RESECT: Self-Learning Traffic Filters for IP Spoofing Defense

Jelena Mirkovic and Erik Kline
USC/ISI
Marina del Rey, CA
sunshine,kline@isi.edu

Peter Reiher
UCLA
Los Angeles, CA
reiher@cs.ucla.edu

ABSTRACT

IP spoofing has been a persistent Internet security threat for decades. While research solutions exist that can help an edge network detect spoofed and reflected traffic, the sheer volume of such traffic requires handling further upstream.

We propose RESECT—a self-learning spoofed packet filter that detects spoofed traffic upstream from the victim by combining information about the traffic's expected route and about the sender's response to a few packet drops. RESECT is unique in its ability to autonomously learn correct filtering rules when routes change, or when routing is asymmetric or multipath. Its operation has a minimal effect on legitimate traffic, while it quickly detects and drops spoofed packets. In isolated deployment, RESECT greatly reduces spoofed traffic to the deploying network and its customers, to 8–26% of its intended rate. If deployed at 50 best-connected autonomous systems, RESECT protects the deploying networks and their customers from 99% of spoofed traffic, and filters 91% of spoofed traffic sent to any other destination. RESECT is thus both a practical and highly effective solution for IP spoofing defense.

CCS CONCEPTS

• **Security and privacy** → **Spoofing attacks**; *Network security*;

KEYWORDS

IP spoofing, traffic filtering, DDoS defense

ACM Reference Format:
Jelena Mirkovic and Erik Kline and Peter Reiher. 2017. RESECT: Self-Learning Traffic Filters for IP Spoofing Defense. In *Proceedings of ACSAC 2017*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3134600.3134644

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ACSAC 2017, December 4–8, 2017, San Juan, PR, USA
© 2017 Copyright held by the owner/author(s). Publication rights licensed to Association for Computing Machinery.
ACM ISBN 978-1-4503-5345-8/17/12...$15.00
https://doi.org/10.1145/3134600.3134644

1 INTRODUCTION

IP spoofing—forging the sender's address in the IP header—has been an Internet security threat for decades. Spoofing is an essential ingredient of reflector DDoS attacks, which are currently on the rise [1, 9, 22, 33], and it is used in a myriad of other threats (see Section 2).

While research solutions exist that can help an edge network under attack detect spoofed and reflected traffic, the sheer volume of such traffic requires handling further upstream. Several approaches have been proposed to detect and filter spoofed traffic [6, 10, 11, 13, 18, 23, 25] upstream from the attack victim. *Spoofed traffic filters* build a table of expected values of some traffic parameters (e.g., traffic direction, previous hop, number of hops, signature) for every given source. When traffic arrives with mismatching values, it is considered spoofed and is then filtered out. Ingress filtering or BCP 38 [11] is one type of spoofed traffic filter; it detects random spoofing near its source because the spoofed traffic comes from an unexpected direction (from inside the network vs. from the outside).

But the filtering approaches that use static information (ingress filtering [11], IDPF [10], spoofing prevention method [6]) are *ineffective*—they cannot filter much of the spoofed traffic unless widely deployed [20]. On the other hand, the approaches that use dynamic information about the true source's traffic path (route-based filtering [23], hop-count filtering [13], path identifier [25] and packet passports [18]), can be effective in sparse deployment, but they are *impractical*. They cannot learn correct filtering information when routes change, or when routes are asymmetric or multipath. Figure 1 illustrates a route-based filter at network C that knows traffic from source A comes via a previous hop network B, and thus it can filter M's attack on E, which spoofs A's addresses. But when A's path changes to go through D, B currently has no way to learn this and update the filter table.

In this paper we propose RESECT, a system that enhances dynamic filters [13, 18, 23, 25] with the ability to autonomously learn correct filtering information. Because dynamic filters are very effective in sparse deployment [20], RESECT is both practical and effective. A RESECT system is coupled with a dynamic filter, and helps maintain its table of expected values for its chosen traffic parameter. We will refer to this joint system as "RESECT filter." When a mismatching packet reaches the RESECT filter, RESECT triggers a learning process to possibly update the values in the filter table. During learning, RESECT drops a small

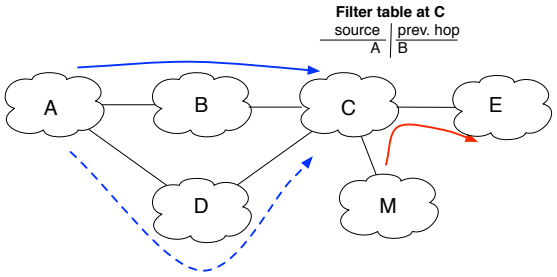


Figure 1: Example of a route-based filter with its filter table. Using information from the table, which says that A's traffic should come via previous hop B, C can filter the attack from M on E, which spoofs A's addresses. However, when routing changes so A comes to C via D, C has no way to update its table.

number of the source's TCP packets. It then identifies retransmissions of these packets and infers the correct previous hop from them.

Our evaluation shows that rare drops introduced by RESECT have no noticeable impact on TCP traffic. Drops to legitimate traffic occur only on route changes, which for most prefixes means one to five times per day [29]. On those rare occasions when packets are dropped, more than 99.94% of TCP connections experience no drops and no ill effects, while the rest experience a single packet drop, and quickly recover. We thus claim that RESECT would be mostly transparent to end users. At the same time, RESECT filters more than 99% of reflector and random-spoofing DDoS attacks, effectively defeating spoofing on the Internet.

RESECT is practical, as it requires modifications only to the deploying networks. It brings great benefits to these networks, removing 74–91% of their incoming spoofed traffic in isolated deployment. Further, if deployed at 50 best-connected ASes (less than 0.1% of the entire Internet), RESECT removes 99% of incoming spoofed traffic at the deploying networks, and 91% of spoofed traffic sent to anyone in the Internet! RESECT is thus very effective in sparse deployment and could be a game-changing solution to IP spoofing.

2 SPOOFING IS PREVALENT

In this section we detail the prevalence of IP spoofing and its use in attacks. We highlight two of the most common and most damaging uses of IP spoofing—volumetric and reflector DDoS attacks. Besides these, IP spoofing is used for decoy scanning [21], in-window TCP reset attacks [34], DNS poisoning [35] and spam filter circumvention [5].

Volumetric DDoS Attacks. These attacks send a large volume of packets to the target. IP spoofing is used to hide the attack machines' identities, usually by forging random addresses from the entire IPv4 address space. Arbor Network's security report for 2016 [1] found that 41% of enterprise and government institutions and 60% of data centers experienced a volumetric attack that exceeded their network capacity.

Reflector DDoS Attacks. In these attacks, the attacker spoofs the IP address of the victim in service requests sent to public servers. The servers respond to the victim, flooding it. Often reflector attacks exploit the *amplification* effect—the fact that some small requests elicit large responses from a server. Recently, there has been a large increase in both frequency and volume of reflector attacks [1], with the largest attack exceeding 1 Tbps [22].

How Many Networks Can Spoof? The Spoofer project [5] measures this by sending spoofed traffic from volunteer machines all over the world to a few select servers. These measurements show that around 56.8% of autonomous systems deploy ingress filtering [11], which ensures that their customers cannot send spoofed traffic. Thus, almost half of the autonomous systems can be used today to source spoofed traffic.

3 RELATED WORK

IP spoofing has been around for a long time, and there is a large volume of research work on combating it. Some spoofing defenses call for Internet redesign, such as TVA [40]. But Internet redesign is not likely to happen soon. We need incrementally deployable spoofing solutions that are effective in sparse deployment.

Other defenses trace spoofed traffic to its original source, such as IP-traceback [30, 31]. Traceback solutions are unattractive because they only detect spoofing after the fact, which means that another solution is needed to actively filter spoofed traffic.

Some defenses filter only reflected traffic (replies to spoofed packets), but not spoofed traffic itself. These defenses include RAD [14], Peng et al. [24] and SNF [4]. They are also ineffective unless deployed in the Internet's core or on most of the reflectors. RESECT filters spoofed traffic, thus handling a wider range of attacks, and it is very effective in sparse deployment.

Like RESECT, Subramanian et al. [32] propose dropping some small portion of TCP packets and using retransmissions to detect established TCP connections. They aim to detect functional routes, not filter spoofed traffic, and do not evaluate the effect of dropping on TCP traffic. Our paper is the first that proposes use of TCP packet drops for spoofed packet filtering, and it fully evaluates the impact of drops on TCP traffic.

Finally, some defenses are *spoofed packet filters* [6, 10, 11, 13, 18, 23, 25], and aim to detect spoofed traffic based on some traffic feature and filter it out upstream from its destination. We discuss these in more detail in the next section and explain their need for a mechanism like RESECT.

In addition to work that focuses on filtering spoofed traffic, other related research seeks to identify networks that allow spoofing [5, 17], understand how spoofing is used for amplification attacks [17, 28], and understand how symmetry in communications can be used to detect and filter unwanted traffic [16].

4 SPOOFED TRAFFIC FILTERS

Spoofed traffic filters build a filter table, which associates each source address or prefix with some traffic parameter (e.g., a route to the filter, a secret mark, etc.). During regular operation, the chosen parameter's values are inferred from each packet and compared to the values noted for the packets' sources in the filter table. Mismatching packets are considered spoofed and dropped.

To date, seven filtering approaches have been proposed, summarized in Table 1. Three approaches use static information for filtering: ingress filtering [11], IDPF [10] and the Spoofing Prevention Method [6]. Ingress filtering [11] uses the direction of traffic. The Spoofing Prevention Method [6] uses a secret exchanged between the source and filter and carried in source's packets. Inter-domain packet filtering [10] uses the set of feasible previous hops that could carry a source's traffic just before it reaches the filter. This information is static because it does not change as routing changes. In [20] we evaluated the performance of filtering defenses. We found that static-information filters are not very effective in sparse deployment. Their effectiveness seems to grow linearly with deployment, and thus very large deployment is needed to effectively filter attacks.

Four filtering approaches use dynamic information about a source's path to the filter. The passport approach [18] uses a sequence of marks, each derived from a secret shared between a source and filters on the path to the destination. A source finds the correct path, which will be taken by traffic, and places appropriate marks in its packets. The path identifier approach [25] uses a similar sequence of marks, but they are placed by the filters without a source's cooperation. They are thus less secure, but the approach is more effective in sparse deployment because it can work with legacy sources. A hop-count filter uses the number of router-hops (inferred from the TTL field) between the source and the filter, which may not be very stable [3]. A route-based filter uses the previous hop traversed by a source's packets before reaching the filter. In our study [20] we found that three dynamic-information filters—route-based filters, hop-count filters and path-identifier filters—were very effective in sparse deployment. When deployed on 0.1% of the most connected autonomous systems, these filters could remove more than 90% of attack traffic. Packet passport filters were less effective than the rest because they require a source's cooperation.

The difficulty in deploying the three dynamic filters that promise to be very effective lies in their inability to update filtering information on realistic routing events, such as route changes, asymmetric and multi-path routing. Currently, these filters have no way to learn up-to-date filtering information on the fly. This is the problem that RESECT aims to solve.

5 RESECT

We now describe how RESECT works, and how it can be interfaced with a route-based filter. Hop-count filters and path-identifier filters could also be enhanced by RESECT in a similar manner.

5.1 Overview

RESECT builds and maintains a *filter table*, which contains expected previous hop information for each source. For scalability, it makes sense to store information per source prefix instead of address. In our evaluation we assume /24 prefix size. In reality, we expect that prefixes in the filter table could have variable size, and that the number of entries would be of the same order of magnitude as the entries in forwarding tables.

RESECT is installed inline and monitors all traffic and all entries in a filter table. We extend the key to the table to be the combination of a source prefix and a previous hop. Thus, one source prefix can be associated with several previous hops (via several entries) to allow for multipath routing. RESECT learns and updates these {prefix, previous hop} combinations via the *learning* process (Section 5.2).

Each filter table entry is associated with *state*, which can be MISSING, NEW, VALID, INVALID and SPARSE. Figure 2(a) illustrates state transitions for a RESECT entry. The table starts empty, i.e., all entries are in the MISSING state. When packets come on such entries, the learning process starts and entry is moved into NEW state. When learning finishes, the entry becomes either VALID or INVALID for some amount of time, and traffic matching this entry will be forwarded or dropped, respectively. A VALID entry has a *ValidTimer*, which is renewed on any traffic match. Its expiration is guided by RESECT's T_{valid} parameter. An INVALID entry has a *FilterTimer*, which expires after a fixed interval, controlled by the parameter T_{filter}. Expired entries are deleted from the table, i.e., they transition into the MISSING state. There may be up to one special entry in the table, called "default entry," in the SPARSE state. This entry is used to filter traffic involved in an ongoing random-spoofing attack. We provide more details about this in Section 5.3.

In addition to learning, RESECT employs two other mechanisms—bounding and flood detection. We describe all three mechanisms next and summarize the cases they handle in Table 2.

5.2 Learning

The goal of the learning is to establish whether the sources of traffic arriving on a NEW entry are legitimate or spoofed. We designed this process so that it quickly and accurately detects valid sources, while it is very difficult for the attacker to manipulate the outcome.

During learning, RESECT drops random D out of the first N TCP packets that match a NEW entry, and forwards the rest. We experimented with data, SYN and FIN packets, which are all retransmitted on drops, and converged on using just data packets because that led to the smallest collateral damage at a reasonable decision delay.[1] We call these N

[1] Dropped SYN and FIN packets are detected by the sender via RTO timeout, which can lead to large delays. Data packets are usually sent in bulk and, with our low drop rates, most drops are detected via triple duplicate acknowledgments.

type	filter	parameter
static	ingress filter	traffic direction
	inter-domain packet filter	set of feasible previous hops
	spoofing prevention	packet mark per dest, placed by source
dynamic	packet passports	sequence of packet marks per route, placed by source
	route-based packet filter	one previous hops
	hop-count filter	hop count between source and filter
	path identifier	sequence of packet marks, placed by filters

Table 1: Filtering approaches and the parameters they use in their filter table.

packets the *test packets*. All other packets matching a NEW entry, such as UDP, TCP SYN, ICMP, etc., are forwarded.

A legitimate sender will learn which packets were dropped via TCP's duplicate acknowledgments or via the retransmission timer's expiry. Such a sender will retransmit all D dropped, and very few of $N-D$ forwarded packets. RESECT detects this behavior by storing unique identifiers (TCP sequence numbers) of dropped and forwarded packets in its *DroppedQueue* and the *ForwardedQueue*, respectively, for each entry. It assigns one *valid* point to the entry when a packet from the DroppedQueue is repeated *for the first time*. Conversely, it assigns one *invalid* point to the entry *every time* a packet from the ForwardedQueue is repeated. All repeated test packets are forwarded. If a NEW entry collects D valid points, its state is changed to VALID. This is how RESECT learns new correct filtering information on route changes.

An attacker may try to manipulate RESECT. He may choose to repeat no packets—*simple TCP-data attack*, or he may try to guess which ones to repeat—*repeating TCP-data attack*. To handle the simple TCP-data attack, RESECT starts a *DropTimer* for the entry in the learning process each time a matching packet is dropped due to learning, and restarts it each time a dropped packet is repeated for the first time. On timeout, the associated entry is declared INVALID. To handle the repeating TCP-data attack, we designed the learning process to penalize repeating of packets that RESECT has forwarded. This is the reason why invalid points are incremented on every repetition of packets in the ForwardedQueue, while valid points are only incremented on the first repetition of packets in the DroppedQueue. A guessing attacker will thus quickly accumulate invalid points. If a NEW entry collects $(N-D)/2$ invalid points, its state is changed to INVALID.

We use D and $(N-D)/2$ as valid and invalid point thresholds, respectively. We tested other values for valid and invalid point thresholds, but these had the best trade-off between false positives and false negatives.

5.3 Bounding and Flood Detection

There are two types of attacks, listed below, that require additional mechanisms apart from learning.

Reflector Attacks. Let a reflector attack, sending spoofed traffic, occur after a long period of no attacks. The attack may arrive on a MISSING or a VALID entry; recall that an entry consists of {prefix, previous hop} combination. Attacks arriving on a VALID entry arrive on an expected previous hop and will not be filtered because they will match the entry. So the deployment goal of RESECT would be to maximize the chance of attacks arriving on MISSING entries. This is achieved when RESECT is deployed on highly connected networks, such as those in the Internet core.

When spoofed traffic for reflector attacks arrives on a MISSING entry, it will likely use non-TCP-data packets, such as DNS or NTP requests. RESECT will trigger the learning process, but without additional mechanisms, learning will never complete because there will not be enough packets in the DroppedQueue and the ForwardedQueue. Reflector attacks are handled through the *bounding* process in RESECT, limiting the number of packets that can be forwarded on NEW entries associated with a given source, using parameter MAX_{pkts}. When this value is exceeded, all the source's entries that were in the NEW state become INVALID.

Random-spoofing Attacks. Because these attacks spoof at random in a large address space (often the entire IPv4 space), many NEW entries would be formed by RESECT, and it would take a long time for their learning to complete—allowing the attack to flood the victim in the meantime. RESECT's *flood detection* handles random-spoofing attacks by detecting likely victims of these attacks and adding a pre-filter step to all traffic arriving on NEW entries. RESECT detects likely victims by caching the most frequent destinations of packets (test or regular packets) that match a NEW entry in a small cache associated with each entry—*dstCache*. When more than MAX_{pref} prefixes have entries in a NEW state, RESECT examines the *dstCache*s of these entries. Each IP that appears in many caches (more than ENT_{dst}) and has received many packets (more than MAX_{pb} packets total) is moved into the *victim set*. RESECT detects a flood whenever the victim set is not empty. The victim set then contains likely victims of the random-spoofing attack.

To pre-filter traffic to these likely victims, RESECT creates a default entry in the SPARSE state, if not already present, and adds the victim set to the entry's *dstCache*. The NEW entries whose dstCache is completely contained in the victim set of the default entry are deleted. There can be only one

Mechanism	Case handled
Learning	Route changes and spoofed TCP-data pkts
Bounding	Reflector attacks with spoofed UDP pkts
Flood detection	Spoofed volumetric attacks

Table 2: RESECT's mechanisms and cases they handle

default entry in a filter table. A packet would match a default entry only if it matches no other entry in the table *and* its destination IP is in the victim set. Such packets are dropped. Similar to an INVALID entry, a SPARSE entry also has *FilterTimer* associated with it, which expires after T_{filter} seconds and leads to the entry's deletion.

It may seem that bounding and flood detection would suffice to filter spoofed traffic, and that learning is redundant. But without learning, bounding and flood detection would have many false positives, filtering traffic from large senders or traffic to popular destinations. Learning enables bounding and flood detection to work only on traffic that arrives with unexpected {prefix, previous hop} combinations, due to route changes or due to spoofing, which greatly reduces false positives.

5.4 Effect on Legitimate Traffic

In this section we discuss how RESECT affects legitimate traffic. During learning, RESECT drops may impose delays on legitimate traffic. We first explain why packet dropping is necessary. We then discuss that RESECT's dropping is *rare*, *limited to a small fraction of connections* and *limited to a single packet drop on a connection*, and thus introduces no perceptible reduction in quality of service (QoS). We end the section with a discussion of other rare situations when RESECT may negatively interact with legitimate traffic.

Why Drop? RESECT learns expected previous hops by observing retransmissions of packets it drops on TCP connections. We considered two alternatives to dropping: sending probe packets to traffic sources, or detecting established TCP connection by observing traffic. Probing does not work, because many Internet hosts filter unsolicited traffic [12] and many others use dynamic IPs [39]. This would lead to sparsely populated filter tables, which would be ineffective in filtering attacks. Observing traffic to detect established TCP connections works reliably only when routing is symmetric. Otherwise the attacker can easily spoof an established connection in unidirectional traffic. This leaves dropping as the only viable mechanism to detect established TCP connections. Dropping establishes a communication channel between RESECT and the alleged packet source that the attacker cannot observe.

Dropping Is Rare. A small fraction of legitimate traffic will be dropped when a route change occurs that affects its source's entry at a RESECT filter, or when its source sends traffic after a long pause (e.g., several hours or days). Redford et al. [27] found that a vast majority of BGP paths are stable, with a change frequency of five times per a day to once a week. Thus, dropping of legitimate traffic by RESECT should be very rare.

Dropping could also occur if a legitimate source's entry were mistakenly flagged as INVALID during learning. This could happen if a prefix experiences a route change or if there is congestion on the path that leads to excessive retransmissions. The case of high congestion and route change affecting the same entry should be very rare, since Internet packet loss is low [36], and route changes are rare [27].

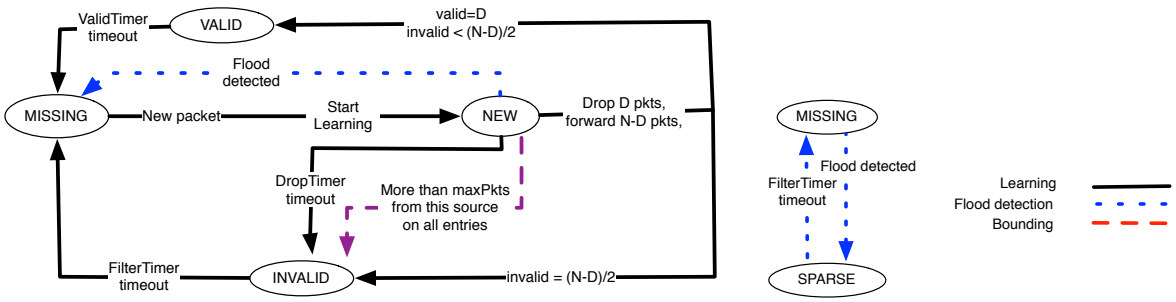
Dropping Is Limited. In Section 6 we show that at small drop rates (0.1–0.2%), which we recommend, less than 1.6% of legitimate connections experience *any delays during learning*. Taking into account learning frequency, less than 0.06% of legitimate connections in a day would be affected by a RESECT filter. These connections would be randomly chosen across multiple addresses and users, which makes it unlikely for any single user's QoS to be affected.

Dropping is not only rare and limited to a few connections, it also usually affects only *one packet per connection*. TCP and various applications recover quickly from such isolated drops. We show in Section 6.3 that only 8.8% of users whose traffic was dropped notice reduced service quality.

To summarize, any single user in any single day would stand a very small chance (among all other users) that *one* of their connections would experience a single packet drop, from which it would quickly recover. Such a user would have only an 8.8% chance of perceiving lower QoS at that time. At other times, RESECT's operation would be transparent.

Dropping Is Serialized. A given connection could experience learning at multiple RESECT filters simultaneously, which could lead some filters to reach the INVALID decision because packets they forwarded were dropped downstream. RESECT handles this through implicit filter synchronization. A filter that has an entry in the learning process places a well-known value in the IPv4's TOS field (or IPv6's flow identifier field) of test packets that it forwards for that entry. A downstream RESECT filter that sees marked packets delays its learning for the given source, until either the traffic stops being marked or when N_{marked} marked packets have been seen. When the first RESECT filter completes learning, it stops marking the packets, which allows the next downstream filter to start learning. The attacker that fakes the well-known value in the TOS field can postpone learning for a maximum of N_{marked} packets.

We analyzed the AS-level Internet topology (see Section 6), and found that 99% of paths were up to four AS hops long; thus at most, four filters could learn about the same source simultaneously. Assuming $N = 10,000$ (see Section 6 for rationale), using $N_{marked} = 4 * 10,000 = 40,000$ ensures no misclassifications due to cascaded filters, while filtering all but the first $40,000$ attack packets. This means that a 1 Gbps attack would be filtered in under half a second. We have also evaluated cascaded RESECT filters, but summarize



(a) Any but default entry

(b) Default entry

Figure 2: State diagram showing state transitions for learning (black lines), bounding (dashed red line) and flood detection (dotted blue lines).

results due to space. The effect of C cascaded filters, each with parameter N, is the same as the effect of one filter with parameter $C \times N$.

Coincidence of Legitimate and Attack Traffic on the Same Entry. Attack and legitimate traffic may both arrive at the same entry and thus will share the same fate. If this entry is in the VALID state (legitimate traffic has made the entry VALID prior to the attack), both traffic types will be forwarded. This situation is rare, and would occur in 3–8% of all possible attack scenarios [20]. If the entry is in the INVALID state or NEW state, which ends up declared INVALID, both attack and legitimate traffic would be dropped. This situation only occurs if the legitimate source experienced a BGP path change *at the same time or shortly after* a spoofing attack, it was actively generating traffic at that time, and the new legitimate traffic's path overlaps the attack path. While spoofing attacks are relatively frequent, attacks that spoof any single prefix should be rare. Routing changes are also rare for most prefixes and legitimate traffic is bursty. Further, an entry should be in the NEW or INVALID state for a very short time (a few minutes, see Section 6). The probability of a spoofing attack, a routing change and legitimate traffic all affecting the same entry at the same time should thus be very low.

Prefixes That Send Mostly UDP Traffic. Filter table entries for prefixes that rarely send TCP packets, such as those hosting popular DNS or NTP servers, may mistakenly be classified as INVALID by RESECT's bounding mechanism. This effect can be controlled by carefully setting the MAX_{pkts} parameter, as we discuss in Section 5.5. In short, the deploying network would profile its transit traffic, measuring the maximum number of non-TCP-data packets sent by any prefix, before N TCP-data packets. It would then set the MAX_{pkts} to a value higher than this maximum. In our measurements on recent MAWI traces (2017), the maximum length of a non-TCP-data sequence was around 38,000 packets. In our evaluation (Section 6), we used a much higher number—1,000,000 or MAX_{pkts}—and have shown that this leads to no misclassifications of legitimate traffic and filters reflector attacks within seconds. Thus, we feel confident that RESECT would not harm legitimate senders that rarely send TCP traffic.

Congestion. If a source prefix experiences packet drops due to congestion *at the same time* as a route change, legitimate senders may repeat packets that were originally forwarded by RESECT, which may lead to an INVALID state. Packet loss in the Internet is usually low [36], and route changes are rare, which greatly minimizes the probability of those two events occurring simultaneously. Even if this happens, the duration of RESECT's filtering and its collateral damage are bounded by FilterTimer to a few minutes.

5.5 Configuring and Bootstrapping

RESECT has eleven parameters summarized in Table 3, but only two need to be tuned by a deploying network, while our recommended values (based on experimental results) can be used for others. The *ValidTimer* value determines how often legitimate traffic will experience learning due to inactivity, i.e., *in absence of attacks or route changes*. A deploying network could measure packet inter-arrival times on filter entries, and set the *ValidTimer* to match a high percentile (e.g., 99.99%). Further, the MAX_{pkts} parameter determines the maximum number of non-TCP-data packets that could be seen on a filter entry before N TCP data packets are seen. A deploying network would measure this value from the traffic it forwards, for each source prefix, and choose the maximum or a high percentile as its threshold. We applied the same approach to calibrate this parameter in our evaluation.

When bootstrapping a RESECT filter at its installation, one could stagger learning on MISSING entries. This would ensure that TCP connections do not experience drops in both directions.

6 EVALUATION

In this section we first evaluate RESECT's impact on legitimate and attack traffic (Sections 6.1 and 6.2). We replay *in congestion-responsive manner* the TCP traffic from several public traffic traces, and in some tests, we overlay it with

Label	Meaning	Recommended
	Learning	
N	# test pkts	10,000
D	# dropped pkts	10–20
$d = D/N$	% dropped pkts	0.1–0.2%
T_{drop}	*DropTimer* timeout	10 s
T_{filter}	*FilterTimer* timeout	190 s
T_{valid}	*ValidTimer* timeout	measure
	Bounding	
MAX_{pkts}	# pkts on NEW entries for a src	measure
	Flood detection	
MAX_{pref}	# prefixes with NEW entries	100
ENT_{dst}	# NEW entries that send traffic to a dst	100
MAX_{pb}	# pkts to a dst on NEW entries	100,000
N_{marked}	# marked pkts	40,000

Table 3: RESECT parameters and recommended values

Trace	Date	pkts/bytes
CAIDA – San Jose	07/21/2011	2.2 B/2025 B
CAIDA – Chicago	03/24/2011	3.8 B/2208 B
MAWI	05/15/2011	34 M/26 B

Table 4: Traces used in evaluation: M=million, B=billion.

attacks. In a congestion-responsive replay, sender and receiver dynamics, as well as network conditions, are faithfully reproduced and traffic drops in simulation lead to retransmissions and connection delays, just like they would in real deployment.

We replay traffic traces in NS-2 simulations using Tmix [38], an NS-2 module that replays TCP traffic from a given trace in a congestion-responsive manner, emulating realistic end-to-end network delays (mined from traffic), and realistic retransmission behavior on any packet drops. We preserve the original IP addresses and port numbers in replayed traffic to faithfully reproduce per-prefix, per-source and per-connection behavior in our tests. We use one backbone trace shared by MAWI [19] and two shared by CAIDA (San Jose and Chicago) [8]. These were the largest and most diverse traces we could find. Table 4 summarizes some statistics about the traces.

We further evaluate RESECT's operational cost and impact on human-perceived QoS by implementing it in the Click 2.0 software router [15], and running tests on the Deterlab testbed [2] (Section 6.3). Finally, we evaluate how much RESECT would help with today's attacks (Section 6.4).

6.1 Parameter Tests

We first examine the effect of N and $d = D/N$ parameter settings on RESECT's accuracy of classification and collateral damage. These tests use only legitimate traffic. The trace is replayed by two Tmix agents through one RESECT filter, and we evaluate the effect of the filter's actions on this traffic. All timers were set to never expire. RESECT starts with an empty filter table, so each entry goes through learning once only, and we simulate no routing changes. This test lets us estimate the damage to all connections from one learning cycle. Our measure of collateral damage is the percentage of connections that experience *any* delay due to packet dropping.

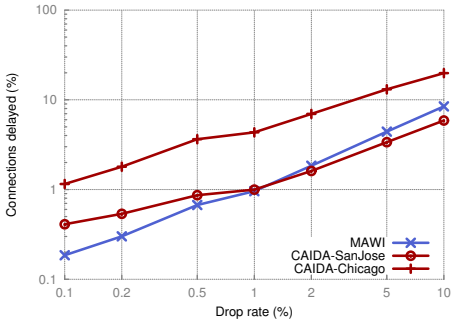


Figure 3: Collateral damage vs drop rate d

Figure 3 shows the percentage of all legitimate TCP connections in the test, which are delayed, as we vary the drop rate d between 0.1% and 10%, on a log-log scale. We run each test 10 times and show only the average, since standard deviation is very low. Number of test packets, N, is set to $1,000$ for sub-1% drop rate and to 100 for drop rates above 1%. These values of N were chosen to strike a balance between number of drops (at least 1 per prefix) and the number of prefixes for which we reach the decision (so we can measure decision accuracy). Smaller N values would lead to zero packet drops on some prefixes, and would not be realistic. And larger N values lead to fewer prefixes that generate enough TCP data packets in our 15-minute traces, for learning to converge.

Collateral damage to legitimate traffic is very small, and grows linearly with the drop rate. It ranges from 0.1–0.4% of connections being affected at 0.1% drop rate to 5.8%–8.4% at 10% drop rate for the CAIDA-SanJose and MAWI traces, respectively. The effect on legitimate connections is more pronounced for the CAIDA-Chicago trace, because it has a few very large connections (0.01% of connections in this trace carry 40% of packets). All the affected connections lose a single packet, and quickly recover. We suggest a 0.2% or 0.1% drop rate based on these tests, which lead to at most 1.6% connections being delayed during learning.

We next examine the effect of the parameter N—how many test packets are used—on collateral damage. For these tests we fix the drop rate d at 0.1%, 0.2% and 1%, and vary N from $1,000$ to $10,000$. Since the effect is similar on all traffic traces we only show the results for the CAIDA-SanJose trace in Figure 4. The damage grows logarithmically with N and

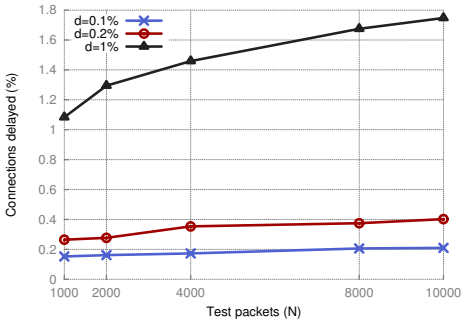


Figure 4: Collateral damage vs N

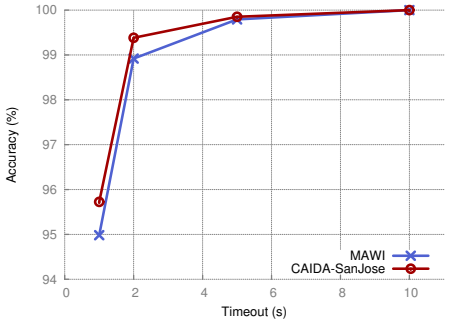


Figure 5: Classification accuracy vs DropTimer timeout

remains very small for all tested values. We recommend the highest value we tested—10,000—because it leads to good resiliency against packet-repeating attacks (Section 7).

We also measured the classification accuracy of RESECT in all these tests; it was 100%, i.e., RESECT always correctly classified the new filter table entry as VALID.

6.2 Timeout Tests

RESECT's decision speed depends on how quickly a source prefix sends N unique TCP data packets during learning, and how quickly it repeats dropped packets. A legitimate source may send traffic quickly or slowly, but it will always promptly repeat dropped packets. In this case, learning is entirely driven by traffic and can take as long as needed. An attacker may send TCP-data traffic aggressively and never repeat any. In this case the learning process will last at most until DropTimer expires, which is controlled by parameter T_{drop}. We tested T_{drop} values of 1, 2, 5 and 10 seconds, and set the N and d to 1,000 and 0.1% respectively. We kept the rest of the settings as shown in the previous section. Note that Tmix mines realistic end-to-end delays from traffic, so our simulation includes realistic RTT distributions. Figure 5 shows the classification accuracy on the MAWI trace, and the results on other traces are similar. The accuracy is 100% for a 10-second value and declines as smaller values are tested due to a few large RTTs. Since RTT distribution is very similar in many public traces [13], we recommend 10 seconds for T_{drop} parameter.

The setting for the FilterTimer, T_{filter}, will determine how much of the attack is filtered before its INVALID entry expires. If we select $T_{drop} = 10$ seconds as the lowest acceptable value for DropTimer timeout, then setting $T_{filter} = 190$ seconds would achieve filtering effectiveness of 95%. It also limits dropping to legitimate traffic, in the case of an erroneous INVALID decision, to a little over 3 minutes.

Timeout setting for the ValidTimer, *Tvalid*, depends on the network's traffic dynamics and should be calibrated by each deploying network.

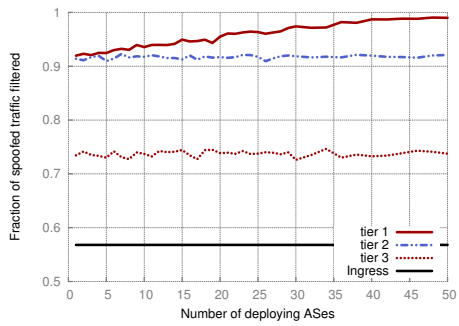
6.3 Human-Perceived QoS

All legitimate connections in our tests that were affected by RESECT's learning experienced a single packet drop. We ran tests with Mechanical Turk participants and our prototype implementation of RESECT in a Click software router to measure impact of such drops on quality of service (QoS). Each participant was asked to click through a set of five small, static Web pages and rate their loading speed. We dropped a random 1 in 10 user data packets. Drops on a client connection in an interactive application, such as Web, create the worst impact on QoS. Further, since our drops occur on a 2-data-packet TCP connection, retransmissions are triggered on the TCP's retransmission timer expiry, which may take two RTTs. This is worse than drops on larger connections that are quickly detected through triple duplicate acknowledgments.

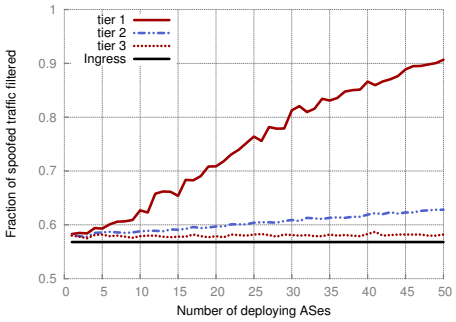
Each participant was asked to rate each page's loading speed on a scale of 0 (worst) to 5 (best). We say that a user has "perceived the RESECT-induced QoS drop" if she gave a lower rating to the page whose initial request was dropped by RESECT than the average of ratings she gave to other pages. We report our findings for 34 participants. Only three participants perceived the RESECT-induced QoS drop, which is 8.8% of all participants. These users had a drop in the middle of their interaction (pages 2 and 3 out of 5) and were able to return to higher QoS perception at subsequent page loads. We thus conclude that (1) many humans (91.2%) do not notice single-packet drops, and (2) there is no lasting impact from isolated drops on a user's QoS perception.

6.4 Effect on Today's Attacks

We now evaluate how well RESECT would handle today's reflector and random-spoofing denial-of-service attacks. We first evaluate the benefit than an AS and its customers would see from deploying RESECT, *assuming all attack traffic arrives on INVALID entries*. We then evaluate how quickly one RESECT filter would respond to an attack, which arrives on a MISSING entry.



(a) Benefit to deploying AS



(b) Benefit to others

Figure 6: Filtered incoming spoofed traffic for (a) a RESECT-deploying network and its customers, (b) everyone in the Internet.

Benefit from RESECT. We obtained AS-level connectivity and relationship information from CAIDA [7] and evaluated the protection that RESECT offers to a deploying AS and its customers from spoofed traffic. We evaluated deployment on a random 1–50 ASes from the following groups: tier-1 (top 50 connected ASes), tier-2 (next top 50 connected ASes), or tier-3 (all other ASes that have at least one customer link). We further assumed that a random 56.8% of ASes deploy ingress filtering [5]. Figure 6(a) shows the median percentage of spoofed traffic *filtered* at the deploying AS after RESECT's filtering and after ingress filtering. At even a single-node deployment, 45–61% more spoofed traffic is filtered by RESECT than by ingress filtering. At 50 AS deployment, tier-3 deployment filters 74% of the spoofed traffic, tier-2 deployment filters 91%, and tier-1 deployment filters more than 99%. We also evaluated the overall reduction of spoofed traffic in the Internet by deploying RESECT; this is shown in Figure 6(b). Only tier-1 deployment significantly filters spoofed traffic; it filters 91% when 50 tier-1 nodes deploy RESECT.

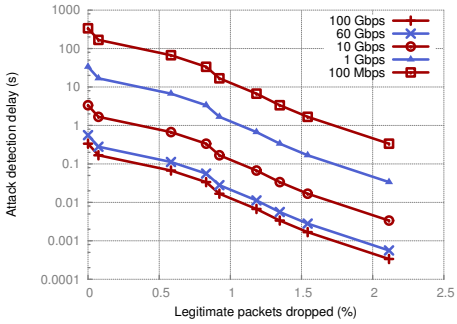


Figure 7: Attack detection delay vs collateral damage

Filtering Reflector Attacks. We now investigate how to set up the parameter MAX_{pkts} for the RESECT's bounding algorithm to minimize risk to legitimate traffic while effectively filtering attacks. We simulate RESECT's handling of legitimate traffic on our three traces using libpcap to read each packet (this includes non-TCP packets), and change its entry's state to INVALID if MAX_{pkts} packets have been seen before N TCP data packets, and to VALID otherwise. We vary N from 100 to 10,000, and vary MAX_{pkts} parameter from 1,000 to 1,000,000 packets. For reflector attacks, we calculate the number of packets per second sent by a DNS amplification attack that are sufficient to flood a link with 100 Gbps, 60 Gbps, 10 Gbps, 1 Gbps and 100 Mbps bandwidth. We assume an amplification effect of 70 and DNS query size of 60 B [26]. Attacks last 15 minutes and are filtered after the first MAX_{pkts}. INVALID entries expire after $T_{filter} = 190$ seconds.

We show the results for the CAIDA-Chicago trace; other results are similar. Figure 7 shows the percentage of all legitimate traffic dropped on the x-axis against the attack detection delay in seconds on the y-axis. Figure 8 shows the legitimate versus attack packets dropped, both as percentages. Zero collateral damage corresponds to the setting $MAX_{pkts} = 1,000,000$. With that setting, 100 Gbps and 60 Gbps attacks are detected within a fraction of a second and 99.8% of attack packets are filtered. A 10-Gbps attack is detected within 4 seconds and 98% of attack packets are filtered. Smaller attacks of 1 Gbps and 100 Mbps take 33.6 and 336 seconds to be detected, respectively, and 85% and 36% of their traffic is filtered.

Filtering Random-Spoofed Attacks. We use the same simulation approach as in the previous test, and explore different values for MAX_{pref}, ENT_{dst} and MAX_{pb} parameters. For legitimate traffic, each entry's state is changed to NEW with probability 0.1 at a random point during the simulation. This leads to 10% of entries being in learning, which is much

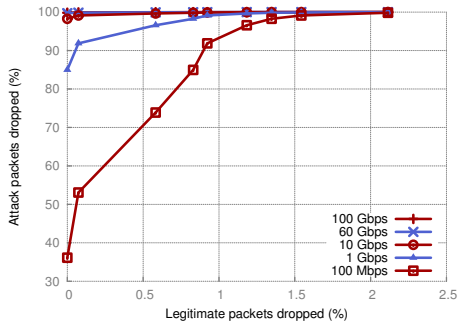


Figure 8: Attack packets filtered vs collateral damage

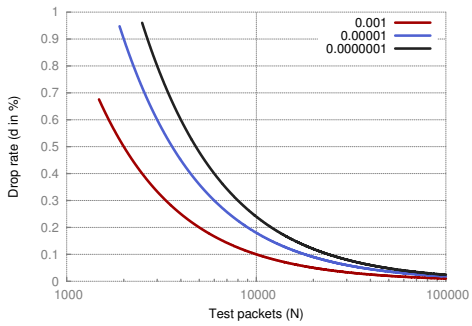


Figure 9: How to set N, d for desired limit on p_{success}, shown in the legend.

higher than would be the case in real deployment.[2] When an entry goes into learning, we start populating its *dstCache*, whose size we fixed at 10 entries. Each second we run the flood detection test on all learning entries and record the number of destinations for which we falsely detect a random spoofing attack. The entry changes from NEW to VALID when N TCP data packets have been seen. We vary N from 100 to 10,000. We summarize results due to space: $MAX_{pref} = 100$ and either $ENT_{dst} > 100$ or $MAX_{pb} < 100,000$ had zero false positives, while detecting 1 Gbps and larger attacks in under 2 seconds. If we used $T_{filter} = 190$ seconds, almost all attack traffic would be filtered by RESECT.

An attacker may choose to perform an *avoidance attack*—to spoof only a subset of IPv4 address ranges, e.g., only within X /24 prefixes. If $X < MAX_{pref}$ and $X < ENT_{dst}$, such an attack would not be detected as a random-spoofing volumetric attack by RESECT. Instead, each entry would go into learning independently, allowing the attacker to send spoofed traffic for a brief interval, until INVALID decision is reached. This attack could then evolve into *cycling attack* to continuously flood the target. We propose how to handle this in Section 7.

7 ATTACKS ON RESECT

We considered several attacks on RESECT, which could lead to an erroneous VALID or INVALID decision.

Packet-Repeating Attack. An attacker familiar with RESECT's operation could try to guess which packets they should repeat to achieve a VALID decision for their spoofed traffic. Because RESECT penalizes wrong guesses (by incrementing invalid points), a *permutation attack* is the optimal attacker strategy. The attacker sends a sequence of N unique packets, and repeats a random permutation of those. Let V be the number of valid points needed for a VALID decision, and S be the number of invalid points needed for an INVALID decision. The attacker must gain V valid points before collecting S invalid points to trick RESECT. The probability of the attacker defeating RESECT using the permutation attack is:

$$p_{\text{success}} = \frac{\sum_{i=V}^{D} \binom{D}{i}\binom{N-D}{V+S-i-1}}{\binom{N}{V+S-1}}$$

We use $V = D$ and $S = (N - D)/2$.

Figure 9 shows how to set N and $d = D/N$ to limit the probability of attack success to some desired low value. For example, for $d = 0.2\%$ and $N = 10,000$, an attacker would have a 1 in 100,000 chance to trick RESECT, but all other attack traffic would be filtered for $T_{filter} = 190$ seconds. Thus, on the average, an attacker would have to continuously launch attacks for $190 * 50,000/24/3,600 = 109$ days before succeeding to convert one MISSING entry into a VALID entry. We conclude that packet-repeating attacks are ineffective for the attacker.

Congestion Attacks. The attacker could lead RESECT to reach an INVALID decision for a rarely active prefix by inducing that prefix to contact the attacker's server, and then the attacker would drop all traffic, which triggers retransmission. This attack will only be successful if several events coincide: (1) the prefix rarely sends traffic through the given RESECT filter, (2) some host from the prefix can be induced to contact the attacker and this traffic passes through the given RESECT filter, (3) the prefix sends some legitimate traffic shortly after the attack through the given filter. Since these conditions cannot be controlled or observed by the attacker, this attack is unlikely.

Collusion Attack. An attacker may collude with a "helper" host, which could observe traffic near the victim and communicate to the attacker which packets were forwarded during learning. The attacker can then correctly repeat the dropped packets to validate an entry for any source. Using an AS-level map of the Internet, we calculated the fraction of the Internet ASes that could be helpers for each possible pair of attack host/target, assuming RESECT was deployed at the 50 tier 1 ASes. For 50% of pairs, less than 0.5% of Internet locations

[2]Let the filter table have E entries, each with 5 route changes per day. Each learning cycle lasts at most $T_{drop} = 10$ seconds. Assuming independently distributed route changes, in any single second $E * 5 * 10/24/3600 = 0.0006 * E$ or 0.06% of entries would be in learning.

were a suitable helper; for 80% of pairs there were less than 2%, and the highest value was 3.5%. Thus, an attacker must work hard to find a suitable helper for each attack host and must repeat this whenever he wants to send spoofed traffic to a new victim. This makes collusion attacks unlikely.

Cycling Attack. An attacker may perform a cycling attack, in which he randomly spoofs a subset of sources, and moves on to a new subset when the previous subset is filtered by RESECT. We can handle this attack with a mechanism similar to flood detection. We can set a low-value threshold on the number of INVALID entries within a time interval (similar to MAX_{pref}). When this threshold is exceeded, RESECT would trigger the same algorithm used for flood detection, but on INVALID entries. This would lead to a default entry in the SPARSE state, which would match all the attack traffic.

8 DEPLOYMENT CHALLENGES

We now discuss the cost of deploying RESECT and deployment challenges such as incentives for filters.

8.1 Cost

RESECT could be deployed on dedicated hardware inline with traffic. We now estimate RESECT's processing and memory cost. For each packet, RESECT performs a filter table lookup, optionally records the packet's sequence number and updates some counters, and stores the destination IP address in *dstCache*, These operations can be performed at line speeds today. To evaluate this processing cost, we implemented RESECT in a Click 2.0 software router [15] on the DeterLab testbed [2]. We regenerated traffic from the MAWI trace, so that we preserve connection volume, timing of packets and packet sizes, as well as packet sources. RESECT's processing cost was 75 ns per packet; it included 1–2 hash table lookups per packet, and was dominated by the inefficient hash table implementation in Click. Taking the MAWI trace's average packet size of 385 B, RESECT could support network links of up to 5 Gbps with this straightforward software router implementation, and it would be much faster in hardware.

RESECT also incurs a storage cost. For route-based filtering, each filter table entry would store the source prefix (32 bits), the expected previous hop (12 bits based on the current connectivity of ASes in the Internet AS map), and the state (2 bits). Thus, in absence of learning, each entry would need 46 bits $\approx$ 6 B. Assuming that RESECT filters can perform the same aggregation of prefixes as routers do for FIBs, there would be around 400,000 entries [37], and a basic filter table would need 2.4 MB of storage.

Each entry may also go into learning process, and this requires storage for N unique packet identifiers, a small hash table for *dstCache* with at most 8 B per entry, seven integer counters, and three timers. For $N = 10,000$, the total size of an entry should be just above 40 KB. In regular operation, only a few entries would go into learning simultaneously. A random-spoofed traffic, which forces learning on many entries, will be quickly detected by RESECT's flood detection, which frees up learning process memory.

8.2 Deployment Options

RESECT performs best when deployed on tier-1 and tier-2 networks, but it may be very difficult to achieve adoption in these large networks. They are in critical positions on the Internet, and must forward traffic as quickly as possible. Requiring them to store state, and to drop some packets voluntarily during learning, is a hard sell. However, most networks today do many complicated operations for different traffic flows, for security or added functionality, beyond transit. Those networks use middleboxes and network-function virtualization (NFV) to support these complicated operations for a small fraction of flows, while maintaining fast handling of the rest of the traffic. RESECT would fit well into this model as another type of functionality to be offered at a middlebox or a VM using NFV.

Ultimately, spoofing is a long-present, serious and increasing problem, as are large-volume DDoS attacks that deploy spoofing. Something must be done! Prior research [20] has shown that core deployment of route-dependent filters would reduce spoofing in the Internet to minuscule levels, and benefit all Internet hosts at a very small deployment. RESECT makes these claims hold in realistic routing situations, and has very low impact on legitimate traffic. We believe that these good results merit further investigation, and possible adoption of RESECT by large, well-connected networks.

RESECT could be deployed at smaller ISPs instead of large networks. Such deployment would protect the ISP's customers from volumetric spoofing attacks, but it would not protect them from reflector attacks. This is because spoofed traffic in reflector attacks traverses many Internet paths and requires Internet core deployment of filters to be detected and removed. Reflected traffic converges at the victim, but such traffic is not spoofed, and cannot be handled by RESECT.

RESECT could further be deployed in proximity of mismanaged networks [41], e.g., by their peers. Since many attacks are launched from a small number of such networks, small RESECT deployment could largely reduce both volumetric and reflector spoofing attacks while having no ill effect on the rest of the Internet or on legitimate traffic from mismanaged networks.

9 CONCLUSION

IP spoofing is an ongoing problem on the Internet that has not yet been solved in a practical and effective manner. RESECT offers a solution that is both practical and effective. RESECT works without changes to Internet hosts, and filters 74–91% of incoming spoofed traffic at the deploying networks. Further, at tier 1 deployment, RESECT filters 99% of incoming spoofed traffic at the deploying networks, and it filters 91% of spoofed traffic sent to any destination. While RESECT drops some legitimate traffic during learning, this dropping is rare and limited, leading to no perceptible QoS loss. RESECT has an excellent classification accuracy and

is resilient to both naive and sophisticated attacks. We thus believe it is a practical and effective solution to IP spoofing.

10 ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation, grant number 0716452. Authors are grateful to Nikola Jevtic, for his help in modeling packet-repeating attackers, and to anonymous reviewers for their helpful comments.

REFERENCES

[1] Arbor Networks, Worldwide Infrastructure Security Report 2016.
[2] DeterLab Testbed. http://deterlab.isi.edu.
[3] Backes, M., Holz, T., Rossow, C., Rytilahti, T., Simeonovski, M., and Stock, B. On the feasibility of ttl-based filtering for drdos mitigation. In *Proceedings of Research in Attacks, Intrusions, and Defenses - 19th International Symposium, RAID 2016* (2016), pp. 303–322.
[4] Basheer, D., and Manimaran, G. Victim-assisted Mitigation Technique for TCP-Based Reflector DDoS Attacks. In *4th IFIP-TC6, Networking* (2005).
[5] Beverly, R., Berger, A., Hyun, Y., and k claffy. Understanding the Efficacy of Deployed Internet Source Address Validation Filtering. In *Proc. Internet Measurement Conference* (2009).
[6] Bremler-Barr, A., and Levy, H. Spoofing Prevention Method. In *Proc. of INFOCOM* (2005).
[7] CAIDA. The CAIDA AS Relationships Dataset, June 1, 2016. http://www.caida.org/data/as-relationships/.
[8] CAIDA. The CAIDA UCSD Anonymized Internet Traces 2011. http://www.caida.org/data/passive/passive_2011_dataset.xml.
[9] CloudFlare. Technical Details Behind a 400Gbps NTP Amplification DDoS Attack. https://blog.cloudflare.com/technical-details-behind-a-400gbps-ntp-amplification-ddos-attack/.
[10] Duan, Z., Yuan, X., and Chandrashekar, J. Constructing Inter-Domain Packet Filters to Control IP Spoofing Based on BGP Updates. In *Proc. of INFOCOM* (2006).
[11] Ferguson, P., and Senie, D. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. IETF RFC 2267.
[12] Heidemann, J., Pradkin, Y., Govindan, R., Papadopoulos, C., Bartlett, G., and Bannister, J. Census and survey of the visible internet. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement* (2008), ACM, pp. 169–182.
[13] Jin, C., Wang, H., and Shin, K. Hop-count filtering: an effective defense against spoofed DDoS traffic. In *Proc. of the 10th ACM conference on Computer and communications security* (2003).
[14] Kline, E., Beaumont-Gay, M., Mirkovic, J., and Reiher, P. RAD: Reflector Attack Defense Using Message Authentication Codes. In *Proceedings of the ACSAC* (2009).
[15] Kohler, E., Morris, R., Chen, B., Jannotti, J., and Kaashoek, M. F. The Click modular router. *ACM Transactions on Computer Systems 18*, 3 (2000).
[16] Kreibich, C., Warfield, A., Crowcroft, J., Hand, S., and Pratt, I. Using packet symmetry to curtail malicious traffic. In *Proceedings of HotNets* (2005).
[17] Kührer, M., Hupperich, T., Rossow, C., and Holz, T. Exit from hell? reducing the impact of amplification ddos attacks. In *23rd USENIX Security Symposium* (2014), pp. 111–125.
[18] Liu, X., Yang, X., Wetherall, D., and Anderson, T. Efficient and Secure Source Authentication with Packet Passports. In *Proc. of SRUTI* (2006).
[19] MAWI Working Group Traffic Archive. http://mawi.wide.ad.jp.
[20] Mirkovic, J., and Kissel, E. Comparative Evaluation of Spoofing Defenses. *IEEE Transactions on Dependable and Secure Computing 8*, 2 (March - April 2011), 218–232.
[21] nixCraft. Security Tip: Avoid Detection with nmap Port Scan Decoys. http://www.cyberciti.biz/tips/nmap-hide-ipaddress-with-decoy-ideal-scan.html.
[22] Paganini, P. OVH hosting hit by 1Tbps DDoS attack, the largest one ever seen. http://securityaffairs.co/wordpress/51640/cyber-crime/tbps-ddos-attack.html.
[23] Park, K., and H.Lee. On the Effectiveness of Route-Based Packet Filtering for Distributed DoS Attack Prevention in Power-Law Internets. In *Proc. of ACM SIGCOMM* (2001).
[24] Peng, T., Leckie, C., and Kotagiri., R. Detecting Reflector Attacks by Sharing Beliefs. In *Proceedings of the IEEE Global Communications Conference* (2003).
[25] Perrig, A., Song, D., and Yaar, A. StackPi: A New Defense Mechanism against IP Spoofing and DDoS Attacks. Tech. Rep. CMU-CS-02-208, CMU Technical Report, February 2003.
[26] Piscitello, D. M. Anatomy of a DNS DDoS Amplification Attack. http://www.corecom.com/external/livesecurity/dnsamplification.htm.
[27] Rexford, J., Wang, J., Xiao, Z., and Zhang, Y. BGP Routing Stability of Popular Destinations. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment* (2005).
[28] Rossow, C. Amplification hell: Revisiting network protocols for ddos abuse. In *Network and Distributed System Security Symposium, NDSS* (2014).
[29] RouteViews.org. BGP Core Routing Table Size. http://www.routeviews.org/dynamics/.
[30] Snoeren, A. C., Partridge, C., Sanchez, L. A., Jones, C. E., Tchakountio, F., Kent, S. T., and Strayer, W. T. Hash-Based IP Traceback. In *Proc. of ACM SIGCOMM* (2001).
[31] Snoeren, A. C., Partridge, C., Sanchez, L. A., Jones, C. E., Tchakountio, F., Schwartz, B., Kent, S. T., and Strayer, W. T. Single-packet IP Traceback. *IEEE/ACM Trans. Netw. 10*, 6 (2002), 721–734.
[32] Subramanian, L., Roth, V., Stoica, I., Shenker, S., and Katz, R. H. Listen and whisper: security mechanisms for BGP. In *1st Symposium on Networked Systems Design and Implementation* (2004).
[33] Times, T. N. Y. How the Cyberattack on Spamhaus Unfolded. http://www.nytimes.com/interactive/2013/03/30/technology/how-the-cyberattack-on-spamhaus-unfolded.html.
[34] Touch, J. Defending TCP Against Spoofing Attacks. IETF RFC 4953, 2007.
[35] US-CERT. Multiple DNS implementations vulnerable to cache poisoning. VU#800113, 2008.
[36] Wang, A., Huang, C., Li, J., and Ross, K. W. Queen: Estimating packet loss rate between arbitrary internet hosts. In *PAM* (2009), vol. 5448, pp. 57–66.
[37] Wang, D., Xu, M., and Yang, J. On the scalability of router forwarding tables: Nexthop-Selectable FIB aggregation. In *Proc. of INFOCOM* (2011).
[38] Weigle, M. C., Adurthi, P., Hernandez-Campos, F., Jeffay, K., and Smith, F. D. Tmix: A Tool for Generating Realistic TCP Application Workloads in NS-2. *SIGCOMM Computing Communications 36*, 3 (2006), 65–76.
[39] Xie, Y., Yu, F., Goldszmidt, M., and Wobber, T. How dynamic are ip addresses. In *Proceedings of the ACM SIGCOMM Conference* (2007).
[40] Yang, X., Wetherall, D., and Anderson, T. A DoS-limiting network architecture. In *Proc. of ACM SIGCOMM* (2005), pp. 241–252.
[41] Zhang, J., Durumeric, Z., Bailey, M., Karir, M., and Liu, M. On the Mismanagement and Maliciousness of Networks. In *Network & Distributed System Security Symposium* (2013).