PathFinder: Collecting Traffic Footprint at Autonomous System Level

Lumin Shi
luminshi@cs.uoregon.edu
Computer and Information Science
University of Oregon

***Abstract*—Distributed Denial-of-Service (DDoS) attacks are simple to launch yet hard to defend against. For many DDoS defense strategies, quickly finding the forwarding paths taken by the attack packets is a critical step for attack mitigation. Techniques such as IP traceback and autonomous system (AS) path inference are used in DDoS defense systems to find the path information for packets in question. Though IP traceback and path inference are well-studied topics, they are not designed toward finding the whole traffic footprint, which contains all packet sources and forwarding paths.**

We propose PathFinder, a log-based traffic footprint collection scheme that allows a recipient to reconstruct the AS-level forwarding path(s) that packets would take. PathFinder has multiple advantages over the traffic footprint methods in the existing DDoS defense solutions: it is scalable and friendly to incremental deployment; it does not impose network overhead when there are no user requests and little when there are; and more importantly, PathFinder does not require hardware or software changes to the existing equipment. Compared to previous solutions, these advantages increase the deployment feasibility for network providers to deploy PathFinder, with fewer drawbacks.

***Index Terms*—Traffic footprint collection, DDoS defense**

I. Introduction

Today's Internet is vulnerable to distributed denial-of-service (DDoS) attacks. During a DDoS attack, an attacker controls many compromised machines to flood the victim with unwanted traffic in order to exhaust the network or computation resources of 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 the overwhelming traffic volume and take down most web services. For example, a recent large-scale DDoS attack on Dyn [2] disabled its DNS service, and crippled many major web services that relied on it such as Twitter, Netflix, PayPal, and over fifty others for hours. Similar incidents [3][4] have well-demonstrated the devastating destructive power of DDoS attacks over the Internet.

An attack with huge traffic volume can bring down the victim's network, and it could also congest the transit networks sit between the victim and bots, which will further cause damages to other edge networks. DDoS attacks with such level of traffic volume pose huge challenge to the traditional single-entity DDoS defense systems, which usually rely on single organization to filter traffic. Distributed DDoS defense systems (such as in [5][6][7][8][9][10]), on the other hand, orchestrate collaboration among multiple autonomous systems (ASes) in order to defend against large-scale DDoS attacks. These systems try to filter attack traffic closer to the traffic sources so as to reduce the load of the victim network and the transit networks, thus being able to handle larger-scale DDoS attacks.

A typical distributed DDoS defense system includes two critical processes:

- finding the corresponding routers/ASes in the system that are carrying the attack traffic;
- installing the traffic filtering rules at the routers/ASes as close to the traffic sources as possible, if the downstream links are inundated.

To filter DDoS attack flows, a distributed defense system first needs to determine the locations for deployment, i.e., discovering the responsible ASes that carry DDoS traffic to the victim. However, finding responsible ASes is not trivial for the following reasons: First, the routing paths on the Internet are asymmetric: for any pair of end hosts, it is likely that the traffic forwarding paths will be different for each direction [11][12]; a distributed DDoS defense system deploys traffic filtering rules based on the forwarding path from the victim to the attack source is inaccurate. To make things worse, some attackers spoofs source addresses to avoid being filtered by defense systems, further complicating the task of locating responsible ASes.

To find the forwarding path information for each DDoS attack packet, a collaborative defense system can choose one of the following approaches:

- using IP traceback approach to figure out which routers are on the path to the victim, that are carrying a certain attack flow in question;
- inferring victim's inbound traffic topology (AS-level path inference).

However, none of these approaches appear to be both accurate and deployable. The first approach, i.e., IP traceback, allows a user to find the source network that generated the packet in question hop-by-hop. We believe such a system should include the following properties for practical deployment:

- The system should require a minimum amount of modifications to the hardware and software of routers/switches.
- The system should scale regardless of the deployment rate.

- The system should be able to handle large-scale DDoS attacks.
- The system should not downgrade the network throughput.

To the best of our knowledge, there is no IP traceback system that meets all the aspects listed above, and we believe these are the major road blocks that prevent traceback systems from being deployed. The second approach, AS-level path inference, does not introduce deployment difficulties. However, the inference results' accuracy varies greatly and it contains no realtime traffic information to ensure whether a path is indeed carrying certain traffic.

In this paper, we introduce PathFinder, a traffic footprint collection system. PathFinder allows traffic destination networks to retrieve latest traffic sources toward them, and the forwarding path(s) of each traffic source. A distributed DDoS defense system can use PathFinder to generate a list of ASes that are carrying DDoS traffic, including bandwidth consumption information, and topology information collected by PathFinder, a DDoS defense system can utilize the information to install traffic filtering rules at the right ASes. Network administrators can use PathFinder to debug the network problems such as reachability or to find the traffic forwarding routes.

The highlights of the system is as follows:

- The system is a practical, deployable solution for current ASes in that no hardware or software changes of the routers/switches are required.
- The system is resilient against large-scale DDoS attacks with different patterns.
- The system allows the user to construct AS-level forwarding path locally without making extra connections to AS participants.

In the remaining of this paper, we organize sections as follows:

- Section II presents the usage models of PathFinder.
- Section III to VI introduce the basics of the system, as well as a detailed view of each system component.
- Section VII presents our evaluation results of PathFinder.
- Section IX presents the related work, and we also compare PathFinder with others.
- Section VIII and X discuss the potential issues of the work and the conclusion.

II. Usage Models

PathFinder provides three types of information to its users:

- **Topology**: AS-level forwarding path(s) observed from AS monitors to the user.
- **Traffic footprint**: source or source-agnostic traffic footprint AS monitors have seen toward the user.
- **Bandwidth information**: average traffic bandwidth consumption or PPS each AS carries for the user.

PathFinder users can utilize one or more types of the information above for various of network applications. We define a PathFinder Log (PFLog) as the message that includes one or more types of the information defined above. In this section, we introduce two practical applications in PathFinder system: DDoS defense systems and network debugging.

A. PathFinder for Distributed DDoS Defense Systems

Depending on the type of a DDoS attack, a collaborative DDoS defense system would require different information of the attack to make effective traffic mitigation strategies. Figure 1 shows a generic example of a collaborative DDoS defense system. When an attack happens, the attack victim will generate a set of traffic filters, and provide them to the defense system. The DDoS defense system then processes the filters from the victim and make decisions on where to install these filtering rules. A defense system can locate the traffic aggregation nodes, AS 2 and 3 in this case, and install traffic filters at these two ASes in order to drop the attack traffic. Before this system can make precise decisions, it needs to have a way to gather latest information about the global network. How does the defense system know that traffic aggregation nodes are AS 2 and 3 in the first place?

In this section, we list possible parameters that a DDoS defense system may need, and use two DDoS defense scenarios as examples to show readers the importance of these parameters.

1) Parameters in DDoS Defense: As we mentioned above, PathFinder provides topology, traffic footprint and bandwidth information to the users. Each PathFinder user (or DDoS defense systems in this context) may use the information differently, and here is a list of options that PathFinder provides:

- **AS participants**: users choose which ASes in PathFinder to gather traffic footprint from;
- **update frequency**: how frequent defense systems need to pull information from PathFinder;
- **source-based footprint**: do defense systems need traffic footprint that contains source IP addresses or prefixes? If enabled, each AS monitor needs to conclude what source IP addresses/prefixes it has seen, otherwise, each AS monitor simply state whether it has seen traffic toward the victim or not.

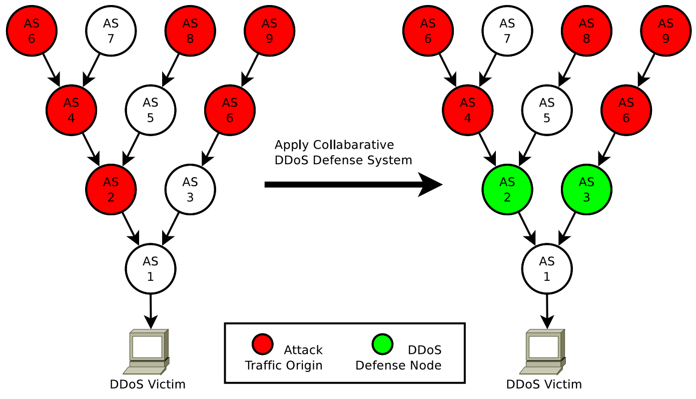

Fig. 1: DDoS attack and distributed DDoS defense system example

- **footprint granularity**: if defense systems require source-based footprint, they can also choose what is the granularity of the source information, e.g., a defense system can choose to gather footprint represented by /20 prefixes.
- **bandwidth information**: do defense systems need bandwidth related information? E.g., bandwidth per flow/AS, packets per second (PPS) per flow/AS.

2) Source-based Defense: In the most recent large-scale DDoS attack on Dyn, Inc., the company faced an excessive amount of DDoS traffic from over 100,000 compromised devices [13]. In this attack, the attacking devices sent bogus traffic to Dyn servers without spoofing their source addresses. Finding the footprints of these source addresses and filter traffic that matches these addresses is a good defense strategy.

In a source-based defense, a DDoS defense system may ask PathFinder to gather the source-based footprint at certain footprint granularity. For example, an attack victim can analyze the network logs, and it finds out the IP addresses of attack sources can be described using multiple /24 subnets, due to the high locality of the source IP addresses. The DDoS defense system can use this knowledge, and ask PathFinder system to record traffic footprint in /24 prefix form.

3) Source-agnostic Defense: In the above scenario, the DDoS traffic does not utilize IP spoofing, a defense system can filter attack traffic by IP source addresses. However, today's DDoS attacks utilizes IP spoofing as well, and DDoS defense systems should not rely on source-based defense in such scenario.

In a DDoS attack practices IP spoofing, the attacker commands the bots to spoof the source address field of each packet to be victim's address, and send these packets to the amplifiers such as public DNS or NTP servers. These amplifiers would then send the replies back to the source address which points to victim's network, and eventually collapse the inbound link of the victim's network.

We list the defense to such attack as source-agnostic defense because a DDoS defense system can block the traffic by the protocols those amplifiers are using; if the amplifiers are DNS servers, the defense system can install filters at different ASes to block DNS traffic toward the victim. Topology or bandwidth related information from PathFinder would be more cost effective: the defense system simply need to install several filters at the traffic aggregation ASes, and PathFinder can provide such information with a low computational cost.

B. PathFinder for Network Debugging

Common tools such as *traceroute* and Looking Glass (traceroute services provided by major ISPs) are used for network diagnosis: destination reachability, traffic forwarding routes. However, traceroute can only provide these information from the machine made the request to the destination, and not all ISPs provides looking glass services.

In PathFinder, users can get path information, latest snapshots of network footprint and traffic information from participating PathFinder ASes, therefore, a more comprehensive view of their networks' visitors. For example, a web service

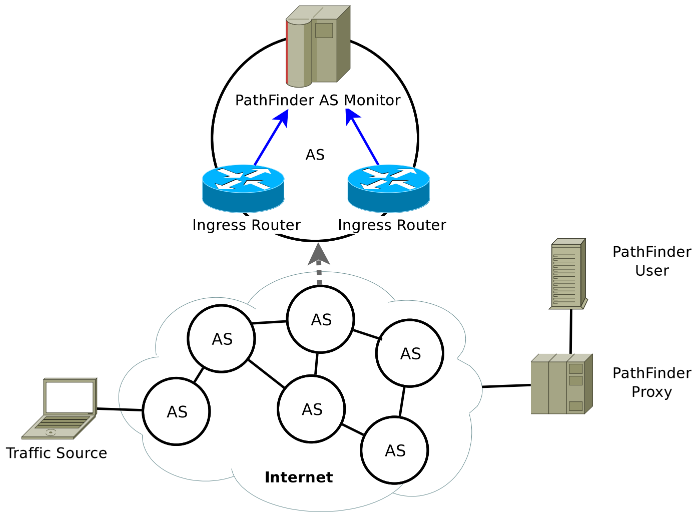

Fig. 2: PathFinder Architecture

administrator could use PathFinder to find out why the web traffic has changed dramatically: could it be some AS on the Internet is blocking the traffic to this web service? or a link failure in the transit network?

III. Overview of PathFinder

For PathFinder to be a practical solution, the design of the system should adopt several basic principles. First, it must be easy for ASes to deploy. A system that requires hardware or software changes to the existing infrastructure would incur a high operational cost which could hinder the deployability of the system. Second, the system needs to be scalable. The system's performance should not be negatively affected by either its level of deployment or the number of different packet sources. Finally, the system should allow a user to determine the possible forwarding path(s) for the realtime traffic in a timely manner.

PathFinder is a log-based system that allows a user to construct the possible AS-level path(s) for its inbound traffic. Figure 2 shows a high-level architecture of the system. The system involves three entities:

- **PathFinder User** who requests PathFinder service and retrieves PFLogs.
- **PathFinder Proxy** who handles user requests and informs ASes to produce PFLogs;
- **PathFinder AS Monitor** who performs computations on the packets/flows toward users and generates PFLogs.

PathFinder requires each participating AS to connect every edge router's traffic monitoring port to the PathFinder AS monitor (directly or indirectly). Popular monitoring techniques such as sFlow, NetFlow/IPFIX or traffic mirroring can be used in this case. Each AS monitor only communicates with PathFinder proxies so that an AS monitor can produces PFLogs without inter-AS collaboration.

Figure 2 shows a high-level view of PathFinder system. First, a PathFinder user chooses and connects with a

PathFinder proxy. This proxy will broadcast the user's request to all AS monitors. AS monitors produce PFLogs based on user's request, and sends the log back to the proxy as soon as the log is generated. Finally, the proxy will relay the PFLogs back to the user, and the user can start to reconstruct the inbound traffic topology, footprint and/or bandwidth information.

A. *PathFinder Workflow*

When a PathFinder user requires PathFinder service, it will first send out a request to the PathFinder proxy to ask for help, which includes the information this user wants from PathFinder.

Proxy allows only the authenticated users or prefix owners to retrieve the path information. If the proxy verified the user identity and ensured that the user does own the IP prefix block, it will broadcast the user request to some or all AS monitors depends on the user's request. Meanwhile, the user will wait for the proxy to relay the PFLogs.

After an AS monitor received the request from the proxy, the AS monitor will check the information that the user wants. If the user wants more than topology information, this monitor will begin to capture the packets/flows toward the user's IP or prefix. Every AS monitor's role is to produce a PFLog that contains some/all information defined in the usage model section II.

If the user requested source-based traffic footprint information, the AS monitor will record each packet it has seen in a traffic digest table, and the AS monitor could further use this digest table to produce PFLogs. To produce a log with source information is not easy; an AS monitor may need to process packets from high bandwidth links, and recording packets in such case would require the data structure to perform a minimal number of operations per packet. We will elaborate the design choice for the data structure in the next section IV.

When the PFLog is ready, the AS monitor will forward the log to the proxy, and the proxy will then relay the log to the user who made the request. User then can use the information in the log to reconstruct the AS-level path.

IV. Flow Logging

A PathFinder AS monitor's main task is to generate PFLogs for users. This task includes finding the AS-level forwarding path from routers to user, and recording packet sources it has seen towards a particular destination. In PathFinder, recording packet sources is a major problem we need to solve and we call such process flow logging, as each AS monitor presents the source-based footprint in an aggregated form to PathFinder users.

To generate PFLogs, each AS monitor needs to perform the following tasks:

- identifying the ingress routers in an AS when the AS has multiple border routers;
- discovering the AS-level path(s) from the AS to the packet destination;
- maintaining a data structure that helps to generate source-based traffic footprint with low computational overhead.

Depending on the tier of an AS, it could have multiple border routers, and each border router could be an ingress or egress router. PathFinder AS monitors utilize the traffic monitoring features on ingress routers to generate PFLogs. For the AS monitor to determine which of the routers are the ingress nodes, it needs to query the Routing Information Base (RIB) on each edge router and ask what the next-hop router is for it to forward traffic toward the user. If the next-hop router is within the AS, the router in question is an ingress node.

Note that BGP router vendors such as Cisco, Juniper and Brocade, all have their own network debugging interface to query AS-level paths [14][15][16]. Thus, AS monitors only need to read the path to determine whether a router is ingress node or egress node, and AS monitors can discover the AS-level path(s) by collecting the path(s) provided by egress router(s).

Figure 3 shows an example of a PathFinder monitor setup within an AS. For the sake of simplicity, we do not include the technical details of what traffic monitoring technique an AS uses. To generate source-based traffic footprint, an AS monitor

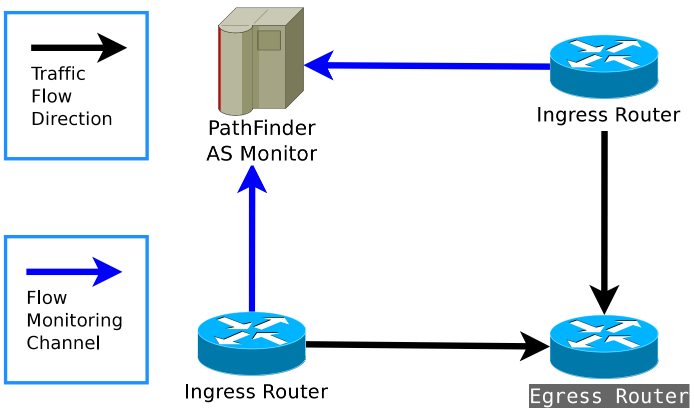

Fig. 3: An example of Flow Logging at an AS

can choose hardware telemetry solutions such as NetFlow, which utilize specialized hardware to aggregate packet headers into flow logs inside the router. However, NetFlow's export process takes time, and the capability varies based on different router hardware design. On the other hand, a software/CPU-bound solution is typically achieved by mirroring or tapping the traffic of a router, and use a generic server to handle the packets or packet headers directly. The software solution allows the whole system to have faster response time, because it can set arbitrary traffic monitoring window size. In this paper, we assume an AS does not run hardware telemetry solution but is capable of running software solution and mirroring/tapping the network traffic. We developed a trie-based data structure for AS monitors to generate source-based traffic footprint at a minimal computational cost even when facing high bandwidth traffic.

As soon as a PathFinder monitor of an AS can retrieve packets or packet headers from its routers, it needs to record them, and produce PFLogs for PathFinder users. Like log-based IP traceback systems, finding a good data structure as a traffic digest table is crucial to PathFinder. We use it to record traffic sources and bandwidth information at each AS monitor.

Given a packet header, the data structure of PathFinder has to provide three basic functions:

1) finding each packet source in the digest table;
2) inserting new packet source to the digest table;
3) updating packet source's corresponding bandwidth information if required.

These functions should invoke the least amount of computation possible, and minimize the storage cost.

As a user in PathFinder can request different information from AS monitors, the time and space complexities for recording different information vary. For example, a user who requests topology information from PathFinder uses less computation power than a user who requests source-based traffic footprint information.

In the following subsections, we first cover the lightweight scenario that does not require source-based user footprint requests, and we then describe the use of PathFinder AS Monitors to produce PFLog with source-based footprints.

A. PFLog without Source-based Footprint

From the usage model section II, we know a PFLog without source-based footprint can still contain topology, source-agnostic footprint and bandwidth information.

To produce topology information, the AS reports its AS-level forwarding path from itself to the PathFinder user who made the request; each AS monitor can either query router's RIB to get the AS-level path, or run *traceroute* program from itself to the PathFinder user's IP, and convert the result to AS-level path.

In addition to topology information, an AS monitor produces source-agnostic footprint by checking whether there are any packet traveling from its ingress routers to the PathFinder user. If it sees any traffic, then this AS is part of the source-agnostic footprint from user's view; if a user requested source-agnostic footprint, only AS monitors who see traffic toward the user in their networks will send AS-level paths to this user.

As this scenario does not consider any source information of the packets, producing bandwidth information in this scenario would involve the AS monitor to extract the packet size in the packet header and add this value to a summation variable. The AS monitor could also count the number of packets it has seen in a monitoring window, and later produce packet per second information as per user's request. Depends on the user's request, an AS monitor can record one or both bandwidth related variables.

B. PFLog with Source-based Footprint

To produce source-based footprint, each AS monitor needs a data structure to hold all the source addresses (IPs or IP prefixes) it has seen toward the user. For example, if an

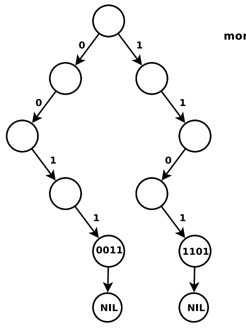

Fig. 4: /4 prefixes in a bit trie

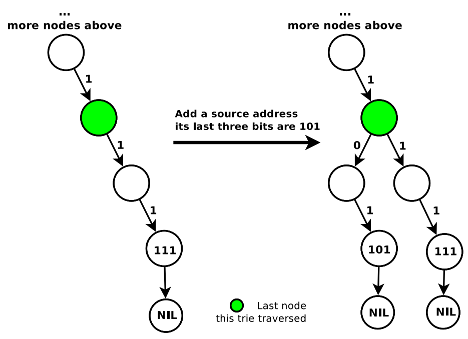

Fig. 5: Partial flow logging trie

AS monitor sees traffic coming from three different IP addresses, and user requested source-based footprint with flow-based bandwidth information, this AS monitor needs to record these IPs in the data structure along with their corresponding bandwidth-related information.

Since AS monitors need to work with high bandwidth links, it is desirable to reduce the cost of the recording process of the digest table, so that AS monitors can process more packets at a time. The obvious solution for this is to use hash table for recording unique addresses. However, AS monitors also need to consider the size of the PFLog: if each AS monitor produces an 1 MB PFLog on average and we assume full deployment of PathFinder; it would cost a user to download more than 5,000 MB worth of PFLogs, and it is certainly not an ideal solution when the user is under a DDoS attack, who wants to gather traffic information as soon as possible.

We need a digest table that provides fast lookup and insert speed, also, it needs to allow AS monitor to aggregate IP addresses and reduce the PFLog size if necessary. Our choice is to use a customized *trie*.

C. Trie-based Data Structure

Trie is similar to search tree, however, a trie does not contain the key associated with each node as in a search tree, instead, it uses the traversed path to represent a key.

Figure 4 shows an example of storing source addresses in a trie, and we use /4 prefixes as source addresses to simply the example.

When an AS monitor performs a source address lookup, it will traverse the trie according to the key value. For example, if an AS monitor were to lookup the source prefix 0011, it starts with the first bit of the key, 0, and it then go to the left child from the root. It then reads the second bit of the key, 0, again, it traverse to the left child. Eventually, it will hit the bottom of this trie with four traversals. The AS monitor can determine if there is an exact match by checking whether the last node it traversed is a leaf node or not. If it is, it concludes there is a prefix 0011 in the trie. In a trie, both lookup and insert speed is in $O(k)$ time, where k is the length of the key. As a comparison, consider a balanced search tree, which requires $O(\log n)$ time for both operations with n nodes, and $O(k)$ time for each node for key comparison. In this case,

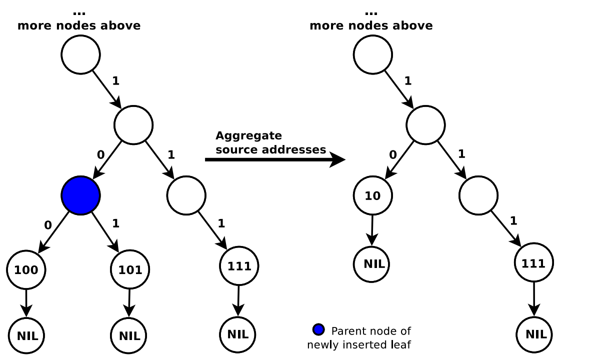

Fig. 6: Aggregate source addresses during flow logging

the value of n is theoretically bounded by the size of the IP space, while k is bounded by the length of the IP address. (32 or 128 bits, depending on the IP version.) Clearly $O(k)$ is much improved over $O(\log n)$.

D. Trie for Source-based Footprint

We have covered the basics of how trie works for flow logging, and we now present details of AS monitors recording source-based footprint.

For each source-based footprint request, each selected AS monitor will create an empty trie to record source addresses. Each AS monitor then start to retrieve packet headers from ingress router(s), and search the trie with packet's source address as the key. As shown in the figure 5, if the AS monitor sees a packet its source address ends with 101 (we assume this source address traversed a same prefix to get to node in the figure); the monitor will not be able to find it in the trie to the left in the figure, it will then create two trie nodes, and add them to the node that *find* function lastly traversed. Next, the AS monitor increments the leaf counter for per-flow bandwidth information, if the user has such requirement. Now, assume the AS monitor sees a source address shares the same prefix as the above example except its last three bits are 100, the AS monitor will repeat the above steps, and proceed to the last step, as shown in figure 6: the AS monitor checks the newly inserted leaf's neighbor, and it find out there is a neighbor node and the neighbor node is a leaf, we then take the summation of both nodes' couters, and save it to their parent node, and we set parent node as a leaf node (two nodes will be freed from memory). Note that this last step will not only apply to the IP address but prefixes as well; the new leaf node in figure 6 may potentially be aggregated with its neighbor, if its neighbor became a leaf node in the future.

From the above steps, we can tell if an AS monitor receives packets with high locality in terms of source addresses, the lookup time for each source address will become faster as they get aggregated to higher levels; trie's *find* function becomes faster as the less nodes it needs to traverse. Listing 1 shows a basic trie node that is being used in flow logging. Note that a trie node does not necessarily need to include the *counter* variable unless the user wants per flow bandwidth information.

```
struct {
    struct Trie* children[2];
    bool isLeaf;
    int counter;
} Trie;
```

Listing 1: Basic Trie Node Structure in Flow Logging

The steps for flow logging is shown in algorithm 1, and we assume the following functions exist:

```
Trie *find
(Trie *root, Trie *latestParent, SrcAddr key):
    return a node that is either the node
    with exact match or longest prefix match,
    and keeps updating the parent pointer
    for the latest traversed node.

void setLeaf
(Trie *node):
    set this node as a leaf node and remove
    both children from this node.
```

Listing 2: Functions in Flow Logging

Algorithm 1: Flow Logging with Lossless IP Aggregation

```
Trie *trie = initialize a trie;
Trie *parentNode;
for each packet P received from a router do
    Source *src = extract source address from P;
    Trie *node = find(trie, parentNode, src);
    if (!node->isLeaf) then
        node = insert(trie, parentNode, src);
    end
    node.counter++;
    Trie *neighborNode = neighbor(parentNode, node);
    if isLeaf(neighborNode) then
        parentNode.counter = node.counter +
          neighborNode.counter;
        setLeaf(parentNode);
    end
end
```

E. Trie Optimization

By using bit trie in our flow logging process, a lookup operation requires 32 trie traversals for each 32-bit IP address, and 64 trie traversals in the IPv6 network.

Traversing the trie is extremely fast as long as the source addresses has high locality; top-level trie nodes in a trie will be cached in CPU cache than main memory. However, an AS in a real world may encounter packets from many different IP prefixes, thus will use more cache space to save the top-level nodes. Intuitively, we want to reduce number of top-level trie nodes to be traversed, so to increase the lookup speed. As the example shown in figure 7, we need a way to logically use 1 node in the trie to represent 24 nodes in the left trie.

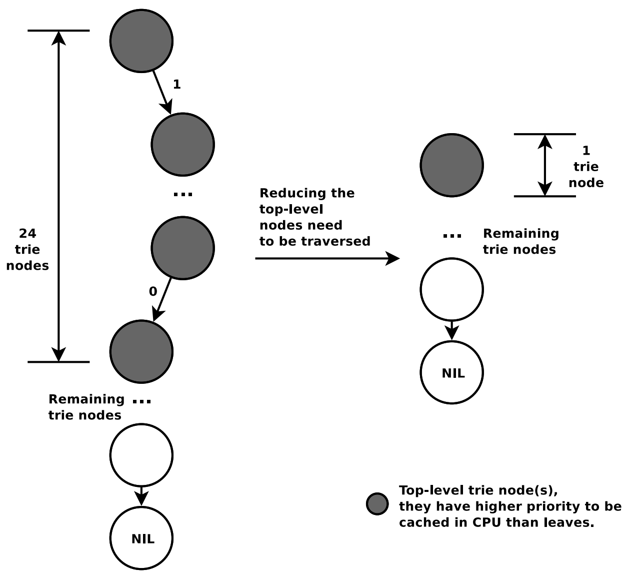

Fig. 7: Trie optimization in flow logging process

We can solve this problem by indexing the prefixes of source addresses; creating an array of trie pointers with its size as the prefix length. As the example shown in figure 8, to reduce 24 top-level trie nodes, we would create an array of pointer with the size of 2^{24}; a /24 prefix in figure 7 now became just one pointer in this array. The lookup cost reduced from 24 trie node traversals to one index lookup. The lookup operation

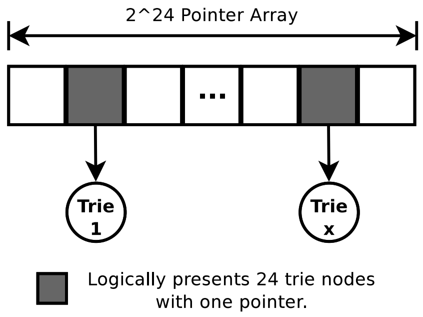

Fig. 8: Trie optimization with auxiliary pointer array

then has eight remaining traversals in the worst case, for an IPv4 network.

To further accelerate the lookup speed, we can attach bit arrays to the pointer array, if we do not wish to update the value of the leaves. In such acceleration scheme, only two pointer lookups are required.

Note that unix-like systems implement *memory overcommitment* by default (virtual paging); if a program tries to allocate a large pointer array in the system as in the example above, instead of allocating the physical space that will use 128 Mbytes ($2^{24} * pointerSize$), the system will create a virtual page table, which will only allocate space at the element being accessed.

V. Proxy Log Collection

As soon as AS monitors logged traffic footprint, they will need to transmit the footprint to the PathFinder proxy. PathFinder proxies play a critical role in making sure users can obtain PFLogs from the AS monitors, and it should evaluate if it is possible to deliver the footprint in time. In this section, we describe the PathFinder proxy design and justify the design choice.

One of the main applications of PathFinder is to help DDoS victims defend against DDoS attacks. The system should have enough redundancies to protect itself before it can help others, and each proxy can be protected by a DDoS solution as it can be a PathFinder user by itself. To achieve this objective, we place multiple proxy nodes distributed at different physical locations, so that in the case of an attack on the proxy nodes, the attack traffic volume has to be large enough to overwhelm multiple routing paths. Thus, the system stands a high chance to serve the user when needed.

Due to the traffic dynamics in the Internet, it is impossible to know total PFLog size of each user. It is possible that the total PFLog size is too big for user to retrieve in seconds. Although flow logging uses leaf aggregation mechanism to reduce PFLog size, it does not provide an upper bound of the PFLog size. If an AS encounters packets to a destination with sparse source addresses, leaf aggregation mechanism will not be triggered at all. Thus, we need a different approach to ensure the PFLog size is reasonable for both proxy and user to fetch.

A PathFinder user can define the total PFLog transmission threshhold value, T_{log}. This value tells a PathFinder proxy the maximum network data a user is willing to spend on PFLogs, and the proxy will ensure the size of all PFLogs from all participating AS monitors, P_{total}, is less than T_{log}. We developed a source-based PFLog aggregation scheme for this purpose, and figure 9 shows an example of this scheme. It requires each AS monitor to calculate the PFLog size

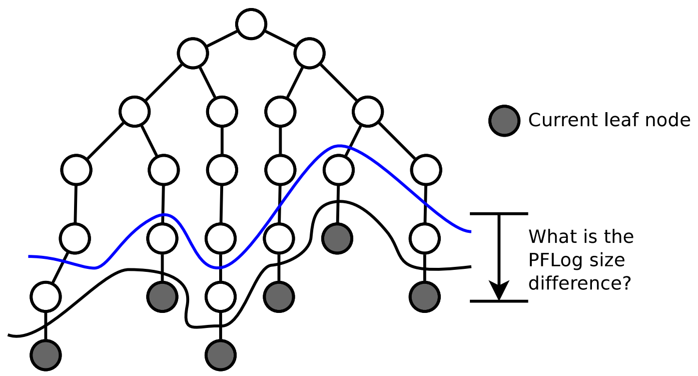

Fig. 9: Calculating Δ_{log}

difference, Δ_{log}, between the current trie's prefix log and the prefix log after aggregation; an AS monitor will count how many leaves to calculate the prefix log size, and it then assumes all leaves' parents are now the new leaves and calculate the prefix log again to get Δ_{log}. Each AS monitor will then send Δ_{log} and the current log size to the proxy.

If the condition $P_{log} \leq T_{Log}$ is not true, proxy will select AS monitors with largest Δ_{log}s for further aggregation in order to meet the condition; it is possible that all AS monitors will be selected if proxy cannot meet the condition in the first round. This scheme is beneficial to PathFinder AS monitors that receive a wide range of source addresses due to the large-scale IP spoofing scenarios, and PathFinder proxy would select these AS monitors to run the aggregation in order to meet $P_{log} \leq T_{Log}$.

VI. Source-based Path Reconstruction

As we mentioned in section II, when a DDoS defense system needs to apply source-based filtering strategy, deploying traffic filters at the ASes carrying corresponding traffic is especially important; a defense system installs filters at other ASes will be wasting precious hardware resources. In this section, we propose a recommended scheme for PathFinder users to reconstruct the AS-level paths when PFLogs contain source-based footprint.

For each PFLog that contains source-based footprint, it includes a AS-level forwarding path from the AS who generated the log to the user. It is important to map each source-based footprint to the ASes who carry the corresponding traffic, and users then have the ability to query the corresponding AS-level path by source address/prefix; e.g., a PathFinder user may be interested in finding ASes who can stop DDoS traffic with source prefix as 10.10/16.

A. Constructing AS-level Topology

As Internet routing paths change all the time, we first need to build the AS-level topology that can support routing updates: if a user requires up-to-date topology information, PathFinder AS monitors will forward the new route to this user if the route to the user changes, it should be able to change its local topology without heavy computation overhead.

One approach is to utilize a linked list to chain the topology together with the help of a hash map. As shown in the figure 10, each AS number is a key to the hash map, and it is either mapped to an AS number or NIL.

Upon the user receives a PFLog, it extracts the AS-level path from the log, and adds AS numbers to the hash map if the number does not exist yet. The user then link each AS number to the next-hop AS number. For each forwarding path update from an AS monitor, the user updates each AS number's mapped value in this hash map. For example, if there is a path update from AS 1 in the figure 10, and the AS path from AS 1 to the PathFinder user is now $1 \rightarrow 4 \rightarrow 5$. The user will go through the hash map, and change the corresponding mapping; key 1 will be mapped to 4 instead of 3 in the figure.

B. Mapping Source Footprints to the Topology

As soon as the user built the AS-level topology, it needs to map each corresponding source address/prefix to the topology.

Note that we have two aggregation modes from flow logging and log collection sections, the leaf aggregation mode from

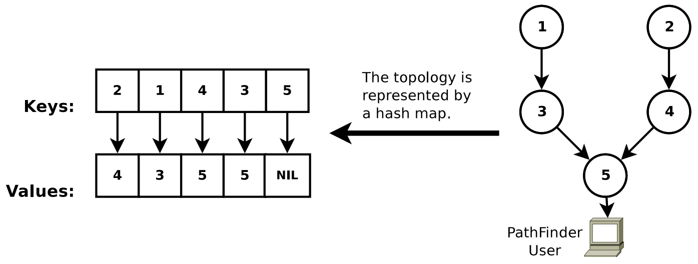

Fig. 10: Represent AS-level topology using a hash map

flow logging section keeps an accurate representation of what source addresses an AS monitor has seen. However, AS monitors selected during log collection section do not ensure the accuracy of the footprint. A PathFinder user should logically separate footprints based on the aggregation mode, and use the trie for loss aggregation mode as a secondary option to locate the responsible ASes.

The user can rebuild the trie using the same data structure defined in section IV for each mode, and each leaf node saves the beginning AS numbers for all AS-level paths. For example, if we assume AS paths $1 \rightarrow 3 \rightarrow 5$ and $2 \rightarrow 4 \rightarrow 5$ in the figure 10 have seen traffic from source address A toward the user, the trie should keep number 1 and 2 at address A's leaf node, and the user can then trace the forwarding path by traverse the topology defined in the above section.

VII. Evaluation

In this section, we present evaluation metrics, experiment setup, evaluation results and result analysis. We answer the following questions: (1) When does PathFinder provide good accuracy results? (2) What is the performance of PathFinder? (3) Is PathFinder trie data structure fast enough to keep up with high bandwidth links? In this section, we only evaluate the whole system under source-based footprint requests.

A. Metrics

We focus on the following metrics in our evaluation: PFLog accuracy, system performance, and processing overhead of trie-based data structure in PathFinder.

1) PFLog Accuracy: Although modern hardware and software allow full packet capturing, AS monitors sometime have to sample the traffic under certain scenarios. Such scenario could be abnormal traffic shape that requires extreme processing power, or administration policy that prevents full packet capturing. Due to the reasons above, we want to understand the system accuracy under different sampling scenarios. We define PFLog accuracy as the ratio between user-restored traffic sources and actual traffic sources in some monitoring period.

2) Performance: The Internet has high traffic dynamics, and the traffic sources one AS sees may change often. It is crucial for PathFinder AS monitors to deliver the latest PFLogs to PathFinder users as quickly as possible. We define PathFinder performance as the total time elapsed from the time a user issues the request to PathFinder proxy, to the point when user collected all traffic footprint.

3) Flow Logging Overhead for Source-based Footprint Generation: For AS monitors to produce source-based traffic footprints, they need to process network packets from routers. If an AS chooses software solution, the cost of such process should not be neglected. We study such cost by analyzing PathFinder trie-based data structure, specifically, we evaluate its insertion speed, lookup speed and memory usage.

B. Experiment Setup

We evaluate PathFinder PFLog accuracy by simulations, formulate the system response time as PathFinder performance, and use emulation to evaluate trie-based data structure (flow logging) overhead. We ran experiments on a desktop with Intel i7-4790 (3.6 GHz, 8 MB L3 cache) and 32 GB RAM @ 1600 MHz. We use two types of traffic source traces for our evaluation:

1) A set of real DDoS traces that represents its attack peak volume (Booter 9 from table I).
2) A set of synthetic DDoS traces contains traffic sources ranges from 150K to 64M traffic sources.

PFLog Accuracy. To evaluate PFLog accuracy, We first map each traffic source in Booter 9 DDoS traces to its AS number, and find the AS number of the traffic destination. We then build an inferred directed AS-level topology toward traffic destination, as the base of our simulation, the topology includes all AS numbers from the DDoS traces. We also assign a network delay value to each AS in the topology to imitate the real networks, and the value of each AS depends on its AS-level distance to the traffic destination AS. For each simulation in this experiment, we use a different combination of sampling rate and monitoring window size. The sampling rates range from 0.5% to 50%, and monitoring window sizes range from 0.5 seconds to 3 seconds.

Performance. To evaluate PathFinder system performance, we first define system components that can introduce delays, and the summation of each delay should represent the overall system performance. We use 25 inferred AS-level topologies to estimate the average network link overhead, in order to estimate the general transmission delay. For each topology, we assign 1 million of traffic sources (IP addresses) to all ASes. We assign number of IPs to an AS based on how big the IP block it owns, e.g., an AS owns a /16 prefix gains more traffic sources than an AS owns a /24 prefix.

Flow Logging Overhead. Since the Booter network traces only have thousands of traffic sources, these traces can not represent flow logging overhead accurately; the lookup cost for such small-scale DDoS traces is not representative; tries are CPU cache friendly, thosand-level of traffic sources can be easily cached in a modern CPU. To avoid CPU cache playing a significant role in our evaluation, we generate several sets of synthetic traffic sources to better estimate the overhead. We use different number of traffic sources and source distribution profiles to evaluate insertion, lookup and storage cost of PathFinder trie and two other tries. We implement our data structure in C, and we use LLVM compiler with optimization-level 2 for compiling.

C. PFLog Accuracy

In this section, we present PathFinder PFLog accuracy results with the method described in previous section.

Figure 11 shows the trend of PFLog accuracy under two variables: packet sampling rate, denoted as s, and monitoring

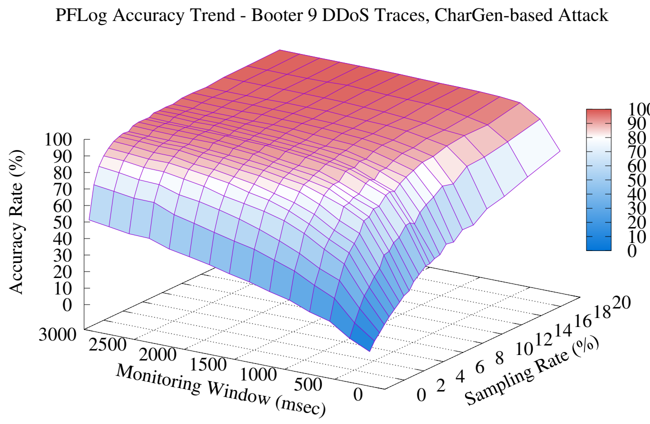

Fig. 11: PFLog accuracy trend under different variables

window size, denoted as w. From this figure, we see in general, PFLog accuracy goes up when either of these two variables increases. We notice that the benefit of increasing s became insignificant once s hits 10%, and the benefit of increasing w reduces as it approaches 2 seconds.

What is interesting from the simulation results is how s and w affect the PFLog accuracy: logically speaking, an AS monitor needs to process a same amount of the packets as long as $s_1 * w_1 = s_2 * w_2$, and our hypothesis is that both cases are likely to have a similar PFLog accuracy level. In other words, if an AS monitor records traffic sources with 1% sampling rate and 2 second monitoring window, the monitor should produce a similar result as if it records traffic sources with 2% sampling rate and 1 second monitoring window. Figure 12 shows PFLog accuracy comparisons for several sets of (s_i, w_j), and the results have validated our hypothesis. We notice that the result with higher sampling rate always outperforms the one with lower sampling rate but longer monitoring window size.

We find the overall PFLog accuracy is sound but the figure does not inform us the details of where traffic sources are being captured. Which AS-level hop captures a certain traffic source directly affects whether a full AS-level can be reconstructed for the traffic source. We thus reformat our simulation data, and use figure 13, 14, 15 and 16 to show readers a detailed breakdown of where traffic sources are being captured. Each figure represents the percentage of captured traffic sources versus total possible traffic sources, i.e., figure 13 shows the the percentage at traffic source ASes (hop 1). From the figures, we see when $s * w$ is small, second-hop and third-hop ASes are helpful to capture traffic sources that were not

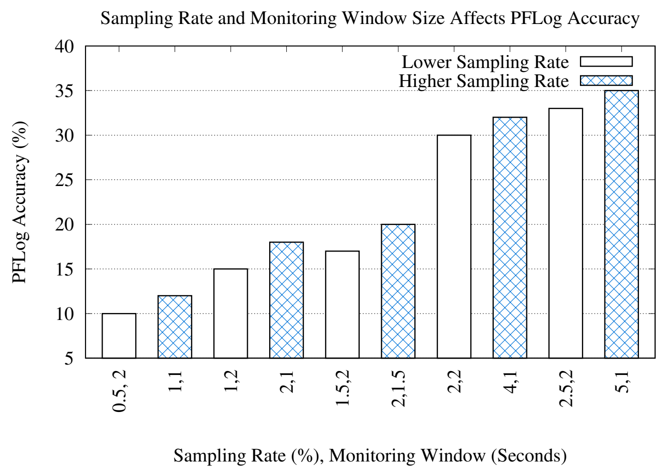

Fig. 12: PFLog accuracy trend when s*w are the same

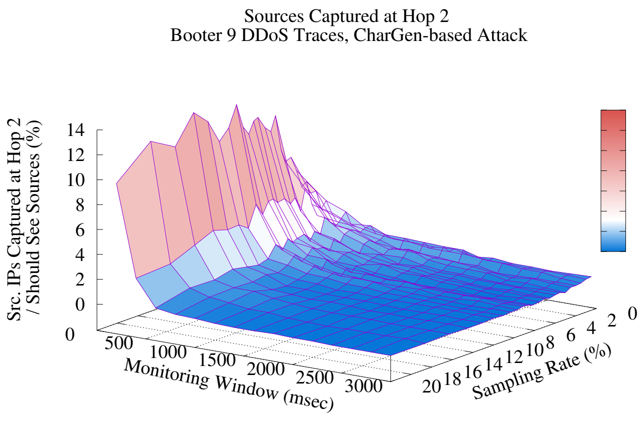

Fig. 14: Source being captured at second-hop ASes

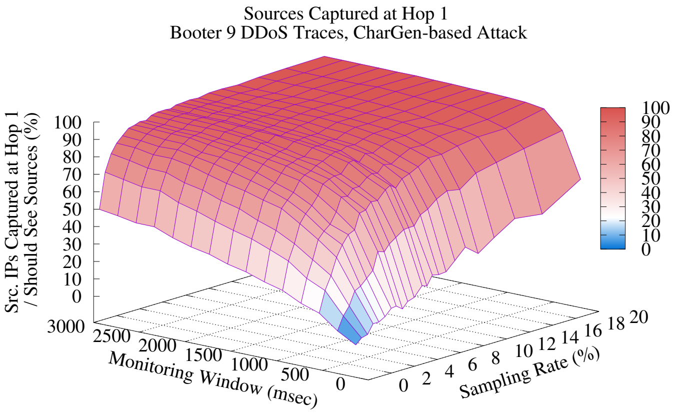

Fig. 13: Source being captured at first-hop ASes

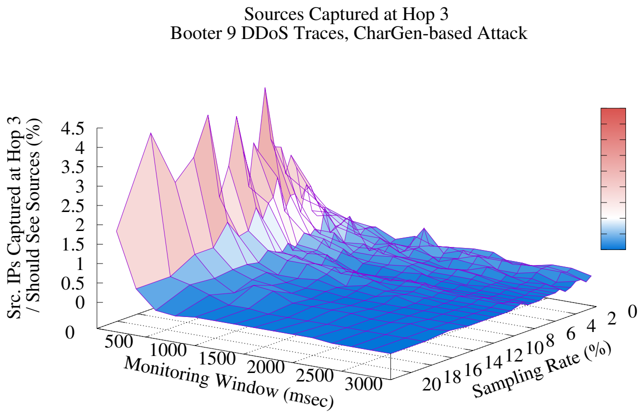

Fig. 15: Source being captured at third-hop ASes

captured at first-hop ASes. And we also see the fourth-hop ASes generally have much less chance to capture new sources that are not captured in previous hops. As soon as the first-hop ASes start to use some larger $s * w$ values, the rest of the ASes in the topology became less effective in capturing new sources.

At this point, we conclude that PathFinder system has good PFLog accuracy when it encounters a DDoS attack similar to Booter attacks or DNS reflection attacks. For AS monitors to achieve high fidelity traffic footprint results, in general they need to set sampling rate at 10% on traffic towards the same destination network.

D. Performance

To capture the system performance, we define the variables that generate delays in PathFinder system:

- N: network delay, time to transmit PFLogs.
- P: traffic footprint processing delay, time to monitor traffic and record IP sources.

For N, it consists of service request time, N_{srv}, between proxy and AS monitors, negotiation time N_{neg} between proxy and AS monitors to ensure total PFLog size is below user's requirement, and data transmission time that moves data from AS monitors to proxy, N_{data}. For P, it consists of insertion delay P_{ins} and lookup delay P_{look}.

To calculate the response time for each AS monitor, T_{AS_i}, we need the link bandwidth between PathFinder proxy and AS_i, denoted as b_{AS_i}, link delay between PathFinder proxy and AS_i, denoted as l_{AS_i}, number of unique attack sources per second rate at AS_i, denoted as s_{AS_i} and packet per second (pps) rate towards the PathFinder user pps_{AS_i}. Thus, the response time for AS_i is:

$$\begin{aligned} T_{AS_i} =& N_{src-AS_i} + N_{neg-AS_i} + N_{data-AS_i} \\ &+ P_{ins-AS_i} + P_{look-AS_i} \end{aligned} \quad (1)$$

Since the request packet and negotiation packet are neglectable, we can substitue both N_{src-AS_i} and N_{neg-AS_i} with link delay l_{AS_i} only, and we then use $\frac{data}{b_{AS_i}}$ to represent PFLog

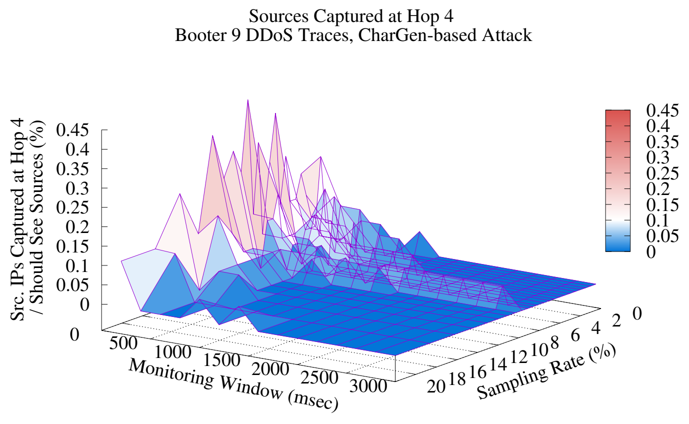

Fig. 16: Source being captured at fourth-hop ASes

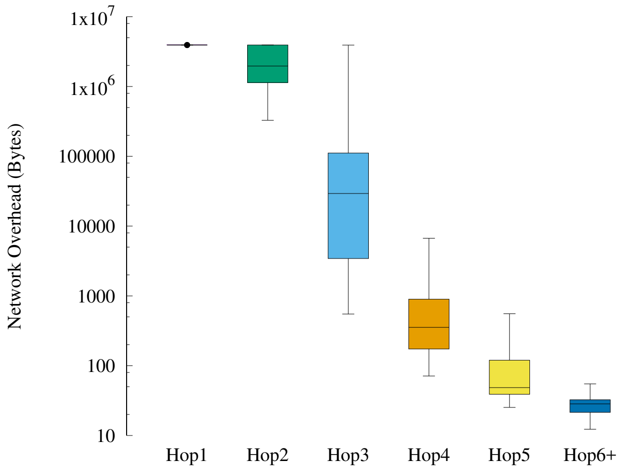

Fig. 17: Average network overhead based on AS-hop distance

transmission time $N_{data-AS_i}$:

$$t_{as_i} = l_{as_i} + l_{as_i} + \frac{data}{b_{as_i}} + p_{ins-as_i} + p_{look-as_i} \tag{2}$$

Finally, we denote the AS monitor's insertion speed as $insert_{AS_i}$ and lookup speed as $lookup_{AS_i}$, and we can replace P_{ins-AS_i} and $P_{look-AS_i}$ with $\frac{s_{AS_i}}{insert_{AS_i}}$ and $\frac{pps_{AS_i}}{lookup_{AS_i}}$, respectively. And the final formulation for T_{AS_i} is:

$$t_{as_i} = l_{as_i} + l_{as_i} + \frac{data}{b_{as_i}} + \frac{s_{AS_i}}{insert_{AS_i}} + \frac{pps_{AS_i}}{lookup_{AS_i}} \tag{3}$$

From this formula, we can see that N and P are both important to the performance. And if the system do not have good performance in P, the system can either choose to reduce monitoring window and sacrifice PFLog accuracy or increase monitoring window to makeup the accuracy due to the packet drop.

Figure 17 shows average network link overhead with a total of 1M traffic sources. We calculated link overhead based on AS-hop distance toward a destination. We see the further away an AS is from the traffic destination, the smaller network overhead it introduces. This figure gives us high confidence to say that PFLogs from most ASes are so small that we can use the link delay time to estimate their network delay in PathFinder.

E. Flow Logging Overhead

In this section, we evaluate and compare PFTrie with Adaptive Radix Tree (ART) [17] under different network source profiles; according to [17], ART in general, has better performance than Generalized Prefix Tree and hash table. As we mentioned in section VII-B, the attack sources from Booter attacks are too small for us to evaluate tries accurately. Therefore, we generated multiple sets of traffic sources ranges from 150K to 64 millions of IP sources, with different source density settings. We use the traffic sources to evaluate insertion/lookup speed and memory usage of PFTrie. As we compare the results for insertion and lookup speed under different source, we find high similarity of the results. Thus, we only include one figure each for insertion and lookup operations.

PFTrie uses a more excessive memory allocation scheme than ART, and we expect it has faster insertion speed than ART. Figure 18 presents trie insertion speed of the two data structures, and we see PFTrie performs better than ART in both figures: It costs around 700 ms for PFTrie to insert 16 millions of IP sources for both dense and sparse IP profiles, while it takes more than 1300 ms for ART to insert the same number of IP sources. In general, PFTrie uses 50% less time than ART to insert a same amount of IP sources.

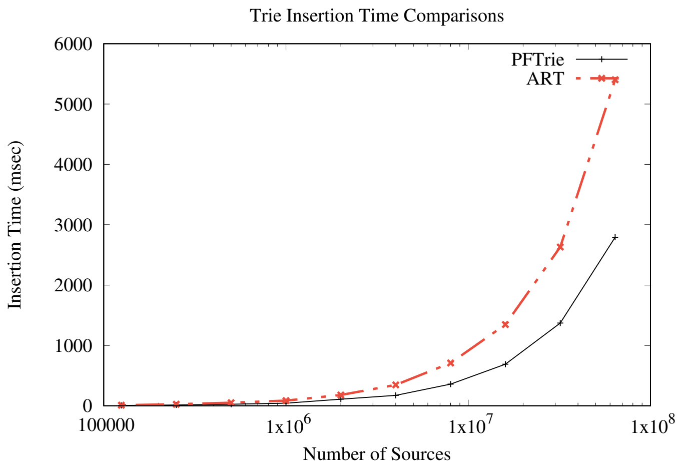

Fig. 18: Insertion speed evaluation

Figure 19 compares the lookup speed of PFTrie and ART, and we measure the time that each data structure takes to finish 15M lookups. As a result, PFTrie performs almost 10 times faster than ART when they both have more than 1M of sources, and the lookup speed of PFTrie is virtually constant. This is because a lookup operation in PFTrie only requires two lookups: one for the prefix array of the data structure, and one for bit array lookup to confirm whether an IP is in the trie. ART on the other hand, has a trie node management scheme, and the scheme helps to manage the height of the trie within a bound. The lookup speed of ART reduced by half at 1 million sources, and we believe the node management scheme reduces the trie height near this data point.

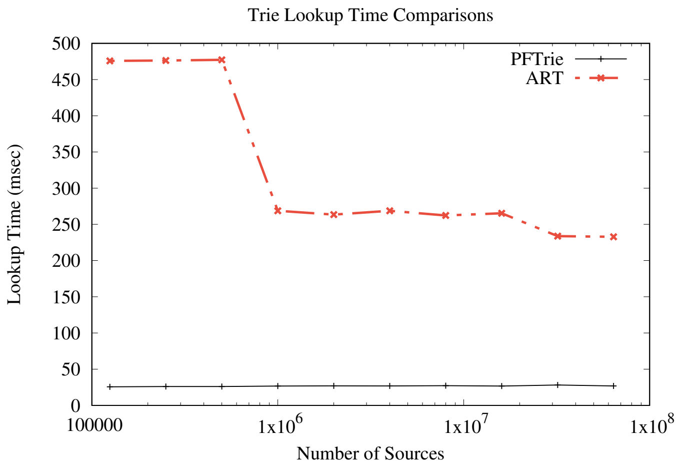

Fig. 19: Lookup speed evaluation: average 15M sources lookup time

In a real network environment, we expect flow logging process to involve a mix of insertion and lookup operations. We analyzed 9 sets of DDoS attack traces from Booter attacks, and we organized the insertion/lookup ratios in table I. We see a wide range of insertion/lookup ratios from 1/2.5 to 1/370, but in every trace the ratio is less than one. Although the scale of Booter attacks are not comparable to the Tbps-Level attacks such as the IoT attack to Dyn, these ratios still indicate that lookup speed is more important than insertion speed during the flow logging process.

Figure 20 presents the memory usage for PFTrie under different traffic source profiles. We see the memory cost is manageable even when we insert 64M sources to the data structure, and the bottom-up aggregation scheme helps reducing the memory usage when we have high percentage of consecutive IP sources.

Dataset	# of Atk. Srcs	Traffic Volume	Insertion/Lookup
Booter 1	4486	700 Mbps	1/72
Booter 2	78	250 Mbps	1/230
Booter 3	54	330 Mbps	1/370
Booter 4	2970	1.19 Gbps	1/14
Booter 5	8281	6 Mbps	1/2.5
Booter 6	7379	150 Mbps	1/5.5
Booter 7	6075	320 Mbps	1/3.6
Booter 8	281	990 Mbps	1/157
Booter 9	3779	5.48 Gbps	1/57

TABLE I: Average number of packets sent from a same source per second

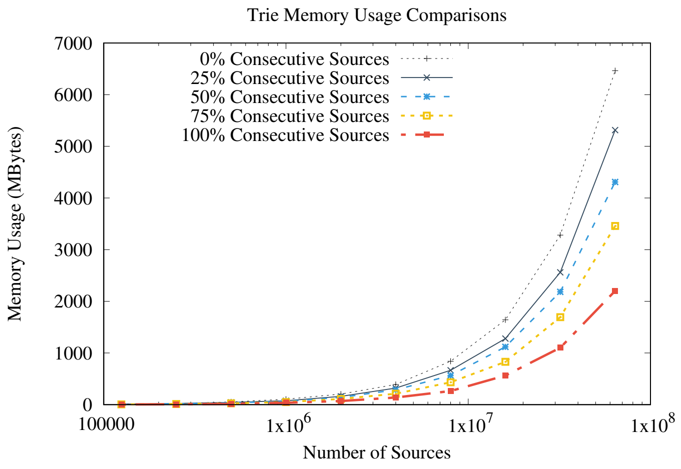

Fig. 20: Memory usage evaluation under different IP Profiles

Finally, PFTrie supports a feature that ART does not: *source address aggregation*. PFTrie maintains the traffic source addresses that have been aggregated during the entire flow logging process. This feature helps PathFinder AS monitors to reduce transmission overhead so they can deliver PFLogs to users faster. On the other hand, for ART to achieve the same transmission overhead, it needs to go through a series of steps to manipulate the internal trie nodes to aggregate IP sources correctly. Although ART uses less memory space than PFTrie, such scheme does not allow us to aggregate traffic sources with highest granularity. Data structures that apply similar memory management schemes need to consider such fact as well. In conclusion, PFTrie provides decent insertion and lookup speed when AS monitors need to process high bandwidth links.

VIII. Discussion

A. *Assumptions*

We made several assumptions for this work, all of which we believe to be reasonable assumptions for both an AS and a user:

- An AS participant has control over its edge routers.
- Packet or flow sampling rate of routers can be adjusted.
- The links between the AS logger and proxy will not be congested.
- There exists an out-of-band link between the user and the proxy that is not congested.
- For the same flow, an AS only has one egress router at a time.
- The forwarding path for an destination prefix is an accurate representation of all the IP addresses it covers.

B. *Deployability of PathFinder*

In this section, we conclude a list of deployment requirements for an AS to deploy PathFinder: 1). the routers need to have either traffic mirroring or NetFlow/IPFIX capability; a majority of the network routers nowadays support at least traffic mirroring if not NetFlow/IPFIX [18][19]; 2). the AS needs to have a generic server to run PathFinder AS monitor software; if the AS has been analyzing its network status, it implies there is a server which can run PathFinder AS monitor software; 3). the machine with processing power similar to the one in the PFTrie experiment is sufficient to handle high bandwidth link; 4). the monitor needs the access to each edge router's management interface to query the AS-level forwarding path from the RIB; the implementation of this work is straightforward, and we do not consider it as a road block to the deployment of PathFinder.

Unlike previous traceback systems, PathFinder has no hardware or software modifications on the routers, and each AS does not need to coordinate with other ASes in the Internet.

C. Network Overhead Optimization

Although PathFinder has an efficient flow logging scheme, it does not determine the system response time; The size of PFLog can also affect the response time, and in this project, we have a contemporary solution to control the total size of PFLogs. Thus, reducing the network overhead is a task we would like to explore on in the future.

D. AS Privacy

ASes are less likely to divulge their private information such as internal network topology to others. In the previous traceback systems, strong collaboration between routers is necessary for producing accurate path information. However, router-level collaboration implies inter-AS collaboration, and it is normally a road block for ASes to overcome. The router-level forwarding path also gives the user the ability to infer the real topology of any AS that is part of a traceback system.

In PathFinder, there are two pieces of information an AS needs to share to the proxy and the user:

- Source IPs/prefixes were seen by an AS
- AS-level forwarding path

Each AS participant only exposes its forwarding decisions and source IPs/prefixes to each user's request. PathFinder does not require any inter-AS collaboration in the system; thus it eliminates the possibility that a malicious AS could acquire sensitive topology or traffic information.

Additionally, each PathFinder user can only obtain the AS-level path information and traffic source prefixes that are related to itself, and cannot know other users' information.

IX. Background and Related Work

There are many approaches to determining the forwarding path of a packet. We group previous packet forwarding path traceback papers into three categories: IP traceback, AS-path inference and Reverse traceroute. Most existing IP traceback solutions are specifically designed to help DDoS victims reconstruct the router-level forwarding path of a packet to defend against DDoS attacks. This approach generally requires modifications to the hardware and/or software of routers. On the other hand, although AS-level path inference systems do not require changes to the network equipment, they may impose a limit on the accuracy of the inferred result. Finally, the reverse traceroute approach combines path inference and traceroute, allowing the packet forwarding path to be reconstructed with a high degree of accuracy. The design of such systems, however, limits their scalability when dealing with large-scale DDoS attacks.

We then introduce how a packet's forwarding path is determined, in existings distributed DDoS defense system.

A. IP Traceback

IP traceback was first introduced by Burch and Cheswick [20], who propose that tracing packets back to the source hop-by-hop is a necessary step to address DDoS attacks, since the source IP of a packet might be spoofed. Many IP traceback systems have been proposed over the years, and several approaches have been attempted, e.g. link testing, messaging (ICMP), marking, logging, overlay, etc [21]. Two of these IP traceback approaches, marking and logging, are most developed over the years.

1) Marking Approaches: The main idea behind a packet marking scheme is to let the participating routers add different segments of the forwarding path to each packet header fields that are less frequently used. The host that requested the traceback service can then examine the modified packets, and can thereby acquire the forwarding path for this packet. Savage et al. proposed a probabilistic packet-marking system that adds route information to the IP identification field [22], and the system does not simply mark all the packets all the time, instead, marking a packet is determined by a probability value. Yaar et al. proposed FIT [23], which is also a probabilistic packet-marking system. FIT mainly focuses on a partial-deployment scenario and requires little hardware change on routers.

The packet-marking approach has a simple operational model, and path information is included in the packets. However, the drawback of such approach is that the traceback accuracy greatly depends on how many marked packets a user can receive. E.g. in a high-volume DDoS attack, it is incredibly hard for a user to examine each packet for path information, and reconstruct the packet forwarding path.

2) Logging Approaches: The packet logging approach requires participating routers to store packet digests in their memory, and then determines if a router has seen a certain packet in the past. For every user's packet forwarding path query, the routers will use the stored digests to produce the path collaboratively. Snoeren et al. proposed a hash-based IP traceback scheme called SPIE [24]. SPIE requires a moderate amount of changes to participating routers, and uses a space-efficient Bloom filter to log each packet in the network. Li et al. also use a Bloom filter to implement a digest table in [25], and further apply a packet-sampling technique to deal with high-speed links.

A packet logging approach imposes virtually no network throughput overhead, as the control plane does not stall packet forwarding behavior. Additionally, as long as the monitoring hardware is capable of processing packet headers at line rate, the accuracy of the approach remains extremely high. This approach does require both a moderate amount of hardware changes and inter-AS collaboration. And inter-AS collaboration introduces extra network overhead and decreases overall system response time.

3) Hybrid Approaches: In addition to the two aforementioned methods, there are many other hybrid approaches which combine elements of different traceback approaches [26], [27]. These solutions contribute more efficient traceback systems, however they also inherit the drawbacks from their parent schemes.

4) PathFinder vs. IP Traceback: An IP traceback system is used to locate the source network that generated the packet in question. This system allows a user to contact the source network to filter the traffic. PathFinder is different from an IP traceback system for two reasons: First, PathFinder is not an always on system like IP traceback, which constantly recording or marking packets. Second, PathFinder only keeps records of the packet source prefixes and traffic information as traffic footprint, as opposed to a log-based IP traceback system that uses a combination of IP packet header fields.

As a result, PathFinder does not trace a particular packet back to the source network, since it only uses the source IP address during the flow logging process. PathFinder may produce path information such as one source prefix maps to multiple AS-level paths, due to IP spoofing or route changes. This information helps a user to quickly narrow down the possible AS-level traffic forwarding paths for each received packet in question.

In general, traceback systems have deployability issues such as fixed network throughput loss in packet-marking scheme or high computational cost in packet-logging scheme. PathFinder, on the other hand, has lower deployability requirement but it also has higher chances to produce multiple AS-level paths for an IP source/prefix.

B. AS-Path Inference

Unlike the router-level IP traceback approaches, AS-level path inference takes a different approach to discover the AS-level forwarding path between two edge networks. AS-level path inference does not require any changes in the network infrastructure, and allows the user to infer the path using merely the BGP information.

According to Mao et al. [28], over 60% of all AS paths are asymmetric. To address this problem without having control over either the network infrastructure or the network edges is particularly challenging. The authors use multiple vantage points of the BGP routers to build an AS graph in order to infer the correct AS-level forwarding path. Authors developed a new AS relationship inference algorithm and a technique to infer the first AS hop. The evaluation shows that the work achieves 70%-88% accuracy in AS path inference. This approach is useful for finding critical ASes for forwarding traffic to a particular destination. However, this approach does not include realtime traffic information, thus made it a less attractive solution to help defend against DDoS attack:

C. Reverse Traceroute

As the most used network debugging tool on the Internet today, traceroute provides a user with a very detailed router-level path from the user to an IP address. However, Internet routing is asymmetric, and traceroute does not provide the reverse path from the IP address to the user.

Katz-Bassett et al. proposed a reverse traceroute scheme [29] that enables the user to query the packet forwarding path even when the user has no control of the packet source. Reverse traceroute schemes employ both measurement studies and probes that are physically distributed on the network to stitch together the forwarding path piece-by-piece. The authors claim that in the median case they can find 87% of the router-level hops, versus 38% accuracy if we trust the traceroute result.

Much like AS-level path inference systems, reverse traceroute is not an appealing solution for DDoS defense, for the following reasons: the response time of the reverse traceroute is simply too long; the cost to perform one reverse traceroute is too high, since during a DDoS attack the number of attack sources can be large, and it is not reasonable for the user to do a query for each attack source during a large-scale DDoS attack which could involve more a million of bots.

D. Approaches in Distributed DDoS Defense Systems

StopIt [10] uses a topology-based approach that utilizes BGP updates between the routers for locating the corresponding routers given a packet identifiers. Authors assume that the IP spoofing problem will be taken care of by their IP spoofing project Passport [30], thus the source address precisely maps to the traffic source network. However, given the asymmetric routing of the Internet, the topology-based approach introduces a flaw in this work: If the traffic source network is not part of the StopIt system, the system will have a hard time finding the next-hop network, since the forwarding path can be very different from the traffic source to the victim than the other way around.

AITF [7] uses a traffic-based approach for finding the responsible routers; an approach similar to packet-marking IP traceback. The routers in the system mark the router information directly to each packet, and the victim can see exactly what are the routers on the path for forwarding a particular packet. This approach however reduces the network throughput by 10% according to the AITF evaluation.

FireCol [9] assumes a full deployment of the system, therefore, filtering rules could be pushed from victim-side network in the beginning, then all the way up to the traffic source networks.

Both StopIt and AITF have major issues for finding the right place to install traffic filtering rules, and the need for a better solution is necessary to improve these collaborative DDoS defense systems. This new solution should not degrade the network throughput, and when a distributed DDoS defense system is under partial deployment scenario, this solution should allow the defense system to find the paths (ASes) that are most likely to carry the attack traffic.

X. Conclusion

In this paper, we introduced PathFinder, a traffic footprint collection system. We explored its use cases for both DDoS defense and network debugging purposes, and we believe PathFinder provides useful information to aid the problems. We proposed a flow logging scheme to produce source-based traffic footprint, and we demonstrated this scheme has the ability to work with high bandwidth traffic links. Both lookup and insertion speed of IP sources are faster than other data structures we tested, and it produces smallest PFLog without losing IP source accuracy. We then presented a log collection scheme that controls the total network overhead for all PFLogs, and it helps to control and reduce system response time.

In our future work, we plan to explore the possibility to shrink the total network overhead for PFLogs while maintaining or reducing the system response time. We also like to setup a testbed to have an AS monitor machine to get packets from real switches/routers, and see how fast the flow logging process is in a real word setting.

References

[1] Akamai. Q4 2016 state of the internet security report. [Online]. Available: http://www.cidr-report.org/as2.0://www.akamai.com/us/en/about/our-thinking/state-of-the-internet-report/global-state-of-the-internet-security-ddos-attack-reports.jsp

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

[3] Cloudflare. 400gbps: Winter of whopping weekend ddos attacks. [Online]. Available: https://blog.cloudflare.com/a-winter-of-400gbps-weekend-ddos-attacks/amp/

[4] Imperva. 650gbps ddos attack from the leet botnet. [Online]. Available: https://www.incapsula.com/blog/650gbps-ddos-attack-leet-botnet.html

[5] D. G. Andersen *et al.*, "Mayday: Distributed filtering for internet services." in *USENIX Symposium on Internet Technologies and Systems*, 2003, pp. 20–30.

[6] G. C. Oikonomou, J. Mirkovic, P. L. Reiher, and M. Robinson, "A framework for a collaborative ddos defense." in *ACSAC*, vol. 6, 2006, pp. 33–42.

[7] K. J. Argyraki and D. R. Cheriton, "Active internet traffic filtering: Real-time response to denial-of-service attacks." in *USENIX annual technical conference, general track*, 2005, pp. 135–148.

[8] R. Sahay, G. Blanc, Z. Zhang, and H. Debar, "Towards autonomic ddos mitigation using software defined networking," in *SENT 2015: NDSS Workshop on Security of Emerging Networking Technologies*. Internet society, 2015.

[9] J. François, I. Aib, and R. Boutaba, "Firecol: a collaborative protection network for the detection of flooding ddos attacks," *IEEE/ACM Transactions on Networking (TON)*, vol. 20, no. 6, pp. 1828–1841, 2012.

[10] X. Liu, X. Yang, and Y. Lu, "To filter or to authorize: Network-layer dos defense against multimillion-node botnets," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 195–206.

[11] Y. He, M. Faloutsos, S. Krishnamurthy, and B. Huffaker, "On routing asymmetry in the internet," in *GLOBECOM'05. IEEE Global Telecommunications Conference, 2005.*, vol. 2. IEEE, 2005, pp. 6–pp.

[12] V. Paxson, "End-to-end routing behavior in the internet," *IEEE/ACM transactions on Networking*, vol. 5, no. 5, pp. 601–615, 1997.

[13] Dyn. Dyn analysis summary of friday october 21 attack. [Online]. Available: http://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/

[14] Brocade. show ip bgp summary. [Online]. Available: http://www.brocade.com/content/html/en/command-reference-guide/netiron-05900-cliguide/GUID-FB705B22-1352-4606-9FB0-43C06131CFCE.html

[15] J. Stretch. Junos-to-cisco ios/xr command reference. [Online]. Available: http://web.archive.org/web/20140114070827/http://packetlife.net/wiki/junos-cisco-iosxr-command-reference/

[16] Cisco. Cisco ios ip routing: Bgp command reference. [Online]. Available: http://www.cisco.com/c/en/us/td/docs/ios/iproute_bgp/command/reference/irg_book/irg_bgp5.html

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

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

[19] ——. Configuring traffic mirroring on the cisco ios xr software. [Online]. Available: http://www.cisco.com/c/en/us/td/docs/routers/crs/software/crs_r4-3/interfaces/configuration/guide/hc43xcrsbook/hc43span.pdf

[20] H. Burch and B. Cheswick, "Tracing anonymous packets to their approximate source." in *LISA*, 2000, pp. 319–327.

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

[22] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Practical network support for ip traceback," in *ACM SIGCOMM Computer Communication Review*, vol. 30, no. 4. ACM, 2000, pp. 295–306.

[23] 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. IEEE, 2005, pp. 1395–1406.

[24] 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 Computer Communication Review*, vol. 31, no. 4. ACM, 2001, pp. 3–14.

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

[26] C. Gong and K. Sarac, "A more practical approach for single-packet ip traceback using packet logging and marking," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 10, pp. 1310–1324, 2008.

[27] X.-j. Wang and Y.-l. Xiao, "Ip traceback based on deterministic packet marking and logging," in *Scalable Computing and Communications; Eighth International Conference on Embedded Computing, 2009. SCALCOM-EMBEDDEDCOM'09. International Conference on*. IEEE, 2009, pp. 178–182.

[28] Z. M. Mao, L. Qiu, J. Wang, and Y. Zhang, "On as-level path inference," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 33, no. 1. ACM, 2005, pp. 339–349.

[29] 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 *NSDI*, vol. 10, 2010, pp. 219–234.

[30] X. Liu, A. Li, X. Yang, and D. Wetherall, "Passport: Secure and adoptable source authentication," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 365–378.