# 6Loda: Pattern Filtering and Ensemble Learning for IPv6 Target Generation and Scanning

Xikai Sun[*], Fan Dang[†✉], Zihao Yang[‡], Xinqi Jin[§], Junhao Li[*], Yunhao Liu[*]
[*]Department of Automation and BNRist, Tsinghua University [†]School of Software Engineering, Beijing Jiaotong University
[‡]School of Information Science and Engineering, Yanshan University [§]School of Software, Tsinghua University
{sxk23, jinxq21, lijunhao22}@mails.tsinghua.edu.cn, dangfan@bjtu.edu.cn
yangzihao@stumail.ysu.edu.cn, yunhao@tsinghua.edu.cn

IEEE INFOCOM 2025 - IEEE Conference on Computer Communications | 979-8-3315-4305-1/25/$31.00 ©2025 IEEE | DOI: 10.1109/INFOCOM55648.2025.11044482

***Abstract*—IPv6 target generation is crucial for surveying the vast IPv6 address space, which is essential for network management and IPv6 deployment policies. However, existing techniques often suffer from low hit rates due to ineffective space partitioning caused by outlier addresses and limitations in current outlier removal algorithms. To address these challenges, we propose 6Loda, a novel approach that combines pattern filtering and ensemble learning to efficiently remove outlier addresses and discover active IPv6 addresses. Given a set of known active addresses, 6Loda first employs a pattern-based filter to preliminarily eliminate some outlier addresses. It then utilizes a two-level (divisive hierarchical clustering) DHC algorithm to partition the seed set and applies the Loda algorithm to automatically remove remaining outliers in address spaces. Finally, 6Loda implements the random generation algorithm to produce addresses with high hit rates. Experiments conducted on large-scale datasets demonstrate that 6Loda achieves a $\times 2.26$ improvement in hit rate compared to state-of-the-art methods, while maintaining the same budget constraints.**

***Index Terms*—IPv6 scanning, ensemble learning, target generation algorithm, outlier detection.**

## I. Introduction

The rapid proliferation of internet-connected devices and the exhaustion of IPv4 addresses have necessitated the widespread adoption of IPv6 in recent years [1]. This transition is evident in the increasing utilization of IPv6, with Google reporting that over 30% of its users accessed services via IPv6 as of July 2021 [2]. The expansive IPv6 address space, encompassing $2^{128}$ unique addresses, effectively mitigates the limitations of address scarcity and accommodates the burgeoning demand for a diverse array of internet connectivity, such as LPWAN [3], WLAN [4], TSN [5], software-defined network [6] and video streaming network [7]. As IPv6 adoption continues to accelerate among organizations and end-users, the imperative to comprehend and monitor this vast address space has become paramount. Comprehensive surveying of the IPv6 address space is critical for various aspects of network operations [8].

In methods for surveying address spaces, traditional active scanning techniques have proven highly effective in the IPv4 address space, which is limited to $2^{32}$ addresses. For instance, zmap [9] can complete a scan of the entire IPv4 address space within hours. However, the vastness of the IPv6 address space presents significant challenges for active scanning methodologies. Active scanning necessitates probing each potential IPv6 address to determine its status, but given the enormous number of possible IPv6 addresses, employing such a brute-force approach for a comprehensive scan is impractical. It has been estimated that a complete scan of the entire IPv6 address space using traditional brute-force methods would require millions of years [10]. This estimated timeframe underscores the inefficiency and infeasibility of applying traditional active scanning methods in an IPv6 environment.

To address the limitations of traditional scanning techniques, researchers have developed more efficient methods for surveying the IPv6 address space. One particularly effective approach is the Target Generation Algorithm (TGA). The core process of TGA comprises three main steps:

1) Clustering the existing IPv6 dataset (commonly referred to as seeds) according to specific patterns, thereby partitioning the addresses into various clustered spaces.
2) Within each clustered space, predictively generating potential IPv6 addresses using techniques such as probabilistic statistics and machine learning.
3) Employing testing tools to verify the existence of the generated addresses, thus efficiently identifying active IPv6 addresses within the global IPv6 address space.

In contrast to exhaustive scanning of the entire address space, TGA aims to generate a subset of potentially active new addresses based on observed patterns and behaviors in IPv6 address allocation and usage. This approach significantly narrows the probing range and accelerates the scanning process. The TGA method substantially reduces the time and computational resources required for scanning, rendering it a more viable option for large-scale IPv6 surveys. For IPv6 addresses, the divisive hierarchical clustering (DHC) algorithm and its variants, which construct a space tree from top to bottom, have been demonstrated to be highly efficient clustering algorithms. These algorithms have been widely adopted in related TGA research works, such as 6Hit [11], 6Tree [12], and 6Forest [13].

However, TGA's performance is significantly influenced by the quality of the seed set. Suboptimal seed quality can result in ineffective clustered spaces, thereby substantially reducing the hit rate. This issue primarily stems from the presence of outlier addresses within the seed set, which adversely affect classification efficacy. An outlier address is defined as one that markedly differs from the majority of other addresses within

[✉] Fan Dang is the corresponding author.

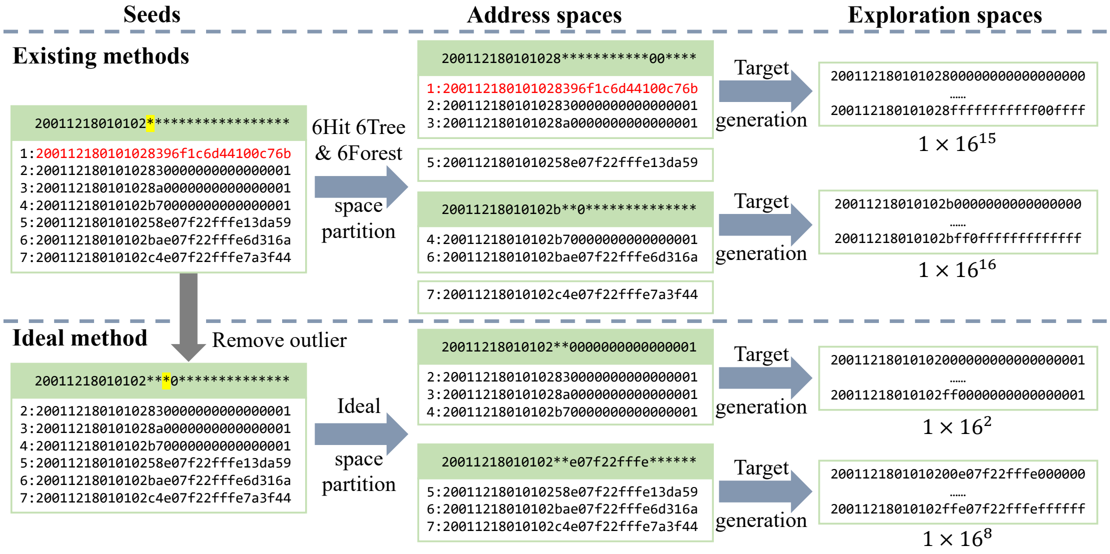
Seeds
Address spaces
Exploration spaces
Existing methods
20011218010102******************
1:20011218010102839 6f1c6d44100c76b
2:20011218010102830000000000000001
3:20011218010102 8a0000000000000001
4:20011218010102b70000000000000001
5:2001121801010258e07f22fffe13da59
6:20011218010102bae07f22fffe6d316a
7:20011218010102c4e07f22fffe7a3f44
6Hit 6Tree
& 6Forest
space
partition
200112180101028**********00****
1:200112180101028396f1c6d44100c76b
2:20011218010102830000000000000001
3:20011218010102 8a0000000000000001
5:2001121801010258e07f22fffe13da59
20011218010102b**0*************
4:20011218010102b70000000000000001
6:20011218010102bae07f22fffe6d316a
7:20011218010102c4e07f22fffe7a3f44
Target
generation
20011218010102800000000000000000
......
200112180101028ffffffffff00ffff
1 × 16^15
20011218010102b0000000000000000
......
20011218010102bff0fffffffffffff
1 × 16^16
Ideal method
Remove outlier
20011218010102***0*************
2:20011218010102830000000000000001
3:200112180101028a0000000000000001
4:20011218010102b70000000000000001
5:2001121801010258e07f22fffe13da59
6:20011218010102bae07f22fffe6d316a
7:20011218010102c4e07f22fffe7a3f44
Ideal
space
partition
20011218010102**0000000000000001
2:20011218010102830000000000000001
3:200112180101028a0000000000000001
4:20011218010102b70000000000000001
20011218010102**e07f22fffe******
5:2001121801010258e07f22fffe13da59
6:20011218010102bae07f22fffe6d316a
7:20011218010102c4e07f22fffe7a3f44
Target
generation
20011218010102000000000000000001
......
20011218010102ff0000000000000001
1 × 16^2
2001121801010200e07f22fffe000000
......
20011218010102ffe07f22fffeffffff
1 × 16^8

Fig. 1. An example of the impact of outlier addresses on address space partition, where the yellow nibble position is the fork point of the address space tree and the red address is the outlier address.

a clustered space, or exists as a unique address within that space. Fig. 1 illustrates this concept: due to the presence of outlier address 1 in the seeds, schemes such as 6Hit, 6Tree, and 6Forest inappropriately partition the parent-node address space into four sub-node addresses. The second and fourth sub-node address spaces, each containing a single unique address, lack a variable nibble dimension. This absence precludes TGA from generating predictions for potential addresses. Conversely, the first and third sub-node address spaces, erroneously generated, possess an exploration space (defined as the exponential of the variable nibble dimension) significantly larger than the optimal partition that would result from excluding outlier addresses. This leads to an excessive expansion of the TGA-predicted address space. Consequently, within a finite number of prediction steps, the hit probability for predicted addresses in the former scenario will be substantially lower than in the latter. In essence, outlier addresses result in incorrect address space partitioning and an increase in the variable nibble dimensions of the address space.

To mitigate this issue, 6Forest incorporates an outlier removal algorithm based on isolation forest, aiming to minimize the dimensions of the final exploration space. For a set of addresses in an address space classified by DHC, this method constructs a depth-limited isolation tree in each free nibble dimension. These trees split and score addresses, with more isolated addresses receiving higher scores. Addresses with cumulative scores exceeding a predetermined threshold are classified as outliers. However, the efficacy of this approach is heavily dependent on the selection of an appropriate threshold and the quality of the initial seed set. Fig. 2 demonstrates how 6Forest generates different sub-node address spaces under varying threshold settings, highlighting the limitations of this threshold-based outlier removal mechanism. This approach requires fine-tuning and is susceptible to dataset-specific influences. Moreover, 6Hit, 6Tree, and 6Forest do not consider the potential impact of pre-filtering addresses with a high probability of being outliers on subsequent DHC space partitioning. Pre-filtering outliers could lead to more optimal space partitioning, as illustrated in Fig.1. The removal of outliers prior to partitioning increases the likelihood of achieving an ideal space partition, as demonstrated in Fig.1.

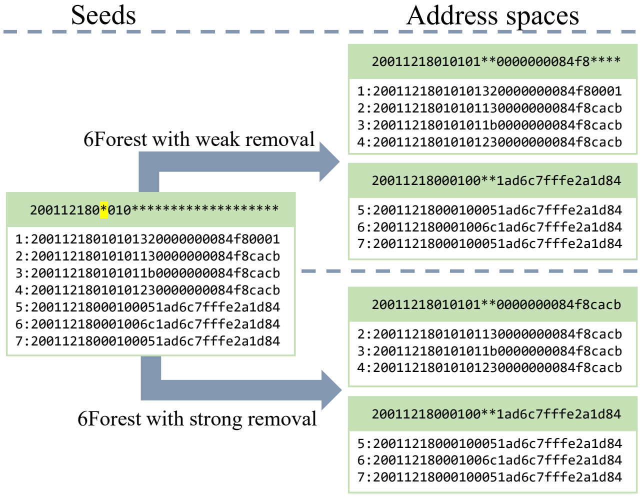
Seeds
Address spaces
6Forest with weak removal
6Forest with strong removal
200112180*010*******************
1:2001121801010132000000084f80001
2:2001121801010113000000084f8cacb
3:200112180101011b000000084f8cacb
4:2001121801010123000000084f8cacb
5:20011218000100051ad6c7fffe2a1d84
6:20011218000100 6c1ad6c7fffe2a1d84
7:20011218000100051ad6c7fffe2a1d84
20011218010101**0000000084f8****
1:2001121801010132000000084f80001
2:2001121801010113000000084f8cacb
3:200112180101011b000000084f8cacb
4:2001121801010123000000084f8cacb
2001121800010 0**1ad6c7fffe2a1d84
5:20011218000100051ad6c7fffe2a1d84
6:200112180001006c1ad6c7fffe2a1d84
7:20011218000100051ad6c7fffe2a1d84
20011218010101**0000000084f8cacb
2:2001121801010113000000084f8cacb
3:200112180101011b000000084f8cacb
4:2001121801010123000000084f8cacb
20011218000100**1ad6c7fffe2a1d84
5:20011218000100051ad6c7fffe2a1d84
6:200112180001006c1ad6c7fffe2a1d84
7:20011218000100051ad6c7fffe2a1d84

Fig. 2. The spatial partitioning result of 6forest varies at different thresholds.

To address the impact of outlier addresses on the TGA hit rate and to overcome the limitations of 6Forest, we propose 6Loda, a novel TGA based on pattern filtering and ensemble learning. 6Loda employs prior knowledge to pre-filter addresses with a high likelihood of being outliers, thereby reducing the probability of misclassification by DHC. To mitigate the effects of threshold fluctuations on outlier address removal, we have developed an adaptive outlier removal algorithm based on the Lightweight On-line Detector of Anomalies. This algorithm generates sparse random projection vectors, projects addresses into low-dimensional spaces, and utilizes histogram density estimation in these spaces to detect and remove outlier addresses. By integrating multiple histogram-based weak detectors, 6Loda achieves robust anomaly detection capabilities.

The contributions of this paper are as follows:

- We have investigated the influence of address generation patterns on DHC space partitioning and elucidated the relationship between these patterns and outlier addresses, which provides novel insights for enhancing TGA design.
- We propose 6Loda, a TGA based on pattern filtering and ensemble learning for IPv6 scanning across the internet. It aims to address the open problem of interference from outlier addresses, even in the most advanced methods. 6Loda ensures performance in outlier address removal without excessively demanding computational resources.
- We have validated the efficacy of 6Loda through extensive experimentation. Results indicate that compared to state-of-the-art methods, 6Loda achieves ×2.26 improvement in the hit rate under the same budgets.

The authors have provided public access to their code at *https://github.com/SunVictor23/6Loda* .

## II. Related work

Due to the non-traversability of the vast IPv6 address space, the work of IPv6 address measurement mainly focuses on two aspects: active scanning based on seed sets and passive collection based on network services.

In active scanning, Barnes et al. firstly hypothesized that seeds provide information that can be used in IPv6 scanning schemes [14], which has become the foundation for subsequent researches [11], [13], [15]–[17]. Research on IPv6 active scanning mainly focuses on two aspects: heuristic-algorithm-based TGAs and machine-learning-based address generation.

Heuristic-algorithm-based TGAs utilize the structural information of seeds to determine scanning regions, then uncover potential IPv6 addresses within these regions. Liu et al. [12] proposed 6Tree, using DHC to partition seeds, generating target addresses based on the density of tree nodes. Hou et al. [11] introduced 6Hit, which used an enhanced DHC and was the first to apply reinforcement learning for dynamic budget allocation. This idea is also adopted in the 6Scan [17]. Hou et al. proposed 6Graph [16], used a DHC similar to 6Hit and leveraged graph theory to discover new addresses. They also introduced 6Forest [13], which partitions seeds using covering-based DHC and incorporates the isolation forest algorithm to removal outlier addresses. To expand the scanning coverage, HMap6 integrates a public seed set with seeds collected under announced BGP prefixes, enabling efficient seed collection across a wide range of BGP prefixes [18]. These TGAs focus on different aspects, aiming to improve the hit rate by improving space partition, outlier removal, potential address prediction and directional correction of predictions.

Machine-learning-based address generation methods attempt to extract semantic information from seeds using machine learning models. These methods treat IPv6 measurement as a special data generation task, e.g., 6VecLM [19], 6GC-VAE [20] and 6GAN [21]. While these methods provide novel solutions, their deep neural network requirements limit large-scale implementation due to high computational costs.

In passive collection, methods aim to extract IPv6 addresses from data packets sent by IPv6 devices when providing network services to them. For instance, [22] leverages the open NTP (Network Time Protocol) servers to passively collect IPv6 addresses from devices requiring NTP time synchronization. The majority of collected addresses are randomized patterns with limited lifespan, typically belonging to end hosts, which have limited significance for IPv6 measurement tasks.

Given the computational constraints of semantic-based models and limitations of passive collection, we opt for a TGA leveraging structural information from seed addresses for IPv6 measurement. As noted in section I, our primary focus is mitigating the impact of outlier addresses on the TGA.

## III. Preliminaries

To help readers understand the IPv6 scanning task and this paper, we will briefly introduce the IPv6 address generation patterns, definitions of the terms used, and the mathematical model of the TGA problem in this section.

### A. IPv6 Address Generation Patterns

IPv6 addresses comprise three components: global routing prefix, local subnet identifier, and interface identifier (IID) [23]. The IPv6 address space, with its potential size of $2^{128}$, offers considerable structural flexibility in the IID, enabling the use of various address generation patterns to accommodate diverse device application requirements. RFC 7707 [24] delineates several IID generation patterns for IPv6 addresses. The main generation patterns are as follows:

- **Low-byte**: The IID primarily sets non-zero nibbles in the low nibbles, e.g., 0000:0000:0000:0001.
- **Randomized**: All nibbles in the IID are pseudo-randomly generated, e.g., 96f1:c6d4:4100:c76b.
- **Embedded-ipv4**: The lowest 32 bits or each field of the IID embeds the IPv4 address, e.g., 192.168.2.10 can be mapped to 00c0:00a8:0002:000a or 0192:0168:0002:0010.
- **Pattern-bytes**: IIDs within the same prefix use specific bytes, e.g., 0113:0000:84f8:cacb and 011b:0000:84f8:cacb.
- **Ieee-derived**: The device's MAC address is mapped to an IPv6 address, characterized by the 23rd to 26th nibbles being fffe, e.g., 51ad:c7ff:fe2a:1d84.
- **Embedded-port**: The lowest nibbles of the IID embed the device's port number. For example, port 80 can be mapped to 0000:0000:0000:0080.

It is important to note that while the IIDs described above are assumed to be 64 bits in length, the actual length of the IID may vary across different autonomous systems.

According to the statistics in the [22], core IPv6 internet infrastructure, such as servers and routers, primarily uses address generation patterns with low randomness, like low-byte, to facilitate network administration and asset management. In contrast, end host addresses more frequently adopt randomized patterns to enhance user privacy and prevent address tracking.

### B. Definitions

*1) Definition 1:* **Seed.** The TGA starts with an actual dataset of IPv6 addresses to discover new addresses. These active IPv6 addresses in the dataset, used as starting points, are referred to as seeds. The dataset consisting of all seeds is denoted as $S$, and the number of seeds is denoted as $|S|$, where $|\cdot|$ denotes the operation of counting the number of elements in the set. Generally, seeds are IPv6 addresses actually used by internet devices in reality, and their validity can be tested through someone protocols such as ICMPv6, TCP/80, TCP/443, UDP/80, and UDP/443.

*2) Definition 2:* **Address space.** The seeds in the dataset are partitioned into small seed clusters according to certain rules, where the seeds in each cluster have similar structural information. Such a cluster is referred to as an address space. An address space with only one seed has no variable nibbles and cannot guide the generation of potential addresses, and is typically discarded. The $m$ address spaces partitioned from the seed set $S$ are denoted as $S_i$, $i = 1, ..., m$.

*3) Definition 3:* **Variable nibble dimension.** When an IPv6 address is represented in nibble form, it has 32 dimensions, with each dimension taking values in the range [0x0,0x$f$]. If the seeds in a particular address space take different values in one nibble dimension, that dimension is considered as a variable nibble dimension. In this paper, we use "$*$" to represent a variable half-byte dimension.

*4) Definition 4:* **Outlier address.** Typically, if an address in an address space is significantly different from the majority of other addresses or if there is a unique address in the address space, that address is considered an outlier address.

*5) Definition 5:* **Exploration space.** For the addresses in an address space, all potential addresses generated by varying the values of the variable nibbles with different structural information constitute the exploration space.

*6) Definition 6:* **Potential Address.** The addresses predicted by the TGA from the exploration space are referred to as potential addresses. These addresses can be verified for existence using address scanning tools such as zmap v6.

*7) Definition 7:* **Budget.** The required number of potential addresses is refer to as budget, denoted as $b$ in this paper.

It should be noted that, for convenience, all IPv6 addresses are represented using 32 nibbles in this paper. All metrics involving address lengths refer to the length in nibbles.

### C. Mathematical Modeling of the TGA

Without loss of generality, the TGA problem requires the algorithm $\tau$ to predict a set of potential addresses $\tau(S, b)$ based on a given seed dataset $S$ and a given budget $b$. Assuming the set of all active IPv6 addresses in the real world is $A$, the TGA problem can be formulated as an optimization problem:

$$\max H(\tau, S, b) = \frac{|\tau(S, b) \cap A - \tau(S, b) \cap S|}{|\tau(S, b)|}, \qquad (1)$$

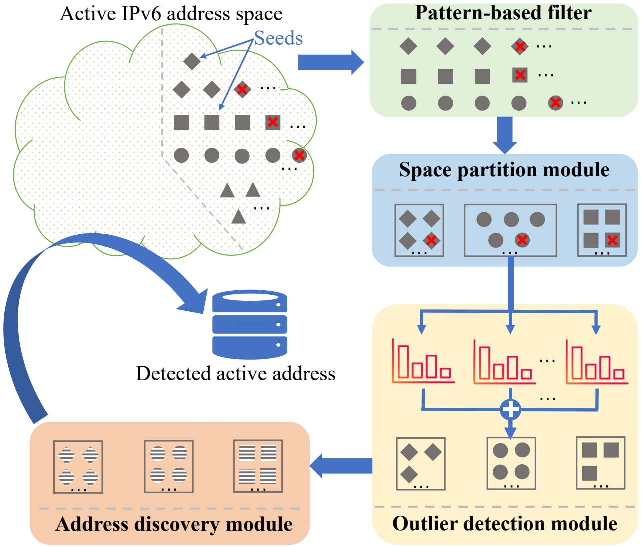

Fig. 3. 6Loda overview. Points of different shapes represent seeds with different generation patterns. Points with '×' represent the outlier addresses.

where $\tau(S, b) \cap A$ represents the portion of predicted addresses that are actually active in the real world, which can be obtained by using existing IPv6 tools to test $\tau(S, b)$. The part subtracted in the numerator means that the generated targets should exclude the known seeds. $H(\tau, S, b)$ can also be defined as the hit rate of generated IPv6 addresses.

## IV. Design of 6Loda

In this section, we first introduce the overview of 6Loda. Then, we describe the design details of each key module of 6Loda, including the pattern-based filter, space partition module, outlier detection module, and Address discovery module.

### A. System Overview

Fig. 3 depicts the primary workflow of 6Loda. The algorithm initiates by identifying address generation patterns within the seed set. Based on empirical observations, 6Loda eliminates addresses exhibiting randomized and ieee-derived patterns, as these are highly probable outliers. Subsequently, a two-level DHC algorithm, leveraging generation patterns and max-covering principles, iteratively partitions the filtered seed set into distinct address spaces. The Loda algorithm is then applied to each address space to detect and remove potential outlier addresses. Finally, 6Loda employs an address discovery module to predict potential addresses within the exploration space corresponding to these address spaces, validating their existence through address scanning tools.

### B. Pattern-based Filter

**Insight**: addresses generated using randomized and ieee-derived patterns are more likely to be outlier addresses, which are not conducive to address generation.

IPv6 address generation follows specific patterns as outlined in section III-A, providing a foundation for investigating IPv6 classification methodologies. Analysis of address pattern

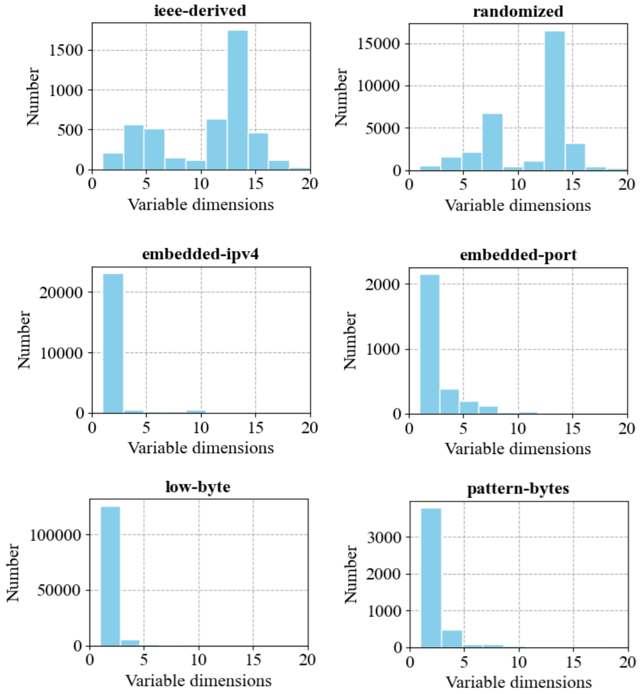

Fig. 4. The histogram of variable nibble dimensions of address spaces for each address generation pattern. The horizontal axis represents the number of variable nibble dimensions, the vertical axis represents the corresponding number of address spaces, and the title represents generation patterns.

characteristics in a large-scale seed set revealed that addresses generated under different patterns result in exploration spaces of varying sizes after space partitioning. The size of these exploration spaces strongly correlates with hit performance [25]. Experiments were conducted using an IPv6 Hitlist containing 1 million seeds [26]. The distribution of generation patterns in this dataset was as follows: low-byte (61.94%), randomized (16.37%), embedded-ipv4 (14.53%), pattern-bytes (2.74%), ieee-derived (2.46%), and embedded-port (1.96%). Further experimental details will be presented in section V-A. The variable nibble dimensions of address spaces obtained after applying the covering-based DHC algorithm for space partitioning were analyzed for each generation pattern. The results of this analysis are illustrated in Fig. 4.

The analysis reveals that both randomized and ieee-derived generation patterns exhibit a substantial proportion of address spaces with high variable dimensions. Specifically, in the randomized pattern, 61.89% of address spaces demonstrate variable dimensions exceeding 7, while in the ieee-derived pattern, this percentage is 53.64%. A variable dimension surpassing 7 indicates that the number of scannable IPv6 addresses within a single exploration space can reach or exceed $16^8 = 2^{32}$, which is equivalent to the total upper limit of IPv4 addresses. Furthermore, these elevated variable dimensions suggest a heightened probability of outlier address occurrence. This is because addresses within the same address space exhibiting high variable dimensions indicate significant heterogeneity at these positions, potentially leading to the classification of all addresses as outliers.

Based on the above analysis, we can believe that addresses with randomized and ieee-derived generation patterns can be considered outliers. Retaining these addresses for address generation is more detrimental than beneficial. For IPv6 address, directly discarding them is a highly efficient strategy. Given the abundance of seeds, the discarded seeds are insignificant compared to the vast IPv6 address space, and we need not worry about the value of these discarded seeds in discovering more active IPv6 addresses. Therefore, we first designed a pattern-based filter, which utilizes the IPv6 Toolkit [27] to identify generation patterns and remove addresses with randomized and ieee-derived patterns.

**Remark**: For datasets that contain too many seeds with randomized or ieee-derived patterns, we propose the application of alternative TGA specifically to these seeds. However, such an approach falls outside the purview of the present study.

### C. *Space Partition Module*

The standard DHC algorithm begins with a complete seed set and iteratively partitions the parent-node address space into sub-node address spaces, starting from the leftmost variable dimension. The iteration stops when the sub-node spaces meet certain threshold requirements. Empirical evidence suggests that DHC algorithms generally outperform agglomerative hierarchical clustering (AHC) algorithms [11], [12], [28]. This superiority is attributed to two main factors: firstly, AHC algorithms necessitate a priori specification of cluster numbers, which is often impractical for large seed sets; secondly, the computational complexity of AHC algorithms varies between $O(nlogn)$ and $O(n^2)$ depending on the clustering method, whereas DHC algorithms and their variants consistently maintain an $O(nlogn)$ complexity, rendering them more scalable for large-scale seed sets [17].

We designed a two-level DHC algorithm for space partition, capitalizing on the superior performance of DHC algorithms. The first level classifies seeds by their generation patterns, which is efficient and prevents misclassification of addresses. This approach aligns with the assumption that servers for a website likely have IPv6 addresses generated using the same pattern. This initial classification achieves efficient preliminary space partition and can be integrated with the pattern-based filter described in Section IV-B, allowing for more refined DHC on addresses with specific patterns.

In the second level, we employ the covering-based DHC proposed by 6Forest to partition seed subsets for each generation pattern. Unlike the original DHC's left-to-right splitting order, which fragments address spaces (Fig. 5), the covering-based DHC splits from the least "chaotic" position among variable nibble dimensions. This approach minimizes the number of unique address spaces generated after each split by comparing the covering of all variable nibbles. For $K$ seeds to be split, with $V_i$ as the vector of variable nibbles in the $i-th$ dimension, the covering $Cover_i$ is defined as:

$$Cover_i = \frac{\Sigma_{j=0,|V_i==j|>1}^{15} |V_i == j|}{K}, \qquad (2)$$

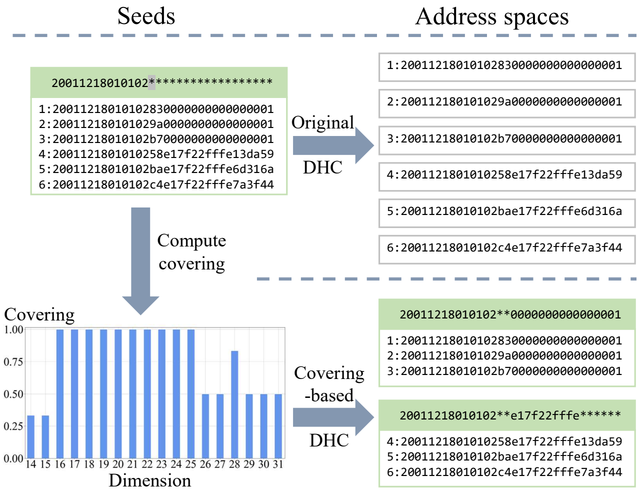
Seeds
Address spaces
20011218010102******************
1:20011218010102830000000000000001
2:200112180101029a0000000000000001
3:20011218010102b70000000000000001
4:2001121801010258e17f22fffe13da59
5:20011218010102bae17f22fffe6d316a
6:20011218010102c4e17f22fffe7a3f44
Original
DHC
1:20011218010102830000000000000001
2:200112180101029a0000000000000001
3:20011218010102b70000000000000001
4:2001121801010258e17f22fffe13da59
5:20011218010102bae17f22fffe6d316a
6:20011218010102c4e17f22fffe7a3f44
Compute
covering
Covering
1.00
0.75
0.50
0.25
0.00
14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Dimension
Covering
-based
DHC
20011218010102**0000000000000001
1:20011218010102830000000000000001
2:200112180101029a0000000000000001
3:20011218010102b70000000000000001
20011218010102**e17f22fffe******
4:2001121801010258e17f22fffe13da59
5:20011218010102bae17f22fffe6d316a
6:20011218010102c4e17f22fffe7a3f44

Fig. 5. Examples of the original DHC and the covering-based DHC.

where $|V_i == j| > 1$ ensures that we focus on the non-unique nibbles in the variable nibble dimensions during the splitting process, and the denominator $K$ is used for normalization. $Cover_i$ indicates that the fewer unique nibbles there are in the current variable dimension, the lower the likelihood of generating address spaces with unique addresses when splitting in that dimension, as shown in Fig. 5.

The complete pseudocode for the two-level DHC is shown in Alg. 1, where `GetPattern()` is used to obtain the address pattern and can be parsed using the RFC tools mentioned in section IV-B. The covering-based DHC is detailed in [13], and will not be reiterated here. In terms of complexity, since the complexity of RFC parsing is $O(1)$ and can be incorporated into section IV-B, the complexity of the two-level classification is mainly determined by covering-based DHC. If $n$ seeds are used for address space partition, the worst-case time complexity of the two-level DHC is $O(nlogn)$; leveraging the FIFO data structure in covering-based DHC, the space complexity can be achieved at $O(n)$.

### D. Outlier Detection Module

Although we filter out addresses that are highly likely to be outliers using the pattern-based filter and partition the other seeds, this does not solve the problem of exploration space inflation caused by the remaining unfiltered outlier addresses. Therefore, in this section, 6Loda employs Loda [29] to further filter out outlier addresses from the address space. This algorithm uses sparse random projections to reduce dimensionality, then applies histogram density estimation for anomaly scoring and outlier removal. Multiple histogram-based weak detectors are combined for robust anomaly detection.

We can assume that outlier addresses have unique nibbles in most variable nibble dimensions, allowing them to be isolated or detected first. This assumption underlies all outlier removal algorithms and is also accepted in 6Loda.

For $n$ seeds $\{x_i\}_{i=1}^n$ in an address space, where $x_i \in N(0,15)^{32}$, 6Loda first randomly generates $k$ sparse projection vectors $\{\omega_j\}_{j=1}^k$, with any element in $\omega_j$ following $N(0,1)$. The projection vector $\omega_j$ projects the sample $x \in \{x_i\}_{i=1}^n$ onto a one-dimensional space $H_j$, with the corresponding projection value being $z_j = \omega_j^T x$. For each projection vector of $\{\omega_j\}_{j=1}^k$, we can construct a one-dimensional histogram $h_j$ for the projection values of all samples. If we assume that the probability estimate of sample $x$ in histogram $h_j$ is $\hat{p}_i(\omega_j^T x)$, the anomaly score of the sample can be written as:

**Algorithm 1** Space partition

```
Require: the seed set $S$ after pattern-based filtering, the seed number threshold $\beta$ in final address space
Ensure: the set of final address space $\phi$
Initialize an empty set $\phi$
Initialize an empty dictionary $patternDict$
for each $seed$ in $S$ do
    $pattern$ = GetPattern($seed$)
    if $pattern$ not in $patternDict$ then
        $patternDict[pattern]$ = []
    end if
    $patternDict[pattern]$.append($seed$)
end for
for each $pattern$ in $patternDict$ do
    $S_i$ = $patternDict[pattern]$
    $results$ = coveringDHC($S_i, \beta$)
    $\phi$.update($results$)
end for
return $\phi$
```

$$f(x) = -\frac{1}{k}\sum_{j=1}^{k} \log \hat{p}_i(\omega_j^T x). \tag{3}$$

If the probability estimate $\hat{p}_i(\omega_j^T x)$ of sample $x$ in a certain projection direction is very low, then $\log \hat{p}_i(\omega_j^T x)$ will be a large negative number, thereby increasing the anomaly score $f(x)$. By performing multiple projections, $f(x)$ will integrate information from multiple projection directions. If sample $x$ has low probability density in several projection directions, $f(x)$ will be high, indicating that the sample is anomalous. Additionally, the Eq. 3 can be reformulated as:

$$\begin{aligned} f(x) &= -\log \left( \prod_{i=1}^{k} \hat{p}_i(\omega_j^T x) \right)^{\frac{1}{k}} \\ &= -\log \hat{p}(\omega_1^T x, \omega_2^T x, ..., \omega_k^T x), \end{aligned} \tag{4}$$

where $\hat{p}(\omega_1^T x, \omega_2^T x, ..., \omega_k^T x)$ is the joint probability of the projections. This equation demonstrates that the anomaly score is inversely proportional to the logarithm of the sample's likelihood. Consequently, samples with lower likelihood values are assigned higher anomaly scores, which is consistent with the fundamental principle of anomaly detection. This relationship provides a quantitative basis for identifying outliers or unusual observations within the dataset.

The complete pseudocode for the outlier detection module is shown in Alg. 2, where `getOutlierScore()` calculates the anomaly score of the input address according to Eq. 3, and

`Sort()` sorts the address data structure in descending order of values. In this algorithm, two parameters must be determined: projection vectors $\{\omega_j\}_{j=1}^{k}$ and number of histogram bins $\hat{b}$.

**Algorithm 2** Outlier detection

```
Require: address space $S_i$, outlier ratio $\alpha$, $b$ bins for histograms
Ensure: address space $\bar{S}_i$ after removing outliers
Initialize $k$ projection vectors $\{\omega_j\}_{j=1}^{k}$
Initialize $k$ empty histograms $\{h_j\}_{j=1}^{k}$
Initialize outlier scores dictionary $O$
Initialize address space $\bar{S}_i$ after removing outliers
for each $\omega_j$ in $\{\omega_j\}_{j=1}^{k}$ do
    for each $x$ in $S_i$ do
        $h_j.\texttt{add}(\omega_j^T x)$
    end for
end for
for each $x$ in $S_i$ do
    $O[x] = \texttt{getOutlierScore}(x, \{h_j\}_{j=1}^{k})$
end for
$num\_outliers$ = $\texttt{Round}(\alpha \times |S_i|)$
$O = \texttt{Sort}(O)$
$keys = \texttt{List}(O.\texttt{keys}())$
$\bar{S}_i = keys[num\_outliers : end]$
return $\bar{S}_i$
```

**Generate sparse projection vectors**. The original Loda algorithm recommends that each sparse projection vector randomly selects $\gamma = \left\lceil 32^{\frac{1}{2}} \right\rceil = 6$ non-zero features, with each non-zero feature randomly sampled from $N(0,1)$. By using multiple sparse random projections, the data can be projected into diverse sub-spaces, thereby increasing the diversity of histograms. This technique is also used in algorithms such as random forests [30].

**Determine the number of histogram bins**. For the number of bins $\hat{b}$ in an equal-width histogram, a simple method to determine $\hat{b}$ is suggested in [31], which maximizes the following penalized maximum likelihood:

$$\sum_{i=1}^{\hat{b}} n_i \log \frac{\hat{b} n_i}{n} - [\hat{b} - 1 + (\log \hat{b})^{2.5}], \tag{5}$$

where $n_i$ is the number of samples falling into the $i-th$ bin, and $[\hat{b} - 1 + (\log \hat{b})^{2.5}]$ prevents having too many bins. In 6Loda, due to the large number of address spaces and the limited number of addresses in each space, 6Loda sets a fixed number of bins to maintain precision (e.g., $\hat{b} = 10$).

**Remark 1: Complexity analysis.** The main computational cost of the outlier detection module lies in sample projection and histogram statistics. The time complexity is $O(nk\gamma)$, where $n$ is the number of seeds in the current address space, $k$ is the number of histograms, and $\gamma$ is the number of non-zero features in the random projection vectors.

**Remark 2: Online update.** 6Loda's histogram-based statistics allow for online updates. After outlier removal, newly discovered IPv6 addresses can be added to the address space, further updating the histograms, enhancing 6Loda's ability to remove some stubborn outlier addresses.

### *E. Address Discovery Module*

The address discovery module performs two key functions: generating and testing potential addresses. Generating involves sampling the exploration space derived from the address space according to specific rules. Testing utilizes scanning tools to verify the existence of generated addresses. As noted in section IV-A, 6Loda prioritizes efficient outlier removal over address generation, eschewing multi-round directional adjustments [11] for higher hit rates. Consequently, 6Loda adopts a static scanning strategy similar to 6Forest. Specifically, 6Loda's prototype implements a single-step static scanning algorithm named the random generation algorithm, which is also adopted by TGAs such as 6Graph and 6Forest. Since the algorithm used in those TGAs has not been open-sourced, we have implemented our version of the random generation algorithm based on the random generation concept in 6Forest. The algorithm samples the address space at the particular variable nibble dimensions, generating a specified number of potential addresses within the budget.

The pseudocode for the random generation algorithm is shown in Alg. 3, where `GetVariableDimension()` is used to obtain the variable nibble dimensions from a given address space; `RandomChooseSeed()` is used to randomly choose one seed from a given address space; `RandomChooseIndex()` is used to randomly choose one dimension from the variable nibble dimensions. In the algorithm, each sampling only modifies one nibble, so each variable nibble dimension can be sampled at most 16 times.

**Algorithm 3** Random generation

```
Require: address space $S_i$, budget $b_i$
Ensure: generated address space $P_i$
Initialize the list of variable nibble dimensions $indexes$
Initialize the set of generated address space $P_i$
$indexes$ = $\texttt{GetVariableDimension}(S_i)$
$max\_gen\_number$ = $16 \times \texttt{len}(indexes) - \texttt{len}(S_i)$
$count = 0$
while $count < b_i$ and $count < max\_gen\_number$ do
    $seed$ = $\texttt{RandomChooseSeed}(S_i)$
    $index$ = $\texttt{RandomChooseIndex}(indexes)$
    $seed[index]$ = $\texttt{Str2Hex}(\texttt{randint}(0, 15))$
    if $seed$ not in $S_i$ and $seed$ not in $P_i$ then
        $P_i.\texttt{append}(seed)$
        $count \leftarrow count + 1$
    end if
end while
return $P_i$
```

The budget allocation is proportionally distributed according to the number of seeds in different address spaces. For the $m$

address spaces $[S_1, S_2, ..., S_m]$, when the total budget is $b$, the budget allocated to each address space $b_i$ is:

$$b_i = \left\lceil \frac{|S_i|}{\sum_{k=1}^{m} |S_k|} \cdot b \right\rceil, i = 1, ..., m, \tag{6}$$

where rounding up ensures that each address space is probed at least once, thereby maintaining compatibility with subsequent expansion algorithms. We cannot abandon the exploration of an address space simply just because it has a small number of seeds and a limited budget; what if this address space has a higher density of active addresses? The excess budget resulting from rounding up does not exceed $m$, and in large-scale IPv6 scanning, usually $m \ll b$.

After obtaining the potential addresses, 6Loda uses zmap to test potential addresses, which scans potential addresses using various protocols or ports.

## V. Evaluation

In this section, we present our experimental setup, all experimental results, and analyses to demonstrate the superiority.

### A. Experimental Setup

*1) dataset:* The seed datasets used in this paper are all public datasets, as follows:

- **C1**: IPv6 Hitlist 1M. An open-source dataset provided by Gasser et al., aimed at creating a comprehensive list of active IPv6 addresses, incorporating results from multiple public address sets and updating daily. We selected the probing results from May 4, 2024, to May 10, 2024. This dataset contains a total of 24.5 million addresses, from which we selected the top 1 million addresses as C1.
- **C2**: Cisco Umbrella Websites [32]. An open-source domain dataset released by the Cisco Umbrella on May 10, 2024. We selected the top 9000 active addresses as C2.
- **C3**: Zone Files for Several Top-Level Domains [33]. Zone files provided by ICANN for top-level domains, containing DNS records.We selected 630 IPv6 addresses from the AAAA records in top 100 domain files as C3.

The datasets contain IPv6 addresses of servers. To ensure seed validity, we conducted zmap tests on all datasets, retaining only active IPv6 addresses as seeds for subsequent experiments. All datasets underwent these validity tests offline in advance to ensure fair performance comparisons.

*2) Baseline:* Since 6Loda focuses on efficiently removing outlier addresses, we selected 6Forest, which also addresses this issue, as the baseline. 6Forest represents the state of the art in this area. As 6Forest has only released the code for space partitioning and outlier detection, to ensure fairness, we used the address discovery module described in Section IV-E to generate new addresses in both systems.

*3) Evaluation Metrics:* The evaluation metrics selected in this paper include the hit rate of predicted addresses and the removal rate of outlier addresses, described as follows:

- **Hit rate**: the hit rate is the proportion of potential addresses that are active in the physical world. Its mathematical definition is given by $H(\tau, S, b)$ in section III-C.

TABLE I
Experimental results of 6Lora and 6Forest

| Seed | | 6Forest | | 6Loda | |
|---|---|---|---|---|---|
| set | Budget | $R$ | $H$ | $R$ | $H$ |
| C1 | $50M$ | 15.64% | 5.02% | 34.42% | **11.34%** |
| C2 | $450k$ | 47.43% | **73.84%** | 52.32% | 66.52% |
| C3 | $31.5k$ | 13.65% | 24.31% | 25.87% | **25.48%** |

- **Removal rate**: the removal rate is the proportion of outlier removed by the TGA relative to the seed set. Although the TGA cannot guarantee that the removed addresses are indeed outliers, we believe this metric can partly reflect TGA's sensitivity to address heterogeneity . Its mathematical definition is as follows:

$$R(\tau, S, b) = 1 - \frac{\bigcup_{i=1}^{m} \bar{S}_i}{S}, \tag{7}$$

where $\bar{S}_i, i = 1, \cdots, m$ are the address spaces returned after outlier removal by the Alg. 2.

### B. Analysis of Experimental Results

All experiments were conducted on a Linux server equipped with an AMD Ryzen 9 5950X 16-Core processor. To ensure consistency and control, all experiments were executed using a single thread. For the IPv6 internet scans, the scanning rate of zmap was strictly limited in accordance with the internet citizenship guidelines proposed by Partridge and Allman [34], ensuring rigorous adherence to ethical scanning practices. All experiments were repeated 10 times, and the averages were calculated as the final results to ensure fairness.

*1) Performance:* We first compared the performance of 6Loda and 6Forest on different datasets. The final experimental results are shown in Tab. I. 6Loda shows comparable performance to 6Forest on small-scale datasets but demonstrates superior results on large-scale datasets. For example, on C1, 6Loda's hit rate is 2.26 times that of 6Forest. The Loda algorithm's performance in 6Loda is based on probability density statistics for outlier detection. On small-scale datasets, statistical distortion can occur due to the limited seeds, affecting 6Loda's performance. However, as the seeds increases in larger datasets, the advantages of statistical analysis become more apparent, resulting in 6Loda's superior performance.

**Remark**: It is important to note that the budget ceiling is set at 50 times the number of seeds. This is because, as pointed out in [13], when the variable nibble dimension does not exceed 3, full scanning is a more efficient and rewarding choice. Therefore, when the variable nibble dimension does not exceed 3, a budget of 50 times is sufficient for the random generation algorithm to toggle a variable nibble to any number in $[0x0, 0xf]$ while keeping the other variable nibbles unchanged (i.e., $15 \times 3 = 45 < 50$). Additionally, when the exploration space is limited, the allocated budget may not be fully utilized. Therefore, only a budget ceiling is set, and the actual budget consumed during the experiments is less than the budget ceiling, ranging from approximately $10\%$ to $40\%$ across different datasets.

To more intuitively observe the performance variations of 6Loda under different prediction steps, we visualized the cumulative hit rate results from the zmap tests, as shown in Fig. 6. It can be observed that the hit rates of both schemes decrease during the scanning process, but the performance of 6Loda remains consistently higher than that of 6Forest overall.

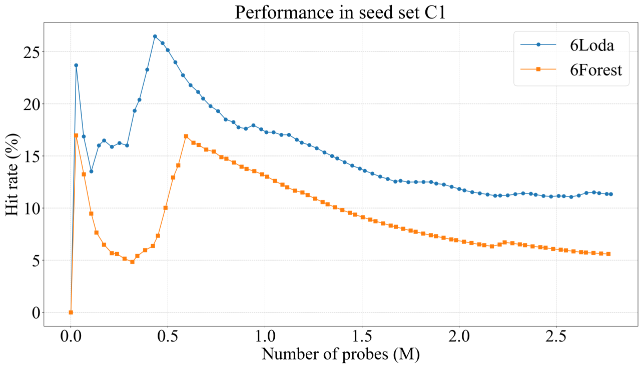

Fig. 6. The cumulative hit rates of 6Lora and 6Forest on C1.

*2) Ablation Studies:* To verify the effectiveness of each module described in section IV, we conducted ablation studies on the C1. To clearly present the results of the ablation studies, we refer to the content of Section IV and decompose 6Loda into four components, naming them as follows:

- **M1**: Pattern-based filter;
- **M2**: Space partition module using the two-level DHC;
- **M3**: Outlier detection module based on the Loda;
- **M4**: Address discovery module.

For any TGAs, space partition and address discovery are essential components. Therefore, in the ablation studies, the absence of the M2 implies that this module uses the default covering-based DHC. The experimental results on the C1 are shown in Tab. II, where covering-based DHC refers to the scheme consisting solely of the covering-based DHC algorithm and address discovery, without removing any seeds.

TABLE II
ABLATION STUDIES OF 6LODA ON THE C1

| **TGA** | $R$ | $H$ |
|---:|:---:|:---:|
| Covering-based DHC | 0 | 2.37% |
| 6Forest | 15.64% | 5.02% |
| M3+M4 | 24.01% | 3.12% |
| M1+M3+M4 | 19.40% | 8.79% |
| M1+M2+M4 | 18.87% | 8.47% |
| M2+M3+M4 | 24.01% | 3.75% |
| 6Loda: M1+M2+M3+M4 | 34.42% | **11.34%** |

The results indicate that the complete 6Loda achieves the highest hit rate. The absence of any module leads to a performance degradation. Furthermore, the presence of the M1 ensures that even an incomplete 6Loda outperforms 6Forest, highlighting the superiority of pre-filtering address patterns.

*3) Portability Analysis:* To further investigate whether these modules are portable and can potentially be used to improve other TGAs, we take 6Forest as an example and embed the modules of 6Loda into the 6Forest for experimentation. The original 6Forest consists of a space partition module using the covering-based DHC, an outlier removal module based on the isolation forest algorithm, and an address discovery module using the random address generation algorithm (i.e. M4). Among these, the second module is the core of 6Forest, and in principle, cannot be modified. Therefore, modules that can be embedded into 6Forest only include M1 and M2. The experimental results on the C1 are shown in Tab. III.

TABLE III
EXPERIMENTAL RESULTS OF 6FOREST EMBEDDED MODULES IN C1

| **TGA** | **Removal rate** | **Hit rate** |
|:---|:---:|:---:|
| 6Forest | 15.64% | 5.02% |
| 6Forest+M1 | 12.61% | **7.56%** |
| 6Forest+M2 | 11.11% | 3.37% |
| 6Forest+M1+M2 | 27.06% | **9.64%** |

It can be observed that the M1 can effectively improve the hit rate of 6Forest, indicating that it has good portability. Although directly introducing the M2 leads to a performance decline in 6Forest, jointly introducing the M1 and M2 can further enhance the hit rate, surpassing the performance achieved with only M1. This improvement is due to the strict dependency of the M2 on the M1. This dependency relationship is also reflected in the ablation studies in Section V-B2.

## VI. CONCLUSION

In this work, we illustrate the impact of outlier addresses and address generation patterns on the effectiveness of IPv6 target generation. To address these problems, we propose 6Loda, an approach based on pattern filtering and ensemble learning, which efficiently removes outlier addresses and discovers active IPv6 addresses. 6Loda uses a pattern-based filter to preliminarily filter out some outlier addresses, employs a two-level DHC algorithm to partition the seed set, utilizes the Loda algorithm to automatically remove outlier addresses in address spaces, and uses the random generation algorithm to generate addresses with high hit rates. Experiments on large-scale datasets have shown that compared to the state-of-the-art solution 6Forest in IPv6 outlier removal, 6Loda can achieve higher address hit rates under the same budgets. Meanwhile, our experiments also demonstrate that the modules developed in 6Loda are portable and can be integrated into other TGAs to enhance their performance.

## ACKNOWLEDGMENT

This work is supported in part by the National Key R&D Program of China under grant No. 2024YFC2607400, the Natural Science Foundation of China under Grant No. 62302259, 62202263, 62232004, and Tsinghua University-Fuzhou Joint Institute for Data Technology.

# References

[1] S. Pack, X. Shen, J. W. Mark, and J. Pan, "Adaptive route optimization in hierarchical mobile ipv6 networks," *IEEE transactions on mobile computing*, vol. 6, no. 8, pp. 903–914, 2007.

[2] Google, "Ipv6 adoption statistics," https://www.google.com/intl/en/ipv6/statistics.html, 2021.

[3] W. J. TONG Shuai, "Progress and challenges of lora low power wide area networks," *Acta Electronica Sinica*, vol. 52, no. 10, pp. 3623–3642, 2024.

[4] Y. Zhu, D. Zhu, G. Pan, K. Chi, and Y. Li, "Using chain of mobile access gateway to reduce delay for pmipv6 protocol applied in wlan," *Chinese Journal of Electronics*, vol. 26, no. 5, pp. 1032–1040, 2017.

[5] Y. Zhao, Z. Yang, X. He, J. Wu, H. Cao, L. Dong, F. Dang, and Y. Liu, "E-tsn: Enabling event-triggered critical traffic in time-sensitive networking for industrial applications," in *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*, 2022, pp. 691–701.

[6] J. Zhang, F. Zhu, Z. Yang, C. Chen, X. Tian, and X. Guan, "Routing and scheduling co-design for holistic software-defined deterministic network," *Fundamental Research*, vol. 4, no. 1, pp. 25–34, 2024.

[7] C. Qiao, J. Wang, and Y. Liu, "Beyond qoe: Diversity adaptation in video streaming at the edge," *IEEE/ACM Transactions on Networking*, vol. 29, no. 1, pp. 289–302, 2021.

[8] R. C. Meena and M. Bundele, "A review on implementation issues in ipv6 network technology," *Ramesh,#, Meena, C., & Bundele, M.(2015). A Review on Implementation Issues in IPv6 Network Technology. International Journal of Engineering Research and General Science*, vol. 3, no. 6, pp. 800–809, 2015.

[9] Z. Durumeric, E. Wustrow, and J. A. Halderman, "Zmap: Fast internet-wide scanning and its security applications," in *22nd USENIX Security Symposium (USENIX Security 13)*, 2013, pp. 605–620.

[10] O. Gasser, Q. Scheitle, S. Gebhard, and G. Carle, "Scanning the ipv6 internet: towards a comprehensive hitlist," *arXiv preprint arXiv:1607.05179*, 2016.

[11] B. Hou, Z. Cai, K. Wu, J. Su, and Y. Xiong, "6hit: A reinforcement learning-based approach to target generation for internet-wide ipv6 scanning," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

[12] Z. Liu, Y. Xiong, X. Liu, W. Xie, and P. Zhu, "6tree: Efficient dynamic discovery of active addresses in the ipv6 address space," *Computer Networks*, vol. 155, pp. 31–46, 2019.

[13] T. Yang, Z. Cai, B. Hou, and T. Zhou, "6forest: an ensemble learning-based approach to target generation for internet-wide ipv6 scanning," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 1679–1688.

[14] R. Barnes, R. Altmann, and D. Kerr, "Mapping the great void: Smarter scanning for ipv6," *Proc. CAIDA AIMS-4*, 2012.

[15] J. Ullrich, P. Kieseberg, K. Krombholz, and E. Weippl, "On reconnaissance with ipv6: a pattern-based scanning approach," in *2015 10th International Conference on Availability, Reliability and Security*. IEEE, 2015, pp. 186–192.

[16] T. Yang, B. Hou, Z. Cai, K. Wu, T. Zhou, and C. Wang, "6graph: A graph-theoretic approach to address pattern mining for internet-wide ipv6 scanning," *Computer Networks*, vol. 203, p. 108666, 2022.

[17] B. Hou, Z. Cai, K. Wu, T. Yang, and T. Zhou, "6scan: A high-efficiency dynamic internet-wide ipv6 scanner with regional encoding," *IEEE/ACM Transactions on Networking*, vol. 31, no. 4, pp. 1870–1885, 2023.

[18] ——, "Search in the expanse: Towards active and global ipv6 hitlists," in *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*. IEEE, 2023, pp. 1–10.

[19] T. Cui, G. Xiong, G. Gou, J. Shi, and W. Xia, "6veclm: Language modeling in vector space for ipv6 target generation," in *Machine Learning and Knowledge Discovery in Databases: Applied Data Science Track: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part IV*. Springer, 2021, pp. 192–207.

[20] T. Cui, G. Gou, and G. Xiong, "6gcvae: Gated convolutional variational autoencoder for ipv6 target generation," in *Advances in Knowledge Discovery and Data Mining: 24th Pacific-Asia Conference, PAKDD 2020, Singapore, May 11–14, 2020, Proceedings, Part I 24*. Springer, 2020, pp. 609–622.

[21] T. Cui, G. Gou, G. Xiong, C. Liu, P. Fu, and Z. Li, "6gan: Ipv6 multi-pattern target generation via generative adversarial nets with reinforcement learning," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

[22] E. Rye and D. Levin, "Ipv6 hitlists at scale: Be careful what you wish for," in *Proceedings of the ACM SIGCOMM 2023 Conference*, 2023, pp. 904–916.

[23] R. Hinden and S. Deering, "Ip version 6 addressing architecture," Tech. Rep., 2006.

[24] F. Gont and T. Chown, "Rfc 7707: Network reconnaissance in ipv6 networks," 2016.

[25] L. Steger, L. Kuang, J. Zirngibl, G. Carle, and O. Gasser, "Target acquired? evaluating target generation algorithms for ipv6," in *2023 7th Network Traffic Measurement and Analysis Conference (TMA)*. IEEE, 2023, pp. 1–10.

[26] O. Gasser, Q. Scheitle, P. Foremski, Q. Lone, M. Korczyński, S. D. Strowes, L. Hendriks, and G. Carle, "Clusters in the expanse: Understanding and unbiasing ipv6 hitlists," in *Proceedings of the Internet Measurement Conference 2018*, 2018, pp. 364–378.

[27] S. Networks, "Ipv6 toolkit," https://github.com/fgont/ipv6toolkit, 2022.

[28] G. Song, L. He, Z. Wang, J. Yang, T. Jin, J. Liu, and G. 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)*. IEEE, 2020, pp. 1–10.

[29] T. Pevnỳ, "Loda: Lightweight on-line detector of anomalies," *Machine Learning*, vol. 102, pp. 275–304, 2016.

[30] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE transactions on pattern analysis and machine intelligence*, vol. 20, no. 8, pp. 832–844, 1998.

[31] L. Birgé and Y. Rozenholc, "How many bins should be put in a regular histogram," *ESAIM: Probability and Statistics*, vol. 10, pp. 24–45, 2006.

[32] Cisco, "Cisco umbrella," https://umbrella.cisco.com, 2024.

[33] PremiumDrops, "Domain zone file and zone changes downloads," https://icann-account.okta.com/, 2024.

[34] C. Partridge and M. Allman, "Ethical considerations in network measurement papers," *Communications of the ACM*, vol. 59, no. 10, pp. 58–64, 2016.