AddrMiner: A Comprehensive Global Active IPv6 Address Discovery System

Guanglei Song, Jiahai Yang, Lin He, Zhiliang Wang, *Institute for Network Sciences and Cyberspace, BNRist, Tsinghua University and Quan Cheng Laboratory, Jinan, Shandong, China;* Guo Li and Chenxin Duan, *Institute for Network Sciences and Cyberspace, BNRist, Tsinghua University;* Yaozhong Liu, *Tsinghua University;* Zhongxiang Sun, *School of Computer and Information Technology, Beijing Jiaotong University*

https://www.usenix.org/conference/atc22/presentation/song

This paper is included in the Proceedings of the 2022 USENIX Annual Technical Conference.

July 11–13, 2022 • Carlsbad, CA, USA

978-1-939133-29-8

Open access to the Proceedings of the 2022 USENIX Annual Technical Conference is sponsored by

AddrMiner: A Comprehensive Global Active IPv6 Address Discovery System

Guanglei Song[1,2], Jiahai Yang[1,2], Lin He[1,2], Zhiliang Wang[1,2], Guo Li[1], Chenxin Duan[1], Yaozhong Liu[1,2], Zhongxiang Sun[3]
[1]*Institute for Network Sciences and Cyberspace, BNRist, Tsinghua University*
[2]*Quan Cheng Laboratory, Jinan, Shandong, China*
[3]*School of Computer and Information Technology, Beijing Jiaotong University*

Abstract

Fast Internet-wide scanning is essential for network situational awareness and asset evaluation. However, the vast IPv6 address space makes brute-force scanning infeasible. Although state-of-the-art techniques have made effective attempts, these methods do not work in seedless regions, while the detection efficiency is low in regions with seeds. Moreover, the constructed hitlists with low coverage cannot truly represent the active IPv6 address landscape of the Internet.

This paper introduces *AddrMiner*, a systematic and comprehensive global active IPv6 address probing system. We divide the IPv6 address space regions into three kinds according to the number of seed addresses to discover active IPv6 addresses from scratch, from few to many. For the regions with no seeds, we present *AddrMiner-N*, leveraging an organization association strategy to mine active addresses. It fills the gap of address probing in seedless regions and finds active addresses covering 86.4K IPv6 prefixes announced by BGP, accounting for 81.6% of the probed announced prefixes. For the regions with few seeds, we propose *AddrMiner-F*, utilizing a similarity matching strategy to probe active addresses further. The hit rate of active address probing is improved by 70%-150% compared to existing algorithms. Moreover, for the regions with sufficient seeds, we present *AddrMiner-S* to generate target addresses based on reinforcement learning dynamically. It nearly doubles the hit rate compared to the state-of-the-art algorithms. Finally, we deploy *AddrMiner* and discover 2.1 billion active IPv6 addresses, including 1.7 billion de-aliased active addresses and 0.4 billion aliased addresses, through continuous probing for 13 months. We would like to further open the door of IPv6 measurement studies by publicly releasing *AddrMiner* and sharing our data.

1 Introduction

Internet-wide active address probing is a prerequisite for Internet-scale network surveys. Existing network research and applications rely heavily on Internet-wide active address scanning. For example, the probed active addresses can be used to examine trends and adoption rates of different technologies [11, 30], measure network topology for reflecting interconnections of nodes [4, 52], probe Internet services for wide-ranging assessments [9, 25] and resource census [23], and test network security by measuring the attack surface [16, 34]. Under IPv4, it is feasible to achieve Internet-wide active address probing by brute-force scanning the entire IPv4 address space at the minute level with high-speed scanning tools such as ZMap [9]. With the rapid development of the Internet as a globally crucial infrastructure, IPv4 no longer meets its development needs, and IPv6 has been promoted and deployed at an accelerated pace worldwide. For example, more than 36.6% of users accessed Google via IPv6 in 2021, compared to fewer than 0.7% in 2012 [21]. However, under IPv6, there are significant challenges to Internet-wide active address discovery. The main reason is that the vast address space of IPv6 makes it more difficult, if not infeasible, to obtain globally active addresses. For example, it would take at least millions of years to scan the entire IPv6 address space using 10 Gigabit links and high-speed scanning tools such as ZMap [9].

To address this issue, researchers usually collect known active IPv6 addresses (i.e., seeds), learn the characteristics of seeds, and generate the target addresses that may have a higher probability of being active for scanning. Although previous research efforts have examined how to detect active IPv6 addresses, the issue of how to perform comprehensive global active IPv6 address discovery remains, mainly in the following aspects:

(1) *Limited usage*. In regions where seeds are missing, existing methods cannot perform effective active IPv6 address probing or even work [7, 15, 18, 24, 33, 36, 47, 51]. This is because they need to learn the characteristics of the seeds to generate target addresses that are more likely to be active. There is still a gap in active IPv6 address probing in regions lacking seeds.

(2) *Limited detection efficiency*. State-of-the-art algorithms [7, 24, 47] have improved the efficiency of active IPv6 address probing. However, these methods are too dependent on seeds. The seed address sampling bias reduces the efficiency of

active address probing because it makes the characteristics of the actual active address inconsistent with those of the seeds.

(3) *Limited coverage*. Previous studies, while building a list of active addresses, called IPv6 hitlist, have tended to be limited to a few IPv6 prefixes announced by BGP [15, 18, 42, 47]. For example, active IPv6 addresses in the latest hitlist [18] cover only 25.5K announced prefixes, which is only ∼21.3% of all announced prefixes. They are not truly representative of the active IPv6 address landscape of the Internet. Active detection methods by analyzing seeds are often also limited by the coverage of the seeds [15, 18, 24, 33, 36, 42, 47, 51].

In general, there still lacks a systematic methodology for comprehensive global active IPv6 address probing. To solve the above problems, we design and implement an active address probing system, *AddrMiner* (§4). At its core, *AddrMiner* divides active address probing into three sub-tasks: active IPv6 address probing for 1) address space regions with no seeds, 2) address space regions with few seeds, and 3) address space regions with sufficient seeds, respectively.

First, we present *AddrMiner-N* for the address space regions without seeds (§5). The core idea is based on the observation that address patterns (i.e., structure) tend to have similarities across network configurations. It obtains common patterns by mining the structural features of active addresses collected in other regions, and then migrate to regions without seeds to generate targets for scanning. *AddrMiner-N* leverages graph data structures to describe the similarity of address structure features under different networks (§5.2). Then, it uses graph community discovery algorithms to mine common address structure features for building a common pattern library (§5.3). We observe that the address configuration patterns are more similar within the same network organization than within different organizations. *AddrMiner-N* selects the most similar patterns from the library based on organization association strategy to generate targets (§5.4).

Second, we propose *AddrMiner-F*, an active address probing algorithm for the case where the address space regions contain few seeds (§6). Existing methods cannot effectively learn seed characteristics for active address probing in this scenario due to the lack of seeds. The core idea of *AddrMiner-F* is also to generate targets for probing by selecting the most relevant patterns from the common pattern library and migrating to regions with only a few seeds. Similar to *AddrMiner-N*, *AddrMiner-F* first uses the same method to build a common pattern library (can reuse the one built by *AddrMiner-N*). Then, it improves the efficiency of address detection by extracting relevant patterns from the common pattern library to generate scanning targets using only a few seeds. This is because a few seeds can also provide some information for guiding pattern selection.

Third, we present *AddrMiner-S*, which learns the density characteristic of seeds and corrects density bias to find the real high-density regions of active addresses for address detection, for the case where the address space regions contain sufficient seeds (§7). The key idea of *AddrMiner-S* is motivated by the higher density regions of active addresses, the higher the hit rate of active addresses. It uses reinforcement learning to update the density distribution of the seeds based on the rewards found for the active addresses and moves toward the actual address distribution to correct the density bias caused by the sampling of seeds.

AddrMiner naturally works in all announced prefix spaces and enables comprehensive active IPv6 address probing in different scenarios by corresponding algorithms to gradually discover active IPv6 addresses from scratch, from few to many.

Contributions. We make the following contributions:

- We present an active IPv6 address probing method, *AddrMiner-N*. It fills the gap of address probing in the seedless address space regions and discovers active IPv6 addresses covering 86.4K prefixes announced by BGP, accounting for 81.6% of all announced prefixes.
- We propose an active IPv6 address probing method, *AddrMiner-F*, which can further discover active IPv6 addresses in address space regions with few seeds. It can find 70%-150% more active addresses than *AddrMiner-N* and the state-of-the-art algorithms.
- We present an efficient active IPv6 address probing method, *AddrMiner-S*, which can efficiently perform active IPv6 address probing in address space regions with sufficient seeds. Compared with state-of-the-art algorithms, the results show *AddrMiner-S* improves the hit rate of active addresses from 28.9% to 56.3%.
- We originally design and implement a global active IPv6 address probing system and discover 2.1 billion active IPv6 addresses, including 1.7 billion de-aliased active addresses and 0.4 billion aliased addresses, through continuous running *AddrMiner* for 13 months. The developed code and continuously probed active addresses are made publicly available at:

  https://github.com/AddrMiner/AddrMiner

2 Background

In this section, we briefly introduce the background of IPv6 addresses and discuss the characteristics of IPv6 addresses.

IPv6 addresses are 128 bits long. IPv6 unicast addresses consist of a global routing prefix, a local subnet identifier, and an interface identifier (IID). We represent IPv6 addresses in a human-readable text format using eight groups of four hexadecimal characters, each group having 16 bits in total, separated by a colon (":"). We refer to each hexadecimal character (corresponding to the four bits

of the address) as a nybble. An example IPv6 address is 2001:0db8:0000:0000:0008:8000:200c:417a. To simplify the IPv6 representation, the leading zeros of each group are usually excluded, and the longest all-zero group sequence is replaced with a double colon ("::"). Thus, the simplified representation of the IPv6 address in this example is 2001:db8::8:8000:200c:417a.

IPv6 addresses have the following characteristics. (1) Vastness of IPv6 address space: IPv6 address space is 2^{128}, 2^{96} times of IPv4 address space. This makes active IPv6 addresses very scarce and more hidden, making it a challenging task to find active IPv6 addresses. (2) Diversity of IIDs: IID can be assigned in various ways, such as static configuration [20], stateless address autoconfiguration [37], and DHCPv6 [40]. IPv6 addresses with random IIDs are more difficult to detect.

3 Related Work and Motivation

This section reviews related work and clarifies the motivation of our work on discovering active addresses in the vast IPv6 address space. The existing work can be divided into the following three categories:

Public Resources Extraction. This method obtains active IPv6 addresses through public resource lookup or resolution [6, 13, 18, 50]. DNS is a common and effective channel. Strowes et al. [50] obtained 965K globally routable IPv6 addresses by exhaustively enumerating the reverse DNS domains in the IPv4 address space and performing AAAA queries on the results. Fiebig et al. [13, 14] walked the rDNS tree and collected 5.8M IPv6 addresses. Borgolte et al. [6] also obtained 2.2M IPv6 addresses through DNSSEC-signed reverse zones. Besides, Gasser et al. [18] collected 58.5M IPv6 addresses from public data sources, including Domain Lists [1, 2, 17, 41, 44, 49], FDNS [46], AXFR [35], Bitnodes [54], and RIPE Atlas [38].

Although we can get IPv6 addresses through public resources, in the latest hitlist [18], these addresses only cover 25.5K announced prefixes, which is only ∼21.3% of all announced prefixes. Therefore, it is challenging to obtain globally active IPv6 addresses from public sources alone.

Passive Collection. This approach entails passively collecting traffic or log files at vantage points and extracting active IPv6 addresses from them [15, 19, 42, 47]. For the first time, Plonka et al. [42] used IPv6 addresses collected from the activity logs of all customers accessing a global CDN as a dataset and analyzed the characteristics of active IPv6 addresses. Subsequently, Foremski et al. [15] proposed a technique for obtaining potentially active IPv6 addresses from the initial seed dataset. A similar attempt was made by [19, 47] using a large Internet Exchange Point as a vantage point to collect active IPv6 addresses.

However, the above studies have the following shortcomings. First, the vantage point used is not publicly available and is difficult for others to access. Second, to obtain global active IPv6 addresses, many vantage points need to be deployed worldwide, with high probing overhead. *AddrMiner* removes the vantage point limitation and decreases the threshold for address probing. Any node configured with IPv6 network can use *AddrMiner* to probe active addresses.

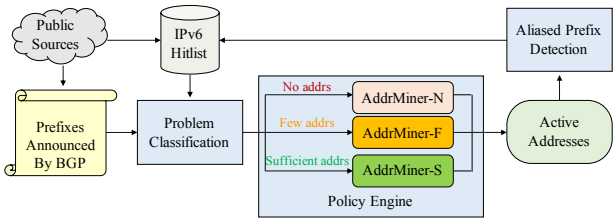


Figure 1: High-level overview of *AddrMiner*

Active Address Probing. A viable approach is to discover more active IPv6 addresses by collecting seeds, mining the structural patterns and characteristics of the seeds to generate target addresses, and probing the target addresses [7, 15, 18, 24, 33, 36, 47, 51]. Ullrich et al. [51] proposed a pattern-based recursive algorithm that greedily includes more seeds for scanning each iteration through a variable address range. Entropy/IP [15, 18] learns the internal structural characteristics of seeds to generate target addresses and then scans them to discover active IPv6 addresses. 6Tree [33] and 6Hit [24] combine the hierarchical characteristic of seeds to construct hierarchical space trees, and dynamically guides the direction of address generation based on the probing results. 6Gen [36] and DET [47, 48] use the density characteristic of seeds to detect active IPv6 addresses in high-density regions of seeds. 6GAN [7] aims to discourage aliased address generation via generative adversarial nets with reinforcement learning.

Although all the above methods improve active IPv6 address probing efficiency, the sampling bias of seeds makes the characteristics of actual active addresses inconsistent with those of seeds, resulting in the inability to efficiently generate target addresses for scanning. Although 6Hit attempts to use reinforcement learning to reduce the dependence on seeds, it simply uses the hierarchical characteristic and uses a space repartition mechanism, which leads to random changes in the probe space to reduce the address probing efficiency.

4 Overview of *AddrMiner*

This section describes an active IPv6 address probing system, *AddrMiner*, capable of performing systematic and comprehensive probing of active IPv6 addresses to achieve the accumulation of detected globally active addresses from scratch.

Figure 1 illustrates a high-level overview of *AddrMiner*. It collects seeds to make an IPv6 hitlist from public sources and divides them into different announced prefix spaces. Then, it classifies these announced prefix spaces into different scenar-

ios based on the number of seeds each prefix space contains. The policy engine uses different policies for active address probing according to different scenarios. 1) For announced prefixes with no seeds, *AddrMiner-N* uses the organization association strategy to select candidate patterns to probe for active IPv6 addresses (§5). 2) For announced prefixes with few seeds, *AddrMiner-F* uses the similarity matching strategy to select candidate patterns to discover active addresses further (§6). 3) For announced prefixes with sufficient seeds, *AddrMiner-S* uses reinforcement learning techniques to learn seed address characteristics while circumventing the shortcomings of similar existing schemes to perform active address probing more effectively (§7). The classification of the scenarios for prefix spaces with seeds is discussed in detail in Appendix C. The detected active addresses discovered from the policy engine are tested for aliased prefixes. After eliminating the aliased prefixes, the de-liased active addresses are the globally active IPv6 addresses and are added to the IPv6 hitlist.

AddrMiner enables comprehensive probing of global active IPv6 address, and provides more and balanced data to support further measurement and security analysis of IPv6 networks.

5 *AddrMiner-N*

This section presents *AddrMiner-N*, which can guide active address detection under an announced prefix without seeds using patterns of active addresses under other prefixes owned by the same organization to which the prefix without seeds belongs.

5.1 Overview of *AddrMiner-N*

Since there are no seeds in the address space region, we cannot use seeds to guide active address probes. An effective way to generate targets under such regions is to use specific IPv6 address patterns, i.e., mining the structural characteristics of active addresses collected in other regions and then migrating to regions without seeds to generate targets for scanning. This idea is feasible due to the observation that address patterns tend to have similarities across network configurations [20]. Our analysis of the tens of millions of IPv6 addresses on the Gasser's hitlist [18] confirms this observation. For example, gateway addresses often have a suffix of ::1 or ::2. Therefore, the crux of the problem is to obtain a common pattern library containing address patterns commonly used in address space regions that have seeds. Our solution, *AddrMiner-N*, is to use undirected graphs to represent the similarity between patterns (§5.2), and then use graph community discovery methods to find communities with high pattern similarity. Each community represents a common address pattern, and these communities construct a common pattern library (§5.3). Finally, it uses the organization association strategy to migrate these common address patterns to any announced prefix for address generation (§5.4).

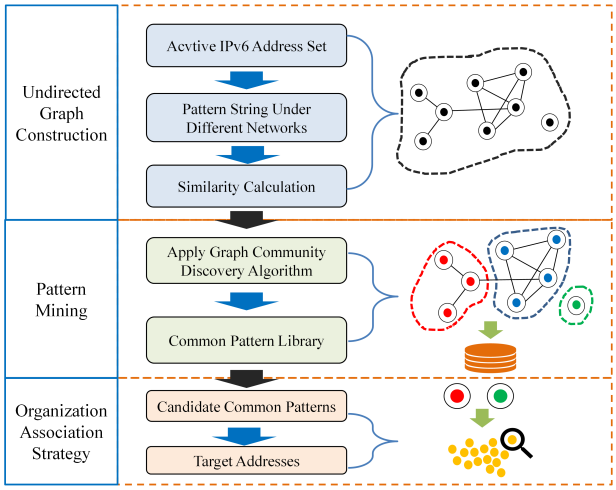


Figure 2: Workflow of *AddrMiner-N*

5.2 Undirected Graph Construction

AddrMiner-N constructs an undirected graph to describe the differences between address patterns. The nodes represent the address patterns, and the weights of the edges represent the similarity between different patterns.

Pattern Representation. The address patterns constructed by existing strategies [20, 22, 29, 32, 47] often do not correspond to an accurate probing space. If the space is too large, such as Embedded-IPv4, it will waste a lot of probing resources and reduce the probing efficiency; if the space is too small, such as low-byte, it will limit the probing range and fail to find a large number of active addresses. To solve such a problem, we use Balanced Spatial Pattern Representation (BSPR) [31] to extract address patterns. BSPR can accept any IPv6 address set as input and generate flexible patterns representing the structural characteristics of that address set, which can be used to generate targets.

First, the BSPR uses four representations to describe the range of values for any nybble of an IPv6 address, including Single, List, Interval, and Wildcard:

Single: The nybble takes a fixed value, which means that the addresses in the address set do not change in the value taken at that nybble position.

List: The value taken for this nybble is variable, and the range is the set of values taken for the IPv6 addresses in the address set at the nybble position.

Interval: The nybble value is variable. The range is a closed interval consisting of the minimum and maximum values of the IPv6 address set at the nybble position, possibly including values that do not appear at the nybble position.

Wildcard: The nybble value is variable and ranges over

Table 1: The relationship between the four representations and three statistics used by BSPR

ID	Range	Entropy	Values	Representation	Example	Number of values
0	0	0.00	1	Single	a	1
1	$\geq t_r$	$\geq t_e$	$\geq t_c$	Wildcard	*	16
2	$\geq t_r$	$\geq t_e$	$< t_c$	List	[1cf]	3
3	$\geq t_r$	$< t_e$	$\geq t_c$	Interval	[1-e]	14
4	$\geq t_r$	$< t_e$	$< t_c$	List	[2be]	3
5	$< t_r$	$\geq t_e$	$\geq t_c$	Interval	[6-b]	6
6	$\geq t_r$	$\geq t_e$	$< t_c$	List	[679]	3
7	$< t_r$	$< t_e$	$< t_c$	List	[67]	2

all hexadecimal values and may include values that do not appear at the nybble position, as indicated by the wildcard *.

To choose a suitable representation, BSPR introduces three statistics for each nybble of IPv6 address in the input address set: range, Shannon entropy, and value count. The range is equal to the maximum value of the value taken by the nybble minus the minimum value; the Shannon entropy can be calculated according to Formula (5-1); the value count is equal to the number of values that have appeared at the position of the nybble.

$$E(x_i) = -\sum_{v=0x0}^{0xf} p(x_i = v)\log_{16} p(x_i = v) \tag{5-1}$$

The base of Formula (5-1) is 16 because the value count of the nybble is 16, which makes the result of the Shannon entropy calculation fall into the interval [0,1], where x_i represents the ith nybble. The value range of i is [1,32], meaning the 32 nybbles of an IPv6 address. $p(x_i = v)$ can be obtained by dividing the number of IPv6 addresses that take the value v at the ith nybble position in the address set by the total number of addresses.

Table 1 shows how BSPR decides the nybble representation based on these three statistics. Whether these three statistics are large or small is determined by three thresholds t_r, t_e and t_c for range, entropy, and value count respectively.

Second, BSPR needs to solve how to determine the value of the three hyperparameter thresholds. If the thresholds are set too small, the modeled address space increases rapidly. At one extreme, all three hyperparameters are set to 0, and the address generation patterns are all in Wildcard. If the thresholds are set too large, the modeled address space is too small. At the other extreme, all three hyperparameters are taken to their maximum values. The patterns of address generation are all in List, and the value of each nybble represented by List depends entirely on the value of the seed address set, which will aggravate the sample bias.

Suppose the counts of List, Interval, and Wildcard in a pattern are l, r, and w, respectively. L_j, R_j represent the count of values taken at the jth List or Interval, respectively. The value range of Wildcard is the 16 values of a single hexadecimal number. Therefore, the size of the space range (SR) of any pattern string is thus calculated as follows:

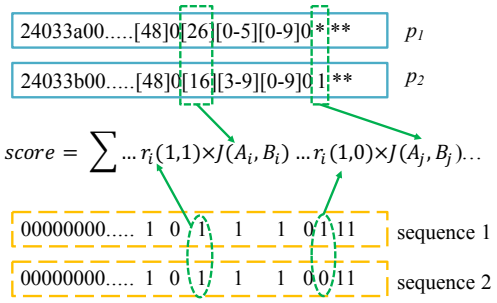


Figure 3: Calculation of the similarity of two patterns

$$SR = 16^w \cdot \prod_{j=1}^{l} L_j \cdot \prod_{j=1}^{r} R_j \tag{5-2}$$

Since l, r, and w are affected by the three hyperparameters, t_r, t_e and t_c, SR is a ternary function with respect to these three parameters. The domains of t_r and t_c are the integer of [0,15] and [1,16], respectively. The domain of t_e is the real number of [0,1], which can be discretized, for example, with an interval of 0.05. Let the range of the ternary function SR be the set Y. Since its domain of definition is a finite set, and the range of Y is also a finite set. Let the number of elements of Y be N. The most balanced space range BSR chosen by BSPR should be the average of the Y range. Here, we use the geometric mean because it is less influenced by extreme values than the arithmetic mean.

$$BSR = \sqrt[N]{\prod_{SR_j \in Y} SR_j} \tag{5-3}$$

Third, BSPR can choose the set of hyperparameters when the value of SR is closest to that of BSR as the values of t_r, t_e and t_c, as shown in Formula (5-4). Finally, a pattern is generated based on Table 1.

$$\arg\min_{t_r,t_e,t_c} |SR - BSR| \tag{5-4}$$

Similarity Calculation. The core of constructing undirected edges is to determine between which nodes undirected edges need to be created and the weights of these undirected edges. Since patterns are represented by strings, the common methods of calculating the similarity between strings can also be used. *AddrMiner-N* introduces Jaccard similarity [28] and Hamming distance-based similarity [53] to calculate the similarity between patterns (specific definitions are given in Appendix A).

Figure 3 shows an example of calculating the similarity between two pattern strings. Two main aspects are considered in the calculation: the similarity of the corresponding nybble representation and the similarity of the values taken by the

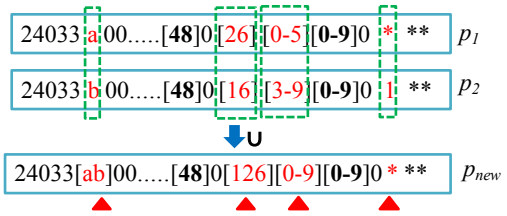


Figure 4: Merging process of different patterns

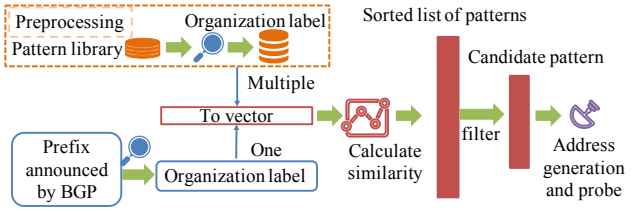


Figure 5: Organization association strategy

representation. For the similarity of the corresponding nybble representation, the main focus is to compare whether the two pattern strings have the same fixed value taken at the same nybble position, i.e., whether they belong to the Single representation or not. Non-Single representations include List, Interval, and Wildcard. We convert all Single representation nybbles to a zero value, and Non-Single representations are represented as one, as shown in Sequence 1 and Sequence 2 in Figure 3. In this way, the similarity of the corresponding nybble representation is obtained by calculating the Hamming distance between Sequence 1 and Sequence 2. Regarding the similarity of the values taken by the representation, Jaccard similarity is used to calculate the similarity of the two sets of values of the representation at the same nybble position. Thus, the similarity of the two pattern strings can be obtained by weighting the Jaccard similarity of the corresponding nybble with the Hamming distance-based similarity of each nybble representation as the weight. The calculation is shown in Formula (5-5), where a_i and b_i denote the values of Sequence 1 and Sequence 2 at the ith position, and A_i and B_i indicate the sets of values of pattern strings p_1 and p_2 at the ith nybble representation, respectively.

$$score = \sum_{i=1}^{32} r_i\left(a_i, b_i\right) \cdot J\left(A_i, B_i\right), \tag{5-5}$$

where r_i indicates Hamming distance-based similarity at the ith nybble and J indicates Jaccard similarity.

Finally, a threshold value h_{min} needs to be determined. If the similarity between two pattern strings exceeds h_{min}, an undirected edge is created. The similarity is used as the weight of that edge. Otherwise, no undirected edge is created. The length of announced prefixes generally does not exceed 56 (14 nybbles). Some prefixes are highly similar, e.g., 2a02:26f0:128:100:/56 and 2a02:26f0:128:500:/56, which causes the merge pattern to contain unannounced prefixes, e.g., 2a02:26f0:128:*00:/56, but addresses in the unannounced space are inactive. Therefore, when constructing the undirected graph, we set h_{min} to 14.0 to avoid generating non-announced spaces as much as possible.

5.3 Pattern Mining

After constructing the undirected graph (§5.2), we apply the community discovery algorithm to cluster similar nodes in undirected graphs and build a common pattern library by mining common patterns from communities.

The graph community discovery algorithm will produce many communities. These nodes have high similarity in the same community but low similarity in different communities. After obtaining the community, we merge the patterns of the nodes contained in the community to extract the common pattern about the community. To make the pattern contain more seeds, we adopt the union method to obtain the pattern. Suppose that C represents a community, which contains k nodes. That is, $C = \{p_1, p_2, ..., p_k\}$. Then the common pattern of the community is: $p_C = \bigcup_{i=1}^{k} p_i$. At the nybble positions corresponding to the different patterns, we take a union of the values corresponding to the nybble. Figure 4 shows the merging process of different patterns.

5.4 Organization Association Strategy

To probe active addresses under announced prefixes with no seeds, *AddrMiner-N* adopts an organization association strategy, the core of which is to extract the most relevant patterns of the specified announced prefix from the pattern library and then use them to generate target addresses for scanning. The reason for adopting this strategy is that address configuration patterns are more similar within the same network organization than within different organizations. In the constructed common pattern library, for example, the announced prefixes of organization "Wireless Broadband Service Provider Malaysia" contain the following common patterns: the sixth group of nybbles is represented by Wildcard, and the other nybbles are represented by Single, where the last nybble is 1, and the other nybbles are 0. Specifically, the prefix 2405:7c00:a004::/48 contains the pattern 24057c00a00400000000****00000001 and the prefix 2405:7c00:a000::/48 contains the pattern 24057c00a00000000000****00000001. *AddrMiner-N* makes full use of the organization information of announced prefixes to filter relevant patterns from the pattern library. The evaluation results in §8.1 show that this strategy can enhance significantly improve the probing efficiency.

Figure 5 shows the method of filtering the pattern library using organization labels. We first construct organization labels for the announced prefixes to which the patterns in the

pattern library belong. We obtain the organization label by querying the whois information in Hurricane Electric [10]. To avoid the influence of generic words on the organization association strategy, we remove generic words, such as corporation, international, etc. For example, the organization label of the prefix 2a01:111:2003::/48 is "Microsoft Corporation", we add an English word "Microsoft" to the organization label of the prefix. Similarly, we obtain the organization labels of the announced target prefixes in the same way. To calculate the degree of similarity between the organization labels, we next convert these labels directly into vectors by using the most popular fastText [5, 26, 27] pre-training model in word embedding. Then, we use Euclidean distance to calculate the similarity between the organization label of the target prefix and the organization labels of each pattern in the pattern library. The calculation is shown in Formula (5-6) and yields a list of k_{org} most similar patterns:

$$similarity = \sum_{i \in T, j \in W} d(v_i, v_j), \tag{5-6}$$

where T is the set of words for the organization label of the target prefix, W is the set of words for the organization label of a common pattern in the pattern library. This approach can identify the same network organization, e.g., identifying "Akamai Technologies, inc" and "Akamai International B.V." as belonging to the organization "Akamai" and thus selecting more relevant patterns. Note that when the similarity is small, i.e., the prefixes belong to different organizations, such as "Akamai" and "Fastly", the candidate patterns are randomly selected from the common pattern library.

After obtaining the candidate patterns, we use them to generate target addresses under the target prefix. More specifically, iterate over each candidate pattern to generate a specified number of targets, and then replace the prefixes of the generated target addresses with target prefix. Finally, we probe whether these addresses are active or not.

6 *AddrMiner-F*

Target regions with few seeds come from both (1) prefix space regions containing few seeds selected from the public IPv6 hitlist, and (2) transformed by detecting few active addresses after running *AddrMiner-N* in regions without seeds. We experimentally find that the active address hit rate of the state-of-the-art algorithm [47] decreases with the number of seeds, especially when the number of seeds is less than 10, the hit rate is already less than 1% (See more details in Appendix C). However, the number of announced prefixes with only few seeds is large. As shown in §8.1, we find more than 30K announced prefixes with less than 10 seeds. To solve this problem, we propose *AddrMiner-F*, which can use few seeds to extract the most relevant patterns from the common pattern library to generate targets for scanning and achieve effective detection of active addresses in announced prefixes

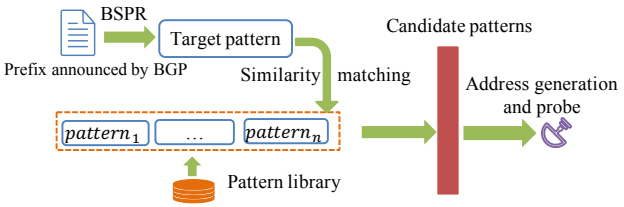


Figure 6: Similarity matching strategy

with few seeds. We call this strategy of matching address patterns using few seeds the similarity matching strategy. *AddrMiner-F* also consists of three steps: undirected graph construction, pattern mining, and similarity matching strategy. Among them, the first two steps have been introduced in §5.2 and §5.3, respectively. The similarity matching strategy is described in detail below.

In this case, we have a small number of IPv6 seeds under the target prefix. Therefore, the similarity matching strategy mainly combines these seeds to filter out a more relevant candidate pattern list from the pattern library. Figure 6 shows the process of similarity matching strategy. We first use BSPR to obtain the patterns of few seeds under the target prefix (i.e., the target pattern). Then we traverse the pattern library and use Formula (5-5) to calculate the similarity between the target pattern and each pattern in the pattern library separately and find the k_{heap} most similar candidate patterns. After getting the candidate patterns, we use the address generation method in §5.4 to generate the target addresses and probe whether they are active or not.

7 *AddrMiner-S*

Target regions with sufficient seeds come from three scenarios: (1) prefix space regions containing sufficient seeds selected from the public IPv6 hitlist, (2) transformed by detecting sufficient active addresses after running *AddrMiner-N* in regions without seeds, and (3) transformed by detecting sufficient active addresses after running *AddrMiner-F* in regions with few seeds. If the address space regions have enough seeds, the state-of-the-art address generation algorithms based on seeds are effective attempts. They learn the characteristics of seeds to generate target addresses for scanning. However, due to the seeds' sampling bias, the characteristics of the seeds do not coincide with the characteristics of the actual active addresses under the address space regions. The sampling bias reduces the probing efficiency and wastes resources. Although 6Hit attempts to use reinforcement learning to eliminate sampling bias, it simply uses the hierarchical characteristic of seeds and it's space repartition mechanism randomly generates target addresses to reduce the efficiency of active address detection. In our work, we propose *AddrMiner-S*, which learns seeds' density characteristic and uses reinforcement learning to correct the discrepancies in the density distribution caused

by the sampling of seeds. In space expansion, *AddrMiner-S* guide the target address generation in a larger address space by merging subspace density characteristic. The model about *AddrMiner-S* is built in Appendix B.

7.1 Target Address Generation Based on Reinforcement Learning

We know the higher the density of active addresses, the higher the hit rate of active addresses (The theoretical proof is in Appendix B). Although the active address density of each region in the real IPv6 network is unknown, we estimate the active address density of each region through Thompson sampling. After discovering seeds' high-density regions, we use the reinforcement learning method to select candidate regions for generating target addresses in the seed address's high-density regions, update the active address density distribution through feedback rewards of each iteration's scanning results, and dynamically adjust the target address generation's direction. As the number of iterations increases, the evaluation of the probability of each action's reward will become more accurate. Eventually, high-density regions of active addresses in the real network will be discovered, and address generation will be performed in the high-density regions.

Space Partition: We first discover the high-density regions $X = \{x_1, x_2, ..., x_k\}$ of seeds. To quickly cluster the density space distribution of seeds, we use the density space tree [47] to find high-density regions of the seeds in linear time. The root node represents the entire active address space, and the leaf node represents a high-density region of seeds. In each node region x_i, there are two attributes α_i and β_i. Where α_i represents the number of active addresses probed in region x_i, and β_i represents the number of inactive addresses discovered in region x_i. Initially, we take out all the leaf nodes from the density space tree as the high-density regions set X.

After discovering regions with a high density of seeds, we dynamically probe active IPv6 addresses based on reinforcement learning. The iterative process of reinforcement learning consists of three main steps: 1) Generate target addresses to probe (action), 2) Update the reward of probed regions with the number of active addresses and inactive addresses (action's reward) to update the density distribution, and 3) Merge the nodes of the space tree to meet the needs of exploring a larger address space.

Target Generation: To adapt to the large-scale probing of addresses and speed up address probing, we select multiple target regions in each iteration, and the budget (the probing number of target addresses) consumed is b. Since the node region with a larger reward is more likely to discover active addresses, in each iteration, we select the top P searchable regions based on the reward for target address generation in the candidate regions X. We use prior events (action's reward) to evaluate the distribution of active address density. However, the larger space, the higher the risk of searching in the node region (more difficult to find active addresses). For example, in extreme cases, the hit rate of active addresses is extremely low in the entire IPv6 address space. To reduce the risk of low address probing efficiency due to excessive space, we use the region address variable space (variable dimensions) to adjust the probability of generating active addresses in each region. The number of target addresses generated in each region is calculated as follows:

$$p(x_i) = \frac{e^{R_i}}{log(V_i) * \sum_{i=1}^{n} \frac{e^{R_i}}{log(V_i)}} \tag{7-1}$$

$$N(x_i) = b * p(x_i) \tag{7-2}$$

Where R_i indicates the expected reward in region i, $p(x_i)$ indicates the probability of generating target addresses in region x_i, $N(x_i)$ indicates the number of target addresses generated in region x_i, V_i represents the number of variable dimensions in active addresses in region x_i, and n represents the probing regions of the top P percent of the candidate regions X, b represents the budget consumed per iteration.

Reward Update: We update node regions' reward to increase the chance of generating target addresses in high-density regions for next-round probing. After each round of probing, we need to update the probed node region's reward value based on the probing result. Initially, we take out all the leaf nodes from the density space tree as the high-density regions set X and each leaf node's reward in x_i is initialized as follows:

$$R_i^1 = Beta(\alpha_i^1, \beta_i^1) \tag{7-3}$$

where R_i represents the expected reward in leaf node region x_i. Initially, α_i^1 is the number of seeds distributed in the leaf node region x_i plus 1, and β_i^1=1.

After each iteration, the expected reward of the probed region x_i is updated as follows:

$$R_i^{t+1} = Beta(\alpha_i^t + \alpha^*, \beta_i^t + \beta^*) \tag{7-4}$$

where α^* represents the number of new active addresses from scanning result in node region x_i, and β^* represents the number of new inactive addresses from scanning result in node region x_i. We assume b^* represent the target address generated in the node region x_i in each iteration. α^*, β^* and b^* satisfy the following relationship: $b^* = \alpha^* + \beta^*$.

Node Merging: The search space in the node is defined as the seed address's variable dimensions, but this will cause the search space to be incomplete. We adopt the method of merging upward after the child node's space search is completed, thus ensuring that space not included in the child node can be searched in the parent node.

When a leaf node region needs to be merged, we need to merge all the leaf nodes of the subtree (T) rooted at this leaf node's parent node to ensure that addresses continue to be generated in the high-density region. We can get all

the leaf nodes recursively, but the time consumption is too high. Because the leaf nodes of T are all included in the density regions X to be searched, we can store all the node's child nodes during the tree-building process and only need to intersect with X to get all the leaf nodes when merging. The merging strategy of the node's parameters is as follows:

1) Probed addresses merge: The active addresses (α_f) and inactive addresses (β_f) found in the parent node (f) region is equal to the union of the set of active addresses found in all child nodes ($C = \{x_1, ..., x_j\}$). The specific relationship is expressed as follows: $\alpha_f = \bigcup_{i=1}^{j} \alpha_i$ and $\beta_f = \bigcup_{i=1}^{j} \beta_i$.

2) Reward merge: The parent node's reward value still satisfies beta distribution, and the reward = $Beta(\alpha_f, \beta_f)$ is calculated based on the active and inactive addresses obtained by strategy 1).

3) Space merge: The target address generation space of the parent node is equal to the variable space of the parent node minus the variable space of the child nodes. The specific relationship is expressed as follows:
$f.var\_space = f.var\_space - \bigcup_{i=1}^{j} x_i.var\_space$.

8 Evaluation

This section highlights the evaluation of the effectiveness of active address probing for *AddrMiner*. *AddrMiner* is an active measurement method to discover active addresses. Therefore in our experimental evaluation, we compare *AddrMiner* with active address probing methods, not with passive collection methods (e.g., vantage point mirroring traffic) or public resource extraction methods (e.g., rDNS, Domain Lists, FDNS, AXFR). In all following experiments, we perform aliased prefix detection and aliased address removal.

Data: We automated the process of obtaining Gasser's publicly de-aliased active addresses from December 2020 to June 2021, and obtained 46.2M active IPv6 addresses, covering 49.2K announced prefixes. In addition, we obtained 105,973 announced prefixes from the Pyasn project [3]. As shown in Table 2, we classify announced prefix spaces into no seed address spaces, few seed address spaces, and sufficient seed address spaces according to the number of seeds each announced prefix space contains. We have explored the number of seeds on the probing efficiency in Appendix C and selected target regions with the number of seed addresses less than ten as few seed scenarios.

Active Detection: When judging whether the target address is active, we send an ICMPv6 request packet using the ZMap to each address. If we receive a response from an address, we determine that it is active at the time of detection.

Default Parameters: We empirically set the important parameters. In undirected graph construction, h_{min} is set = 14.0. In pattern mining, the Louvain algorithm is used for graph community discovery. Pattern strings with space range *SR* greater than 10^7 are filtered. In *AddrMiner-S*, we set P to 0.05, i.e., the top 0.05 percent of the highest reward nodes are selected for probing. We set k_{org} and k_{heap} to 10.0. The maximum number of seeds contained in each leaf node is 4, i.e., $\delta = 4$. We set the granularity of the IPv6 address representation to 4, i.e., $\gamma = 4$.

Table 2: Scenarios classification in the data set

Scenarios Classification	The number of announced prefixes
No seeds	56,730
Few seeds (≤ 10)	31,771
Sufficient seeds	17,472

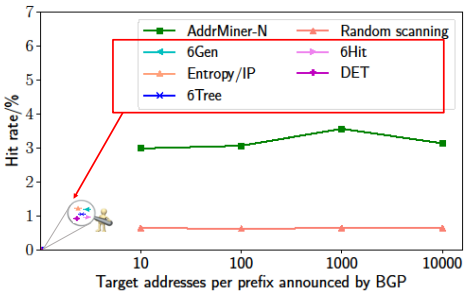


Figure 7: Hit rate of active addresses in the no seed scenario.

8.1 Efficiency of *AddrMiner-N*

Suppose b_{count} represents the number of announced prefixes to be probed, p_{count} indicates the number of candidate patterns for each announced prefix, and g denotes the number of addresses generated in each pattern. Thus, the number of addresses generated in each announced prefix is $p_{count} \times g$, and the number of target addresses generated in all announced prefixes is $M = b_{count} \times p_{count} \times g$. We generate different numbers of target addresses for each announced prefix without seeds, i.e., b_{count} = 56,730, p_{count} = 10, and g=1, 10, 100, 1000.

Figure 7 shows the active address probing results of *AddrMiner-N* and other existing methods in the seedless address scenario. The vertical axis represents the average hit rate of probed prefix spaces without seeds (b_{count}). We found that state-of-the-art target address generation algorithms, including Entropy/IP [15], 6Gen [36], 6Tree [33], 6Hit [24], and DET [47], do not work in seedless regions since they need to learn seeds' characteristics. Compared with random scanning, which has extremely hit rate of only about 0.6%, *AddrMiner-N* has a higher hit rate of up to 3.6% for active address probing.

Furthermore, we perform a more comprehensive probing through the announced prefix space. We use 105,973 announced prefixes as probing regions, employ 500 patterns under each announced prefix, and generate 100 target addresses under each pattern, i.e., $b_{count} = 105,973$, $p_{count} = 500$, and

Table 3: The probing results of the two probing methods

Probing Method	#Active Addrs	#BPFXs	Coverage
AddrMiner-N	158,959,500	86,423	81.6%
Random Scanning	708,697	1,421	1.3%

BPFXs: prefixes announced by BGP.

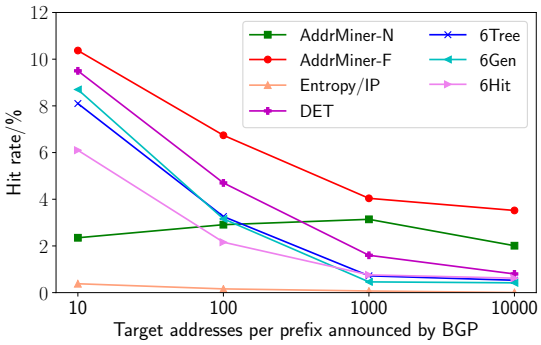


Figure 8: Hit rate of active addresses in the few seed scenario.

$g = 100$. Thus, we generated about 5.2 billion target addresses for probing. Table 3 shows the number of active addresses discovered by the two probing methods and the number of announced prefixes covered by the active addresses. *AddrMiner-N* discovered approximately 159.0M active addresses, covering 86.4K announced prefixes, accounting for 81.6% of all announced prefixes. It indicates that *AddrMiner-N* can perform active address probing over a broader address space and is more suitable for global active IPv6 address probing.

8.2 Efficiency of *AddrMiner-F*

In few seed scenario, we evaluate *AddrMiner-F* by comparing active address hit rate with *AddrMiner-N* and the state-of-the-art target address generation algorithms. We generated different numbers of target addresses among 31,771 announced prefixes containing few seeds, i.e., b_{count} = 31,771, p_{count} = 10, and g=1, 10, 100, 1000. Figure 8 shows the hit rate of active address probing. When target addresses are small, DET, 6Hit, and 6Tree randomly generate target addresses in low nybble space. As the target addresses increase, the larger the target address will cause the hit rate to decrease. Therefore, in the scenario with few seeds, the state-of-the-art target address generation algorithms are inferior. The hit rate of active addresses is very low when the target space represented by the pattern is too large or too small. We probed all mined common patterns and found that the hit rate was highest when the pattern contained 2 variable nybbles ([32-256] targets). This is because we choose the pattern space closest to the probing number (G) to generate targets randomly. This explains why the hit rate of *AddrMiner-N* increases and then decreases

Table 4: Ratio of common patterns in the pattern library

Patterns	Example of patterns in pattern library	Ratio/%
Low-byte	20010db800000000000000000000000[1-a]	25.886
Embedded-IPv4	20010db80122034400000000874b2b[3-f][4-f]	7.420
Embedded-port	20010db8000000000000000000000[01]***	0.100
ISATAP	fe8000000000000002005efec0000***	0.002
EUI-64	fe8000000000000002aa00fffe3f[2-f][a-c]1c	3.100
Other	240085001000000000de00e300**00**	63.490

in Figure 7 and Figure 8. As the number of target addresses increases, the expansion of the probe space reduces the effectiveness of similarity matching, which affects the efficiency of *AddrMiner-F* probing. However, *AddrMiner-F* is more effective overall than the other methods. When *AddrMiner-F* generates 10-10,000 target addresses for each announced prefix containing few seeds, the active address probing efficiency is improved by 70%-150% compared to existing methods. *AddrMiner-F* enables a more efficient transition from few active address scenario to sufficient address scenario.

8.3 Common Pattern Library Analysis

The common pattern library generated by *AddrMiner* is a set of pattern strings. We analyze this pattern library and provide some further guidance for active IPv6 address probing.

RFC documents [20, 32] presents several common patterns in the IID of IPv6 address, e.g., may be a low-byte IID with a run of zeroes followed only by a low number, an embedded-IPv4 IID inserting one IPv4 address, an embedded-port IID including the service port in the lowest-order byte of the IID, an ISATAP IID with "0200:5EFE" flag and IPv4 address, an EUI-64 IID with an embedded MAC address. Table 4 shows the common address patterns, examples of pattern strings, and the ratio in the pattern library. We find that EUI-64 IID and ISATAP take fixed values at some locations where the address is fixed, so the pattern strings use the Single policy, which is a non-dynamically changing nybble, at these fixed locations. Low-byte address patterns will have more consecutive nybbles in the middle that takes on a value of zero and only change at the end in multiple consecutive nybble positions, such as 20010db800000000000000000000000[1-a] for 2001:db8::1 or 2001:db8::4 for these types of IPv6 address structures. The high percentage of the low-byte address pattern in the pattern library means that a brute-force scanning can be attempted for such structures, i.e., fixing the value of the middle nybbles to 0 and traversing only the last consecutive nybbles. It explains the high hit rate of incremental scanning when the number of targets is small. In addition, *AddrMiner* can find many additional address patterns, such as the other types in Table 4, which also account for 63.49% of the total. Although such address patterns do not correspond to address patterns known from RFC documents, their stronger regularity can reduce the difficulty in brute-force scanning of such addresses and improve the efficiency of address probing.

In short, *AddrMiner* can dig out address patterns that not only contain the address patterns of RFC documents, but can also discover more valuable address patterns. We do not consider the assignment of given address space to one of the classes to be static. Furthermore, *AddrMiner* adds newly discovered addresses to the IPv6 hitlist, and updates the common patterns promptly for dynamic IPv6 address spaces.

8.4 Efficiency of *AddrMiner-S*

We evaluate the efficiency of *AddrMiner-S* in probing active addresses by comparing the active address hit rate of *AddrMiner-S* and the state-of-the-art algorithms.

Here, we randomly select announced prefixes that contain more than 1K seeds. We run the above algorithms for each announced prefix to generate target addresses with a budget of 10 times the seeds. Figure 9(a) illustrates the address probing efficiency of *AddrMiner-S* in announced prefixes with sufficient seeds. We observe that the active address hit rate of *AddrMiner-S* outperforms other state-of-the-art algorithms in every announced prefix. In particular, the active address hit rate of *AddrMiner-S* reaches 35.2% in prefix 2001:1291::/32.

To further validate the efficiency of *AddrMiner-S*, we randomly select 1M active addresses as seeds from Gasser's public hitlist. We use *AddrMiner-S* and state-of-the-art algorithms (Note that 6GAN is not suitable for large-scale global active address detection since the time complexity is too high based on deep learning framework) to generate target addresses with budgets ranging from 10M to 50M. We set the budget b consumed for each iteration to 10K. Figure 9(b) shows the probing results after removing the aliased addresses. We find that *AddrMiner-S* outperforms the other algorithms. When the budget is 50M, the hit rates of the algorithms from highest to lowest are *AddrMiner-S* (56.3%), DET (28.9%), 6Tree (12.9%), 6Gen (14.6%), and Entropy/IP (3.1%), 6Hit (2.6%), and the hit rate of *AddrMiner-S* is almost twice as much. In particular, 6Hit has a high hit rate when the budget is small. Still, space expansion leads to a rapid decrease in hit rate as the budget increases because the target addresses are generated randomly due to the spatial repartition mechanism of 6Hit. *AddrMiner-S* maintains the state learned from subspace during space expansion to avoid a rapid decrease in hit rate and effectively improve detection.

In the ideal sampling case, the density distribution of seeds is consistent with the density distribution of active addresses in the actual network. The reward (R_i) of each iteration reflects the active address density distribution of the real network. The density distribution of seeds updated by rewards in the next iteration (R_i^{t+1}) is more convergent to the distribution of active addresses in the actual network compared to the previous iteration (R_i^t). Therefore, the similarity between the current seed address density distribution and the actual network's active address density distribution can be obtained by calculating the difference in reward ranking after each iteration using Hamming distance. As shown in Figure 9(c), the density distribution of seeds increasingly converges to the density distribution of active addresses in the actual network as the number of iterations increases. In addition, *AddrMiner-S* strikes a balance between exploration and exploitation (the specific analysis is given in Appendix D).

9 IPv6 Hitlist

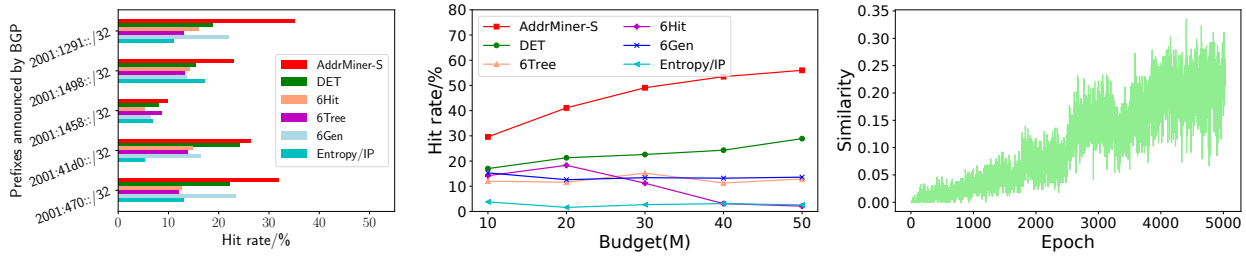
AddrMiner probes each IPv6 prefix announced by BGP, requiring approximately one month to probe all announced prefixes. As the number of all announced prefixes exceeds 100K, this results in a long probing time. Therefore, to deal with the dynamic changes in the IPv6 space, we repeat the probing of IPv6 prefixes announced by BGP every month. The probe period should be set as short as possible to get a more accurate view of active IPv6 addresses, depending on the probe resources. We have developed *AddrMiner* for continuously probing active IPv6 addresses worldwide for 13 months and discovered 2.1B active addresses (covering 86.4K announced prefixes), including 1.7 billion de-aliased active addresses (IPv6 hitlist) and 0.4 billion aliased addresses. Meanwhile, we found 1.1M aliased prefixes, which are described and analyzed in Appendix E. The IPv6 hitlist is analyzed as follows:

Time Characteristics. Active IPv6 addresses have time characteristics. IPv6 addresses, especially client addresses, have a short lifetime. Therefore, when a probe response is received, we can only determine that the IPv6 address is active at the response time. We analyze the stability of addresses to determine addresses' lifetime, mainly by separating server addresses, router addresses, persistent or stable client addresses, and temporarily active client addresses.

We define nd-stable to represent the stability of the address. For example, 1d-stable is active for at least one day during the continuous detection period, and nd-stable address means active for at least n days. The active address of nd-stable is also the address of (n-1)d-stable. We send an ICMPv6 request to each address we collect every day and record these addresses' lifetime according to the response information from January 8, 2021. As shown in Table 5, we found that the long-term active addresses(100d-stable addresses) are more than 46%. Compared with temporarily active client addresses, long-term active addresses are more meaningful for detection.

IID Analysis. We analyze the IID types of active address assignments to understand the global IPv6 address configuration landscape. Utilizing the addr6 tool [12], we have divided the IID portion of IPv6 addresses into different types.

In Table 5, we analyze the IID allocation types of different stable addresses. The 1.7 billion 1-stable de-aliased addresses mean IPv6 hitlist we collected. We found that pattern-bytes (some discernible patterns) IID addresses accounted for as high as 40.8%, closely related to our detection strategy because we mainly detect active addresses by constructing common pattern library (similarly, low-byte IID and embedded-



(a) Hit rate in prefixes with sufficient seeds (b) Hit rate in Gasser's hitlist. (c) Consistency of density.

Figure 9: Comparisons between the *AddrMiner-S* and the state-of-the-art algorithms with sufficient seeds.

Table 5: IID Analysis of Discovered n-stable Addresses

-	#IPs	EUI-64	Embedded-IPv4	Pattern-bytes	Randomized	Low-byte
1d-stable(Hitlist)	1.7B	71.4M (4.2%)	251.6M (14.8%)	676.6M (39.8%)	411.4M (24.2%)	277.1M (16.3%)
7d-stable	1.1B (65.8%)	57.8M (3.4%)	212.5M (12.5%)	506.6M (29.8%)	113.9M (6.7%)	227.8M (13.4%)
30d-stable	919.4M (54.1%)	760.8K (0.0%)	204.0M (12.0%)	498.1M (29.3%)	13.6M (0.8%)	202.3M (11.9%)
60d-stable	860.2M (50.6%)	701.6K (0.0%)	190.4M (11.2%)	464.1M (27.3%)	13.5M (0.8%)	188.7M (11.1%)
100d-stable	783.7M (46.1%)	680.4K (0.0%)	173.4M (10.2%)	425.0M (25.0%)	10.3M (0.6%)	173.3M (10.2%)

IPv4 IID addresses). In addition, the IPv6 hitlist contains 24.2% of randomized IID addresses, which are randomly generated in high-density regions. The EUI-64 IID addresses are only 4.2%. This is because Gasser's hitlist contains a small proportion of EUI-64 IID addresses. At the same time, the detected address space is small, and there is no address generation in the EUI-64 flag. In vertical analysis, we found that randomized IID and EUI-64 IID addresses are more unstable during continuous detection. The proportion of temporarily active client addresses is high, and the lifetime is less than seven days. Embedded-IPv4 IID, low-byte IID, and pattern-bytes IID addresses have high stability and a long lifetime. These are more likely to contain long-term active and stable client addresses, server addresses, router addresses, etc.

We further analyze the organization and location distribution of active addresses in the IPv6 hitlist in Appendix F.

10 Ethical Considerations

To perform global IPv6 address probing, we follow ethical conventions for network measurement, including recommendations provided by Partridge et al. [39] and Dittrich et al. [8]. We first evaluate whether active address measurements induce harm to the probed hosts and networks. We send only one probe packet to each IP address, which minimally affects the host and the network where the IP is located. To avoid duplicate probes, *AddrMiner* removes IPv6 addresses that have already been probed from the generated target addresses. Next, we evaluate whether the probing behavior will cause harm to the local network. We will use distributed probes with a probing rate limit of 10 Mbps per probe to avoid causing problems to the network where the probing point is located during active address probing.

11 Conclusion and Future Work

This work proposes a systematic methodology, *AddrMiner*, which comprehensively probes the global active IPv6 addresses. We follow ethical conventions for network measurement. *AddrMiner* divides the global active IPv6 address probing into three scenarios and accumulates active addresses from none to many. We used *AddrMiner* to probe the global active IPv6 addresses and found 2.1 billion active addresses within 13 months. *AddrMiner* removes the limitation of using a vantage point for active IPv6 address probing. Our work will effectively support more researchers to conduct in-depth IPv6 network measurement and security research. We share code and data at: https://github.com/AddrMiner/AddrMiner.

In future work, we will continue to probe active IPv6 addresses. In addition, the blocking strategies and middle boxes can affect the detection of active addresses [25], we will further study their impact on active address detection.

12 Acknowledgments

We would like to thank our shepherd, Adrian Perrig, and the anonymous reviewers for their insightful comments. We also thank Chenglong Li, Enhuan Dong, Yichao Wu, Jinjin Wei, Jinlei Lin, Long Pan, Hao Gao, Yirui Luo, and Leyao Nie for their feedback and suggestions. This work is supported by the National Key Research and Development Program of China under Grant No. 2018YFB1800200 and Beijing Natural Science Foundation under Grant No.4222026. Lin He and Jiahai Yang are the corresponding authors of this paper.

References

[1] Johanna Amann, Oliver Gasser, Quirin Scheitle, Lexi Brent, Georg Carle, and Ralph Holz. Mission accomplished?: HTTPS security after diginotar. In *Proceedings of the 2017 Internet Measurement Conference*, pages 325–340. ACM, 2017.

[2] APWG. Apwg: Cross-industry global group supporting tackling the phishing menace. http://antiphishing.org, 2018.

[3] Hadi Asghari and Arman Noroozian. Pyasn. https://pypi.org/project/pyasn/, 2020.

[4] Robert Beverly, Ramakrishnan Durairajan, David Plonka, and Justin P Rohrer. In the ip of the beholder: Strategies for active ipv6 topology discovery. In *Proceedings of the 2018 Internet Measurement Conference*, pages 308–321, 2018.

[5] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomás Mikolov. Enriching word vectors with subword information. *Trans. Assoc. Comput. Linguistics*, 5:135–146, 2017.

[6] Kevin Borgolte, Shuang Hao, Tobias Fiebig, and Giovanni Vigna. Enumerating active ipv6 hosts for large-scale security scans via dnssec-signed reverse zones. In *2018 IEEE Symposium on Security and Privacy (S&P)*, pages 770–784, 2018.

[7] Tianyu Cui, Gaopeng Gou, Gang Xiong, Chang Liu, Peipei Fu, and Zhen Li. 6gan: Ipv6 multi-pattern target generation via generative adversarial nets with reinforcement learning. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pages 1–10, 2021.

[8] David Dittrich, Erin Kenneally, et al. The Menlo Report: Ethical Principles Guiding Information and Communication Technology Research. Technical report, US Department of Homeland Security, 2012.

[9] Zakir Durumeric, Eric Wustrow, and J Alex Halderman. Zmap: Fast internet-wide scanning and its security applications. In *22nd USENIX Security Symposium (USENIX Security 13)*, pages 605–620, 2013.

[10] Hurricane Electric. Hurricane electric bgp toolkit. https://bgp.he.net/, 2021.

[11] Adrienne Porter Felt, Richard Barnes, April King, Chris Palmer, Chris Bentzel, and Parisa Tabriz. Measuring https adoption on the web. In *26th USENIX security symposium (USENIX security 17)*, pages 1323–1338, 2017.

[12] F.Gont. Ipv6 toolkit. https://www.si6networks.com/research/tools/ipv6toolkit, 2021.

[13] Tobias Fiebig, Kevin Borgolte, Shuang Hao, Christopher Kruegel, and Giovanni Vigna. Something from nothing (there): collecting global ipv6 datasets from dns. In *International Conference on Passive and Active Network Measurement*, pages 30–43. Springer, 2017.

[14] Tobias Fiebig, Kevin Borgolte, Shuang Hao, Christopher Kruegel, Giovanni Vigna, and Anja Feldmann. In rdns we trust: revisiting a common data-source's reliability. In *International Conference on Passive and Active Network Measurement*, pages 131–145. Springer, 2018.

[15] Pawel Foremski, David Plonka, and Arthur Berger. Entropy/ip: Uncovering structure in ipv6 addresses. In *Proceedings of the 2016 Internet Measurement Conference*, pages 167–181, 2016.

[16] Kensuke Fukuda and John Heidemann. Who knocks at the ipv6 door? detecting ipv6 scanning. In *Proceedings of the Internet Measurement Conference 2018*, pages 231–237, 2018.

[17] Oliver Gasser, Benjamin Hof, Max Helm, Maciej Korczynski, Ralph Holz, and Georg Carle. In log we trust: Revealing poor security practices with certificate transparency logs and internet measurements. In *PAM*, volume 10771 of *Lecture Notes in Computer Science*, pages 173–185. Springer, 2018.

[18] Oliver Gasser, Quirin Scheitle, Pawel Foremski, Qasim Lone, Maciej Korczyński, Stephen D Strowes, Luuk Hendriks, and Georg Carle. Clusters in the expanse: Understanding and unbiasing ipv6 hitlists. In *Proceedings of the 2018 Internet Measurement Conference*, pages 364–378, 2018.

[19] Oliver Gasser, Quirin Scheitle, Sebastian Gebhard, and Georg Carle. Scanning the ipv6 internet: towards a comprehensive hitlist. *arXiv preprint arXiv:1607.05179*, 2016.

[20] Fernando Gont and Tim Chown. Network Reconnaissance in IPv6 Networks. RFC 7707, March 2016.

[21] Google. Google ipv6. https://www.google.com/intl/en/ipv6/statistics.html, 2021.

[22] Lin He, Gang Ren, Ying Liu, and Jiahai Yang. Pavi: Bootstrapping accountability and privacy to ipv6 internet. *IEEE/ACM Transactions on Networking*, 29(2):695–708, 2021.

[23] John Heidemann, Yuri Pradkin, Ramesh Govindan, Christos Papadopoulos, Genevieve Bartlett, and Joseph Bannister. Census and survey of the visible internet. In

Proceedings of the 8th ACM SIGCOMM conference on Internet measurement, pages 169–182, 2008.

[24] Bingnan Hou, Zhiping Cai, Kui Wu, Jinshu Su, and Yinqiao Xiong. 6Hit: A reinforcement learning-based approach to target generation for internet-wide ipv6 scanning. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, 2021.

[25] Liz Izhikevich, Renata Teixeira, and Zakir Durumeric. LZR: Identifying unexpected internet services. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[26] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hervé Jégou, and Tomás Mikolov. Fasttext.zip: Compressing text classification models. *CoRR*, abs/1612.03651, 2016.

[27] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomás Mikolov. Bag of tricks for efficient text classification. In *EACL (2)*, pages 427–431. Association for Computational Linguistics, 2017.

[28] Fatih Karabiber. Jaccard similarity. *https://www.learndatasci.com/glossary/jaccard-similarity/*.

[29] Seiichi Kawamura and Masanobu Kawashima. A Recommendation for IPv6 Address Text Representation. RFC 5952, August 2010.

[30] Platon Kotzias, Abbas Razaghpanah, Johanna Amann, Kenneth G Paterson, Narseo Vallina-Rodriguez, and Juan Caballero. Coming of age: A longitudinal study of tls deployment. In *Proceedings of the 2018 Internet Measurement Conference*, pages 415–428, 2018.

[31] Guo Li, Lin He, Guanglei Song, Zhiliang Wang, Jiahai Yang, Jinlei Lin, and Hao Gao. Ipv6 active address discovery algorithm based on multi-level classification and space modeling. *Journal of Tsinghua University (Science and Technology)*, 61(10):1177–1185, 2021.

[32] Xing Li, Mohamed Boucadair, Christian Huitema, Marcelo Bagnulo, and Congxiao Bao. IPv6 Addressing of IPv4/IPv6 Translators. RFC 6052, October 2010.

[33] Zhizhu Liu, Yinqiao Xiong, Xin Liu, Wei Xie, and Peidong Zhu. 6tree: Efficient dynamic discovery of active addresses in the ipv6 address space. *Computer Networks*, 155:31–46, 2019.

[34] Soo-Jin Moon, Yucheng Yin, Rahul Anand Sharma, Yifei Yuan, Jonathan M Spring, and Vyas Sekar. Accurately measuring global risk of amplification attacks using ampmap. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[35] Ayman Mukaddam, Imad H. Elhajj, Ayman I. Kayssi, and Ali Chehab. IP spoofing detection using modified hop count. In *AINA*, pages 512–516. IEEE Computer Society, 2014.

[36] Austin Murdock, Frank Li, Paul Bramsen, Zakir Durumeric, and Vern Paxson. Target generation for internet-wide ipv6 scanning. In *Proceedings of the 2017 Internet Measurement Conference*, pages 242–253, 2017.

[37] Dr. Thomas Narten, Tatsuya Jinmei, and Dr. Susan Thomson. IPv6 Stateless Address Autoconfiguration. RFC 4862, September 2007.

[38] RIPE NCC. Ipmap. https://ftp.ripe.net/ripe/ipmap/, 2018.

[39] Craig Partridge and Mark Allman. Ethical Considerations in Network Measurement Papers. *Communications of the ACM*, 59(10):58–64, 2016.

[40] Charles E. Perkins, Bernie Volz, Ted Lemon, Michael Carney, and Jim Bound. Dynamic Host Configuration Protocol for IPv6 (DHCPv6). RFC 3315, July 2003.

[41] PhishTank. A nonprofit anti-phishing organization. http://www.phishtank.com, 2018.

[42] David Plonka and Arthur Berger. Temporal and spatial classification of active ipv6 addresses. In *Proceedings of the 2015 Internet Measurement Conference*, pages 509–522, 2015.

[43] Daniel J. Russo, Benjamin Van Roy, and Abbas Kazerouni. A tutorial on thompson sampling. https://www.overleaf.com/project/60827af280c8e85011d3b800, 2021.

[44] Quirin Scheitle, Taejoong Chung, Jens Hiller, Oliver Gasser, Johannes Naab, Roland van Rijswijk-Deij, Oliver Hohlfeld, Ralph Holz, David R. Choffnes, Alan Mislove, and Georg Carle. A first look at certification authority authorization (CAA). *Comput. Commun. Rev.*, 48(2):10–23, 2018.

[45] Aleksandrs Slivkins. Introduction to multi-armed bandits. https://arxiv.org/pdf/1904.07272.pdf, 2019.

[46] Rapid7 Project Sonar. Forward dns data. https://opendata.rapid7.com/sonar.fdns_v2/, 2018.

[47] Guanglei Song, Lin He, Zhiliang Wang, Jiahai Yang, Tao Jin, Jieling Liu, and Guo Li. Towards the construction of global ipv6 hitlist and efficient probing of ipv6 address space. In *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*, pages 1–10. IEEE, 2020.

[48] Guanglei Song, Jiahai Yang, Zhiliang Wang, Lin He, Jinlei Lin, Long Pan, Chenxin Duan, and Xiaowen Quan. Det: Enabling efficient probing of ipv6 active addresses. *IEEE/ACM Transactions on Networking*, 2022.

[49] Spamhaus. The spamhaus project6. https://www.spamhaus.org, 2018.

[50] Stephen D Strowes. Bootstrapping active ipv6 measurement with ipv4 and public dns. *arXiv preprint arXiv:1710.08536*, 2017.

[51] Johanna Ullrich, Peter Kieseberg, Katharina Krombholz, and Edgar Weippl. On reconnaissance with ipv6: a pattern-based scanning approach. In *2015 10th International Conference on Availability, Reliability and Security*, pages 186–192. IEEE, 2015.

[52] Kevin Vermeulen, Justin P Rohrer, Robert Beverly, Olivier Fourmaux, and Timur Friedman. Diamond-miner: Comprehensive discovery of the internet's topology diamonds. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 479–493, 2020.

[53] Wikipedia. Hamming distance. https://www.tutorialspoint.com/what-is-hamming-distance, 2022.

[54] Addy Yeowr. Bitnodes api. https://bitnodes.earn.com/, 2018.

A Similarity Definition

In this section, we give the definition of Jaccard similarity and Hamming distance-based similarity.

- **Jaccard similarity:** Jaccard similarity can be used to calculate the similarity between any two sets. The calculation is shown in Formula (1), where U_1 and U_2 are both sets. In particular, if U_1 and U_2 are both empty sets, then $J(U_1,U_2)$ is 0.

$$J(U_1,U_2)=\frac{|U_1\cap U_2|}{|U_1\cup U_2|} \tag{1}$$

- **Hamming distance-based similarity:** For two sequences of the same length z_1 and z_2, their similarity based on Hamming distance is calculated as follows:

$$S_{HD}=\sum_{i=1}^{n} r(z_1[i],z_2[i])$$
$$r(a,b)=\begin{cases}1, & if\ a=b\\ 0, & otherwise\end{cases} \tag{2}$$

B Model Building of *AddrMiner-S*

We use a multi-armed bandit model [45] based on Thompson sampling [43] to dynamically update the density distribution in the active address space to correct the inconsistency in the density distribution between the seeds and the actual active addresses.

We divide the IPv6 address space into different density regions $X=\{x_1,x_2,...,x_k\}$, and each address region is an arm of the multi-armed bandits. There are k actions $A=\{a_1,a_2,...,a_k\}$, and a_i refers to scanning the target address in x_i and probing whether it is an active address, where $i\in[1,k]$. $\Theta=\{\theta_1,\theta_2,...,\theta_k\}$ represents the mean reward. The distribution of each arm reward is the Bernoulli score with Θ as the parameter:

$$P(r|a_i,\theta_i)=\begin{cases}\theta_i, & \text{if } r=1\\ 1-\theta_i, & \text{otherwise}\end{cases} \tag{3}$$

When a_i is played, the action produces a reward of one with probability θ_i, and a reward of zero with probability $1-\theta_i$. The θ_i can be interpreted as an action's success probability or mean reward. Let the agent begin with an independent prior belief over each θ_i. Take these priors to be beta-distributed with parameters $A=\{\alpha_1,...,\alpha_k\}$ and $B=\{\beta_1,...,\beta_k\}$. In particular, for each action a_i, the prior probability density function of θ_i is:

$$P(\theta_i)=\frac{\Gamma(\alpha_i+\beta_i)}{\Gamma(\alpha_i)\Gamma(\beta_i)}\theta_i^{\alpha_i-1}(1-\theta_i)^{\beta_i-1} \tag{4}$$

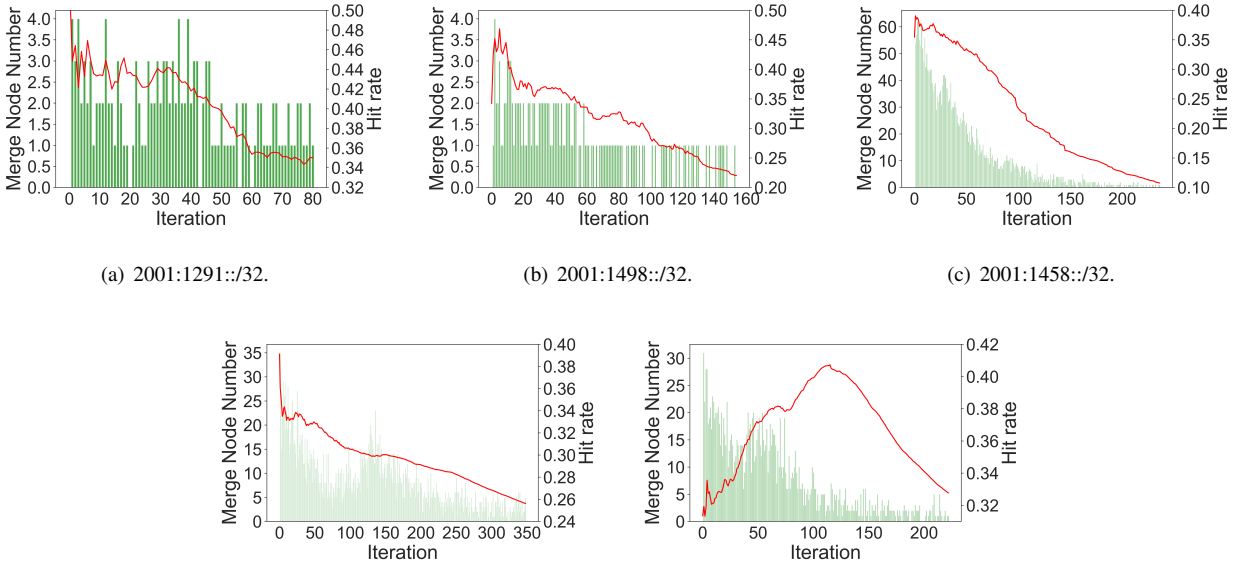
where Γ denotes the gamma function. As observations are gathered, and the distribution is updated according to Bayes's rule. It is particularly convenient to work with Beta distributions because of their conjugacy properties. In particular, each action's posterior distribution is also beta distribution with parameters that can be updated according to a simple rule:

$$(\alpha_i,\beta i)\leftarrow\begin{cases}(\alpha_i,\beta i), & \text{if } a_t\neq i\\ (\alpha_i,\beta i)+(r_t,1-r_t), & \text{otherwise}\end{cases} \tag{5}$$

In other words, we choose region x_i to scan a target address. If we find the target address is active (reward = 1), we will add one to the corresponding α_i (β_i remains unchanged); otherwise (reward = 0), will add one to the corresponding β_i (α_i unchanged). α_i represents the active address probed in region x_i, $\alpha_i+\beta_i$ represents the address budget consumed in region x_i, so the hit rate of active addresses in region x_i is $\frac{\alpha_i}{\alpha_i+\beta_i}$. As in Formula (6), the x_i's active address density is proportional to the hit-rate of active address.

$$x_i.density=\frac{x_i.active\ addresses}{x_i.size}\propto\frac{\alpha_i}{\alpha_i+\beta_i} \tag{6}$$

In active IPv6 address probing, the key issue is to achieve a high active address hit rate within a given budget. Assuming



(a) 2001:1291::/32. (b) 2001:1498::/32. (c) 2001:1458::/32.

(d) 2001:41d0::/32. (e) 2001:470::/32.

Figure 10: Comparisons of the probing efficiency between the *AddrMiner-S* and other address generation algorithms for different budgets in announced prefixes with sufficient seeds.

that our probing budget is B and the objective function of address probing is f, the active address probing is a combinatorial optimization problem of (X, B, f), they need to satisfy the following relationship:

$$\sum_{i=1}^{n}(\alpha_i+\beta_i) \leq B \quad \& \quad \alpha_i+\beta_i \leq x_i.size \tag{7}$$

n represents the number of regions where the target addresses are generated. Our objective function f represents the hit rate of the active address within the target address budget B.

$$f = \frac{\sum_{i=1}^{n} \alpha_i}{\sum_{i=1}^{n} \alpha_i+\beta_i} \tag{8}$$

A feasible solution $x \subseteq X$ that satisfies Formula (7), then the most effective solution x^* is only if $f(x^*) \geqq f(x), \forall x \subseteq X$.

C Seed Number vs. Probing Efficiency

State-of-the-art address generation algorithms learn the structural and distributional characteristics of seeds to generate target addresses that are more likely to survive. However, these techniques are overly dependent on the quality, quantity, and distribution of seeds. Theoretically, seed address-based target address generation algorithms cannot work in target regions with no seeds, nor can they work efficiently in target regions with few seeds. Next, we further explore the impact of the number of seeds on the efficiency of active IPv6 address probing.

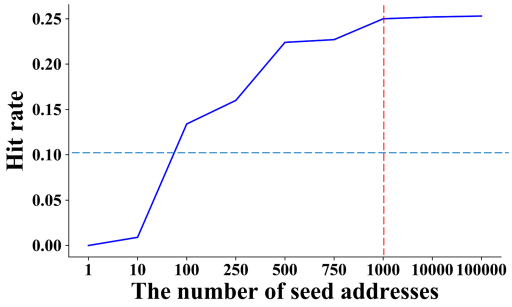


Figure 11: The effect of the number of seeds on the efficiency of active IPv6 address probing.

From the previous measurements, we find that DET [47] has better probing efficiency than 6Tree [33], 6Gen [36], and Entropy/IP [15]. Therefore, we randomly select a different number of active addresses as seeds in any announced prefix with a sufficient number of seeds, and use DET to generate target addresses, with a budget of 10K.

Figure 11 shows the hit rate of active address probing for DET with a different number of seeds. We find that the hit rate of the active address is the lowest when the extreme case of the seed address is 1. At this point, the address probing strategy is the same as 6Tree and fixed-space brute force

Table 6: Overview of our IPv6 Hitlist on September 8, 2021

Name	#IPs	#IPs[1]	#PFXes	#PFXes[2]	#Top AS1	#Top AS2	#Top AS3	#Top AS4	#Top AS5
1d-stable	2.1B	1.7B	86.4K	83.8K	20.40%★	16.39%■	13.20%♦	9.45%★	4.65%►
7d-stable	1.5B	1.1B	85.7K	83.1K	23.41%★	21.48%■	14.44%♦	14.02%★	2.49%■
30d-stable	1.3B	919.4M	80.6K	78.0K	34.96%★	29.75%■	24.05%♦	3.85%★	1.73%■
60d-stable	1.3B	860.2M	80.3K	77.6K	36.74%★	31.83%■	19.62%♦	4.11%★	1.85%■
100d-stable	1.2B	783.7M	80.1K	78.5K	39.58%■	34.93%★	13.58%★	4.52%♦	2.03%■

[1] Removing aliased addresses using aliased prefix detection ★ Amazon, ■ Fastly, ♦ Imperva, ► ChinaTelecom, ★ Cloudflare, ■ Akamai.
[2] Removing aliased prefixes using aliased prefix detection

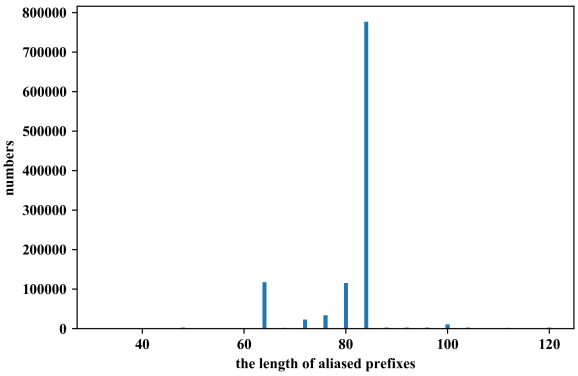


Figure 12: The length distribution of aliased prefixes.

scanning. It builds a hierarchical space tree and prioritizes randomly generates target addresses in low nybble space, so the probing efficiency of this active address is poor (the hit rate is less than 0.01%). As the number of seeds increases, the hit rate of active address probing increases. When the number of seeds in the target region exceeds 100, the active address hit rate exceeds 10%. When the number of seeds exceeds 1000, the hit rate of active addresses remains stable. Therefore, the seed address-based target address generation algorithms are very dependent on seeds and cannot effectively perform global active IPv6 address probing when the number of seeds is insufficient. In this paper, we divide the global active IPv6 address probing task into three sub-tasks according to the number of seeds in the announced prefix spaces, including scenarios with no seeds, scenarios with few seeds, and scenarios with sufficient seeds. We empirically classify the announced prefix spaces. When the number of seeds in the announced prefix space exceeds 1000, we classify the announced prefix space with sufficient seeds. We classify the announced prefix space with few seeds when the number of seeds is not greater than 10. Solutions are designed for each of the above scenarios, effectively solving the global active IPv6 address probing problem.

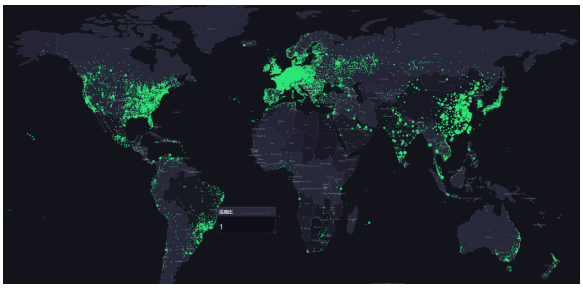


Figure 13: The location distribution of IPv6 hitlist.

D Verification of *AddrMiner-S*

Figures 10(a) to 10(e) show how the hit rate of active addresses and the number of merged nodes vary with the number of reinforcement learning iterations. *AddrMiner-S* can find more optimal address regions by merging nodes to discover active addresses (exploration) and generate target addresses (exploitation) in high-density regions. When the node space is fully explored, the node merging operation can further expand the address search space. During the exploration process, the hit rate of active addresses does not drop suddenly, ensuring the efficiency of active address probing and discovering new high-density regions. Thus, *AddrMiner-S* strikes a balance between exploration and exploitation. In Figure 10(c), the sampling bias of the seeds is small, so the hit rate is less volatile. In Figure 10(e), the sampling bias is obvious. As the number of iterations increases, high-density regions of the actual network are discovered, which effectively improves the efficiency of address probing.

E Aliased Prefix Analysis

During our evaluation process, we found that aliased prefixes profoundly impact the generation of IPv6 addresses. Because the entire prefix is configured on the same network device, when we do not judge the aliased prefix, all addresses in the aliased prefix space will return an ICMPv6 response packet. These addresses will consume probing resources and cause a lot of false active addresses. Therefore, in our probing system, we consciously detect the aliased prefix and remove aliased

addresses.

Figure 12 shows the length distribution of aliased prefixes. We found that the length of aliases prefix is mostly between /64 and /84. We also surprisedly found that 2,685 aliased prefixes are announced prefixes, such as 2401:5e40:8000::/33, assigned to Japan Network Information Center. We further analyzed and found that the target network opened FTP and Telnet ports and took anti-probing measures. Therefore, we inferred that the device where the prefix 2401:5e40:8000::/33 was located could be a honeypot, a decoy network, or a clustered storage system.

The aliased prefix, especially the entire announced prefix as an aliased prefix, may bring the following security problems: First, heuristic active address probing will guide a large number of probes into the aliased prefix space for probing (if there is no ability to remove the aliased prefix), it will cause excessive load on the network device configured with the aliased prefix, and even multiple normal probers will cause DDoS attacks. In addition, it may also affect the performance of the upper network.

F IPv6 Hitlist Introduction

We further analyze the organizational distribution and address location distribution of the IPv6 hitlist in Table 6 and Figure 13, respectively.