SM'98/ACM'88) received the B.S. degree in mathematics and computer science from Emory University, Atlanta, GA, in 1985 and the M.S. and Ph.D. degrees in computer science from the University of Virginia, Charlottesville, VA, in 1988 and 1992, respectively. He joined BBN Technologies, Cambridge, MA, in 1997, where he is a Division Scientist in the Internetworking Research Department. His research interests include transport protocols, active networks, satellite packet switching, virtual private networks, and routing systems. He is an author of *Virtual Private Networks: Technologies and Solutions* (Addison-Wesley).