An Empirical Analysis of Target-Resident DoS Filters

(Extended Abstract)

Michael Collins[*] Michael K. Reiter[†]

Abstract

*Numerous techniques have been proposed by which an end-system, subjected to a denial-of-service flood, filters the offending traffic. In this paper, we provide an empirical analysis of several such proposals, using traffic recorded at the border of a large network and including real DoS traffic. We focus our analysis on four filtering techniques, two based on the addresses from which the victim server typically receives traffic (*static clustering *and* network-aware clustering*), and two based on coarse indications of the path each packet traverses (*hop-count filtering *and* path identifiers*). Our analysis reveals challenges facing the proposed techniques in practice, and the implications of these issues for effective filtering. In addition, we compare techniques on equal footing, by evaluating the performance of one scheme under assumptions made by another. We conclude with an interpretation of the results and suggestions for further analysis.*

1. Introduction

Denial-of-service (DoS) attacks are among the most prominent types of attack on the internet today, and those that overwhelm a victim by flooding it with large numbers of packets constitute a substantial fraction of all DoS attacks. Today, such DoS attacks are conducted by large-scale networked applications capable of flooding a victim from thousands of compromised computers. In addition to the number of attacking computers, DoS attacks vary on a number of axes (e.g., see [19, 11]). One axis of interest here is whether traffic is *spoofed* or not, i.e., whether the source addresses of attack packets bear the correct IP address of the attacking machine. Spoofed traffic hinders the ability to identify the originating hosts—a problem for which "traceback" has been widely studied to address (e.g., [4, 25, 17, 1, 7, 26]). However, spoofing is of less utility when the attack is launched indirectly through compromised computers, since these computers do not expose the location of the original attacker, and can interfere with the attack traffic reaching its target due to increased deployment of ingress filtering [9].

Given the inertia of the deployed network infrastructure, several proposed DoS defenses have advocated a victim-centered approach in which the victim host (or an ingress router at the border of the victim's network) filters DoS traffic, with no or small change to the networking infrastructure itself. Here we focus on two classes of such techniques. In the first, which we call *address based*, the victim profiles the source IP addresses of traffic it receives under normal conditions to produce a model of "expected" traffic. It then applies this model to the IP addresses of incoming traffic during times of abnormally heavy load or otherwise anomalous traffic characteristics (which may indicate a DoS attack) to filter out packets with addresses not consistent with the model [14]. In the second, which we call *path based*, the victim gleans from each packet an indicator of the path the packet traversed, either the length (hop count) of the path [13] or a "path marker" that is created by the routers on the path to the victim [31], and filters traffic based on whether this indicator is consistent with a previously formed model of either the network itself or of paths that attackers are known to use.

In this paper we undertake an empirical analysis of these filters, based on data collected at the border of a large (greater than one million host) network. Data was collected in the form of NetFlow records from multiple border routers; since the routers cover all known interfaces between the client network and the internet proper, the collection system provides access to all incoming traffic. In the course of collecting this data, we have recorded several DoS attacks, both spoofed and non-spoofed, *in situ*. This permits us to analyze the traffic targeted at the victim before, dur-

[*] CERT Network Situational Awareness and Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA; mcollins@cert.org. CERT is part of the Carnegie Mellon Software Engineering Institute, sponsored by the U.S. Department of Defense.

[†] Department of Electrical and Computer Engineering, Department of Computer Science, and CyLab, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA; reiter@cmu.edu

ing, and after the DoS and, in particular, to simulate the behavior of the aforementioned proposed filtering techniques against these attacks.

In this extended abstract, we report on this analysis using a representative DoS attack. In an effort to generalize somewhat from this one attack, however, we simulate this attack against ten different servers for which we have records of normal traffic (one of which is the actually attacked server), to determine the filters' abilities to distinguish attack addresses from normal addresses for each of these servers. Moreover, since path-based filters are sensitive to the topology of routes to the attacked server, we also evaluate the attack (and normal) traffic as it would have been "seen" at each of these ten servers, had it traveled to the server over one of 26 different, real route topologies. We thus believe that our analysis offers insight beyond the behavior of these filters against this particular attack on one server. Still, we caution the reader from inferring too much from our study: as is any empirical study, ours is influenced by the particulars of the environment in which it is conducted. For this reason, we intend to replicate this analysis to multiple attacks in the full version of this paper.

Throughout this paper, the primary measures by which we quantify the efficacy of a filtering scheme are the false positive rate (the percentage of normal addresses from which traffic is filtered out) and the false negative rate (the percentage of attacker addresses from which traffic is accepted, or in a spoofed attack, the percentage of such traffic accepted). We use these measures to clarify the impact of several factors on the efficacy of these filtering techniques, many of which are not adequately explored in previous work. For example, each of these filtering techniques requires a learning period in which normal or attack traffic is modeled. We evaluate the impact of this learning period on the effectiveness of these filtering techniques. In addition, different techniques employ different assumptions during their learning phases, e.g., that it is possible to identify a packet as an attack packet based on its contents. A component of our analysis is to evaluate the effect of such an assumption even for techniques that did not previously employ it, thereby comparing these techniques on equal footing. This analysis leads us to preliminary comparison of these approaches and recommendations for further study.

The remainder of this paper is structured as follows. We survey related work in Section 2, and we describe the filtering techniques that we evaluate in Section 3. We describe our data set in Section 4. Section 5 constitutes the main portion of our analysis. We interpret our results in Section 6, and describe future work in Section 7.

2. Related Work

The dearth of widely accessible network-level traces that are appropriate for evaluating DoS filters has forced most analyses to simulate attacker locations and traffic characteristics artificially (e.g., as in [31, 13]), or to utilize less suited data sources (e.g., HTTP logs and worm traffic, as in [14]). To our knowledge, our study is the first direct comparison of target-resident DoS filters in which they are trained on recorded network traffic and tested against a recorded DoS event. The use of a recorded attack frees our analysis from assumptions about traffic characteristics or attacker locations that are often required in studies lacking appropriate data.

Outside the focus of evaluating filtering techniques, though within the related realm of characterizing network traffic anomalies and DoS attacks, a number of studies have been performed utilizing recorded data sets. Moore et al. utilize "backscatter" resulting from spoofed attacks to evaluate the prevalence of these attacks and to characterize them on several axes [20]. Barford et al. utilize NetFlow records (as we do, see Section 4) collected on the University of Wisconsin–Madison network to provide statistical descriptions of outages, flash crowds, DoS attacks and measurement failures [2, 3]. Hussain et al. utilize a large data set capturing numerous attacks to characterize and classify DoS attacks, and to explain the reasons underlying their behavior [11]. We reiterate that our goal is different from these studies: we strive to evaluate filtering techniques on representative DoS attacks, rather than to characterize DoS attacks more broadly.

More distantly related are empirical studies of common-case network packet traffic (e.g., [5, 21, 22, 30]), in contrast to the anomalous events we consider here; of network fault detection methods, i.e., that focus on accurate detection of anomalies induced by failures (e.g., [15, 10, 29]), rather than filtering DoS attacks as we consider here; and of malicious network activity other than the DoS variety that we consider here (e.g., [28, 33]).

As a first effort at comparing techniques, our analysis focuses on a few techniques within a narrow segment of proposed DoS defenses, namely those that filter at the target with no change (or, in one case we consider, a small change) to routers within the network. We have not considered numerous, more ambitious approaches to filtering DoS traffic within the network (e.g., [18, 27, 12, 23]). It is possible that our results can form a baseline measure against which these more ambitious approaches can be compared.

3. Filtering Techniques

In this section we summarize the DoS filters that we evaluate. We describe these techniques in the context of IPv4 and use corresponding terminology, in particular the use of CIDR notation, i.e., $/x$ to denote the x high-order bits of IPv4 addresses. We note, however, that these approaches can extend to IPv6 and other networking regimes.

Each of these techniques operates on the basis of a *filtering attribute* that the target extracts from each packet it receives. Recall that we informally separate filtering techniques into *address based* and *path based*. Address based filtering utilizes only the source address of a packet to generate its filtering attribute value. The address based techniques that we explore in this work are the following:

Network-aware clusters (NAC) [14] NAC characterizes networks by grouping IP addresses into clusters. These clusters are not uniform size, but rather are derived from CIDR blocks determined by examining BGP tables [16]. More precisely, when a target receives a packet with source address s, it clusters this packet by examining BGP tables to determine the longest (most specific) prefix that matches s; i.e., this prefix is the attribute value of this packet.

Static clusters (SC) [16] SC is a simpler form of clustering discussed in [16], though the authors discard this technique because static clusters are not an accurate reflection of the administrative relationships between networks. However, it is the simplest of the techniques we consider. SC uses clusters defined by a fixed constant $x \in [1, 32]$. For example, if $x = 16$, then upon receiving a packet from source address 192.143.14.7, the corresponding cluster (attribute value) is $192.143.14.7/16 = 192.143.*.*$.

Path based techniques work on the premise that packets from the same network and sent to the same destination will typically travel the same path. As such, these techniques extract a (possibly coarse) indicator of the path taken by the packet and use this as the filtering attribute value. We consider the following path based filtering attributes:

Hop counts (HCF) [13] If packets from the same network and sent to the same destination travel the same path, then the distance (hops) the packets travel will be the same. HCF estimates the hop count of each packet it receives based on its time-to-live field (TTL). (The TTL permits only an estimation of the hop count, since initial TTL values differ across platforms.) The filtering attribute for this algorithm is the pair consisting of the /24 prefix of the received packet, and its hop count estimate.[1]

Path identifiers (PI) [31] This is the only technique we study that presumes changes to routers in the network. In this approach, each network router contributes to a fixed-size marker field (e.g., 16 bits) in the IP header of each packet that traverses the router. The router's contribution is b bits (e.g., $b = 1$ or $b = 2$) that are computed deterministically from a combination of the router's IP address and the previous router's address. The router inserts these bits into the marker field either in a location determined by the TTL value of the packet [31] or by shifting the current marker value and slotting this router's contribution in the low-order bits vacated by the shift [32]. As such, packets that traverse identically the same path will bear the same marker field upon receipt; this marker field is the filtering attribute. It is important to note that because of limited space for the marker field, which is created by co-opting portions of the IPv4 header, distinct paths can induce the same marker value. In particular, for a k-bit marker field, any path ending with the same k/b routers will bear the same marker.

Each of the filtering attributes described above are used as input to train a model of network traffic and then to filter based upon it. During *learning*, the target develops a model of network traffic based upon the attribute values of packets received. Afterward, the target filters traffic based upon this model and packets' attribute values, with the goals of eliminating as much DoS traffic as possible and accepting as much legitimate traffic as possible.

A coarse characterization of filters that we will find useful later is whether the learning algorithm learns on the basis of only legitimate (negative) traffic—which we call a *normalcy* learning algorithm—or if it presumes to be given labeled examples of both legitimate (negative) traffic and attack (positive) traffic—which we call an *attacker* learning algorithm. The distinction between normalcy and attacker learning has significant ramifications to both the feasibility of the learning algorithm and to the posture of the filtering algorithm. First, an attacker learning algorithm requires some means of correctly distinguishing between legitimate packets and attack packets, presumably via an automatic classifier if learning is to involve traffic of any significant quantity. So, in some measure an attacker learning algorithm already solves the filtering problem, and thus constitutes a powerful assumption. Second, an attacker learning algorithm permits the filter to deny a packet on the grounds that it possesses the same attribute value as attack packets seen during learning, provided that the attacker does not (as in a non-spoofed attack) or cannot (as in PI) forge attribute values. Though different filters were proposed using different types of learning—NAC, SC, and HCF presume normalcy learning; PI assumes attacker learning—one aspect of our analysis is to consider the performance of, e.g., NAC, against non-spoofed attacks when given the benefit of an attacker learner.

1 [13] considered finer-grained clustering of the IP address component of the attribute value, as well, though this will not be important to our analysis.

4. Attacks

In this section we describe the attack that we use to evaluate the DoS filtering techniques described in Section 3. We begin with an overview of the system we use to collect the traffic, and then describe the attack that we utilize for evaluation.

4.1. Attack collection system

Our data collection was performed using a system called SiLK (System for Internet-level Knowledge), developed by the CERT. SiLK is a data collection and analysis system for monitoring large volumes of network traffic. SiLK provides tools to analyze traffic over large networks (currently over 1 million nodes) and long periods of time (several months). By installing SiLK on a client network, we have been able to collect network traffic before, during and after attacks.

The SiLK system collects CISCO NetFlow records and converts them into a space-efficient format. NetFlow is a collection system developed by CISCO to capture flows, i.e., summaries of traffic consisting of packets closely grouped over time [6]. These flow records consist of a characteristic 5-tuple of (source IP address, destination IP address, protocol, source port, destination port[2]). CISCO NetFlow is a commercially available package for reporting this flow traffic using a well-defined data format.

While NetFlow is a traffic analysis tool, it provides features that make it desirable for security analysis. In particular, NetFlow records are collected at the router: by collecting flows it is possible to see all traffic crossing a network's border, including traffic which the router drops because of ACL violations and overflows. Flow analysis is used elsewhere for security purposes, e.g., [24].

SiLK optimizes NetFlow data for rapid analysis. The records are compressed to a more space-efficient form and then recorded in flat files that are accessed using a specialized collection of tools. Given the volume of data collected by SiLK (in excess of 60 GB and 300 million records per day on the network we monitor), traditional database solutions would require a far more expensive implementation to provide the same access speed.

DoS attacks are unpredictable; the best way to acquire data on a DoS attack is to instrument a sufficiently large network and lie in wait. SiLK's primary strength is its capacity to manage large volumes of traffic data over long periods of time, making a long wait feasible.

Protocol	UDP
Packet Size	34 bytes
Duration	6h, 1m
Bytes	22,009,778,272
Packets	687,805,571
Source addresses	3,102

Table 1. Traffic characteristics of attack

4.2. Attack data set

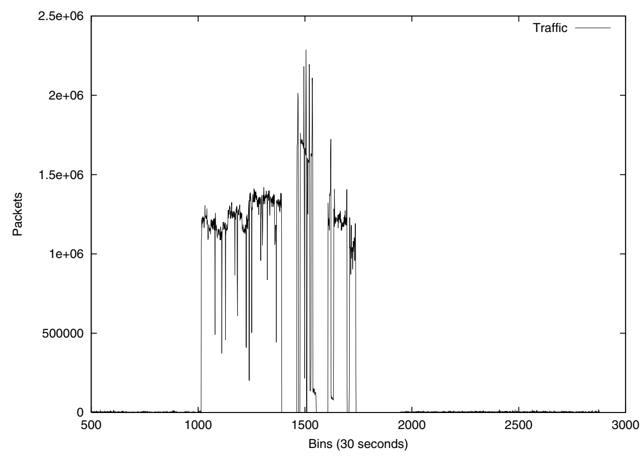
For the purposes of this paper, we discuss a representative DoS attack from the SiLK data. This attack targeted a web server, but the attack was not itself HTTP-specific and was not aimed at a specific service: it struck at a range of destination ports that are largely unrelated to specific services. This attack was aimed at either the IP stack or the network infrastructure surrounding the targeted machine. The attack was identified after the fact: analysts notified us of the basic characteristics of the event, and relevant traffic was then extracted from the SiLK data warehouse.

The basic characteristics of the attacks are given in Table 1, and progression of the attack over time is shown Figure 1. The attack packets originated from a limited set of IP addresses: throughout the attack, 3102 previously unseen addresses repeatedly sent 32-byte UDP packets at random destination ports. While it is impossible to say definitively whether these attacks were spoofed, there is evidence to suggest that the attacker addresses are authentic. First, the addresses all originated from used address spaces.[3] Second, the ephemeral port assignments for the attackers are consistent with single addresses: for each address, the port numbers increase linearly and then cycle back to an initial port.[4]

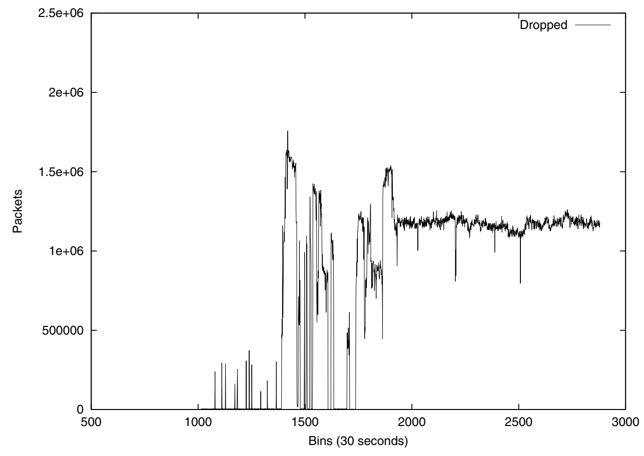
During the course of the attack, certain addresses appear and disappear from the set of attackers; see Figure 1(iii). We hypothesize three reasons for these changes: data lost by the router dropping reports (see below), user intervention to remove DoS attackers, and attacker intervention to add additional attackers. At this point, there appears to be no consistent pattern to why an address appears or disappears: when new addresses appear, address changes do not appear to be grouped uniformly together in time and addresses that stop do not appear to be from the same networks.

As shown in Figure 1(ii), approximately half of the traffic hitting the server was dropped by the router. This excessive drop rate is likely a result of the severity of the attack.

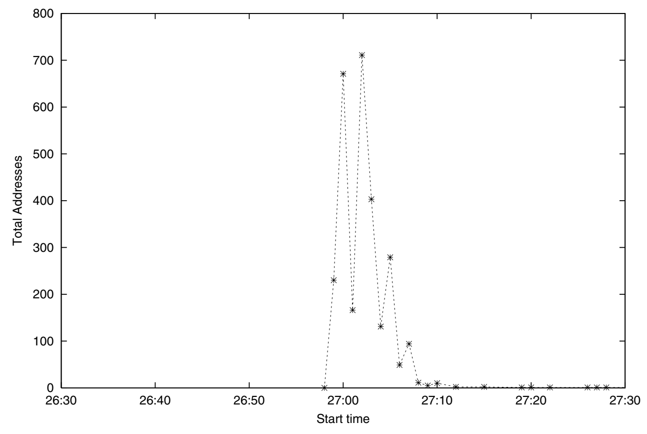
2 For ICMP traffic, type and code are stored in the destination port field.
3 Used addresses are those assigned an authority at `http://www.iana.org/assignments/ipv4-address-space`. At the time of this writing, roughly 120 of the 255 /8's available are listed as reserved or otherwise unroutable.
4 While the IP specification dictates that ephemeral port numbers should be assigned randomly, very few stack implementations actually do so.



(i) Packets per 30s



(ii) Packets dropped by router



(iii) New attackers per second

Figure 1. Attack

In addition, the heightened drop rate in the latter half of Figure 1(ii) is the result of an ACL modification to protect the target.

NetFlow reporting is a secondary process for routers, and so during stress, routers will naturally drop a certain percentage of NetFlow records before sending them for collection. As such, the information conveyed in Table 1 and Figure 1 does not represent the entirety of attack traffic. Approximately 8% of NetFlow records are lost every day off of the router adjacent to the target and, during the attack, the router drops a larger percentage, up to 17%, undoubtedly due to the impact of the attack. This implies that the volume, and possibly the sources, of the attack are underreported.

4.3. Network map reconstruction

The traffic records collected by SiLK provide IP addresses, times and volumes. However, for three of the techniques discussed (PI, NAC and HCF), we require information about the network in order to evaluate them. To generate this information, we used two forms of network maps.

4.3.1. Networks maps for path-based filters Evaluating PI requires the most detailed network information: an inventory of routers between the sources and the targets. To build this inventory, we constructed partial internet maps from the Skitter internet map data.[5] Skitter collects routing information by regularly issuing `traceroute` calls from sensors across the internet. From this information, we constructed a table of routes to each of 26 Skitter sensors and, in each evaluation run of a path-based filter, treated one of these sensors as the machine targeted in the attack. (Mapping routes to the network we monitor was not permissible.) As such, this experiment more precisely evaluates how a server at this sensor (rather than the actually attacked server) would fare under the attack.

HCF also uses route-specific information, but this information is less detailed than that used by PI: the length (hop count) of the route from the source to the target. To develop a table of lengths, we used the lengths of the paths derived from Skitter.

Since the exact IP addresses of the attackers in the attack are not in the Skitter dataset, we reduced the Skitter maps to a /24 resolution. More precisely, suppose we fix a Skitter sensor. For this sensor, each attacking IP address is treated as the node mapped by the Skitter sensor with the same /24 prefix as the attack IP address, if such a node exists. Even then, the Skitter map covered a very limited portion of the addresses contacting the server: e.g., on average, the route map to a Skitter sensor contained nodes with

5 `http://www.caida.org/tools/measurement/skitter`

IP addresses matching the /24 prefixes of 4.63% of the attack addresses, with a standard deviation of 2.33%. As a result of this address reduction, we used a reduced internet when evaluating PI and HCF. In this reduced internet, the only IP addresses available are ones found in the Skitter maps (using the /24 rule above). Addresses which do not appear in the map are not used. Furthermore, when spoofing traffic for evaluating HCF and PI, spoofed addresses are drawn only from the networks in the reduced internet, but otherwise are chosen uniformly at random.

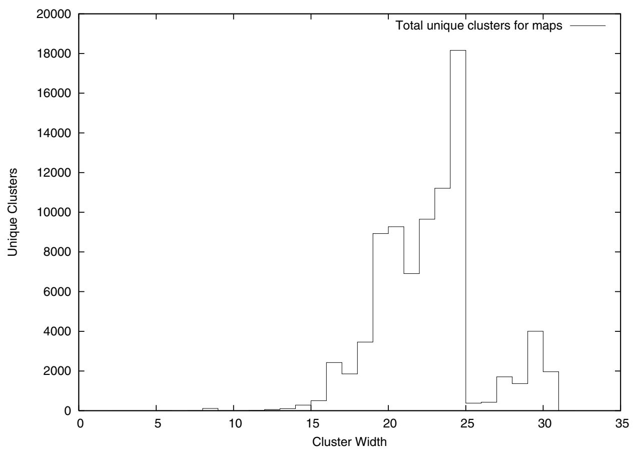


Figure 2. Cluster lengths in union of three BGP routing tables

4.3.2. Network maps for address-based filters NAC uses BGP tables to determine the filtering attribute value for a packet. Though NAC recommends drawing BGP tables from across the internet [16], we used a more limited set of tables for this evaluation, drawn from an ATT Canada route server[6], a Telus route server[7], and a CERFnet route server[8]. The information provided by the union of these tables is incomplete, providing routing prefixes of at least 16 bits for only roughly 10% of the total internet address space. Given the limits of this data, we implemented an approach where the cluster table was initialized with "default" static clusters to use if no network-aware cluster (routing prefix) was available. Figure 2 shows the distribution of cluster lengths in the tables; since the overwhelming majority of clusters are 16 bits or more, we initialized the network-aware cluster space using default 16 bit static clusters. Due to this approximation, we believe our results give a moderately pessimistic view of the performance of NAC.

6 `route-server.east.attcanada.com`
7 `route-views.on.bb.telus.com`
8 `route-server.cerf.net`

5. Evaluation

In this section we evaluate each of the filtering approaches described in Section 3 using the attack data described in Section 4. In addition to this attack data, our evaluation employs two weeks of normal traffic records for each of ten different HTTP servers (one of which is the server actually attacked). At a high level, in our evaluation we build a model of normal traffic (in normalcy learning) based upon thirteen days of normal traffic for a server, or a model of attack traffic (in attacker learning) based upon the attack data set. Once this model is built, we determine a false negative and false positive rate as described below. By averaging over the models constructed using the sets of normal server data from 10 different servers and, for path-based filters, over the 26 network maps corresponding to the 26 Skitter sensors (260 combinations in total), we obtain average false positive and false negative rates for each filter against the attack we consider. Since each scheme provides parameters that can be tuned, tuning these parameters in fact yields a curve of false positive rate versus false negative rate, that characterizes the points that can be achieved by varying these parameters.

In all of our evaluations, the false positive percentage for a server is the percentage of addresses in a day of normal traffic other than the thirteen used to train that server's model that would be discarded by the filter. The false negative percentage for a server in a non-spoofed attack is analogous, i.e., the percentage of attacker addresses in our attacker data set that the filter would permit to pass. In a spoofed attack, the false negative percentage is computed to be the percentage of spoofed packets that the filter would permit to pass, assuming that each attacking computer emitted traffic at the same rate and spoofed the source address of each packet by selecting it uniformly at random.

5.1. HCF

The filtering attribute of HCF is the pair consisting of the /24 prefix of the packet and its hop count, i.e., the length of the path it traversed. As described in [13], this hop count is an estimate only, though for the purposes of our analysis here, we ignore the potential for error in this value and assume that the target can exactly determine the hop count of the packet. We revisit this (generous) assumption below.

HCF assumes normalcy learning only. The filter drops packets for which the hop count does not match the hop count seen for the same /24 during learning. A more permissive version drops only packets for which the hop count differs from that seen for the same /24 by greater than ϵ, where $\epsilon = 1$ and $\epsilon = 2$ are considered by the authors [13].

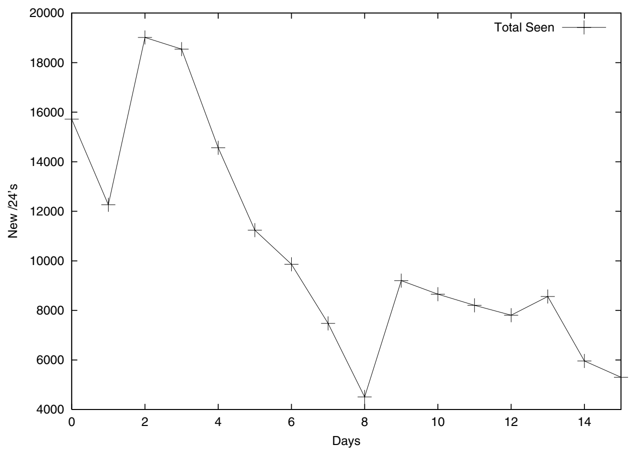
In our experiment, we trained the HCF learning algorithm on thirteen days of normal traffic. For the servers we considered, thirteen days of normal traffic (without map reduction) permitted the filter to observe only a relatively small portion of the internet. For example, Figure 3(i) demonstrates that in thirteen days, the actually attacked server was exposed to a maximum of 160,000 /24's, assuming no map reduction, and that the number of new /24's the target saw per day generally decreased. Since the internet consists of approximately 16 million /24 blocks, roughly 8 million of which are used, servers handling this much activity would require well over a year to see even half of the used internet space. We note that this learning time is an order of magnitude longer than that projected in the original HCF paper, which is roughly consistent with the thirteen days of training that we chose for our experiment.[9]

Such a learning period has dramatic implications for HCF. During a spoofed attack, the server typically sees a huge number of previously unseen addresses (/24s). HCF will drop such packets, leading to a very low false negative rate. In this way, however, HCF induces a significant false positive rate, dropping traffic from roughly 25% of normal addresses on average in our tests, even in the most permissive configuration recommended by the authors ($\epsilon = 2$); see Figure 3(ii). Here and throughout this paper, error bars show one standard deviation.

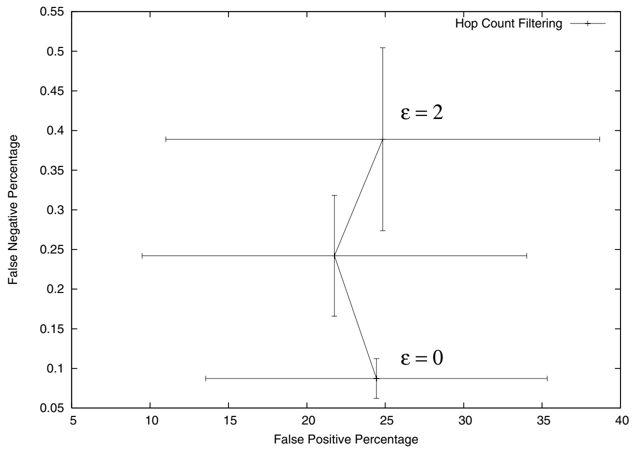
Figure 3(ii) also shows a false negative rate for HCF for $\epsilon \in \{0, 1, 2\}$. This false negative rate reflects the optimistic assumption, described above, that the filter can determine the accurate hop count of each packet. In reality, and as noted in [13], an attacker can intelligently modify the initial TTLs of the packets it sends to achieve roughly a 10% false negative rate, even without detailed knowledge of the network topology.

A final point about HCF regards the recommended algorithm to prevent an attacker from "poisoning" the hop count data during training with forged packets. The authors of HCF recommend that during learning, only hop counts associated with packets received on a completed TCP connection be used to learn the hop count of a new /24. In this way, forged TCP SYN packets will not poison the hop count data; presumably the attacker will be unable to complete the TCP handshake for a forged SYN. While reasonable, we comment that this approach is effective only for servers for which TCP connections are the dominant form of interaction. This is true for most, but certainly not all, types of servers; for example, DNS servers communicate with the majority of their clients using UDP.

9 "For a very busy site, a collection period of a few days could be sufficient, while for a lightly-loaded site, a few weeks might be more appropriate." [13, Section 5.2]



(i) New /24s seen per day at attacked server, without map reduction (example)



(ii) False negative vs. false positive rate

Figure 3. HCF Results

As a spoof detection mechanism, HCF does not defend against non-spoofed attacks, and so we did not evaluate it under such conditions.

5.2. PI

In PI, recall that the filtering attribute is a path marker field in the IP header that is determined as a function of the path the packet traversed. As noted in [31], PI improves when paths taken by attack packets intersect paths taken by legitimate packets as little as possible. Since paths tend to intersect as they aggregate close to the target, Yaar et al. recommend placing PI filters several hops away from the target, so as to minimize the effects of this aggregation on the

PI marks. This observation is consistent with the reduced internet maps we described in Section 4.3, and as such, we evaluate the PI algorithm assuming that the filter is placed the recommended three hops in front of the target.

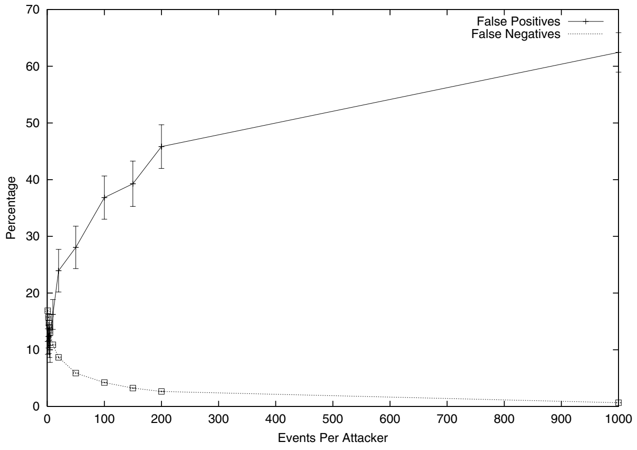
This improvement does come with a cost (also discussed in [31]), however, since this effectively reduces the distance of each attacker to the filter by three hops. For those attackers at a distance $d < k/b$ hops to the filter—where k is the bit length of the marker field and b is the bit length of each router's contribution to that field; we take these as $k = 16$ and $b = 2$ as in [31]—this means that $k/b - d$ bits of the marker will not be set by any router. These bits can thus be set by the attacker randomly, for example, and thereby induce collisions with the markers of legitimate packets. It is thus in an attacker's interest to initialize the marker field to a random value.

The effect of these collisions, unchecked, would be significant in our attacks. Figure 4(i) shows example false positive and negative rates for PI (though simplified; see below) after training on both attack traffic and normal traffic, labeled accordingly. Here, the error rates are plotted as a function of the number of attack packets seen from each attacking computer during learning. If attackers were to send packets during learning at the rate they send during the real attack, then receiving 1,000 packets per attacker computer would take a relatively short period of time, e.g., roughly 20 seconds. In addition, if the attackers are initiating heterogenously, as Figure 1(iii) implies, a filter would have to learn for at least that long before all attackers have been observed. Under these conditions, and assuming that markers associated with attacker traffic during training are rejected, the false positive and negative rates would result as in the right edge of Figure 4(i). Intuitively, the large false positive rate results from attackers inducing collisions with the markers of legitimate packets.

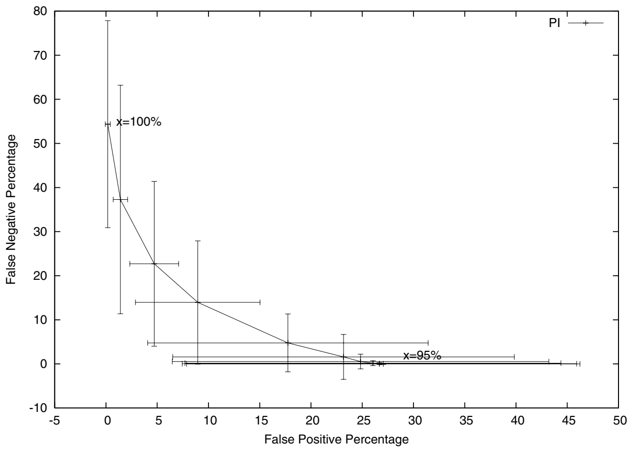
To compensate for collisions, PI permits relaxing its strategy for rejecting packets. In this strategy, a packet will be dropped iff it bears a mark for which at least $x\%$ of the traffic during learning was attacker traffic. That is, Figure 4(i) shows results for small x, though varying x between 95% and 100% yields the plot in Figure 4(ii). This plot shows false negative versus false positive results after training that includes 1000 packets from each compromised computer; i.e., this plot corresponds to the rightmost values in Figure 4(i), though as x is increased. As this graph shows, tuning x is of significant benefit for PI in our tests. A comparison of these results to other approaches will be given in Section 6.

5.3. NAC and SC

As described in Section 3, the filtering attribute of a packet utilized by NAC and SC is a "cluster" determined as a prefix of the source IP address of the packet. In NAC, this prefix is a CIDR block determined as the longest (most specific) BGP routing prefix for the IP address. In SC, this prefix is simply a fixed-length prefix, for which we consider /8, /16 and /24 in this section.



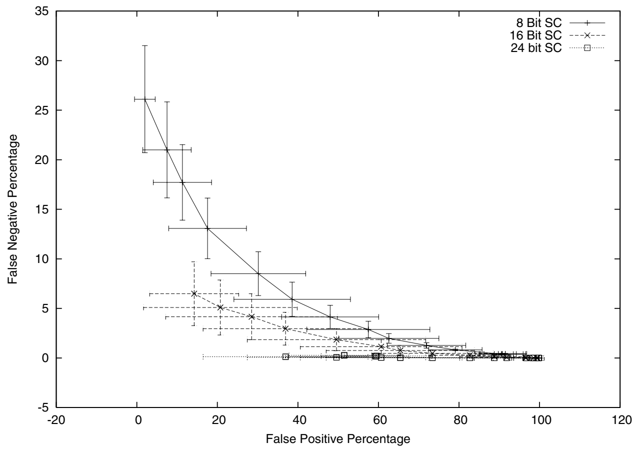
(i) Error rates as a function of learning (example), x small



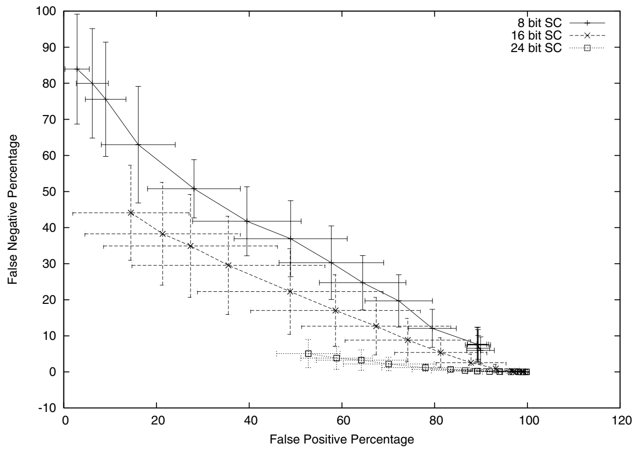
(ii) Error rates as x is varied

Figure 4. PI results

Learning for NAC and SC is envisioned in [14] to be normalcy learning, and so to test these algorithms under this assumption, we utilized our thirteen-day training set to train these approaches. After training, we employ the following filtering approach, based on a suggestion in [14]: let $v_1, v_2, \ldots$ denote the attribute values (clusters) sorted so that $\%(v_i) > \%(v_j)$ if $i < j$, where $\%(v_i)$ denotes the percentage of packets received during training that have attribute value v_i. Then, for a fixed percentage x, we say that

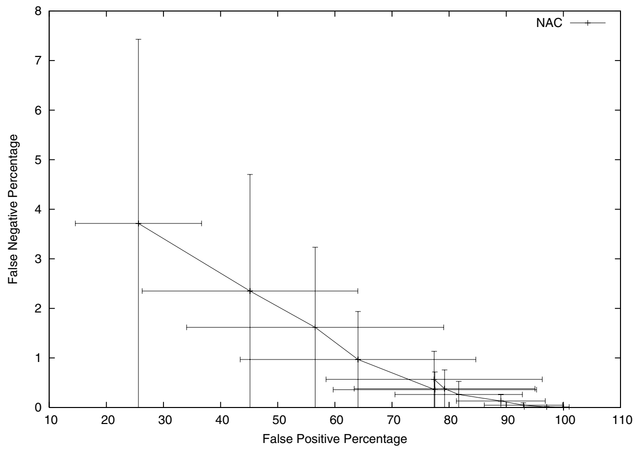


(i) Spoofed attack

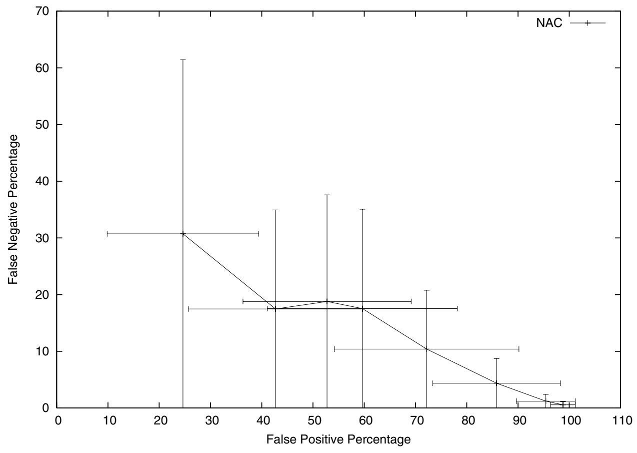


(ii) Non-spoofed attack

Figure 5. SC (normalcy learning)



(i) Spoofed attack



(ii) Non-spoofed attack

Figure 6. NAC (normalcy learning)

an attribute value (cluster) v_ℓ is *common* if

$$\ell \leq \arg\min_i \left(\sum_{j=1}^{i} \%(v_i) \geq x \right). \tag{1}$$

Intuitively, v_ℓ is *common* if the smallest set of attribute values that account for a total of $x\%$ of the learning packets contains value v_ℓ. During filtering (given a fixed percentage x), packets received bearing a common attribute value are accepted; others are rejected. We tested this algorithm for both spoofed attacks and non-spoofed ones. The percentage x was varied to exhibit its effect on false positive and false negative rates.

The performance of SC for 8-bit, 16-bit and 24-bit clustering is shown in Figure 5. Figure 5(i) shows performance against spoofed attacks, and Figure 5(ii) shows performance against nonspoofed attacks. The extreme performance of static clustering based on /24-bit clusters is at least partially due to the fact that the attacker addresses (in both the spoofed and non-spoofed cases) virtually never appeared in common clusters for the servers we considered, and so the false negative rate was very close to zero.

It is also interesting to compare static clustering to NAC, shown in Figure 6. There we can see that NAC performs very similarly to 16-bit SC in the ranges where both are plotted, probably owing to the fact that if the most specific BGP routing prefix in our routing tables that matched an incoming packet was of length less than 16 bits, then we still utilized the full 16 bits of the packet address as its attribute value. That said, NAC does slightly outperform 16-bit SC on average as x is increased. A plausible explanation is the

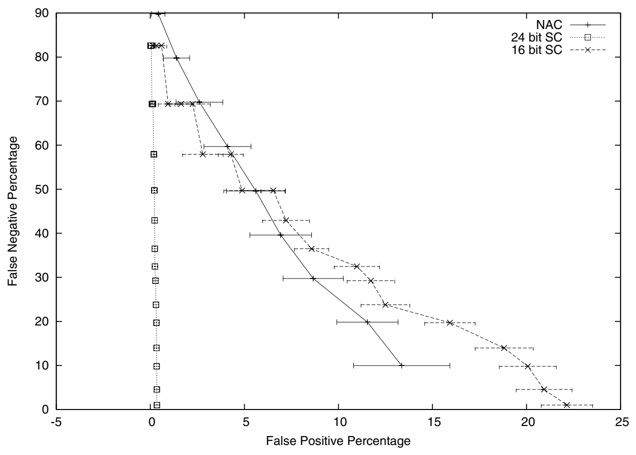


Figure 7. NAC, 16-bit SC, and 24-bit SC (attacker learning)

hypothesis that larger clusters (16-bit prefixes) tended to account for the largest portions of normal traffic in NAC, and as x was increased, smaller clusters (of prefix length greater than 16 bits) became common according to definition (1). In 16-bit SC, however, these clusters were represented more coarsely (still by 16-bit prefixes), thus permitting more traffic and a higher false negative rate. Presumably with a more precise set of routing tables, this difference would become more pronounced.

As discussed in Section 3, other approaches, notably PI, have been proposed for use with attacker learning. As a point of comparison, we also tested NAC, 16-bit SC and 24-bit SC under an attacker learning scenario. In this case, we utilized attacker and normal traffic (as in Section 5.2) to train the NAC and SC filters, with attack packets labeled as such. Now, for a fixed percentage x, we say that an attribute value (cluster) v_ℓ is *corrupt* if the smallest set of attribute values that account for a total of $x\%$ of the attacker packets contains value v_ℓ, or more precisely, if ℓ satisfies (1) where $\%(v_i)$ now is the percentage of attacker packets seen during training that bear attribute value v_i. During filtering (for a fixed percentage x), each packet bearing a corrupt attribute value is rejected; others are admitted.

The results of this test are shown in Figure 7. In this case, NAC performs slightly better than 16-bit SC in reasonable ranges for both false positives and false negatives. In addition, the availability of attacker learning induces a far smaller false positive rate for both NAC and SC than did normalcy learning. In the case of 24-bit SC, this difference is extreme. The performance of 24-bit SC is presumably due to the fact that, with filtering attributes of this level of granularity, attacker addresses almost never overlapped normal addresses for the server.

An alternative form of filtering after attacker learning would be to utilize a threshold as in PI. That is, a packet with a certain filter attribute value is kept only if attack packets bearing the same filter attribute value constitute below a threshold percentage of all traffic bearing that filter attribute value. We will report on this evaluation in the full paper.

6. Discussion

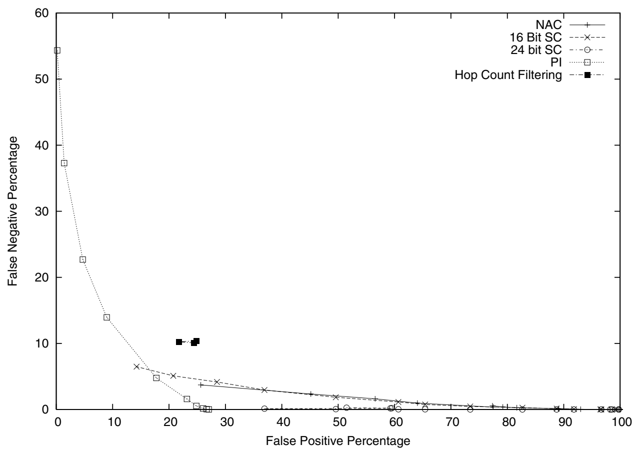
In the course of Section 5, several observations about factors that affect the performance of these various filters became apparent. A factor of some significance was the learning time afforded to each filter. For example, for the scenario we studied, a learning period adequate to support HCF was far longer than predicted by the authors of that technique. Another significant factor was the type of learning permitted, i.e., attacker learning as assumed by PI, or merely normalcy learning as assumed by other techniques.

A comparison of the various techniques is summarized in Figure 8, which is obtained by merging selected results from Section 5. For readability, we have omitted error bars from Figure 8, though we caution the reader that these error bars are important in determining whether there is a statistically significant difference between various approaches. Figure 8(i) summarizes filter effectiveness against spoofed attacks, and Figure 8(ii) summarizes filter effectiveness against non-spoofed attacks.

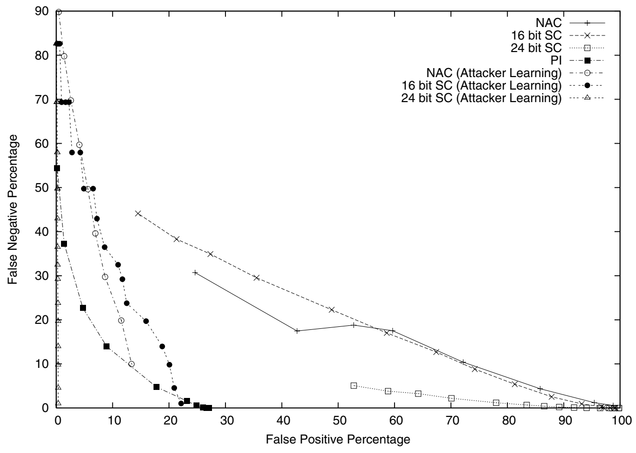
We first discuss Figure 8(i). Though HCF yielded very low false negative rates in our evaluations as shown in Figure 3, we remind the reader that our analysis there was performed under the optimistic assumption that the true hop count of each packet could be reliably determined; see Section 5.1. As already noted in [13], however, corrupted computers conducting spoofed attacks can manipulate hop count measurement (by manipulating initial TTL values) and thereby achieve roughly a 10% false negative rate. As such, in Figure 8(i) we have shifted the HCF curve upward to account for this attack. In light of this, the PI and 16-bit SC curves reach closest to the origin. For PI, this can be explained since PI marks are immune to spoofing. The success of 16-bit SC (and the similar performance of NAC) can presumably be attributed to the fact that spoofed addresses induce a very different address distribution from normalcy, and thus can be filtered relatively easily based on address alone.

The dramatic impact of attacker learning (versus normalcy learning) is a central conclusion from Figure 8(ii). The plot demonstrates that attacker learning in the case of non-spoofed attacks yields vastly superior performance for each of NAC and SC.

Finally, we again caution the reader from concluding too much from Figure 8. Though we believe these graphs con-



(i) Spoofed attacks



(ii) Non-spoofed attacks

Figure 8. Summary

tain useful commentary on the relative performance of these filtering techniques, our results are obviously dependent on the situation in which our data was recorded and on the particulars of our methodology. Further studies are required to confirm these results.

7. Future Work

We recognize that as an initial study of target-resident filtering techniques, our evaluation has several limitations. One is the small degree to which attacker addresses and normal addresses were represented in the network maps that we were able to gather for evaluating path-based filters; see Section 4.3.1. A second is the fact that our evaluation is based on a single attack event. Improving these aspects of our analysis are topics of ongoing work.

We have also left several parameters for future study. For example, for path based schemes, it is important to consider changes in network topology and their effect on the performance of these schemes. Specifically, as the network adapts to outages and load—possibly DoS-induced—the resulting changes to paths will affect filters such as HCF and PI.

Another topic for future study is motivated by Figure 1(iii), which shows the variation over time with which compromised computers entered this attack. A more pronounced variation was recently employed by the My-Doom viruses, each copy of which launched its DoS attack (against a web server) according to the *local* time on the computer on which it was residing. As such, computers in different time zones entered the attack at different global times. Such a prolonged, staged entry of attackers could pose challenges to learning algorithms.

The changes to routers that PI requires suggests another topic of work, namely evaluating the relative performance of this technique when only partially deployed. Our analysis here utilized the assumption of total deployment.

Finally, we intend to systematically evaluate combinations of filters that we have studied here. An example of such a combination is described in [32].

Acknowledgements

We are deeply grateful to Abraham Yaar for useful technical discussions.

References

[1] M. Adler. Tradeoffs in probabilistic packet marking. In *Proceedings of the 34th ACM Symposium on Theory of Computing*, pages 407–418, May 2002.

[2] P. Barford and D. Plonka. Characteristics of network traffic flow anomalies. In *Proceedings of the 1st ACM SIGCOMM Internet Measurement Workshop*, November 2001.

[3] P. Barford, J. Kline, D. Plonka and A. Ron. A signal analysis of network traffic anomalies. In *Proceedings of the 2nd ACM SIGCOMM Internet Measurement Workshop*, November 2002.

[4] S. M. Bellovin, M. Leech, and T. Taylor. ICMP traceback messages. Internet-draft: draft-ietf-itrace-01.txt, work in progress, October 2001.

[5] R. Cáceres. Measurement of wide-area internet traffic. Technical Report UCB/CSD 89/550, Computer Science Department, University fo California, Berkeley, 1989.

[6] K. C. Claffy, H.-W. Braun and G. C. Polyzos. A parameterizable methodology for internet traffic flow profiling. *IEEE Journal on Selected Areas in Communications* 13(8):1481–1494, 1995.

[7] D. Dean, M. Franklin and A. Stubblefield. An algebraic approach to IP traceback. *ACM Transactions on Information and System Security* 5(2):119–137, May 2002.

[8] S. Dietrich, N. Long, D. Dittrich. Analyzing Distributed Denial of Service Attack Tools: The Shaft Case. In *Proceedings 14th Systems Administration Conference*, LISA 2000.

[9] P. Ferguson and D. Senie. Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing. Request for Comments: 2267, the Internet Society, January 1998.

[10] C. Hood and C. Ji. Proactive network fault detection. In *Proceedings of IEEE INFOCOM '97*, April 1997.

[11] A. Hussain, J. Heidemann and C. Papadopoulos. A framework for classifying denial of service attacks. In *Proceedings of the 2003 ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (SIGCOMM), August 2003.

[12] J. Ioannidis and S. M. Bellovin. Implementing pushback: Router-based defense against DDoS attacks. In *Proceedings of the 2002 ISOC Symposium on Network and Distributed Security*, February 2002.

[13] C. Jin, H. Wang and K. G. Shin. Hop-count filtering: An effective defense against spoofed DDoS traffic. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, October 2003.

[14] J. Jung, B. Krishnamurthy and M. Rabinovich. Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites. In *Proceedings of the 11th International World Wide Web Conference*, May 2002.

[15] I. Katzela and M. Schwartz. Schemes for fault identification in communications networks. *IEEE/ACM Transactions on Networking* 3(6):753–764, December 1995.

[16] B. Krishnamurthy and J. Wang. On network-aware clustering of web clients. In *Proceedings of the 2000 ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (SIGCOMM), August 2000.

[17] A. Mankin, D. Massey, C. L. Wu, S. F. Wu, and L. Zhang. On design and evaluation of intention-driven ICMP traceback. In *Proceedings of the 2001 IEEE International Conference on Computer Communication and Networks*, October 2001.

[18] J. Mirkovic, G. Prier and P. Reiher. Attacking DDoS at the source. In *Proceedings of the 10th IEEE International Conference on Network Protocols*, November 2002.

[19] J. Mirkovic and P. Reiher. A taxonomy of DDoS attack and DDoS defense mechanisms. Available at `http://www.cs.ucla.edu/~sunshine/publications/ccr.pdf`.

[20] D. Moore, G. Voelker and S. Savage. Inferring internet denial-of-service activity. In *Proceedings of the 10th USENIX Security Symposium*, August 2001.

[21] V. Paxson. Fast, approximate synthesis of fractional gaussian noise for generating self-similar network traffic. *Computer Communications Review* 27(5):5–18, October 1997.

[22] V. Paxson. *Measurements and Analysis of End-to-End Internet Dynamics*. Ph.D. thesis, University of California–Berkeley, 1997.

[23] C. Papadopoulos, R. Lindell, J. Mehringer, A. Hussain, and R. Govindan. COSSACK: Coordinated suppression of simultaneous attacks. In *Proceedings of the 3rd DARPA Information Security Conference and Expo* (DISCEX III), April 2003.

[24] S. Romig, M. Fullmer and S. Ramachandran. Cisco flow logs and intrusion detection at the Ohio State University. *;login: – The Magazine of the USENIX Association*, September 1999. Available at `http://www.usenix.org/publications/login/1999-9/osu.html`.

[25] S. Savage, D. Wetherall, A. Karlin and T. Anderson. Network support for IP traceback. *IEEE/ACM Transactions on Networking* 9(3), June 2001.

[26] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, B. Schwartz, S. T. Kent, and W. T. Strayer. Single-packet IP traceback. *IEEE/ACM Transactions on Networking* 10(6), December 2002.

[27] M. Sung and J. Xu. IP traceback-based intelligent packet filtering: A novel technique for defending against internet DDoS attacks. In *Proceedings of the 10th IEEE International Conference on Network Protocols*, November 2002.

[28] J. Toelle and O. Niggemann. Supporting intrusion detection by graph clustering and graph drawing. In *Proceedings of the 3rd International Workshop on Recent Advances in Intrusion Detection*, October 2000.

[29] A. Ward, P. Glynn and K. Richardson. Internet service performance failure detection. *Performance Evaluation Review*, August 1998.

[30] W. Willinger, M. Taqqu, R. Sherman, and D. Wilson. Self-similarity through high-variability: Statistical analysis of Ethernet LAN traffic ant the source level. *IEEE/ACM Transactions on Networking* 5(1):71–86, February 1997.

[31] A. Yaar, A. Perrig and D. Song. Pi: A path identification mechanism to defend against DDoS attacks. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, May 2003.

[32] A. Yaar, A. Perrig and D. Song. StackPi: A new defensive mechanism against IP spoofing and DDoS attacks. Technical Report CMU-CS-02-208, Carnegie Mellon University, February 2003.

[33] N. Ye. A markov chain model of temporal behavior for anomaly detection. In *Proceedings of the Workshop on Information Assurance and Security*, June 2000.