Building an Open, Robust, and Stable Voting-Based Domain Top List

Qinge Xie[†] Shujun Tang[*] Xiaofeng Zheng[*◊] Qingran Lin[*]
Baojun Liu[◊] Haixin Duan[*◊] Frank Li[†]

[†]*Georgia Institute of Technology* [*]*QI-ANXIN Technology Research Institute* [◊]*Tsinghua University*

Abstract

Domain top lists serve as critical resources for the Internet measurement, security, and privacy research communities. Hundreds of prior research studies have used these lists as a set of supposedly popular domains to investigate. However, existing top lists exhibit numerous issues, including a lack of transparency into the list data sources and construction methods, high volatility, and easy ranking manipulation. Despite these flaws, these top lists remain widely used today due to a lack of suitable alternatives.

In this paper, we systematically explore the construction of a domain top list from scratch. Using an extensive passive DNS dataset, we investigate different top list design considerations. As a product of our exploration, we produce a voting-based domain ranking method where we quantify the domain preferences of individual IP addresses, and then determine a global ranking across addresses through a voting mechanism. We empirically evaluate our top list design, demonstrating that it achieves better stability and manipulation resistance than existing top lists, while serving as an open and transparent ranking method that other researchers can use or adapt.

1 Introduction

Internet measurement, security, and privacy research heavily relies on domain top lists, which provide a set of purportedly popular or commonly used domains to investigate. Existing top lists, such as Alexa [10], Umbrella [51], and Tranco [39], have been used in hundreds of prior academic studies [39,45], making it a critical research resource.

Despite many years of use in both academia and industry, these domain top lists received little empirical investigation until late 2018, when Scheitle et al. [45] and Le Pochat et al. [39] characterized modern domain top lists and identified various undesirable properties that could affect their use in measurement efforts. These issues include high volatility in the rankings, limited data granularity, and easy manipulation to achieve high rankings for target domains. In addition, the data sources and ranking methods for these top lists remain opaque[1], inhibiting a concrete understanding of these top lists and their appropriate uses.

In spite of the serious concerns uncovered about existing top lists, they continue to be used today throughout networking and security research, as there ultimately remain no viable alternatives. Few existing data sources can serve as such top lists, and there has been little investigation into the construction of such datasets. In this work, we take the first step in rectifying these shortcomings by investigating the development of a domain top list from the ground up. We seek to develop a top list that provides transparency into the data source and ranking method, stability in rankings over time, and stronger resistance to manipulation attacks.

Designing such a top list is technically challenging though, especially as there is no single objective or universal ranking to obtain. Instead, one must explore various design considerations and their implications on the resulting top list properties. Furthermore, to provide representative rankings, a top list requires global, large-scale network traffic data as input, as well as a computationally efficient data processing method.

In this paper, we explore building a top list using an extensive passive DNS (PDNS) dataset from one of the largest DNS service providers (similar to Cisco's OpenDNS [22]). Passive DNS has been widely used in both academia and industry, making it a more transparent and accessible top list data source compared to proprietary datasets (e.g., Alexa [10]). We propose a voting-based domain ranking method, where individual IP addresses express their domain preferences, and the global top list ranking is produced across IP addresses through a voting mechanism. We then systematically explore the implications of different design decisions on our produced top list's properties, and compare it to existing top lists, finding that our produced list demonstrates more favorable properties compared to others. Ultimately, our work provides the following contributions.

- We systematize the prior work on top list limitations, while also identifying new methods ourselves for more effective top list ranking manipulation.
- We develop a voting-based domain ranking method, based on PDNS. Other researchers can adopt our method for similar ranking scenarios.
- We systematically evaluate different top list design consid-

[1]Tranco [39] was created to account for some existing top list flaws. While Tranco's ranking method is published, it ultimately aggregates existing lists, thus it still relies on opaque input data.

erations when building a top list using PDNS, providing insights into the nature of top list construction.
- We provide access (both via a web interface and an API endpoint) to a regularly updated domain top list constructed using our ranking method at https://secrank.cn/topdomain. We also open-source our top list construction implementation at https://github.com/secrank.

2 Domain Top Lists

In this section, we describe existing domain top lists and their notable limitations. These limitations motivate our investigation into building a top list from scratch.

2.1 Opaque Existing Top Lists

Hundreds of research papers spanning various fields have relied on domain top lists [39, 45]. Here we describe three domain top lists that have been commonly used to rank domains by user visits or traffic. We focus on free and publicly available lists, as these are amenable for use by researchers (particularly for longitudinal studies). We note that there are other top lists that are paid (or otherwise restrict usage), or are catered towards search-engine optimization rather than domain popularity. For example, the Majestic Million [34] ranks websites based on backlinks from other sites [32, 33], rather than actual website visits. There are also industry proposals for top list constructions [20, 35] that have not been used in practice nor evaluated, and exhibit major limitations[2]. We do not focus on these other domain rankings[3].

A common theme across all these domain top lists is the lack of transparency into how these lists are actually constructed, both in the data sources used and the ranking methods themselves. We argue that this is a severe limitation, as it inhibits a detailed and complete understanding of what researchers are actually measuring when using these top lists.

Alexa. Alexa's Top Million Sites [10] is a popular and free top list. The ranking method is not public, although it is based on web traffic telemetry collected from user installs of Alexa's browser extension (on Chrome and Firefox), as well as from participating "certified" websites that subscribe to Alexa's Certify service [12]. These certified sites include Alexa's measurement JavaScript code on their web pages, allowing Alexa to directly monitor all site visits. Alexa ranks second-level domains (SLDs) only, rather than fully qualified domain names (FQDNs).

[2]Castro [20] proposed using TF-IDF [41] for deriving domain popularity, but such a method penalizes domains accessed by many IP addresses, which is contrary to expectations for popular domains. Mayrhofer et al. [35] proposed simply ranking domains by the number of normalized IP address visitors. This simple method ignores request volume and IP characteristics though, so it is unlikely to produce a representative top list.

[3]We also do not consider the Quantcast Top Million [45] as this list has not been available since April, 2020.

Umbrella. The Cisco Umbrella Top Million [51] is another popular top list based on PDNS, constructed using DNS requests observed across Cisco Umbrella's global network (e.g., OpenDNS [22], PhishTank [37]). Umbrella ranks FQDNs by computing a score for each domain, considering the number of different IP addresses issuing DNS lookups for the domain compared to others [50]. However, the actual scoring algorithm is not public.

Tranco. The Tranco top list [39] was created in 2019 by Internet measurement researchers seeking to develop a more manipulation-resistant top list. Tranco is not a top list construction method itself, but rather a method for combining the rankings of domains across multiple existing top lists (e.g., Alexa, Umbrella). While Tranco's ranking algorithm is published, we note that it relies on the existing top lists as input data, and we still lack transparency into how those lists are constructed. In addition, there is no clear interpretation of what Tranco's ranking actually represents, as each input list defines top domains differently. Our work is heavily inspired by Tranco, which we view as the first major effort in securing top lists. Our top list is inherently different from Tranco by nature though, as our approach is to build an open, stable, and manipulation-resistant top list from raw network data itself (which Tranco could use as an additional input list). In contrast, Tranco's manipulation resistance arises from using multiple lists which each must be manipulated to successfully manipulate the overall Tranco rankings.

2.2 Issues with Existing Top Lists

Beyond a lack of transparency into existing top lists, undesirable characteristics plague these lists. In this section, we discuss these issues for Alexa and Umbrella (we elide Tranco here as it relies on the other lists as inputs). We survey previously identified issues, and present new issues we uncover (involving more efficient manipulation attack methods).

2.2.1 Alexa

Undocumented Changes to Alexa's Data Collection. We identify that Alexa has modified its data collection over time without publicly disclosing the changes. These changes impact Alexa's characteristics and what Alexa's ranking actually measures, likely rendering Alexa-based research as non-reproducible (and potentially affecting the accuracy of the research itself). As one notable example, in May 2018, Le Pochat et al. [39] observed that Alexa's browser extension had halted data collection from countries in the European Union (presumably due to GDPR [4]). This change was not announced, yet it skewed data collection towards non-European populations without Alexa users' knowledge, and shrunk the top 1M list to be consistently less than a million domains (in early 2021, the Alexa top 1M list was as small as ∼500K

domains). Appendix A provides another example of an undisclosed change in Alexa's data collection.

Limited Data. Alexa has acknowledged that they do not receive enough data to make rankings beyond 100K statistically meaningful [11]. This lack of data manifests in a number of undesirable top list properties. The list has been observed to be highly volatile [45], and large portions of the list are simply alphabetically ordered [43], suggesting that many domains have identical visit metrics and Alexa breaks ties arbitrary through alphabetical ordering. Prior work [39] also demonstrated how easy it is to enter the Alexa top list by generating a few visits to target domains in browsers with the Alexa extension installed. Together, these issues raise questions about the usefulness of much of the Alexa top list.

Easier Manipulation through Alexa's Certify Service. While prior work [39, 44] identified ways to enter the Alexa top list, achieving a high ranking (i.e., below 100K) has been challenging. We identify an easier way to manipulate Alexa rankings that achieves high rankings, through leveraging Alexa's Certify service [12]. Certify is a paid service where websites obtain detailed audience insight by including a JavaScript snippet on their web pages. Beyond telemetry from its browser extension, Alexa also uses data from Certify.

We detail the manipulation approach in Appendix B. In short, using only a single IP address, we forge fake visits to a target Certified website that are interpreted as arriving from many distinct visitors, through generating requests to the Alexa data collection endpoint with the target website's Certify ID and random visitor ID values. To validate this attack, we registered two of our own test domains with Alexa Certify, and observed their Alexa rankings when generating different numbers of daily forged visitors to these domains (shown in Figure 9 in Appendix B). With 20K unique forged visitors (one request per visitor), we obtained a ranking as high as 51,530 (and similar rankings can be consistently obtained over extended periods), highlighting how effective a low-cost (e.g., $20/month) single address attack can be. We note that Le Pochat et al. [39] also performed manipulation through the Certify service, sending requests to a target Certified domain over the Tor network [49] to obtain IP diversity, but our method does not require IP diversity.

Ethical Considerations. We conduct our Alexa experiment on our own domains that host webservers indicating that they are test servers. Using only two domains should have negligible impact on Alexa rankings overall (particularly given its daily instabilities). We also rate limit our requests to Alexa's telemetry endpoint to avoid potentially inducing high load.

Bias towards Certified Domains. While telemetry from the Alexa extension and Certify are characteristically different, the Alexa top list is generated using both data sources (in an undisclosed manner). We investigate the Certified domains in Alexa's list (details in Appendix C), and identify that Alexa is heavily biased towards Certified domains.

2.2.2 Umbrella

Umbrella shares similar lack of transparency concerns as Alexa, and significant portions of its list are also simply alphabetically ordered (indicating many domains have identical ranking scores that are arbitrarily broken alphabetically) [43]. Previous studies [39, 44, 45] have successfully manipulated Umbrella's ranking by generating DNS requests sourced from multiple IP addresses. They obtained IP diversity through various methods, including using cloud providers, Tor [49], and VPN services. These prior evaluations identified that Umbrella's ranking relies heavily on the number of addresses generating DNS requests, moreso than the DNS request volume per address. Scheitle et. al [45] found that 10K addresses generating 1 DNS request each for a target domain (resulting in 10K total requests) achieved a rank of 38K, while 1K addresses with 10 requests (also resulting in 10K total requests) only achieved a rank of 199K. Rweyemamu et al. [44] also performed similar manipulation evaluations on Umbrella.

We identify an additional method for generating IP diversity for DNS-based manipulation that can achieve higher rankings than prior work (reaching a stable ranking within the top 10K), without requiring the attacker to directly control many IP addresses (thus reducing the costs/barriers for attacks). The approach is to rely on public Internet infrastructure that generates DNS lookups, such as open DNS resolvers. For example, an attacker can issue DNS lookups to open DNS resolvers, which subsequently send recursive requests to the PDNS resolver vantage point. Thus, the PDNS data captures DNS requests from the many IP addresses of these open resolvers, rather than the attacker. We detail the open DNS resolver manipulation process in Appendix D. In Section 6.2.3, we will evaluate the manipulation resistance of Umbrella and our own top list ranking method, finding that Umbrella is indeed less manipulation resistant than our approach.

3 Overview

Given the limitations of existing domain top lists, we aim to develop a new top list from the ground up. We start by discussing desired properties for top lists, and outline our approach for constructing a top list that satisfies these properties.

3.1 Top List Properties

In general, domain ranking is an inherently subjective task, as it depends on the metrics used for domain popularity. As there is no "ground-truth" or "universal" ordering, different ranking methods are arguably reasonable. However, there are still desirable properties for domain top lists, given their extensive use in measurement research. We list these properties, derived from the limitations discussed in Section 2 and from the top list properties evaluated in prior work [45].

Construction Transparency. The entire method for top list construction, including the data sources used and any data processing, should be public. Without such information, researchers lack a concrete understanding of the limitations and biases that arise from using a list, potentially polluting the research efforts that depend on the list. The existing domain top lists lack this property, as their data sources and/or list construction algorithms remain unpublished.

Construction Consistency. Ideally, the top list data collection and construction method should stay consistent over time, affording longitudinal measurement studies based on the list. Unannounced inconsistencies may introduce unexpected changes in ongoing measurements. We note that should changes be needed, they should be publicly announced and well documented. We identified (in Section 2) that the existing top lists are not consistently constructed, with undocumented changes to their data collection or ranking method.

Manipulation Resistant. Top lists should be constructed to resist manipulation, limiting the extent to which resource-constrained attackers can affect the rankings of target domains. Attackers can benefit in practice from entering top lists, especially at higher rankings. Several security analysis tools and services (e.g., DNSthingy [3] and Quad9 [7]) whitelist domains on existing top lists, and higher-ranked sites are more likely to be whitelisted (e.g., DNSthingy whitelists the Alexa top 10K [3]). Malicious domains may escape scrutiny by appearing highly ranked on top lists. In addition, attackers also manipulate top lists for attracting traffic to their sites [8, 48]. Thus, attackers are incentivized to not only enter their target domains into top lists, but to achieve high rankings. In practice, there is enough demand for top list manipulation that several websites [2, 9] offer top list manipulation as a paid service, where the costs increase significantly for achieving higher rankings (e.g., $40 for entering the Alexa list versus $4K for entering the top 10K [9]). We note that it is unreasonable to prevent manipulation altogether though, as a sufficiently powerful attack may be indistinguishable from a legitimate popular domain. However, prior work [39, 43, 45] and our own evaluation in Section 2 demonstrate that existing lists are easily manipulated with limited resources (e.g., a small number of IP addresses or requests).

Ranking Stability. A top list should exhibit ranking stability, where the majority of the list should not churn within a short period of time. In contrast, volatile lists can result in drastically different observations when measuring at different times, inhibiting measurement generalization and reproducibility. Note that a top list must still be reactive to notable short-term effects though, such as a viral new domain. Prior work [39, 45] observed high volatility in existing top lists.

3.2 Building a Top List from Scratch

In this work, we investigate constructing an open and transparent domain top list satisfying the desired properties mentioned, which existing popular top lists do not. Our exploration into how to systematically construct such a list also provides a deeper understanding of the factors that affect domain ranking. We aim for our work to drive further investigation into top list construction and top list use in Internet measurement, security, and privacy research.

To start, we require a data source that reflects the actual usage of a domain. For this, we use passive DNS (like Umbrella). We describe our PDNS dataset in Section 4, which provides large-scale global visibility into domain activity. Passive DNS has been used extensively in prior research [14–16, 18, 24–27, 53], affording other researchers an opportunity to adopt our domain ranking method on other DNS datasets. In contrast, a method requiring proprietary data (e.g., Alexa) cannot be broadly used.

By using PDNS data, we aim for our list to rank the most used domains, with DNS lookups as proxies for usage. From PDNS data for a domain, we can observe two metrics that intuitively reflect domain usage: 1) the number of IP addresses that perform lookups, and 2) the lookup volume. More IP addresses querying for a domain indicates that the domain is more widely used. Meanwhile, the lookup volume for a domain correlates with how strongly IP addresses prefer the domain, as more heavily used domains should be queried more frequently. We aim to incorporate both metrics in our ranking method, rather than only one, for several reasons[4]. First, the philosophical motivation behind using both metrics is to more comprehensively characterize domain usage patterns compared to using a single metric, as "usage" can be defined by both how widely something is used, as well as how frequently/intensely it is used. In addition, using both metrics rather than only one can also improve manipulation resistance, one of our core design goals, as attacks must exhibit both high lookup volume and high IP diversity, both of which incur costs. Finally, using both metrics can also provide more stability than using a single metric, as the resulting ranking would be less sensitive to minor fluctuations along only one metric/dimension. However, aggregating the two metrics is non-trivial. Directly statistically aggregating the two metrics for a domain lacks a clear meaningful interpretation, as it is unclear how to compare two domains where one has higher lookup volume but lower IP diversity, and the other has higher IP diversity but lower volume.

Instead, we develop an interpretable ranking approach where each client IP address expresses their individual domain preferences, and the global top list ranking is based on aggregating these domain preferences across all IP addresses through a voting scheme (where voting has been a well-studied approach for combining preferences across multiple units). Thus, our ranking can be interpreted as an Internet-wide election (across IP addresses) on popular domains. More specifically, our ranking measures the most used domains as

[4]We note that Umbrella is based on these two DNS metrics as well, although its method is not public.

reflected by DNS lookups in PDNS data (thus the per-IP domain preferences and the aggregated IP address preferences are factored in). In Section 5.1, we describe how we determine the per-IP domain preferences using domain lookup metrics. We discuss potentially weighting IP addresses in Section 5.2, as not all addresses are equal in their user populations. Finally, in Section 5.3, we adopt voting theory to aggregate IP address preferences, leveraging existing voting mechanisms for reaching joint decisions across heterogeneous populations. This multi-phase approach allows us to incorporate information from both DNS-based domain usage features in determining a top list ranking. In Section 6, we evaluate the properties and parameters of our top list in comparison to existing lists, demonstrating its practical value for Internet measurements.

4 Data Source

Here, we detail our PDNS dataset and discuss the considerations for its use.

4.1 Our PDNS Dataset

Our study is based on continuous large-scale PDNS data from the public DNS resolvers[5] for 114DNS, the largest DNS provider in China [1]. The dataset contains all client DNS requests, as well as the DNS responses returned by the 114DNS recursive resolvers. Due to ethical considerations, we avoid investigating DNS behavior for specific IP addresses and focus on aggregate analysis.

The PDNS data used in this study is collected from Nov. 2020 to Apr. 2021. On average, the PDNS dataset consists of ∼500B unique DNS requests from 70M clients per day, for ∼550M FQDNs. These FQDNs are distributed across an average of 13M SLDs a day, and cover 99.9% of IANA TLDs [28]. (Note that our daily data volume remains consistent over time.) This PDNS data provides large-scale and longitudinal visibility into domain lookup activities, particularly compared to the datasets used by other top lists and PDNS data used in prior work [14–16, 18, 24–27, 53]. Compared to the PDNS data reportedly used for Umbrella, our PDNS data has 5M more IP addresses and 2.5× more DNS requests observed per day [51, 52]. While Alexa does not use PDNS, it is reported to collect telemetry from about 705K distinct clients [21, 23], two orders of magnitude less than our data. We also briefly note that our PDNS data is more than 35× larger than the largest PDNS dataset we identified used in prior research [24].

4.2 Considerations for using PDNS

Data Skew. Using a PDNS dataset, our top list reflects domain usage based on DNS lookup metrics. These metrics are dependent on the PDNS vantage point though, and different PDNS vantage points may observe varying levels of DNS traffic for a domain. We recognize that as 114DNS is a Chinese DNS provider, the client population exhibits regional biases (e.g., skewing towards users in Asia), which can affect which domains are most highly ranked (e.g., domains popular in Asian countries). Such biases are inherent to any data source used for top list construction though (including Alexa, Umbrella, and prior PDNS datasets), as no network traffic dataset offers a truly global vantage point.

To better understand the nature of our dataset, we randomly sampled one day of PDNS data to investigate the geolocation distribution of client IP addresses. Using the Maxmind geolocation database [6], we identified that 8.2% of IP addresses were located outside of China, and 12.6% were not in Maxmind but we suspect are also external to China (as we also further used IPIP [5], a Chinese-centric IP geolocation database, and did not find these IP addresses either). Though our data is skewed towards Chinese clients, it consists of a large portion of clients in other countries (∼17M IP addresses) still providing domain lookup activities. We further use the Maxmind ISP database [6] to analyze the distribution of our clients across networks (using the same sampled day). Over 85% of queries originate from telecom companies (e.g., China Telecom/Unicom, Viettel Group, Vodafone Idea), which include both residential and enterprise users. Other notable networks observed include those of research institutions (e.g., China Education and Research Network Center, National University of Singapore, University of California San Diego, MIT, and Stanford) and large cloud service providers (e.g., Amazon, Google, Oracle, Alibaba, Tencent).

Given the scale and scope of our dataset, we believe it is still a meaningful dataset for understanding DNS dynamics and leveraging DNS data for constructing a top list. We also note that the primary focus of this work is not to characterize this specific PDNS dataset, but rather to develop a general method for top list construction, which can apply broadly to other DNS datasets (which will exhibit their own skew).

Data Analysis Time Window. Any domain lookup metric must be defined over some time period. Here, we prioritize daily PDNS snapshots, as such granularity is fine-grained enough to capture the daily patterns of users/devices, such as behavioral changes on weekdays versus weekends, or special events and holidays. One can aggregate rankings across multiple days [39] if long-term rankings are desired.

DNS Caching. DNS employs caching, where client lookups that match unexpired records in a DNS cache will not result in further DNS requests. This naturally impacts analysis based on DNS request metrics observed at a resolver. Existing DNS-based domain ranking methods [20, 44] do not factor in DNS caching and the *Time-to-Live* (TTL) values. Prior work [27, 40] proposed modeling the true DNS query rate with a Poisson distribution. We investigate the characteristics of our PDNS dataset to understand if we can reliably factor

[5] Resolver IP addresses:114.114.114.114 and 114.114.115.115

in DNS caching (details are in Appendix E). Ultimately, we encounter three significant issues with using TTLs to predict caching behavior: CNAME caching discrepancies across operating systems, different client-specific caching policies, and unclear DNS cache boundaries. These issues cannot be resolved using DNS-based data, which provides visibility only at IP address granularity rather than client granularity. Thus, we empirically justify not factoring in DNS caching in our ranking, as we lack a reliable way to do so.

5 Voting-Based Domain Ranking

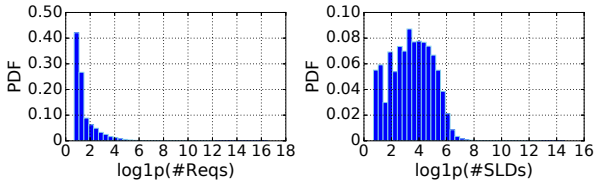
Sections 4 discussed our PDNS data characteristics and considerations when using this data. Here, we introduce our voting-based domain ranking method. In Section 5.1, we first determine the domain preferences of individual IP addresses. In Section 5.2, we discuss weighting IP addresses to account for the differences in the number of clients that might reside behind a single address. Finally, in Section 5.3, we investigate aggregating the domain preferences across all IP addresses using a voting scheme, producing a global domain ranking.

5.1 IP-Specific Domain Preferences

We start by considering how to calculate the domain preferences for a single IP address. From the DNS requests observed from an IP address (within a daily snapshot, as discussed in Section 4.2), we can measure the request volume observed for each domain. Beyond only considering volume, we can also quantify the duration over which the requests are made, which we term the *active duration*. Intuitively, the clients residing on an IP address should more strongly prefer domains that they request more frequently and more consistently over time.

Request Volume γ. For an address-domain pair, the number of requests reflects the address's preference for the domain. We observe that request volume exhibits a long tail, where there are few requests for most pairs, but some pairs exhibit up to millions of requests. When considering request volume metrics, to avoid significantly overweighting the minority of pairs with large request volumes, we apply the common $log1p(x) = ln(x+1)$ data smoothing function to request volume values. This data smoothing also aids manipulation resistance by preventing an attacker generating a high request volume from obtaining an overly high request volume metric (i.e., with smoothing, generating multiple times more requests does not increase γ by the same extent). Figure 1(a) plots the PDF of γ values for pairs on a randomly sampled day.

Active Duration α. Intuitively, popular domains are consistently queried over time. Thus, we also consider the duration over which we observe DNS requests for a domain, and for domain preferences, we weight domains higher that are regularly queried. As a side effect, manipulation efforts are incentivized to continue for longer periods of time, which can increase attack costs and the likelihood of detection.



(a) γ (Smoothed #Reqs) (b) δ (Smoothed #SLDs)

Figure 1: PDFs of the $log1p$-smoothed numbers of requests (γ values across address-domain pairs) and requested SLDs (δ values across addresses), on a randomly sampled day.

To measure active duration, we split DNS requests into different time intervals (we discuss our choice of using 10-minute intervals in Section 6.1). We consider an <IP address i, domain d> pair as *active* within an interval if we observe a DNS lookup from address i for domain d within that time window. For an address i, the active duration of domain d is the total number of intervals that <i, d> is active for. Thus, more regularly queried domains will be active for more intervals. Inspecting the distribution of active durations on a randomly sampled day (other days exhibit similar behavior), we observe that $\sim$60% of pairs are active for one 10-minute slot, indicating that most requests are bursty and only a few domains are queried regularly over time.

Combination. With two domain preference metrics, active duration α and (smoothed) request volume γ, we need to combine them to reach the overall domain preferences for an IP address. First, to scale each metric equivalently, we max-min normalize each metric across domains in DNS requests for a given IP addresses. We then determine an aggregate domain preference metric as the geometric mean of α and γ (as a geometric mean essentially scales the request volume by the active duration, factoring in both metrics). We then sort the aggregate domain preference metrics to determine the domain preference ranking for the address.

5.2 IP Address Weighting

With PDNS data, we observe requests at the IP granularity (as also discussed in Section 4.2). However, multiple client devices may reside behind an address, each generating independent DNS requests. Philosophically, treating each IP address equally when computing domain ranks could unfairly represent the contributions of individual clients' preferences. Thus, we explore weighting addresses to account for their differing client populations. PDNS data does not provide visibility behind IP addresses. However, we perform a best-effort estimation by assuming that an address's client population correlates with the diversity of domains resolved via DNS as well as the address's total request volume.

Domain Diversity δ. Our intuition is that domain diversity, the number of distinct domains requested by an IP address, positively correlates with the number of clients behind the

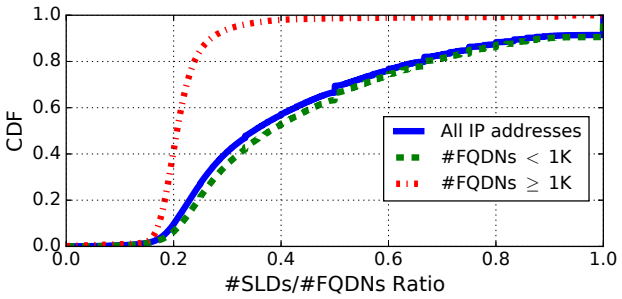


Figure 2: CDF of the #SLDs to #FQDNs ratio across IP addresses on a randomly sampled day, distinguishing between addresses requesting 1K+ FQDNs from other addresses.

address (as more clients with their own user behaviors exhibit more diverse domain requests). To distinguish distinct domains, we could consider distinct SLDs or FQDNs. We observe many addresses that request many FQDNs which correspond to few SLDs. Such behavior actually suggests low domain diversity, often accessing the same Internet services. Figure 2 depicts the CDF of the #SLDs to #FQDNs ratio for IP addresses on a randomly selected day (other days exhibit similar behavior). We observe that few addresses ($\sim$10%) request the same number of FQDNs and SLDs. Notably, addresses that request more than 1K FQDNs correspond to significantly fewer (often $\sim$20% as many) SLDs. Thus, we choose to distinguish distinct domains at the SLD granularity, and measure an address's domain diversity in terms of its number of SLDs requested. This choice also avoids overweighting addresses (such as an attacker's) that simply request many FQDNs for a small set of SLDs. There is again a long-tail distribution observed for #SLDs, so to avoid overweighting the addresses in the long tail, we again use the $log1p$ function for smoothing domain diversity values (with the PDF of address δ values on a randomly selected day shown in Figure 1(b)).

IP Address Total Request Volume μ. For an IP address, the maximum number of requests for any given domain and the total number of requests among all domains intuitively correlate with the number of clients on that address. However, as we will weight addresses relative to each other, a metric that is harder for an attacker on one address to manipulate relative to other addresses is preferable. If using the maximum number of requests for a domain per address as a weighting factor, an attacker would only need to generate more requests for a target domain than most other addresses (for a domain) to weight attack addresses more. In comparison, if using the total request volume as a weighting factor, an attacker would need to generate more requests than other addresses altogether to manipulate weighting, a higher bar. Thus, we choose to use total request volume as a weighting factor. We observe a long tail in the distribution of total request volume across addresses, and again apply $log1p$ data smoothing to avoid overweighting addresses in the long tail.

Combination. As with domain preferences, we have two weighting factors that must be aggregated to determine an address's weight. We again normalize both to scale each metric equivalently. Since an address's weight should never be zero (otherwise it does not contribute to the global ranking), we use max normalization (rather than min-max normalization). Specifically, δ and μ are scaled to range between $[1/\delta_{max}, 1]$ and $[1/\mu_{max}, 1]$, respectively. Then, we compute the geometric mean of the normalized δ and μ values to determine an address's weight. Using a geometric mean intuitively scales an address's total request volume by its domain diversity, such that addresses with higher volumes and higher diversity are weighted more (correlating with larger client populations).

5.3 Voting across IP Addresses

Having established per-IP domain preferences (as well as potential weights for addresses to account for differing client populations), we require a method for aggregating these preferences across addresses, for ranking the most preferred (popular/used) domains. A voting scheme serves as a natural solution for such preference aggregation, as it combines the preferences for candidates (domains in this case) for multiple voters (IP addresses in this case) and determines a global ranking of candidates. Rely on a voting mechanism also allows us to tap into the existing voting theory literature, which has analyzed the computational complexities and election properties of various schemes [17], as well as their resistance to manipulation attempts [36]. Philosophically, voting to determine a global top list seems apropos, as voting is a social process and domain ranking is ultimately an application of social choice.

In our application of voting, we have both a large number of voters (tens of millions of IP addresses) and a large number of candidates (hundreds of millions of FQDNs). Thus, we require a computationally efficient voting method. While a diverse set of voting schemes exist, we choose to concentrate on those based on scoring rules, rather than pairwise comparisons. While simpler, a scoring-based scheme exhibits computational complexity that grows linearly with the number of candidates/domains, whereas a pairwise comparison scheme (e.g., *Condorcet* voting [54]) exhibits quadratic growth.

Besides complexity, we also require three properties for a voting mechanism, which are not provided by all schemes: 1) The voting method must allow a voter to vote for multiple candidates, as we consider multiple domains queried by each IP address. This property excludes *Plurality/FPTP* voting, where each voter chooses a single candidate; 2) We avoid rating-based voting methods where each voter assigns a score to all candidates. Such methods would require a per-IP scoring function that provides equivalent interpretation across all addresses, which is unsuitable for addresses in practice which have different user populations, DNS activities, and domains requested; 3) We cannot use a voting scheme that involves multiple voting rounds (e.g., *run-off* voting), as we have only one static set of preferences from our voting IP addresses.

We identify four well-established voting schemes that satisfy our time complexity requirements and desired properties.

Approval. With *Approval* [42], each voter can vote (give approval) to any number of candidates. Candidates are ranked in order by the number of approving voters (with ties broken arbitrarily). In our application, this method is equivalent to only considering the number of IP addresses that query for a domain. If factoring in IP weighting, then a domain's score is the sum of the weights of approving IP addresses. While simple, this voting scheme does not leverage information from per-IP domain preferences (although we will still evaluate it).

Borda. Using *Borda count* [42], each voter ranks the candidates/domains, and a candidate receives one point for every other candidate ranked below it by the voter. Candidates are then ranked in order by the sum of points across all voters. In our application, as the number of domains requested by IP addresses vary, we adopt *truncated voting* [42], where each voter ranks a threshold number of candidates. In Section 6.1.1, we evaluate different truncation thresholds.

Dowdall. *Dowdall* [42] is a variation on *Borda*, where voter points are instead assigned following Zipf's law. A voter's top-ranked candidate receives one point, whereas the N-th ranked candidate receives $1/N$ points. Candidates are again ranked in order by the sum of points across all voters. Unlike *Borda*, *Dowdall* does not require truncated voting, as it has a score cap (of one) regardless of the number of candidates. However, we note that the score differences between a voter's consecutively-ranked candidates are more extreme than with *Borda* (i.e., a voter's top-ranked candidate receives twice as many points as the second-ranked candidate under *Dowdall*, whereas the difference is only 1 point under *Borda*).

Bucklin. With *Bucklin*, voter top choices are first considered. One candidate wins if it has a majority among voters. If not, voter second choices are added into consideration, and the candidate wins if it now has a majority among voters. This process continues until a candidate obtains majority[6]. We note that once one candidate has a majority among voters, *Bucklin* is equivalent to a truncation of *Approval* where each candidate/domain receives one point. Given the similarities, we elide further investigation of *Bucklin*.

6 Evaluation

Here, we evaluate our voting-based domain ranking method, investigating how different design decisions affect top list properties, and how our domain top lists compare to existing ones. We also characterize the top domains ranked by our method. Recall that our ranking method involves domain preferences, IP weighting, and a voting mechanism. We consider two main design decisions: use of IP weighting and the voting method chosen.

Note that as our input data collection and top list construction methods are openly described (although the raw PDNS data is not publicly available) and should remain constant, we argue that our top list provides more construction transparency and consistency compared to existing top lists (e.g., Alexa and Umbrella, which remain black boxes, and Tranco, which relies on other lists). Thus, for evaluating top lists properties, we focus the remainder of this section on manipulation resistance and stability properties.

6.1 Ranking Implementation and Runtime

We first discuss our system setup and computational costs for processing PDNS data and computing a daily top 1M list.

We receive real-time PDNS data from 114DNS using Apache Kafka, the open-source distributed data streaming platform. The data stream is divided into 10-minute chunks that we process immediately. Given this data chunking, we naturally consider a domain's active duration (used when computing per-IP domain preferences, as discussed in Section 5.1) in terms of 10-minute intervals. We extract the relevant fields from all DNS record (e.g., A and CNAME) requests for our ranking methods, which amount to ∼800 GB per day.

We execute our ranking method in a distributed fashion using Apache Spark on YARN, with 300 executors each configured with 2 cores and 4 GB of memory. We perform a minor optimization before the voting phase by filtering out domains with fewer than 15 total DNS requests or requested by fewer than 15 IP addresses, as this reduces the voting phase's computation and these domains will not make it into the top 1 million domains (this filtering leaves approximately 7M domains a day). Our total end-to-end ranking computation takes 1.5 hours. Over half of the runtime is for per-IP domain preference calculations, the most computationally intensive component. IP weighting requires about a quarter of the time, whereas the voting phase is quick (∼5 minutes).

6.1.1 Borda Truncated Voting Threshold

Recall from Section 5.3 that when using *Borda* voting, one can truncate the IP-specific domain preferences to a threshold length to account for varying numbers of domains requested by different IP addresses (other voting schemes do not require such truncation, as their point ranges are set and are not dependent on voting behavior). Here we define $Borda_N$ as our domain ranking method using *Borda* voting with a truncation threshold of N (where $N = full$ indicates no truncation), and investigate the impact of different threshold values.

Consider an IP address that ranks domains A and B as its first and second preference, respectively. (The following argument still holds for lower-ranked adjacent domain pairs.) Under *Borda* voting with truncation threshold N, this address

[6] In our PDNS dataset, on average, a daily *Bucklin* winner can be elected after approximately 54 rounds with IP weighting (i.e., the top 54 domains in each IP address's preference ranking are counted), and 200–250 rounds without weighting. If all domains are counted, there are multiple domains with a majority of the votes.

awards N voting points to A and $N-1$ points to B. While the raw difference between A and B's points is always one irrespective of N, the relative difference will be $\frac{1}{N}$. As a consequence, larger truncation thresholds N result in less impact from IP-specific domain preferences, and more emphasis on the number of addresses voting/querying for a domain (as adding one voting address may contribute far more points to a domain than obtaining a higher preference ranking at an already-voting domain). Thus, large N values shift *Borda* to behave more similarly to *Approval*, which disregards address domain preferences.

We empirically confirm the similarities between $Borda_{full}$ and *Approval* by comparing the top lists produced by the two methods. With $Borda_{full}$, N is set to the maximum number of FQDNs requested by any IP address. In our dataset, we observe that N is consistently large (1.08M on average, and can be up to 4.5M). When using IP weighting (results are similar without weighting), we find on average a 99.98% intersection of the daily top 1K domains for $Borda_{full}$ and *Approval* (Kendall's τ coefficient[7] = 0.99), and a 97.84% intersection of the daily top 1M lists ($\tau = 0.95$).

On the other hand, small N values limit the degree to which addresses can express their domain preferences. We observe that a small fraction of addresses request high numbers of FQDNs (resulting in the high N values when using $Borda_{full}$), but 91% of addresses request up to 1K FQDNs daily (on average). In comparison, only half of IP addresses request 100 or fewer FQDNs a day.

We consider $N = 1K$ to be a philosophically reasonable threshold[8], as it captures the full domain preferences for the vast majority of addresses without being too large of a value[9] (which would minimize the value of domain preferences). In our subsequent evaluations though, we still consider N in $\{10, 100, 1K, full\}$. As we will discuss, we find that $N = 10$ and $N = 100$ result in poor manipulation resistance, and $Borda_{full}$ shares undesirable stability characteristics with *Approval*. Thus when using *Borda* voting, we will focus on evaluating $Borda_{1K}$ (often relegating evaluation results for other *Borda* variants to Appendix F).

6.2 List Property: Manipulation Resistance

Here, we evaluate the manipulation resistance of our voting-based top lists, across different top list parameters (IP weighting and voting methods) and in comparison to the existing top lists (i.e., Alexa, Umbrella, and Tranco). As discussed earlier, when evaluating *Borda*, we will focus on $Borda_{1K}$, relegating results for other *Borda* variants to Appendix F. We will briefly evaluate using *Dowdall*, but we identify that it produces top lists with significantly lower manipulation resistance than $Borda_{1K}$. *Approval* also exhibits undesirable properties (as discussed in Section 5.3), so we relegate its analysis to Appendix F.

Threat Model. We consider an attacker that aims to manipulate our top list to obtain a high ranking for a target domain. The attacker has a presence on a number of IP addresses (e.g., through compromised hosts, botnets, cloud hosting, or VPNs). To manipulate our top list, the attacker generates DNS requests on those addresses (to our PDNS vantage point) for two potential purposes. First, the attacker could strive for a higher address-specific domain preference rank for its target domain, which requires sending DNS requests for the target domain[10]. We will call these *domain preference boosting* DNS requests. Second, when using IP weighting for top list construction, the attacker could also generate DNS requests to other domains to increase their address's weight, which we will call *weight boosting* DNS requests.

In this section, we consider different attacker variants, including different numbers of attack IP addresses, whether attackers fully control traffic from those addresses (as attackers may reside on addresses with benign users generating their own traffic), and different numbers of DNS requests generated (for both domain preference and IP weight boosting).

6.2.1 Design Decision: Using IP Weighting

Isolated Attackers. We first investigate the impact that IP weighting has on our top lists' manipulation resistance. To isolate IP weighting's effect, we consider an attacker that has full control over its IP addresses, and thus can readily rank its target domain as the top preference for its addresses (eliminating address domain preference as a variable) and optimize its addresses' weights (based on the number of weight boosting DNS requests it generates). Then, we empirically evaluate how high the attacker can manipulate its target domain's ranking in both the weighted and unweighted top lists, depending on the number of attacker addresses and the number of weight boosting requests generated. We compute the average attacker domain's ranking across the daily top lists constructed from our PDNS data using different voting methods. Figure 3 depicts the average attack success for different top lists (created using $Borda_{1K}$ and *Dowdall*) and attack parameters (the results of *Approval*, $Borda_{full}$, $Borda_{100}$ and $Borda_{10}$ are shown in Figure 11 in Appendix F).

For $Borda_{1K}$ top lists, we observe that without IP weighting, an attacker only requires 45 IP addresses (on average) to enter the top 1M ranking. Note that this attacker need not generate any weight boosting requests, as the top list is unweighted. In

[7] Kendall's τ coefficient [31] is used to measure the similarity between two ordered lists, and ranges from -1 to 1. The closer τ is to 1, the more similar the ordered lists are.

[8] Note that the computational costs of using different threshold lengths do not vary significantly, and thus is not a major factor in the design decision.

[9] The distribution of the number of FQDNs requested by IP addresses exhibits a long tail. Thus, we observe that N values larger than $1K$ (e.g., $10K$, $100K$) provide diminishing returns in capturing the full domain preferences of more addresses, and we did not explore these thresholds.

[10] For top lists constructed using *Approval*, only one DNS request to the target domain would be needed, as address domain preference is not involved.

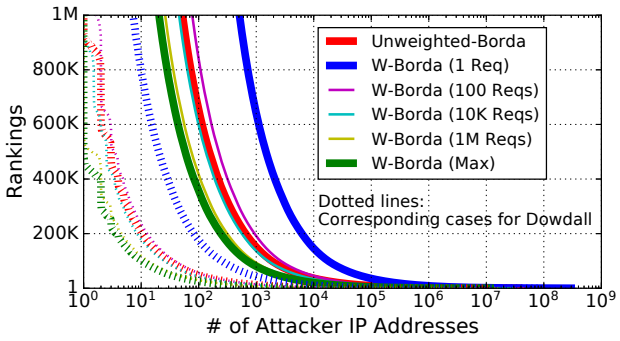


Figure 3: Average rankings of the attacker's target domain, depending on different top list construction parameters (weighted versus unweighted, $Borda_{1K}$ versus *Dowdall* voting) and attack strengths (different numbers of attacker IP addresses and weight boosting DNS requests). Curves labeled as "W-" represent weighted top lists, and the number of weight boosting requests are listed in parentheses ("Max" indicates an attacker obtaining the maximum normalized weight).

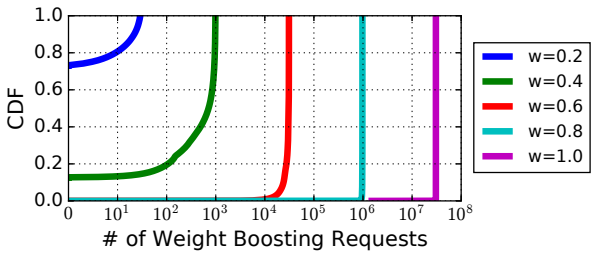


Figure 4: CDF of the # of weight boosting requests that attackers must send per IP address to obtain a desired IP weight, for addresses in a randomly sampled day of our PDNS data.

comparison, with IP weighting, the identical attack (without weight boosting requests) requires an average of 495 IP addresses to enter the top 1M, an order of magnitude increase. For an attacker to achieve similar manipulation rankings on a weighted list compared to an unweighted one, using the same number of attacker IP addresses, we can see that they require about 10K weight boosting requests for each attacker IP address, a more resource-intensive attack. (We observe the same trend for weighting with other voting methods, including *Dowdall*, as visible in Figures 3 and 11.)

We do identify that weighting is not strictly better for manipulation resistance, for all voting methods. As seen in Figure 3 (and Figure 11 for other voting methods), if an attacker sends enough weight boosting requests (e.g., 1M requests per address), they can obtain higher rankings (with the same number of attacker addresses) in the weighted top list than in the unweighted list. Thus, there exists a trade-off where weighting improves manipulation resistance against low-resource attacks (requiring few DNS requests per attacker address), but affords higher upper-bound success to high-resource attacks.

Overall though, we recommend using IP weighting for two reasons. First, the philosophical motivation behind applying weighting is to more realistically reflect IP characteristics, more heavily weighting addresses with presumably more clients, treating clients fairer. Second, even with strong attacks that are more successful against weighted top lists versus unweighted ones, the degradation due to weighting is relatively limited. For example, with weighted $Borda_{1K}$, attackers with 100 addresses who obtain maximum address weights (which may not be practical, as it requires millions of weight boosting requests per address) can achieve a ranking about half that without weighting (150K versus 300K, respectively). Meanwhile, without weighting, the lowest-cost attack (requiring one DNS request to the target domain per attacker address) achieves an order of magnitude better ranking on the unweighted list than the weighted one (as discussed). Thus, without weighting, attackers need not launch resource-intensive attacks to achieve high rankings, whereas weighting forces them to launch such attacks to obtain higher rankings. Therefore, we focus on weighted top lists moving forward.

Co-located Attackers. We also consider an attacker that does not have full control over its IP addresses, where benign co-located clients may generate their own traffic which contributes to the addresses' weights. We randomly sampled one day of PDNS data to simulate this scenario. In Figure 4, we plot the distribution of the number of weight boosting requests an attacker must send per address to obtain a certain IP weight (factoring in benign traffic). We observe that attackers can obtain low weights (e.g., 0.2) without many weight boosting requests (less than 100 requests), but even moderate weights require a significant number of requests. For example, over 10K requests are typically necessary to obtain a weight of 0.6 and approximately 1M requests for a weight of 0.8. Thus, even with benign traffic, attackers must generate significant amounts of weight boosting requests to obtain high IP weights, and our support for the use of IP weights remains.

6.2.2 Design Decision: Choice of Voting Method

In Figure 3, all *Dowdall* curves are to the left of all $Borda_{1K}$ curves, regardless of weighting or attacker parameters, indicating that top lists constructed using *Dowdall* are significantly more vulnerable to manipulation than $Borda_{1K}$-based top lists. As we do not identify additional properties where *Dowdall* outperforms $Borda_{1K}$, such as with stability (as will be discussed in Section 6.3), we do not advocate for the use of *Dowdall* over $Borda_{1K}$. We observe that $Borda_{10}$ and $Borda_{100}$ behave similar to *Dowdall*, with significantly worse manipulation resistance (as shown in Figure 11) and no favorable properties compared to $Borda_{1K}$ (as will be discussed in Section 6.3). Thus, we also prefer $Borda_{1K}$ over these two *Borda* variants. As discussed in Section 6.1.1, $Borda_{full}$ and *Approval* behave similarly. We actually observe that they exhibit slightly better manipulation resistance compared to $Borda_{1K}$, as shown in Figure 11. However, as will be discussed in Section 6.3, they exhibit significantly worse stability compared to $Borda_{1K}$,

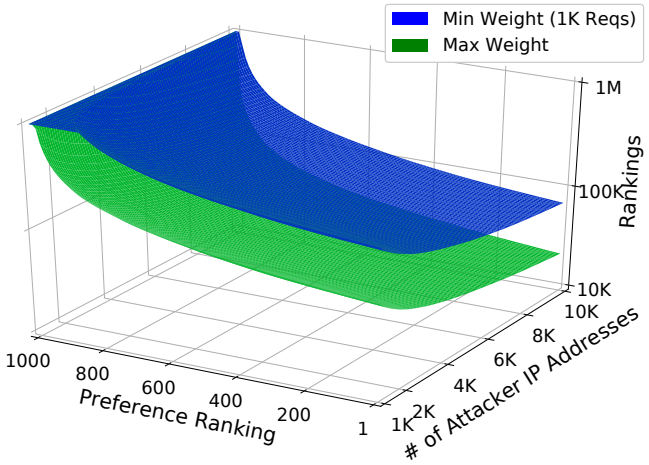


Figure 5: Average rankings of the attacker's target domain, depending on its per-address preference ranking and the # of attack addresses (when using weighted $Borda_{1K}$). We plot curves for the attacker addresses exhibiting minimum and maximum weights, representing attack lower bound and upper bound success, respectively. Note that as we consider domain preferences varying up to 1K, we consider the minimum weight where 1K requests are sent on attack addresses.

with approximately half of the top 1M list churning daily. Thus there is a trade-off between stability and manipulation resistance. We believe that $Borda_{1K}$ serves as a reasonable operating point, balancing manipulation resistance against stability while factoring in address-specific preferences.

Influence of Address-Specific Domain Preferences. Here, we investigate the influence of the attacker's address-specific domain preference rankings, focusing on $Borda_{1K}$ (as justified above). If an attacker fully controls an address, they can readily place their target domain as the address's top preference (as analyzed in Section 6.2.1). Thus here we consider only scenarios where the attacker is co-located with other users on the address, and must compete for the preference rankings. To isolate the effect of address domain preference, we evaluate cases where the attacker addresses have either the minimum or the maximum possible weight, and vary the address preference ranking of the attacker's target domain. The minimum address weight case corresponds to a lower bound on attack success, whereas the maximum weight case reflects an upper bound.

In Figure 5, we plot the average daily rank of the attacker's target domain, varying the number of attacker addresses and the address preference ranking of the target domain, using weighting and $Borda_{1K}$ voting. As expected, as the target domain's address preference ranking increases (with the same number of attacker addresses), it achieves a higher overall top list rank, although improving preference rankings results in increasingly diminishing returns.

To understand the attack strength needed to obtain a certain address preference (when attackers are co-located with benign clients), Figure 6 shows the distribution of the number of domain preference boosting requests that the attacker must send for addresses on a randomly sampled day, while varying the attack's active duration and the targeted address-specific preference ranking. For about 20% of addresses, an attacker can readily rank its target domain first with just 1 request. However, for most addresses, attackers must generate significant numbers of preference boosting requests to obtain a top preference ranking. Attacks that are active over an entire day still require hundreds to thousands of preference boosting requests to secure a top preference ranking. With lower active attack durations, attacks require significantly more requests to obtain a top preference. Thus, attackers must expend additional resources (sending more requests and remaining active for longer) for attack success when using domain preference.

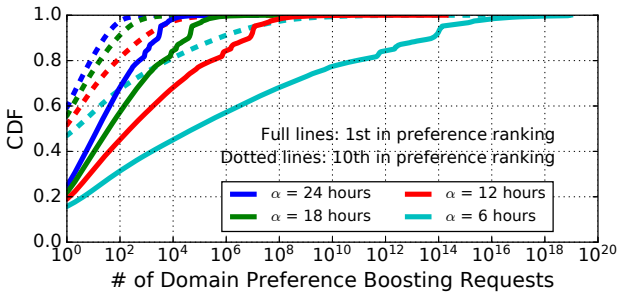


Figure 6: CDF of the # of domain preference boosting requests that an attacker must send to obtain a desired per-address preference ranking (ranked first versus 10th) for its target domain, on a randomly sampled day. α = attack active durations.

6.2.3 Comparison to Existing Top Lists

Next, we compare the manipulation resistance of our top list (created using weighted $Borda_{1K}$) to Umbrella and Tranco. Alexa does not use PDNS, so a direct comparison is infeasible, although Alexa is easy to manipulate even using a single IP address and a limited number of requests (as discussed in Section 2). Without visibility into Umbrella's data and method, we must compare through a real-world evaluation, performing (controlled) manipulation efforts and observing the differences in manipulation success. As mentioned in Section 2, we identified an additional method for easier DNS-based ranking manipulation (i.e., using open DNS resolvers to obtain IP diversity). Here we use this approach for evaluating top list manipulation with many IP addresses.

Experiment Method. We use two target domains under our control for testing manipulation efforts. For each attack IP address (i.e., an open DNS resolver), we trigger five DNS lookups for each target domain to be sent to both Umbrella and 114DNS's DNS resolvers, evaluating up to 20K attack addresses (while larger attacks are possible, we limit our evaluation due to ethical considerations). In essence, we cause requests from 20K different IP addresses to appear in Um-

brella and 114DNS's PDNS data, performing identical attacks on both top lists. We then monitor the target domains' rankings (performing our evaluation from Apr. 8–20, 2021). The manipulation approach is detailed in Appendix D.

Ethics. We conduct our manipulation experiment on our own domains that host webservers indicating that they are test servers. Using only two domains should have negligible impact on top lists overall. We also send a small number of requests per evaluated IP address, and rate limit our total requests generated to Umbrella's and 114DNS's DNS resolvers to minimize the traffic load induced.

Results. The two target domains successfully achieved a consistent ranking within Umbrella's top 10K each day (reaching a rank as high as 2,859). For our recommended top list construction (weighted $Borda_{1K}$), the two target domains obtained stable rankings between 29K to 41K (reaching at most a rank of 29,486) under the identical manipulation effort. While our comparison is limited to this black box evaluation, we observe that our ranking appears to be significantly more manipulation resistant compared to Umbrella (and Alexa).

When comparing to Tranco, we must combine Alexa and Umbrella manipulation. We note that default Tranco combines the Alexa, Umbrella, and Majestic top lists over a 30-day window. However, for a more direct and fair comparison of the two top list methods, we use daily snapshots of Alexa and Umbrella as Tranco input sources. Using daily snapshots allows us to directly compare the manipulation resistance of Tranco's algorithm with ours on an equal time scale. We also exclude Majestic when computing Tranco's rankings, as we focus on top lists that rank domains by user visits/traffic (as discussed in Section 2.1). Including Majestic, which ranks websites by their backlinks, further convolutes Tranco's domain popularity interpretation.

Figure 7 depicts the Tranco ranking an attacker can obtain, based on its manipulation of Alexa and Umbrella rankings. Tranco proportionally reweights the contributions of the two contributing top lists to account for the differences in list sizes (we discuss Alexa's size change in Section 2.2.1). Thus Tranco manipulation success differs when targeting only the Alexa top list compared to the Umbrella top list. In general, an attacker must obtain the same target ranking in both Alexa and Umbrella in order to obtain that target rank in Tranco. However, we note that as Alexa and Umbrella are easily manipulated with low-resource attacks, transitively Tranco's list can likewise be affected.

From Alexa manipulation experiments (through the Certify service), attackers can achieve a ∼2K ranking with ∼360K requests (sent by one IP address). For Umbrella manipulation, with 20K IP addresses (sending 5 requests each, 100K requests in total), attackers similarly achieve a ∼2.8K ranking. As a result, such attack resources achieve a ∼2K ranking in Tranco. In comparison, 20K IP addresses with 460K total requests (matching the total request volume for both Alexa and Umbrella manipulation) can only obtain a ranking of

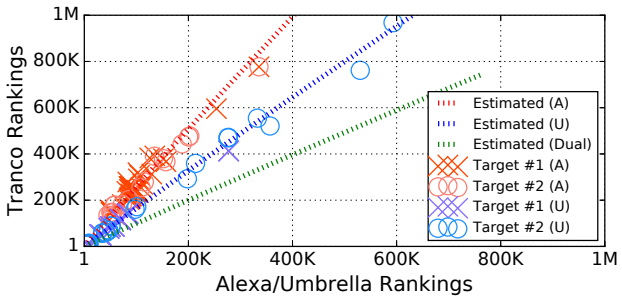


Figure 7: Rankings obtained in Tranco through Alexa (A) and Umbrella (U) manipulation. Data points (X's and O's) represent the observed Tranco rankings for our two targeted domains (labeled as "Target") during our Alexa and Umbrella manipulation experiments. The dotted lines represent the extrapolated relationship between a domain's Alexa/Umbrella ranking and its Tranco ranking (with "Dual" representing concurrent ranking in both Alexa and Umbrella).

∼20K in our list. Thus, our ranking also demonstrates more manipulation resistance compared to Tranco. (We note that our list differs from Tranco in how it resists manipulation, as discussed in Section 2.1.)

6.3 List Property: Stability

Here we evaluate how our design decisions impact the stability of our top lists. We quantify stability (similar to prior work [39, 45]) as the percentage of a top list for one day that remains on (i.e., intersects) the list for the next day (averaging across days), considering different portions/ranking ranges of the top list.

Design Decision: Using IP Weighting. We find that for all voting methods, stability does not vary noticeably when using weighting versus not (we elide details for space). There is only a 1.5% difference in the average stability for our top 1M list with and without IP weighting, when using $Borda_{1K}$. This likely arises because IP weights do not drastically shift from one day to the next. As argued in Section 6.2, we advocate for weighting as it provides both philosophical and manipulation resistance benefits, without impacting stability.

Design Decision: Choice of Voting Method. Table 1 lists the average stability of each top list across different ranking ranges. Overall, we observe that $Borda_{1K}$ exhibits the highest stability for its top 1M list, with over 80% of the list remaining the same day-to-day. *Approval* exhibits the highest churn in its top 1M list, with almost half of the list changing daily ($Borda_{full}$ behaves similarly, as discussed in Section 6.1.1). $Borda_{10}$ and $Borda_{100}$ also exhibit high churn (although not as extreme as *Approval*). We note that among the top 100K (or more highly ranked domains), all lists are similarly stable.

Given that *Dowdall*, $Borda_{10}$, and $Borda_{100}$ exhibit lower stability and worse manipulation resistance compared to $Borda_{1K}$ (as evaluated in Section 6.2), we advocate for the use

Ranking Range	Our Top Lists						Existing Top Lists		
	Borda$_{1K}$	**Dowdall**	**Approval**	**Borda$_{10}$**	**Borda$_{100}$**	**Borda$_{full}$**	**Alexa**	**Umbrella**	**Tranco**
Top 1K	97.7%	97.7%	97.6%	97.4%	97.5%	97.6%	84.9%	94.9%	90.8%
Top 10K	97.2%	97.0%	97.0%	96.4%	96.8%	97.0%	73.8%	94.4%	85.6%
Top 100K	94.1%	94.2%	94.1%	91.5%	93.9%	94.1%	58.1%	93.0%	82.0%
Top 1M	81.7%	80.8%	51.3%	62.8%	79.2%	51.6%	44.9%	87.0%	72.7%

Table 1: For various top list constructions, we show the average list stability (the average percentage of intersecting domains for a top list on consecutive days) for different portions/ranking ranges of the top list.

of *Borda$_{1K}$* over these other voting methods. When compared to *Approval* and *Borda$_{full}$* (which behave similarly), there is a trade-off between manipulation resistance and stability (as discussed in Section 6.2). While a less stable list is not necessarily inherently worse or less realistic, prior work [39,45] has noted the challenges in using unstable lists. Thus, we believe *Borda$_{1K}$* is a suitable voting method choice as it offers the highest stability with reasonable manipulation resistance.

Comparison to Existing Top Lists. Again we compare our top list (using weighted *Borda$_{1K}$*) to existing top lists (shown in Table 1). We note that existing lists use different windows of data, and there exists a trade-off where top lists built on longer-term traffic metrics can provide more stability, but such lists are less responsive to real-time changes. Prior work [39, 45] observed that Alexa exhibits low stability, with its top 1M's daily churn rates reaching over 50% of domains (we observed an average stability of 45% over our 6-month data analysis window). Umbrella exhibits higher stability in its top 1M; we observed an 87% average stability, while prior work [45] found a 20% churn rate (i.e., 80% stability). However, Umbrella's ranking is constructed on two-day windows of data [39]. Our evaluated version of the Tranco top 1M, which combines Alexa and Umbrella (as discussed in Section 6.2.3), exhibits a 73% average stability[11] (between Umbrella's and Alexa's stability rates). Thus, our chosen top list, which also exhibits about 82% average stability in the top 1M while using daily snapshots, offers favorable stability and reactivity properties compared to existing lists. (We note that regardless of Umbrella's two-day data window, our list also exceeds the stability of Umbrella in the top 100K.)

6.4 Top List Characteristics

We characterize our top list constructed using weighted *Borda$_{1K}$* (as previously justified), along dimensions evaluated for existing top lists in prior work [39, 45].

Top List Similarity. Our top 1M list shares 14.4% of its domains with Umbrella's top 1M on average, and 18.3% with Alexa. This is expected, as prior work observed limited overlap between existing top lists due to data source differences; Alexa and Umbrella top 1M lists overlapped on only 15% of domains [45]. Our input data source has broad domain coverage though (covering 100% of domains observed in Alexa and Umbrella over our data analysis window).

[11] The default Tranco ranking, which combines Alexa, Umbrella, and Majestic over 30 days, exhibits a 1% daily churn rate, but given the much larger data time window, it is significantly less reactive.

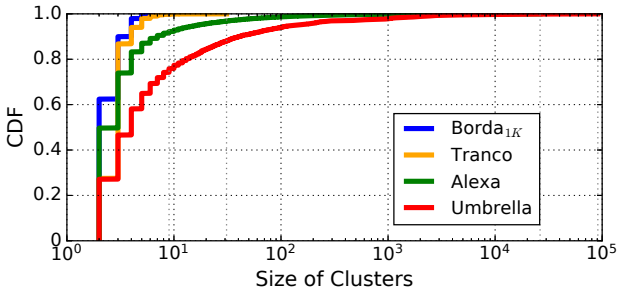


Figure 8: CDF of the sizes of alphabetically-ranked domain clusters among the different top 1M lists, for a randomly selected day.

SLD Coverage. Our top 1M list is distributed across an average of 476K distinct SLDs a day, which indicates a wide diversity of Internet services covered. Among the most popular domains (top 1K), services are more concentrated across 228 distinct SLDs (on average).

TLD Coverage. Our top 1M list covers on average 532 valid TLDs a day (∼36% of all TLDs [28]). Among the top 1K, there are 10 TLDs on average, which is similar to Umbrella [45]. Note that high TLD coverage is not necessarily desirable or expected for a top list, as prior work showed that most traffic skews towards a small number of TLDs [39]. Our input data source has broad TLD coverage though (covering 99.9% of TLDs, as discussed in Section 4).

Statistical Significance of Lower-Ranked Domains. In Alexa, rankings beyond the top 100K have limited statistical meaning, as these domains often receive only a single request (as discussed in Section 2.2.1). For comparison, we evaluate the statistical significance of our list's lower-ranked domains. Considering the bottom 100K domains in our top 1M list, we receive enough data per domain to make meaningful analysis. For a randomly selected day, we analyze the distribution of the number of requests and IP addresses for these bottom 100K domains. The median number of requests per domain is ∼1.5K (ranging from ∼1K to over 1M requests). The median number of addresses per domain is ∼500 (ranging from ∼100 to ∼10K). Thus, we observe a non-trivial amount of DNS traffic for even the lowest-ranked domains in our list.

Alphabetically-Ranked Domain Clusters. Prior studies [39, 43, 44] found that Alexa and Umbrella contain large alphabetically-sorted clusters of domains, likely representing domains with identical ranking scores that are arbitrarily ordered alphabetically. We evaluate the distribution of such clusters for our top 1M list compared to Alexa, Umbrella, and Tranco, depicted in Figure 8 for a randomly selected day. In our list, the largest cluster of domains with identical ranking scores is 9 domains, while Umbrella and Alexa exhibit clusters of over 10K domains. Tranco largely avoids the massive ranking clusters in Alexa and Umbrella, as it merges both lists, interleaving domains from clusters within both lists. Nonetheless, Tranco still exhibits larger clusters than our list, with a cluster of 31 domains. Thus, our list avoids ranking clusters more than existing lists.

7 Conclusion

In this paper, we systematically explored the construction of a robust, manipulation-resistant, and stable domain top list from scratch. Motivated by the limitations of existing top lists, particularly concerning a lack of transparency, low stability, and easy manipulation, we designed a voting-based domain ranking method where individual IP addresses express their domain preferences, and a weighted vote is performed across all addresses to construct a global top list. We realized our list in practice using an extensive passive DNS dataset, and evaluated how different design choices impact our list and its characteristics. Ultimately, we identified trade-offs in the behavior of our top list, although we recommend one specific construction method that offers favorable stability and manipulation resistance properties (while also being transparent). We demonstrated that our top list outperforms existing ones across the different list properties considered. Thus, we believe our top list construction method can serve as a valuable contribution to the networking and security community.

This work is truly an initial exploration of the nature of top lists though. Future work can investigate improving the robustness of our list construction method (e.g., using *hop-count filtering* [30] to combat IP spoofing attacks [46] on PDNS-based top lists), better quantifying domain popularity characteristics (e.g., characterizing user populations behind IPs), and evaluating the longitudinal characteristics of domain ecology using our top list. We aim for our work to help guide the use of Internet top lists for networking and security research, while driving further investigation into this space.

Acknowledgement

We thank the anonymous reviewers and our shepherd Michel van Eeten for valuable feedback on improving this paper. We also thank Victor Le Pochat for providing guidance on Tranco.

This work was supported in part by the National Natural Science Foundation of China (U1836213, U19B2034, 62102218). Baojun Liu is partially supported by the Shuimu Tsinghua Scholar Program. Opinions expressed in this paper are solely those of the authors.

References

[1] 114DNS. https://www.114dns.com/, 2022.

[2] Alexa Specialist. http://www.improvealexaranking.com/, 2022.

[3] Dnsthingy. https://www.dnsthingy.com/, 2022.

[4] General Data Protection Regulation (GDPR). https://gdpr-info.eu/, 2022.

[5] IPIP. https://en.ipip.net/, 2022.

[6] MaxMind. https://www.maxmind.com/, 2022.

[7] Quad9. https://www.quad9.net/, 2022.

[8] Rankboostup. https://rankboostup.com/, 2022.

[9] RankStore. https://www.rankstore.com/check-alexa-rank/, 2022.

[10] Alexa. Alexa Top 1 Million. http://s3.amazonaws.com/alexa-static/top-1m.csv.zip, 2022.

[11] Alexa. How are Alexa's traffic rankings determined? https://support.alexa.com/hc/en-us/articles/200449744-How-are-Alexa-s-traffic-rankings-determined-, 2022.

[12] Alexa. How do I get my site's metrics Certified? https://support.alexa.com/hc/en-us/articles/200450354-How-do-I-get-my-site-s-metrics-Certified-, 2022.

[13] Mario Almeida, Alessandro Finamore, Diego Perino, Narseo Vallina-Rodriguez, and Matteo Varvello. Dissecting DNS Stakeholders in Mobile Networks. In *International Conference on emerging Networking EXperiments and Technologies (CoNext)*, 2017.

[14] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. Understanding the Mirai Botnet. In *USENIX Security Symposium*, 2017.

[15] Manos Antonakakis, Roberto Perdisci, David Dagon, Wenke Lee, and Nick Feamster. Building A Dynamic Reputation System for DNS. In *USENIX Security Symposium*, 2010.

[16] Manos Antonakakis, Roberto Perdisci, Wenke Lee, Nikolaos Vasiloglou, and David Dagon. Detecting Malware Domains at the Upper DNS Hierarchy. In *USENIX Security Symposium*, 2011.

[17] John J Bartholdi, Craig A Tovey, and Michael A Trick. The Computational Difficulty of Manipulating An Election. *Social Choice and Welfare*, 6(3):227–241, 1989.

[18] Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis. In *Network and Distributed System Security Symposium (NDSS)*, 2011.

[19] Thomas Callahan, Mark Allman, and Michael Rabinovich. On Modern DNS Behavior and Properties. *ACM SIGCOMM Computer Communication Review*, 43(3):7–15, 2013.

[20] Sebastian Castro. Domain Popularity Ranking Revisited. https://cdn.nzrs.net.nz/88vkAeP4GA9wm/r4dXMp9Vn0EV-/Domain%20Popularity%20Ranking%20Revisited.pdf, 2016.

[21] Chrome Web Store. Alexa Traffic Rank. https://chrome.google.com/webstore/detail/alexa-traffic-rank/cknebhggccemgcnbidipinkifmmegdel?hl=en, 2022.

[22] Cisco. OpenDNS. https://www.opendns.com/, 2022.

[23] Firefox Browser ADD-ONS. Alexa Rank. https://addons.mozilla.org/en-US/firefox/addon/minimalist-alexa-rank/, 2022.

[24] Pawel Foremski, Oliver Gasser, and Giovane CM Moura. DNS Observatory: The Big Picture of the DNS. In *ACM Internet Measurement Conference (IMC)*, 2019.

[25] Hongyu Gao, Vinod Yegneswaran, Yan Chen, Phillip Porras, Shalini Ghosh, Jian Jiang, and Haixin Duan. An Empirical Reexamination of Global DNS Behavior. In *ACM Special Interest Group on Data Communication (SIGCOMM)*, 2013.

[26] Hongyu Gao, Vinod Yegneswaran, Jian Jiang, Yan Chen, Phillip Porras, Shalini Ghosh, and Haixin Duan. Reexamining DNS from A Global Recursive Resolver Perspective. *IEEE/ACM Transactions on Networking*, 24(1):43–57, 2014.

[27] Chris Grier, Lucas Ballard, Juan Caballero, Neha Chachra, Christian J. Dietrich, Kirill Levchenko, Panayiotis Mavrommatis, Damon McCoy, Antonio Nappa, Andreas Pitsillidis, Niels Provos, M. Zubair Rafique, Moheeb Abu Rajab, Christian Rossow, Kurt Thomas, Vern Paxson, Stefan Savage, and Geoffrey M. Voelker. Manufacturing Compromise: the Emergence of Exploit-as-a-service. In *ACM Conference on Computer and Communications Security (CCS)*, 2012.

[28] Internet Assigned Numbers Authority (IANA). TLDs - Alphabetical by Domain. https://data.iana.org/TLD/tlds-alpha-by-domain.txt, May 2021.

[29] Collin Jackson, Adam Barth, Andrew Bortz, Weidong Shao, and Dan Boneh. Protecting Browsers from DNS Rebinding Attacks. *ACM Transactions on the Web (TWEB)*, 3(1):1–26, 2009.

[30] Cheng Jin, Haining Wang, and Kang G Shin. Hop-count Filtering: An Effective Defense against Spoofed DDoS Traffic. In *ACM Conference on Computer and Communications Security (CCS)*, 2003.

[31] Maurice G Kendall. A New Measure of Rank Correlation. *Biometrika*, 30(1/2):81–93, 1938.

[32] Majestic. Majestic Million – Reloaded! https://blog.majestic.com/company/majestic-million-reloaded/, 2011.

[33] Majestic. Majestic Million CSV now free for all, daily. https://blog.majestic.com/development/majestic-million-csv-daily, 2012.

[34] Majestic. The Majestic Million. https://majestic.com/reports/majestic-million, 2022.

[35] Alexander Mayrhofer, Michael Braunöder, and Aaron Kaplan. DNS Magnitude - A Popularity Figure for Domain Names, and its Application to L-root Traffic. In *ICANN DNS Symposium*, 2020.

[36] Nina Narodytska and Toby Walsh. The Computational Impact of Partial Votes on Strategic Voting. *arXiv preprint arXiv:1405.7714*, 2014.

[37] OpenDNS. PhishTank. http://phishtank.org/, 2022.

[38] Jeffrey Pang, Aditya Akella, Anees Shaikh, Balachander Krishnamurthy, and Srinivasan Seshan. On the Responsiveness of DNS-based Network Control. In *ACM Internet Measurement Conference (IMC)*, 2004.

[39] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. Tranco: A Research-oriented Top Sites Ranking Hardened against Manipulation. In *Network and Distributed System Security Symposium (NDSS)*, 2019.

[40] Moheeb Abu Rajab, Fabian Monrose, Andreas Terzis, and Niels Provos. Peeking through the Cloud: DNS-based Estimation and Its Applications. In *International*

Conference on Applied Cryptography and Network Security (ACNS), 2008.

[41] Juan Ramos. Using TF-IDF to Determine Word Relevance in Document Queries. In *Instructional Conference on Machine Learning (ICML)*, 2003.

[42] Benjamin Reilly. Social Choice in the South Seas: Electoral Innovation and the Borda Count in the Pacific Island Countries. *International Political Science Review*, 23(4):355–372, 2002.

[43] Walter Rweyemamu, Tobias Lauinger, Christo Wilson, William Robertson, and Engin Kirda. Clustering and the Weekend Effect: Recommendations for the Use of Top Domain Lists in Security Research. In *International Conference on Passive and Active Network Measurement (PAM)*, 2019.

[44] Walter Rweyemamu, Tobias Lauinger, Christo Wilson, William Robertson, and Engin Kirda. Getting Under Alexa's Umbrella: Infiltration Attacks Against Internet Top Domain Lists. In *International Conference on Information Security (ISC)*, 2019.

[45] Quirin Scheitle, Oliver Hohlfeld, Julien Gamba, Jonas Jelten, Torsten Zimmermann, Stephen D Strowes, and Narseo Vallina-Rodriguez. A Long Way to the Top: Significance, Structure, and Stability of Internet Top Lists. In *ACM Internet Measurement Conference (IMC)*, 2018.

[46] Kyle Schomp, Onkar Bhardwaj, Eymen Kurdoglu, Mashooq Muhaimen, and Ramesh K Sitaraman. Akamai DNS: Providing Authoritative Answers to the World's Queries. In *ACM Special Interest Group on Data Communication (SIGCOMM)*, 2020.

[47] Craig A Shue, Andrew J Kalafut, Mark Allman, and Curtis R Taylor. On Building Inexpensive Network Capabilities. *ACM SIGCOMM Computer Communication Review*, 42(2):72–79, 2012.

[48] Laura Spaventa. Behind the Scenes Part II: What Makes HARO So Successful in Getting You Media Coverage. https://www.cision.com/2012/10/behind-the-scenes-part-ii-what-makes-haro-so-successful-in-getting-you-media-coverage/, 2012.

[49] Tor Community. Tor Project. https://www.torproject.org/, 2022.

[50] Cisco Umbrella. Cisco Umbrella 1 Million. https://umbrella.cisco.com/blog/cisco-umbrella-1-million, 2016.

[51] Cisco Umbrella. Umbrella Popularity List. http://s3-us-west-1.amazonaws.com/umbrella-static/index.html, 2022.

[52] Cisco Umbrella. Umbrella Popularity List—Top Million Domains. https://docs.umbrella.com/investigate-api/docs/top-million-domains, 2022.

[53] Tielei Wang, Yeongjin Jang, Yizheng Chen, Simon Chung, Billy Lau, and Wenke Lee. On the Feasibility of Large-scale Infections of iOS Devices. In *USENIX Security Symposium*, 2014.

[54] H Peyton Young. Condorcet's Theory of Voting. *The American Political Science Review*, pages 1231–1244, 1988.

A Undisclosed Alexa Changes

Here we provide another example of an undisclosed Alexa data collection change, demonstrating its top list's lack of construction consistency. Prior work [39, 45] observed that in early 2018, the telemetry collected by Alexa from its browser extension included the domains visited and a user ID specific to the browser instance. At that time, Alexa did not limit the number of user IDs observed on the same IP address nor filter out any telemetry. Subsequent work [44] identified that Alexa had begun filtering out repeat visits to the same domain from the same user ID, presumably to combat manipulation efforts. Aligning with this prior finding, in our reverse-engineering of the Alexa browser extension in Dec. 2020, we identified that the extension does not send telemetry to Alexa when a URL is revisited within the same browser within a certain time period. This notably filters out legitimate repeat visits to domains by benign users, undercounting domain visits.

B Manipulating Alexa through Certify

Here, we detail our Alexa manipulation approach using Alexa's Certify service. We reverse-engineered the Alexa Certify JavaScript snippet [12], as of Feb. 2021. When visiting an Alexa Certified website, the Certify script provides visit metrics to Alexa's data collection endpoint along with a *user_cookie* data field value used to distinguish site visitors. Thus, one can forge fake visits to the Certified website by generating requests to the Alexa data collection endpoint using the same Certify ID (for the target website) with a large number of unique *user_cookie* values. Figure 9 shows the Alexa rankings we obtained for a manipulation experiment targeting two of our own Certified domains (as discussed in Section 2.2.1).

C Alexa's Bias Towards Certified Domains

Here, we discuss our analysis of Alexa Certified domains. Subscribers of Alexa's Certify service can query whether

[12] https://certify-js.alexametrics.com/atrk.js

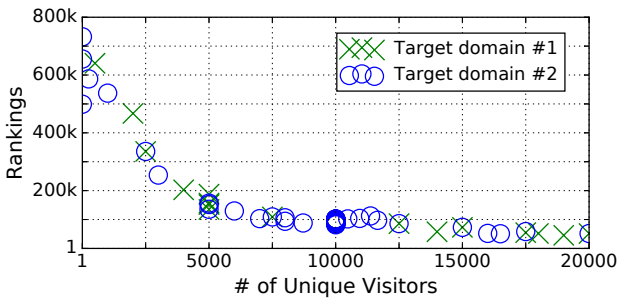


Figure 9: Rankings obtained in Alexa using different numbers of forged distinct visitors to our two Alexa Certified test domains (from Feb. to Apr. 2021).

other domains on Alexa's top list are also Certified sites. We subscribed to the Certify service and queried for the status of domains in the Alexa top list on March 21, 2021. We identify that 3.7K out of 487.8K Alexa top list domains (0.7%) use the Certify service. While some such domains are highly ranked, 80% are ranked beyond the top 10K and nearly half beyond the top 100K. Using our test domains subscribed to the Certify service (as discussed in Section 2.2.1), we identified that as long as a Certified domain receives even a single page visit in a day, it will be ranked in the Alexa top list, demonstrating that Alexa is biased towards Certified domains (although Alexa's methods remain undisclosed).

D PDNS Manipulation using Open DNS Resolvers

To manipulate PDNS data collected at a recursive resolver (e.g., those of Umbrella or 114DNS), without an attacker directly controlling a large number of distinct IP addresses, we develop a manipulation approach that leverages open DNS resolvers across the Internet. Figure 10 illustrates the manipulation workflow.

We first directly send a query for a target domain to the PDNS recursive resolver (Step 1), which will recursively query the target domain's authoritative name server (Step 2), caching the authoritative response (Step 3). We then reconfigure the target domain's authoritative name server on-the-fly to be set to the PDNS recursive resolver itself (Step 4). We query open DNS resolvers across the Internet for the target domain (Step 5), each of which will subsequently recursively query the domain's authoritative name server, which is now the PDNS resolver (Step 6). So long as the target domain's DNS record remains cached at the PDNS resolver, it will reply to the open DNS resolvers with valid DNS responses. From the PDNS data collection perspective, the recursive resolver has received and responded to DNS lookups for the target domain from various IP addresses (belonging to open DNS resolvers), thus manipulating the PDNS data (Step 7). It is critical that the target domain's DNS record is cached at the PDNS recursive resolver throughout the manipulation effort.

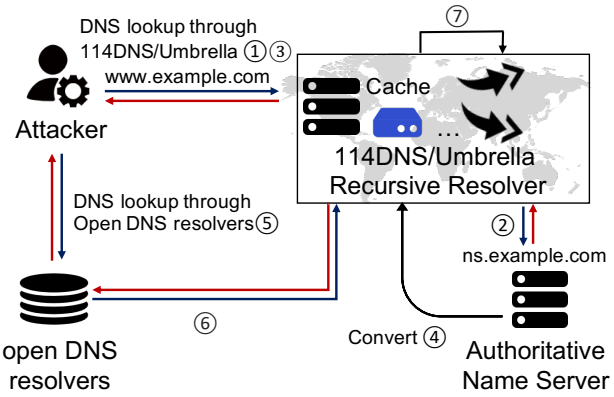
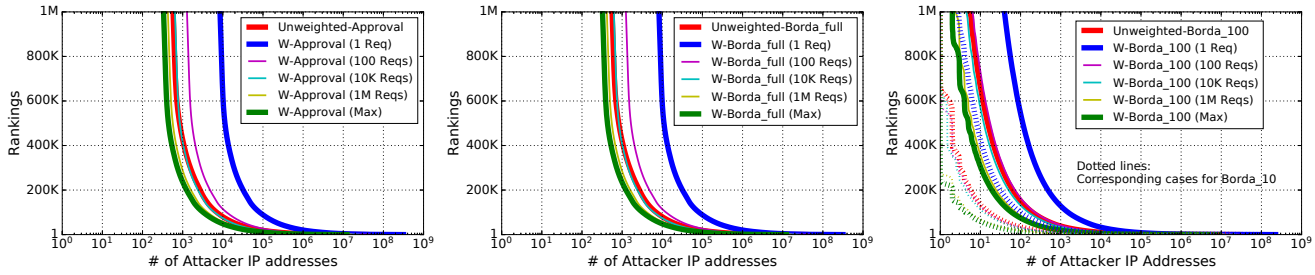


Figure 10: PDNS manipulation through open DNS resolvers. The blue arrows represent DNS queries and the red arrows represent DNS responses.

If not, the recursive resolver will issue a fresh query to the domain's authoritative name server, which has been set to itself, resulting in a *SERVFAIL* error that can prevent PDNS logging. However, as long as the attacker controls the target domain's initial authoritative name server (a reasonable threat model), the attacker can configure a long TTL for the initial authoritative record, resulting in long-term caching.

Experiment Implementation. Here we discuss the details of the Umbrella and 114DNS manipulation experiment described in Section 6.2.3. In Jan. 2021, we conducted an Internet-wide scan for open DNS resolvers on port 53, finding 1.7M distinct IP addresses with an open DNS resolver. Using two servers (located in Beijing, China and Fremont, US), we issued DNS requests to the 114DNS and Umbrella recursive resolvers as well as the open DNS resolvers, following our PDNS manipulation workflow. (Note, we never test the full set of open DNS resolvers to limit the total traffic load to the PDNS recursive resolvers.) We used two of our own domains as targets (which host webservers indicating that they are test servers), where we controlled the domains' authoritative name servers.

Both 114DNS and Umbrella load-balance traffic to its recursive resolvers through anycast. By tracing DNS queries from our servers to the PDNS recursive resolvers, we observe that even from the same source IP address, DNS lookups can be routed to different recursive resolver servers (due to anycasting). To cache our target domain's authoritative records at many 114DNS and Umbrella recursive resolver servers, we first repeatedly issue 1K DNS lookups for each target domain from both of our servers (where different lookups may be routed to different recursive resolver servers, resulting in that resolver caching our target domain's authoritative record). Then when performing PDNS manipulation, we issue 5 repeat DNS queries for our target domain to each open DNS resolver, to overcome intermittent packet loss.



(a) *Approval* (b) $Borda_{full}$ (c) $Borda_{100}$ and $Borda_{10}$

Figure 11: Average rankings of the attacker's target domain, depending on different top list construction parameters (weighted versus unweighted; using *Approval*, $Borda_{full}$, $Borda_{100}$, or $Borda_{10}$) and attack strengths (different numbers of attacker IP addresses and weight boosting DNS requests). Curves labeled as "W-" represent weighted top lists, and the number of weight boosting requests are listed in parentheses (with "Max" representing the attacker obtaining the maximum normalized weight).

E DNS Caching Considerations

Here, we investigate our PDNS data to understand if we can reliably factor in DNS caching behavior. A DNS record is cached for a certain amount of time, the TTL. The authoritative name server sets a TTL for a DNS record (the ATTL), but the TTL that a client receives from a recursive resolver's response (the RTTL) depends on how long the record has been cached at the resolver, and thus changes over time and with each DNS response. We note that previous studies [13, 19, 38, 47] have observed that recursive resolvers across the Internet often do not respect the ATTL (i.e., recursive resolvers select their own maximum/minimum RTTL values irrespective of the ATTL). Thus, for many DNS datasets (including our own), caching behavior is dependent on the RTTL rather than the ATTL, and thus we focus on RTTL considerations here.

Ultimately, we find that due to the following three issues, it is infeasible for us to reliably predict DNS caching behavior from the PDNS vantage point. Thus, in our work, we do not attempt to adjust for DNS caching.

E.1 CNAME Caching Discrepancies

Prior work [45] reported that 44% of the Alexa top 1M and 28% of the Umbrella top 1M use CNAME records, and we observe similar CNAME usage in our PDNS dataset (for nearly 33% of DNS lookups). We investigate how different OSes handle DNS caching based on the TTLs observed when resolving a chain of CNAME records (including both CNAME and A records). We find that Windows 10 caches based on the minimum TTL observed in the CNAME resolution chain whereas Unix-based systems (including Ubuntu and Mac OS X) use only the first CNAME record's TTL. As a consequence, clients on different OSes will cache CNAME records for different durations. Our PDNS data does not provide visibility into client OSes, preventing reliable caching predictions for a substantial fraction of DNS lookups.

E.2 Client-Specific Caching Policies

Beyond CNAME-related caching differences across OSes, different client software also exhibit varying caching behaviors. Many browsers (e.g., Opera, Chrome, Firefox) implement their own DNS caches rather than rely on OS-level caches (particularly to defend against DNS rebinding attacks [29]). For example, Chrome[13] and Firefox[14] cache each domain for 60 seconds, irrespective of DNS record RTTLs. As we cannot determine the specific software clients used from our PDNS data, we again lack visibility into caching dynamics.

E.3 Unclear DNS Cache Boundaries

Beyond local device caching, DNS caching also occurs at the local network resolvers. However, given an IP address, it is unclear where DNS caches may be located. Multiple IP addresses may rely on the same DNS cache, meanwhile, multiple clients behind the same IP address (due to network address translation) may use different DNS caches. We evaluated whether DNS caching might be occurring at the IP address granularity. If so, we would expect that if a PDNS recursive resolver sent a DNS response for a domain to an IP address with a given RTTL, that IP address would not generate subsequent DNS lookups to the recursive resolver for that domain until the RTTL period passes. We observe that for a randomly-sampled one-hour window of our PDNS data, this behavior was seen for only 5% of IP addresses, indicating that caching boundaries are not typically at the IP-level granularity, which is the granularity of data provided by PDNS.

F Manipulation Resistance of *Approval*, $Borda_{full}$, $Borda_{100}$, $Borda_{10}$

Figure 11 depicts the average attack success for different top lists and attack parameters.

[13] https://bugs.chromium.org/p/chromium/issues/detail?id=164026
[14] http://kb.mozillazine.org/Network.dnsCacheExpiration

A Artifact Appendix

A.1 Abstract

In this work, we developed a voting-based domain ranking method that operates on passive DNS (PDNS) data to construct a domain top list. We open-source our top list construction implementation at *https://github.com/SecRank/secrank-sourcecode/releases/tag/v1.0.0*, to provide transparency into the design of our ranking method. The code provided (written in Scala) processes proprietary PDNS data to compute a daily top 1M domains list, running in a distributed fashion using Apache Spark on YARN. As we are unable to release the raw proprietary PDNS data used, the code is not directly runnable. Instead, it serves as a reference for understanding the details of our ranking method, and as a template that can be modified for other PDNS datasets and computing environments.

As our top list design achieves favorable stability and manipulation resistance properties, we also provide public access to a regularly updated domain top list constructed using our ranking method at *https://secrank.cn/topdomain*, for others to use in their research. After registering for an account on the website, a user can download a daily top 1M domains list as well as historical top lists.

A.2 Artifact check-list (meta-information)

- **Algorithm:** We provide a new voting-based domain ranking algorithm that operates on PDNS data, where the domain preferences of individual IP addresses are first computed, and then a global top list ranking is produced by applying a voting scheme across all IP addresses.
- **Security, privacy, and ethical concerns:** This artifact does not raise any security, privacy, or ethical concerns.
- **Publicly available?:** Our top list construction implementation is publicly available at https://github.com/SecRank/secrank-sourcecode/releases/tag/v1.0.0. A regularly updated domain top list constructed using our ranking method is publicly available at *https://secrank.cn/topdomain*, upon registering for a user account.
- **Code licenses?:** MIT License.
- **Archived?:** Our top list construction implementation is publicly archived at https://github.com/SecRank/secrank-sourcecode/releases/tag/v1.0.0.

A.3 Description

A.3.1 How to access

Source Code Access. We open-source our top list construction implementation at *https://github.com/SecRank/secrank-sourcecode/releases/tag/v1.0.0*. Our implementation relies on proprietary PDNS data, which we are unable to release for privacy and commercial reasons. Thus, the code is not directly runnable, and rather provides transparency into the design of our domain ranking method.

Those interested may adapt our code for their own PDNS data/format and computing environment. Note that our implementation uses Apache Spark on YARN, with input and output data stored in HDFS. For users with a similar computing environment, our code can be most directly applied by providing the proper input and output data file paths, as well as adjusting the data field names extracted from the input data. Further details are provided in the *README.MD* file.

The code repository consists of the following main files:

- *README.MD*: The README file providing guidance on using the code.
- *TopFQDNDailyRelease.scala*: The main algorithm source code file, containing detailed comments for each algorithm component that reference the relevant sections in our paper describing the algorithm's design. We additionally document the input/output file paths and data formats that must be modified if adapting this code for other PDNS datasets.
- *pom.xml*: The XML file that contains information about the software package and configuration details (including software and library dependencies).
- *submit.sh*: The shell script to submit the Spark application to a YARN cluster.

Daily Top 1M Domains List Access. Public access to a regularly updated top 1M domains list is available at https://secrank.cn/topdomain, through registering for a free account. With an activated account, users can download the latest daily list as well as historical lists.

A.3.2 Hardware dependencies

N/A

A.3.3 Software dependencies

Our released implementation is written in Scala and runs on Apache Spark on YARN. While our released code is not directly runnable, those modifying it for use will likely require IntelliJ IDEA, Java 1.8, Maven JDK 1.8, Scala 2.11.8, Apache Spark 2.4.5, and Hadoop 2.7.2. (These dependencies are also shown in the *pom.xml* file in the release package.)

A.3.4 Data sets

The source code relies on proprietary PDNS data, which we are unable to release for privacy and commercial reasons.

A.3.5 Models

N/A

A.3.6 Security, privacy, and ethical concerns

N/A

A.4 Installation

While our released code is not directly runnable (as we cannot release our raw input PDNS data), we provide guidance on how one could modify the code to run on their own input PDNS dataset. As our implementation is executed via Apache Spark on YARN, we assume a similar computing environment (i.e., Java 1.8, Maven JDK 1.8, Scala 2.11.8, Apache Spark 2.4.5, and Hadoop 2.7.2).

1. We suggest using IntelliJ IDEA to create an Apache Maven project, and replacing the default *pom.xml* file with the *pom.xml* file in our Github repository (which contains all dependency configurations and package requirements).
2. Next, place *TopFQDNDailyRelease.scala* in the path *$PROJECT_PATH$/src/main/java/com/secrank/examples/*.
3. In *TopFQDNDailyRelease.scala*, modify the *trends_path* and *access_path* variables to reference the input PDNS data file paths on HDFS, and also modify accordingly the output file path (in the code's final stage).
4. As documented in the comments of *TopFQDNDailyRelease.scala*, the code assumes certain fields are present in the input data format. If those fields are not available, either the source data must be modified to provide these fields, or the field names must be adjusted accordingly in the code to reflect the source data.
5. After modifying the code, package the Maven project into a JAR file, upload this JAR file to your Spark client, and execute *submit.sh* to submit the Spark application to the YARN cluster. After the code fully executes, the output will contain the top 1M domains list.

A.5 Experiment workflow

N/A

A.6 Evaluation and expected results

Users are expected to modify the code for their own input PDNS data and computing environments, with the expected output being a top 1M domains list computed using our domain ranking method.

A.7 Experiment customization

N/A

A.8 Notes

N/A

A.9 Version

Based on the LaTeX template for Artifact Evaluation V20220119.