Punching Bag: A Tool for Testing IPv6 Scans and Target Generation Algorithms

Lion Steger
Technical University of Munich
Munich, Germany
stegerl@net.in.tum.de

Christian Junginger
Technical University of Munich
Munich, Germany
chris.junginger@tum.de

Georg Carle
Technical University of Munich
Munich, Germany
carle@net.in.tum.de

Johannes Zirngibl
Max Planck Institute for Informatics
Saarbrücken, Germany
jzirngib@mpi-inf.mpg.de

Abstract

Scanning the IPv6 address space exhaustively is infeasible. Besides hitlists with known active addresses, Target Generation Algorithms (TGAs) are a common tool for generating input for IPv6 measurements. They generate candidate addresses based on known responsive addresses using address patterns or machine learning. Some TGAs additionally use active scans during address generation as feedback mechanisms. However, the Internet is a changing ecosystem and scans can be impacted by network events, e.g., rate limits. These effects impact TGA results, a proper evaluation of their quality and benefit, and the comparability of TGAs.

We propose an *IPv6 Punching Bag*, a local environment to test IPv6 scans and TGAs before actually using them on the Internet. It allows configuring prefixes with different response rates, *e.g.*, aliased prefixes, or address patterns. It can be used on a single machine with a low memory footprint.

We evaluate six dynamic TGAs with the Punching Bag and show that their adherence to scanning budgets and limits, and their inability to detect aliased prefixes necessitates local tests before Internet scans. We also demonstrate that 6Scan, a dynamic TGA, is not adapting to different response behavior.

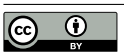
CCS Concepts

• **Networks → Network simulations**.

Keywords

IPv6, Target Generation Algorithms

ACM Reference Format:
Lion Steger, Christian Junginger, Georg Carle, and Johannes Zirngibl. 2026. Punching Bag: A Tool for Testing IPv6 Scans and Target Generation Algorithms. In *Proceedings of the 2026 ACM Internet Measurement Conference (IMC '26), October 12–16, 2026, Karlsruhe, Germany*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3777912.3839826



IMC '26, Karlsruhe, Germany

ACM ISBN 979-8-4007-2327-8/26/10
https://doi.org/10.1145/3777912.3839826

1 Introduction

The sheer size of the IPv6 address space makes exhaustive scans infeasible, presenting a challenge for Internet researchers. Target Generation Algorithms (TGAs) are tools that generate IPv6 addresses, which can be used as input for Internet measurements. They use a set of known responsive addresses to identify or learn address patterns and generate promising candidates to test for responsiveness. Throughout the last years, many algorithms have been published, *e.g.*, [12, 14, 15, 18, 26, 31]. With the increasing importance of IPv6, the research community also requires better tools and a good understanding of their properties.

Some TGAs implement scanning mechanisms during the target generation to dynamically adapt their generation process to the results. Therefore, results are heavily dependent on the current state of the Internet, and comparisons rely on the stability of the network during different runs. Changes in address usage or even network conditions, e.g., congestion or rate limiting, might influence results. Furthermore, while there are ethical guidelines for conducting Internet scans, such as rate limits and opt-out mechanisms, running these algorithms without fully understanding their implementation can result in the violation of these guidelines. While different studies [20, 21, 27] have compared various TGAs and shown significant differences in their performance and behavior, they relied on Internet measurements, potentially impacting results and impacting the network itself. It is therefore crucial to provide a deterministic, scalable test environment for IPv6 scans to compare and evaluate TGAs and to test new algorithms thoroughly before using them in Internet-wide measurements.

In this paper, we present a testing environment for TGAs, which enables researchers to test the behavior of algorithms locally under different conditions. This is enabled by a tool we call *Punching Bag*, which generates configurable ICMPv6 echo replies. It allows configuring different prefixes with different response rates or address patterns. To show the necessity of testing TGAs locally and the functions of the Punching Bag that aid these tests, we analyze the adherence of different TGAs to rate limits and scanning budgets, their reaction to *aliased prefixes* as well as prefixes with different response rates and the consistency of their results across multiple experiment repetitions.

Our main contributions in this work are:

(i) We developed and publish an IPv6 Punching Bag, a tool to test IPv6 scans and TGAs locally, in a deterministic environment. It

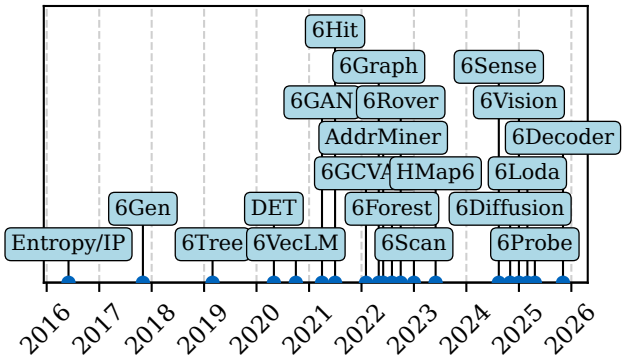


Figure 1: Timeline of TGAs being released.

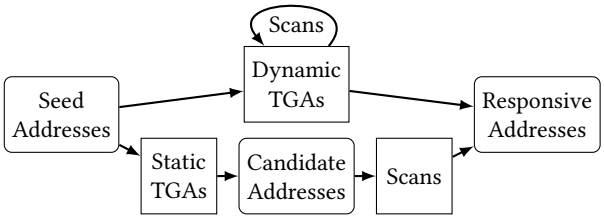


Figure 2: The TGA pipeline: Seed addresses are used to generate address candidates. Static TGAs create one list of addresses while dynamic TGAs create candidates in multiple rounds and use scans for feedback.

can be configured with different responsive prefixes, answer rates and address types.

(ii) We evaluate six TGAs using the Punching Bag. We find that they only partially adhere to scanning budgets, but generally adhere to rate limits. Furthermore, most fail to recognize aliased prefixes.

(iii) We evaluate the reaction of these TGAs to prefixes with different response rates and behavior. We show different behavior between the TGAs. 6Scan, a dynamic TGA, does not react to different responses within prefixes.

The source code of the Punching Bag is available at:

https://github.com/tumi8/punching-bag

2 Background & Related Work

We provide background information on TGAs and related concepts such as aliased prefix detection (APD) and discuss related work on these topics as well as other approaches to test frameworks for local IPv6 scanning.

Target Generation Algorithms. Many TGAs have been published over the last decade, especially since 2020, as visible in Fig. 1 and Tab. 3. TGAs are an approach to identify responsive IPv6 addresses given a set of known addresses. The general process of TGAs is visualized in Fig. 2. TGAs use a set of known active addresses, referred to as *seed data set*, as input. This set is often taken from public sources such as the IPv6 Hitlist Service [7, 34]. They apply different methods, such as clustering [15, 19], machine learning [2, 3, 26, 31] and graph theory [29] to detect patterns in the seed set and generate new, potentially active addresses, called *candidate addresses* based on those patterns. *Static TGAs* present these addresses as output to the users, who can use them as input for Internet measurements. *Dynamic TGAs*, on the other hand, scan for the responsiveness of the candidates during the generation process and adapt their generation to the results of the scans, returning the active addresses. Some algorithms implement custom scanning tools [12, 26], while others use ZMapv6 [24], an IPv6-enabled variant of ZMap [4]. While the responsiveness of a candidate address set can be checked with arbitrary protocols, ICMPv6 echo requests are the standard for all TGAs.

TGAs rarely compare themselves to existing approaches in a reproducible way, based on experiments, but rather based on results from publications. Zirngibl et al. [34] improved the IPv6 Hitlist service and added new addresses based on TGAs. They show that generation and response rates differ for each used algorithm. Steger et al. [20, 21] did a more structured comparison of TGAs. They created different input sets based on network type, *e.g.*, educational or network provider, to evaluate the impact on TGAs. Williams and Pearce [27] added further comparison metrics and input sets. They further highlight the importance of live aliased prefix detection during the target generation. All comparisons rely on active scans of live addresses on the Internet. Thus, they induce network load and might be impacted, *e.g.*, by network changes and loss. Our work provides an environment to test and compare future algorithms before scanning the Internet.

Aliased Prefix Detection. Aliased prefixes, a complete prefix used by a single host, or fully responsive prefixes, a complete prefix responsive and used by multiple hosts (*e.g.*, in CDNs) [34], are a well-known problem for IPv6 scans. These prefixes are infeasible to scan by themselves. TGAs need to be aware of them to not focus solely on these highly responsive prefixes (strong confirmation bias). While lists of known aliased prefixes exist, *e.g.*, by the IPv6 Hitlist [7, 34], they are not complete and new addresses or prefixes need to be tested. Williams and Pearce [27] have recently highlighted the impact of aliased prefixes on TGAs. The Punching Bag allows to test different detection mechanisms, their effectiveness and their adaptation to rate limits.

Test Environments. Local test environments for other protocols have been published by related work. However, the LDPlayer from Zhu and Heidemann [32] focuses on DNS, while Zirngibl et al. [33] focus on QUIC. To the best of our knowledge, there is no comparable, scalable environment for IPv6 scans. Existing environments focus on the correctness of protocols, *e.g.*, a stateful approach running complete QUIC servers. This impacts their scalability, *i.e.*, preventing us to use these environments to emulate millions of responsive IPv6 targets. The Punching Bag focuses on scans testing for responsiveness, *e.g.*, receiving an ICMPv6 echo reply. Therefore, it focuses on scalability without offering complete, higher layer protocols.

3 Design

We present the design decisions and the implementation of the Punching Bag. Furthermore, we highlight scripts and technologies used to enable reproducible experiments.

Punching Bag. The Punching Bag is a binary compiled from C++ code, which can generate ICMPv6 echo replies to received requests, which can be configured on a prefix level. Incoming packets are captured from a network interface using libpcap [23], filtered for ICMPv6 echo requests, and placed into a mutex-protected queue. The implementation is multithreaded: the main thread captures packets, while multiple worker threads process them. Each worker performs Longest Prefix Matching (LPM) using a custom trie-based data structure, which provides near-logarithmic lookup times. When a reply should be sent, a worker generates and sends an ICMPv6 echo reply via libnet [17]. Our custom Patricia-style trie stores prefixes together with configurable response rates and is built at startup from a JSON configuration. Each prefix can be assigned different response rates for different address types, *e.g.*, for lower-byte addresses [8], EUI-64 addresses or a default response rate. The distinction between address types enables the emulation of different address assignment patterns, such as incremental assignment, random generation or EUI-64. When an address is matched, a response is generated with the configured probability as described in detail in App. C. This also enables the emulation of *aliased prefixes*, the recognition of which is very important to TGAs. We run TGAs in a separate, isolated Linux network namespace, using `ip netns`, which we connect to the Punching Bag through a virtual network interface. This ensures experiments without interference from other processes and is part of our proposed solution. We evaluate the Punching Bag performance in App. D.

Experiment Generation. The experiments in this paper are defined by a scanning budget and rate limit, and a set of prefixes. For every prefix, the response rates of the Punching Bag are defined, as well as the amount and structure of the addresses within the prefix. We derive the addresses from an address structure including wild cards such as `2001:db8:1:****::***`, from which we uniformly choose random addresses by replacing each asterisk with a random half byte. We use a deterministic randomization method by using a fixed seed value to ensure reproducibility. We further ensure reproducibility by running each algorithm in an isolated, new directory to ensure no state is kept between iterations. Experiment inputs, definitions, logs and results are also kept in a unique folder each to make sure that each result can be mapped to a unique set of inputs and configurations. These configurations can be published in the future alongside TGAs to allow testing and reproducible results.

Ethical Considerations. We design our experiments around different metrics that can negatively impact networks and raise ethical concerns. This includes the total amount of scanning probes and the amount of scanning probes within a time interval. We do not include experiments assessing *opt-out mechanisms* such as block lists since none of the tested algorithms implement any such mechanisms. The design of the Punching Bag does, however, allow for an analysis of adherence to block lists by comparing the destination of the probes reaching the Punching Bag to the block list. The total amount of scanning probes is usually controlled by configuring a scanning budget for an algorithm, the adherence to which can easily be analyzed with the Punching Bag. Scanning probes within a time interval are usually controlled by scanning rate limits, which we also set and check.

Limitations. Currently, the Punching Bag can only generate responses to ICMPv6 echo requests. While some TGAs support other probes such as TCP SYN, ICMPv6 echo replies remain the most common probe type for evaluating the responsiveness of an address and are supported by all TGAs we found; no TGAs we found relies on any other probe type. Furthermore, the possibility to configure response rates to address patterns such as wildcards or wordy addresses [8] would enable more specific experiments and the emulation of other address assignment strategies. We leave an extension to further protocols, *e.g.*, reacting with a TCP SYN-ACK, and address patterns to future work.

4 Evaluation

To demonstrate the TGA Platform, we use all dynamic algorithms from Tab. 3 with available source code. This amounts to **6Hit**, **6Sense**, **6Scan**, **6Tree**, **AddrMiner-S** and **DET**. We exclude static TGAs since they do not generate any scanning traffic which the Punching Bag can respond to, wherefore no interaction between the TGA and Punching Bag takes place. We argue that static TGAs therefore do not benefit from the features of the Punching Bag and a static analysis of the generated candidate addresses is preferable to analyze static TGAs. We chose not to use 6Vision [30] as it focuses on *few-seed scenarios*, i.e., prefixes with less than 10 known responsive addresses, which is not comparable to other algorithms. HMap [13] is not included since it does not use a provided seed set. While the Punching Bag can be used to evaluate these algorithms, their goal differs from the remaining six algorithms and our current tests. We leave their evaluation to future work. We try to run the algorithms as published, only configuring parameters and applying minor modifications necessary to run inside our environment, without changing the generation logic. Lastly, we provide the TGAs with the exact input formats and all additional data required to run it. No TGAs except for 6Sense, which uses a list of known aliased prefixes and an IP-to-AS mapping as auxiliary input, requires any additional data. All experiments are run 10 times per algorithm and configuration. Presented results show the median over all runs and the standard deviation.

4.1 Adherence to Scanning Guidelines

In this experiment, we focus on the behavior of the algorithms which is relevant for ethical scanning, rather than their performance. While this behavior can also be measured during Internet-wide scans, the Punching Bag provides means to do this without the chance of violating these guidelines. We use the responsive addresses from the IPv6 Hitlist Service as input (randomly sampling to 1 M addresses for 6Sense due to hardware limitations) and configure the Punching Bag to respond with a 1% chance to all probes. The impact of the configured response rate on the TGAs is not relevant for this experiment since the TGAs reaction to different response rates is analyzed in Sec. 4.3. We run each algorithm 10 times with a budget of 10 M addresses and a rate limit of 10 k packets per second. The results are checked for adherence to the scanning budget and the configured scan rate. Additionally, the maximum traffic rate for

Table 1: Adherence to budgets and rate limits. All values represent the mean value over 10 iterations, brackets denote the standard deviation.

		Rate (packets per second)					
Algo.	Probes	Max.		Avg.		Max. per pfx	
6Hit	24.11 M	(6)	13.55 k	(83)	12.30 k	(20)	13.43 k	(3)
6Sense	10.92 M	(6.3 k)	13.87 k	(106)	9.34 k	(7)	306 k	(0)
6Scan	10.00 M	(0)	11.73 k	(267)	10.53 k	(142)	10.65 k	(272)
6Tree	188.00 M	(0)	10.82 k	(128)	9.82 k	(136)	1.19 k	(17)
AddrM.	9.96 M	(3.4 k)	10.90 k	(99)	9.27 k	(48)	10.69 k	(135)
DET	10.00 M	(1)	10.59 k	(111)	9.87 k	(4)	1.18 k	(13)

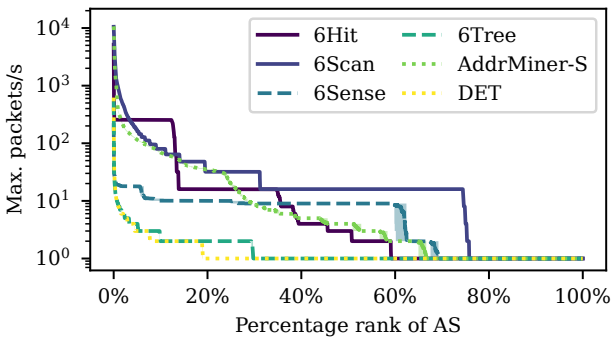


Figure 3: Distribution of peak traffic rates per BGP prefix. Lines represent the mean over 10 repetitions, the colored area shows the difference between minimum and maximum values. Note the logarithmic y-axis.

each BGP prefix is analyzed, as high traffic rates directed at single networks can result in packet loss through rate limiting, which in turn can skew scan results or cause problems for APD methods [5]. While checking the adherence to blocklists is possible with the Punching Bag, none of the evaluated algorithms take such a blocklist as input.

Results. Tab. 1 shows the results of this experiment. 6Tree and 6Hit massively and consistently exceed the configured budget. 6Tree calculated the remaining budget after every iteration, leading to negative and exceeded budget, while 6Hit appears to be using different counters as it reports budget adherence during scanning, but shows the correct, exceeded budget use in the end. The remaining algorithms adhere to the budget very closely; the slightly exceeded budget of 6Sense is caused by the APD probes which 6Sense does not count as spent budget. 6Tree, as well as DET and AddrMiner-S adhere well to the maximum scanning rate, because they all call ZMapv6 inside their code, which is a well-tested scanning tool adhering to scanning rates. 6Sense and 6Scan implement custom scanning mechanisms; 6Hit shares the scanning mechanisms with 6Scan. These mechanisms can exceed scanning rates by up to 39%; 6Scan even reports the values calculated by the Punching Bag. While this maximum rate is the exception for 6Sense, 6Hit and 6Scan exceed the limit on average as well. 6Scan, 6Hit and AddrMiner-S focus a large amount of scanning probes on a single prefix at least once, resulting in a peak traffic rate exceeding the maximum scanning rate for this prefix. While adherence to the maximum scanning rate is implemented by the scanning mechanisms, the distribution of scan rates across networks depends on the logic of the TGA.

Fig. 3 shows the distribution of peak traffic rates over a one-second window per BGP prefix. It shows that, consistently, for all algorithms, only the top fraction of prefixes is scanned with a peak rate that is close to the maximum scanning rate, which differs between the algorithms. For 70% of all scanned prefixes, 6Tree and DET never send more than one packet per second, less than 10 packets for 99%.

Key Takeaways. 6Tree and 6Hit do not adhere to a configured budget. If this budget has to be strictly adhered to, checks have to be implemented to overrule the algorithm's decisions for scanning budget. While no algorithm exceeds the configured rate limit drastically, peak scanning rates per prefix should be monitored to avoid negatively impacting specific prefixes, but also being rate-limited by single networks, which could interfere with scan results.

Table 2: Share of aliased addresses within reported active and probed addresses.

	6Hit	6Scan	6Tree	AddrM.-S	6Sense	DET
Active	99.40 %	98.53 %	99.29 %	99.98 %	0 %	99.70 %
Probed	72.45 %	51.62 %	61.95 %	98.69 %	9.30 %	84.06 %

4.2 Aliased Prefix Detection

Aliased prefixes drastically impact scans while providing limited amount of information for Internet measurements. Therefore, TGAs try to avoid generating addresses from these prefixes. As this includes the intermediate scanning steps of dynamic TGAs, many algorithms implement mechanisms to detect aliased prefixes dynamically, including the evaluated algorithms. In order to evaluate the algorithms' detection algorithm, we design the following experiment configuration: The input consists of 100 k addresses, generated randomly from wildcard patterns from two subnets each. We generate such addresses to ensure clear patterns in the seed data set, which the TGAs can recognize. The Punching Bag is configured to respond to probes in prefix one with 100% probability, making the prefix aliased. For prefix two, the Punching Bag responds with a 1% chance, simulating a non-aliased but active prefix. 6Sense is used with a third prefix, which is configured as aliased and also provided to the algorithm via a list of known aliased prefixes to analyze the difference between known and unknown aliased prefixes. The TGAs are configured with a budget of 10 M. We analyze how many of the probed addresses (i.e., probes received by the Punching Bag) and reported active addresses reside within the aliased prefix and repeat each experiment 10 times.

Results. All algorithms except for 6Sense report addresses from the aliased prefix as active, resulting in close to 100 % aliased addresses within the results, see Tab. 2. This shows that none but one of the algorithms correctly identify the aliased prefix as such. The scanning budget spent on each prefix, however, varies between the algorithms, indicating different reactions to the different response rates. There is close to no variance within these results (standard deviation $< 10^{-4}$). To better highlight the algorithms' reactions,

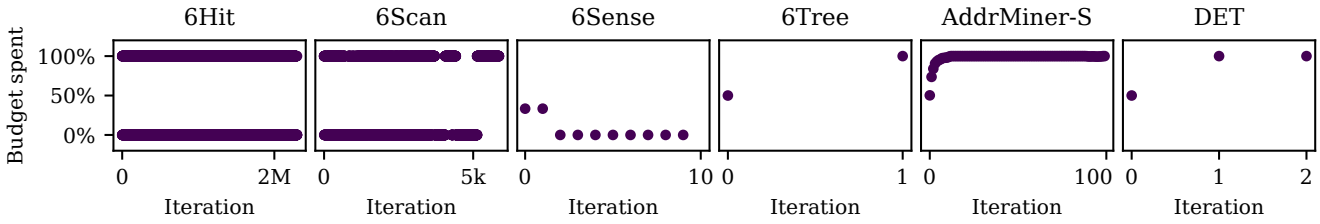


Figure 4: Fraction of probing budget spent on the aliased prefix per scanning iteration.

Fig. 4 shows the scanning budget which the algorithms allocate to the aliased prefix for each scan iteration, i.e., each step after which scan results are processed. It shows that 6Tree, DET and AddrMiner-S immediately react to the increased response rate by allocating a larger scanning budget for the aliased prefix by each iteration, while being unable to recognize this response rate as aliasing. 6Scan, on the other hand, shows no reaction to the difference in response rate and allocates nearly 50 % of their budget to each prefix, alternating between prefixes after almost every iteration. This aligns with our findings in Sec. 4.3. 6Hit allocates more budget to the aliased prefix and does not report it as aliased, there is however no observable trend in the probing of 6Hit as it probes both prefixes for the duration of the experiment. Only 6Sense recognizes the aliased prefix as such and allocates no scanning budget to it after the second iteration. Furthermore, 6Sense does not generate any addresses from the known aliased prefix at all.

To better understand possible reasons for these results, we conduct an analysis of the algorithms' APD methods and implementations. For 6Scan we find that neither the aliased prefix detection described in the paper nor any other detection method is implemented in the code and no filtering for aliased prefixes is applied. For 6Tree, the APD mechanism segfaults after the second iteration, which is why only two iterations are visible. This happens only when configuring the Punching Bag with an aliased prefix (i.e., any prefix with 100% response rate), which in turn triggers the function in which the bug resides to be called. We verified that the results of the other experiments were not impacted by this bug and that the probed addresses are still counted as candidates by relying on the Punching Bag log instead of the addresses logged by the algorithm. For AddrMiner-S and 6Hit, online detection of aliased prefixes is not intended, i.e., not described in the paper or implemented in the code. No filtering for aliased prefixes is applied by the algorithms and is instead intended to be run on the reported active addresses after the algorithm finishes. This means that only less than 1 % of addresses would be left after filtering. Lastly, for DET, the algorithm FAPD proposed in the paper is not implemented in the code and no detection is executed.

Key Takeaways. All algorithms but 6Sense fail to recognize aliased prefixes as such. The detection mechanisms of 6Tree cause the algorithm to crash. The other algorithms neither show any difference in scanning behavior nor in reporting active addresses. Therefore, they might waste a large part of the budget on aliased prefixes on the Internet, negatively impacting that infrastructure and biasing results.

4.3 Scanning Distribution

One metric many TGAs optimize is prefix coverage. A set of addresses from a variety of different networks can be favorable for Internet measurements. This often contradicts the aim of TGAs to probe addresses with a high likelihood of responsiveness, as this favors known active network areas. The tendency of TGAs to favor few responsive regions over a broad prefix coverage can therefore indicate the expected prefix coverage in Internet measurements. We construct multiple experiments in which we use the same input addresses as in the aliased prefix experiment. The Punching Bag is configured with different response rates for each prefix, to *steer* the TGAs into probing the prefix with the higher response rate. We compare a response rate of 1% to 1%, 2%, 5%, 10%, 30% and 100%; we additionally compare 10% to 30%. The experiment is repeated 10 times for each algorithm.

Results. The amount of scanning budget spent on prefixes with different responsiveness varies greatly between algorithms as seen in Fig. 5. All algorithms except 6Scan spend increasingly more scanning budget on the prefix with the higher response rate for any rate up to 30 % in which case AddrMiner-S spends more than 96 % of its budget on this prefix. As the prefix with a 100% response rate is equal to an aliased prefix, we can see the gradual convergence to the results of Sec. 4.2, as 6Sense is the only TGA to spend less budget on the aliased prefix. It is also visible that 6Tree reacts to the aliased nature of the high-response prefix, allocating less budget to the prefix than for the `1-30` case. As 6Tree crashes during this experiment, the true budget distribution for this case is unknown. The results for the `10-30` case show that for some algorithms, the difference in response rate is important, as 6Sense, 6Hit and AddrMiner-S show a more even allocation distribution than for `1-30`, while for 6Tree and DET the response rate of the higher-responding prefix seems more important, as they show a similar distribution to `1-30`. Lastly, 6Scan,shows no reaction towards any difference in response rate, which also indicates that the results of the aliased prefix test are independent of the aliased nature of the prefix.

Key Takeaways. Different algorithms show a very strong or no reaction to the response rate of addresses within a prefix. This indicates that algorithms might focus a large portion of their scanning budget on a small amount of more responsive prefixes, resulting in an uneven distribution of results across different networks. Depending on the requirements of the scanning campaign, this dictates the choice of algorithm.

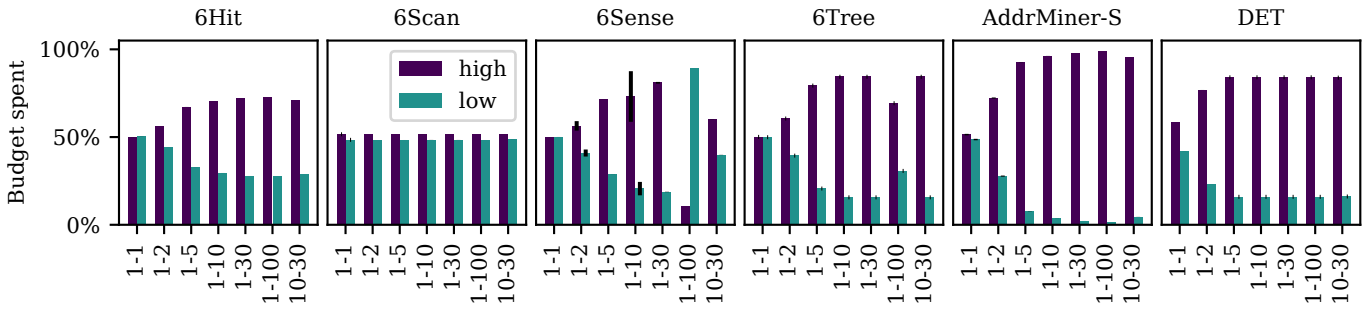


Figure 5: Differences in probing budget spent on prefixes with different response rate. Labels on the x-axis indicate the two compared response rates (*high* and *low*) in percentage probability. Error bars indicate standard deviation.

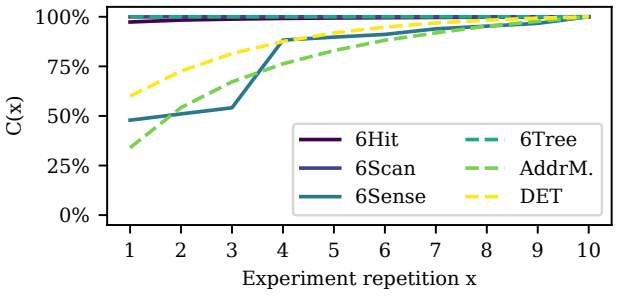


Figure 6: Comparison of unique probed addresses per experiment repetition x compared to the cumulative set of unique probed addresses. 6Scan and 6Tree overlap on the top line.

4.4 Consistency

The scanning behavior of dynamic TGAs aims to react to responses they receive from the Internet. As these responses are subject to unpredictable fluctuations on the Internet, the algorithms' influence on the consistency of multiple generations under the same conditions cannot easily be quantified. The deterministic design of the Punching Bag enables repetitions of experiments under more similar conditions. We therefore analyze the 10 repetitions of one experiment (`1-10` from Sec. 4.3) for the overlap between the probed addresses of each repetition. Additionally, we repeat the experiment with a version of the Punching Bag with random instead of deterministic responses (see App. C).

Results. Fig. 6 shows a comparison between the probed addresses of each repetition. With the set of addresses probed in a repetition denoted as P_i, we compare the cumulative number of unique probed after each repetition x to the total amount of unique probed addresses across all n repetitions, denoted as $C(x)$:

$$C(x) = \frac{|\bigcup_{i=1}^{x} P_i|}{|\bigcup_{i=1}^{n} P_i|}$$

Therefore, we can analyze how much the sets of probed addresses differ per repetition. It can be observed that DET, AddrMiner-S and 6Sense implement randomization into the algorithm itself, which is why we see differences between the address sets regardless of the deterministic responses of the Punching Bag. Contrary to that, 6Hit, 6Scan and 6Tree, implement very little randomization. For 6Scan, this also aligns with the results from Sec. 4.3, as it does not adapt to the responses of the Punching Bag at all. The results from App. C show that random response behavior from the Punching Bag does not change these results.

Key Takeaways. The key idea of dynamic TGAs is to adapt to the responses they get from the Internet. This ability can uniquely be analyzed locally, in a reproducible fashion with the Punching Bag. We show that under deterministic conditions, DET, AddrMiner-S and 6Sense show differences across repetitions due to intrinsic random behavior, while the rest responds almost identically across repetitions.

5 Conclusion

IPv6 scans are an important topic for the community but difficult due to the address space size. A vast list of TGAs promises to help researchers in this field. However, they are often complex systems and black boxes. In this paper, we propose the Punching Bag for local, performant experiments with TGAs without having to rely on Internet-wide measurements. We evaluate the behavior of six TGAs and find that scanning budgets are only adhered to partially, while scanning rate limits are generally complied with. The TGAs often fail to detect aliased prefixes and allocate different amounts of probing budget depending on the response rates of the Punching Bag. We observe different amounts of randomness in the probing strategies of the TGAs when comparing different experiment repetitions.

This demonstrates the necessity to evaluate TGAs before using them for Internet-wide measurement campaigns impacting real networks. We suggest using the Punching Bag to evaluate new and compare existing TGAs, regarding their performance given different response scenarios, their behavior in edge cases and their ability to identify aliased prefixes to prevent a strong confirmation bias. Furthermore, the high configurability of the Punching Bag allows to publish address response scenarios alongside TGAs that enable reproducibility and comparability independent of Internet changes over time.

Acknowledgments

We thank the anonymous reviewers and our shepherd for their valuable feedback. This work was funded by the Horizon Europe programme, projects GreenDIGIT (101131207), and SLICES-IP (101287692), by the German Federal Ministry of Research, Technology and Space (BMFTR), projects ALL-HANDS (02K25A026) and 6G-life (16KIS2414), and by the German Research Foundation (DFG), project SLICES-SUSTAINABILITY (566292327).

References

[1] Tianyu Cui, Gaopeng Gou, and Gang Xiong. 2020. 6GCVAE: Gated Convolutional Variational Autoencoder for IPv6 Target Generation. In *Advances in Knowledge Discovery and Data Mining - 24th Pacific-Asia Conference, PAKDD.*

[2] Tianyu Cui, Gaopeng Gou, Gang Xiong, Chang Liu, Peipei Fu, and Zhen Li. 2021. 6GAN: IPv6 Multi-Pattern Target Generation via Generative Adversarial Nets with Reinforcement Learning. In *Proc. IEEE Int. Conference on Computer Communications (INFOCOM).* https://doi.org/10.1109/INFOCOM42981.2021.9488912

[3] Tianyu Cui, Gang Xiong, Gaopeng Gou, Junzheng Shi, and Wei Xia. 2020. 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 (Lecture Notes in Computer Science).* https://doi.org/10.1007/978-3-030-67667-4_12

[4] Zakir Durumeric, Eric Wustrow, and J Alex Halderman. 2013. ZMap: Fast Internet-wide scanning and its security applications. In *22nd USENIX Security Symposium.*

[5] Mert Erdemir, Frank Li, and Paul Pearce. [n. d.]. *Understanding IPv6 Aliases and Detection Methods.* Springer Nature Switzerland, 44–73. https://doi.org/10.1007/978-3-031-85960-1_3

[6] Pawel Foremski, David Plonka, and Arthur Berger. 2016. Entropy/IP: Uncovering Structure in IPv6 Addresses. In *Proc. ACM Internet Measurement Conference (IMC).* https://doi.org/10.1145/2987443.2987473

[7] Oliver Gasser, Quirin Scheitle, Pawel Foremski, Qasim Lone, Maciej Korczyński, Stephen D. Strowes, Luuk Hendriks, and Georg Carle. 2018. Clusters in the Expanse: Understanding and Unbiasing IPv6 Hitlists. In *Proc. ACM Internet Measurement Conference (IMC).* https://doi.org/10.1145/3278532.3278564

[8] Fernando Gont and Tim Chown. 2016. Network Reconnaissance in IPv6 Networks. RFC 7707. https://doi.org/10.17487/RFC7707

[9] Shunlong Hao, Liancheng Zhang, Hongtao Zhang, Lanxin Cheng, Yi Guo, Zhanbo Li, Bin Lin, Haojie Zhu, Mingyue Ren, and Lanyun Zhang. 2025. An IPv6 target generation approach based on address space forest. *Scientific Reports* (2025). https://doi.org/10.1038/s41598-025-97640-w

[10] Nabo He, Dandan Li, and Xiaohong Huang. 2024. 6Diffusion: IPv6 Target Generation Using a Diffusion Model with Global-Local Attention Mechanisms for Internet-wide IPv6 Scanning. *CoRR* (2024). https://doi.org/10.48550/ARXIV.2412.19243

[11] Bingnan Hou, Zhiping Cai, Kui Wu, Jinshu Su, and Yinqiao Xiong. 2021. 6Hit: A Reinforcement Learning-based Approach to Target Generation for Internet-wide IPv6 Scanning. In *Proc. IEEE Int. Conference on Computer Communications (INFOCOM).* https://doi.org/10.1109/INFOCOM42981.2021.9488794

[12] Bingnan Hou, Zhiping Cai, Kui Wu, Tao Yang, and Tongqing Zhou. 2023. 6Scan: A High-Efficiency Dynamic Internet-Wide IPv6 Scanner With Regional Encoding. *IEEE/ACM Transactions on Networking* (2023). https://doi.org/10.1109/tnet.2023.3233953

[13] Bingnan Hou, Zhiping Cai, Kui Wu, Tao Yang, and Tongqing Zhou. 2023. Search in the Expanse: Towards Active and Global IPv6 Hitlists. In *Proc. IEEE Int. Conference on Computer Communications (INFOCOM).* https://doi.org/10.1109/INFOCOM53939.2023.10229089

[14] Yongqing Liu, Jialong Ke, Jiuxin Cao, Xiaopeng Wang, and Yiwen Xiong. 2026. 6Decoder: A neural network model for IPv6 target generation. *Computer Networks* (2026). https://doi.org/10.1016/j.comnet.2025.111842

[15] Zhizhu Liu, Yinqiao Xiong, Xin Liu, Wei Xie, and Peidong Zhu. 2019. 6Tree: Efficient dynamic discovery of active addresses in the IPv6 address space. *Computer Networks* (2019). https://doi.org/10.1016/j.comnet.2019.03.010

[16] Austin Murdock, Frank Li, Paul Bramsen, Zakir Durumeric, and Vern Paxson. 2017. Target Generation for Internet-wide IPv6 Scanning (6Gen). In *Proc. ACM Internet Measurement Conference (IMC).* https://doi.org/10.1145/3131365.3131405

[17] Mike D. Schiffman and Sam Roberts. 2025. Libnet: A portable framework for low-level network packet writing and injection. https://github.com/libnet/libnet. Accessed: 2025-08-10.

[18] Guanglei Song, Jiahai Yang, Lin He, Zhiliang Wang, Guo Li, Chenxin Duan, Yaozhong Liu, and Zhongxiang Sun. 2022. AddrMiner: A Comprehensive Global Active IPv6 Address Discovery System. In *2022 USENIX Annual Technical Conference (USENIX ATC 22).*

[19] Guanglei Song, Jiahai Yang, Zhiliang Wang, Lin He, Jinlei Lin, Long Pan, Chenxin Duan, and Xiaowen Quan. 2022. DET: Enabling Efficient Probing of IPv6 Active Addresses. *IEEE/ACM Transactions on Networking* (2022). https://doi.org/10.1109/TNET.2022.3145040

[20] Lion Steger, Liming Kuang, Johannes Zirngibl, Georg Carle, and Oliver Gasser. 2023. Target Acquired? Evaluating Target Generation Algorithms for IPv6. In *Proc. Network Traffic Measurement and Analysis Conference (TMA).* https://doi.org/10.23919/tma58422.2023.10199073

[21] Lion Steger, Liming Kuang, Johannes Zirngibl, Georg Carle, and Oliver Gasser. 2026. Still on Target? An Evaluation of IPv6 Target Generation Algorithms. *IEEE Transactions on Network and Service Management* (2026). https://doi.org/10.1109/TNSM.2026.3705935

[22] Xikai Sun, Fan Dang, Zihao Yang, Xinqi Jin, Junhao Li, and Yunhao Liu. 2025. 6Loda: Pattern Filtering and Ensemble Learning for IPv6 Target Generation and Scanning. In *Proc. IEEE Int. Conference on Computer Communications (INFOCOM).* https://doi.org/10.1109/INFOCOM55648.2025.11044482

[23] The Tcpdump Group. 2025. *libpcap.* https://www.tcpdump.org Packet capture library for Unix-like systems.

[24] TUM i8. 2025. ZMapv6: Internet Scanner with IPv6 capabilities. https://github.com/tumi8/zmap Accessed: 2025-11-11.

[25] University of Oregon Route Views Project. 2026. RouteViews6 BGP RIB Dump. http://archive.routeviews.org/route-views6/bgpdata/

[26] Grant Williams, Mert Erdemir, Amanda Hsu, Shraddha Bhat, Abhishek Bhaskar, Frank Li, and Paul Pearce. 2024. 6Sense: Internet-Wide IPv6 Scanning and its Security Applications. In *33rd USENIX Security Symposium (USENIX Security 24).*

[27] Grant Williams and Paul Pearce. 2024. Seeds of Scanning: Exploring the Effects of Datasets, Methods, and Metrics on IPv6 Internet Scanning. In *Proc. ACM Internet Measurement Conference (IMC).* https://doi.org/10.1145/3646547.3688449

[28] Tao Yang, Zhiping Cai, Bingnan Hou, and Tongqing Zhou. 2022. 6Forest: An Ensemble Learning-based Approach to Target Generation for Internet-wide IPv6 Scanning. In *Proc. IEEE Int. Conference on Computer Communications (INFOCOM).* https://doi.org/10.1109/INFOCOM48880.2022.9796925

[29] Tao Yang, Bingnan Hou, Zhiping Cai, Kui Wu, Tongqing Zhou, and Chengyu Wang. 2022. 6Graph: A graph-theoretic approach to address pattern mining for Internet-wide IPv6 scanning. *Comput. Networks* (2022). https://doi.org/10.1016/J.COMNET.2021.108666

[30] Wenjian Zhang, Guanglei Song, Lin He, Jinlei Lin, Songyun Wu, Zhiliang Wang, Chenglong Li, and Jiahai Yang. 2024. 6Vision: Image-Encoding-Based IPv6 Target Generation in Few-Seed Scenarios. In *Proc. IEEE ICNP.* https://doi.org/10.1109/ICNP61940.2024.10858550

[31] Zhichao Zhang, Zhaoxin Zhang, Yanan Cheng, and Ning Li. 2024. 6Rover: Leveraging Reinforcement Learning-based Address Pattern Mining Approach for Discovering Active Targets in IPv6 Unseeded Space. *CoRR* (2024). https://doi.org/10.48550/ARXIV.2401.07081

[32] Liang Zhu and John Heidemann. 2018. LDplayer: DNS Experimentation at Scale. In *Proc. ACM Internet Measurement Conference (IMC).* https://doi.org/10.1145/3278532.3278544

[33] Johannes Zirngibl, Florian Gebauer, Patrick Sattler, Markus Sosnowski, and Georg Carle. 2024. QUIC Hunter: Finding QUIC Deployments and Identifying Server Libraries Across the Internet. In *Proc. Passive and Active Measurement (PAM).* https://doi.org/10.1007_978-3-031-56252-5_13

[34] Johannes Zirngibl, Lion Steger, Patrick Sattler, Oliver Gasser, and Georg Carle. 2022. Rusty Clusters? Dusting an IPv6 Research Foundation. In *Proc. ACM Internet Measurement Conference (IMC).* https://doi.org/10.1145/3517745.3561440

A Ethics

This paper does not raise any ethical questions. All tests and measurements are done in our local Punching Bag environment, not interfering with the Internet. Inputs are either specifically generated or based on the publicly accessible IPv6 Hitlist [7, 34].

B Target Generation Algorithms Overview

Tab. 3 provides an overview about existing Target Generation Algorithms, their publication year and type. The list might not be complete, but covers all algorithms with a published paper, identified by the authors of this work. More information can be found in Sec. 2.

We evaluate six different TGAs in this paper which are highlighted in bold in Tab. 3. **6Tree** is first published dynamic algorithm we identified. It introduced Divisive Hierarchical Clustering (DHC) as a method to construct a address space tree from the seeds, which is traversed by the algorithm during online scanning, adapting the

Table 3: IPv6 TGAs with categorization, scanning method and references. Highlighted algorithms are evaluated in this work.

Algorithm	Year	Type	Scanning	Reference
Entropy/IP	2016	Static		Foremski et al. [6]
6Gen	2017	Static		Murdock et al. [16]
6Tree	**2019**	**Dynamic**	**ZMapv6**	**Liu et al. [15]**
6VecLM	2020	Static		Cui et al. [3]
DET	**2020**	**Dynamic**	**ZMapv6**	**Song et al. [19]**
6GAN	2021	Static		Cui et al. [2]
6Hit	**2021**	**Dynamic**	**Custom**	**Hou et al. [11]**
6GCVAE	2022	Static		Cui et al. [1]
6Graph	2022	Static		Yang et al. [29]
6Forest	2022	Static		Yang et al. [28]
AddrMiner	**2022**	**Dynamic**	**ZMapv6**	**Song et al. [18]**
6Rover	2022	Dynamic	ZMapv6	Zhang et al. [31]
6Scan	**2023**	**Dynamic**	**Custom**	**Hou et al. [12]**
HMap6	2023	Dynamic	Custom	Hou et al. [13]
6Diffusion	2024	Static		He et al. [10]
6Sense	**2024**	**Dynamic**	**Custom**	**Williams et al. [26]**
6Loda	2025	Static		Sun et al. [22]
6Vision	2025	Dynamic	ZMapv6	Zhang et al. [30]
6Probe	2025	Static		Hao et al. [9]
6Decoder	2026	Static		Liu et al. [14]

search direction based on the feedback of the scanner. When finding response amounts exceeding a configured threshold for an address region, the region is marked as aliased. 6Tree is implemented in C++ and calls ZMapv6 for scanning. **DET** uses similar methods to 6Tree, also constructing a space tree using DHC, however modifying the space tree during scanning. The authors also claim that DET can work on prefixes without any seed addresses compared to 6Tree. The paper introduces FAPD, an fingerprint-based APD approach, which is not implemented in the published code. DET is implemented in Python, using ZMapv6 for scanning. **6Hit** also constructs a address space tree from the seeds using an enhanced DHC algorithm, estimating the response rate of dense address regions, updating this density upon scan feedback. 6Hit is implemented as part of the code of 6Scan, using a custom scanner implementation build in C++. **AddrMiner-S** is part of the AddrMiner software, which proposes different TGAs for different use cases. AddrMiner-S is used for inputs with sufficient seeds in every prefix. It uses similar methods to DET, constructing a density space tree and scanning dense regions from this tree, updating density based on scan feedback. APD is not done during scanning, results are expected to be de-aliased after generation. AddrMiner-S is implemented in Python, based upon the code of DET, also using ZMapv6 for scanning. **6Scan** works similarly to 6Hit, building upon the idea of address space partitioning using DHC. 6Scan improves upon 6Hit regarding runtime performance and hit rate according to the authors. 6Scan is implemented in C++, using its own scanning mechanisms. **6Sense** generates candidate addresses by employing different mechanisms for the upper and lower 64 address bits. This enables recognizing the different properties of address allocation schemes, subnet configurations and device identifiers. The generator generates the most likely combinations of upper and lower 64 bits and hands them to custom scanning and de-aliasing mechanisms, both implemented in Go. 6Sense itself is implemented in Python and uses machine learning methods for generation, requiring a GPU setup.

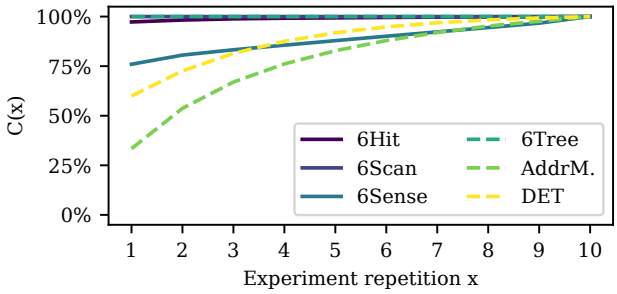


Figure 7: CDF of unique probed addresses per repetition compared to the cumulative set of unique probed addresses. The Punching Bag is configured to respond randomly. 6Scan and 6Tree overlap on the top line.

C Response Behavior of the Punching Bag

The Punching Bag implements deterministic response behavior according to a configured response probability. This is achieved by hashing the destination IP address of each incoming packet and comparing this value modulo 100 to the configured percentage response probability. In cases where truly random response behavior is desired (see *e.g.*, Sec. 4.4), we leverage randomness provided by the operating system and compare a uniformly sampled random value between one and 100 to the configured percentage response probability.

This random response behavior enables tests for the consistency of the evaluated TGAs across different experiment repetitions, as described in Sec. 4.4. The comparison to the results of the experiments with deterministic response behavior of the Punching Bag shows to what degree algorithms react to a difference in responses by the Punching Bag, as the responses are different for each iteration in this test. Fig. 7 shows that 6Hit, 6Scan and 6Tree show no reaction to the difference in response behavior, probing nearly identical candidate sets for each iteration of the experiment. The results for DET and AddrMiner-S is identical to the results in Fig. 6, indicating that their change in probed addresses over time is due to intrinsic random behavior and not a reaction to a difference in responses.

D Punching Bag Performance Evaluation

We conduct a performance evaluation to highlight the capabilities of the Punching Bag. We evaluate the maximum request rate at which the Punching Bag can be operated without packet loss and stable Round Trip Times (RTTs) as well as the influence of the configuration. The experiments are run on a system with an Intel Xeon D-1518 (2.20 GHz) with four cores, 32 GB of DDR3 RAM; requests are generated using ZMapv6.

Limits. Scanning random addresses from 5 k prefixes, we achieved a maximum rate of 60 k P/s (kilopackets per second) with a single response thread, and 70 k P/s with two. Exceeding this rate causes congestion in the request queue, leading to increasing RTTs and, ultimately, dropped incoming requests. For setups with stacked prefixes, where prefixes contain multiple nested prefixes, raising the thread count can help sustain higher rates.

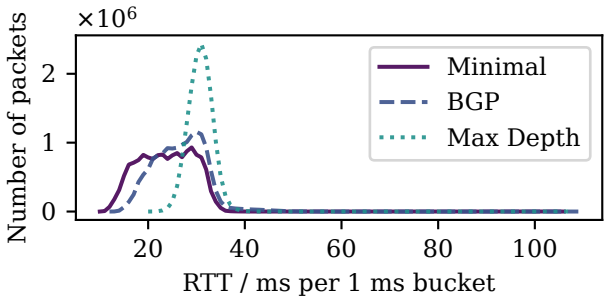


Figure 8: Round Trip Time (RTT) distribution of the Punching Bag with different configurations.

Influence of Prefix Configurations. To assess the influence of the number of prefixes stored in the trie on performance, we compared three different configurations. We chose a *minimal* configuration with a single prefix, a *maximum depth* configuration with 129 prefixes of lengths 0 through 128, each prefix containing the next, to produce worst-case lookup times, and a *BGP* configuration with 255 k prefixes from the BGP routing table released by Routerviews.org [25]. For the minimal and maximum depth configurations, the Punching Bag allocated 4 MB of RAM, whereas it allocated 233 MB for the *BGP* configuration. To measure the influence of these configurations on the performance of the Punching Bag, we then analyzed the RTTs for the ICMPv6 requests at a rate of 50 k P/s. Fig. 8 shows only a small RTT increase from the *minimal* to the *BGP* configuration; both yield RTTs of 15 to 40 ms, with minor outliers up to 150 ms. The *maximum depth* configuration concentrates RTTs around 40 ms, making it the most expensive of the three while remaining acceptable.

Key Takeaways. The Punching Bag can handle packet rates up to 50 k P/s and up to 255 k configured prefixes, which enables effective and fast evaluation of IPv6 scans and TGAs in a local environment before impacting the Internet.

E Guidelines for Using the Punching Bag

In this section, we try to summarize how we intend the Punching Bag to be used for tests of existing TGAs or during the development of new algorithms. We highlight why tests with the Punching Bag should be conducted before online scanning.

Adherence to Scanning Guidelines. When running a new or existing TGA, we recommend setting or implementing a scanning budget, a maximum scanning rate and a list of blocked networks. Using a large, diverse address collection like the data from the IPv6 Hitlist Service provides realistic input for the emulation of a real scanning campaign. Manually including addresses from the blocked networks in the input can help to verify that the blocklist mechanisms work. After a successful execution of the algorithm, the packet log of the Punching Bag can be reviewed to check the adherence to the scanning budget, the adherence to the maximum scanning rate and also the maximum scanning rate per subnet, e.g. per BGP-announced prefix. This is possible since the Punching Bag includes timestamps for received packets. Checking scanning rates can, among other things, avoid focusing the full scanning rate on a single network which might lead to rate limiting or packet loss. It is also possible from this log to verify if any addresses from blocked networks were scanned. These tests are essential for ethical scanning and should therefore be conducted before any online scanning.

Verification of APD Methods. aliased prefix detection (APD) is important for TGAs to avoid spending too much scanning budget on aliased prefixes, skewing the results. It is also important that this happens during target generation for dynamic algorithms, as dynamic TGAs learn from the results of unrecognized aliased prefixes. The recognition of aliased prefixes can be verified with the Punching Bag by setting the response rate for a subnet to 100 % for all address types, and including addresses from this subnet in the input. The addresses which the algorithm reports as active can easily be checked for aliased prefixes. The packet log of the Punching Bag can also be used to verify whether the algorithm adapted to the responses from this aliased prefix over time. Conducting these tests can avoid unwanted scanning overhead and negative impact on networks.

Reaction to Response Rates. The Punching Bag enables the analysis of an algorithms' reaction to response rates of different networks. An algorithm which shows a strong reaction to the response rate of a network, *e.g.*, by allocating a majority of scanning budget to a network with high response rates, would probe addresses from a very uneven distribution of networks in real world tests. While this can also be due to the distribution of networks in the input of the algorithm, the Punching Bag enables a differentiation between the influence of the input and the influence of the responses. An uneven distribution might not be unwanted since this can result in a higher total amount of active addresses, but depending on the use case, the results of the Punching Bag should be analyzed first to estimate the behavior of the algorithm during online scanning before conducting any real-world measurements.

Consistency and Randomness. Lastly, the design of the Punching Bag gives the unique opportunity to conduct experiments under a completely reproducible environment. Analyzing the inherent randomness of an algorithm as well as the difference to results when configuring the Punching Bag to respond truly randomly can give insights into how consistently a algorithm behaves. Quantifying the difference between multiple iterations of the same experiment can influence the choice of how often to repeat an experiment in the real world. When repeated measurements yield different results, combining multiple of these results might lead to a larger or more diverse address set. On the contrary, running tests with the Punching Bag can avoid repeating the same experiment with very similar results, which would lead to higher unnecessary scanning traffic.