Target Generation for Internet-wide IPv6 Scanning

Austin Murdock[1,2], Frank Li[1,2], Paul Bramsen[1], Zakir Durumeric[2], Vern Paxson[1,2]

{austinmurdock, frankli, paulbramsen, vern}@berkeley.edu, zakir@icsi.berkeley.edu

[1] University of California, Berkeley [2] International Computer Science Institute

ABSTRACT

Fast IPv4 scanning has enabled researchers to answer a wealth of new security and measurement questions. However, while increased network speeds and computational power have enabled comprehensive scans of the IPv4 address space, a brute-force approach does not scale to IPv6. Systems are limited to scanning a small fraction of the IPv6 address space and require an algorithmic approach to determine a small set of candidate addresses to probe. In this paper, we first explore the considerations that guide designing such algorithms. We introduce a new approach that identifies dense address space regions from a set of known "seed" addresses and generates a set of candidates to scan. We compare our algorithm 6Gen against Entropy/IP—the current state of the art—finding that we can recover between 1–8 times as many addresses for the five candidate datasets considered in the prior work. However, during our analysis, we uncover widespread IP aliasing in IPv6 networks. We discuss its effect on target generation and explore preliminary approaches for detecting aliased regions.

CCS CONCEPTS

• **Networks** → *Network properties*; • **Security and privacy** → Network security;

KEYWORDS

IPv6, Scanning, Network Measurement

1 INTRODUCTION

Internet measurement has greatly benefited from recent advances that facilitate effective exploration of the global IPv4 address space. By exploiting the ability of modern hardware and connectivity to transmit over a million packets per second, tools like ZMap [12] and Masscan [18] have fundamentally enhanced the ability of researchers to conduct wide-ranging assessments of Internet services, including the use of cryptography in practice [4], uncovering network administrator behaviors [21], and tracking vulnerability remediation [11].

IMC '17, November 1-3, 2017, London, UK

ACM ISBN 978-1-4503-5118-8/17/11…$15.00
https://doi.org/10.1145/3131365.3131405

These tools leverage the density and limited size of the IPv4 address space: today's scanning speeds are such that it is feasible to exhaustively enumerate all possible IPv4 addresses in order to conduct comprehensive scans. However, as has long been recognized [3], IPv6's much larger address space renders exhaustive probing completely infeasible. This then raises the question for measurement researchers of how to obtain at least a degree of global IPv6 address visibility somewhat comparable to the comprehensive IPv4 visibility provided by tools such as ZMap.

While prior work has developed sophisticated techniques for inferring the underlying structure of how IPv6 network operators assign addresses in their networks [14], and, separately, for how to leverage IPv6 address assignment policies to abet network reconnaissance [17], the question of how to employ these insights to facilitate effective global IPv6 scanning remains.

In this work we consider the basic problem of how to draw upon a set of IPv6 *seeds*—i.e., collections of IPv6 addresses known to host systems of a particular nature—to determine additional addresses that are likely to prove fruitful for scanning. We consider the salient properties pertinent for leveraging a given set of seeds, and develop the general notion of *Target Generation Algorithms* (TGAs) that extrapolate from the set of seeds to identify target addresses to scan. To manage the intractable scale of the IPv6 address space, our analysis incorporates the key notion of a *probe budget* that specifies constraints on how many scan packets a researcher can tractably send. TGAs strive to extrapolate from a set of seeds the most promising IPv6 addresses to probe to find additional systems of interest, given the constraint of a fixed probe budget.

As a concrete instantiation of a TGA, we develop *6Gen*, an algorithm that leverages a set of seeds to identify dense address space regions. It operates under the assumption that dense regions of seeds correlate with dense regions of active hosts. 6Gen clusters similar seeds together into high density regions, generating scan targets of addresses within those regions.

While 6Gen's design is fairly simple, its utility becomes evident when using real-world datasets as seed inputs. From a train-and-test evaluation comparing 6Gen with Entropy/IP [14]—the state of the art algorithm in analyzing IPv6 address structure—we find that 6Gen can recover between 1–8 times as many addresses for five candidate seed datasets considered in the prior work; for one of the network datasets, it was able to predict over 99% of addresses. In addition, we employed 6Gen on an extensive DNS-based seed dataset and conducted active IPv6 scans of the generated targets, discovering over 55 M new active addresses. However, we also uncover the presence of large-scale IP aliasing, where all addresses within massive network prefixes respond. We develop an initial technique for detecting large aliased regions, and find that even after filtering discovered addresses in those regions, 6Gen still discovered over a million new IPv6 addresses across thousands of networks.

While our preliminary evaluation indicates that 6Gen may generate more effective target lists than Entropy/IP, this comes with the key consideration that Entropy/IP was designed specifically to illuminate overall address structure, rather than to produce scanning targets given a particular probe budget. An important item for future work is to analyze the underpinnings of these differences and subsequently refine such algorithms to enhance their efficacy at identifying promising targets for IPv6 scanning. Additionally, further developments in IPv6 scanning must contend with the IP aliasing that occurs in the IPv6 address space, where responsive addresses may not meaningfully equate to distinct hosts. A promising direction for more effective IPv6 scanning is tighter integration between the TGA and the network scanner, where a feedback loop can incorporate fresh scan results to inform the target generation as a scan progresses. With such developments, we can move towards wide-ranging assessments of the IPv6 Internet.

2 BACKGROUND

In this section, we provide a brief background on IPv6 addressing and introduce the domain-specific terms we use in this paper. We refer the reader to RFC 2460 [9] for a detailed description of the protocol.

IPv6 has a much larger address space than IPv4; IPv6 addresses are 128 bits. IPv6 unicast addresses consist of three parts: (1) a global routing prefix, (2) a local subnet identifier and (3) an interface identifier [20]. The interface identifier may be generated by the client using Stateless Autoconfiguration [30], by the server using DHCPv6 [10], or statically assigned [17].

We represent IPv6 addresses in a human-readable text format using eight groups of four hexadecimal digits, with each group representing 16 bits and separated by colons (":"). We call each hexadecimal digit (corresponding to four bits of the address) a *nybble*. An example IPv6 address is `2001:0db8:0000:0000:0000:0000:0011:2222`. As IPv6 addresses often contain many zero-valued nybbles, a compressed representation excludes leading zeros in each group, and substitutes the longest sequence of all-zero groups with a double colon notation ("::"). Thus, the example IPv6 address has a compressed representation of `2001:db8::11:222`. CIDR notation for IPv4 is identically defined for IPv6. We also denote an address range using nybbles with the wildcard value "?", indicating a dynamic nybble that can range across multiple values. For example, `2001:db8::?:100?` represents 256 addresses, including `2001:db8::5:1000`, `2001:db8::8:100a`, and `2001:db8::1003`.

3 RELATED WORK

Prior work on IPv6 scanning falls into three broad categories: (1) developing methods for extracting IPv6 addresses from publicly accessible data sources (e.g., DNS records), (2) analyzing known addresses to understand allocation patterns, and (3) designing algorithms that generate address targets to scan.

3.1 Extracting IPv6 Addresses

When querying for the IPv6 PTR record for an address prefix, Fiebig et al. [13] identified that many DNS servers respond differently if there exists a PTR record for some address within that prefix, than when such a record does not exist. Leveraging this insight, they mined IPv6 addresses from DNS servers by recursively querying for PTR records for address prefixes. However, not all DNS servers conform to this observed behavior, preventing Fiebig et al. from comprehensively extracting all IPv6 addresses in DNS records. In total, they collected a dataset of 5.8 M unique addresses.

Gasser et al. [16] explored a more extensive collection of addresses from various active and passive sources. Passive sources included network taps on a European Internet Exchange Point and the Munich Scientific Network's Internet uplink. Addresses were also actively collected from crawls of the Alexa Top 1 Million domains [2], TLD zone files, and DNS datasets from Rapid7 [26, 28] and CAIDA [6]. In total, the authors collected 148.6 M addresses via passive sources and 2.7 M addresses from active sources. To assess the liveness of these addresses, Gasser et al. extended ZMap [12] to support IPv6 scanning. They found that 76% of addresses from active sources were responsive to ICMPv6 pings, compared to 13% from passive network taps. They released their dataset of collected addresses on their website [15].

3.2 Identifying Addressing Patterns

RFC 7707 [17] describes several known practices for assigning IPv6 addresses that a network scanner may be able to leverage. These practices include embedding human-readable text in the address (e.g., `DEADBEEF`), using only the least significant address bits, encoding the port of a network service, or inserting the network interface's MAC address. The RFC additionally enumerates several methods for discovering active addresses, such as analyzing DNS records, performing traceroutes to known addresses, and monitoring peer-to-peer trackers.

Czyz et al. [8] compared the IPv4 and IPv6 firewall policies of dual-stack hosts. They identified 520 K dual-stack servers by performing DNS A and AAAA queries for domain names in the Rapid7 DNS ANY dataset [28]. They also found 25 K dual-stack routers by conducting the same DNS lookups on hosts in the CAIDA Ark traceroute dataset [5]. The team noted that 55% of the routers and 64% of the servers had addresses with non-zero values in only the least significant 8 bits or the most significant 4 bits of the subnet identifier. Furthermore, 80% of the routers and 22% of the servers had addresses with non-zero values in only the least significant 16 bits of the interface identifier.

Plonka and Berger [25] developed an address visualization technique called Multi-Resolution Aggregate plots. The technique involves analyzing a set of addresses to produce a novel metric that quantifies how relevant each portion of an address is to grouping addresses together into dense address space regions. The authors showed that plots of this metric can be useful for manually discovering network-specific addressing practices. They also introduced a method for identifying dense network prefixes from the given addresses that can be leveraged for scanning. We note that while 6Gen is similarly density-driven, it considers any address space region, beyond just network prefixes.

3.3 Generating Scan Targets

The recursive algorithm developed by Ullrich et al. [31] takes a set of seed addresses and a threshold N as input, and determines

values for all but N bits of an IPv6 address range. The algorithm requires a user-specified address range to start, with at least one bit determined (e.g., with a set value). Then in each level of recursion, the algorithm finds all seed addresses encapsulated by the current range, and identifies which bit and value pair matches the largest number of such seeds. It sets that bit in the current range to the corresponding value, and recurses until only N undetermined bits remain. The addresses in the final range are used as scan targets. The authors evaluated this algorithm against the target prediction methods described in RFC 7707 [17], such as varying the low-order bytes of seed addresses, and against brute-force guessing. Using 10-fold cross validation, where they used a subset of seeds for training and the rest for testing, the authors observed that their algorithm outperformed the other strategies in predicting test addresses. This algorithm shares similarities with 6Gen, as it produces targets from address ranges. However, it can only output ranges of constant size (dependent on the parameter N) and requires an initial range as input, whereas 6Gen automatically can produce multiple address ranges of varying sizes.

Most recently, Foremski et al. [14] introduced Entropy/IP, an algorithm for discovering structure in a set of IPv6 addresses. Entropy/IP identifies adjacent nybbles whose values have similar levels of entropy across the addresses, and groups them together into segments. For each segment, it clusters segment values along several metrics. Entropy/IP utilizes a Bayesian network to model the statistical dependencies between values of different segments. This learned statistical model can then generate target addresses for scanning. Across 10 datasets of router and server seed addresses, the authors employed Entropy/IP on a 1 K random sample of each dataset and generated 1 M targets each, finding 770 K addresses in total were responsive on ICMPv6. 6Gen's design contrasts with Entropy/IP's as it does not aim to learn structure in seed addresses. Instead, it identifies dense regions of similar seeds to generate targets from, constrained by a user-provided probe budget.

4 IPV6 SCANNING CONSIDERATIONS

While increased network speeds and computational power have enabled researchers to scan the full IPv4 address space in minutes using a brute-force approach [12, 18], the IPv6 address space is simply too expansive to ever comprehensively scan. To effectively find active hosts, IPv6 scanners need to be more intelligent and target ranges that are more likely to contain active hosts. In this section, we highlight these challenges and explore considerations for IPv6 target generation and scanning.

4.1 Seed Selection

Algorithms that generate a candidate set of addresses to scan typically consume a set of *seeds*—known addresses that are mined to uncover allocation patterns in order to predict other hosts to scan. Prior work [16] has typically categorized seeds by service provider or data source. We observe that while one approach is to consume a large number of seeds, Internet-wide scans are typically performed horizontally in order to uncover a class of hosts for future analysis. For example, one might seek all hosts that respond on TCP/443 in order to measure a characteristic of HTTPS.

Algorithms may be able to more aptly generate targets for a specific protocol with this additional data. To further the example, it may be that an algorithm is better able to predict the IPs of hosts that support TCP/443 by only considering known hosts that support TCP/80 or TCP/443 (and potentially exclude hosts that support TCP/22). Selecting seeds is an important question as they can greatly influence the targets produced by a TGA.

4.2 Seed Analysis

At the core of IPv6 scanners is an algorithm that attempts to mine structure in known addresses and generate additional candidates to be scanned. We observe that there are two approaches to modeling addresses: dependent and independent. In the dependent model, there are (perhaps hidden) dependencies between seed addresses. Thus, the existence of a particular address in the seed dataset affects the probability that certain other non-seed addresses are also active. Under such a model, TGAs may be able to leverage these dependencies to uncover patterns that can be used for more efficient scans. For example, a simple algorithm could perform a linear regression on seeds (perhaps per network prefix), and predict target addresses based on the linear model.

In the second model, seeds are treated as independent and identically distributed random samples of true active addresses. In this model, the regions of address space with the highest density of active addresses will likely have the highest density of seeds. Unlike the first model, the observation of a specific seed has no implications on other potentially active addresses. Under this model, a natural scanning approach would be to comprehensively scan the most dense regions.

The dependent seeds model can lead to TGAs that may be more efficient, as they attempt to model the seed data to identify patterns. In comparison, it may seem that the independent seeds model results in more naive, less systematic approaches. However, this second model does benefit from simplicity and flexibility, as its prediction methodology can be applied across any network prefix at any time. Computationally, it can also be more efficient as it does not depend on any learning process, which may have to be performed individually for every network prefix.

Relevant to the data modeling is the data source's completeness, a property that is inherently unknown. The independent seeds model may be more appropriate when the collected seeds represent only a small portion of the actually active addresses, as patterns in seeds may not emerge enough to be inferred by a learned model. Conversely, when the seeds represent a significant fraction of addresses, it may be easier to identify and leverage patterns for effective scanning, although in such a case there are inherently fewer addresses left to discover.

Additionally, it is important to design a target generation algorithm that remains effective when operators deviate from standard practices, given that organizations do not always adhere to RFCs. For example, several of the routed prefixes from RouteViews [7] are longer than 64 bits. Although this behavior does not conform with RFC 4291 [20], there is no technical reason prohibiting this practice. A TGA should exhibit the flexibility necessary to predict beyond proposed standard practices.

5 6GEN ALGORITHM

In this section we introduce our target generation algorithm, 6Gen. We present a conceptual overview of the algorithm and the intuition behind our design decisions. Then, we discuss the algorithm in detail and the optimizations that we implemented to operate it on real-world datasets.

5.1 Overview

One natural approach for generating IPv6 scan targets would be to try to reverse engineer organizational IP allocation schemes. However, this approach has several downsides. It may be difficult to determine an allocation pattern from a limited number of seeds. Networks may use multiple assignment policies for the same region of address space (e.g., based on host type). Or, it might be difficult to determine the boundaries between independently managed networks.

Instead, we develop an algorithm that identifies dense regions of similar seeds. We assume that dense regions of seeds are associated with dense regions of active addresses and model seed addresses as independent and identically distributed (IID) random samples of active addresses. This contrasts with approaches that assume dependencies between seeds. We note that although modeling seeds as IID random samples leads to a simpler and more flexible target generation, it may be less efficient because it cannot learn patterns.

6Gen greedily clusters similar seeds into address space regions with high seed density, and outputs the addresses within these regions as scan targets. The algorithm operates iteratively, first identifying the most similar seeds and then clustering together those that form the densest regions, until the total size of the clustered regions grows larger than a user-provided scan budget. In other words, the algorithm allocates portions of its scan budget to "hot spot" regions with many similar seeds, which—under the assumption that seed density is positively correlated with active host density—will maximize the opportunity for finding previously-unknown, active hosts.

Note that 6Gen is not purely density-driven, as it first identifies similar seeds before clustering them into dense regions. It is possible that clustering between more distant seeds results in higher density regions. The motivation behind prioritizing similarity is budget conservation, as clusters of more similar seeds form smaller regions that consume less budget.

5.2 Distance Metric

To cluster similar addresses, we must define an address similarity metric. We use the Hamming distance [19] between the nybble-level representation of addresses and ranges. This metric counts the number of nybble positions that differ between two addresses. To calculate the distance between two regions of IP space, we consider the distance from any wildcard (?) nybble to be zero. For example, the distance between `2001:db8::58` and `2001:db8::51` is one; the distance between `2001:db8::51` `2001:db8::5?` is zero. We note that the Hamming distance also equals the number of nybbles that would become newly dynamic if two addresses were clustered into a range. Intuitively, this indicates the addresses are less similar as a larger region is needed to encapsulate them.

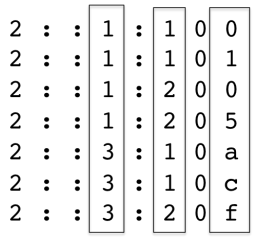


Figure 1: Dynamic nybbles for a cluster of 7 seeds (the cluster's seed set). The cluster has three dynamic nybbles (the other 29 nybble indices have identical values for all addresses) and a range of `2::?:?0?`

We calculate distance at the nybble granularity because addressing schemes are potentially allocated at this specificity and because we observe that bit-level granularity can lead to pairs of addresses that intuitively seem less similar while sharing the same bit-wise Hamming distance. For example, `2::20` and `201::` are 2 bits apart, as are `2::` and `2::3`. However, the second pair intuitively seems more similar and potentially suggests exploration of the range `2::?`.

5.3 Cluster Range Definitions

We use ranges to encapsulate the seeds in a cluster, as shown in Figure 1. While it is natural to represent ranges of IPv6 addresses with nybble wildcards accepting any legal value, we additionally consider nybble wildcards with bounded values. We extend the nybble wildcard notation to denote specific nybble value ranges with the following syntax `[1-2,8-a]`. We describe the tradeoffs between clustering at the nybble wildcard granularity ("loose" clustering) and at nybble specificity ("tight" clustering) in §6.3.

5.4 Algorithm Details

We provide pseudocode for 6Gen in Algorithm 1 and walk through the algorithm in this section.

6Gen accepts a set of input seeds (i.e., known addresses) and internally maintains a set of clusters defined by a *range* (the region of address space that encompasses the seeds in that cluster) and a *seed set* (the seeds that lie within the cluster's range). The algorithm instantiates with a cluster for each seed, containing the single seed address and with a range equal to the seed (as shown in Function *InitClusters*). In each successive iteration, 6Gen calculates the impact of growing each cluster by adding the single closest seed. We note that 6Gen does *not* merge similar clusters. Instead, it allows seeds to belong to multiple clusters, and grows clusters independent of one another.

In each iteration, we first identify the closest seed(s) to each cluster based on Hamming distance (as shown in Function *FindCandidateSeeds*). We consider all non-cluster seeds that are minimally equidistant as *candidate* seeds. For each potential cluster growth by a candidate seed, the cluster range would expand, potentially encapsulating additional seeds beyond the candidate seed (thus further growing the cluster seed set). We identify what the full seed set of the grown cluster would be under the expanded range, and compute the resulting *seed density*—the grown cluster's seed set size divided by its range size (as detailed in Function *GrowCluster*). The

iteration concludes by growing the *one* cluster and candidate seed pair that results in the highest resulting seed density. If there are multiple growth options that result in the same maximum density, we prioritize smaller grown clusters as they consume less budget. Further tiebreaking is performed at random. 6Gen iterates until the sum of cluster range sizes consumes the user-provided probe budget or all seeds belong to a single cluster (as seen in Function *6Gen*). If using the most recently grown cluster exceeds the probe budget, we consume the budget exactly by randomly selecting addresses in the newly grown cluster's range that were not in the cluster's pre-growth range.

We note that the algorithm can result in overlapped clusters because we consider every non-cluster seed for potential growth, and we grow clusters independently. We do not attempt to simply merge partially overlapping clusters because this can result in a significantly less dense supercluster. Instead, we allow clusters to partially overlap. We do delete any cluster that becomes fully encapsulated by another, by comparing a grown cluster's new range with all other cluster ranges to find any that are strict subsets. To ensure that we do not double-count addresses against the budget, we uniquely track all addresses that would be generated by the clusters.

5.5 Optimizations

As described in the previous section, 6Gen is conceptually simple but computationally expensive. For example, the naive implementation involves iterating over all clusters, and for each cluster, iterating over all external seeds to find candidates. As clusters grow independently, we can easily parallelize cluster growth computation. We can further reduce the computational complexity with a couple optimizations, which we describe here.

In each iteration, 6Gen finds candidate seeds to grow each cluster with and computes the potential change in density for each cluster growth. We note that only one cluster is changed per iteration and that because clusters grow independently, all other clusters remain unchanged and their best growths can be cached between iterations. This reduces the naive implementation's runtime by a factor of $O(N)$ for N seeds.

It is also possible to optimize finding the seeds that need to be added to a cluster when it expands. We store all seeds in a *nybble tree*—a 16-ary tree where each level in the tree represents a nybble position and branching corresponds to that position's nybble value. This allows us to quickly iterate over the seeds that fall within a given range instead of iterating over all seeds. The nybble tree also allows reconstructing a cluster's seed set given its range. As a space optimization, we only store a cluster's range and seed set size, instead of the seed set itself.

5.6 Performance Evaluation

To measure 6Gen's performance, we implemented a prototype of 6Gen in 3.6K lines of C++ code, using OpenMP [1] for multithreading support. We evaluated it against a set of 2.96M seeds (described in §6.1) on a Linux server with dual 10-core Intel Xeon E5-2650 (2.30 GHz) CPUs and 256 GB of memory. We grouped the seeds by routed network prefix and separately ran 6Gen on each of

Algorithm 1 6Gen pseudocode, simplified to illustrate conceptual steps, and without optimizations (see §5.5).

```
clusterList = []

function InitClusters(seedList)
    for seed in seedList do
        cluster = new Cluster()
        cluster.addSeedUpdateRange(seed)
        clusterList.add(cluster)

function FindCandidateSeeds(cluster, seedList) ▷ Computes the minimum Hamming distance between cluster.range and all seeds in seedList not already in cluster, and returns the list of seeds that are this minimum distance away.

function GrowCluster(seedList)  ▷ Consider growing all clusters by candidate seeds, and select the growth resulting in the highest seed density and smallest cluster range size.
    maxDensity, maxIndex, maxRangeSize = 0, 0, Infinity
    maxCluster = None
    for index in [0, ..., clusterList.length() − 1] do
        cluster = clusterList[index]
        candidateSeeds = FindCandidateSeeds(cluster, seedList)
        for seed in candidateSeeds do
            tmpCluster = cluster.copy()
            tmpCluster.addSeedUpdateRange(seed)
            for otherSeed in candidateSeeds do
                if otherSeed in tmpCluster.range then
                    tmpCluster.addSeedUpdateRange(otherSeed)
    ▷ Does not further change the range.
            newDensity = tmpCluster.seedSet.size() / tmpCluster.range.size()
            if (newDensity > maxDensity) or (newDensity == maxDensity and tmpCluster.range.size() < maxRangeSize) then
                maxDensity, maxIndex = newDensity, index
                maxRangeSize = tmpCluster.range.size()
                maxCluster = tmpCluster
    return (maxIndex, maxCluster)

function 6Gen(seedList, budgetLimit)  ▷ Grow clusters until the sum of cluster range sizes exceeds the budget. For simplicity, we elide here details about handling cluster overlap and final cluster growth sampling to use up the budget exactly.
    InitCluster(seedList)
    budgetUsed = 0
    while True do
        grownIndex, grownCluster = GrowCluster(seedList)
        oldRangeSize = clusterList[grownIndex].range.size()
        newRangeSize = grownCluster.range.size()
        budgetCost = newRangeSize − oldRangeSize
        budgetUsed = budgetUsed + budgetCost
        if (budgetUsed ≤ budgetLimit) and (seedList.size() > grownCluster.seedSet.size()) then
            clusterList[grownIndex] = grownCluster
        else
            return clusterList
```

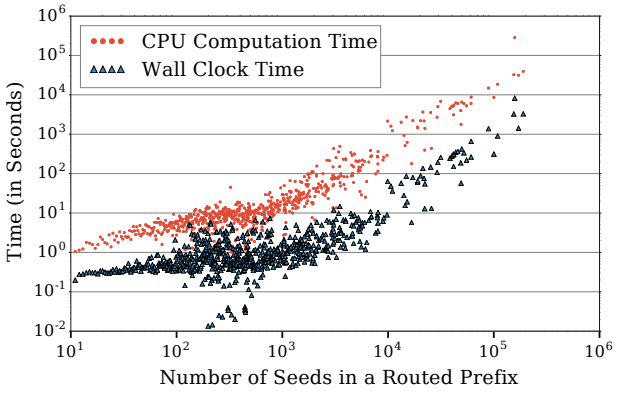


Figure 2: Median execution time of running 6Gen on routed prefixes with differing numbers of seeds. Routed prefixes with fewer than 10 seeds are omitted as they all required less than one second to process.

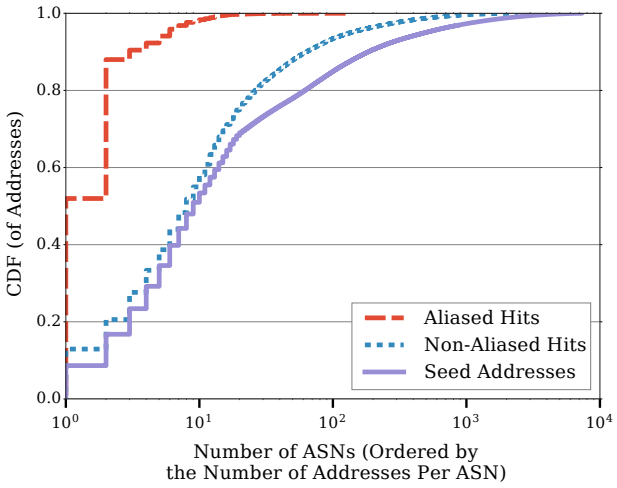


Figure 3: The distribution of seed addresses, filtered aliased hits, and hits after dealiasing across ASNs. We collected hits from active scans of targets generated by 6Gen run with a budget of 1 M probes per routed prefix.

the 10,038 prefixes. We note that while we could parallelize execution across different prefixes, dedicating all CPUs to running 6Gen on each prefix linearly was most performant.

We show the median runtime for differing number of seeds in Figure 2. CPU compute time divides the total number of CPU cycles spent executing 6Gen across all CPU cores with the CPU clock rate. Wall clock time measures the actual runtime of 6Gen. Naturally, as the number of seeds increases, the runtime increases. However, we note that runtime is heavily dependent on the set of seeds—prefixes with complex address structures can involve more clustering. For example, while the prefix with the second most number of seeds (157K) required two hours of wall clock time, the largest prefix (189K seeds) required only 55 minutes. Running 6Gen on the entire seed dataset required 9 hours and a maximum 3.3 GB of memory.

6 EVALUATION

In this section, we explore 6Gen's performance under several operating conditions to better understand its utility and the scan targets it generates. For each experiment, we evaluate 6Gen by running it against the IPv6 addresses in the Rapid7 Forward DNS ANY dataset grouped by routed block (§6.1) and then scanning generated addresses on TCP/80 using the IPv6 version of ZMap from Gasser et al. [16]. We note that we do not address how to best allocate probe budget across networks, and instead scan networks independently with a static probe budget in our analysis.

For each variable we tested, we sent approximately 5.8 B probes at 100 K packets per second. We randomized the order of the destination hosts and ran each scan in serial to avoid overloading networks. We follow the guidelines outlined by Durumeric et al. [12] for ethical scanning. In particular, we signal the benign intent of our scans through WHOIS and reverse DNS records, and provide project details and point of contact on a website hosted on each scanning host. We respect all scanning opt-out requests, blacklisting them from any further scans.

6.1 Seed Collection

We collected seed addresses by extracting AAAA records from the Rapid7 Forward DNS ANY dataset [26]. The dataset consists of DNS responses to DNS ANY requests for fully-qualified domain names seen in Rapid7's Project Sonar scans [27]. We specifically used the August 11, 2017 snapshot that contained 2.96 M unique IPv6 addresses. We grouped seeds by BGP origin routed prefix. The seeds resided in 10,038 routed prefixes in 7,350 ASes. In Figure 3, we observe that the distribution of seed addresses is not heavily skewed towards any particular ASes; we show the top ten ASes in Table 1a.

6.2 Dealiasing

During our evaluation, we found that address aliasing has a profound effect on IPv6 address generation and that large-scale IPv6 aliasing is more common than previously discussed. While prevalence varied slightly across different scan parameters, we found that more than 98% of active addresses belonged to large aliased regions in many experiments. In one example, all addresses in a single /56 prefix belonging to Akamai responded to probes on TCP/80. In this section, we discuss how we filter out these prefixes from our analysis.

Existing techniques to dealias IPv6 addresses focus on routers [22–24] or identifying IPv4 and IPv6 addresses that belong to the same dual-stack host [29]. Although these approaches might generalize, they are limited by the features supported on the investigated hosts. For example, SpeedTrap [22] is an IPv6 alias resolution method that uses IP fragmentation information as a side-channel to detect identical hosts at multiple addresses. However, the authors reported only being able to use this side-channel for approximately a third of routers they measured. Scheitle et. al. [29] used variable clock skew to associate IPv4 and IPv6 address pairs with

AS Name	ASN	% Seeds
Linode	63949	8.6%
Amazon	16509	8.1%
HostEurope	20773	6.6%
DTAG ISP	3320	5.8%
home.pl	12824	5.4%
Masterhost	25532	5.2%
Hurricane	6939	4.4%
Cloudflare	13335	3.7%
TuxBox	47490	3.0%
OneAndOne	8560	2.4%

(a) Seed Addresses

AS Name	ASN	% Hits
Akamai	20940	52.0%
Amazon	16509	36.0%
CenturyLink	209	2.5%
GTT	3257	1.8%
Fastly	54113	1.8%
Google	15169	1.8%
Masterhost	25532	1.0%
Cloudflare	13335	0.8%
XO Comms	2828	0.4%
Lidero	13189	0.3%

(b) Aliased Hits

AS Name	ASN	% Hits
Amazon	14618	12.9%
Amazon	16509	7.7%
OVH	16276	7.1%
Hetzner	24940	5.7%
HostEurope	20773	5.3%
RH-TEC	25560	5.3%
Globe	25234	4.3%
GoDaddy	26496	3.5%
Uvensys	58010	3.2%
DigitalOcean	14061	3.1%

(c) Non-Aliased Hits

Table 1: The top 10 ASes and the percent of addresses within each, for our seed addresses, aliased hits, and hits after dealiasing.

dual-stack hosts. They found that 31% of responsive IPv6 addresses did not offer the TCP Timestamp option necessary to assess clock properties. Furthermore, the prior works have only demonstrated that these methods scale to several thousand addresses, which is orders of magnitude smaller than the large aliased networks we have observed.

As a best-effort attempt at identifying aliased regions, we performed active probing of /96 address prefixes that contained responsive scan targets (e.g., hits). For each prefix, we randomly generated three addresses and sent three TCP SYN probes on port 80 to each address. If all three addresses within a prefix responded to at least one of the probes, we considered the prefix aliased. Given the size of a /96 prefix (2^{32} addresses), the probability of randomly selecting three responsive addresses in a non-aliased prefix is negligible. Even if a prefix is non-aliased and has a million responsive addresses, the likelihood of our experiment falsely flagging aliasing is less than 10^{-10}. Our method will reliably detect prefix aliasing at the /96 (or larger/shorter) granularity, modulo extensive network outages. We chose to operate at the /96 granularity as it is a relatively small prefix but the number of probe packets required proved manageable.

In total, our responsive target addresses resided in 10.2 M /96 prefixes, of which 10.0 M (98%) were aliased on TCP/80. These /96 prefixes corresponded to 205 routed prefixes in 138 ASes. After filtering out hits in aliased /96 prefixes, we additionally clustered our remaining hits by AS, and manually investigated the top 10 ASes for aliasing at a granularity smaller than /96 prefixes. We found that the top two ASes (Cloudflare (13335) and Mittwald (15817)) were aliased at a /112 prefix granularity, while the remaining 8 ASes did not exhibit signs of large-scale aliasing. We further excluded those two ASes (37% of remaining hits).

In general, IP aliasing is heavily concentrated in a small number of ASes. Of the 7,421 ASes in our seed and hit address datasets, only 140 (1.9%) ASes exhibited extensive aliasing. Akamai accounted for over half of the aliased hits and Amazon accounted for over one third (Table 1b). This skew of aliased hits towards a few ASes is also visible in Figure 3, as nearly 95% of all aliased hits were localized in five ASes. For these networks, 6Gen did find large numbers of aliased hits that may not meaningfully represent distinct hosts or network services—a limitation of the algorithm's design. However, as aliasing is concentrated to relatively few networks, 6Gen can still be effective on other networks. Unless noted otherwise, we count removed /96s as zero hits to provide as close as possible to a lower bound on the number of addresses we found in the remainder of the section.

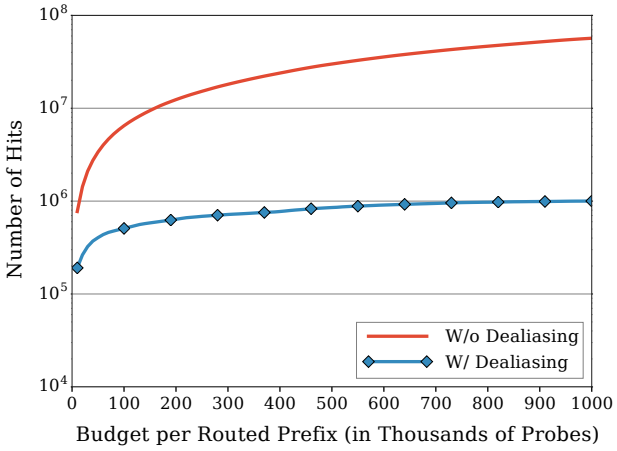
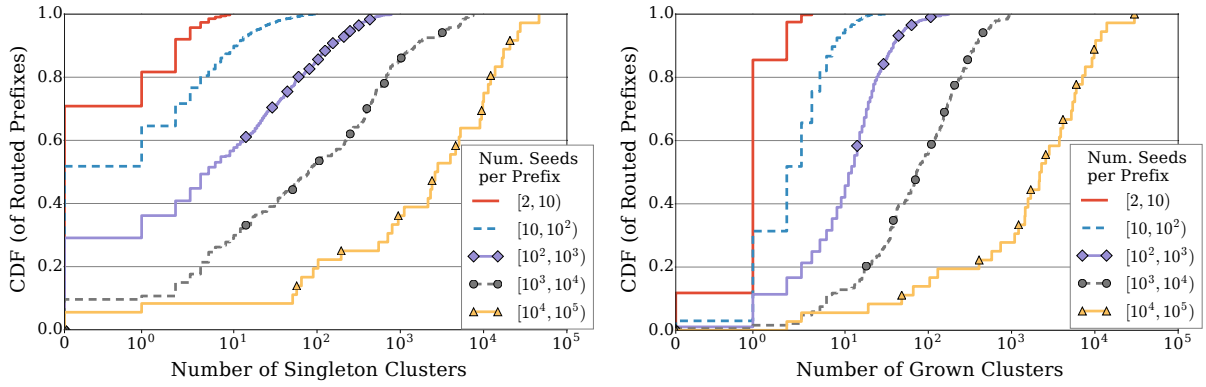


Figure 4: The number of TCP/80 hits for targets generated by 6Gen, for varying budgets.

6.3 Design Decision: Tight versus Loose Ranges

As discussed in §5.3, 6Gen can generate clusters at the nybble specificity granularity ("tight" ranges) or can use the fully undetermined nybble wildcard granularity ("loose" ranges). A loose range emphasizes more extensive exploration of early-formed clusters with higher densities, as these clusters are fully expanded by the loose range, consuming more of the budget early on. In contrast, a tight range results in exploring more or larger clusters, as each cluster's range is tightened. To understand the tradeoff, we compared the raw number of hits found using both approaches. With a budget of 1 M probes per routed prefix, 6Gen with loose ranges found 56.7 M hits on TCP/80 whereas tight ranges resulted in 55.9 M hits. The pattern held after filtering out aliased hits with 1.0 M versus 973 K hits and we use loose ranges in our later experiments.



(a) Singleton Clusters **(b) Grown Clusters**

Figure 5: CDFs of the number of singleton and grown clusters that 6Gen outputs for routed prefixes with varying number of seeds. We bucket routed prefixes by the number of seeds, and plot a curve for each bucket. Note we elide routed prefixes with more than 100 K seeds as the population size is too small for a meaningful distribution.

6.4 Selecting the Budget Parameter

In our analysis, we treat destination networks independently and do not consider how to share scan budget across networks. However, even within a single network, scan budget can drastically change the patterns that are uncovered. To explore the effect of the budget size, we evaluate 6Gen's predictions on TCP/80 at varying budgets. From Figure 4, we observe a plateau in the increase in the number of dealiased hits as the budget approaches 1 M probes per routed prefix. This suggests that for many prefixes, meaningful clustering has halted as the last grown clusters were not useful for generating hits. Increasing the budget further may have diminishing returns, thus we chose to operate by default with a budget of 1 M probes for each routed prefix.

6.5 6Gen's Clusters

We find that 6Gen grows at least one cluster for the vast majority of routed prefixes: only 3% of routed prefixes with 10 or more seeds had no grown clusters and likewise for only 12% of prefixes with 2–10 seeds (Figure 5b). There are also few clusters (both singleton and grown) relative to the number of seeds in each routed prefix. For example, all seeds were part of a grown cluster for half of the routed prefixes with 10–100 seeds and 80% of those routed prefixes had five or fewer singletons. Half of the prefixes with 100–1000 seeds had 10 or less grown clusters. In other words, 6Gen clustered the majority of seeds together and formed only a small number of clusters rather than a large number of small clusters.

We also characterize our clusters by the location of dynamic nybbles. As can be seen in Figure 6, there are two modes in the frequency of dynamic nybbles across routed prefixes. The first mode is from the 9th to the 16th nybble. This behavior is likely due to addresses conforming to RFC 2460 [9], which specifies using the first 64 bits of the address (or the first 16 nybbles) as a network identifier. The second mode is after the 29th nybble. This behavior is likely explained by RFC 7707 [17], which describes that using the least significant bits of the address (e.g., the highest nybbles indexes) is common practice.

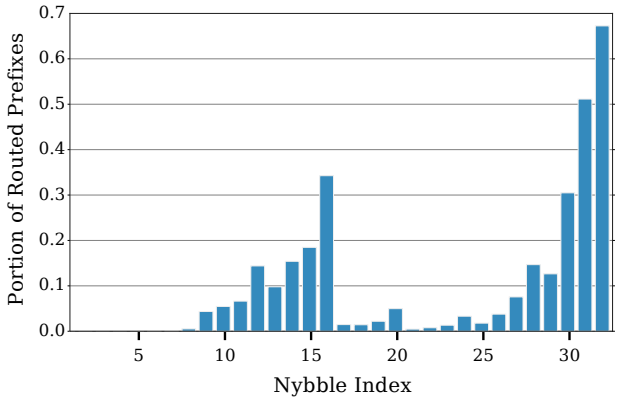


Figure 6: For each nybble (with an index between 1 and 32), we plot the portion of routed prefixes that have any cluster ranges with that nybble dynamic.

6.6 Address Analysis

6Gen generated 5.8 B scan targets when run with a budget of 1 M probes per routed prefix. Of those, 56.7 M responded on TCP/80. However, as discussed in §6.2, the vast majority (98%) are located in extensively aliased regions. Here we explore the 1.0 M responsive addresses 6Gen discovered in non-aliased regions.

6Gen was able to find meaningful targets in 2,840 routed prefixes associated with 2,368 ASes. This is 28% of the routed prefixes and 32% of the ASes that contained non-aliased seeds in our input set. We show the distribution of hits across ASes in Figure 3 and list the top ten ASes in Table 1c. We observe that the dealiased hits

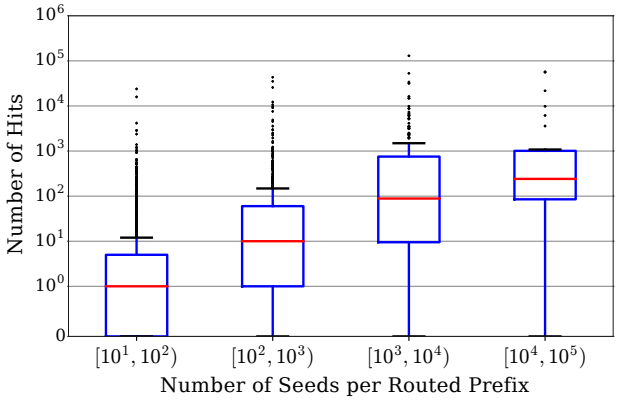


Figure 7: The distribution of TCP/80 hits per routed prefix, bucketed by the number of seeds in a prefix. We ran 6Gen with a budget of 1 M probes per prefix. We excluded prefixes with less than 10 seeds as over 90% had zero hits, and prefixes with more than 100 K seeds as the population size was too small for a meaningful distribution.

are slightly more skewed towards fewer ASes than the input seed dataset. In particular, the top two ASes both belong to Amazon and accounted for over 20% of our aliased hits. It is interesting to note that Amazon AS-16509 contained both aliased and dealiased hits (recall we manually inspected that certain Amazon subnets were not fully responsive and aliased). Thus, ASes may vary their aliasing policies across subnets and AS-level filtering of aliasing is too coarse.

Figure 7 shows the distribution of the number of hits per routed prefix bucketed by number of seeds. As indicated by the median values for each bucket, we are able to find active addresses for a majority of prefixes with more than 10 seeds. We observe a positive correlation between the number of hits and the number of seeds per routed prefix. One potential explanation is that 6Gen is not discovering new addresses, but rather address churn. Hosts at now-inactive seed addresses may have moved to new addresses, and 6Gen is only rediscovering these hosts. To assess this explanation, we subtract the number of inactive seeds from the number of hits for each routed prefix. For a quarter of prefixes, the difference was positive, demonstrating 6Gen's utility as it must have found new addresses that cannot be due to network churn. For the remaining prefixes, it is unclear whether 6Gen found churned addresses or simply discovered fewer addresses than inactive seeds.

6.7 Seed Sensitivity

As 6Gen selects address regions to scan based on seeds, the seed characteristics can impact 6Gen's performance. Here we explore how the input seeds affect 6Gen.

6.7.1 Type of Host. Our seed dataset contains IPv6 addresses from DNS records that correspond with a diverse set of hosts (e.g., DNS, web, and SMTP servers). As an initial exploration of whether seeds of one host type are suitable for discovering other types of hosts, we executed 6Gen on only DNS name server seeds. We identified name server hosts as those addresses in DNS NS records. We fed these 61 K seeds into 6Gen, ran it with a budget of 1 M probes per routed prefix, and scanned the outputted predictions on TCP/80.

	W/o Dealiasing		W/ Dealiasing	
Downsampling Level	Num. Hits	% vs All	Num. Hits	% vs All
1%	758 K	1.3%	225 K	22.5%
10%	13.3 M	23.5%	713 K	71.3%
25%	27.3 M	48.2%	825 K	82.5%
100%	56.7 M	100.0%	1.0 M	100.0%

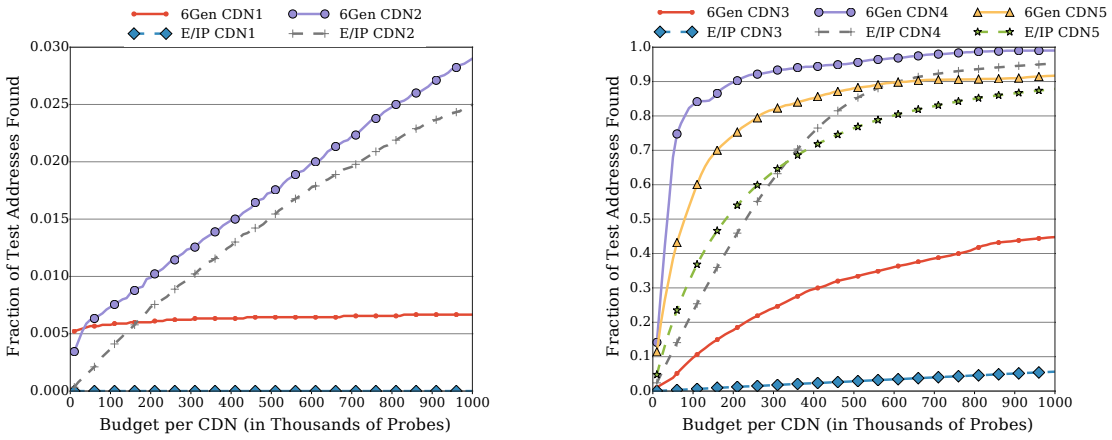
Table 2: For each downsampling level, both before and after dealiasing, we list the number of hits 6Gen discovered and what percentage that number is compared to the number of hits 6Gen found using all seeds. In all experiments, the budget was 1 M probes per routed prefix.

In total, we found 1.2M responsive addresses, of which 308K were in non-aliased regions. While the number of discovered addresses is drastically smaller when only using name server addresses, the input seed dataset is also significantly reduced. Compared to the name server seed set, 6Gen was able to find five times as many non-aliased addresses and 19 times as many hits overall, indicating that using seeds of one host type may be fruitful still in discovering other hosts. However, we acknowledge this is an initial exploration, and some host types may prove less useful as seeds.

6.7.2 Downsampling Seeds. As 6Gen executes faster with fewer seeds, one may wish to downsample a seed dataset to improve performance. To investigate the effect of downsampling, we explore 6Gen's performance when run on 1%, 10%, and 25% of our full seed dataset. For each downsampled seed input, we executed 6Gen with a budget of 1 M probes per routed prefix. We can see in Table 2 that 6Gen discovers fewer hits when given fewer seeds. However, the decrease is not commensurate with the downsampling rate. For example, compared to 6Gen with the full seed datset, 6Gen with a 10% sample of seeds still finds 24% as many hits before dealiasing and 71% after. At initial glance, 6Gen is quite robust to seed downsampling, and still discovers a large number of hits even when operating on a significantly reduced input dataset.

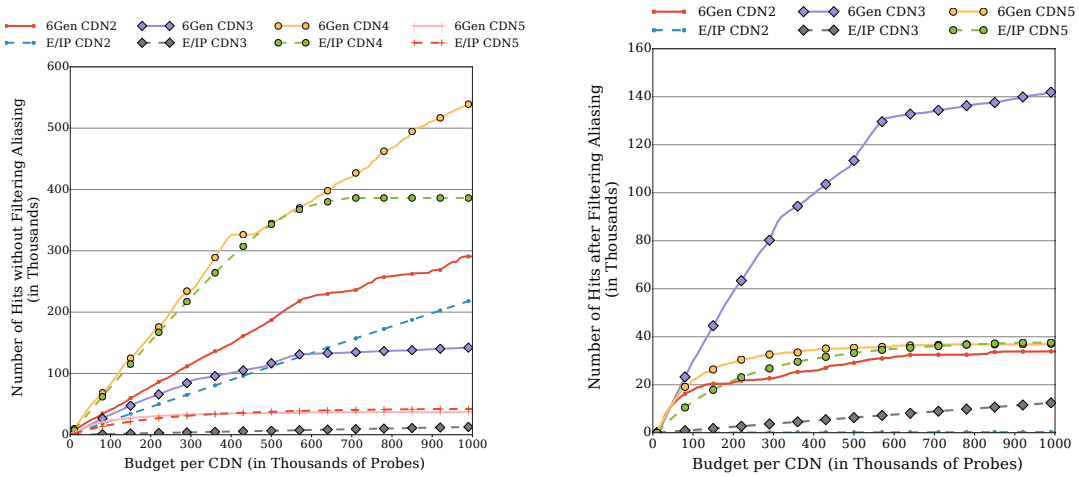
7 COMPARISON TO ENTROPY/IP

While Entropy/IP [14] is foremost an analysis tool for identifying patterns in IPv6 addresses, we can use it to predict new addresses that align with found patterns. As one of the state-of-the-art target generation algorithms, Entropy/IP serves as a comparison point for 6Gen. We compare the algorithms through two evaluations: a train-and-test experiment where we train on a subset of seeds and evaluate what fraction of the remaining seeds are predicted, and an active scan where we compare the number of hits predicted. We conducted our evaluations using a copy of Entropy/IP provided by its authors. Our seed dataset was likewise obtained from the Entropy/IP authors, containing a random sample of 10 K addresses collected from five content distribution networks (labeled as CDNs 1–5) used in the original Entropy/IP evaluation [14].



(a) CDNs 1 and 2 **(b) CDNs 3, 4, and 5**

Figure 8: The fraction of addresses in the five CDN test sets found by 6Gen and Entropy/IP during the train-and-test evaluation, for varying budgets.



(a) Without Filtering for Aliasing **(b) Filtering for Aliasing**

Figure 9: The number of TCP/80 hits for targets generated in five CDN networks by 6Gen and Entropy/IP, for varying budgets, both with and without filtering for aliasing. Both 6Gen and Entropy/IP performed poorly on CDN 1 and did not find a significant number of hosts, and hence we elided CDN 1 from the graphs. We removed CDN 4 from the post-filtering graph as it extensively aliased.

7.1 Train-and-Test Evaluation

For each of the five CDN datasets, we split the addresses into 10 groups at random (each with 1 K addresses). We then ran both 6Gen and Entropy/IP on each 10% sample and validated against the remaining 90%—a form of inverse k-fold validation. We can see in Figure 8 that 6Gen found a similar or larger number of test addresses than Entropy/IP for all networks. Excluding CDN 1, where Entropy/IP failed to predict any test addresses, 6Gen with a budget of 1 M probes predicted 1.04–7.95 times more addresses than Entropy/IP (with the same budget). We note that both algorithms failed to discover the majority of test addresses for CDNs 1 and 2. For CDNs 4 and 5, both algorithms predicted over 88% of the test addresses. Notably, 6Gen was able to find over 99% of the test addresses for CDN 4.

We observe that 6Gen is significantly more effective than Entropy/IP when operated with a limited budget. However, Entropy/IP may output a greater diversity of address predictions than 6Gen at lower budgets, as Entropy/IP samples targets from multiple address regions without prioritizing density. This behavior also results in the smoother performance curves for Entropy/IP compared to 6Gen, as a small increase in the probe budget may allow 6Gen to greedily incorporate a new dense region, causing a jump in address discoveries. This difference in how the probe budget affects each algorithm is noteworthy. Entropy/IP uses the budget only to adjust

the number of targets generated, while 6Gen also uses the budget to determine the regions of address space it selects. For Entropy/IP's original goal of address pattern analysis, the concept of a budget was not necessary to make decisions. However, modifying the algorithm to specifically cater to scanning purposes, such as through factoring in a budget when identifying probable address patterns, may enhance its applicability to Internet-wide scanning.

7.2 Active Scans Comparison

As a second evaluation, we conducted active TCP/80 SYN scans of each algorithm's predictions. Similar to the previous experiment, we find that 6Gen performed nearly equivalent to or better than Entropy/IP for all CDN networks in terms of both total and dealiased hits (Figure 9). For filtered hits, with the budget set to 1 M probes, 6Gen found 0.99–134.48 times more hits than Entropy/IP. Entropy/IP and 6Gen found a similar number of hits in CDN 5, and neither algorithm received a significant number of responses in CDN 1. However, 6Gen greatly outperformed Entropy/IP in CDNs 2, 3, and 4. Note that we removed CDN 4 when considering filtered hits as it extensively aliased. While the number of hits discovered serves as a basic metric for evaluating and comparing 6Gen's and Entropy/IP's ability to produce promising address targets, further evaluation is required to understand the differences between the algorithms.

8 FUTURE WORK

In this paper, we introduced 6Gen and evaluated its effectiveness. 6Gen shows promise and discovers millions of previously-unknown addresses. However, we also uncovered widespread aliasing. Here, we discuss several future directions that the research community needs to consider in order to understand IPv6 address allocation and improve IPv6 scanning.

IPv6 Dealiasing. Our investigation of IPv6 aliasing in §6.2 uncovered extensive aliasing in a number of large networks. In total, 98% of the /96 prefixes we discovered responsive targets in exhibited aliasing on TCP/80. Filtering addresses in aliased regions significantly reduced our hits, resulting in a different characterization of 6Gen's performance. Future efforts in IPv6 target generation and scanning must factor in aliasing when quantifying performance. We note that prior work on IPv6 target generation has not factored in aliasing [14, 31] and their findings may not represent the number of distinct hosts that each algorithm can discover.

Detecting IPv6 aliasing is itself a significant challenge. In this work, we used a simple approach of detecting aliasing at the /96 prefix granularity, but this method naturally has limitations (such as identifying smaller-scale aliasing). Prior efforts to develop effective IPv6 dealiasing have shown promise, but are limited by features enabled on the target host. Further exploration is needed to develop scalable and reliable alias resolution, to better understand the topology of the IPv6 Internet, and to more accurately characterize IPv6 scanner performance.

Deeper Exploration of Target Generation Algorithms. Our initial evaluation of TGAs focused on the ability of 6Gen and Entropy/IP to discover hosts responsive on TCP/80 using seeds gathered from DNS records. We must more deeply understand how each algorithm is affected by different network characteristics, the types of hosts found by each, and the tradeoffs between the two. Further exploration of other network services and seed address inputs will also help shed light on the operating characteristics of these algorithms. For example, how do 6Gen and Entropy/IP perform when seeking SMTP or SSH servers? Do their predictions differ when run on only active seeds (seeds freshly probed for responsiveness), or on seeds that are first dealiased?

We also lack a deep understanding of when each algorithm is effective and why. Are there certain types of address assignment patterns that an algorithm is not amenable to discovering? Are there optimizations we can apply to better improve probing efficiency, based on common patterns? For example, 6Gen's budget use is suboptimal when a network employs a clear address assignment pattern, as it fails to leverage learnable information for budget conservation.

There is still much to consider about 6Gen's operations. For example, we employed 6Gen with an identical budget for all routed prefixes. However, it might be natural to allocate budgets differently for various routed prefixes. For example, a routed prefix's budget could be dependent on the number of seeds within, or the size of the prefix itself. This may heavily skew the target generation towards denser networks though, trading off diversity for number of active addresses found. What the most suitable budget allocation policy is, and how this differs based on target generation goals, is still an open question.

Scanner Integration. In this paper, we have considered the TGA as a distinct module that produces targets fed to a network scanner. However, tight integration between the target generation and the scanning processes should allow for more effective scanning. The target generation could provide the initial regions of address space to begin exploring. As a scan progresses, the results can be fed back to the generation algorithm, allowing it to dynamically adapt its predictions based on the additional information. For example, we can early terminate scanning of a region originally predicted as promising but that has yielded few discovered hosts. Similarly, we can test regions that have high hit rates for aliasing, and halt scanning if aliasing is detected. These measures would allow the scanner to reallocate budget to networks that prove promising in reality. Ultimately, such integration allows the target generation to behave more intelligently, as it is has a feedback loop to better inform its decisions.

9 CONCLUSION

In this work, we explored the basic challenge of generating promising IPv6 addresses to scan. We presented 6Gen, a new target generation algorithm that executes on a set of input seed addresses, given a user-supplied probe budget. 6Gen operates under the assumption that address space regions with high densities of similar seeds correlate with those rich with active hosts, and seeks out targets in these regions.

When evaluated on real-world seed datasets containing addresses in thousands of network prefixes, 6Gen showed promise and discovered over 55 M responsive addresses (with a budget of 1 M probes per routed prefix). However, upon inspecting these hits, we also uncovered a small number of networks exhibiting large-scale IP aliasing. Over 98% of our original hits were in aliased networks,

highlighting the extent that IP aliasing can affect performance metrics for target generation. When filtering out aliased hits, 6Gen was still able to find over a million new addresses. Our comparison with Entropy/IP [14]—the current state of the art TGA—likewise demonstrated 6Gen's utility. On the five seed datasets used in the original Entropy/IP evaluation, 6Gen recovered between 1–8 times as many addresses during a train-and-test evaluation.

Our exploration of the IPv6 target generation space has highlighted important areas for future investigation. In particular, IPv6 aliasing will be a network feature that future TGAs and network scanners must contend with. In addition, target generation must become more intelligent, perhaps through tightly integrating with the network scanner. With an IPv6 scanner that can obtain a significant degree of global visibility into the IPv6 address space, researchers can begin to tackle the myriad of security and measurement questions that have been explored for IPv4.

10 ACKNOWLEDGEMENTS

We thank Arthur Berger, David Plonka, and Paul Pearce for extensive discussion and feedback. We are grateful to Jed Crandall and the University of New Mexico network administrators for providing us with infrastructure to conduct Internet-scale IPv6 scanning, and supporting our efforts. We additionally appreciate Jon Hart's assistance in obtaining and understanding our seed dataset of IPv6 addresses. This work was supported in part by CNS-1111672, CNS-1237265, and CNS-1518921. The opinions expressed in this paper do not necessarily reflect those of the research sponsors.

REFERENCES

[1] OpenMP. http://www.openmp.org.
[2] Alexa. Top 1,000,000 Sites. http://www.alexa.com/topsites.
[3] S. M. Bellovin, B. Cheswick, and A. Keromytis. Worm Propagation Strategies in an IPv6 Internet. *;login: The USENIX Magazine*, 2006.
[4] J. W. Bos, J. A. Halderman, N. Heninger, J. Moore, M. Naehrig, and E. Wustrow. Elliptic Curve Cryptography in Practice. In *International Conference on Financial Cryptography and Data Security*, 2014.
[5] Center for Applied Internet Data Analysis. Archipelago (Ark) Measurement Infrastructure. http://www.caida.org/projects/ark/.
[6] Center for Applied Internet Data Analysis. CAIDA IPv6 DNS Names Dataset. http://www.caida.org/data/active/ipv6dnsnamesdataset.xml.
[7] Center for Applied Internet Data Analysis. RouteViews Prefix to AS Mappings Dataset. http://www.caida.org/data/routing/routeviews-prefix2as.xml.
[8] J. Czyz, M. Luckie, M. Allman, and M. Bailey. Don't Forget to Lock the Back Door! A Characterization of IPv6 Network Security Policy. In *Network and Distributed Systems Security (NDSS)*, 2016.
[9] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Standards Track), 1998.
[10] R. Droms, J. Bound, B. Volz, T. Lemon, C. Perkins, and M. Carney. Dynamic Host Configuration Protocol for IPv6 (DHCPv6). RFC 3315 (Standards Track), 2003.
[11] Z. Durumeric, F. Li, J. Kasten, N. Weaver, J. Amann, J. Beekman, M. Payer, D. Adrian, V. Paxson, M. Bailey, and J. A. Halderman. The Matter of Heartbleed. In *ACM Internet Measurement Conference (IMC)*, 2014.
[12] Z. Durumeric, E. Wustrow, and J. A. Halderman. ZMap: Fast Internet-Wide Scanning and Its Security Applications. In *Usenix Security*, 2013.
[13] T. Fiebig, K. Borgolte, S. Hao, C. Kruegel, and G. Vigna. Something From Nothing (There): Collecting Global IPv6 Datasets From DNS. In *Passive and Active Measurement (PAM)*, 2017.
[14] P. Foremski, D. Plonka, and A. Berger. Entropy/IP: Uncovering Structure in IPv6 Addresses. In *ACM Internet Measurement Conference (IMC)*, 2016.
[15] O. Gasser, Q. Scheitle, S. Gebhard, and G. Carle. IPv6 Hitlist Collection. http://www.net.in.tum.de/pub/ipv6-hitlist/.
[16] O. Gasser, Q. Scheitle, S. Gebhard, and G. Carle. Scanning the IPv6 Internet: Towards a Comprehensive Hitlist. *Network Traffic Measurement and Analysis Conference (TMA)*, 2016.
[17] F. Gont and T. Chown. Network Reconnaissance in IPv6 Networks. RFC 7707 (Informational), 2016.
[18] R. D. Graham. MASSCAN: Mass IP Port Scanner. https://github.com/robertdavidgraham/masscan.
[19] R. W. Hamming. Error Detecting and Error Correcting Codes. *Bell Labs Technical Journal*, 1950.
[20] R. Hinden and S. Deering. IP Version 6 Addressing Architecture. RFC 4291 (Draft Standard), 2006.
[21] F. Li, Z. Durumeric, J. Czyz, M. Karami, M. Bailey, D. McCoy, S. Savage, and V. Paxson. You've Got Vulnerability: Exploring Effective Vulnerability Notifications. In *USENIX Security Symposium*, 2016.
[22] M. Luckie, R. Beverly, W. Brinkmeyer, et al. Speedtrap: Internet-Scale IPv6 Alias Resolution. In *ACM Internet Measurement Conference (IMC)*, 2013.
[23] P. Marchetta, V. Persico, and A. Pescapè. Pythia: Yet Another Active Probing Technique for Alias Resolution. In *ACM Emerging Networking Experiments and Technologies (CoNEXT)*, 2013.
[24] R. Padmanabhan, Z. Li, D. Levin, and N. Spring. UAv6: Alias Resolution in IPv6 Using Unused Addresses. In *Passive and Active Measurement (PAM)*, 2015.
[25] D. Plonka and A. Berger. Temporal and Spatial Classification of Active IPv6 Addresses. In *ACM Internet Measurement Conference (IMC)*, 2015.
[26] Rapid7. DNS Records (ANY). https://scans.io/study/sonar.fdns.
[27] Rapid7. Project Sonar. https://sonar.labs.rapid7.com/.
[28] Rapid7. Reverse DNS. https://scans.io/study/sonar.rdns.
[29] Q. Scheitle, O. Gasser, M. Rouhi, and G. Carle. Large-scale Classification of IPv6-IPv4 Siblings with Variable Clock Skew. In *Network Traffic Measurement and Analysis Conference (TMA)*, 2017.
[30] S. Thomson, T. Narten, and T. Jinmei. IPv6 Stateless Address Autoconfiguration. RFC 4862 (Draft Standard), 2007.
[31] J. Ullrich, P. Kieseberg, K. Krombholz, and E. Weippl. On Reconnaissance with IPv6: A Pattern-Based Scanning Approach. In *Availability, Reliability and Security (ARES)*, 2015.