PathFinder: Capturing DDoS Traffic Footprints on the Internet

Lumin Shi*, Mingwei Zhang*, Jun Li*, Peter Reiher†
* University of Oregon
{luminshi, mingwei, lijun}@cs.uoregon.edu
† University of California, Los Angeles
reiher@cs.ucla.edu

***Abstract*—While distributed denial-of-service (DDoS) attacks are easy to launch and are becoming more damaging, the defense against DDoS attacks often suffers from the lack of relevant knowledge of the DDoS traffic, including the paths the DDoS traffic has used, the source addresses (spoofed or not) that appear along each path, and the amount of traffic per path or per source. Though IP traceback and path inference approaches could be considered, they are either expensive and hard to deploy or inaccurate. We propose PathFinder, a service that a DDoS defense system can use to obtain the footprints of the DDoS traffic to the victim as is. It introduces a PFTrie data structure with multiple design features to log traffic at line rate, and is easy to implement and deploy on today's Internet. We show that PathFinder can significantly improve the efficacy of a DDoS defense system, while PathFinder itself is fast and has a manageable overhead.**

***Index Terms*—distributed denial-of-service; DDoS; traffic footprint; autonomous system (AS); PFTrie**

I. Introduction

Today's Internet is vulnerable to distributed denial-of-service (DDoS) attacks. During a DDoS attack, an attacker controls many compromised machines to flood the victim with unwanted traffic in order to exhaust the network or computational resources of the victim. DDoS attacks have become more frequent and damaging to many network services [1]. For example, a recent large-scale DDoS attack on Dyn [2] disabled its domain name service, and crippled many major web services that relied on it such as Twitter, Netflix, PayPal, and over fifty others for hours.

While many DDoS defense systems have been proposed, a primary challenge in effectively defending against DDoS attacks is that a DDoS defense system usually has little knowledge regarding which paths DDoS traffic has traveled along, how much traffic traveled along each path, and also which source addresses or prefixes of the DDoS traffic are associated with each path. Such information about the DDoS traffic, which we collectively call the **footprints** of the DDoS traffic, if available, can enable a DDoS defense system to most effectively handle the DDoS attack. It can better decide at which positions to deploy DDoS traffic filters or take other defense actions, such as the autonomous systems (ASes) that have seen a large amount of traffic to the victim; it can also know better which source addresses (or more likely the source IP prefixes, for scalability) to filter in the case of source-based filtering; and it could also conduct traffic pattern analysis if the footprints are continuously provided.

Various approaches to obtaining such information could be considered, including numerous IP traceback approaches and path inference methods that aim to address the asymmetric nature of Internet paths and ascertain the paths traveled by DDoS packets to reach the victim. Unfortunately, they have serious drawbacks. For example, the path inference methods are often inaccurate, and IP traceback approaches introduce significant changes to router hardware or software, rely on inter-AS collaboration, and need routers on the Internet to *constantly* monitor the traffic. These approaches are also not well-equipped to provide other footprint information, such as total or per-source bandwidth consumption information or the addresses or prefixes of DDoS sources. (We discuss these approaches in more detail in Sec. II.)

We therefore introduce the **PathFinder** system as a service for DDoS defense systems. Upon request from a DDoS defense system on behalf of a DDoS victim, PathFinder can gather and provide the footprints of the traffic to the victim. We make the following contributions:

- PathFinder consists of an architecture that is easy to implement and deploy on today's Internet. Every AS can join PathFinder without reliance on other ASes, and it employs an *on demand* service model with low overhead.
- We design the setup and operations of each component of the architecture while considering a series of real-world factors and the high speed and large scale of DDoS traffic.
- We design a new data structure called **PFTrie** that supports fast and easy storage and retrieval of traffic footprint information. And we also design a set of PFTrie optimization methods.
- We show the benefits of using PathFinder for DDoS defense, and we further evaluate PathFinder to show it is fast and has a manageable overhead.

ISBN 978-3-903176-08-9
This project is in part the result of funding provided by the Science and Technology Directorate of the United States Department of Homeland Security under contract number D15PC00204. The views and conclusions contained herein are those of the authors and should not be interpreted necessarily representing the official policies or endorsements, either expressed or implied, of the Department of Homeland Security or the US Government.

An apparent issue here is IP spoofing. The DDoS traffic could carry spoofed IP source addresses. However, we note that regardless whether the IP source addresses of DDoS traffic are spoofed or not, PathFinder will still discover the correct set of paths that DDoS traffic take to reach the victim, thus enabling a DDoS defense system to use the path information, together with other traffic footprints information, to block the DDoS traffic *en route* accordingly. On the other hand, if a DDoS defense system needs to filter DDoS traffic based on the source addresses of DDoS traffic, the DDoS defense itself needs to handle the IP spoofing separately, which is out of the scope of PathFinder.

The rest of this paper is organized as follows. We first discuss related work in Sec. II, followed by an overview of PathFinder in Sec. III. We then describe individual components of PathFinder, including the PathFinder monitor in Sec. IV, the PFTrie data structure for traffic logging in Sec. V, and the PathFinder proxy in Sec. VI. We evaluate PathFinder in Sec. VII, discuss some open issues in Sec. VIII, and conclude the paper in Sec. IX.

II. Related Work

A. IP Traceback

IP traceback, first introduced in [3], allows a victim to trace the source of an IP packet it has received and reconstruct the router-level path taken by the packet, even if the source address of the packet is spoofed. While many IP traceback solutions have been proposed [4], marking and logging are the two most well-developed approaches.

In a marking approach, such as those described in [5], [6], [7], when a router along a path forwards a packet to the victim, the router marks the packet with its own IP address (or its hashed result) or an edge that the packet has traversed, typically using some unused fields in the IP header of the packet. When the victim receives *enough* marked packets, even if routers *en route* mark packets with certain probability rather than all the time, the victim can then reconstruct the paths of these packets (assuming the paths are stable).

In a logging approach such as [8], [9], before a router along a path forwards a packet to the victim, instead of marking the packet, the router uses some data structure (e.g., a Bloom filter) to store the digest of the packet (rather than the packet itself in order to save space), enabling it to later determine whether it has seen the packet. When the victim wants to trace a packet, it can query its upstream routers, asking whether they have seen the packet. Similarly, a router that has seen the packet can query its own neighboring routers about the packet, and so on. Eventually the victim can reconstruct the packet's path using an ordered list of routers that have seen the packet.

PathFinder has advantages over existing IP traceback approaches in the following respects:

- *Operation*: PathFinder is also essentially a logging approach, but while the previous logging-based IP traceback approaches try to trace the path(s) of any packets (such as packets from a specific source) that a victim has received, PathFinder aims to discover the paths of *upcoming* packets (often all upcoming packets within a time window) toward a victim when requested. So, while existing IP traceback approaches record packet information or mark packets constantly, PathFinder is an on-demand service and PathFinder monitors will only record traffic information when requested.
- *Overhead*: Because it operates on demand, PathFinder incurs much less operational overhead than existing IP traceback approaches. In addition, packet marking approaches will modify packets before forwarding them, which will introduce delays in processing packets and could downgrade the network throughput significantly, especially when dealing with a high-bandwidth link.
- *Accuracy*: The accuracy of the marking approaches depends on how many marked packets the victim can receive to reconstruct the paths of packets. The accuracy can suffer if the victim cannot receive enough marked packets, such as when its inbound link is congested with DDoS traffic. The existing logging approaches (and also PathFinder), on the other hand, as long as their monitoring mechanism can process packet headers at line speed, can log packet information with little loss and thus reach a high accuracy in tracing packets.
- *Deployability*: Existing IP traceback approaches face obstacles for deployment: Whether based on marking or logging techniques, they introduce significant hardware or software changes to routers, and also require inter-AS collaboration. PathFinder instead introduces few changes to routers, and PathFinder-participating ASes talk directly with a PathFinder proxy and do not need to communicate with each other.

B. Path Inference

Researchers have studied how to infer the path between two end points on the Internet. Without assuming any control over the network infrastructure or access to end points, research in [10] investigated how to leverage Border Gateway Protocol (BGP) tables collected from multiple vantage points to infer the AS path between any two end points on the Internet. Also, via the probing from multiple vantage points and the IP timestamp and record route options, research in [11] proposed a "reverse traceroute" to allow a user to infer the path from a remote end point to the user, without accessing the remote end point. Inference-based approaches do not require changes to network equipment and are easy to deploy, however, in general they are subject to some degree of inaccuracy (e.g., the accuracy from the research in [10] is 70-88%). Moreover, since they are not based on watching traffic in real time, they will not be able to report the bandwidth consumption and other traffic-related information associated with the path.

III. PathFinder Overview

A. PathFinder as a Service for DDoS Defense

We design PathFinder as a service for DDoS defense. Upon request from a DDoS defense, PathFinder can provide the footprints of all the traffic toward a victim that each participating AS has witnessed. Note that PathFinder does

not distinguish DDoS traffic from the legitimate traffic, which PathFinder assumes to be the job of the DDoS defense. The footprints include:

- all the AS paths taken by the traffic;
- if requested, the source IP addresses or prefixes of the traffic; *and*
- if requested, the amount of traffic per source address, or per source prefix, or per AS *en route*, or other information about the traffic.

When requesting the PathFinder service, a DDoS defense can specify to PathFinder a set of parameters regarding the traffic footprints, including:

- the destination address of the victim, which could be an IP address or prefix, or an IP address plus a port number;
- the length of time for which to collect the traffic footprints (typically during the DDoS attack);
- from which ASes (if not all the ASes supporting PathFinder) to collect the traffic footprints;
- whether to collect the source addresses or prefixes of the traffic, and if so, the prefix granularity (e.g., /24 is to learn all the /24 source prefixes; /32 is to learn all the /32 source prefixes, i.e., all the source IP addresses); *and*
- whether to collect the bandwidth consumption of the traffic, and if so, the granularity of the bandwidth information (per source address, per source prefix, or per AS *en route*) and the unit (packets per second or bits per second).

B. Architecture

PathFinder is a log-based system that enables a user (which is DDoS defense in this paper) to learn the AS paths, sources, bandwidth consumption, and other information of the traffic toward a DDoS victim, i.e., the footprints of the traffic. As we will show in the following, it is easy to deploy as it requires minimal reconfiguration of routers; it is scalable as it will continue to perform well if there is more traffic from more sources or if more ASes support PathFinder; and it is accurate, fast, and efficient in providing the traffic information.

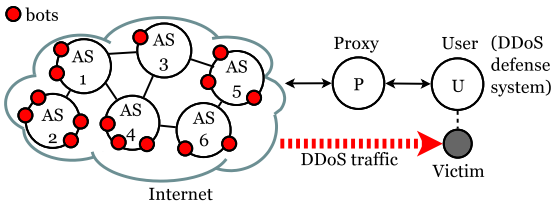


Fig. 1: **PathFinder architecture.**

Fig. 1 shows the high-level architecture of PathFinder. It consists of three types of entities:

- **PathFinder users** who interact with their proxy to request the PathFinder service, and retrieve from their proxy the footprints of the inbound traffic to a DDoS victim;
- **PathFinder proxies** which (1) pass their users' requests (including all parameters described in Section III-A) to all participating ASes—actually their PathFinder monitors; (2) receive and process from these ASes the PathFinder logs, which we call **PFLogs**, to derive the DDoS traffic footprints; and (3) return the footprints to their user; *and*
- **PathFinder monitors** at all participating AS which, according to the request from a proxy, (1) process the traffic that their AS originates or forwards towards the victim specified in the request; (2) generate PFLogs of the traffic, which record the AS path, source addresses (if requested), and amount (if requested) of the traffic; and (3) return the PFLogs to the proxy. Note that monitors from different ASes do not need to interact with each other, thus not introducing into the architecture any reliance on inter-AS collaboration.

IV. PathFinder Monitor

A. Addressing Design Requirements

Foremost, the monitor at each PathFinder-participating AS faces the following two design requirements. First, the monitor needs to consult routers within the AS to learn the AS path from the AS to the victim. Second, it also needs to access the traffic toward the victim in order to record their source addresses and/or amount, if requested. As an AS can have a complicated topology with inter-connected border routers and inside routers, some routers may not be on any path toward the victim at all and some may be on the same path. To meet both requirements, for every path of the traffic to the victim, the monitor must be able to talk with at least one router that is on the path, in order to learn its AS path to the victim or access its traffic to the victim. For the former (to learn its AS path), as every border router on the Internet runs BGP and maintains a Routing Information Base (RIB), the monitor can query the RIB at the router. Note that BGP router vendors such as Cisco [12] and Juniper [13] all support the query of AS paths. For the latter (to access traffic), the monitor needs to receive a copy of the traffic by applying traffic mirroring or tapping techniques (we rule out the possible hardware telemetry support from routers; although they produce traffic records such as those in NetFlow or IPFIX format, the records are only exported periodically, often with a long interval).

The monitor may further face a third requirement if it needs to produce PFLogs with source information or also the traffic amount records. There may be a huge amount of traffic from many distinct sources toward the victim, especially if the victim is currently under a severe DDoS attack, thus making it challenging for the monitor to record all the sources and their corresponding bandwidth consumption at a high speed. The most obvious solution is to use a digest-oriented data structure such as a Bloom filter or hash table. However, while the monitor can use a Bloom filter to easily answer whether it has seen an IP address or prefix or not, it is not good at recording which source IP addresses or prefixes it has seen. A hash table is better, but it can only output whatever is stored in itself as is; it is not flexible in processing or aggregating IP address and prefix information, thus scaling poorly when the logs are of a huge size. We therefore design a new, trie-based data structure called **PFTrie** to facilitate the recording and transmission of PFLogs, which we detail in Section V.

B. Setup

A PathFinder-participating AS needs to set up its PathFinder monitor and its working environment as follows. First, the monitor needs to set up the traffic mirroring or tapping with every border router from which it will need to obtain traffic *in real time*. To do so, given the autonomy of ASes, each AS may adopt a different procedure that it prefers. For a small AS without many routers, it can physically wire the monitor with every router for wiretapping (Fig. 2a shows an example). For a large AS with many routers over a large geographic region, we assume it can first learn which routers the AS has and then use virtual circuits for traffic mirroring with the routers [14], [15]. Further, a large AS may employ multiple monitors, with one monitor configured as a master monitor that can assign the workload across all the monitors. (Without losing generality, we assume one monitor per AS in the rest of the paper.)

Second, the monitor needs to be able to remotely login to each router that it is wired with and execute commands on the router, such as querying the AS path or the next hop from the router to any destination IP address. To do so, we assume every router supports secure shell (*ssh*), which is true for most routers nowadays, such as Cisco and Juniper routers [16], [17].

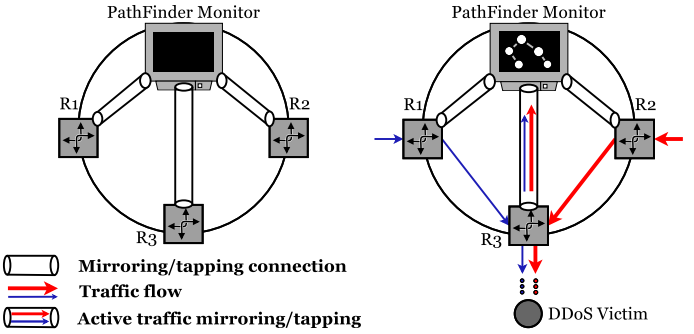
Finally, the monitor must be easy to discover by every PathFinder proxy. While the monitor can employ a running daemon process with a publicly known port number, proxies must also know the monitor's IP address. Many options exist; for example, the monitor can have its IP address and other information maintained at a web page. Or, the AS can set up a Domain Name System (DNS) record for its PathFinder monitor; e.g., `3582.pathfinder.org` may point to the PathFinder monitor of `AS 3582`.

C. Operation

The monitor at every PathFinder-participating AS operates on demand, remaining idle unless it receives a request from a PathFinder proxy, in which case the monitor will learn the IP address or prefix of the victim in question, together with the parameters in the request as defined in Sec. III-A, and start to generate PFLogs on behalf of the victim.

The very first step that the monitor takes is to identify a set of routers in its AS that are both sufficient and necessary to capture all the possible traffic that the AS may originate or forward toward the victim. Note the AS may also originate traffic toward the victim from any router of the AS itself. We therefore use all possible egress routers from which the traffic to the victim may exit the AS. For example, in Fig. 2b as the AS forwards two traffic flows toward the victim and both exit the AS from egress router $R3$, the monitor will select $R3$ to produce PFLogs for the victim. Further, it is straightforward to decide which routers are possible egress routers for the traffic to the victim. The monitor can query every border router's RIB to learn its next hop to reach the victim's IP. If the next hop is a router still within the AS, the border router in question is not an egress router; otherwise, it is an egress router.

Once the routers are selected, the monitor then talks with them to collect and produce PFLogs based on the request by



(a) The setup of a PathFinder-participating AS.

(b) The snapshot when the monitor is capturing traffic.

Fig. 2: **An example setup of a PathFinder-participating AS.**

the victim. First, the monitor will query each selected router to retrieve its AS path to the victim from its RIB. Furthermore, if the user requests the bandwidth consumption information of the total traffic to the victim via the AS, since the monitor is mirroring or tapping the traffic from these routers (among others), the monitor can observe the traffic from these routers and count their total volume (# of packets or # of bits), either per time unit or over a period of time (as specified in the request or based on a default value). Note that the mirrored or tapped traffic is only the traffic to the victim and more importantly, not on the path of the production traffic, therefore they will not impose a burden on the production traffic.

At this point, if the user did not request the source addresses or prefixes of the traffic, or source-based bandwidth consumption or other information, the monitor has collected all the PFLogs for the user, and thus can return them to the proxy of the user. We call this mode of operation **source-agnostic mode**. Otherwise, the monitor will operate in the **source-aware mode** to further produce source-based PFLogs via PFTrie (see Sec. V) before it returns them to the proxy.

Finally, note that each monitor only communicates with PathFinder proxies. No inter-AS collaboration is needed. In other words, monitors from different ASes are not required to communicate or collaborate, and each AS independently participates in PathFinder without any reliance on other ASes.

V. PFTrie—A PathFinder Data Structure for Traffic Logging

When in source-aware mode, the monitor at every PathFinder-participating AS will need to record all the sources that the AS has seen sending traffic to the victim in question, and if requested, the bandwidth consumption information per source. In doing so, the monitor needs to employ a data structure and accompanying algorithms to log sources, count bandwidth consumption, and transmit such data, all at a high speed to keep up with the line-speed packet arrival rate. Meanwhile, due to its speed, the trie data structure has been popular in storing IP addresses and prefixes, such as those in the Forwarding Information Base (FIB) of routers. A trie is also called a *prefix tree*, where every node on the trie uses its position on the tree to store the key of the node, such as the IP

address or prefix represented by the node. We therefore adopt the trie data structure for this purpose. Furthermore, to meet the design requirements discussed in Sec. IV-A, we enhance the trie data structure and design the PFTrie as follows.

A. Basic PFTrie Operations

The monitor captures and processes every packet toward the victim. It will make sure the IP source address of the packet is stored into the PFTrie via a ***put*** process. In this process, the monitor may modify the PFTrie by adding new nodes to store the IP, or discover that the address is already stored due to a previous packet with the same IP. The *put* process will return a node representing the IP, which the monitor can further update with bandwidth consumption information incurred by the current packet, if requested.

In the *put* process, the monitor will traverse the trie from the root downwards. It will traverse a node at each level—which we also call an *anchor*—to further move to the next level; clearly, when the traversal starts the anchor is the root of the trie. At the same time, it iterates through the bits of the IP address, starting from the leftmost bit, as follows:

(1) If the current bit in the IP address is `0`, it traverses to the left child of the anchor, otherwise it traverses to the right child; either way, the chosen child will become the new anchor.

(2) In case the new anchor does not exist, the monitor will detect a fault, i.e., the trie has not stored this IP address yet; the monitor will then add the missing child onto the trie, and use this child as the new anchor. (Note that this new anchor will also avoid the same fault for the next time.)

(3) If the current bit is already the rightmost bit of the IP address, the monitor then knows that the IP address is stored in the trie as represented by the new anchor, and thus return the new anchor node. Otherwise, it still needs to move to the next bit of the IP address; it then uses the new anchor as the current anchor to repeat step (1) above. Note the returned node is always a leaf node on the trie.

Fig. 3 shows an example of inserting a new IP address that ends with `101`. When it traverses to node a by following bit `1`, it needs to follow bit `0` to go to a's left child; since it does not exist, the monitor adds node b as a's left child. It then needs to follow the last bit `1` to go to b's right child; since it does not exist either, node c is then added, which also represents the newly stored IP address.

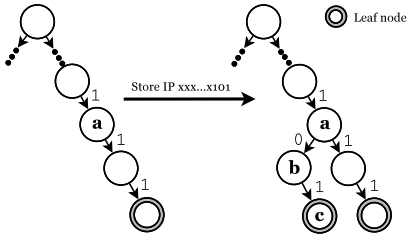


Fig. 3: **Store an IP address that ends with 101.**

B. Trie Optimization

We further optimize the PFTrie toward a faster traversal process. First, with the design in Sec. V-A, for every bit of an IP address, the trie must maintain a node at each level; for example, an IPv4 address will lead to 32 nodes at 32 respective levels on the PFTrie. We address this issue with two optimization methods that can go in parallel: the bottom-up aggregation of leaf nodes and the top-down collapse of prefixes. Furthermore, we also introduce a method to avoid duplicate traversal of the PFTrie when a source address is already stored. We describe each method below.

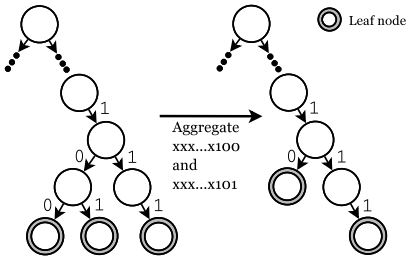


Fig. 4: **PFTrie optimization: Aggregating sibling leaf nodes into one new leaf node.**

1) Bottom-up Aggregation of Leaf Nodes: Because of traffic locality, sometimes there can be multiple sources from the same prefix sending traffic to the victim. For example, besides seeing the 32-bit source IP `xxx...x101` to the victim, the monitor may also see traffic to the victim from another source IP `xxx...x100`, which only differs from the former source IP by the very last bit; in other words, they share the same 31-bit prefix. When such locality is detected, two leaf nodes at level 32 are not needed to represent the two IP addresses. Instead, as shown in Fig. 4, we can aggregate the two leaf nodes into one new level-31 *leaf* node, indicating the monitor has seen traffic from both IP addresses in the 31-bit prefix. Furthermore, this aggregation can continue if the new leaf node has a sibling leaf node. Clearly, this bottom-up aggregation process can reduce the depth of certain branches of the PFTrie, thus speeding up the *put* process. One challenge here is the logging of the bandwidth consumption information. After aggregation, the monitor could simply copy the bandwidth consumption of each old leaf node into the new leaf node; or, it can also sum the bandwidth consumption of the two leaf nodes, thus recording the bandwidth consumption of the IP prefix represented by the new leaf node. The choice here depends on the user's request regarding the prefix granularity for recording the bandwidth consumption information (e.g., a /32 prefix granularity means to record the information per IP address, while a /0 prefix means the total bandwidth consumption for the whole IP space).

2) Top-down Collapse of Prefixes: We notice that in the *put* process the nodes at the top portion of the PFTrie are frequently traversed. Rather than traversing these nodes one by one each time, we collapse them into all the IP prefixes they represent, allowing the *sub-trie* below each prefix to be reached by directly indexing an array. Fig. 5 shows an example of collapsing /24 prefixes into an array (the monitor can collapse prefixes of other lengths, such as all the /16 prefixes, similarly). We populate the array with all /24 prefixes that exist in the PFTrie. For every /24 prefix, the monitor treats the 24

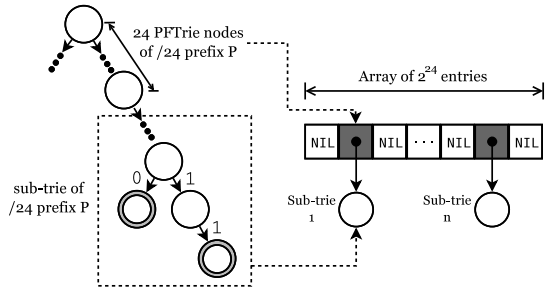


Fig. 5: **PFTrie optimization: Collapsing all /24 prefixes into an array with 2^{24} entries.**

bits of the prefix as an integer, use the integer as the index to directly locate the entry of the array, and have that entry point to the sub-trie originally below the prefix. So, instead of traversing 24 nodes of a /24 prefix and then traversing the sub-trie of the prefix, the monitor can immediately locate the entry for this prefix in the array, access from the entry the sub-trie of this prefix, and then traverse the sub-trie as before.

3) Avoidance of Duplicate Traversal: So far, if the PFTrie has recorded an IP address, when a packet with the same IP address arrives, the monitor will still run the *put* process and traverse the PFTrie, only to find the IP address is already stored. With n more packets from the same IP address, the extra overhead will be multiplied by n times.

We introduce a bitmap for each one of M most recently visited sub-tries. After storing an IP address in a sub-trie, the monitor will also set the bit in the bitmap corresponding to this IP to `1`, so that the *put* process for the same IP later will return very quickly. For example, the sub-trie for prefix `a.b.c/24` can have a bitmap of 2^8 bits, with the bit at index d corresponding to IP address `a.b.c.d`.

If we also need to update the bandwidth consumption information for the IP address (or its prefix), we will need to access its leaf node. To still have a speedy *put* process for duplicate IP addresses, we replace the bitmap with an array of pointers, and setting a bit to `1` above becomes inserting into the array a pointer that points to this leaf node. In the above example, the pointer at index d will be either *null* or point to the leaf node for IP address `a.b.c.d` (or its prefix).

VI. PathFinder Proxy

A. Addressing Design Requirements

The purpose of the proxy is to learn the footprints of the traffic toward a victim. Since on the same AS-path of the traffic there can be multiple ASes, when the monitors of such ASes report the AS-paths of the traffic, the AS-paths reported by them may overlap. The proxy must determine which AS-path includes the largest number of ASes. Furthermore, if source-based traffic footprint is requested, these monitors may also store and report the same IP address or prefix. The design of the proxy should resolve the potential conflict between monitors regarding the same IP address or prefix. Finally, the proxy's design should be cost-aware. Since the proxy does not contact monitors unless requested by the user, clearly the best operation mode of the proxy is also on demand.

B. Setup

Every PathFinder proxy will make itself available to potential PathFinder users. If a user needs PathFinder service, it will register itself at a proxy, including setting up all security credentials. The user then can send a request to the proxy when it needs to obtain the traffic footprints of a DDoS victim.

We assume the proxy has a list of PathFinder-participating ASes (which the proxy can obtain, for example, through a web page). Further, it knows how to locate the PathFinder monitor of each AS, as described in Sec. IV-B.

C. Operation

Like any PathFinder monitor, every proxy also operates on demand. Once a proxy receives a request from a user, it will verify that the request is authentic and valid, and if so, learn who the victim is from the request and forward the request to monitors at PathFinder-participating ASes (or a subset of them if specified in the request).

If the footprints do not need to be source-based, each monitor will function in source-agnostic mode and the proxy will receive PFLogs from each monitor that contain the AS path from the monitor's AS to the victim, as well as bandwidth consumption information if requested. The proxy then adds the path to a path pool, with two exceptions: (1) If the path is just a part—i.e., a **sub-path**—of another path in the pool, the proxy can ignore this path. (2) Conversely, if a path from the pool is a sub-path of this path, the latter will replace the former in the pool. As a result, the proxy will learn a set of AS paths to the victim, and can return them to the user, together with the bandwidth information if requested.

However, if the footprints need to be source-based, the proxy will construct a local PFTrie based on the PFTries it receives from monitors. Every monitor incrementally transmits its PFTrie to the proxy; e.g., whenever it can fit newly added PFTrie nodes into an IP packet or a timer expires, the monitor will transmit the updates of its PFTrie to the proxy. For each leaf node of the PFTrie from each monitor, which represents an IP address or prefix `ip` that the monitor has captured, the proxy will store `ip` in its local PFTrie, following the same *put* process described in Sec. V-A. Furthermore, assuming the monitor's AS is `AS k`, the proxy also marks the leaf node that represents `ip` with `k`, in order to indicate the AS-path of traffic from `ip` to the victim is the AS-path from `AS k` to the victim. However, if `ip` is already in the local PFTrie, the proxy will retrieve the marked AS number of the leaf node for `ip`, say `x`, and compare `AS k` (i.e., the monitor's AS) with `AS x` to see which AS is upstream. If `AS k` is upstream, the proxy marks the leaf node with `k`; if `AS x` is upstream, the leaf node continues to be marked with `x`. We thus always can obtain the most complete AS-path from `ip` to the victim.

VII. Evaluation

A. Goals

We now evaluate PathFinder in terms of the following:

- *Benefits of PathFinder for DDoS defense:* Since PathFinder footprints include path and traffic information, any DDoS

defense system that deploys filters inside the network to discard DDoS traffic can take advantage of the footprints to make better filter deployment decisions. We build a DDoS attack and defense simulation to study how PathFinder can benefit a DDoS defense system and make it more effective. We look at four different strategies of placing DDoS traffic filters and show that a DDoS defense system—when utilizing PathFinder—uses much less resource and achieves a much higher level of success.

- *Speed and Overhead of PathFinder:* At the core of PathFinder are the PFTrie-based operations at each monitor and proxy, particularly the *put* process. Therefore, we evaluate the time to store an IP address or prefix `ip` in a PFTrie under two scenarios. In one scenario, `ip` is new and the PFTrie needs to be updated with a set of new nodes, including the leaf node that represents `ip`. In another scenario, `ip` is in the PFTrie, so the *put* process will check and return the current leaf node that represents `ip`. We call these two scenarios **put_a_new** and **put_an_old**, respectively. Furthermore, we evaluate the memory overhead of the PFTrie at each monitor and proxy when producing source-aware footprints and the network overhead when transmitting a PFTrie.

B. Benefits of PathFinder for DDoS Defense

To quantify the benefits of PathFinder for DDoS defense, we first define the model of DDoS attack and defense, and then compare the simulation results of a DDoS defense system with and without PathFinder.

1) DDoS Attack and Defense Model: The DDoS attack and defense model includes a botnet, an attack victim, a DDoS defense system, and the PathFinder system. The botnet contains 100,000 bots, where each bot is at a random tier-3 AS and has a fixed uplink bandwidth of 25 Mbit/s. The victim can at most handle 10 Gbit/s incoming traffic. During an attack, the victim applies the DDoS defense system to filter DDoS traffic, with or without the help of PathFinder to locate best locations for filters. We allow only tier 2 and 3 ASes that are close to the attack sources to deploy filters, as in practice the ASes close to the victim are subject to link congestion under DDoS attack (often too late for them to filter the DDoS traffic).

Since the same set of bots tend to take the same paths to send traffic toward the victim, these bots will no longer be effective once filters are deployed to filter their traffic. An intelligent attacker therefore would periodically switch to a new batch of bots to launch its attack. We define the *attack cycle* as the time window for every batch of bots used by the attacker. On the other hand, upon a newly seen DDoS traffic, we assume it takes T seconds in total for PathFinder to collect traffic footprints and also for the DDoS defense system to place the filters. In this study, we evaluate PathFinder system under the worst-case scenario where the attack cycle is no greater than T; in another words, before the DDoS defense places filters for the current bots attacking the victim, the attacker already switches to a new attack cycle with a new batch of bots.

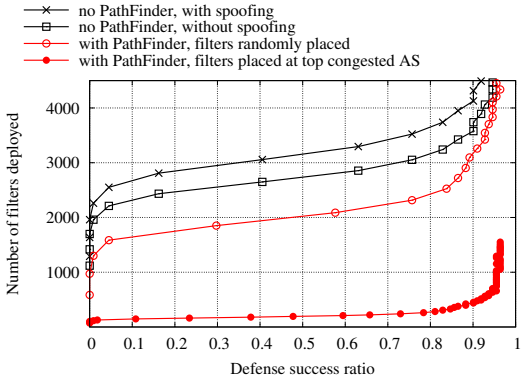


Fig. 6: **DDoS defense with and without PathFinder**

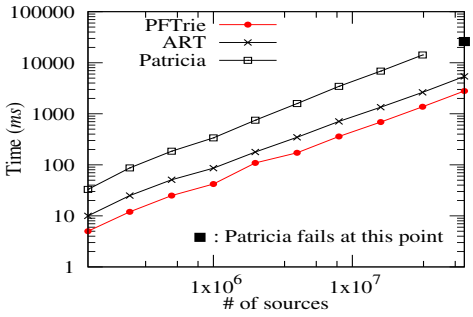
2) DDoS Defense Efficacy with PathFinder: We then compare the DDoS defense efficacy without and with PathFinder. When the DDoS defense does not have help from PathFinder, it uses the inferred AS-level topology to place filters, with two cases: 1) there is a 30% chance that a filter deployed is ineffective due to asymmetric routing on the Internet [10]; 2) further with some portion of the bots using IP spoofing, there is then a 50% chance that a filter will be ineffective. When PathFinder is in place, we use two AS selection methods for filter placement with the help of PathFinder: 1) randomly select an upstream AS; 2) select an AS that belongs to top k ASes that carry most of the DDoS traffic.

Figure 6 shows results for all four cases defined above. We see the results of DDoS defense system without PathFinder performs much worse than both cases when PathFinder is available. With PathFinder in place, and by applying filters at top congested ASes (which requires path and bandwidth information from PathFinder), the victim can survice 90% of the attack cycles with roughly 500 filters, whereas it takes at least 3,500 filters for a DDoS defense system to subdue 90% of the attack cycles if there was no PathFinder. In the case when the DDoS defense system uses only path information from PathFinder and places filters randomly at ASes, the system still uses much less number of filters compared to the two defense cases without PathFinder.

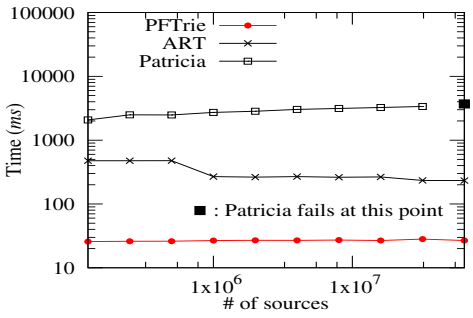
C. Speed and Overhead of PathFinder

1) Experiment Setup: To evaluate the PFTrie speed and its memory overhead, we used a desktop with Intel i7-4790 at 3.6 GHz with an 8-MB L3 cache and a 32-GB RAM at 1600 MHz. We implemented the PFTrie in *C*, and used the *Clang* compiler with the optimization level 2 to compile the code. We also created 50 synthetic traffic traces that contain 150 thousand to 64 million IP addresses; for each size we created five traces with different levels of source address locality, with 0%, 25%, 50%, 75%, and 100% addresses, respectively, that belong to the same IP prefix and can be aggregated.

2) Speed of PathFinder (i.e., PFTrie): We compare PFTrie's performance against Adaptive Radix Tree (ART) [18] and the well-known Generalized Prefix Tree [19] (also called Patricia Trie) under the two scenarios defined in Sec. VII-A. We use the synthetic traces that contain one to four magnitude



(a) put_a_new scenario.



(b) put_an_old scenario.

Fig. 7: **PFTrie speed.**

more IP addresses, in order to evaluate the three different data structures under stress.

Fig. 7a shows the comparison results under the put_a_new scenario for storing all IP addresses in a trace as new addresses into a data structure. For every synthetic trace size ranging from 160,000 to 64 million source addresses, when storing a new IP address or prefix, PFTrie always outperforms ART and Patricia. For example, to store 16 million IP addresses, it takes more than 1300ms for ART but it only takes around 700ms for PFTrie. In general, PFTrie spends 50% less time than ART to store the same number of IP addresses. Even to store 64 million IP addresses, it takes only 2.93s.

Fig. 7b shows results under the put_an_old scenario for performing 15 million *put* processes of storing an IP address or prefix already stored. Here, the time needed by PFTrie is virtually constant at about 27.0ms, much less than that in the put_a_new scenario; e.g., we can deduce with 64 million IP addresses, it would be about 115ms as opposed to 2.93s in the put_a_new scenario. Moreover, the time is also further less compared to ART and Patricia, and PFTrie is at least 10 times faster than ART in every case. This speed is because of the PFTrie optimizations we introduced, including the top-down collapse of prefixes (Sec. V-B2) and the avoidance of duplicate traversals (Sec. V-B3).

While the PFTrie operations are a combination of the two scenarios, from various real-world traces we notice that the put_an_old scenario is more frequent than the put_a_new scenario. For example, in the Booter 3 DDoS trace [20][21], the put_an_old scenario will happen 369 times more than the put_a_new scenario.

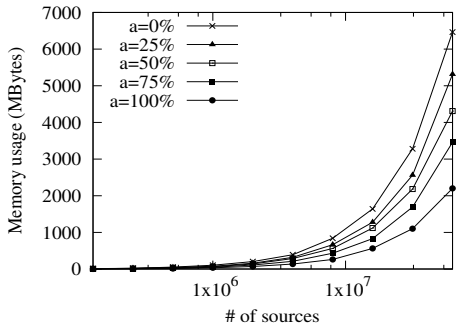


Fig. 8: **PFTrie memory overhead across five different source profiles.** Each profile has a different percentage of aggregatable addresses (a).

3) Overhead of PFTrie: We are particularly interested in the memory cost of PFTrie when there are millions of IP source addresses. We used synthetic traces that include a huge number of IP source addresses, and evaluated the memory usage for each number of sources under five different profiles, as shown in Fig. 8. We can see the logarithm of the memory cost is basically a linear function of the logarithm of the number of sources, and overall the memory cost is manageable under all five profiles. Moreover, a profile with a higher address locality can have much lower memory cost. This feature is due to the optimization via bottom-up aggregation of PFTrie leaf nodes (Sec. V-B1). Because of the tree nature of PFTrie, the memory cost complexity for storing 2^n addresses in a PFTrie is $O(2^n)$ with n levels of nodes, but if it shrinks to k levels, the memory cost will become $O(2^k)$, a reduction of $O(2^{n-k})$ times.

We also evaluated the network overhead across 25 different AS-level Internet topologies, using one million IP source addresses. For each AS-level topology, we assigned every IP address to an AS, where the number of addresses assigned to each AS is proportional to its IP address space size. Fig. 9 shows the network transmission overhead for an AS to transmit the PFTrie for 1 million source addresses to a proxy. Clearly, the further away an AS is from the victim, the smaller the network overhead it introduces. The AS that is the last hop to reach the victim would see traffic from all addresses, thus incurring the largest overhead, but only about 3.9MB.

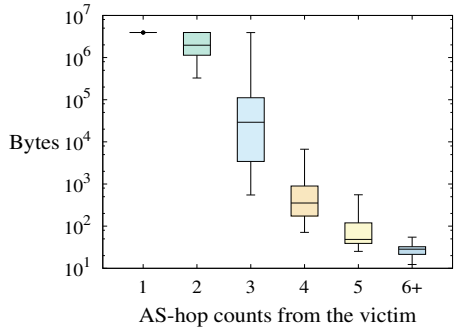


Fig. 9: **Network overhead in transmitting PFTries of 1 million source addresses.**

VIII. Discussions and Open Issues

PathFinder is an approach to obtaining the DDoS traffic footprints at the Internet scale, and we have made many design choices in order to provide a line-rate, cost-effective, and deployable solution. Nonetheless, some issues remain to be addressed due to space limitations:

One obvious issue is IP spoofing. Clearly, nothing would be affected if a monitor's mode of operation is source-agnostic, but if a user requests source-based traffic footprints, the PathFinder system may learn some source IP addresses that are spoofed. An attacker may even generate many spoofed sources to overwhelm every monitor and the proxy of the user. We point out that even if a source address is spoofed, the path that the user learns about the source will still be valid, since the monitor that reported the source was on the path of the packet with the spoofed source. Furthermore, if the user notices multiple paths for the same source address or prefix, the user will know that either a routing change occurred, or at least some of them are spoofed sources.

We have also assumed that every AS (via its PathFinder monitor) is willing and able to communicate with any PathFinder proxy for the common good from DDoS defense. While we have shown the traffic overhead when a proxy communicates with every AS is not concern (Sec. VII-C3), it is likely that this assumption is not true for *some* ASes due to incentive or connectivity issues, which we treat as an open issue out of the scope of this paper.

Another issue is to make PathFinder work for IPv6. In fact, the design of the PFTrie is independent of the length of an IP address and works for both IPv4 and IPv6. In the future, we plan to evaluate its speed and memory cost when handling millions of IPv6 addresses.

We do not fully discuss the security of PathFinder. We assume that every node in the PathFinder system must be authenticated before it can talk to other nodes in the system. We also assume that the system employs state-of-the-art defense mechanisms to protect itself against any security attacks; for example, a PathFinder proxy can employ a DDoS defense solution to protect itself and its communication with PathFinder monitors from DDoS attacks.

IX. Conclusions

While DDoS attacks have become more frequent and damaging and, once launched, can cause severe damage to services on the Internet, defense against DDoS attacks has often suffered from the lack of relevant knowledge of the DDoS traffic. However, it is fairly challenging to grasp the topological nature of the DDoS traffic while the attack is occurring: the DDoS traffic often originates from many different locations, follows various paths to reach the victim, sometimes carry spoofed source addresses, and can be extremely dynamic. Currently the best options are various IP traceback or path inference approaches, but they impose stringent demands to run and deploy. We fill this gap by proposing the PathFinder system as a service that a DDoS defense system can use to obtain the footprints of the DDoS traffic to a victim, including specifying many details of the footprints such as whether the source address and/or bandwidth information is needed. In particular, PathFinder embraces an architecture that not only eases its deployment in today's Internet, but also ensures it has a low cost (e.g., its on-demand model) and is fast to meet the line rate of the packets it must capture.

References

[1] Akamai. (2016) Q4 2016 state of the Internet security report. http://www.akamai.com/us/en/about/our-thinking/state-of-the-internet-report/global-state-of-the-internet-security-ddos-attack-reports.jsp.

[2] K. York. (2016) Dyn statement on 10/21/2016 DDoS attack. http://dyn.com/blog/dyn-statement-on-10212016-ddos-attack.

[3] H. Burch and B. Cheswick, "Tracing anonymous packets to their approximate source," in *USENIX Large Installation System Administration Conference (LISA)*, 2000, pp. 319–327.

[4] K. Singh, P. Singh, and K. Kumar, "A systematic review of IP traceback schemes for denial of service attacks," *Computers & Security*, vol. 56, pp. 111–139, 2016.

[5] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Practical network support for IP traceback," in *ACM SIGCOMM*, 2000, pp. 295–306.

[6] A. Yaar, A. Perrig, and D. Song, "FIT: fast Internet traceback," in *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 2, 2005, pp. 1395–1406.

[7] K. J. Argyraki and D. R. Cheriton, "Active Internet traffic filtering: Real-time response to denial-of-service attacks," in *USENIX Annual Technical Conference*, 2005, pp. 135–148.

[8] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer, "Hash-based IP traceback," in *ACM SIGCOMM*, 2001, pp. 3–14.

[9] J. Li, M. Sung, J. Xu, and L. Li, "Large-scale IP traceback in high-speed Internet: Practical techniques and theoretical foundation," in *IEEE Symposium on Security and Privacy*, 2004, pp. 115–129.

[10] Z. M. Mao, L. Qiu, J. Wang, and Y. Zhang, "On AS-level path inference," in *ACM SIGMETRICS*, 2005, pp. 339–349.

[11] E. Katz-Bassett, H. V. Madhyastha, V. K. Adhikari, C. Scott, J. Sherry, P. Van Wesep, T. E. Anderson, and A. Krishnamurthy, "Reverse traceroute," in *USENIX Symposium on Networked Systems Design and Implementation*, vol. 10, 2010, pp. 219–234.

[12] Cisco. (2016) Cisco IOS IP routing: BGP command reference. http://www.cisco.com/c/en/us/td/docs/ios/iproute_bgp/command/reference/irg_book/irg_bgp5.html.

[13] J. Stretch. (2016) JUNOS-to-Cisco IOS/XR command reference. http://web.archive.org/web/20140114070827/http://packetlife.net/wiki/junos-cisco-iosxr-command-reference.

[14] Cisco. (2016) Catalyst switched port analyzer (SPAN) configuration example. http://www.cisco.com/c/en/us/support/docs/switches/catalyst-6500-series-switches/10570-41.html.

[15] Juniper. (2016) Example: Configuring port mirroring for local monitoring of employee resource use on EX series switches. https://www.juniper.net/documentation/en_US/junos/topics/example/port-mirroring-local-ex-series.html.

[16] Cisco. (2007) Configuring secure shell on routers and switches running cisco ios. http://www.cisco.com/c/en/us/support/docs/security-vpn/secure-shell-ssh/4145-ssh.html.

[17] Juniper. (2015) Configuring SSH service for remote access to the router or switch. https://www.juniper.net/documentation/en_US/junos/topics/task/configuration/ssh-services-configuring.html.

[18] V. Leis, A. Kemper, and T. Neumann, "The adaptive radix tree: ARTful indexing for main-memory databases," in *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pp. 38–49.

[19] D. R. Morrison, "PATRICIA—practical algorithm to retrieve information coded in alphanumeric," *Journal of the ACM*, vol. 15, no. 4, pp. 514–534, 1968.

[20] SimpleWeb.org. (2015) Traces. https://www.simpleweb.org/wiki/index.php/Traces.

[21] J. Santanna, R. van Rijswijk-Deij, R. Hofstede, A. Sperotto, M. Wierbosch, L. Zambenedetti Granville, and A. Pras, "Booters - an analysis of ddos-as-a-service attacks," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2015, pp. 243–251.