StackPi: A New Defense Mechanism against IP Spoofing and DDoS Attacks

Adrian Perrig   Dawn Song   Abraham Yaar

20 December 2002
Update: 5 February 2003
CMU-CS-02-208

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Today's Internet hosts are threatened by IP spoofing attacks and large scale Distributed Denial-of-Service (DDoS) attacks. We propose a new defense mechanism, *StackPi*, which unlike previous approaches, allows the host being attacked, or its upstream ISP, to filter out attack packets and to detect spoofed source IP addresses, on a per-packet basis.

In StackPi, a packet is marked *deterministically* by routers along its path towards the destination. Packets traveling along the same path will have the same marking so that an attack victim need only identify the StackPi marks of attack packets to filter out all further attack packets with the same marking. In addition, the victim can associate StackPi marks with source IP addresses to detect source IP address spoofing by changes in the corresponding StackPi mark.

StackPi filtering can thus defend against not only DDoS attacks, but also many IP spoofing attacks - such as TCP hijacking, and multicast source spoofing attacks. Because each complete mark fits within a single packet, the StackPi defense responds quickly to attacks and can be effective after the first attack packet in a IP spoofing attack, or after a small number of attack packets in the case of a DDoS attack. StackPi also supports incremental deployment, such that significant benefits are realized even if only one third of Internet routers implement StackPi marking. We show these results through analysis and simulations based on several real Internet topologies.

Keywords: DDoS, Denial-of-service, DoS, DoS defense, Internet, IP address spoofing, packet marking, path identifier.

1 Introduction

1.1 IP Spoofing and DDoS Attacks

Internet security is becoming of critical importance in today's computing environment, as our society, government, and economy is increasingly relying on the Internet. Unfortunately, the current Internet infrastructure is vulnerable to attacks — in fact, malicious attacks on the Internet have increased in frequency and severity. Large scale Distributed Denial-of-Service (DDoS) attacks disrupt critical Internet services and cause significant financial loss and operational instability. For example, the February 2000 DDoS attacks brought down some of the largest sites on the Internet including Yahoo! and eBay for hours. In October 2002, a DDoS attack flooded the root DNS servers with traffic aiming to deprive the Internet of the DNS name lookup service (which would have paralyzed the majority of Internet applications). The attack brought down eight out of thirteen root servers [25], and a slightly longer attack would have had devastating effects on Internet connectivity.

One of the most difficult challenges in defending against DDoS and many other attacks is that attackers often spoof the source IP address of their packets and thus evade traditional packet filters. Unfortunately, the current routing infrastructure cannot detect that a packet's source IP address has been spoofed or from where in the Internet a spoofed IP packet has originated from. The combination of these two factors makes IP spoofing easy and effective for attacks. In fact, many different types of Internet attacks utilize spoofed IP addresses for different purposes:

- DDoS attacks such as TCP SYN flooding use spoofed source IP addresses, because spoofed source IP addresses break filtering mechanisms that are based on source IP addresses [7]. As source address based filtering mechanisms become more widely deployed, DDoS attacks using IP spoofing will become even more popular and difficult to combat with existing techniques.

- DDoS reflector attacks, as shown in Figure 1, (see [29] for a good description) use IP spoofing, where the attacker machines do not send traffic directly to the victim, rather they first send traffic to some intermediate servers using the (spoofed) victim's IP address as the source IP address in the packets. The intermediate servers are unable to detect that the packets they are receiving have spoofed source IP addresses, so they simply reply to whatever source address is in the packet, namely, the victim's. When properly coordinated, attackers can cause a flood of packets to hit the victim without ever sending a single packet directly to the victim itself. Also, the intermediate servers often amplify the attack traffic: for example, in DNS server based reflector attacks, attackers send short DNS lookup requests (50 bytes each), whose replies can be over a thousand bytes long, thus giving the attacker a 20 time amplification of the traffic it could otherwise attack with directly. Other popular reflectors are Internet game servers, where attackers can use similar methods to gain two orders of magnitude of traffic amplification [21].

- In a TCP hijacking attack (see [8, 19]), an attacker can inject malicious data into a TCP connection and potentially even hijack the connection if it knows the IP address, TCP port number, and correctly guesses the sequence numbers being used. Consider a *telnet* session, where the attacker inserts the Unix command `rm -rf /`, which would delete all files of the current user. Another form of this attack is for the attacker to send TCP RST packets to close an existing connection. Such attacks can be particularly disruptive if the TCP connection is a BGP session between two routers.

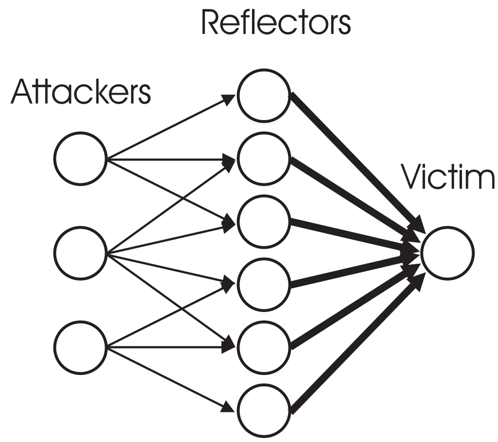


Figure 1: Example of a reflector attack. Reflectors are often used as traffic amplifiers to attack the victim.

- Port Scanning attacks use spoofed source IP addresses to hide the identity of the attackers. For example, in Nmap port scanning, when an attacker's machine probes a port on the victim's machine, the attacker not only sends a probe packet with its own source IP address, but also sends many fake probe packets with other (often random) source IP addresses. Thus, even when the victim realizes that it is being probed, it cannot figure out exactly which host is probing it.

1.2 Desired Properties of Defense Mechanisms against IP Spoofing and DDoS Attacks

Because the current Internet routing infrastructure has few capabilities to defend against IP spoofing and DDoS attacks, we need to design a new defense mechanism against these attacks. In particular, a good solution to defend against these attacks should satisfy the following properties:

- Fast response: The solution should be able to rapidly respond and defend against attacks. Every second of Internet service disruption causes economic damage. We would like to immediately block the attack.

- Scalable: Some attacks, such as TCP hijacking, involve only a small amount of packets. However, many DDoS attacks are large scale and involve thousands of distributed attackers and an even larger number of attack packets. A good defense mechanism must be effective against low packet count attacks but scalable to handle much larger ones.

- Victim filtering: Almost all DDoS defense schemes assume that once the attack path is revealed, upstream routers will install filters in the network to drop attack traffic. This is a weak assumption because such a procedure may be slow, since the upstream ISPs have no incentive to offer this service to non-customer hosts and networks. We observe that because large Internet servers are processing, not bandwidth, constrained, the servers themselves should be responsible for filtering out attack traffic that reaches them.

- Per-packet filtering: A defense mechanism against IP spoofing attacks should allow the victim to identify each individual spoofed IP packet so that legitimate packets can still be accepted. It is also important that a DDoS defense mechanism operate on a per packet basis so that the attack victim can participate in filtering traffic that it receives.

- Efficient: The solution should have very low processing and state overhead for both the routers in the Internet and, to a lesser degree, the victims of the attacks.
- Support incremental deployment: The solution is only useful and practical if it provides a benefit when only a subset of routers implement it. As an increasing number of routers deploy the scheme, there should be a corresponding increase in performance.

Also, the deployment of the solution should not leak proprietary information about an ISP's internal network, as some ISPs keep their network topology secret to retain a competitive advantage.

1.3 Overview of Our Approach: StackPi

Even though there has been much research done in defending against Internet attacks such as IP spoofing and DDoS attacks, previous solutions have several shortcomings. Researchers have proposed ingress filtering against IP address spoofing [12], but this approach requires deployment on all edge routers, to be effective. Most of the previous research focused on the problem of IP traceback, which enables the victim to reconstruct the IP addresses of the routers along the path of an attack flow. However, even with this approach, once the victim has reconstructed the router path, it has to rely on contacting upstream routers to filter traffic. Furthermore, many IP traceback schemes require large numbers of packets to reconstruct the entire attack graph, and thus have a very slow response time. Because these approaches work over multiple packets they cannot be used to detect IP address spoofing on a per packet basis.

In this paper, we propose a new approach, called *StackPi* (short for *Stack Path Identifier*), which is the first defense mechanism that satisfies all of the above desired properties. In StackPi, as a packet traverses routers on the path towards its destination, the routers deterministically mark bits in the packet's IP Identification field. The deterministic markings guarantee that packets traveling along the same path will have the same marking. StackPi allows the victim and routers on the attack path to take a proactive role in defending against a DDoS attack by using the StackPi mark to filter out attack packets on a per packet basis. In addition, the victim can build statistics over time relating StackPi marks to IP addresses. Then if an attacker spoofs an IP address, it is likely that the StackPi mark in the spoofed packet will not match the StackPi mark corresponding to the legitimate IP address in the database, thus enabling the victim to tag packets with possibly spoofed source IP addresses.

StackPi is not only effective against large scale DDoS attacks, but also effective against other IP spoofing attacks such as TCP hijacking and multicast source spoofing attacks. Our scheme is extremely efficient and responds quickly to attacks. StackPi can be effective for the first packet in an IP spoofing attack, or after a few packets in the case of a DDoS attack. Our scheme supports incremental deployment and offers significant defense capabilities even if only one third of the routers in the Internet implement it. We show the effectiveness of StackPi through simulation results based on several real-world Internet topologies.

The remainder of this paper is organized as follows: In Section 2, we describe the design of both the marking and filtering parts of our proposed defense mechanism, StackPi. In Section 3 we present StackPi's performance using both theoretical analysis and simulation results. In Section 4 we discuss some issues and potential extensions to the StackPi scheme. In Section 5 we describe related work. Section 6 concludes the paper.

2 StackPi: A New Defense Mechanism Against IP Spoofing and DDoS Attacks

The StackPi defense mechanism is composed of two parts: the *marking* scheme and the *filtering* scheme. The marking scheme defines how StackPi enabled routers mark all the packets that they forward. We describe the design of this scheme in Section 2.1. The filtering scheme defines how end hosts utilize the packet markings to most efficiently defend against DDoS and IP spoofing attacks. Unlike the marking scheme, which must be consistent at every StackPi enabled router, the filtering scheme can change from end-host to end-host, depending on the host's constraints and the attack properties. We describe several different filtering strategies in Section 2.2.

2.1 StackPi Marking Design

The intuition behind the StackPi marking scheme is simple: in order to generate a path identifier that is representative of a particular path from a source to a destination in the Internet, each router along the path must contribute some small amount of information whose aggregate among the routers of the path will be the StackPi marking. We store the StackPi mark in the IP Identification field of the IP header. [1] Because the IP Identification field is only 16 bits long, we are limited to using only 16 bits for our marking, thus allowing for 2^{16} unique StackPi marks. Although there are likely more than 2^{16} possible paths from any end-host in the Internet to our DDoS victim, we believe (and later show) that this level of granularity is sufficient to allow the victim of an attack to accurately filter out attack packets, while still maintaining good service for legitimate users.

2.1.1 The Basic Marking Scheme

The StackPi marking scheme is simple in concept: each router treats the IP Identification field as though it were a *stack*. Upon receipt of a packet, a router shifts the IP Identification field (hereon referred to as the *marking field*) of the packet's header to the left by n bits, and writes a precalculated set of n bits (represented by the marking m) into the least significant bits that were cleared by the shifting. This is the equivalent of *push*ing a marking onto the stack. Every following router in the path does the same until the packet reaches its destination. Because of the finite size of the marking field, after $\lfloor 16/n \rfloor$ routers have pushed their markings onto the marking field, additional markings simply cause the oldest markings (the ones pushed first onto the stack) to be lost. The packet's StackPi mark is merely the concatenation of all the markings in the marking field when the packet arrives at its destination. Because routers always push their markings onto the least significant n bits of the marking field, their markings will always appear in the same order; and because every router's bit markings are precalculated, each StackPi marking is deterministic — packets that follow the same path will have the same marking.

Before we present improvements to this basic marking scheme, we will first explain how each router's n marking bits are calculated. Because each StackPi marking is supposed to identify a particular path; which is characterized uniquely by the list of IP addresses of the routers along that route, we would like the markings of each router to be a function of the IP addresses of the routers in the path in order to maximize the likelihood of the StackPi marking being unique to a path. We would also like the markings

[1] This field is traditionally used in packet fragmentation. Because a very low percentage of Internet traffic uses fragmentation, its use for packet marking has been shown in the literature to have small effect on Internet traffic [32]. We discuss fragmentation compatibility with StackPi in Section 4.5.

to be uniformly distributed within the marking space, such that as few paths as possible map to the same marking. To achieve both of these goals, our initial choice for the marking function at each router was simply the last n bits of the hash (e.g., MD5 hash [30]) of that router's IP address. The problem with this marking strategy is that it does not distinguish between two closely related paths with sufficient probability. This problem becomes evident in a fan-in situation, where two different routers (*A* and *B*) both forward their packets to the same router (*C*) at the next hop. In this situation, using our basic marking scheme, the markings for the two paths *AC* and *BC* will be $m_A \mid\mid m_C$ and $m_B \mid\mid m_C$, respectively (where '||' stands for concatenation). Since m_C is common to both paths' markings, there is a $1/2^n$ probability that m_A and m_B will be identical, and hence that paths *AC* and *BC* will be indistinguishable. To correct this, we begin with the assumption that each router is aware of the IP address of the last-hop host or router which sent it a packet. If we add this information to each router's marking, such that the marking for the x'th hop of a path, m_x, equals the MD5 hash of its IP address, IP_x, concatenated with the last-hop's IP address, IP_{x-1} – then the markings for the two paths *AC* and *BC* become $m'_A \mid\mid m_{C1}$ and $m'_B \mid\mid m_{C2}$, respectively. m'_A and m'_B still have a $1/2^n$ probability of colliding at the same value, but now the marking for router *C* has changed such that $m_{C1} = MD5(IP_C \mid\mid IP_A)$ and $m_{C2} = MD5(IP_C \mid\mid IP_B)$, which also have a $1/2^n$ probability of colliding with each other, independent of m_A and m_B. Thus, the new probability of having identical marks for two paths joining at the same node using our last-hop IP knowledge has dropped by a factor of $1/2^n$ and is now $1/2^{2n}$. This approach marks each router's incoming link, instead of its IP address. Figure 2 shows this basic StackPi marking scheme.

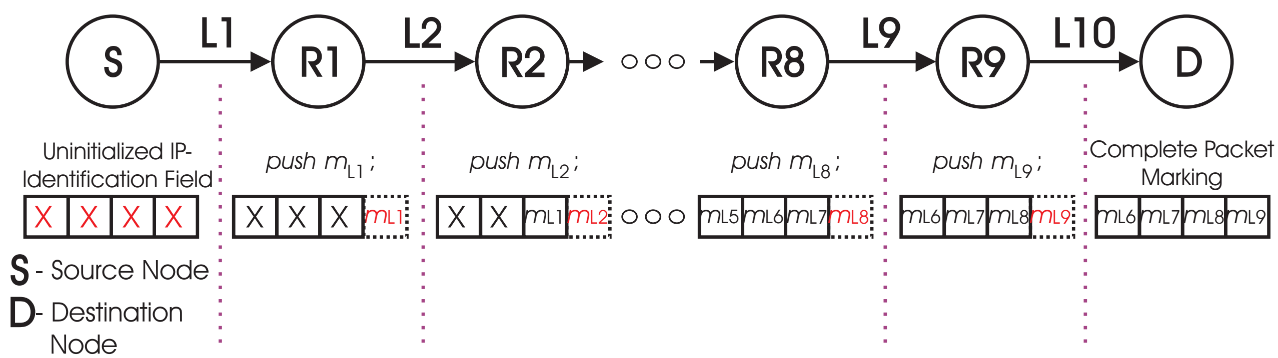


Figure 2: The basic StackPi Marking Scheme. This figure shows how the StackPi mark evolves as the packet traverses routers R1 through R9. Initially, the marking field contains arbitrary data. In this example, the field has space for four router markings. Each router marks the incoming link.

Because of the finite size of our marking field, a StackPi mark with $n = 2$ for each router will only capture information from the last 8 hops of a path. Since routers local to the victim do not usually add useful path information, but erase distinctive markings from routers further away; we propose that the local routers do not perform StackPi marking on packets destined for the victim. Figure 3 shows that on average, most path lengths are between 10 to 15 hops. Usually, local routers constitute 2 to 5 hops (from sample traceroutes we performed to large web-servers). So choosing n=2 for our marking scheme is likely to include the marks from routers close to the origin of the attacker network, provided that local routers do not mark.

2.1.2 The Extended Marking Scheme: Write Ahead to Deal With Legacy Routers

Another important issue is the interaction of StackPi with legacy routers that do not implement the marking scheme. Because only StackPi enabled routers will mark bits into the marking field, any legacy routers

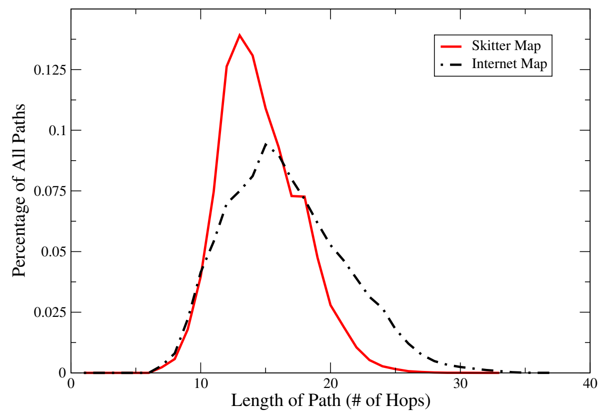


Figure 3: Distribution of Internet path lengths using the Skitter map and the Internet map

along the path will not contribute to that path's StackPi marking. However, we can modify our stack-based scheme slightly to improve this situation. If we assume that each router knows the IP address of the next-hop router or host to which it is forwarding a packet, then the router is capable of marking its outgoing link (which is the incoming link of the following router) as well as its incoming link. So if the next-hop router after a StackPi enabled router is a legacy router, its incoming link will already be marked as the outgoing link of the StackPi router, and the next-hop router after the legacy router will mark the legacy router's outgoing link (assuming it is a StackPi router) as its own incoming link. We modify the basic marking scheme as follows: instead of each router simply pushing a packet's incoming link mark into the marking field, each router pushes the mark for a packet's incoming link and outgoing link. As we have shown, each router's markings depend on both its own IP address and the IP address of the last-hop router. Each router, then, has enough information to mark for the incoming link of the next-hop router simply by using the next hop's IP address as the "current" IP address and its own IP address as the "last-hop" IP address. This improvement allows us to include the marking of the first legacy router to appear after a StackPi-enabled router, because that StackPi-enabled router will mark on-behalf of the legacy router.

Unfortunately, in the case of non-legacy routers, this improvement halves the space efficiency of the marking scheme because each router's marks are included twice. However, if we return to the assumption that we made in improving our marking bits (that each router knows the IP address of the last-hop host or router) we can further change our scheme so that it is just as space efficient as the basic marking scheme. Upon receipt of a packet, a router *peeks* (the process of looking at the top of the stack without modifying it) at the topmost marking of the stack and compares it to what its own incoming link marking should be. If the markings are identical, then the router will assume that the last hop router has already marked the incoming link on its behalf and will only push the outgoing link marking into the marking field. If the topmost marking is different from what the router would mark for its incoming link, then the router will assume that the last-hop router was a legacy router and will push its incoming link marking onto the stack as well as its outgoing link marking. We call this scheme router *write-ahead* and we adopt it for our StackPi marking scheme as illustrated in Figure 4. This mechanism errs in one case: if the incoming and outgoing link marking of a legacy router are equal, the two adjacent routers only mark for one link of the legacy router. This only happens with $\frac{1}{2^n}$ probability, and when it does happen, the write-ahead scheme performs exactly as if it were not applied at all. Therefore, the write-ahead scheme is a strict improvement

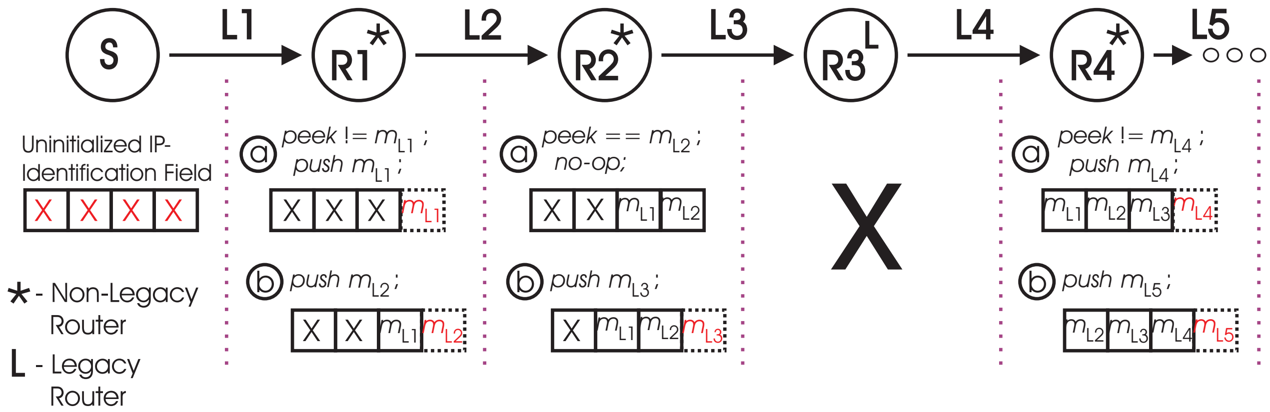


Figure 4: The StackPi marking scheme with write-ahead. The new scheme allows the inclusion of markings from router R3, despite the fact that it is a legacy router. Each router along the path first (a) checks the topmost marking in the stack to see if it equals the router's incoming link, and if it doesn't, the router adds its incoming link to the marking; and then (b) adds the incoming link of the next-hop router to the stack.

over our basic marking scheme. [2]

2.2 Packet Filtering

In this section, we explain how a DDoS victim can use the StackPi mark to defend against a DDoS attack. StackPi is one of the first schemes that allow the victim to take a proactive role in filtering out attack packets. [3] Because StackPi allows for per-packet filter decisions and is geared to defend against DDoS attacks and IP spoofing attacks, it is extremely important that the filters at the end-host have a low per-packet computation cost, as an end-server will need to be able to filter every packet that arrives over the network. In this section, we present two efficient packet filtering methods. Note that the filtering mechanism is not limited to deployment only at the victim. To prevent last-hop bandwidth starvation DDoS attacks, StackPi filters can be deployed at the victim's ISP. In fact, we show later in Section 4.4 that StackPi filters can be deployed throughout the Internet to enhance the filtering capability of upstream routers in schemes such as Pushback [17, 23] and SPIE [33, 34].

2.2.1 Using Only StackPi Markings (StackPi Filter)

The simplest filtering strategy for an end-host under attack is to build a list of the StackPi markings of offending packets and compare each incoming packet's marking to that list; if there is a match, then the incoming packet is assumed to be an attack packet and is dropped without further processing.

The problem with this simplified filter is that it is vulnerable to *marking saturation* — that is, it is possible for an attacker to DoS legitimate users with the same StackPi marking by causing a server to add the attacker's StackPi marking to its drop list, even if the attacker's traffic is only a marginal portion

[2] If we build a mechanism such that a router could detect that its predecessor is a legacy router, then that router could simply add the legacy router's marking without performing any comparison. Such a mechanism could be as simple as noting a variation, between packets, in the least-significant bits of the incoming StackPi mark (because a non-legacy router will *always* mark the incoming link of the next-hop router). With this detection mechanism, there is no longer a $\frac{1}{2^n}$ probability of missing a legacy router.

[3] In an independent work, Sung and Xu also propose victim filtering [38]. We review their approach in Section 5.

of the total traffic with that marking. Theoretically, a distributed attack with one attacker at each possible StackPi mark could accomplish this, although this would require a large number of topologically distributed attackers.

The marking saturation problem can be solved with a slightly more complex, threshold based filter. The intuition behind threshold filters is that it may be in the victim's best interest to accept a small number of attack packets if that allows it to accept a large number of legitimate users' packets. More formally, we define a filter that flags incoming packets as either *positive*, if they are suspected attack packets and *negative* if they are not. A threshold filter allows a server to decrease the *false positive* rate (the rate at which legitimate traffic is dropped due to collision with known attacker markings) at the expense of increasing the *false negative* rate (the rate at which attack packets are accepted by the server). In a threshold based filter, the server calculates a value t_i for each possible StackPi mark, i, given the number of user and attack packets with StackPi mark i, u_i and a_i, respectively. If the percentage of user packets to all packets with StackPi mark i is such that:

$$\frac{u_i}{a_i + u_i} < t_i$$

then the victim drops all packets with StackPi mark i. Threshold filtering prevents the trivial case where a single attacker node can cause all the nodes that share a StackPi mark with it to have their packets dropped at the filtering server. Now, attack traffic must present a significant component (adjustable by setting t_i) of traffic with a particular marking, before packets bearing that marking are dropped.

2.2.2 Using StackPi Marking and the Source IP Address for Filtering (StackPi-IP Filter)

The StackPi filtering presented in the previous section is extremely light-weight and efficient. In this section, we present a slightly more complex but more accurate filtering method. The filter itself is simple: examine an incoming packet's StackPi mark *and* source IP address and allow access based on that tuple. Ideally, a database of legitimate users' $\langle$StackPi, IP$\rangle$ tuples will be built during times when there are few or no ongoing attacks. Any packet with a StackPi marking that does not match the StackPi marking of the same IP address in the database will be flagged as a packet with a spoofed IP address.

In the case of filtering based only on StackPi markings (as we describe in Section 2.2.1), we have to assume that our filters get feedback from some higher layer algorithm that can classify some sampled packets as legitimate packets or attack packets, and tell the filter which StackPi markings correspond to attack traffic and should be dropped. The StackPi-IP filter does not rely as strongly on this assumption, because the StackPi-IP filter does not need to be bootstrapped with attack traffic. Quite the opposite, the StackPi-IP filter is bootstrapped during non-attack periods and identifies attack periods by an increase in the incidence of packets with spoofed IP addresses.

3 Analysis and Evaluation of the StackPi Mechanism under DDoS Attack

We have presented the design of the StackPi marking scheme, as well as two StackPi filtering techniques that use packet markings to filter out attack packets. In this section, we evaluate these mechanisms under a simulated DDoS attack. We first explain our DDoS attack model in depth, in Section 3.1. We then derive an optimal value for the StackPi *theshold filter* and evaluate its performance under DDoS attack in Section 3.2. We evaluate the effect of resource constraints in the victim on the StackPi filter's performance in Section 3.3. We next evaluate the effect of legacy routers on the StackPi filter's performance in Section 3.4. We also present an analysis of the StackPi-IP filter's performance in Section 3.5.

3.1 DDoS Attack Model

In order for a DDoS victim to protect itself against attack packets, it must have a way to differentiate them from legitimate users' packets. Although StackPi filtering provides such a method, the filter must be initialized with the StackPi marks of known attack packets. It is outside the scope of this paper to define an algorithm for attack packet identification. [4] In order to reflect the use of such an algorithm to bootstrap our filter, we have modeled our DDoS attack simulation in two phases. In the first phase, the *learning phase*, all packets are assumed to be analyzed by the victim, using some packet identification algorithm. In other words, the victim is given the power to differentiate between attack and user packets that arrive in the learning phase. Thus, the victim is able to generate an attacker StackPi markings list. In the second phase, the *attack phase*, we assume that the victim is no longer able to automatically differentiate attack and user packets, and thus is forced to rely on StackPi filtering to make accept and drop decisions. The learning phase of our DDoS attack lasts for 16 packets and the attack phase lasts for 10 packets. All our results are taken from the attack phase of the DDoS attack.

For our experiments, we use sampled Internet topologies from Burch and Cheswick's Internet Mapping Project [4, 16] and from the Skitter Project distributed by Caida [6]. Both datasets are constructed by having a single host send `traceroute` probes to a large number of destinations, listing all router IP addresses on the resulting paths. We filtered the data sets to remove all incomplete and duplicate routes (although multiple routes to the same end-host were not removed). We also removed all routes of path length shorter than seven hops.[5] In our simulated attacks, the single traceroute host is used as the DDoS victim server.

Our DDoS simulations proceed as follows: a certain number of *paths* are selected, at random, from the topology file and assigned to be either attack or legitimate user paths.[6] Unless otherwise noted, all of the DDoS simulations select 10,000 paths for legitimate users and vary the number of attackers. Because compromised machines and dedicated attackers generally send more traffic to the victim during a DDoS attack, in our simulations, we assume that each attacker sends 10 times more packets than each user. When we refer to the duration, in packets, of a phase of the DDoS attack, we do it from the users' point of view. Thus, in the learning phase, each user sends 16 packets and each attacker sends *160* packets. Likewise, for the attack phase, each user sends 10 packets and each attacker sends *100* packets. Our results are the averages of 3 runs of an attack with a particular topology and particular set of parameters.

3.2 Threshold Filtering Performance in StackPi

In the first experiment, we quantify the effectiveness of the StackPi threshold filter. Recall from Section 2.2 the *false positive* and *false negative* rates, which represent the rate at which legitimate user's packets are

[4] Jung, Krishnamurthy and Rabinovich further discuss this problem [20].

[5] Although this step will slightly bias the experimental results in favor of the StackPi scheme, the number of paths removed is insignificant compared to the total number of paths in either of the data sets.

[6] Note that because we are selecting paths rather than end-hosts, there is the possibility that an end-host with multiple paths in the topology file will be selected as an attacker and as a legitimate user. This situation is more realistic because it is possible for a user's machine to be compromised without their knowledge.

dropped and the rate at which attacker packets are accepted, respectively. For the purpose of our evaluation, we refer to the following two metrics: the *user acceptance ratio*; which is 1 minus the false positive rate, and the *attacker acceptance ratio*; which is exactly the false negative rate. We define these two metrics in terms of the following simulation variables:

p_j - The total number of packets sent by entity j

v_j - The total number of packets sent by j accepted by the victim.

The acceptance ratio, a_j, for a given entity j is defined as:

$a_j = \frac{v_j}{p_j}$

Thus, for the set of all users, U, and all attackers, A, the acceptance ratios are defined as $a_U = \frac{v_U}{p_U}$ and $a_A = \frac{v_A}{p_A}$ for the users and attackers, respectively.

Under a DDoS attack, the victim would like to maximize the user acceptance ratio and minimize the attacker acceptance ratio. As we discuss in Section 2.2, the two acceptance ratios are correlated in threshold filtering schemes, as a decrease of one can result in the increase of the other. The goal of the victim, then, is to maximize the difference between the two ratios:

$$\Delta = \frac{v_U}{p_U} - \frac{v_A}{p_A}$$

which we refer to as, Δ, the *acceptance ratio gap*. The acceptance ratio gap is a useful metric to determine how a particular StackPi filter performs relative to no filter. Without StackPi marking and filtering, the victim can only make accept/drop decisions at random, which intuitively gives an equal user and attacker acceptance ratio, or an acceptance ratio gap of zero.

In order to maximize the acceptance ratio gap, we must find an optimal threshold value, which is the ratio of user traffic to total traffic at a particular StackPi mark below which all packets arriving with that StackPi mark are dropped by the victim. We derive this optimal value as follows:

Let v_{j_i} and p_{j_i} equal the number of packets accepted by the victim and the total number of packets sent by entity j, *with StackPi mark i*, respectively.

The acceptance ratio gap, Δ, is then:

$$\Delta = \frac{v_U}{p_U} - \frac{v_A}{p_A} = \frac{\sum_{i=0}^{2^{16}-1} v_{U_i}}{p_U} - \frac{\sum_{i=0}^{2^{16}-1} v_{A_i}}{p_A} = \sum_{i=0}^{2^{16}-1} \left(\frac{v_{U_i}}{p_U} - \frac{v_{A_i}}{p_A} \right)$$

When the ratio of user traffic to attacker traffic at a particular StackPi mark is above the threshold value, then all packets arriving with that StackPi mark are accepted. Thus, we can introduce our threshold filtering function, f_i, which will return 1 if the user traffic ratio at StackPi mark i is above the threshold and 0 if it is below. We can now define the packets accepted by the victim at a particular StackPi mark in terms of the total packets arriving at that mark:

$$v_{U_i} = f_i \cdot p_{U_i}$$

$$v_{A_i} = f_i \cdot p_{A_i}$$

We now include these definitions in our acceptance gap equation:

$$\Delta = \sum_{i=0}^{2^{16}-1} \left(\frac{f_i \cdot p_{U_i}}{p_U} - \frac{f_i \cdot p_{A_i}}{p_A} \right) = \sum_{i=0}^{2^{16}-1} f_i \left(\frac{p_{U_i}}{p_U} - \frac{p_{A_i}}{p_A} \right)$$

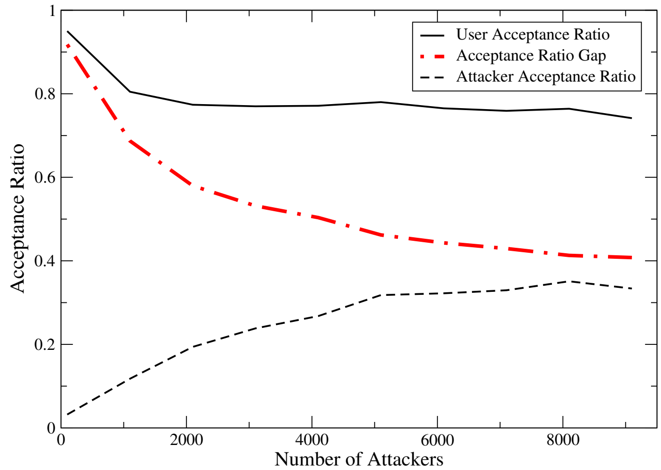


Figure 5: Acceptance ratios and gap with optimal threshold filter using the Internet map.

To maximize Δ, it is clear that we must only accept packets with StackPi mark i where the ratio of user packets with StackPi mark i to the total number of user packets, $\frac{p_{U_i}}{p_U}$ is greater than the ratio of attack packets with StackPi mark i to the total number of attack packets, $\frac{p_{A_i}}{p_A}$. In terms of our filtering function f_i:

$$f_i = \begin{cases} 1 & \text{if } \frac{p_{U_i}}{p_U} < \frac{p_{A_i}}{p_A}, \\ 0 & \text{otherwise.} \end{cases}$$

Since our threshold is expressed as the ratio of user traffic to attack traffic at a particular StackPi mark, we simply manipulate the terms of our filtering function to obtain the optimal threshold:

$$f_i = \begin{cases} 1 & \text{if } \frac{p_{U_i}}{p_{A_i}} < \frac{p_U}{p_A}, \\ 0 & \text{otherwise.} \end{cases}$$

$$t_{opt} = \frac{p_U}{p_A}$$

This result indicates that in order to maximize the acceptance ratio gap, unless the ratio of user traffic to attack traffic at a particular StackPi mark is greater than the ratio of user traffic to attack traffic *over all StackPi marks*, then all packets bearing that marking should be dropped. Because we prefer to deal with thresholds as percentages, we normalize our optimal threshold value to be:

$$t_{opt} = \frac{p_U}{p_A + p_U}$$

To calculate the value of the threshold and determine whether to accept or drop packets with a particular StackPi mark, the victim uses the the information from packets in the learning phase of the DDoS attack. Figure 5 shows the performance of the StackPi threshold filter. It is clear from the figure that StackPi provides strong protection to the DDoS victim, allowing it to accept over 70% of legitimate user traffic, while rejecting almost 70% of attacker traffic, even when the attack traffic is ten times that of the user traffic.

3.3 Effect of Victim Resource Constraints on StackPi Filter Performance

In the previous section, we derived the optimal threshold value for a StackPi threshold based filter. Such an analysis is useful, but lacks realism because there are no constraints imposed on the DDoS victim server. Specifically, it does not take into account the victim's packet processing capacity. In this section, we impose a packet processing constraint on the victim server. We assume that the server is carefully (or perhaps, irresponsibly) provisioned to handle *exactly* the anticipated amount of legitimate user traffic.

When the server has a limited capacity, the goal during DDoS attack is no longer to maximize the acceptance ratio gap, but rather to maximize the utilization of the server by legitimate user's packets. Assume that the server's packet capacity is denoted as C and that the user and attack traffic are denoted as p_U and p_A, respectively. Without any StackPi marking or filtering, the victim would only be able to accept packets at random from the network, up to its full capacity. Thus, the expected value of the percent utilization of the server by legitimate user's packets, γ, becomes:

$$E[\gamma] = \frac{\frac{C \cdot p_U}{p_A + p_U}}{C} = \frac{p_U}{p_A + p_U}$$

With a finite packet capacity, the victim's best strategy is to accept all packets with the StackPi marking that exhibits the highest percent of user traffic to total traffic, then to accept all packets from the marking with the next highest percentage, and so on, until the victim reaches its full capacity. Let b_i be the StackPi mark that has the i-th highest percentage of user traffic to total traffic with StackPi mark i. Let m be the maximum integer, such that:

$$\sum_{i=0}^{m} (p_{U_i} + p_{A_i}) < C$$

Any remaining packet capacity, u, then becomes:

$$u = C - \sum_{i=0}^{m} (p_{U_i} + p_{A_i})$$

Thus, if the victim follows this strategy, its utilization will be:

$$\gamma = \frac{\sum_{i=0}^{m} (p_{U_i}) + \frac{p_{U_{m+1}} u}{p_{U_{m+1}} + p_{A_{m+1}}}}{C}$$

The strength of this result can be shown by taking the worst case scenario for this filter: that user and attacker packets are distributed uniformly across all possible StackPi markings. Formally:

$$p_{A_i} = \frac{p_A}{2^{16}} \quad (1 \leq i < 2^{16})$$

$$p_{U_i} = \frac{p_U}{2^{16}} \quad (1 \leq i < 2^{16})$$

Because each p_{A_i} and p_{U_i} is constant for all i's, we have:

$$\gamma = \frac{m \cdot p_{U_i} + \frac{p_{U_{m+1}} u}{p_{U_{m+1}} + p_{A_{m+1}}}}{C}$$

Without loss of generality, we can assume that u = 0, so that m divides C. With this simplification, we eliminate the second term of the fraction:

$$\gamma = \frac{m \cdot p_{U_i}}{C}$$

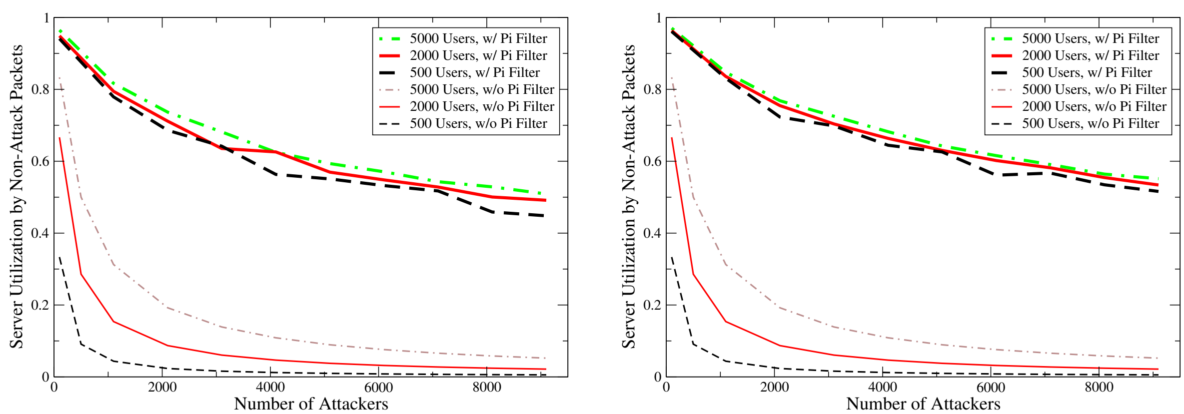
m is expressed as the server's capacity divided by the number of packets per StackPi mark:

$$m = \frac{C}{p_{U_i} + p_{A_i}} = \frac{C}{\frac{p_U}{2^{16}} + \frac{p_A}{2^{16}}} = \frac{2^{16} \cdot C}{p_U + p_A}$$

Substituting for m, we get:

$$\gamma = \frac{\frac{2^{16} \cdot C}{p_U + p_A} \cdot \frac{p_U}{2^{16}}}{C} = \frac{p_U}{p_A + p_U}$$

which is exactly the same result as the expected value for the random selection strategy. Thus, we have shown that even in the worst-case scenario, our StackPi filter reduces to the same effectiveness as random selection. However, as Figure 6 shows, the StackPi filter performance is considerably better than random selection. Even when attack traffic is 160 times user traffic (in the attack scenario of 500 users and 8000 attackers), 50% of the server's capacity is utilized for servicing legitimate user's packets when using the StackPi filter, while only 0.6% of the server's capacity is used to serve legitimate user's packets when the server uses a random selection strategy.



(a) Internet Map

(b) Skitter Map

Figure 6: Server utilization by legitimate user's packets under packet processing constraint in the Internet map and the Skitter map.

3.4 Legacy Router Analysis

Any scheme that requires modifications to Internet routers must be able to coexist and operate together in an environment with unmodified routers, or *legacy routers*. As we described in Section 2.1, legacy routers do not break our marking scheme, but when there are a high percentage of them in the network, there may not be enough StackPi routers on a given path to mark all the bits of the marking field, thus allowing attacker initialized bits to reach the victim. This is a problem because, presumably, the attacker will randomly select the bits that it uses to initialize the marking field, allowing it to randomly switch between different StackPi markings at the victim. Of course, using write-ahead marking in the marking scheme eliminates the effect of the first legacy router after a StackPi enabled router. However, a series of two or more consecutive legacy routers will result in one or more lost link markings.

We evaluated StackPi's performance for both metrics from the previous two sections with varying percentages of legacy routers in our topologies. Our methodology in selecting legacy routers is as follows: when each router is encountered in the simulation for the first time, it has a certain probability of being chosen to be a legacy router for the remainder of the simulation. This selection algorithm results in a uniform distribution of legacy routers throughout the topology. [7]

Figure 7 shows the effect of legacy routers on the acceptance ratio gap metric from Section 3.2. As expected, the decrease in the gap, per percentage of legacy routers, grows with the increase in the probability because single legacy routers are more likely to occur at low probabilities, and have no detrimental effect on the performance of the scheme. Figure 8 shows the effect of legacy routers on the server's utilization by legitimate users' packets. Both of these results show that even when 60-70% of the routers are legacy routers, the StackPi filter still provides considerable advantage over random filtering.

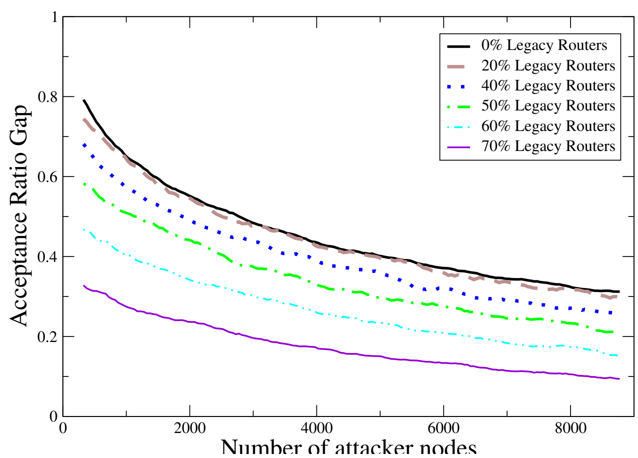


Figure 7: Acceptance ratio gap with a threshold of t=0.5 and a varying percentage of legacy routers using the Internet map.

3.5 Using the StackPi-IP Filtering Technique Against IP Spoofing Attacks

In this section, we review the effectiveness of the StackPi-IP filter, which uses the StackPi marking *and* IP address of each incoming packet to detect spoofing. The filter simply checks any incoming packet's StackPi mark and compares its IP address to a list of $\langle$IP address, StackPi mark$\rangle$ tuples to see if there is a match. Like the threshold filtering scheme, the StackPi-IP filter requires bootstrapping however, in this case, with packets bearing non-spoofed source IP addresses. Such a scenario is ideal for a server that has a static set of authorized users.

[7]Unfortunately, a uniform distribution of legacy routers may be unrealistic, since it is reasonable to assume that clumps of routers in a path - perhaps belonging to a particular organization - will be updated over a short period of time. A uniform distribution also biases the results in favor of our scheme because it is more likely that single legacy router, rather than a series of them, will appear in a uniform distribution. We are working on experiments that utilize different distributions to model the occurrence of legacy routers in the network topology.

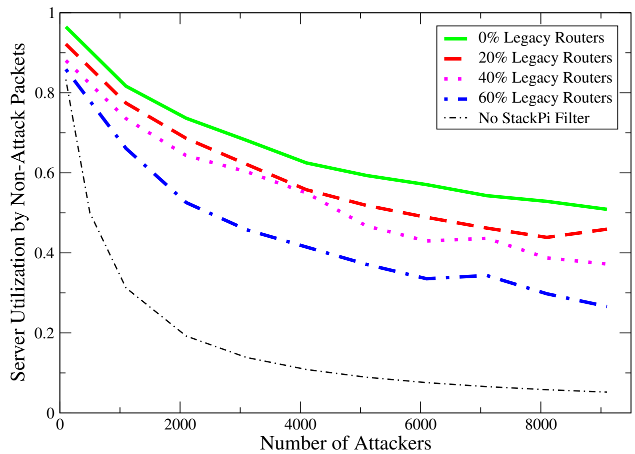


Figure 8: Server utilization by 5000 legitimate users under processing constraint and a varying percentage of legacy routers using the Internet map.

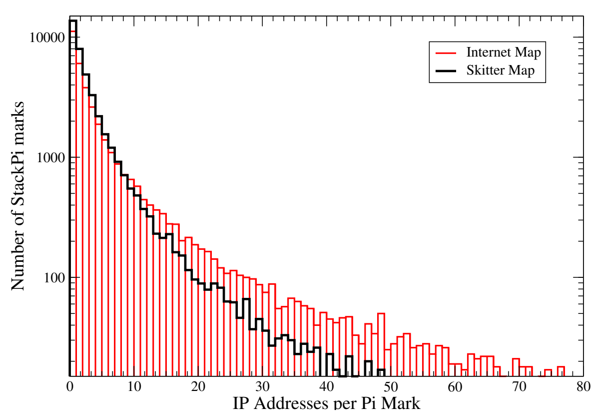


Figure 9: Histogram of the frequency of StackPi marks with a particular number of IP addresses that map to them.

The metric that best quantifies the performance of the StackPi-IP filter is the probability that a randomly selected attacker will be able to spoof an IP address that will be accepted by the victim. The only way for this to happen, is for an attacker to spoof the IP address of an end-host that happens to have the same StackPi mark as the attacker itself. This is hardest for the attacker when the IP addresses of end-hosts in the topology are distributed uniformly over the possible StackPi marks, because no StackPi mark has a large number of IP addresses that map to it and thus there are fewer IP addresses for that StackPi mark that will be accepted by the filter.

In this experiment, each end-host in the topology sends 10 packets with non-spoofed source IP addresses for the victim to bootstrap its filter. Figure 9 shows a histogram of the number of StackPi markings with a particular number of unique IP addresses that map to them (note that the y-axis is logarithmic), after the initial 10 packets have been sent. The histogram shows us that the IP addresses are somewhat uniformly distributed over the possible StackPi marks, with the large majority of StackPi marks having 1 to 4 unique IP addresses that map to them and very few StackPi marks with greater than 20 unique IP addresses that map to them.

Given this distribution, we can calculate the probability that a randomly chosen end-host from the topology will succeed in spoofing the IP address of an end-host with the same StackPi marking. Because we are dealing with a topology that only contains a small subset of all the possible legitimate IP addresses, we assume that our attacker has access to a list of the unique IP addresses of all end hosts in our topology and selects addresses from this list when spoofing a packet. We begin by calculating the probability that an attacker with a particular IP address, k, will succeed in spoofing a packet that will be accepted by the filter. This probability depends on how many StackPi markings have been recorded at the victim for address k. We define the set of n distinct StackPi markings recorded for address k as $\{m_0, m_1, \ldots, m_n\}$. For each StackPi mark recorded at the victim for IP address k, there is a set of other IP addresses that also map to the same StackPi mark. If the attacker were to spoof any of these, the attack packet would be accepted by the filter. Thus, the probability of an attacker with IP address k successfully spoofing is:

$$P_k = \frac{\sum_{i=0}^{n} uniqueIPs(m_i, k)}{N}$$

where the $uniqueIPs()$ function returns the number of unique IP addresses that map to StackPi mark m_i, excluding IP address k as well as any duplicates between function calls, and N represents the number of end-hosts in the topology; which is the size of the list of possible IP addresses that the attacker can spoof. Given the probability of an attacker with a specific IP address of successfully spoofing a packet, we can now calculate the probability of an attacker with a random IP address successfully spoofing:

$$P = \frac{\sum_{k=0}^{N} P_k}{N}$$

The lower bound for this probability occurs when the distribution of IP addresses over the StackPi marks is exactly uniform. In that case, the probability P of an attacker spoofing becomes $\frac{1}{2^{16}}$. It is important to note that this result is independent of the number of attackers or the number of unique end hosts in the topology. Using the values from the 10 packet bootstrapping experiment, we calculated this probability to be 0.005 for the Internet Map topology and 0.003 percent for the Skitter topology. Although this result is two orders of magnitude worse than the ideal case, it still represents an incredibly small chance that a spoofed packet will be accepted by the StackPi-IP filter.

4 Discussion

The StackPi scheme has great potential as a DDoS and IP spoofing defense mechanism. In this section, we discuss a number of issues and extensions relating to the StackPi scheme that could substantially improve on the results we have obtained thus far. In Section 4.1 we discuss how the addition of machine learning techniques can improve the performance of a StackPi filter. In Section 4.2 we discuss an extension to the StackPi marking scheme that allows routers to mark with a variable number of bits. In Section 4.3 we show how StackPi-IP filters can be used to implement a form of standard IP traceback. In Section 4.4 we describe how StackPi marking can improve the performance of other IP traceback schemes. In Section 4.5 we discuss StackPi's compatibility with IP fragmentation. Finally, in Section 4.6 we explain briefly how StackPi can be applied in an IPv6 environment.

4.1 Using Machine Learning for Higher Filtering Accuracy and Fast Adaptation to Route Changes

We have shown in Section 3.5 that the StackPi-IP filter can be very effective in detecting IP spoofing attacks. However, in order to make this approach practical, we need to address the following challenges:

- It may take a long time to build up a table of ⟨StackPi, IP⟩ tuples, and the table may never be complete. If the victim receives a packet with a source IP address that is not in the table, it cannot decide whether the IP address is spoofed or not.
- Some end hosts are positioned close enough to the victim in the network topology that some of the initially random bits in the marking field may not be overwritten when the packet reaches its destination, because there are too few marking routers on the path. As a result, there may be multiple StackPi marks that correspond to the same end-host. This may also be the case for an end-host at an arbitrary distance from the victim that has traffic routed over multiple paths.
- Routing changes may affect packets' StackPi markings. When route changes happen, a packet originating from a legitimate user may have a new StackPi marking which a simple table look-up would label as spoofed.

We propose to use machine learning techniques to address the above issues in the following ways:

- **Infer StackPi markings of previously unseen IP addresses.** We observe that for a given destination, all the packets originating from the same network region (sometimes from the same CIDR block) will usually be routed along the same path and have the same StackPi marking. If we have seen a StackPi mark from a given network, we could infer the StackPi marks of other hosts within the same network. How can we reliably derive information about hosts that are on the same network? We will consider using the CIDR block information from BGP routing, and using machine learning techniques in conjunction with longest prefix matching of IP addresses with their associated StackPi marks.
- **Infer multiple StackPi markings in case of multi-path or short path.** For a given destination, if a region has multiple paths to the destination, then the StackPi marking of any host within that region may have multiple values. For example, given two hosts, A and B, from the same region, and whose StackPi marks are X and Y, respectively, then it is likely that the StackPi marking of A could also be Y, and the StackPi marking of B could also be X. We could use machine learning techniques to automatically detect this case and infer the multiple StackPi markings from observed data.

  In the case where some bits in the StackPi mark still contain the original bits of the IP Identification field, we observe that the StackPi markings have the same low order bits (from router markings pushed onto the stack) and only vary in the high-order bits (those bits that were not overwritten by router markings). Using machine learning techniques, we could automatically detect this case and filter only based on those bits that were not originally in the IP Identification field.
- **Infer StackPi marking change caused by route change.** When routes change, StackPi markings will change for some end-hosts. Because packets from the same region will have the same StackPi markings, the change of StackPi marking for one IP address will have a similar change for another IP address from the same region. Using machine learning techniques we could infer the StackPi marking change caused by a route change with a small number of packets. Also, with machine

learning techniques, we may be able to infer how route changes affect the StackPi markings and hence infer the StackPi marking change of one network region by observing the StackPi marking change of another network region.

4.2 Variable Bit Marking

Recall from Section 2.1.1 that the StackPi marking scheme has a global parameter n, which is the number of bits that each router uses to mark a single link. We may gain more space efficiency if routers were allowed to choose for themselves a particular n to use for marking. For example, in the current StackPi marking scheme, a router with only two interfaces would mark two bits in the packet, although that router does not truly affect the path at all, since it can be abstracted as simply a link between its last-hop and next-hop neighbors. In a variable bit marking scheme, such a router would not mark the packet. Each router would calculate its own n, possibly as a function of the number of interfaces it has. We are working on simulations that incorporate the variable bit marking scheme into StackPi.

4.3 Enabling Traceback with StackPi-IP filters

A properly bootstrapped StackPi-IP filter can be used to perform standard traceback, that is, complete path reconstruction from a packet's destination to its sender. When a destination receives a packet that is flagged because its source IP address does not match its StackPi marking in the StackPi-IP filter's database, the victim can consult its database to generate a list of IP addresses that correspond to the packet's StackPi mark. Once this list is compiled, the victim can determine the paths by simply executing `traceroutes` to the addresses on the list. Although this method does not guarantee a unique path to the sender (because there may be multiple IP addresses that map to the same StackPi mark), it does reduce the space of potential attackers and may allow the victim's administrator to cull the true attack path using external knowledge and intuition.

4.4 Synergies with Other Anti-DDoS Approaches

StackPi marking can greatly enhance the power and accuracy of existing anti-DDoS and IP traceback mechanisms. For example, the Pushback system [17, 23] uses downstream routers to identify packet *aggregates* (packets from one or more flows that have similar characteristics, such as source or destination addresses) and send rate-limit requests for those aggregates that are identified as attack flows. The StackPi mark can serve as both a method of aggregating packets (to ensure that they originate from the same host) and as an aggregate identifier for upstream routers.

The Source Path Isolation Engine (SPIE) system [33, 34] could also benefit from using the StackPi marking. In SPIE, routers keep hashes of the packets that they forward. When the victim of a DDoS attack presents a router with the hash of a packet that it received during the attack, that router queries its upstream routers for the presence of the hash in their tables. Rather than querying all upstream routers though, an intelligent router can reduce and even identify the upstream router to query based on the StackPi marking, which encodes link information along the packet's path. Reducing the number of routers queried could heavily reduce the frequency of false positives, where two different packets hash to the same value resulting in multiple possible paths attack paths.

4.5 Compatibility with IP Fragmentation

The use of a deterministic marking placed in the IP Identification field of the IPv4 header makes IP fragmentation difficult, at best. Under very strict network assumptions, (namely, no packet or fragment re-ordering or loss) StackPi marking can coexist with fragmentation. However, because fragmented traffic represents less than 0.25% of packets in the Internet [36], we have an alternative solution, proposed by Vern Paxson: routers should only mark packets that never get fragmented. There are two such classes of packets. The first class are those packets that have the `do not fragment (DFT)` bit set. Most modern TCP implementations set the `DFT` bit by default [39], as specified by the Path MTU Discovery standard in RFC 1191 [26]. The second class of packets are those packets shorter than 577 bytes. RFC 1122 [3] requires a minimum MTU size of 576 bytes for an IP traffic link, so packets that are shorter than that will never be fragmented. Thus if we only mark packets that have the `DFT` bit set or are shorter than 577 bytes; without marking any fragments of packets, which can be detected by either a zero fragment offset field or a one in the M flag bit, we can avoid and conflicts with IP fragmentation. During a DDoS attack, the victim can easily filter out all packets that do not satisfy this predicate, leaving only those packets that have already been marked to pass through to the StackPi filter. With these adjustments to the StackPi marking scheme, IP Fragmentation is preserved, yet our DDoS defense performance is uncompromised.

4.6 StackPi in IPv6

Although StackPi has been specifically designed for deployment in IPv4, the principal ideas of our scheme are equally applicable in an IPv6 environment. Rather than mark in the IP Identification field of the packet header – which does not exist in IPv6 – StackPi marks would be implemented as a *hop-by-hop* option, where each router along the path would mark its bits into the option's payload. StackPi is superior to other IP traceback schemes in an IPv6 environment because it does not transmit the IP addresses of the routers along a packet's path, and hence does not suffer from the quadrupling of the space required to transmit a 128 bit IPv6 address versus a 32 bit IPv4 address. However, StackPi's performance does depend on the number and degree of routers in the path of the packet. Relating the optimal payload (marking field) length to the size and path composition of the Internet remains an open research question.

5 Background and Related Work

There have been several studies of the frequency and nature of Internet DoS attacks [13, 15]. Moore, Voelker, and Savage use *backscatter* packets (the unsolicited responses that a DoS victim sends to the spoofed IP address that it receives in attack packets) to gauge the level of Internet DoS activity [27]. Jung, Krishnamurthy, and Rabinovich attempt to answer the question of how a server can differentiate between a DoS attack and a simple high load condition by analyzing client request rates and file access patterns [20].

Many approaches have been proposed for securing against DoS and DDoS attacks. Early methods focused on detecting the ingress and egress points of DoS traffic within a single network administration. Ferguson and Senie propose to deploy network ingress filtering to limit spoofing of the source IP address [12]. However, unless every ISP implements this scheme, there will still be entry points in the Internet where spoofing can occur. Also, the additional router configuration and processing overhead to perform this filtering is another reason why it is rarely used.

Park and Lee propose a distributed packet filtering (DPF) mechanism against IP address spoofing [28]. DPF relies on BGP routing information to detect spoofed IP addresses. Their approach is interesting, but will likely suffer in accuracy unless a majority of routers implement it. DPF would be quite easy to circumvent by an attacker, and would benefit from the approaches we propose in this work.

Stone proposes the CenterTrack mechanism, which uses routers capable of input debugging (the ability to tell through which router interface a particular packet was received) that would be virtually connected through IP tunnels to all border routers on a network [37]. When a node in the network comes under attack, the overlay network is activated, and all border routers channel traffic through the overlay routers. These routers would use input debugging to tell from which border router, and hence which neighbor network, the DoS traffic is coming from.

Burch and Cheswick present another scheme for source tracing [5]. This unique scheme uses a limited form of DoS attack to attack the exact path which the DoS traffic is traversing. By selectively exhausting network resources and monitoring the perturbations in DoS attack traffic, it is possible to detect the links that a DoS attack is traversing. Unfortunately, this method does not scale well for the multiple attackers in a DDoS attack, nor does it solve the problem of administrative coordination between ISPs.

None of the methods described previously relied on the IP protocol itself to assist in protecting against DoS attacks. A new class of protections seeks to modify parts of the IP protocol to assist in determining the origin of DoS and DDoS traffic. Early works in this category suggested adding a new type of ICMP message: traceback messages [2, 18]. For each packet received, routers would, with small probability, generate an ICMP message to the destination address of the packet containing the IP address of the router. The problem with this initial scheme is that there is a tension between providing fast path identification (in the number of packets received at the victim), and low network overhead generated by added messages. Mankin et al. present an improvement to this scheme, which puts some state on the routers to generate better traceback messages [24]. Better traceback messages are defined both as those originating from routers far away from the victim and as those that have not been seen previously by the victim. Although this improvement reduces the overhead of ICMP traceback significantly, it relies on either a shared key distribution mechanism to prevent attacker forged traceback messages, (which is a very difficult problem on an Internet scale); or on asymmetric cryptography, which could potentially be exploited by attackers in a DoS attack by exhausting server resources with failed, yet time consuming signature verifications. Several researchers propose to embed traceback information within the IP packet [1, 9, 10, 11, 14, 22, 31, 32, 35]. Most of these schemes use the 16-bit IP Identification field to hold traceback information. Routers along the packet's path probabilistically mark certain bits in the IP Identification field in certain ways. Markings generated by any router may be overwritten by other routers further down along the path toward the victim. The markings are reassembled at the victim to reconstruct the IP addresses of the upstream routers toward the attacker. While the traceback schemes could be used to find the origins of the attacks, they often require a huge number of packets and thus cannot be used to filter out packets on a per packet basis. A fundamental limitation of these IP traceback schemes is that they cannot detect spoofed IP addresses.

To surmount the problem of large numbers of packets necessary at the victim to trace back multiple attacker paths, Snoeren et al. propose a solution using router state to track the path of a single packet [33, 34]. Upon receipt of a packet, each router hashes specific, invariant fields of the packet and stores the hash in a table. When traceback is needed, the victim presents its upstream router with the hash of the packet to be traced. The routers at each hop away from the victim then recursively query the routers at the next hop away for the presence of the hash of the packet in their hash tables. Besides the ability to traceback single packets, this method also offers the advantage of storing saturated hash tables for traceback after an attack has taken place.

Ioannidis and Bellovin, and Mahajan et al. propose Pushback, a packet filtering infrastructure leveraging router support to filter out DDoS streams [17, 23]. The mechanisms we propose in this paper can be used to greatly increase the effectiveness of Pushback, as the filters can take the packet markings into account and thus distinguish packets from various origins (increasing the accuracy of filtering).

Sung and Xu propose an altered IP traceback approach, where the victim tries to reconstruct the attack path but also attempts to estimate if a new packet lies on the attack path or not [38]. Their scheme is probabilistic and each router either inserts an edge marking for the IP traceback scheme or a router marking identifying the router. Unfortunately, their approach requires the victim to collect on the order of 10^5 attack packets to reconstruct a path, and once the path is reconstructed, this scheme will likely have a high false positive rate as the routers close to the victim will all lie on some attack path and frequently mark legitimate packets which will then get rejected.

We have recently proposed a marking scheme *Pi*, a Path Identification algorithm [40]. The original Pi marking is based on the use of the packet's TTL field as an index into the IP Identification field where a router should add its marks. This method is not as lightweight as the StackPi method. Legacy routers have a harmful affect on the original Pi scheme because they decrement the TTL of a packet but do not add any markings. The StackPi scheme is robust to legacy routers and even includes the write-ahead scheme to incorporate markings for single legacy routers in the path.

6 Conclusion

In this paper, we present StackPi, a novel approach to defending against DDoS and IP spoofing attacks. The StackPi defense is split into two parts: StackPi marking and StackPi filtering. The StackPi marking scheme defines how the routers along a packet's path create a deterministic marking in the packet such that each packet traversing the same path has the same marking. StackPi marking includes a write-ahead method whereby some legacy routers' markings can be included in the StackPi mark. The StackPi filter defines how an end-host utilizes the packet markings to filter out attack traffic. We present two types of StackPi filters: threshold and IP based. Threshold filters allow the DDoS victim to minimize the false positive and false negative rates of incoming traffic. IP based filters allow the victim to detect, with high probability, a spoofed source IP address in an incoming packet. We show that StackPi provides strong protection to a DDoS victim, and shows strong incremental deployment properties. StackPi is compatible with IPv4 fragmentation and can operate in an IPv6 environment. StackPi can also be used to complement certain existing anti-DDoS schemes such as Pushback and SPIE. There are many extensions to StackPi that have yet to be explored, such as variable bit marking and the application of machine learning techniques to StackPi filtering, that promise to make StackPi a critical deterrent to today's most common Internet attacks.

References

[1] Micah Adler. Tradeoffs in probabilistic packet marking for IP traceback. In *Proceedings of 34th ACM Symposium on Theory of Computing (STOC)*, 2002.

[2] S. Bellovin, M. Leech, and T. Taylor. The ICMP traceback message. Internet-Draft, draft-ietf-itrace-01.txt, October 2001. Work in progress, available at `ftp://ftp.ietf.org/internet-drafts/draft-ietf-itrace-01.txt`.

[3] Robert Braden. Requirements for internet hosts – communication layers. Internet Request for Comment RFC 1122, Internet Engineering Task Force, October 1989.

[4] Hal Burch and Bill Cheswick. Internet watch: Mapping the Internet. *Computer*, 32(4):97–98, April 1999.

[5] Hal Burch and Bill Cheswick. Tracing anonymous packets to their approximate source. Unpublished paper, December 1999.

[6] Caida. Skitter. `http://www.caida.org/tools/measurement/skitter/`, 2000.

[7] CERT. TCP SYN flooding and IP spoofing attacks. Advisory CA-96.21, September 1996.

[8] CERT. Advisory CA-1995-01 IP spoofing attacks and hijacked terminal connections. `www.cert.org/advisories/CA-1995-01.html`, February 2001.

[9] Drew Dean, Matt Franklin, and Adam Stubblefield. An algebraic approach to IP traceback. In *Network and Distributed System Security Symposium, NDSS '01*, February 2001.

[10] Drew Dean, Matt Franklin, and Adam Stubblefield. An algebraic approach to IP traceback. *ACM Transactions on Information and System Security*, May 2002.

[11] T. Doeppner, P. Klein, and A. Koyfman. Using router stamping to identify the source of ip source address spoofing. In *Proc. ACM Computer and Communication Security Symposium*, November 2000.

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

[13] L. Garber. Denial-of-service attacks rip the internet. In *IEEE Computer*, volume 33, April 2000.

[14] Michael Goodrich. Efficient packet marking for large-scale IP traceback. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 117–126. ACM Press, 2001.

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

[16] Internet mapping. `http://research.lumeta.com/ches/map/`, 2002.

[17] John Ioannidis and Steven M. Bellovin. Implementing pushback: Router-based defense against DDoS attacks. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS 2002)*, San Diego, CA, February 2002. Internet Society.

[18] ICMP traceback (itrace). IETF working group, `http://www.ietf.org/html.charters/itrace-charter.html`.

[19] L. Joncheray. Simple active attack against TCP. `www.insecure.org/stf/iphijack.txt`, February 2001.

[20] Jaeyeon Jung, Balachander Krishnamurthy, and Michael Rabinovich. Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites. In *The Eleventh International World Wide Web Conference (WWW 11)*, May 2002.

[21] Mike Kristovich. Multi-vendor game server DDoS vulnerability. `http://www.pivx.com/kristovich/adv/mk001/`, November 2002.

[22] Heejo Lee and Kihong Park. On the effectiveness of probabilistic packet marking for IP traceback under denial of service attack. In *Proceedings IEEE Infocomm 2001*, April 2001.

[23] Ratul Mahajan, Steven M. Bellovin, Sally Floyd, John Ioannidis, Vern Paxson, and Scott Shenker. Controlling high bandwidth aggregates in the network. *CCR*, 32(3):62–73, July 2002.

[24] 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 IEEE International Conference on Computer Communications and Networks*, October 2001.

[25] David McGuire and Brian Krebs. Attack on internet called largest ever. *washingtonpost.com*, October 2002. `http://www.washingtonpost.com/wp-dyn/articles/A828-2002Oct22.html`.

[26] Jeffrey Mogul and Steve Deering. Path MTU discovery. Internet Request for Comment RFC 1191, Internet Engineering Task Force, November 1990.

[27] David Moore, Geoffrey Voelker, and Stefan Savage. Inferring internet denial of service activity. In *Proceedings of the 10th USENIX Security Symposium*, Washington, D.C., August 2001. USENIX.

[28] Kihong Park and Heejo Lee. On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets. In *ACM SIGCOMM '01*, pages 15–26, 2001.

[29] Vern Paxson. An analysis of using reflectors for distributed denial-of-service attacks. *Computer Communication Review*, 31(3), 2001.

[30] Ron Rivest. The MD5 message-digest algorithm. Internet Request for Comment RFC 1321, Internet Engineering Task Force, April 1992.

[31] Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson. Practical network support for IP traceback. In *Proceedings of the 2000 ACM SIGCOMM Conference*, August 2000.

[32] Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson. Network support for IP traceback. *ACM/IEEE Transactions on Networking*, 9(3), June 2001.

[33] Alex C. Snoeren, Craig Partridge, Luis A. Sanchez, Christine E. Jones, Fabrice Tchakountio, Stephen T. Kent, and W. Timothy Strayer. Hash-based IP traceback. In *Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '01)*, pages 3–14, August 2001.

[34] Alex C. Snoeren, Craig Partridge, Luis A. Sanchez, Christine E. Jones, Fabrice Tchakountio, Beverly Schwartz, Stephen T. Kent, and W. Timothy Strayer. Single-packet IP traceback. *IEEE/ACM Transactions on Networking (ToN)*, 10(6), December 2002.

[35] Dawn X. Song and Adrian Perrig. Advanced and authenticated marking schemes for IP traceback. In *Proceedings IEEE Infocomm 2001*, April 2001.

[36] Ion Stoica and Hui Zhang. Providing guaranteed services without per flow management. In *SIGCOMM'99*, pages 81–94, 1999.

[37] Robert Stone. CenterTrack: An IP overlay network for tracking DoS floods. In *Proceedings of the 9th USENIX Security Symposium*, Denver, Colorado, August 2000. USENIX.

[38] Minho Sung and Jun Xu. IP traceback-based intelligent packet filtering: A novel technique for defending against internet DDoS attacks. In *Proceedings of IEEE ICNP 2002*, November 2002.

[39] R. van den Berg and P. Dibowitz. Over-zealous security administrators are breaking the internet. In *Proceedings of 2002 LISA Conference*, November 2002.

[40] Abraham Yaar, Adrian Perrig, and Dawn Song. Pi: A path identification mechanism to defend against DDoS attacks. In *IEEE Symposium on Security and Privacy*, May 2003.