On Capturing DDoS Traffic Footprints on the Internet

Lumin Shi, Jun Li, Mingwei Zhang, and Peter Reiher

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. PathFinder employs an architecture that is easy to implement and deploy on today's Internet, a PFTrie data structure that introduces multiple design features to log traffic at line rate, and streaming and zooming mechanisms that facilitates the storage and transmission of DDoS footprints more efficiently. Our evaluation shows that PathFinder can significantly improve the efficacy of a DDoS defense system, its PFTrie data structure is fast and has a manageable overhead, and its streaming and zooming mechanisms significantly reduce the delay and overhead in transmitting DDoS footprints.

Index Terms—Distributed Denial-of-Service; DDoS; Traffic Footprint; Autonomous System (AS); PFTrie

1 Introduction

Today's Internet is vulnerable to distributed denial-of-service (DDoS) attacks. During a DDoS attack, an attacker controls many compromised machines to send unwanted traffic toward the victim in order to exhaust the network or computational resources needed to access the victim. DDoS attacks have become more frequent and damaging to many network services [1]. An attacker with ample resources can easily launch DDoS attacks with an overwhelming traffic volume and take down nearly any Internet service. For instance, a large-scale DDoS attack on Dyn [2] easily disabled its domain name service and crippled many major web services that relied on it such as Twitter, PayPal, and over fifty others for hours. DDoS attacks have also become more sophisticated, such as those described in [3], [4], [5].

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 **DDoS traffic footprints**, if available, can enable a DDoS defense system to become more informed and thus more effective in handling DDoS attacks. For example, it may learn which autonomous systems (ASes) or AS paths have seen a large amount of traffic to the victim, thus more preferably deploying DDoS traffic filters there; 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. Fig. 1 shows an example: while the DDoS traffic toward the victim originates from—and is forwarded by—many ASes, if the DDoS defense system knows the AS paths of the DDoS traffic toward the victim, it can request that specific ASes on the paths, such as AS 2 and AS 6, filter the DDoS traffic.

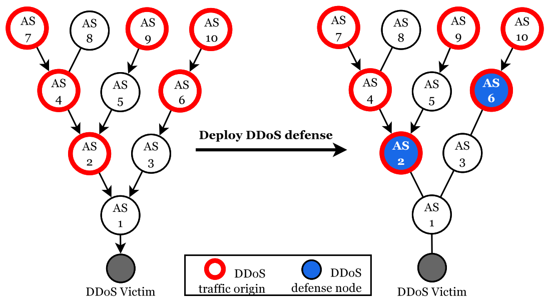


Fig. 1: **A DDoS attack and defense example.**

In fact, even when facing more complex DDoS scenarios, a DDoS defense system can still benefit from DDoS traffic footprints. First, when a DDoS attack employs large-scale IP spoofing, as long as the defense can determine which traffic is DDoS traffic, with the DDoS traffic footprints, such as the paths of DDoS traffic or which ASes carry the traffic, the defense will still know correctly where to deploy DDoS traffic filters *en route* and mitigate the DDoS attack effectively. On the other hand, the traffic footprints can help detect IP spoofing if they show that traffic from the same IP addresses to the DDoS victim originated from different ASes. (Here, we leave out the details on determining whether given traffic is DDoS or spoofed (or both), which

• *L. Shi, J. Li, and M. Zhang are with the Department of Computer and Information Science, University of Oregon, Eugene, OR, 97403. E-mail: {luminshi, lijun, mingwei}@cs.uoregon.edu.*
• *P. Reiher is with the Department of Computer Science, University of California, Los Angeles, CA, 90095. E-mail: reiher@cs.ucla.edu.*

Manuscript received July 1, 2020; revised March 1, 2021; accepted April 6, 2021. (Corresponding author: J. Li.)

is the responsibility of DDoS defense itself and out of the scope of this paper.) A potentially concerning issue is the dynamic nature of DDoS traffic paths since a DDoS attack may periodically switch to a new batch of DDoS bots to launch its attack, likely from different paths [6], [7]. Here, we can safely assume the resource for any DDoS attack is limited, including the number of DDoS bots as well as the paths incurred by the DDoS traffic. So, while it is possible for a DDoS attack to change the bots and shift the attack paths, it will eventually exhaust all potential bots and possible attack paths, and a DDoS defense system can still benefit if it learned the IP sources of bots and their paths used to attack a DDoS victim. Plus, as the DDoS traffic approaches the victim, even if the DDoS traffic keeps changing their paths, the paths will converge, allowing the PathFinder to discover their common hops where DDoS filters may be deployed. Finally, some DDoS attacks are transient; however, as long as they last longer than what it takes to learn and report their footprints, a DDoS defense solution can still leverage the footprints in mitigating such DDoS attacks.

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, as we will discuss in more detail in Sec. 2, these approaches have serious drawbacks. For example, the path inference methods are often inaccurate; 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 IP addresses or prefixes of DDoS sources.

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. PathFinder does not require *every* AS to participate in order for PathFinder to benefit DDoS defense.
- 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, with a set of PFTrie optimization methods.
- We design a streaming mechanism that manages the timing of transmitting traffic footprints, including adaptively adjusting the transmission interval to lower the overhead.
- We also introduce a new mechanism called *zooming* that enables PathFinder to dynamically adjust its granularity in collecting DDoS footprints for better efficiency.
- We evaluate PathFinder and show that PathFinder can significantly improve the efficacy and efficiency of DDoS defense, its PFTrie is fast and has a manageable overhead, its streaming and zooming mechanisms are fast and efficient, and PathFinder can also sample DDoS traffic at certain rate over a time window to still capture DDoS footprints with high accuracy.

The rest of this paper is organized as follows. We first discuss related work in Sec. 2, followed by an overview of PathFinder in Sec. 3. We then describe individual components of PathFinder, including the PathFinder monitor in Sec. 4, the PFTrie data structure for traffic logging in Sec. 5, the PathFinder proxy in Sec. 6, the streaming mechanism in Sec. 7, and the zooming mechanism in Sec. 8. We evaluate PathFinder in Sec. 9, discuss several open issues in Sec. 10, and conclude the paper in Sec. 11. This work is an extended version of our conference paper [8]. In comparison to the conference paper, this work introduces streaming and zooming mechanisms and their evaluation results, conducts new evaluation of PathFinder's efficacy in partial deployment scenarios and the traffic capturing strategy, and updates the discussions on system scalability, security, and deployment issues.

2 Related Work

DDoS defense can be broadly categorized into on-the-path defense and cloud-based defense. On-the-path defense requires a node (a router or an AS) that is on the paths of DDoS traffic toward the victim to filter the DDoS traffic. As defined in [9], on-the-path defense may happen at the victim networks, which may be too late to mitigate an overwhelming amount of DDoS traffic, or at the source ends where the DDoS traffic originate, which then requires to locate a large number of DDoS sources and deploy defenses there. It is thus no surprising that DDoS defense not at either end, i.e., in-network defense, have become attractive; many in-network solutions exist, including Poseidon [10], VIF [11], Stellar [12], SENSS [13], DrawBridge [14], StopIt [15], and TVA [16]. A crucial challenge here is to know with little delay the DDoS footprints such as the paths of DDoS traffic and the volume distribution along different directions in order to deploy DDoS filters at the right nodes *en route*. In contrast, cloud-based defense, also known as scrubbing-center-based defense, such as those defined in [17], [18], first redirects a victim's traffic via the DNS and/or BGP protocols to a traffic scrubbing service, which then cleanses the traffic by filtering out the DDoS traffic before forwarding the cleaned traffic to the victim. Cloud-based defense can introduce extra overhead and additional delay as the traffic has to travel extra hops before reaching the victim; moreover, it may not be able to redirect enough attack traffic as attackers could bypass DNS and/or BGP. Clearly, if an in-network DDoS defense knew the paths of DDoS traffic, or more generally, the DDoS traffic footprints, it would become more informed and thus effective, including becoming more competitive against cloud-based solutions.

2.1 IP Traceback

IP traceback, first introduced in [19], 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 [20], marking and logging are the two most well-developed approaches.

In a marking approach, such as those described in [21], [22], [23], [24], [25], [26], [27], 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 those described in [28], [29], [30], as 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 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 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 as well as PathFinder, on the other hand, so long as their monitoring mechanism can process packet headers at line speed, can log packet information with little loss and reach a high accuracy.
- *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. Further, as marking approaches overload existing IP header fields with marks, they may interfere with the defined functionality of those fields and even cause confusion at network equipments that are not aware about marking functionalities. 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.

2.2 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 [31] 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 [32] 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, they are generally subject to some degree of inaccuracy (e.g., the accuracy from the research in [31] is between 70% and 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 inferred path. Similarly, path inference approaches are not suited for finding the paths of DDoS traffic using spoofed IP addresses, as they require the true IP addresses of DDoS bots to infer the paths of DDoS traffic.

3 PathFinder Overview

3.1 PathFinder as a Service for DDoS Defense

We designed PathFinder as a service for DDoS defense. Upon request from a DDoS defense system, 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.

Note that DDoS defense systems typically inform ASes, not routers or mitigation appliances in those ASes, which traffic should be filtered. It is up to an AS to determine internal locations of filtering DDoS traffic. PathFinder therefore only needs to record AS-level paths of DDoS attacks.

When requesting the PathFinder service, a DDoS defense system 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 to learn all the /24 prefixes; /32 to learn all the /32 prefixes, i.e., all the source IP addresses); *and*
- whether to collect the bandwidth consumption of the traffic, and if so, the granularity (per source address, per source prefix, or per AS *en route*) and the unit (packets per second or bits per second).

3.2 Architecture

PathFinder is a log-based system that enables a client—which is any DDoS defense system in this paper—to learn the AS paths, sources, bandwidth consumption, and other information concerning the traffic toward a DDoS victim, i.e., the footprints of the traffic. As we will show in the rest of this paper, PathFinder 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.

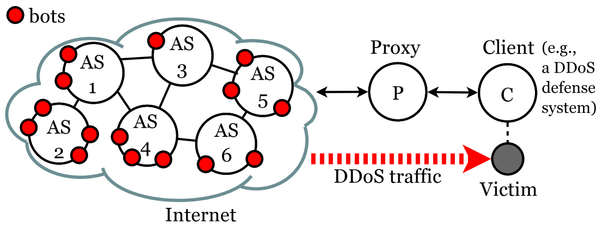


Fig. 2: **PathFinder architecture.**

Fig. 2 shows the high-level architecture of PathFinder. It consists of three types of entities:

- **PathFinder clients** which interact with their proxy to request the PathFinder service and retrieve from their proxy the footprints of the traffic to a DDoS victim;
- **PathFinder proxies** which (1) pass their clients' requests (including all parameters described in Sec. 3.1) to the PathFinder monitors at all participating ASes; (2) receive and process PathFinder logs (**PFLogs**) from these monitors, which they use to derive the DDoS traffic footprints; and (3) return the footprints to their clients; *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 client 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 avoiding any reliance on inter-AS collaboration.

4 PathFinder Monitor

4.1 Addressing Design Requirements

The monitor at each PathFinder-participating AS faces two design requirements. First, the monitor must consult routers within the AS to learn the AS path from the AS to the victim. Second, it must access the traffic toward the victim in order to record their source addresses and/or amounts, if requested. As an AS can have a complicated topology with inter-connected border routers and internal 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 on that path in order to learn its AS path to the victim and to access the traffic it forwards to the victim. For the former (i.e., to learn the AS path), as every border router of the AS runs BGP and maintains a Routing Information Base (RIB), the monitor can query the RIB at a border router of the AS. Note that BGP implementations such as Cisco IOS [33] and FRRouting [34] all support such a query. For the latter (i.e., to access the traffic), the monitor must apply traffic mirroring or tapping techniques (we rule out possible hardware telemetry support from routers; although they produce traffic records in formats like NetFlow or IPFIX, the records are only exported at a fixed interval, often with a long delay).

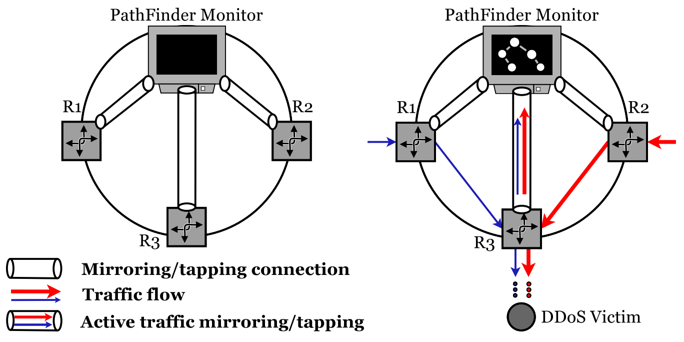
The monitor may further face a third requirement if it needs to produce PFLogs to record traffic sources as well as source-based information such as traffic amounts per source. There could be a huge amount of traffic from many distinct sources toward the victim, especially if the victim is currently under a severe DDoS attack, thus making it challenging for the monitor to record all the sources and their corresponding bandwidth consumption at 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 listing which specific source IP addresses or prefixes it has seen. A hash table is better, but it is not flexible in processing or aggregating IP address and prefix information, thus not scalable 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 Sec. 5.

4.2 Setup

A PathFinder-participating AS must set up its PathFinder monitor and its working environment. First, the monitor needs to arrange traffic mirroring or tapping with every border router in order to obtain their traffic *in real time* when needed. To do so, given the autonomy of ASes, each AS may adopt its own preferred procedure. For a small AS without many routers, it can physically wire the monitor with every router for wiretapping (Fig. 3a shows an example). For a large AS with many routers over a large geographic region, it can use virtual circuits for traffic mirroring with the routers [35], [36]. 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 must be able to remotely login to each router that it is wired with and execute commands on that 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 [37], [38].

Finally, the monitor must be easily discoverable 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 maintain its IP address and other information 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



(a) The setup of a PathFinder-participating AS. (b) The snapshot when the monitor is capturing traffic.

Fig. 3: **An example setup of a PathFinder-participating AS.**

PathFinder monitor of `AS 3582`. The AS can also, at its discretion, use standard access control mechanisms to limit which remote parties can query its PathFinder monitor.

4.3 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. 3.1, and start to generate PFLogs on behalf of the victim.

The first step that the monitor takes is to identify a set of routers in its AS that are both necessary and sufficient 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 its own routers. We therefore use all possible egress routers from which the traffic to the victim may exit the AS. For example, in Fig. 3b, 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.

Once the routers are selected, the monitor then talks with them to collect and produce PFLogs. First, the monitor will query each selected router to retrieve its AS path to the victim from its RIB. Furthermore, if the client 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 only the traffic to the victim is mirrored or tapped and, more importantly, this traffic does not share the path of the production traffic, so it will not interfere with the production traffic.

If the client did not request the source addresses or prefixes of the traffic, or source-based bandwidth consumption or other information, PathFinder operates in **source-agnostic mode**. In this mode, the procedure already outlined above has collected all the PFLogs that the client requires. Otherwise, the monitor will operate in the **source-aware mode**, which requires gathering further information to produce source-based PFLogs, using the PFTrie data structure (see Sec. 5). For both modes, the monitor employs a **streaming** mechanism (see Sec. 7) to deliver the PFLogs to the proxy of the client.

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

5 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. Due to its speed, the trie data structure has been popular in storing IP addresses and prefixes for other purposes, such as those in the Forwarding Information Base (FIB) of software 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. 4.1, we enhance the trie data structure and design as follows the PFTrie—a PathFinder data structure for logging traffic.

5.1 Basic PFTrie Operations

The monitor captures and processes every packet toward the victim. It stores the IP source address of the packet into the PFTrie via a **put** process. In this process, the monitor may modify the PFTrie by adding new nodes to store the IP address, or it may discover that it has already created those nodes due to a previous packet with the same IP address. In either case, the put process will return a node representing the IP address, 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 becomes the new anchor.

(2) If 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 return the new anchor node. (Note the returned node is always a leaf node on the trie.) 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.

Fig. 4 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, for which it adds a new node c. Now that the entire address is processed, the process returns node c representing the newly stored IP address.

5.2 PFTrie Optimization

We further optimize the PFTrie for a faster traversal process. First, with the design in Sec. 5.1, for every bit of an IP address, the trie must maintain a corresponding 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 independent optimization methods: the bottom-up aggregation of leaf nodes 5.2.1 and the top-down collapse of prefixes 5.2.2. Furthermore, we introduce a method to avoid duplicate traversal of the PFTrie when a source address is already stored 5.2.3. We describe each method below.

5.2.1 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. 5, we can aggregate the two leaf nodes into a 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 sum the bandwidth consumption of the two leaf nodes, recording the bandwidth consumption of the IP prefix represented by the new leaf node. The choice here depends on the client'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).

5.2.2 Top-down Collapse of Prefixes

During a *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. 6 shows an example of collapsing /24 prefixes into an array (the monitor can collapse prefixes of other lengths similarly, such as all the /16 prefixes). We populate the array with all /24 prefixes that exist in the PFTrie. For every /24 prefix, the monitor treats the 24 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.

5.2.3 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 packets from the same IP address, the overhead will multiply for 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; in other words, instead of setting a bit to `1`, we insert a pointer to the leaf node into the array. 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).

6 PathFinder Proxy

6.1 Addressing Design Requirements

The purpose of a PathFinder proxy is to act as an intermediary between PathFinder clients and PathFinder monitors and help the clients learn the footprints of the traffic toward a DDoS victim. Since there can be multiple ASes on the AS-path of the traffic, when the PathFinder monitors of such ASes report the AS-paths of the traffic, the paths they report may overlap. The proxy must determine which AS-path includes the largest number of ASes. Furthermore, if a 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 a client, clearly an on-demand mode is best for the proxy.

6.2 Setup

Every PathFinder proxy will make itself available to potential PathFinder clients. If a PathFinder client needs

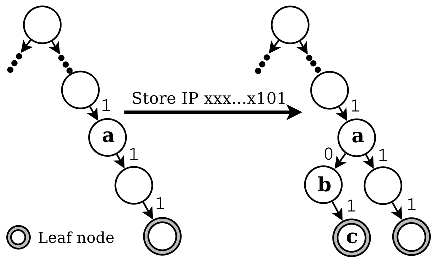


Fig. 4: **Store an IP address that ends with 101.**

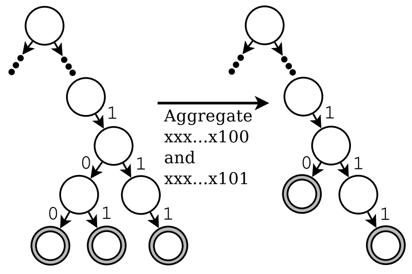


Fig. 5: **PFTrie optimization: Aggregating sibling leaf nodes into one new leaf node.**

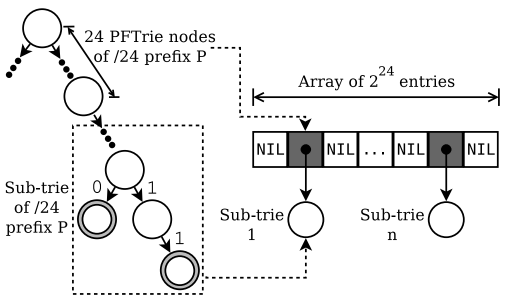


Fig. 6: **PFTrie optimization: Collapsing all /24 prefixes into an array with 2^{24} entries.**

PathFinder service, it will register itself at a proxy, including setting up all necessary security credentials. The client 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. 4.2.

6.3 Operation

Like any PathFinder monitor, every proxy also operates on demand. Once a proxy receives a request from a client, it will verify if the request is authentic and valid or not, 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). The proxy then receives the PFLogs from monitors via the streaming mechanism (Sec. 7), processes them (including merging PFLogs to save storage and transmission overhead), and forwards PFLogs to the client (also via the streaming mechanism).

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 merges the PFLogs. Specifically, it 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 client, together with the bandwidth information if requested.

However, if the footprints need to be source-based, the proxy will construct a local PFTrie by merging the PFTries it receives from monitors. For each leaf node of the PFTrie from each monitor, which represents an IP address or prefix `S` that the monitor has captured, the proxy will store `S` in its local PFTrie, following the same *put* process described in Sec. 5.1. Furthermore, assuming the monitor's AS is `AS k`, the proxy also marks the leaf node that represents `S` with `k`, to indicate the AS-path of traffic from `S` to the victim is the AS-path from `AS k` to the victim. However, if `S` is already in the local PFTrie, the proxy will retrieve the marked AS number of the leaf node for `S`, say `o`, and mark the leaf node for `S` with either `k` or leave it as `o`, whichever one is upstream of the other. As a result, the proxy then builds a local PFTrie that contains the most complete AS-path from `S` to the victim, and forwards it to the client.

7 PathFinder Streaming

A DDoS defense system is more effective if it can respond to a DDoS attack at an early stage, which is only possible if it obtains necessary information about the attack (such as DDoS traffic footprints) with little delay after the attack starts. The methods described in Secs. 4 and 6 allow a PathFinder monitor and proxy to collect PFLogs (i.e., DDoS traffic footprints). Yet, they do not address *when* a monitor should transmit PFLogs to a proxy or a proxy transmit them to a client to achieve the most timely notification of attack characteristics. PathFinder addresses this issue using a **streaming mechanism**. We first discuss what factors to consider in designing the mechanism, then describe our design of the PathFinder streaming.

7.1 Factors to Consider

There are two primary, orthogonal factors to consider when deciding *when* to transmit PFLogs. The first factor is the *preferred interval or frequency* for a PathFinder monitor or proxy to periodically offload PFLogs, as indicated by the DDoS defense system. On one hand, a short interval (i.e., a high frequency) is better, as short intervals allow the DDoS defense system to use information in close to real time. However, short intervals potentially lead to small updates that only contain footprints from a brief time window. Also, shorter intervals incur a higher total network bandwidth overhead. Further, frequent updates may be redundant due to traffic locality and cause each update to more likely overlap with previous updates. On the other hand, a long interval (i.e., a low frequency) would cause a long delay in informing the DDoS defense system about the ongoing DDoS attack. Also, using a long interval may fail to detect certain DDoS attacks; for example, one may need an interval of 200 ms to detect a pulsing attack such as CICADAS [4].

Another factor to consider is the *limited size of the PFLogs buffer* that each PathFinder monitor or proxy allocates to store the PFLogs for its clients. If a client is under source-agnostic mode, its PFLogs (i.e., AS paths and bandwidth information) will be of a small size and a buffer of a fixed small size will be sufficient. On the contrary, if a client is under source-aware mode, the monitor or proxy will be maintaining PFTries in order to log source-based traffic footprints for this client, which can become fairly large. The monitor or proxy then needs to dynamically allocate the buffer space for this client. As it collects more traffic footprints and stores them in the buffer, if the buffer is becoming full it can try to allocate more space to store PFLogs. However, it may not succeed in allocating new space every time, especially if there is an enormous amount of traffic footprints for the client, the interval to offload PFLogs is relatively long, or there are also many other clients to serve simultaneously. In this case, rather than replacing some PFLogs in the buffer to lose these PFLogs, it is preferable to offload PFLogs of this client or other clients to make more space.

7.2 Streaming Design

We use a receiver-driven approach to have the client set the preferred interval for receiving streamed PFLogs. When a proxy or a monitor receives a request for collecting traffic footprint for a client, it also learns the preferred interval of transmitting PFLogs for the client, which we call the **streaming interval**. Further, if the client is under source-agnostic mode, it will allocate a fixed-sized buffer for PFLogs. Otherwise, the client is under source-aware mode and the proxy or monitor then dynamically allocates a PFLogs buffer for the client. The proxy or monitor then transmits PFLogs of the client at the streaming interval for the client, which we call **in-sync streaming**. PFLogs transmitted then makes space for new PFLogs in the next streaming interval. If a monitor or proxy cannot allocate more buffer space for a client under source-aware mode, but the streaming interval for the client is not up yet, it also immediately transmits enough PFLogs of the clients with the most PFLogs to free half the buffer space. (These clients may or may not include the client in question.) We call this streaming **out-of-sync streaming**. (PathFinder also supports a third streaming method called on-demand streaming, which is not the focus of this paper; basically, at any time if needed by DDoS defense, a client may also request its proxy to transmit the current PFLogs to the client, and the proxy in turn can request PFLogs from monitors, thus causing on-demand streaming.)

A PathFinder client under source-aware mode thus may receive PFLogs via in-sync or out-of-sync streaming from its proxy, which in turn may receive PFLogs via multiple in-sync or out-of-sync streamings from monitors. While every streaming above uses an independent channel and can be either in-sync or out-of-sync, it is very likely that if the streaming to the client is out-of-sync, some or many streamings to the client's proxy are also out-of-sync. When out-of-sync streaming happens, although the client can continue to use the current streaming interval, as discussed in Sec. 7.1, it is important for the client to use a reasonable streaming interval in order to conduct timely, effective defense of DDoS. However, it is often hard for the client to know how long its streaming interval should be, and the dynamic nature of DDoS traffic only makes it harder.

PathFinder allows its client under source-aware mode to either use a fixed interval for streaming, i.e., **fixed streaming**, or adaptively adjust its streaming interval, i.e., **adaptive streaming**. Via the adaptive streaming, a client can choose an initial streaming interval, such as a default value of 1 second, and then adjust this interval using the following two complementary strategies.

The first strategy uses the number of new DDoS sources from an in-sync streaming to adjust the interval. Each time an in-sync streaming occurs after a current interval, the client inspects PFLogs from the interval to calculate the number of new DDoS sources per time unit, i.e., the current rate of new DDoS sources, or S_{curr}. It then uses an exponential moving average method to estimate the average rate of new DDoS sources, or S, using the follow equations:

$$S = S_{curr} \tag{1}$$

$$S = \alpha S_{curr} + (1-\alpha)S \tag{2}$$

where Equation (1) is the estimate after the first in-sync streaming, and Equation (2) is the estimate after an in-sync streaming, with α being configured to a suitable value, such as 0.5, by the client. If the client wishes to receive a specific number of new DDoS sources in the next interval, say N, it then can set the next streaming interval as $\frac{N}{S}$.

The second strategy uses the occurrence of out-of-sync streaming as a signal to indicate that the buffer space at the proxy and probably some monitors are full and that the client should use a shorter streaming interval. So, whenever an out-of-sync streaming happens, a client will reset its streaming interval to be half of the previous streaming interval value. Furthermore, the client will also record this interval value as a threshold, such that if later the first strategy is applied to adjust the streaming interval, the new interval value should not surpass this threshold.

8 On-Demand Zooming

Sometimes a DDoS attack can be of a very large scale where the DDoS traffic may come from millions of IP addresses. If a PathFinder client wishes to capture DDoS traffic footprints of such an attack under source-aware mode, PathFinder monitors and proxies may have to store and transmit a high volume of PFLogs, while the PathFinder client may also be overwhelmed by the volume of PFLogs it receives. The latency of streaming will also become higher, potentially lengthening the DDoS defense response time. The PFTrie optimization techniques we introduced in Sec. 5.2 may help, but not always. For example, as there is no guarantee in DDoS traffic locality, leaf nodes of PFTrie may not be aggregatable; or, if the attack involves a lot of unique source IP addresses, many new IPs need to be recorded in the PFTrie, thus not causing duplicate traversals that can be avoided.

PathFinder addresses this issue using an **on-demand zooming mechanism** that can significantly reduce the size of PFLogs. Specifically, when a PathFinder client initially issues a request for source-aware PFLogs, it can specify a coarse granularity for recording the traffic footprints. Only if the client needs to know the DDoS sources at a finer granularity later will the client *zoom in* and request PFLogs of DDoS sources at a finer granularity. It can repeat this on-demand zooming process until it is satisfied with the granularity. Similarly, a client can also zoom out and begin to request PFLogs at a coarser granularity.

For example, initially, a client can ask for traffic footprints of /16 IP prefixes. As it receives PFLogs of /16 IP prefixes, because a /16 IP prefix consumes the most network bandwidth, it decides to zoom in on this prefix to collect PFLogs about it at a finer granularity. It thus issues a new request of PFLogs; in addition to still requesting PFLogs of other /16 IP prefixes, it also requests PFLogs of /20 sub-prefixes of the /16 IP prefix in question. The on-demand zooming process can repeat recursively; for instance, the client may further zoom in on /24 sub-prefixes on some of the /20 sub-prefixes, or subsequently on IP addresses of certain /24 sub-prefixes.

The benefits of on-demand zooming are obvious. By only zooming in on IP prefixes of interest, monitors and proxies could store PFTries at a coarse granularity for most IP Prefixes. The depth of PFTries could be shortened, and the size of PFTries could become smaller. For instance, before zooming in on any IP prefixes, a PFTrie storing */p* IP prefixes will be no more than *p* levels; compared to a 32-level PFTrie storing IPv4 addresses (i.e., /32 IP prefixes), it has a significant storage and transmission overhead reduction of $O(2^{16})$. After zooming in on certain IP prefixes, the savings on storage and transmission overhead will become less. However, unless the on-demand zooming mechanism is repeatedly used and eventually results in a PFTrie that records every IP address, which rarely happens, the savings will continue to be significant.

On-demand zooming also increases the chance to apply the PFTrie optimization methods from Sec. 5.2. Assume a PFTrie leaf node representing an `X0`/*p* IP prefix, where `X` represents the first *p-1* bits of the prefix. First, it is more likely to optimize a PFTrie via the bottom-up aggregation of leaf nodes (Sec. 5.2.1). As long as there is another packet from IP prefix `X1`/*p*, we can aggregate this leaf node with a new leaf node representing `X1`/*p* and replace them with their parent node `X`/*p-1* as a new leaf node. Clearly, the smaller *p* is, the bigger address space we have with `X1`/*p*, thus the more likely for this optimization to happen. Second, the smaller *p* is, the bigger address space the leaf node represents, the more likely to have more packets from this space to avoid duplicate traversal of the PFTrie, another optimization method (Sec. 5.2.3).

9 Evaluation

9.1 Goals

We now show our evaluation of 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 built 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 fewer resource and achieves a much higher level of success. We then show PathFinder is beneficial even when only partially deployed.
- *Speed and overhead of PFTrie:* 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 `S` in a PFTrie under two scenarios. In one scenario, `S` is new and the PFTrie needs to be updated with a set of new nodes, including the leaf node that represents `S`. In another scenario, `S` is in the PFTrie, so the *put* process will check and return the current leaf node that represents `S`. We call these two scenarios **put_a_new** and **put_an_old**, respectively. Further, we evaluate the memory overhead of the PFTrie at a monitor or proxy when producing source-aware footprints and the network overhead when transmitting a PFTrie.
- *Benefits of PathFinder streaming:* PathFinder uses a streaming mechanism to enable its clients, i.e., DDoS defense systems, to obtain the DDoS traffic footprints quickly. Via our simulation, we evaluate the delays and network overhead when using streaming to obtain traffic footprints. In particular, we show how a client under source-aware mode can use adaptive streaming to achieve both timely traffic footprint delivery and low network overhead.
- *Benefits of PathFinder zooming:* In addition to designing optimization methods to save the overhead of PFTries, we further designed the zooming mechanism that can keep PFTries at a potentially much smaller size and only grow PFTries on demand. Using synthetic traces with up to 16 million IP addresses, we show PFTries can indeed not only dramatically reduce their size if zooming is employed, but also keep their size small if some prefixes are zoomed in.
- *Traffic capturing strategy:* Besides relying on the streaming and zooming mechanisms to save the cost in capturing the footprints of DDoS traffic for a client under source-aware mode, the monitor at a PathFinder-participating AS may consider capturing only a sample of the traffic *if* doing so would still capture most, if not all, the source addresses or prefixes of the traffic. Furthermore, the monitor may lack modern hardware or software support (e.g., wiretapping is infeasible) or be constrained by the policy of the AS (e.g., only sampled mirroring is allowed), so it can only provide sampled traffic, anyway. We therefore investigate how long a monitor should observe the traffic and whether it can collect a sample of the traffic rather than all the traffic, while still learning most IP sources of the traffic.

9.2 Benefits of PathFinder for DDoS Defense

To quantify the benefits of PathFinder for DDoS defense, we first define the model of DDoS attack, then compare the results of a DDoS defense system with and without PathFinder, and finally study the partial deployment of PathFinder. Rather than using actual measurements over the Internet, this study is simulation based because our evaluations are at a large scale with more than 60,000 ASes, where the victim may be in any stub AS, DDoS bots may be distributed throughout all stub ASes, and PathFinder may be deployed at certain ASes, all of which are extremely hard to arrange on the real Internet. Simulation allows us to experiment with different configurations (e.g., different victim ASes) and collect results of many different scenarios (e.g., with or without PathFinder, with or without IP spoofing, different PathFinder deployment rates, etc.).

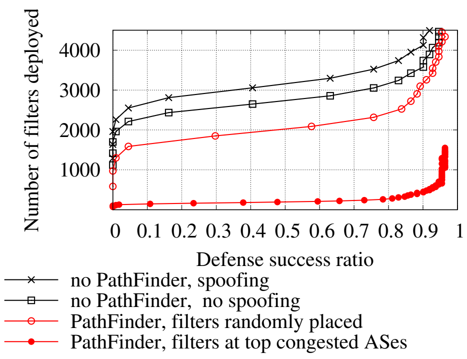


Fig. 7: **DDoS defense with and without PathFinder.**

9.2.1 *DDoS Attack Model*

The DDoS attack model includes the following components:

- **DDoS victim (target AS).** Every DDoS victim is located as a random tier-3 AS. We also call this AS the *target AS*. The victim can at most handle 10 Gbit/s incoming traffic. We use tier-3 ASes because DDoS attacks often target the services or applications hosted in such networks (e.g., educational networks, small data centers). Their limited link bandwidth makes them subject to most DDoS attacks reported [39]. On the other hand, tier-1 and tier-2 ASes, whose business is primarily to forward traffic and link capacity is generally very high [40], can hardly be threatened by DDoS attacks [41], [42].
- **DDoS bots.** The DDoS attack is based on high-volume traffic from DDoS bots where each bot has a fixed uplink bandwidth of 25 Mbit/s. Since a batch of bots will no longer be effective once filters are deployed to filter their traffic, an intelligent attacker 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. There are a total of 100,000 bots residing in stub ASes. (An AS may be either a stub AS, which is a source or sink of traffic and announces IP prefixes that it is in charge of, or a transit AS, which forwards traffic for stub ASes.) Except for the target AS, we assign every stub AS a number of DDoS bots that is proportional to the number of IP addresses announced by the AS.
- **Topology.** For every target AS, we derive an AS-level topology based on the relationships of ASes on the Internet according to CAIDA [43]. We use AS-level topologies rather than those at the router or PoP level because PathFinder is designed to operate at AS level (Sec. 3.1). The AS path from the target AS to every other AS follows the valley-free model [44]. The AS path from an AS to the target AS is either the reverse of the path from the target AS to the AS, or an asymmetric path with 30% probability.

9.2.2 *DDoS Defense Enhanced with PathFinder*

We simulate a DDoS defense as follows. During a DDoS attack, the victim employs a DDoS defense system that attempts to place a DDoS traffic filter at an AS on every path of DDoS traffic. To not have the DDoS traffic, as they converge, overwhelm links close to the target AS, that are on the first half of the AS paths of DDoS traffic. With PathFinder's help, the defense knows accurately the AS paths of DDoS traffic and the bandwidth consumption information of DDoS traffic along these AS paths. Here, we evaluate two AS selection methods for filter placement: 1) randomly select an upstream AS; 2) select an AS that belongs to top k ASes that carry most of the DDoS traffic.

Otherwise, without PathFinder, the defense has to guess the paths of DDoS traffic. It assumes AS paths are symmetric and regards the AS path from any DDoS bot to the target AS is always the reverse of the AS path from the target AS to the bot. From our model of the topology in Sec. 9.2.1, this assumption leads to a 30% miss in placing filters due to path asymmetry. Worse, if a DDoS bot spoofs its source address, the defense will even use a wrong AS path from the target AS to the bot to begin with, leading to a higher rate of misplaced filters; we assume in this case 50% filters are misplaced.

We evaluate PathFinder's benefits under the worst-case DDoS attack scenario. Specifically, the attacker employs an attack cycle that is no greater than the defense response time, which is the time for PathFinder to collect footprints of newly seen DDoS traffic and also for the DDoS defense system to place the filters against the traffic. Thus, before the DDoS defense places filters for the current bots attacking the victim, the attacker already switched to a new attack cycle.

Figure 7 compares the DDoS defense with or without the help of PathFinder. It shows the number of DDoS traffic filters needed for filtering out different DDoS traffic ratios, i.e., defense success ratio, under four different scenarios. To avoid potential selection bias of target ASes, we conducted 80 different experiments, each with a different target AS, and then averaged the results from all experiments. The DDoS defense system without PathFinder performs much worse than the two cases that use PathFinder. Without PathFinder, it takes at least 3,500 filters to subdue 90% of the attack traffic and even more (about 4,130 filters) when some DDoS bots employ IP spoofing. However, with PathFinder in place, if applying filters at top congested ASes (which requires AS path and bandwidth information from PathFinder), the victim can survive 90% of the attack traffic with only roughly 500 filters; in the case when the DDoS defense system uses only AS path information from PathFinder and places filters randomly at ASes, the system still uses a much smaller number of filters compared to the two defense cases without PathFinder.

9.2.3 *Partial Deployment of PathFinder*

We studied the partial deployment of PathFinder when not every AS runs PathFinder. If on the path from a DDoS bot to the victim there is at least one AS running PathFinder, we can say this bot is "covered" since the DDoS traffic footprints from this bot will be reported by PathFinder; otherwise, the bot is not "covered". Therefore, to evaluate the coverage provided by PathFinder when it is only partially deployed, we can measure the percentage of DDoS bots that are "covered," i.e., traffic footprints coverage by PathFinder.

Not every AS on the Internet is equal. It is difficult for tier-1 ASes, which usually could span an entire country or even beyond, to deploy PathFinder throughout its entire network due to its sheer size and high cost. Tier-3 ASes are small and much easier to deploy and run PathFinder, but given the most ASes on the Internet are tier-3 ASes at

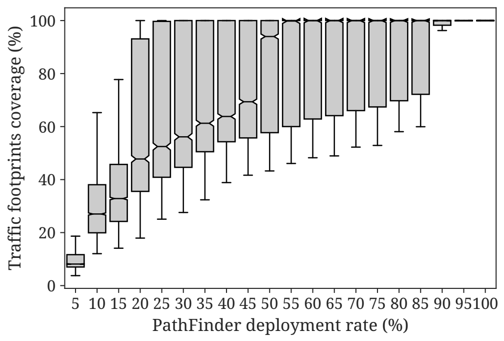
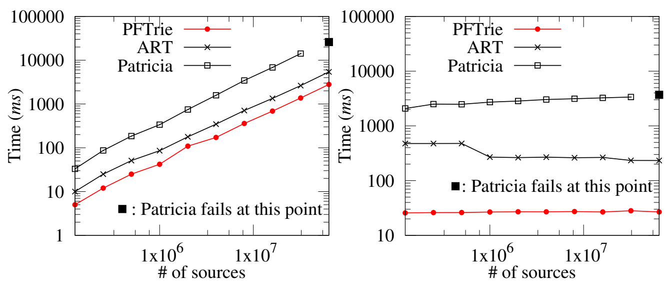


Fig. 8: **Partial deployment of PathFinder at tier-2 ASes.**



(a) put_a_new scenario. (b) put_an_old scenario.

Fig. 9: **PFTrie speed.**

the edge of the Internet, and they can only report footprints of traffic originated from their network, PathFinder has to be deployed at a substantial amount of tier-3 ASes to have a clear impact. The best trade-off is to look at deploying PathFinder only at tier-2 ASes, which are not only easier to deploy and run PathFinder but can also report footprints of DDoS bots located at a multitude of ASes.

Therefore, we measured the traffic footprints coverage by PathFinder using the DDoS attack model described in Sec. 9.2.1. We experimented $5,000$ different topologies, each with a different target AS and a range of PathFinder deployment rates among tier-2 ASes. Figure 8 shows that when 55% or more tier-2 ASes run PathFinder, virtually 100% of the DDoS bots (i.e., their traffic footprints) will be covered for half of the experimented topologies. Note that given about 25 tier-1 ASes, 10,000 tier-2 ASes, and 58,000 tier-3 ASes on the Internet, 55% tier-2 ASes is about 8.1% all ASes [43]. Even when it is less than 55%, PathFinder can be very effective. For example, even when only 20% of tier-2 ASes (i.e., 2.9% of all ASes) run PathFinder, half of the evaluated topologies will have at least 48% DDoS bots covered, and a quarter of evaluated topologies will have at least 94% DDoS bots covered.

9.3 Speed and Overhead of PFTrie

9.3.1 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 125 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. We choose not to use publicly available DDoS traces as their attack volume/sources are too small of a scale to benchmark the scalability of PFTrie.

9.3.2 Speed of PFTrie

We compare PFTrie's performance against Adaptive Radix Tree (ART) [45] and the well-known Generalized Prefix Tree [46], also called Patricia Trie, under the two scenarios, **put_a_new** and **put_an_old**, as defined in Sec. 9.1. We use all the 50 synthetic traces that contain different orders of magnitude numbers of IP addresses to evaluate the three different data structures under stress.

Fig. 9a shows the comparison results under the put_a_new scenario. It reports the time for ART, Patricia Trie, and PFTrie to store all the IP addresses in a trace with 25% address locality as new addresses. Our results with traces of different locality are similar. For every synthetic trace size ranging from 125,000 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 and about 6,600ms for Patricia 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. Note that Patricia fails when presented with this many addresses.

Fig. 9b shows results under the put_an_old scenario for performing 15 million *put* processes that stores an IP address *already* stored. These IP addresses are from the same trace as the one used in Fig. 9a. Here, as there are no new nodes to be inserted into a PFTrie, the time needed by a 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 less than that of 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. 5.2.2) and the avoidance of duplicate traversals (Sec. 5.2.3).

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 [47], [48], the put_an_old scenario will happen 369 times more than the put_a_new scenario.

9.3.3 Overhead of PFTrie

We are particularly interested in the memory cost of PFTrie when there are millions of IP source addresses. Using synthetic traces that include a huge number of IP source addresses, we evaluated the memory usage for each number of sources under five different profiles, as shown in Fig. 10. We can see the logarithm of the memory cost is basically a linear function of the logarithm of the number of sources,

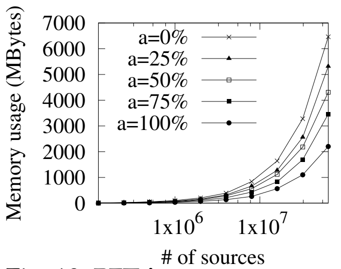


Fig. 10: **PFTrie memory overhead across five different source profiles.** Each profile has a different percentage of aggregatable addresses (a).

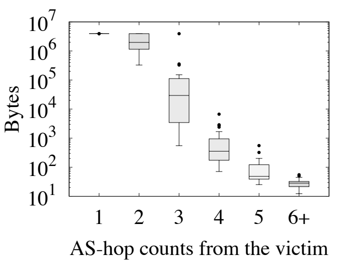


Fig. 11: **The boxplot of network overhead in transmitting PFTries of 1 million source addresses.**

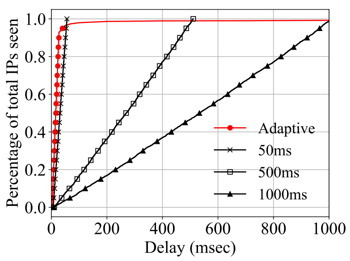


Fig. 12: **Delay CDF with fixed and adaptive streaming (DDoS trace: Booter 5).**

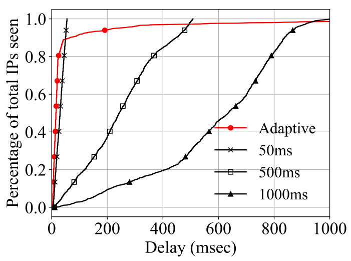


Fig. 13: **Delay CDF with fixed and adaptive streaming (DDoS trace: Booter 9).**

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, due to the optimization via bottom-up aggregation of PFTrie leaf nodes (Sec. 5.2.1). 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 with 25 different AS-level Internet topologies. We generated these AS-level topologies using the same method described in Sec. 9.2.1, with each topology using a different random tier-3 AS as a target AS. For each topology, we further populate it with one million IP addresses across all the ASes, where the number of IP addresses assigned to each AS is proportional to its IP address space size. Fig. 11 shows the network transmission overhead for an AS to transmit its PFTrie 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 one million IP addresses, thus incurring the largest overhead, which is only about 3.9 MB.

9.4 Benefits of PathFinder Streaming

We ran simulation studies over an inferred AS-level topology to measure the delay and overhead with PathFinder streaming, including how effective and efficient the adaptive streaming design is. Below we describe our simulation model and results.

9.4.1 Streaming Simulation Model

The streaming simulation model includes all PathFinder components (monitor, proxy, client) and an inferred AS-level topology generated using the method described in Sec. 9.2.1. While the overall Internet connection speed has become faster over the years, we use conservative numbers in our simulation. Specifically, for each link on the topology we assume it has a random delay between $1ms$ and $10ms$ and a fixed bandwidth of 10 Mbps. (According to the 2019 report from Opensignal [49], the connection speed over a *mobile network* is 10+ Mbps in 67 out of 87 surveyed countries and 5+ Mbps in more than half of the remaining 20 countries.) Moreover, we use two Booter DDoS traces ([47] [48]) to drive the streaming simulations. We map the IP addresses in these traces to ASes (or PathFinder monitors) in our simulation.

9.4.2 Delay with Streaming

We now evaluate the delay with both fixed streaming and adaptive streaming, i.e., the time it takes for a client to learn the IP address of a DDoS bot after the first DDoS packet departs from the bot toward a victim.

Figures 12 and 13 show the delay distribution over Booter 5 and Booter 9 DDoS traces, respectively. In both figures, adaptive streaming outperforms fixed streaming at three different fixed intervals. For both attack traces, adaptive streaming delivers more than 90% of the DDoS IP addresses in less than 50 milliseconds. The adaptive streaming tends to have longer delays for the later half of the DDoS traces as the traces contain many fewer new sources in their second half (the attacker has a limited number of bots for each attack); this behavior is consistent with the style of adaptive streaming as it is not aggressive in learning a trickle of new DDoS IP addresses.

9.4.3 Network Bandwidth Overhead with Streaming

We further evaluate the network bandwidth overhead over an extended period of time with both fixed streaming and adaptive streaming. Figures 14 and 15 show the cumulative network bandwidth overhead from adaptive or fixed streaming at each time point using Booter 5 and Booter 9 traces, respectively. Clearly, although the fixed streaming with a short interval ($50ms$) has a small delay (Figures 12 and 13), it incurs a network bandwidth overhead that is approximately 4 and 7 times, respectively, more than the adaptive streaming. The fixed streaming with a medium and long intervals ($500ms$ and $1000ms$) incur less bandwidth overhead than the fixed streaming with a short interval ($50ms$), but such a bandwidth overhead is at the cost of the delay in obtaining traffic footprints as shown above and is still more than that of the adaptive streaming.

9.5 Benefits of PathFinder Zooming

We now show how much memory overhead that the zooming mechanism can help reduce. (We have shown the memory cost of PFTries when zooming is not in effect in Sec. 9.3.3.) We use the synthetic traces whose source address locality is 0% (worst case).

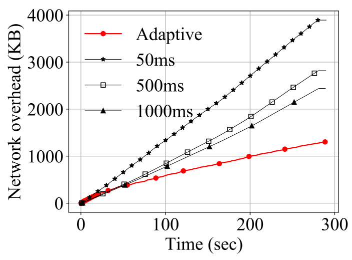


Fig. 14: **Network cumulative bandwidth overhead with fixed and adaptive streaming (DDoS trace: Booter 5).**

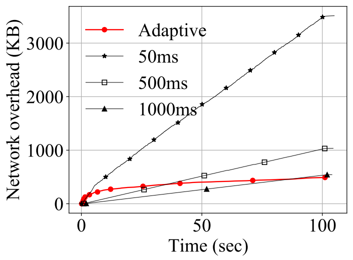


Fig. 15: **Network cumulative bandwidth overhead with fixed and adaptive streaming (DDoS trace: Booter 9).**

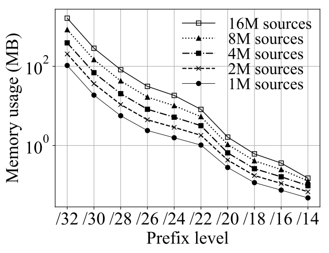


Fig. 16: **Memory overhead with zooming (without zooming in on any prefix).**

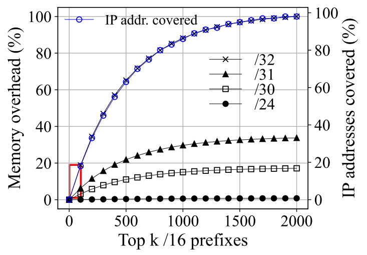


Fig. 17: **Memory overhead (as a percentage of the baseline) and IP addresses covered, both with top k /16 prefixes zoomed in.**

Figure 16 shows the memory overhead for storing PFTries when the zooming technique is used with varying prefix levels and different numbers of sources but *without* zooming in on any prefix. In all cases, we can see the memory overhead decreases exponentially as we reduce the monitored prefix level. For example, it costs 1.6 GB of memory space to save 16 million IPv4 addresses (/32) while it costs only 0.36 MB of memory space to save the same traces at /16 prefix level. We also notice that as the prefix level decreases the slope of each line also varies; this is due to the different degree of bottom-up aggregation optimization of PFTries at each prefix level and a bigger slope means a higher rate of aggregation and vice versa.

We further measured the memory overhead when a PathFinder client requests to zoom in on certain prefixes that contribute more traffic with more DDoS bots than other prefixes. Such disproportional distribution can be seen in, for example, the Mirai DDoS attack [50]. We used a synthetic trace with totally 16 million IP addresses and a 0% source locality (i.e., a=0%). With totally 2^{16} (i.e., 65536) /16 IP prefixes, in this trace the number of IP addresses allocated to each /16 prefix follows a power law distribution, with a small percentage of /16 prefixes crowded with a large portion of IP addresses; for example, top 100, 200, 400, 800, and 1600 crowded prefixes respectively cover 19%, 33%, 56%, 80%, and 96% of all 16 million IP addresses.

Figure 17 shows the results. Define the baseline as the memory usage of the PFTrie that stores every IP address from this trace (i.e., zooming is *not* in place), which is 1.6 GB as seen in Figure 16. Figure 17 shows the memory overhead as a percentage of the baseline when the client, which initially monitors /16 prefixes, decides to zoom in on top-*k* crowded /16 prefixes to all their /24, /30, /31, and /32 sub-prefixes. We can see that when the client zooms in on a small number of prefixes to any prefix level, the memory overhead is quite small; for example, if zooming in on top 100 prefixes, which is only 0.15% of all prefixes but with 19% of the 16 million IP addresses, the memory overhead is 18.8% in the worse case when reaching all /32 prefixes (as shown in the red box in Figure 17). Also, when the client zooms in on *all* prefixes to /31, /30, or /24 prefix levels, the memory overhead will also be fairly small at 33.8%, 17.1%, and 0.79%, respectively. Overall, the flexibility offered by zooming allows the client to choose which prefixes and to which prefix levels to zoom in; except for rare cases (such as zooming in on all prefixes to every single IP address level), the memory overhead saving with zooming is significant, often at multiple orders of magnitude.

9.6 Traffic Capturing Strategy Analysis

Rather than monitoring a DDoS attack over its entire life cycle and capturing every single packet toward the DDoS victim, a monitor may only observe the DDoS traffic for a time window and also only sample the traffic. However, doing so could cause certain source addresses in the traffic not to be captured, reducing the traffic capturing accuracy to below 100%. The question is, is it possible to maintain a high accuracy with a certain size of monitoring window and a certain traffic sampling rate?

We measured the accuracy of capturing the source addresses from real DDoS traffic traces (Table 1). By varying the size of the monitoring window from 0.2 to 3.0 seconds and the traffic sampling rate from 0.5% to 20%, we obtained the results depicted in Fig. 18. We can see that while within 3 seconds the sampling rates up to 2.5% can never reach an accuracy of more than 90%, starting from a sampling rate of 3%, the accuracy is already as good as 90% when monitoring the traffic for 3 seconds. On the other hand, if the monitoring window needs to be as short as possible, the sampling rate has to be higher; e.g., if only monitoring traffic for 0.4 seconds, the accuracy can be as high as 85% with 18% sampling rate.

As the sampling rate and the window size can both affect the accuracy, we further compare the accuracy with a faster sampling rate in a shorter window and the accuracy with a lower sampling rate but a longer window. Fig. 19 shows that when the product of the window size and the sampling rate is the same, a higher sampling rate in a shorter window consistently performs slightly better than its counterpart.

TABLE 1: **The Booter DDoS traffic traces (9 datasets) [47], [48]. (The last column shows the ratio of the new source addresses versus the duplicate addresses in the trace.)**

Dataset	# of attack sources	DDoS bandwidth (Mbps)	new/duplicate sources
Booter 1	4486	700	1/72
Booter 2	78	250	1/230
Booter 3	54	330	1/370
Booter 4	2970	1,190	1/14
Booter 5	8281	6	1/2.5
Booter 6	7379	150	1/5.5
Booter 7	6075	320	1/3.6
Booter 8	281	990	1/157
Booter 9	3779	5,480	1/57

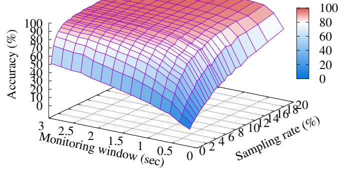


Fig. 18: **Accuracy of capturing the source addresses in the Booter 9 DDoS traces.**

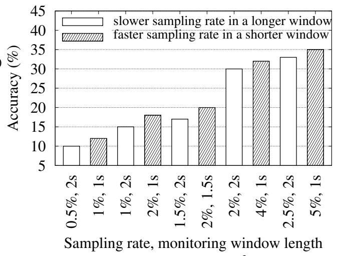


Fig. 19: **Accuracy of capturing sources with a faster sampling rate in a shorter window.**

10 Discussions

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

10.1 System Scalability

Since the PathFinder system is stateful, it must consider scalability issues related to, for example, the memory and input/output resources. For example, an adversary can employ IP spoofing, generate traffic from arbitrary IP addresses, and overwhelm the system if it operates in source-aware mode. In fact, PathFinder has embraced a suite of different mechanisms to make itself scalable. We have designed three different methods to optimize a PFTrie (Sec. 5.2). Also, a PathFinder client can always utilize the zooming mechanism to consume least possible memory and network resources and only zoom in on prefixes of interest, such as those with a large number of IP addresses or voluminous traffic. Moreover, a monitor or a proxy can always use out-of-sync streaming to proactively free more buffer space for more footprints later. Finally, in case out-of-sync streaming cannot work either due to network congestion, the monitor (or the proxy) could prune its PFTrie bottom up level by level as needed; for instance, it could prune all leaves at level 32 to shrink the PFTrie to have 31 levels, or even less by repeating the pruning of leaves at the current level, where the traffic footprint information carried by every leaf node is transferred to its parent. The pruning procedure is similar to the bottom-up aggregation of leaf nodes, except that here a leaf is removed even if it is the only child of its parent. A PFTrie with x levels after pruning also looks the same as a PFTrie with zooming at $/x$ IP prefixes in place, except that the former is invoked by a monitor (or a proxy) and the latter is requested by a client. As a result, PathFinder will be able to collect and deliver traffic footprints from an arbitrary number of IP addresses, including when an attacker overwhelms PathFinder with a vast amount of spoofed traffic as mentioned in the example above.

10.2 System Security

We assume that the PathFinder system employs state-of-the-art defense mechanisms to protect itself against any security attacks. Using standard security mechanisms such as mutual TLS that enable two parties to authenticate each other and establish a secure channel [51], every PathFinder client can establish a secure channel with its PathFinder proxy, and similarly, every proxy can establish a secure channel with every PathFinder monitor. These channels can then protect the integrity and confidentiality of both requests for DDoS traffic footprints (which are sent from a client to its proxy and then to PathFinder monitors) and DDoS traffic footprints themselves (which are delivered from monitors to proxies and then to clients). Note that PathFinder employs a trust model in which both proxies and monitors are trustworthy. As a PathFinder proxy is to provide service to multiple clients, it is closely monitored and well-protected. And it is safe to assume that PathFinder monitors are not ill-intentioned to provide false fingerprints; otherwise, the AS hosting a malicious monitor is likely to be malicious or compromised as a whole, in which case the attacker is not going to waste his time generating false PathFinder fingerprints as they are not nearly the worst trouble the attacker can cause.

10.3 System Deployment

PathFinder addresses a critical missing gap of many DDoS defense systems that often lack sufficient knowledge of DDoS traffic footprints, including attack paths, bandwidth consumption along each path, and the source addresses or prefixes of DDoS traffic. Indeed, while the network community is moving towards a fine-grained DDoS defense style (e.g., BGP FlowSpec [52]), the need of fine-grained DDoS traffic footprint information becomes more urgent. Moreover, PathFinder is deployment friendly. It employs an architecture that is easy to implement and deploy on today's Internet; PathFinder monitors and proxies, in particular, can be easily set up by their AS. The streaming and zooming mechanisms of PathFinder further make it affordable to deploy PathFinder. Also, the PFTrie data structure includes multiple design features to allow a monitor to log traffic at a line rate. Finally, we emphasize PathFinder is beneficial to DDoS defense even when its deployment rate is relatively low (Sec. 9.2.3).

11 Conclusions

While DDoS attacks have become more frequent and can cause severe damage to services on the Internet, defense against such attacks has often suffered from the lack of relevant knowledge of the DDoS traffic. However, it is 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 carries 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. It incorporates a PFTrie data structure that introduces multiple design features to log traffic at a line rate, as well as streaming and zooming mechanisms that facilitate the storage and transmission of DDoS footprints more efficiently.

In addition to building a pilot version of the PathFinder system, we have performed substantial performance experiments to characterize its effectiveness, speed, and costs. These experiments show that PathFinder can deliver accurate information about DDoS flows to a site under attack at reasonable costs both to the systems gathering the data and the defense system that consumes it. Our evaluation shows that the PathFinder system is viable: Even with a low-end desktop machine it only takes about 2.93 seconds to store 64 million unique IP addresses, or only 115 milliseconds in total to check if they are previously stored, while the memory and network overhead is manageable and scalable, especially with the help of the streaming and zooming mechanisms.

Acknowledgments

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.

References

[1] Akamai. (2020) 2020 State of the Internet Security: Financial Services – Hostile Takeover Attempts. https://www.akamai.com/us/en/resources/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] M. S. Kang, S. B. Lee, and V. D. Gligor, "The Crossfire Attack," in *IEEE Symposium on Security and Privacy*, 2013.
[4] Y.-M. Ke, C.-W. Chen, H.-C. Hsiao, A. Perrig, and V. Sekar, "CICADAS: Congesting the Internet with Coordinated and Decentralized Pulsating Attacks," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, 2016.
[5] L. Shi, D. Sisodia, M. Zhang, J. Li, A. Dainotti, and P. Reiher, "The Catch-22 Attack (Poster)," in *Annual Computer Security Applications Conference*, 2019.
[6] A. Wang, W. Chang, S. Chen, and A. Mohaisen, "Delving into internet ddos attacks by botnets: Characterization and analysis," *IEEE/ACM Transactions on Networking*, vol. 26, no. 6, 2018.
[7] A. Wang, W. Chang, S. Chen, and A. Mohaisen, "A data-driven study of ddos attacks and their dynamics," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 3, 2020.
[8] L. Shi, M. Zhang, J. Li, and P. Reiher, "Pathfinder: Capturing ddos traffic footprints on the internet," in *2018 IFIP Networking Conference (IFIP Networking) and Workshops*. IEEE, 2018, pp. 1–9.
[9] M. Zhang, L. Shi, D. Sisodia, J. Li, and P. Reiher, "On multipoint, in-network filtering of distributed denial-of-service traffic," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2019.
[10] M. Zhang, G. Li, S. Wang, C. Liu, A. Chen, H. Hu, G. Gu, Q. Li, M. Xu, and J. Wu, "Poseidon: Mitigating volumetric ddos attacks with programmable switches," in *Proceedings of NDSS*, 2020.
[11] D. Gong, M. Tran, S. Shinde, H. Jin, V. Sekar, P. Saxena, and M. S. Kang, "Practical verifiable in-network filtering for ddos defense," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, 2019.
[12] C. Dietzel, M. Wichtlhuber, G. Smaragdakis, and A. Feldmann, "Stellar: Network Attack Mitigation Using Advanced Blackholing," in *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT'18, 2018.
[13] S. Ramanathan, J. Mirkovic, M. Yu, and Y. Zhang, "Senss against volumetric ddos attacks," in *Proceedings of the 34th Annual Computer Security Applications Conference*, ser. ACSAC '18, 2018.
[14] J. Li, S. Berg, M. Zhang, P. Reiher, and T. Wei, "Drawbridge: software-defined DDoS-resistant traffic engineering," in *ACM SIGCOMM Computer Communication Review*, vol. 44, 2014.
[15] X. Liu, X. Yang, and Y. Lu, "To filter or to authorize: Network-layer DoS defense against multimillion-node botnets," *ACM SIGCOMM Computer Communication Review*, 2008.
[16] X. Yang, D. Wetherall, and T. Anderson, "Tva: A dos-limiting network architecture," *IEEE/ACM Transactions on Networking*, vol. 16, pp. 1267–1280, 2008.
[17] G. C. Moura, R. d. O. Schmidt, J. Heidemann, W. B. de Vries, M. Muller, L. Wei, and C. Hesselman, "Anycast vs. ddos: Evaluating the november 2015 root dns event," in *Proceedings of the 2016 Internet Measurement Conference*, ser. IMC '16, 2016.
[18] G. C. M. Moura, C. Hesselman, G. Schaapman, N. Boerman, and O. de Weerdt, "Into the ddos maelstrom: a longitudinal study of a scrubbing service," in *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*, 2020.
[19] H. Burch and B. Cheswick, "Tracing Anonymous Packets to Their Approximate Source," in *USENIX Large Installation System Administration Conference*, 2000.
[20] K. Singh, P. Singh, and K. Kumar, "A Systematic Review of IP Traceback Schemes for Denial of Service Attacks," *Computers & Security*, vol. 56, 2016.
[21] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Practical Network Support for IP Traceback," in *ACM SIGCOMM*, 2000.
[22] 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.
[23] 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.
[24] V. Aghaei-Foroushani and A. N. Zincir-Heywood, "Autonomous System Based Flow Marking Scheme for IP-Traceback," in *IEEE/IFIP Network Operations and Management Symposium*, 2016.
[25] A. Y. Nur and M. E. Tozal, "Record Route IP Traceback: Combating DoS Attacks and the Variants," *Computers & Security*, vol. 72, 2018.
[26] V. Murugesan, M. S. Selvaraj, and M.-H. Yang, "HPSIPT: A High-Precision Single-Packet IP Traceback Scheme," *Computer Networks*, vol. 143, 2018.
[27] H. Patel and D. C. Jinwala, "LPM: A Lightweight Authenticated Packet Marking Approach for IP Traceback," *Computer Networks*, vol. 140, 2018.
[28] 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.
[29] 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.
[30] E. Hilgenstieler, E. P. Duarte, G. Mansfield-Keeni, and N. Shiratori, "Extensions to the Source Path Isolation Engine for Precise and Efficient Log-Based IP Traceback," *Computers & Security*, vol. 29, 2010.
[31] Z. M. Mao, L. Qiu, J. Wang, and Y. Zhang, "On AS-Level Path Inference," in *ACM SIGMETRICS*, 2005.
[32] 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.
[33] Cisco. (2013) 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.
[34] FRRouting. (2020) FRRouting User Guide. http://docs.frrouting.org/en/latest/bgp.html#displaying-bgp-information.
[35] Cisco. (2019) Catalyst Switched Port Analyzer (SPAN) Configuration Example. http://www.cisco.com/c/en/us/support/docs/switches/catalyst-6500-series-switches/10570-41.html.

[36] Juniper. (2018) 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.

[37] 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.

[38] Juniper. (2019) 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.

[39] V. Ganti and O. Yoachimik. (2021) Network-layer DDoS attack trends for Q4 2020 . https://blog.cloudflare.com/network-layer-ddos-attack-trends-for-q4-2020/.

[40] PeeringDB. (2021) Cogent Communications, Inc. https://www.peeringdb.com/net/267.

[41] C. Cimpanu. (2020) AWS said it mitigated a 2.3 Tbps DDoS attack, the largest ever. https://www.zdnet.com/article/aws-said-it-mitigated-a-2-3-tbps-ddos-attack-the-largest-ever.

[42] KrebsOnSecurity. (2016) KrebsOnSecurity Hit With Record DDoS. https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos.

[43] CAIDA. The CAIDA AS Relationships Dataset, 2020-12-01. http://www.caida.org/data/active/as-relationships/.

[44] R. Mahajan, D. Wetherall, and T. Anderson, "Understanding BGP Misconfiguration," in *ACM SIGCOMM Computer Communication Review*, 2002.

[45] V. Leis, A. Kemper, and T. Neumann, "The Adaptive Radix Tree: ARTful Indexing for Main-Memory Databases," in *IEEE 29th International Conference on Data Engineering*, 2013.

[46] D. R. Morrison, "PATRICIA—Practical Algorithm To Retrieve Information Coded in Alphanumeric," *Journal of the ACM*, vol. 15, 1968.

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

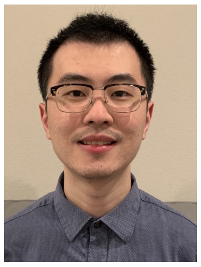
[48] J. Santanna, R. van Rijswijk-Deij, R. Hofstede, A. Sperotto, M. Wierbosch, L. Zambenedetti Granville, and A. Pras, "Booters - An Analysis of DDoS-as-a-Service Attacks," in *IFIP/IEEE International Symposium on Integrated Network Management*, 2015.

[49] Opensignal. (2019) The State of Mobile Network Experience. https://www.opensignal.com/sites/opensignal-com/files/data/reports/global/data-2019-05/the_state_of_mobile_experience_may_2019_0.pdf.

[50] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis *et al.*, "Understanding the Mirai Botnet," in *USENIX Security Symposium*, 2017.

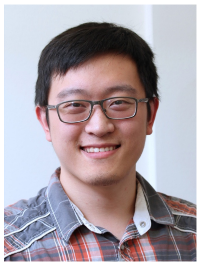
[51] B. Campbell, J. Bradley, N. Sakimura, and T. Lodderstedt, "OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens," RFC 8705, Feb. 2020. [Online]. Available: https://rfc-editor.org/rfc/rfc8705.txt

[52] P. Marques, N. Sheth, R. Raszuk, B. Greene, J. Mauch, and D. McPherson, "Dissemination of Flow Specification Rules," https://tools.ietf.org/html/rfc5575, 2009.



Lumin Shi is a Ph.D. candidate in the Department of Computer and Information Science at the University of Oregon (UO). He also serves as an information security analyst at UO. He received his B.S. degree in computer science from the University of Alabama. He was a visiting scholar at CAIDA, UC San Diego, in 2019, where he worked on the OpenBMP project. His main research interests include network security, network measurement, and high-fidelity network experimentation.

Jun Li is a Professor in the Department of Computer and Information Science and founding director of the Center for Cyber Security and Privacy at the University of Oregon. He received his Ph.D. from UCLA in 2002 (with Outstanding Doctor of Philosophy honor), M.E. from Chinese Academy of Sciences in 1995 (with Presidential Scholarship), and B.S. from Peking University in 1992, all in computer science. His research is focused on networking, distributed systems, and network security, with about 100 peer-reviewed publications. He has served on US National Science Foundation research panels and more than 70 international technical program committees, including chairing six of them. He is a senior member of ACM and IEEE and an NSF CAREER awardee in 2007.



Mingwei Zhang received his master and Ph.D. degrees from the University of Oregon in 2016 and 2019, both in computer science. He is currently an Internet data scientist with Center for Applied Internet Data Analysis (CAIDA) at UC San Diego, where he works on various projects on Internet infrastructure monitoring, Internet routing security, and open source data toolkits for the public and research communities.

Peter Reiher Dr. Peter Reiher is an Adjunct Professor in the Computer Science Department at UCLA. His research interests include cybersecurity (particularly Internet security issues, such as distributed denial of service attacks), computer networks, ubiquitous computing, security of autonomous vehicles and vehicular networks, security of wireless medical devices, and file systems. He received his Ph.D. from UCLA in 1987.