A Tree in a Tree: Measuring Biases of Partial DNS Tree Exploration

Florian Steurer[1,2][0009−0003−7767−7386], Anja Feldmann[1][0000−0002−5530−699], and Tobias Fiebig[1][0000−0002−0163−5134]

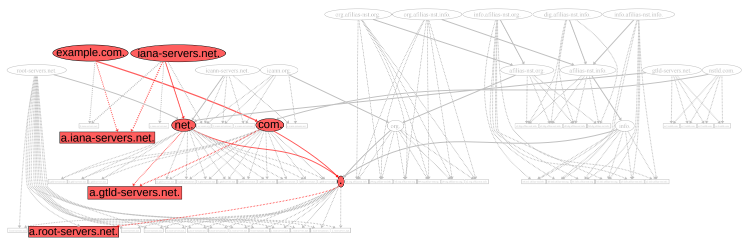
[1] Max Planck Institute for Informatics, Saarbruecken, Germany
{fsteurer,anja,tfiebig}@mpi-inf.mpg.de
[2] Saarland University, Saarbruecken, Germany

Abstract. The Domain Name System (DNS) is a cornerstone of the Internet. As such, it is often the subject or the means of network measurement studies. Over the past decades, the Internet measurement community gathered many lessons-learned and captured them in widely available measurement toolchains such as ZDNS and OpenINTEL as well as many papers. However, for feasibility, these tools often restrict DNS tree exploration, use caching, and other intricate methods for reducing query load. This potentially hides many corner cases and unforeseen problems. In this paper, we present a system capable of exploring the full DNS tree. We gather 87 TB of DNS data covering 812M domains with over 85B queries over 40 days. Using this data, we replicate four earlier studies that used feasibility and time-optimized DNS datasets. Our results demonstrate the need for care in selecting which limitations regarding the perspective on DNS can be accepted for a given research question and which may alter findings and conclusions.

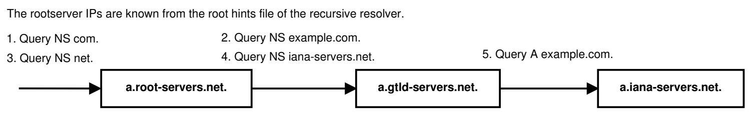
1 Introduction

The Domain Name System (DNS) was introduced in the late 1980s to replace the host file [35] for mapping names to numbers. Today, it has become one of the cornerstones of the Internet. DNS is continuously evolving. It has seen a multitude of additions, e.g., to enhance its security [5], or to facilitate additional use-cases [37, 71, 69]. This resulted in almost 300 RFCs related to DNS [64], making it a canonical example of a complex protocol. Not surprisingly, this complexity facilitates misconfigurations and corner cases. Hereby, part of the complexity arises from the fact that DNS is a hierarchical distributed database, a tree, where different name servers (NSs) are responsible for different branches.

In the past, DNS has been involved in many network studies, either as *subject* or as *means*, i.e., for studying other aspects of the Internet with help of the DNS. Studies about DNS include those on authoritative DNS (e.g., [11, 55]), recursive DNS (e.g., [38, 12, 31]), and those on DNS features (e.g., [13, 43]). Studies leveraging DNS data include questions regarding cloud deployments [28], IPv6 scans [26, 8], or email [16, 48, 47]. Yet, DNS is a notoriously difficult protocol to measure efficiently and without (accidentally) causing harm [25] due to its sheer size, complexity, and abundance of corner cases.



(a) Full DNS resolution graph for `A example.com.`



(b) Minimal set of queries for resolving `A example.com.`

Fig. 1: DNS tree for querying `A example.com.`, including all zones (round) and NSs (square) that *may* be involved in a resolution (Figure 1a). The minimal set of queries, five, for the best case is shown in Figure 1b. It assumes that glue for resolving `example.com` can be trusted. The corresponding zones and authoritative servers of Figure 1a are highlighted in red. ZDNS would explore a sub-tree close to this subset, while our system explores *all* possible paths (gray).

Past efforts have led to methodological insights and measurement toolchains for conducting DNS studies. Among the lessons learned are that DNS replies may depend on the probe's vantage point [68] and that caching can reduce scan duration as well as measurement overhead/impact [41]. Hereby, ZDNS [41]–a tool for performing active DNS queries—has made large-scale DNS measurements feasible. ZDNS leverages adaptive caching, external DNS-resolution services, and–by default[3]–opportunistic traversal of the DNS tree, i.e., not all possible paths in the DNS tree are evaluated (see also Section 3.4). OpenINTEL [68] uses Unbound for DNS resolution, which will terminate name resolution when an answer is found without exploring additional paths, i.e., also performs opportunistic traversal. This is in line with OpenINTEL's objective of creating a historical dataset by *daily* querying as many records as possible.

To visualize how these optimizations inflict on the gathered data's completeness, Figure 1 shows the minimally necessary DNS queries to resolve the `A` record for `example.com` as seen by optimized measurements in comparison to the full tree that *may* be involved in DNS resolution for that name. While the optimizations of ZDNS and, to a degree, OpenINTEL reduce overhead and runtime, they miss major parts of the DNS tree and may hide corner cases or inconsistencies, potentially biasing results.

The biases induced by such partial tree exploration *may* be reasonable tradeoffs in comparison to an overall better runtime of measurements. Still, biases and trade-offs have not been systematically explored. Thus, in this paper, we answer: **What is the impact of partial tree exploration on DNS measurements?**

[3] Even though an iterative and 'all-nameservers' mode is available, its implementation is incomplete, see Section 3.4.

To capture the extent of the existing limitations, we start by discussing the methodological impact of exploring subsets of the DNS tree. Next, we discuss how to realize a DNS measurement system, `YoDNS`, that can potentially capture the full DNS tree. Among the important differences is that `YoDNS` gathers responses from all authoritative NSs of a DNS zone and chases all emerging paths even if they are inconsistent with prior responses. We use `YoDNS` to gather an extensive dataset using over 812M input names from various public and non-public sources, sending 85B queries over 40 days. Finally, we utilize the collected data to assess the impact of partial DNS tree exploration on four earlier studies, reproducing their results and quantifying biases induced by opportunistic DNS traversal. Our results show that some research questions, e.g., inconsistency between NSs, are prone to biases of limited tree exploration, while others are not.
Contributions: *All* active DNS measurement studies need to accept limitations in DNS tree exploration to remain feasible. A common limitation is opportunistic traversal, where only a subset of possible DNS resolution paths is explored. However, the impact of this choice has not been systematically assessed. We address this crucial methodological gap with our contributions:

– We present and publish `YoDNS`, an open-source framework for exhaustive DNS-tree exploration and measurements.
– We collect and share a large DNS dataset, containing full resolutions for over 812M names, spanning over 85B queries and 87 TB of compressed DNS data, enabling researchers to reproduce further studies based on our data.
– We evaluate the impact of limiting DNS tree exploration in four earlier studies and derive recommendations to balance exploration vs. efficiency for a given research question in order to not impact results, or–at least–to make the impact quantifiable.

2 Reliably Measuring DNS: A Primer

In this section, we first revisit how DNS in general and DNS resolution specifically, work. Subsequently, we review techniques for accelerating DNS resolution often used during DNS measurements and their impact on the results.

2.1 DNS Resolution Revisited

DNS was first introduced in RFC1032-RFC1035 [76, 51, 54, 53]. RFC8499 [36] gives a contemporary overview of DNS terminology and developments since then.
DNS Abstraction and Terminology: DNS is a tree, where each node ('label') is connected via dots as delimiters ('`.`') to the root. The root is an empty label behind a single dot '`.`', and a path in the DNS tree from a leaf to the root is a 'Fully Qualified Domain Name' (FQDN). 'Zones' are sub-trees, often operated by a different authority than their parents. Such 'delegations' take place at vertices referred to as 'zone cuts'. Nodes in the DNS tree can have values ('Resource Records', or 'RR') of a type ('RRtype') assigned to them, whereby multiple RRs of the same type form an 'RRset'.

DNS Implementation: Each zone, including the root, is hosted by at least one (ideally multiple [22]) 'name server' (NS) providing authoritative answers for the zone. These are 'authoritative NSs'. To delegate a zone, a parent contains an `NS` RRset for the child's name, listing the FQDNs of authoritative NS for the child zone. For authoritative NSs whose names are in or below the child, the parent zones' operator must add additional 'glue records', which explicitly list the NS' IP address(es). The child zone should contain corresponding RRsets.
DNS Resolution: DNS queries resolve a FQDN to an RRset given an RRType. To reply to a query, a *recursive* resolver needs to traverse the DNS tree to find the authoritative NS for a name's zone. A recursive resolver first checks the FQDN with the root servers or the label closest to the root, if QNAME minimization [10] is used. Since the root servers are typically not authoritative for the requested name, they respond with a referral, i.e., the NS records plus potential glue records of the NS authoritative below the next zone-cut in the FQDN. Next, the resolver repeats this process with one of the new NSs until an NS returns an authoritative answer instead of a referral.

2.2 DNS Measurement Trade-Offs

DNS measurements can be active or passive [68]. Passive measurements, e.g., rely on traces captured at authoritative NSs or DNS resolvers. Thus, they only capture those parts of the DNS tree that the NS is responsible for or that the resolver queries to answer user queries. In contrast, active DNS measurements, the focus of this paper, explore the DNS tree in a structured manner. Still, traversing the full DNS tree without any optimizations is practically impossible due to the size of the tree and abundance of parameters. Below, we summarize possible optimizations. See also Table 2 for how related-work handles these limitations.
Limiting exploration depth: Here, the measurement considers only zones higher up in the tree, e.g., by including only second level domains in the target list or stopping resolution at a certain depth. This may bias results towards zones which are likely to receive more scrutiny from delegating authorities.
Limiting exploration width: By limiting the number of input zones, or not chasing all out-of-zone records, one may bias the results towards specific TLDs, DNS operators, or popular sites. Similarly, one may restrict the number of followed paths, e.g., by considering only one resolution path (just like a resolver) rather than all possible paths, potentially hiding inconsistencies.
Vantage point selection: DNS responses differ by vantage point, e.g., due to load balancing [78] or anycast [1].
Caching: Using caches of previous DNS responses speeds up DNS lookups [29]. However, it can hide short-lived effects and inconsistencies between NSs. For example, when records are cached based solely on the tuple (`RRName`, `Type`, `Class`), as done by some resolvers, inconsistencies become invisible.
Using external recursive resolvers: Using professionally operated, well-provisioned resolvers, e.g., Google or PCH, can speed up resolution but hides many details, such as caching strategy, needed to analyze results. Moreover, anycasted resolvers may respond from different nodes having different (cache) states.
Dynamic resources: DNS trees of arbitrary depth [26], e.g., via `LUA` records [65],

and responses that are unique *per request* need to be pruned carefully.
Transport protocol: DNS supports UDP/TCP and requires handling ICMP/ ICMPv6. Missing any of these may lead to biases in observed NS reachability or loss of responses due to packet fragmentation [57].
Internet protocol: Not considering IPv6 reduces measurement overhead, but may bias results with respect to IPv6-only and dual-stack resolvability.
Relying on RFC compliance: Presuming RFC compliance can lead to unexplored parts of the DNS tree when encountering corner cases. For example, `CNAME`s should not be used in certain RRs (e.g. NS records [23]) or responses should be consistent, regardless of whether a FQDN or minimized query name is asked. However, Internet reality does not adhere to this. Thus, relying on RFCs compliance may limit the ability to study the effect of such misconfigurations.

2.3 Impact of Challenges on Query Load

DNS resolution in practice, i.e., using the optimizations above, requires significantly fewer queries than a full exploration of the tree, see Figure 1. But how many more queries are actually necessary? We can calculate a lower bound on the number of queries for the full resolution of `example.com` as follows:

- For all zones, ask all authoritative NSs for the zones' `NS` records.
- For all zones, ask all of its parents' authoritative NSs for referrals to the zone.
- For all NS names, ask all NS authoritative for them for the `A`/ `AAAA` records.

For example, the zone `icann-servers.net` has four authoritative NSs. Thus, we need to ask for `NS icann-servers.net` four times. We need to ask for a referral to `icann-servers.net` at all 13 nameservers of the `net`-zone. Finally, there are three NSs names in the zone, namely `{a,b,c}.icann-servers.net`. Asking the four authoritative NSs for the three `A` records requires 12 queries. Thus summing the queries for each zone from Figure 1, leaves us with 637 queries, over 100 times more than the minimal resolution shown in Figure 1.

We see, that full tree exploration entails more than simply identifying the answer to a specific query. However, certain measurements (IPv4 vs. IPv6, TCP *and* UDP, etc.) effectively double the number of queries again, see Table 1. Requesting DNSSEC or using small EDNS0 buffer sizes may cause truncated responses, so to accurately capture UDP-only behavior, queries have to be sent with/without DNSSEC requested, yet again doubling the number of queries. Adding more RRtypes adds queries linear to the number of discovered names and servers, similar to adding more vantage points.

This combinatorial explosion of queries inevitably forces measurements to accept *some* trade-offs. However, these trade-offs need to be chosen carefully and w.r.t. the research question. This work, for example, limits itself to using QNAME minimization, relies on a single vantage point and uses TCP only as a fallback mechanism. We discuss these limitations in detail in Section 3.

3 Related Work

Here, we discuss related work using a selection of publications representing the state of the art. We cluster them into three categories: *(i)* Work performing

Exploration	**#VPs**	**RR types**	**Transp.**	**#IPv4 Q.**	**#IPv6 Q.**	$\sum$
Minimal	1	`NS, A`	UDP	5	-	5
Full	1	`NS, A`	UDP	637	-	637
Full	1	`NS, AAAA`	UDP	-	637	637
Full	1	`NS, A, AAAA`	UDP	980	980	1,960
Full	1	`NS, A, AAAA`	UDP+TCP	1,960	1,960	3,920
Full	10	`NS, A, AAAA`	UDP+TCP	19,600	19,600	39,200

Table 1: Minimal queries needed for resolving `example.com` for selected measurement parameters, when only retrieving `NS`, `A`, `AAAA` records for all discovered names and involved zones, all using QNAME minimization. It is clear, that a truly exhaustive exploration of the parameter space is infeasible for a large number of domains. Note that the resolution paths for `example.com` are full dual-stack, i.e., IPv4 and IPv6 resolution require the same number of queries.

measurements *of* the DNS, *(ii)* Work performing measurements *with* the DNS, i.e., work where DNS is the *means* rather than the main subject being studied, *(iii)* Work describing DNS measurement methods and frameworks. Passive DNS measurements, e.g., work using traces collected at resolvers or NSs, are out of scope. We map challenges from Section 2 to related work in Table 2.

3.1 Measurements of DNS

With DNS being over 30 years old, there has been an abundance of DNS-related work. Initially, this work was industry-focused, e.g., Thompson et al. [81] noted the volume of DNS traffic on the Internet in a broader passive study. Darst & Ramanathan [17] discussed active DNS measurements to assess network performance in 1999, Huitema & Weerahandi [38] discussed the impact of DNS on active measurements in 2000. Last, in 2002 Liston et al. [49] conducted one of the first studies focusing on measuring DNS itself. Moreover, a study using data from 2003 by Pappas et al. [63] connects earlier work on DNS measurements to current times, measuring DNS misconfigurations and their impact on the DNS. Since 2002, there has been an explosion in DNS measurement work.

Focussing on transitive DNS dependencies, Ramasubramanian & Sirer [66] show that the number of NSs that *may* be involved in DNS resolution can be surprisingly large (>400). Further formalizing the dependency graph model, Deccio et al. [19] find that transitive dependencies can lead to additional lookups and false redundancy.

Other notable examples of DNS measurements include Nosyk et al. [62], who used RFC8914 [44] extended DNS errors for DNS measurements, Streibelt et al. [79], who measured IPv6 support in the DNS, and Fukuda et al. [30], who characterized DNS query response sizes. Similarly, Yajima et al. [89] measured the adoption of DNS security measures.

Akiwate et al. [3] measured 'Lame Delegations', taking a more general, yet IPv4-centric, approach to DNS. Furthermore, work by Rijswijk-Deij et al. [67] from 2014 assessed the potential of DNSSEC for DDoS attacks.

3.2 Measurements Using DNS

DNS is not a purpose unto itself but enables other services and applications. Hence, in addition to measurements *of* DNS, researchers regularly leverage active DNS measurements to infer information about other protocols and services.

For example, Gojmerac et al. [33] used active DNS measurements to assess the deployment state of email security mechanics. This research track is continuing, with recent work investigating specific–often new–email security mechanics like DMARC [6], SPF [16], and TLSA/DANE [47, 48].

Zirngibl et al. [91] used active measurements to study domain parking, finding that artifacts produced by it are often overlooked in measurement studies. Other use-cases for active DNS measurements are identifying IPv6 hosts [26, 9, 27], reassessing and probing targets [8] or asset discovery [85, 28].

3.3 DNS Measurement Frameworks

DNS measurement frameworks only became prevalent in the recent past. Early DNS measurements often used commodity utilities such as `dig` [50] which was possible as they were often only used for scanning top lists, e.g., the now discontinued Alexa list. However, with larger and more abundant domain sources, such as ICANN CZDS [39] ($\approx$220M names) and Certificate Transparency (CT) logs ($\approx$589M in our dataset), DNS measurement frameworks became necessary.

The first example of an elaborate DNS measurement framework is OpenINTEL [68], which has performed daily measurements of the DNS since 2015, claiming coverage of around 60% of the DNS. Due to the availability of the gathered historical data, OpenINTEL is also frequently used in other studies (e.g. [72, 73, 75, 84, 83, 82, 91]). Subsequently, in 2023, ZDNS was published as open-source software, providing a framework that allows researchers to perform their own active DNS measurement studies [41]. Both frameworks focus on resolvability, i.e., the red part of Figure 1a, rather than extensive exploration of the DNS tree. Furthermore, MassDNS [7], a high performance stub resolver, can query resolvers at scale. However, it does not support internal recursion, and thus, cannot easily be used for studies that rely on the resolution path. Both Streibelt et al. [79] and Naab et al. [59] report to have implemented their own frameworks and utilized them to measure 476k and 1M domains (note that our target list contains 812M). However, the tool of Streibelt et al. lacks scalability for Internet-scale studies (running 4 days for 476k zones [79]) and the tool of Naab et al. is not fully feature-ready, e.g. is missing features related to `CNAME` handling. As of Oct. 2024, neither tool is publicly available.

Finally, there is DNSViz [18], which was created for troubleshooting (DNSSEC-related) misconfigurations and is a well-known resource for DNS operators. DNSViz is able to query records from all NSs of a zone (and its parents) and, necessarily, resolves all *direct* dependencies of a zone. Contrary to `YoDNS`, transitive dependencies are learned through normal resolution. Given that DNSViz is not primarily designed for large scale measurement, features such as rate-limiting or a storage-efficient output format are also not natively integrated in the tool.

However, it does provide similar (though not the same) functionality to `YoDNS`, albeit for a different use-case.

Example Packet Traces: To further highlight the practical differences between `YoDNS`, OpenINTEL, DNSViz, ZDNS v1.0.0, and ZDNS v1.1.0, we included packet captures for a resolution of `example.com` with these tools in our published dataset [77].

When it comes to existing DNS measurement frameworks, the natural choice for a study like ours would have been OpenINTEL. However, OpenINTEL's focus is the efficient resolution of as many *different* names and RRsets as possible during a day to collect a historical dataset. For that, it leverages commodity Unbound resolvers. It does not attempt to find *all possible* resolution paths for a single name or RRset, making it not suitable for our objective.

With Streibelt et al. [79] and Naab et al. [59] not yet having published their frameworks, this only leaves ZDNS as a viable option. However, when evaluating ZDNS, we noted that the implementation of iterative resolution across all NSs is incomplete[4] and uses cached responses from one NS to synthesize responses for *other* NSs of the same zone. Furthermore, ZDNS does not use QNAME minimization, preventing the evaluation of zone-cuts. When evaluating ZDNS against common misconfigurations like parent-child NS mismatches[5], and RFC violations like the use of `CNAME`s in `NS` records, it did not provide reliable results. Finally, ZDNS does not track ICMP responses and lacks methods for effective rate-limiting when running in iterative mode due to its parallelization approach which relies on a large number of parallel sockets instead of asynchronous I/O.

As addressing these challenges in ZDNS would require significant architectural changes, we decided to use a clean-slate approach, see Section 4.

3.4 Reflections on Related Work

Earlier studies made methodological choices around the challenges we summarize in Section 2, see Table 2. Oftentimes, this includes relying on opportunistic DNS resolution, i.e. following a single resolution path instead of exploring all possible paths. In this section, we argue why this might influence results and how an open measurement framework for full DNS tree traversal can be helpful. However, we do *not* argue that all these studies should have used full tree traversal, rather that a systematic quantification of these biases is necessary.

Studies using OpenINTEL rely on opportunistic resolution. While this allows to take a longitudinal perspective, problematic resolution paths and NS inconsistencies may remain undiscovered and can bias results.

For example, van der Toorn et al. [84] identified private keys in `TXT` records as a security concern. However, exposing a private key has security implications, regardless of whether it was exposed via all or only a single authoritative NS. Yet,

[4] `https://github.com/zmap/zdns/issues/362`
[5] `https://github.com/zmap/zdns/issues/352`

	2009	2014	2015	2016	2017	2018	2019	2020	2021	2022			2023				2024	
	Pappas et al. [63]	Rijswijk-Deij et al. [67]	Gojmerac et al. [33]	OpenIntel & rel. [68][a]	Fiebig et al. [27]	Scheitle et al. [70][b]	Moura et al. [56]	Akiwate et al. [3]	Yajima et al. [89]	Fukuda et al. [30]	Zirngibl et al. [91][b]	Izhikevich et al. [41]	Streibelt et al. [79]	Naab et al. [59]	Ashiq et al. [6]	Nosyk et al. [62][d]	Zhang et al. [90]	This Paper
Full Tree Depth																		
Below SLD	✓	✗	✗	~	-	✓	~	✗	✗	✗	-	✓	✗	✗	✗	✓	~	✓
No Pruning	?	-	?	?	✗	?	?	?	?	?	?	?	✓	✗	?	?	?	✓
Full Tree Width																		
All NS	✓	✓	✗	✗	✗	✓	?	~	✓	✗	✗	~	✓	✓	?	-	✓	✓
Targets	✗	~	✗	~[c]	✓	✓	✗	✗	✗	✗	~	✓	✗	✗	✗	✓	~	✓
Res. Strategy																		
Contr. Recursion	✓	✓	?	✓	✓	?	-	✓	?	✗	?	✓	✓	✓	?	-	✓	✓
QMIN	-	✗	?	✗	?	?	?	?	?	?	?	✗	✓	✓	?	-	✗	✓
Traditional	✓	✓	?	✓	?	?	?	?	?	?	?	✓	✗	✗	?	-	✓	✗
Multi-Vantage Pts.	?	✗	?	~	✗	✓	✗	✗	?	?	✓	✗	✗	✗	?	?	✓	✗
No Over-Caching	✓	✓	?	✓	?	?	✓	✓	?	?	?	✗	✓	✓	?	-	✓	✓
Parallelization	?	✓	✓	✓	~	✓	?	?	?	?	?	✓	✓	✓	?	?	✓	✓
Enum. Dynamic R.	?	-	?	✗	✗	?	?	?	?	?	?	?	?	?	?	-	?	✓
CNAME Misconf.	?	~	?	✓	?	?	✗	?	?	?	?	?	✓	✗	?	-	✗	✓
TCP/ICMP	?	✗	?	~	?	?	?	?	?	✗	?	~	~	?	?	?	✗	~
IPv6	?	✓	?	✓	✓	✓	?	✗	✗	✗	✓	✗[e]	✓	✓	?	?	✗	✓

Legend: ✓: Addressed
~: Part. addr.
✗: Not addr.
-: Not rel.
?: No inform.

[a] Used in various studies, e.g.: [72, 73, 75, 84, 83, 82, 91]
[b] Also uses OpenIntel data.
[c] Claims ≈60% coverage, input is zonefiles/toplists/rDNS.
[d] Resolver study: some fields are not applicable.
[e] IPv6-support added after the paper was presented.

Table 2: Overview of related active DNS studies

the latter is easily missed by opportunistic traversal. Similar considerations apply to issues such as very large response sizes that allow for DDoS amplification [75] or stale glue records that might redirect clients to malicious IP addresses [72].

For a study of delegation inconsistency by Sommese et al. [73] using OpenINTEL, we explore the effects of such partial tree exploration in Section 6. We find that inconsistencies can amplify, especially for domains below second level.

A study from Akiwate et al. [3], investigating unresponsive NSs, limits probing to five authoritative NSs per domain, thus providing a lower bound on the actual unresponsiveness. Furthermore, they do not consider IPv6.

Studies that focus on public resolvers, such as that by Nosyk et al. [62], exploring extended DNS error codes [44] in resolver responses, might have used an exhaustive measurement framework to obtain ground-truth for certain misconfigurations and to verify results obtained from public resolvers even further.

In a recent study, Zhang et al. [90] measure the dependencies involved in DNS resolution using ZDNS [41], focusing on the implications for security and robustness, yet, they limit tree exploration by focussing on IPv4 and UDP. We discuss some of their findings in detail in Section 6.

Overall, we note that many prior studies are susceptible to biases introduced from partial tree exploration and NS inconsistencies. A flexible DNS measurement framework for full tree exploration would have allowed for better quantification of measurement errors, e.g., through supplemental measurements.

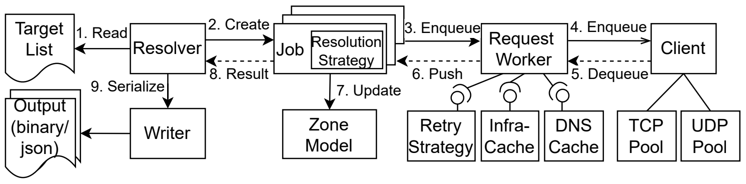


Fig. 2: `YoDNS` architecture.

4 `YoDNS` Design

Here, we present the architecture of `YoDNS`. It allows for more extensive exploration of the DNS tree than previous tools, and therefore, to better quantify effects induced by partial tree exploration.

`YoDNS` is configurable regarding DNS exploration depth and width, caching strategy, and rate limits. It remains robust to RFC incompliance and misconfigurations – following an 'accept liberal, send conservative' approach – and ensuring that all failures are recorded. Furthermore, our system supports standard quality-of-life features including monitoring integration and restart capabilities.

4.1 Architecture

To enable scale-out, `YoDNS` consists of multiple independent components, i.e., we decided *against* a monolithic architecture. See Figure 2 for an overview.

Overview We implement `YoDNS` in Go. Go's concurrency model maps independent units of work to separate virtual threads called Go-routines and communication is handled by passing messages via channels, allowing to easily scale components horizontally with more workers.

To execute a measurement, a `target list` with domain names to be scanned and a `configuration` specifying how to scan are supplied to the system. The `target list` is processed by the `resolver` which creates several independent, parallel `resolution jobs` based on the specification.

A `resolution job` entails resolving one or multiple names sharing a path towards the root. Jobs use asynchronous I/O to handle DNS requests and mitigate wait times from, e.g., transport delays. Jobs are parameterized with a `resolution strategy`, defaulting to QNAME minimization [10].

Queries are processed by the `RequestWorker`, which is responsible for cache-lookup, enqueueing wire queries, and retrying failed queries. Responses are stored in the infrastructure cache. Additionally, an `ICMP worker` (not shown) updates the cache when receiving ICMP `unreachable` messages.

Upon receiving DNS answers, the `resolution job` uses callbacks to the selected `resolution strategy`. During processing of the answer, the resolution strategy updates the `domain model` and enqueues new requests as necessary. The `domain model` also provides callback capabilities enabling us to 'go back in time' and enqueue more requests, e.g. when new NSs/IPs are discovered.

Queries are forwarded to the DNS client, which is based on Miekg DNS [32] but offers additional features including asynchronous I/O, rate limiting, UDP port reuse, and TCP connection reuse.

Termination is ensured by using a `been-there-map` to prevent duplicate queries from being asked (per NS address). Whenever a resolution job finishes, the results are written to offset-encoded protobuf messages and compressed. In addition, `YoDNS` provides JSON output for debugging purposes.

Extensibility and Versatility The configuration is used to adjust `YoDNS`'s base properties, including which queries to ask, rate limits, backoff-strategy and various other settings. Changes to the resolution strategy, e.g., not minimizing QNAMEs or not iterating over all NSs, are possible through a plugin interface. Modules allow intercepting the program flow whenever NSs, IPs, or zones are modified or responses are received. Data adapters can adjust data persistence, e.g., to use a database instead of files. This allows researchers to adapt `YoDNS` to their demands without having to reimplement a full DNS resolver.

Caching and Ordering The order of the target list greatly impacts performance and cacheability. If domain names which share a common suffix are scanned in temporal proximity on the same machine, more queries can be answered from the local cache. However, when too many related names are grouped together, this creates activity hotspots, putting strain on authoritative NSs.

To strike a balance between cacheability and randomization, we first group our input targets by their first non-public suffix, as determined using Mozillas Public Suffix list [58]. For our input (see Section 6), 99.71% of these groups have 20 or less domains, whereas the remaining 0.29% contain 27.78% of all domains–potentially causing activity hotspots for shared nameservers. Thus, we order domains in these groups by number of common labels, split them into subgroups of 20 and distribute them randomly within the target list. In test runs, this strategy resulted in a 280% speedup and reduced the actual queries by 50% compared to a randomized target list.

Given this pre-processing, `YoDNS` can use standard DNS caching, i.e., limiting the cache time to the TTL set by the authoritative servers, while being able to scale-out the measurements to multiple machines with dedicated caches.

To limit memory used by the cache, `YoDNS` supports a least-recently-used eviction policy. Lastly, while QNAME minimization may require more queries [88], it boosts cacheability, as more queries are common between FQDNs.

Performance To fully take advantage of parallelism we cannot afford to halt resolutions while waiting for a response, especially, given the per-IP rate limits, see Section 4.1. Instead, `YoDNS` advances resolution as far as possible and handles delayed responses and new resolution paths asynchronously. We use asynchronous UDP port reuse so each socket can have multiple outstanding queries, limiting each socket to 2^{16} active queries per remote IP (size of the ID field). However, to account for misconfigured servers responding from a different IP than the one queried, `YoDNS` does not reuse ID field values for outstanding queries for a cool-down period of 5 seconds. Under the rate limits of our measurement, our scans need less than 100 UDP ports per machine. `YoDNS` also pools TCP connections, supporting connection-reuse [20], and keep-alive [10], increasing performance and reducing the number of connections to remote servers [20].

Restart-Capabilities We combine target list state-keeping and the use of atomic groups to allow for graceful restarts. Even facing a sudden power loss, `YoDNS` can restart in-flight groups without having to re-resolve completed groups.

Rate Limiting While ethical measurements entail questions unique to each individual study, it is imperative that a general instrument supports measurement best practices necessary for ethical measurements, e.g., rate limiting of requests, opt-out [42], as well as curbing bandwidth requirements [46]. To realize rate limiting, `YoDNS` implements both, a query-per-second and a queries-in-flight limit per DNS server IP per measurement instance.

Both are needed. The first one limits how many queries have to be handled by well-provisioned servers. The latter limits how many queries have to be handled by low-resource or busy servers. Using only an in-flight limit may impose an undue burden on well-provisioned servers. For example, servers able to handle a single query in less than 10ms would receive up to 5,000 queries per second despite using a modest queries-in-flight limit of 50. Still, the in-flight limit avoids overloading slow servers that cannot keep up with the query-per-second rate.

In addition, `YoDNS` uses a time-slotted N-strike rule. If it receives no reply N times in a row the server is marked unreachable for 5 minutes. The same applies upon receiving an ICMP(v6) unreachable message. For opt-out, `YoDNS` implements a block list for IPs, NSs (by name), or queries for specific names.

RFC-compliant but resilient to incompliance To account for remote servers deviating from DNS best practices, `YoDNS` is resilient to various non-standard behaviors. Specifically, it gracefully re-attempts resolution when the remote closes TCP connections when pipelining is attempted. Similarly, `YoDNS` accepts `CNAME`s at apex and `NS` records pointing at `CNAME`s interpreting them based on the likely, yet non-standard, intention, as well as multiple `CNAME`s at the same name. Finally, it allows for out-of-zone glue and responses with invalid or incomplete flags, e.g., lacking the AA bit. Moreover, `YoDNS` always *records* such incompliance.

Monitoring For monitoring, `YoDNS` can be instrumented with a Prometheus [80] metrics endpoint. This allows for easy integration with off-the-shelf visualization and alerting solutions such as Grafana [34]. Current metrics include resolved domains per second, number of TCP connections, query response codes, rate limiting statistics, cache size, queries in-flight, RTTs, memory, and CPU usage.

5 Data Collection & Dataset

Here, we discuss the dataset we collected using `YoDNS`. For applicable ethical considerations, please see Appendix A.

5.1 Domain Name Input Datasets

As active DNS measurements need a list of target domains, we utilize the superset of all lists commonly used in the literature for `YoDNS`, Table 3. The corresponding lists have been retrieved on 5th December 2023. (1) `ct:` Names from unexpired certificates from Certificate Transparency logs: Argon, Xenon, Oak, Sectico Sabre, CloudFlare Nimbus, DigiCert Nessie, DigiCert Yeti, and TrustAsia. (2) `zf:` Zone files from ICANN's Centralized Zone Data Service (CZDS) [39] and available TLDs (.se, .nu, .ee, .ch, and .li). (3) `opendata:` Names from the open-data efforts of AFNIC [2] and SK-NIC [60]. (4) `tranco` [45], `majestic` [52], `radar` [15], `umbrella` [14]: Names from the corresponding domain top-list.

In total, we find almost 813M different names, of which 319M names are second level domains (SLD) and 494M are below. Of the 319M SLDs, 280M resolve in our scan[6], indicating a coverage of ∼78% of an estimated 360M total registrations for 2023 [21].

589M names are unique to CT logs, including 104M AWS-related domains and domains from many ccTLDs [74] not available via ICANN CZDS.

From zone files we gather another 217M (113M unique) names, including 23K names below the second level from the `.name` TLD (such as `john.doe.name`). The open data efforts contribute roughly 4.6M domains (2.5M unique).

By including names from the popular top list (Radar [15], Umbrella [14], Majestic [52], Tranco [45]) we add more than 4M domains. Yet, only the Cisco Umbrella list adds a notable set of new names. Among the top-level domains com with ≈56% is by far the largest contributor. It is followed by net (41M), org (23M), de (18M), io (13M), uk (12M) and ru (10M). Finally, we see that most names under a TLD are obtained through a single source only, for example, 377M out of 455M domains in `.com` are unique to a source.

5.2 `YoDNS` Configuration

Even though technically engineering-focused, documenting the exact configuration and design parameters for `YoDNS` is essential for reproducibility, and to avoid the presence of implicit limitations. Hence, we document our exact settings during data collection and applicable considerations here.

DNS tree exploration strategy: `YoDNS` is configured to query all encountered NSs, i.e., those listed in a parent zone as well as those in a zone apex (root level of that domain) and to follow `CNAME`s up to a depth of 64. It uses all discovered IPv4 and IPv6 addresses (encountered either as Glue or as authoritative response). This would cause significant load on the NSs for root and top-level

[6] Please note that these 319M SLDs include a full year of CT logs, i.e., also a high number of SLDs that have been unregistered in the meantime.

		#Domains	#Unique to source	#Below SLD
By Source	ct	696,487,135	589,186,623	492,898,606
	zf	217,438,044	112,862,815	23,341
	opendata	4,626,781	2,489,116	72
	tranco	1,000,000	18,203	0
	majestic	1,000,000	45,760	921
	radar	1,000,488	18,169	345
	umbrella	1,000,000	602,276	790,167
$\textbf{Sum}_{bySource}$		922,552,448	705,222,962	493,509,151
By TLD	com	454,938,301	377,277,850	276,016,574
	net	41,284,999	36,081,396	26,418,624
	org	22,798,581	17,671,810	11,033,936
	de	17,569,119	17,522,361	10,866,642
	io	12,770,554	12,750,067	11,456,784
	uk	11,503,887	11,468,806	7,062,585
	ru	9,767,399	9,694,778	7,323,877
	rest	242,113,140	222,755,894	143,330,129
$\textbf{Sum}_{byTLD}$		812,745,980	705,222,962	493,509,151

Table 3: Target list composition. #Domains for $\text{Sum}_{bySource}$ includes duplicates.

zones, e.g., com with its 13 authoritative NSs would receive almost 12B queries for the second-level domains of our input set. Thus, similar to Naab et al. [59], `YoDNS` only queries one of the authoritative NSs (selected at random) for root, root-servers.net, com, net, and org. Hereby, it includes the IPv4 as well as the IPv6 addresses of the chosen NSs. We assess the impact of this choice on the completeness of our results in Section 6.5.

Queried Records: `YoDNS` uses QNAME minimization by default and, thus, issues queries for all full and partial names of the target list as well as dependencies encountered during resolution. For names from the target list, `YoDNS` queries for `A`, `AAAA`, and `TXT` records of all of its private suffix parents as well as `www.` of the first private parent. At zone cuts, `YoDNS` queries for `DS`, `DNSKEY`, `CDS`, `CDNSKEY`, `CAA`, `TXT`, `MX`, `SOA`, plus the `TXT` records for the name `_dmarc.` in that zone. Also, every NS is asked for version.bind.

If the referral is bogus, i.e. the referred-to-zone is not the queried name or a parent thereof, we ask for the `SOA`, `NS`, `A`, `AAAA`, `TXT`, `SOA`, `DNSKEY`, `DS`, `MX` of the bogus name, but do not chase this path further. If a parent NS does not serve required glue, `YoDNS` asks it for the `A` and `AAAA` of the child NS directly.

Query Parameters: `YoDNS` announces a EDNS0 buffer size of 1232 and requests DNSSEC records (`DO=1`) in initial queries, but disables EDNS0 if the server responds with FormErr (`RCode=1`) or truncation (`TC=1`) even though TCP was used. Queries are retried up to 6 times with increasing back-offs and TCP being used at least once. The DNSSEC chain is not evaluated during the scan, but can be reconstructed from the results.

5.3 Measurement Platform

We ran the data collection from 4 virtual machines within a dedicated IPv4/IPv6 network segment. Each machine has 16 cores, 128 GB of RAM, and a 10 GBit/s network connection. Memory use averaged around 30GB per machine. CPU use averaged 13 fully utilized cores. Bandwidth utilization averaged around 15 MBit/s outbound (60MBit/s inbound), following our rate limits, see Section 4.1.

5.4 Scanning and Dataset

The data collection lasted for 40 days from 7th Dec 2023 09:13 UTC to 16th Jan 2024 11:40 2023. During the measurements, multiple events briefly impacted network connectivity. First, from 8th Jan 2024 15:30 UTC to 16:05 UTC and from 9th Jan 2024 17:50 UTC to 18:05 UTC one of the upstreams encountered connectivity problems due to anomalies in their peering relationships. Next, from 11th Jan 2024 18:10 UTC to 21:55 UTC one of the upstreams was impacted by a denial-of-service attack, which reduced throughput and caused timeouts. Finally, from 15th Jan 2024 00:06 UTC to 00:20 UTC and from 15th Jan 2024 03:00 UTC to 07:00 UTC, another one of the upstreams encountered a denial-of-service attack. All in-flight domains were remeasured after these events.

Our final dataset consists of 85B individual queries collected during these 40 days. Its compressed size on-disk is 87TB and spans over 812M input names.

5.5 Notable Events

During dataset collection, we encountered multiple domains with unexpected behavior which may have impacted `YoDNS` if we had sized it smaller. The first group of domains have an unusual parent-child inconsistency. While the parent lists four authoritative NSs, the zone apex contains 300 authoritative NSs with glue for IPv4 and IPv6. The latter implies sending queries to these 600 addresses (for 300 names and 5 record types). This leads to notable spikes in memory usage. Just the raw message bytes already need 18GB of memory. Another domain created an infinite, non-repeating chain of zone delegations. Here, `YoDNS` chases this sub-tree up to the maximum allowed length of 255 labels or until a 3M query limit per target list domain is reached. Finally, 3,384 domains in the Tranco top million list (3 in the top 10K) are linked to the socks5systemz malware. While this did not impact the measurement itself, we were contacted by the National CSIRT regarding a possible infection of the vantage point.

6 Evaluation

In this section, we evaluate the impact of limiting DNS tree exploration in various ways by revisiting four prior studies using our dataset. As the raw datasets from prior work are not available, we focus on comparing the published results and inferred metrics. The four studies we selected for our comparison are:

- **Delegation Inconsistency (OpenINTEL)** by Sommese et al. [73], as they only query a single NS per zone, thus limiting the explored DNS tree width.
- **A-Record Inconsistency (ZDNS)** by Izhikevich et al. [41], as the study identifies significant differences to prior work and requires querying all NSs.
- **Dependency Complexity (ZDNS)** by Zhang et al. [90], as the study depends on full dependency resolution but limits exploration to IPv4 and UDP.
- **DNS IPv6 Resolvability** by Streibelt et al. [79], as they run comparable active measurements, but limit their (active) tree exploration to 1M domains. Most of their analysis is based on a passive approach which does not provide full visibility into the DNS tree.

6.1 Delegation Inconsistency

Study Description: To delegate a zone, two conditions must be met: (i) the parent zone must serve NS records for the child, and, (ii) the child must authoritatively serve the same NS records from the zone apex. A delegation inconsistency is, when different record sets are served from zone apex and parent NSs. This may result in unresponsiveness, longer resolution times or security risks [3].

Sommese et al. [73] study the prevalence of delegation inconsistencies below three large TLDs, finding up to 8.2% of inconsistent delegations below `net`.

The study acknowledges that inconsistencies in authoritative NSs can impact the observed delegation inconsistency when only a single NS is queried. Moreover, inconsistencies between parent NSs can affect results, too.

Original Dataset: The original study uses OpenINTEL [68]. Since OpenINTEL only queries a single authoritative NS for the parent and the child zone, it is susceptible to biases induced by partial tree exploration. Thus, the study offers a lower bound on the actual prevalence of delegation inconsistency.

Reproduction Setup: We reproduce the original study with our dataset, recall Section 5, which considers all possible resolution paths. Furthermore, our dataset covers roughly 3 times more domains than used in the original study, and features domains below the second level. Note, that our data is from late 2023/early 2024, while the original data is from 2020.

Result Comparison: Table 4 summarizes the results. To quantify biases that may occur due to partial tree exploration, we simulate three perspectives using our comprehensive data: A 'best case', where we always select the consistent resolution path (if such a path exists), a 'worst case' where we always select the path with inconsistency, and a 'randomized' one, simulating the behavior of a normal DNS resolver—similar to the OpenINTEL approach.

Like Sommese et al., we categorize domains as unresponsive, if no authoritative answer can be obtained from the nameservers of the delegated zone. We classify retrieved NS sets as either consistent ($P{=}C$) or inconsistent ($P{\neq}C$). Here, P denotes the set of `NS` records served from a NS of the parent zone and C denotes the `NS` records served from a child NS. Assessing the set intersections of '$P{\neq}C$' we find the majority of inconsistent NS sets to be disjoint, with around 1.61% of all zones having an NS set at the child that is a superset of the one from the parent. 0.73% have a parent set that is a superset of the child's NS set. Our results roughly match those of Sommese et al. for `.com`, `.org`, and `.net`. As expected, the random strategy is between best- and worst-case.

Looking beyond `.com`, `.org`, and `.net`, we find inconsistencies between two NSs of the same level (serving the parent or the child) differ considerably based on the type of zone investigated. For zones below second level, we find far fewer inconsistencies between parent and child 2.47%, likely because parent and child are often operated by the same organization. However, the inconsistency among children and parents is larger (5.99% for $C{-}C$ and 7.67% for $P{-}P$). Due to this, a single delegation path may see *up to 9 times fewer inconsistent NS sets* than our approach (0.28% in the best-case vs 2.47% in the worst-case). Similarly, we

	$\sum$		Unresp.	$P = C$	$P \neq C$	Disjoint	$P \supset C$	$P \subset C$	Rest	C-C incon	P-P incon
.com		Best	10.38	83.67	5.95	3.72	0.41	1.60	0.22		
	156M	Wst.	10.43	80.44	9.14	6.44	0.69	1.72	0.28	1.04	-
		Rnd.	9.35	82.61	8.04	5.52	0.58	1.69	0.25		
	142M	[73]	14.0	78.0	8.0	4.64	0.48	2.47	0.41	-	-
.org		Best	8.97	85.23	5.80	3.50	0.38	1.63	0.29		
	11M	Wst.	9.70	83.42	6.88	4.16	0.59	1.74	0.39	0.30	-
		Rnd.	8.96	84.61	6.43	3.89	0.50	1.70	0.34		
	10M	[73]	9.5	82.9	7.6	4.20	0.64	2.37	0.39	-[a]	-
.net		Best	11.84	82.46	5.52	3.18	0.38	1.73	0.22		
	13M	Wst.	11.93	80.09	7.97	5.10	0.68	1.89	0.30	0.74	-
		Rnd.	11.09	81.79	7.11	4.40	0.60	1.84	0.27		
	13M	[73]	12.6	79.2	8.2	4.18	0.75	2.82	0.45	-	-
.top		Best	16.76	76.17	7.06	3.38	0.13	3.54	0.01		
	3M	Wst.	15.89	70.74	13.37	9.27	0.35	3.66	0.09	0.35	0.55
		Rnd.	13.95	74.75	11.27	7.32	0.30	3.63	0.05		
>SLD		Best	2.03	97.68	0.28	0.18	0.01	0.09	0.00		
	37M	Wst.	10.23	87.29	2.47	1.84	0.28	0.21	0.15	5.99[b]	7.67
		Rnd.	6.17	92.58	1.24	0.84	0.16	0.16	0.09		
All		Best	8.14	86.75	5.11	3.05	0.38	1.49	0.18		
	316M	Wst.	9.72	82.76	7.52	4.91	0.73	1.61	0.27	1.70	0.96
		Rnd.	8.13	85.24	6.63	4.23	0.60	1.57	0.23		

[a] Though not directly C-C inconsistency, [73] report ∼2% of P-C inconsistent cases in `.org`, *also* have C-C inconsistency on a sample of 10k domains.
[b] While a higher C-C than P-C inconsistency may seem odd, it is an artifact of DNS and *only* visible when doing a full tree traversal, see Appendix B.

Table 4: Delegation Inconsistency. C denotes the set of all `NS` records served from the zone apex, whereas P is the `NS` records as served by the zones' parents.

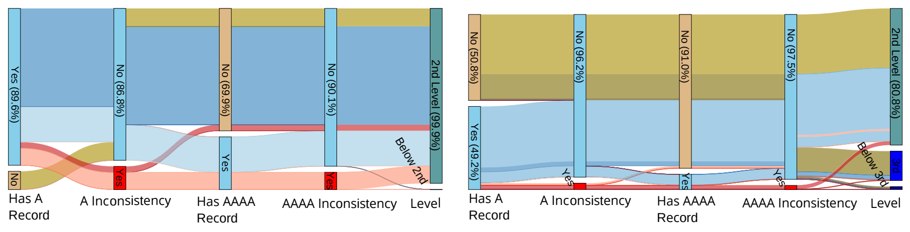
find more inconsistencies for some TLDs, e.g., `.top` shows a $P{-}C$ inconsistency of 13.37%, of which 52% are due to a recurring combination of two DNS hosters.
Conclusion: We can reproduce the results of Sommese et al., especially given their goal of providing a lower bound for the number of inconsistent delegations.

However, due to our extensive exploration of the DNS tree and larger input set, we find zones below second level and TLDs exhibiting different behaviors. Especially below second level, $C{-}C$ and $P{-}P$ inconsistencies, which are not captured by OpenINTEL, have comparable impact to $P{-}C$ inconsistencies. Hence, we argue that measurements of domains below second level, should consider full tree exploration to avoid biases induced by NS inconsistencies.

6.2 A-Record Inconsistency

Study Description: For a given RRset, all authoritative NS should always return the entire set of resource records [23], and be in sync, i.e., provide the same data [87]. Returning inconsistent records can lead to seemingly random problems that only affect a subset of clients. For example, out-of-sync `A`/`AAAA` records may direct traffic to unresponsive or, in the worst case, malicious IPs [8].

Izhikevich et al. [41] studied `A` record inconsistency as a case study for ZDNS. The study focusses on IPv4 only. By comparing our results, we can also compare the different DNS exploration strategies of ZDNS and `YoDNS`.
Original Dataset: The dataset used by Izhikevich et al. was gathered using ZDNS with names from CT logs. It used an input set of 234M FQDNs in 93M base domains. ZDNS successfully resolves 70% of these names. The paper notes a 99.99% consistency for authoritative `A` RRsets over its input dataset.



(a) Tranco Top 1M. (b) All names incl. dependencies (1072M).

Fig. 3: Inconsistencies in A and AAAA records. Each flow (by color) corresponds to a unique combination of parameters. Its height corresponds to its share of unique domains. For example, the large light-blue flow in Figure 3a contains domain which have an `A` Record, show no inconsistency, and are lacking an `AAAA` record. Flows which exhibit `A`/`AAAA` record inconsistency are colored red and flows which have no `A` record are colored yellow.

Reproduction Setup: We mirror Izhikevich et al.'s analysis on our data and the Tranco subset, i.e., comparing the returned `A` record sets from all authoritative NSs of a name, extending the analysis to `AAAA` records. In total, we evaluate 1072M names, including names from our target list and resolved dependencies.

We present a Sankey plot of the following parameters in Figure 3:

– **Has A/AAAA:** domains with an `A`/`AAAA` record.
– **A/AAAA-Inc:** with a mismatch for `A`/`AAAA` between NSs.
– **Level:** Level 2 for PSL private suffixes, Level 3 below.

Result Comparison: We find significantly higher inconsistencies for `A` and `AAAA` records than previous work. Recall that they found 99.99% consistency. For the Tranco list, 89.6% have an `A` record and 13.2% have an `A` record inconsistency across NSs. Notably, most of these records *do* also have an `AAAA` record, which *also* shows inconsistency across different NSs.

Looking at all names, we find 49.2% have an `A` record and 3.8% have an `A` record inconsistency across NSs. Again, we find a notable stream of `A` record inconsistencies for names that also have an `AAAA` record inconsistency. Due to our target list, a large number of third-level domains (dark yellow flow) lack `A` and `AAAA` records. This is due to the AWS domains (14.9% of all names) in our target list, which often have no associated records.

Next, we determine the DNS hosters for names with inconsistent `A` records. For this, we inspect the name of the authoritative NSs which might result in Akamai being underrepresented as they often use in-domain NS. Figure 4 shows that a large portion of domains with inconsistency is hosted by Cloudflare, likely due to CloudFlare's dynamic IP assignment at query time [24]. In this case however, adverse effects are unlikely as services are reachable on all served IPs.

Conclusion: In conclusion, we find notably more `A` record inconsistencies between NSs than the 0.01% reported by Izhikevich et al. when using ZDNS [41]. With a restricted dataset, which closely matches the one used by Izhikevich et al.–including only records served from IPv4 addresses and names from CT logs–we find that 50.2% have an `A` record, of which 5.15% have inconsistent `A`

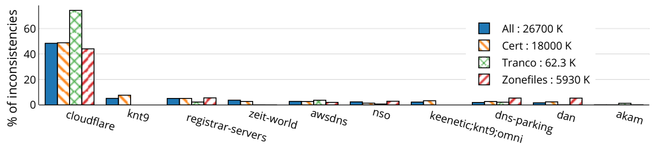


Fig. 4: Hosters of domains with inconsistent `A` records.

records. Given our observations for CloudFlare, it is unlikely that a difference in the used domain samples cause this disparity. A review of the source code of ZDNS indicates that over-caching in `all-nameserver` mode, i.e., reuse of prior responses from *other* authoritative NS, may be responsible. The resulting limited tree exploration, may have introduced a bias towards consistency.

In summary, we are unable to reproduce the results of Izhikevich et al. [41]. Their conclusion that `A` record inconsistencies are rare due to the ongoing centralization of the Internet cannot be supported by our dataset.

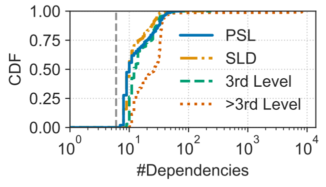
6.3 Dependency Complexity

Study Description: In order to resolve a zone, resolvers need to resolve the zones' parent zones as well as the zones of their nameservers.

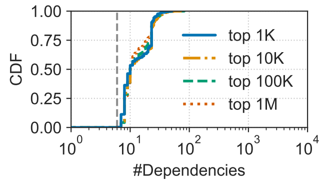
However, a large number of transitive dependencies can negatively affect resolution times, make zones appear more redundant than they actually are [19], or even pose security risks, as compromised domains can affect the resolution of dependent domains [66, 86]. A recent study by Zhang et al. [90] measures effects of dependencies on robustness and security of DNS resolution for 217M domains.
Original Dataset: The dataset from Zhang et al. [90] consists of zones obtained via ICANN CZDS and the Tranco and Umbrella top lists. The measurement is conducted using ZDNS on 8 distributed vantage points. For their study, Zhang et al. implemented a custom caching strategy for ZDNS, ensuring requests are cached per NS IP. However, they conduct their study using IPv4 and UDP only.
Reproduction Setup: Our dataset contains full-dependency resolutions for each encountered domain, enabling us to quantify the number of dependencies. In addition to the original study, our measurement features IPv6 and a TCP fallback mechanism. Furthermore, our target list contains a considerable number of additional domains from certificate transparency logs and open data efforts.

Like Zhang et al., we define the set of dependencies of a zone as all zones that are encountered during the resolution. This transitively includes the zone itself, its parents, and the zones of its nameservers.
Result Comparison: We present the number of zone dependencies as CDFs in Figure 5. Contrary to Zhang et al., we consider `root-servers.net` to be a dependency of the root zone, which is why we observe a minimum of 6 dependencies (`root`, `root-servers.net`, `gtld-servers.net`, `nstld.com`, `net` and `com`) for all zones. This minimum is marked by the vertical line. The first notable increase in zones is at 7 dependencies for in-domain hosted SLD zones.

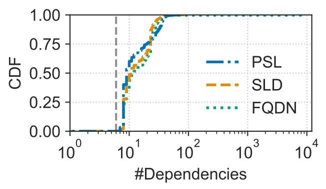
Like Zhang et al., we find that popular Tranco zones exhibit slightly more dependencies. For Umbrella, we find only slightly more dependencies when look-



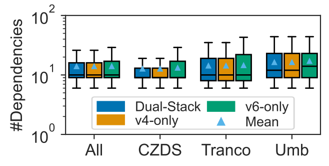
(a) All by PSL-depth



(b) Tranco by popularity



(c) Umbrella by PSL-depth



(d) By source and IP version

Fig. 5: Zone Resolution Dependencies

ing at the fully qualified domain names as compared to the SLDs in the list. However, we observe a long tail of zones with over 9,000 dependencies. Upon closer inspection, this tail is caused by related domains (sharing the same SLD), generating endless delegation chains. However, these domains were not present in the version of Umbrella used by Zhang et al. (25th of April, 2024). In total, we find 150K out of 316M (0.05%) zones with more than 100 dependencies. Again, all of these appear misconfigured to automatically generate delegations.

To quantify biases induced by scanning via IPv4 only, we consider the dependencies of *resolvable* zones in Figure 5d. IPv4-only and dual-stack resolution show almost identical numbers of (resolvable) dependencies. For IPv6-only, we see slightly more dependencies, indicating that those zones that *are* IPv6-only resolvable, have slightly more dependencies than those that are not. However, less zones overall are IPv6-only resolvable, see Section 6.4.

Conclusion: Our results match those of Zhang et al. qualitatively, although we count more dependencies due to different handling of the `root-servers.net` dependency. Additionally, dependencies in the original study may be missed due to the lack of TCP support. However, we show that the number of zone dependencies is only marginally biased by the lack of IPv6 in the measurement.

6.4 DNS IPv6 Resolvability

Study Description: The complexity of DNS results in a plethora of potential misconfigurations that can impair resolvability and redundancy of a zone. In case of IPv6, this is especially difficult, as redundancy and fallback mechanisms can make zones *appear* to be resolving even though misconfigurations do not allow resolution in an IPv6-only scenario. This was first studied by Streibelt et al. [79], using passively collected traces and a small scale active measurement.

In their study, Streibelt et al. find that 55.1% of zones are IPv6 resolvable as of August 2022 and that IPv6 adoption steadily increased since 2015. Furthermore, they note that zones deeper in the DNS tree are less likely to be IPv6

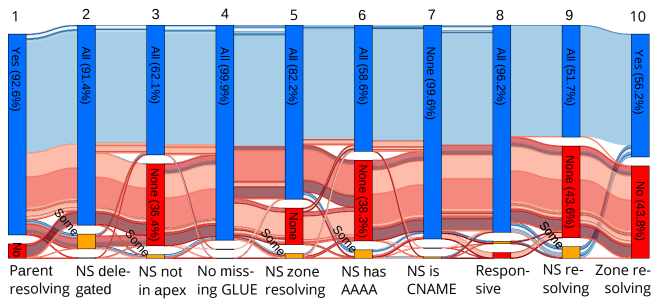


Fig. 6: IPv6-resolvability problems (316M zones). Each flow (by color) corresponds to a unique value combination of parameters. Its height corresponds to its share of unique domains. IPv6-only-resolving domains are colored in shades of blue, while non-IPv6-resolving domains are colored in shades of red. At each column, blue indicates that all NSs of that domain are ok and red (yellow) indicates that all (some) NSs exhibit a specific problem.

resolvable and that individual operators can have significant impact on global IPv6 resolvability if they host a large number of zones. They find that IPv6 resolution problems are often due to NSs in the parent zone not resolving, i.e., because the parent zone or the zone of the NSs does not resolve via IPv6.

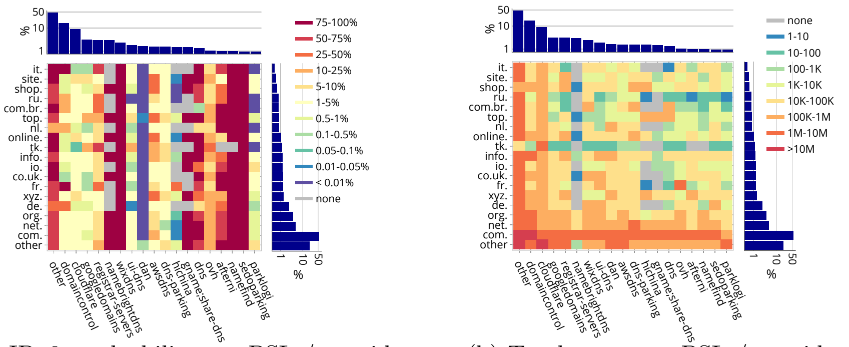
Using our active approach, we can re-evaluate and extend their study by providing reasons for unresolvability from the perspective of a resolver.

Original Dataset: The original study relies on passively collected traces that are collected on globally distributed DNS resolvers. The dataset contains only the recorded and aggregated cache misses of the resolvers. Naturally, this might limit visibility of the DNS tree, as records that are never requested cannot appear. The passive approach is verified using a small scale active measurement.

Reproduction Setup: Given that the work by Streibelt et al. is based on passive DNS data, we leverage data from all zones we measured in our study. However, we extend the categories of unresolvability reasons. The results are shown as Sankey plot, in Figure 6.

Specifically, we say a zone is resolvable (column 10 in Figure 6) if its parent zone is resolvable (column 1) and if at least one of its NSs is resolvable (col. 9) and authoritatively responds (col. 8), i.e., has none of these issues:

– **Missing Delegation (col. 2):** If an NS is not in the delegation for the zone, i.e., only returned from the apex, the NS cannot be used to resolve the zone.
– **NS record not in zone (col. 3):** If an NS is delegated but not returned from the zone apex. Some resolvers require this to harden glue, e.g. Unbound [61].
– **No AAAA glue (col. 4):** As with IPv4, in-domain NSs need glue [4].
– **Unresolvable NS zone (col. 5):** If an out-of-domain NS's name is in an unresolvable zone the NS cannot be used.
– **No AAAA records for NS (col. 6):** An NS needs a resolvable authoritative `AAAA` record, as some resolvers are hardening glue, e.g., Unbound [61].
– **NS CNAME (col. 7):** RFC2181 [23] prohibits `CNAME`s as NS names. Thus, resolvers often do not support this [40].



(a) IPv6-resolvability per PSL / provider (b) Total zones per PSL / provider

Fig. 7: Connection between public suffixes, DNS hosters, and IPv6-resolvability.

Result Comparison: Overall, we find 56.2% of 316M zones are IPv6-only resolvable, which is well-aligned with the 55.1% from Streibelt et al. [79]. 8.6% of zones have non-delegated NSs, i.e., NSs not listed in the parent, which only prevents resolution in 0.16% of cases, e.g., when the NSs only listed in the child are IPv6 resolvable, while the ones in the parent are not. Only a fraction of 0.005% of zones is unresolvable due to missing glue. However, like Streibelt et al., we find non-resolving NS zones to be a major problem for resolvability, with 16% of zones being unresolvable and 1.8% of zones being partially affected by this. Furthermore, `CNAME`s in `NS` records are not frequent, with only 0.01% of domains being unresolvable (0.4% affected) due to `CNAME`s being entered as NS names. Finally, 2.7/1.2% of domains are unresolvable/affected because the listed NSs are not (authoritatively) replying. In summary, we find 51.7% of domains have only correctly configured NSs, whereas 4.8% have some NSs with problems.

Next, we revisit the correlation between public suffixes, DNS hosters, and IPv6 resolvability. Figure 7 shows heatmaps of (a) percentage not IPv6-resolving zones (b) the number of zones for the top 18 public suffixes vs. the top 18 largest hosters. Note, we may again underestimate Akamai-hosted domains. The impact of individual hosters on IPv6 resolvability is clearly visible in Figure 7a, with, e.g., WixDNS (wix.com, a SaaS web hoster) being completely IPv6 unresolvable. Interestingly, for DomainControl (GoDaddy), IPv6 resolvability is overall good, except for some European zones, most notably `.fr` and `.it`, where essentially all zones are not IPv6-resolvable. Investigating further, we find that GoDaddy uses different NSs for these two TLDs, and, while they added IPv6 glue for most of their authoritative servers in the past, this was not done for these NSs, leaving these zones still not IPv6 resolvable. Similarly, we see the positive impact of CloudFlare on IPv6 resolvability, hosting a major portion of domains.

Conclusion: Overall, we can reproduce the results of Streibelt et al., while we also find additional corner cases missed by their analysis. The relatively small number of zones with unresponsive NSs (3.8%) indicates that–especially for long-term perspectives–the use of passive DNS data is a viable option for IPv6 resolvability assessments, despite not having full tree visibility. However, such studies should reconstruct the resolution path, to distinguish failing IPv6 resolution due to parent zones and zones hosting out-of-zone NS, not IPv6 resolving. Overall, active measurements allow a more fine-grained classification of the reasons for lacking IPv6 resolution, see Figure 6. Finally, we recommend contacting major hosters contributing large portions of non-IPv6 resolving zones directly to rectify these. We reached out to GoDaddy, but a conclusive reply is still pending.

6.5 Result Sensitivity

To reduce the load on Internet core infrastructure, we configured `YoDNS` to only query a single authoritative NS via IPv4 and IPv6 for the `root`, `root-servers.net`, `com`, `net`, and `org` zones, assuming consistency between NSs.

To estimate the impact of this choice on our results, we conduct a measurement on a random sample of 1M zones from `com`, `net`, and `org` each on March, 27th 2024, querying all the NSs of these zones.

We observe no cases where the authoritative nameservers of trusted zones disagree. To further ensure no results have been influenced, we run all our analyses twice. For the first run, we analyze the data considering all the responses from the TLD NSs. For the second run, we only consider the responses of a single TLD NS, emulating the behavior of our large-scale measurement. For the delegation inconsistency analysis that involves random sampling, results deviate only by 0.02%. For all other analyses, the results remain unchanged.
Conclusion: We cannot discard the possibility that the resolution of individual domains would have differed, had we queried all NSs of our five trusted zones. However, an analysis of 3M domains shows no such incident. We conclude that not querying all NSs for selected zones had no notable influence on our results.

7 Concluding Discussion

In this paper, we present `YoDNS`, a DNS measurement framework for full zone exploration. We employ this framework to replicate four studies, finding that incomplete explorations of the DNS tree can impact results.

For a study on delegation inconsistencies, by Sommese et al. [73], we show that the inconsistency measured by full exploration can be up to 9 times higher than with opportunistic traversal for domains below second level (0.28% in the best-case vs. 2.47% in the worst-case). Similarly, for a study on IPv6-readiness by Streibelt et al. [79], our approach provides richer data and improves classification compared to their passive measurements. It is on par with their active measurements, yet using a significantly larger sample.

However, for both studies, our results also highlight the trade-offs, and–most importantly–that the selected data sources of these two studies *are* reasonable choices for their research questions. Even though our results indicate the need for full tree traversal for zones *below* second level, for Sommese et al.'s selection of second level zones, a full-tree traversal only marginally improves accuracy. Similarly, passive data enabled Streibelt et al. to take an eight-year perspective, which is impossible with full-tree data. Even though having limited visibility into the tree, their numbers are well-aligned with ours.

For other studies, our full tree exploration shows differences in results. For a study Zhang et al. [90], we find more dependencies. However, we also show, that the number of zone dependencies is only marginally influenced by the lack of IPv6. We cannot reproduce a study of `A` record inconsistencies by Izhikevich et al. [41], finding over two orders of magnitude more inconsistencies.

Recommendations: Our evaluation shows that the DNS tree is not homogeneous, while DNS is notoriously complex. Subsets, e.g., specific TLDs, limited zone depth, or resolution path exploration may hide or amplify effects, making measurements challenging. Hence, we have the following recommendations:

- **Consider full-tree traversal for zones below second level**: We have seen that inconsistencies can have severe effects deeper in the tree. Therefore, we argue that measurements of zones below second level, should strongly consider full tree exploration to avoid or quantify the induced uncertainty.
- **Find qualitative explanations and cross-check results:** In our evaluation, studies that cross-checked results with supplemental measurements have shown to be less prone to biases of partial tree exploration. When identifying differences from prior work, it is important to find qualitative explanations and verify conclusions, for example, by using longitudinal data and/or identifying potential root causes.
- **Accept and Document Limitations:** Given the complexity of the DNS, it is infeasible for a single study to cover all (possible) aspects of DNS that *may* influence results. For example, full tree traversal uses significant time and resources, while mitigating some, but not all, limitations. Hence, researchers should carefully assess if their research question is amendable to optimizations (see Section 2) and transparently document these optimizations.

Limitations: While exploring the full DNS tree, the `YoDNS` configuration used in our measurements only uses (a) TCP fallbacks instead of always forcing TCP *and* UDP queries, (b) one vantage point, (c) QNAME minimization, and (d) naturally does not know *all* possible zones. We decided to accept these limitations since the gathered dataset is sufficient to reach our objective of evaluating whether incomplete DNS tree exploration can bias results. However, `YoDNS` could easily be run from multiple vantage points given available measurement systems.

Artifact Availability: Our measurement instrument is available at `https://github.com/DNS-MSMT-INET/yodns`. The collected data [77] is available at `https://doi.org/10.17617/3.UBPZXP`.

References

1. Abley, J. *et al.*: Operation of Anycast Services. RFC 4786,
2. afnic, Données partagées : l'open-data du .fr, (2023). `https://www.afnic.fr/produits-services/services-associes/donnees-partagees/`.
3. Akiwate, G. *et al.*: Unresolved Issues: Prevalence, Persistence, and Perils of Lame Delegations. In: IMC (2020)
4. Andrews, M. *et al.*: DNS Glue Requirements in Referral Responses. RFC 9471,
5. Arends, R. *et al.*: DNS Security Introduction and Requirements. RFC 4033,
6. Ashiq, M.I. *et al.*: You've Got Report: Measurement and Security Implications of DMARC Reporting. In: USENIXSEC (2023)
7. Blechschmidt, B.: MassDNS, (2024). `http://github.com/blechschmidt/massdns`.
8. Borgolte, K. *et al.*: Cloud Strife: Mitigating the Security Risks of Domain-Validated Certificates. In: ANRW (2018)
9. Borgolte, K. *et al.*: Enumerating active IPv6 hosts for large-scale security scans via DNSSEC-signed reverse zones. In: SP (2018)
10. Bortzmeyer, S.: DNS Query Name Minimisation to Improve Privacy. RFC 7816,
11. Brownlee, N. *et al.*: DNS measurements at a Root server. In: GLOBECOM (2001)
12. Callejo, P. *et al.*: Measuring the Global Recursive DNS Infrastructure: A View From the Edge. IEEE Access (2019)
13. Chung, T. *et al.*: Understanding the Role of Registrars in DNSSEC Deployment. In: IMC (2017)
14. Cisco, Umbrella List, (2023). `https://s3-us-west-1.amazonaws.com/umbrella-static/index.html`.
15. CloudFlare, Cloudflare Radar List, (2023). `https://radar.cloudflare.com/`.
16. Czybik, S. *et al.*: Lazy Gatekeepers: A Large-Scale Study on SPF Configuration in the Wild. In: IMC (2023)
17. Darst, C. *et al.*: Measurement and Management of Internet Services. In: IM (1999)
18. Deccio, C.: DNSViz, `https://dnsviz.net/`.
19. Deccio, C. *et al.*: Measuring Availability in the Domain Name System. In: INFOCOM (2010)
20. Dickinson, J. *et al.*: DNS Transport over TCP - Implementation Requirements. RFC 7766,
21. DNIB Quarterly Report Q4 2023, `https://dnib.com/articles/the-domain-name-industry-brief-q4-2023`.
22. Durand, A. *et al.*: DNS IPv6 Transport Operational Guidelines. RFC 3901,
23. Elz, R. *et al.*: Clarifications to the DNS Specification. RFC 2181,
24. Fayed, M. *et al.*: The ties that un-Bind: Decoupling IP from web services and sockets for robust addressing agility at CDN-scale. In: SIGCOMM (2021)
25. Fiebig, T.: Crisis, Ethics, Reliability & a measurement.network. In: ANRW (2023)
26. Fiebig, T. *et al.*: Something from Nothing (There): Collecting Global IPv6 Datasets from DNS. In: PAM (2017)
27. Fiebig, T. *et al.*: In rDNS We Trust: Revisiting a Common Data-Source's Reliability. In: PAM (2018)
28. Fiebig, T. *et al.*: Heads in the Clouds? Measuring Universities' Migration to Public Clouds: Implications for Privacy & Academic Freedom. In: PETS (2023)
29. Fujiwara, K. *et al.*: Aggressive Use of DNSSEC-Validated Cache. RFC 8198,
30. Fukuda, K. *et al.*: Characterizing DNS query response sizes through active and passive measurements. In: NOMS (2022)

31. Gao, H. *et al.*: Reexamining DNS From a Global Recursive Resolver Perspective. TON (2014)
32. Gieben, M.: MiekgDNS Git, (2023). `https://github.com/miekg/dns`.
33. Gojmerac, I. *et al.*: Large-Scale Active Measurements of DNS Entries Related to E-Mail System Security. In: ICC (2015)
34. Grafana Labs, Grafana, (2024). `https://grafana.com/`.
35. Harrenstien, K. *et al.*: DoD Internet host table specification. RFC 952,
36. Hoffman, P. *et al.*: DNS Terminology. RFC 8499,
37. Hoffman, P. *et al.*: The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA. RFC 6698,
38. Huitema, C. *et al.*: Internet Measurements: the Rising Tide and the DNS Snag. In: ITC Specialist Seminar on Internet Traffic Measurement and Modelling (2000)
39. ICANN, ICANN CZDS, (2023). `https://czds.icann.org/home`.
40. Internet Systems Consortium, Can an NS record refer to a CNAME?, `https://kb.isc.org/docs/aa-00203`.
41. Izhikevich, L. *et al.*: ZDNS: a Fast DNS Toolkit for Internet Measurement. In: IMC (2022)
42. Kenneally, E. *et al.*: The Menlo Report: Ethical Principles Guiding Information and Communication Technology Research. SSRN Electronic Journal (2012)
43. Kosek, M. *et al.*: Measuring DNS over TCP in the Era of Increasing DNS Response Sizes: A View from the Edge. ACM SIGCOMM Computer Communication Review (2022)
44. Kumari, W. *et al.*: Extended DNS Errors. RFC 8914,
45. Le Pochat, V. *et al.*: Tranco List, (2023). `https://tranco-list.eu/`.
46. Learmonth, I.R. *et al.*: RFC Guidelines for Performing Safe Measurement on the Internet. Tech. rep., (2023). `https://datatracker.ietf.org/doc/draft-irtf-pearg-safe-internet-measurement`
47. Lee, H. *et al.*: A Longitudinal and Comprehensive Study of the DANE Ecosystem in Email. In: USENIXSEC (2020)
48. Lee, H. *et al.*: Under the Hood of DANE Mismanagement in SMTP. In: USENIXSEC (2022)
49. Liston, R. *et al.*: Diversity in DNS Performance Measures. In: SIGCOMM Workshop on Internet Measurement (2002)
50. Liu, C. *et al.*: DNS and Bind. " O'Reilly Media, Inc." (2006)
51. Lottor, M.: Domain Administrators Operations Guide. RFC 1033,
52. Majestic, Majestic List, (2023). `https://majestic.com/reports/majestic-million`.
53. Mockapetris, P.: Domain names - Implementation and Specification. RFC 1035,
54. Mockapetris, P.: Domain names: Concepts and facilities. RFC 1034,
55. Moura, G.C.M. *et al.*: Anycast vs. DDoS: Evaluating the November 2015 root DNS event. In: IMC (2016)
56. Moura, G.C.M. *et al.*: Cache Me If You Can: Effects of DNS Time-to-Live. In: IMC (2019)
57. Moura, G.C.M. *et al.*: Fragmentation, Truncation, and Timeouts: Are Large DNS Messages Falling to Bits? In: PAM (2021)
58. Mozilla Foundation, Public Suffix List, (2022). `https://publicsuffix.org/`.
59. Naab, J. *et al.*: Gotta Query 'Em All, Again! Repeatable Name Resolution with Full Dependency Provenance. In: ANRW (2023)
60. sk-nic, sk-nic OpenData, (2023). `https://sk-nic.sk/subory/domains.txt`.
61. NLnet Labs, Unbound, (2023). `https://unbound.docs.nlnetlabs.nl`.

62. Nosyk, Y. *et al.*: Extended DNS Errors: Unlocking the Full Potential of DNS Troubleshooting. In: IMC (2023)
63. Pappas, V. *et al.*: Impact of Configuration Errors on DNS Robustness. In: SIGCOMM (2004)
64. PowerDNS, DNS Camel, (2024). `https://powerdns.org/dns-camel/`.
65. PowerDNS B.V., Lua Records, (2024). `https://doc.powerdns.com/authoritative/lua-records/`
66. Ramasubramanian, V. *et al.*: Perils of Transitive Trust in the Domain Name System. In: IMC (2005)
67. van Rijswijk-Deij, R. *et al.*: DNSSEC and Its Potential for DDoS Attacks. In: IMC (2014)
68. van Rijswijk-Deij, R. *et al.*: A High-Performance, Scalable Infrastructure for Large-Scale Active DNS Measurements. JSAC (2016)
69. Rosenbaum, R.: Using the Domain Name System To Store Arbitrary String Attributes. RFC 1464,
70. Scheitle, Q. *et al.*: A First Look at Certification Authority Authorization (CAA). CCR (2018)
71. Schlyter, J. *et al.*: Using DNS to Securely Publish Secure Shell (SSH) Key Fingerprints. RFC 4255,
72. Sommese, R. *et al.*: The Forgotten Side of DNS: Orphan and Abandoned Records. In: EuroS&P Workshop (2020)
73. Sommese, R. *et al.*: When Parents and Children Disagree: Diving into DNS Delegation Inconsistency. In: PAM (2020)
74. Sommese, R. *et al.*: This Is a Local Domain: On Amassing Country-Code Top-Level Domains from Public Data, (2023).
75. Sperotto, A. *et al.*: TIDE: Threat Identification Using Active DNS Measurements. In: SIGCOMM Posters and Demos (2017)
76. Stahl, M.: Domain administrators guide. RFC 1032,
77. Steurer, F. *et al.*: *A Tree in a Tree: Measuring Biases of Partial DNS Tree Exploration*, version 1.0 (2024). `https://doi.org/10.17617/3.UBPZXP`.
78. Streibelt, F. *et al.*: Exploring EDNS-client-subnet adopters in your free time. In: IMC (2013)
79. Streibelt, F. *et al.*: How Ready is DNS for an IPv6-Only World? In: PAM (2023)
80. The Linux Foundation, Prometheus - Monitoring system & time series database, (2024). `https://prometheus.io/`.
81. Thompson, K. *et al.*: Wide-area Internet traffic patterns and characteristics. IEEE network (1997)
82. van der Toorn, O. *et al.*: Melting the Snow: Using Active DNS Measurements to Detect Snowshoe Spam Domains. In: NOMS (2018)
83. van der Toorn, O. *et al.*: Saving Brian's privacy: the perils of privacy exposure through reverse DNS. In: IMC (2022)
84. der Toorn, O.v. *et al.*: TXTing 101: Finding Security Issues in the Long Tail of DNS TXT Records. In: EuroS&PW (2020)
85. Vermeer, M. *et al.*: SoK: A Framework for Asset Discovery: Systematizing Advances in Network Measurements for Protecting Organizations. In: EuroS&P (2021)
86. Vissers, T. *et al.*: The Wolf of Name Street: Hijacking Domains Through Their Nameservers. In: SIGSAC (2017)
87. Vixie, P.: A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY). RFC 1996,

88. de Vries, W.B. *et al.*: A First Look at QNAME Minimization in the Domain Name System. In: PAM (2019)
89. Yajima, M. *et al.*: Measuring Adoption of DNS Security Mechanisms with Cross-Sectional Approach. In: GLOBECOM (2021)
90. Zhang, S. *et al.*: Robust or Risky: Measurement and Analysis of Domain Resolution Dependency. INFOCOM (2024)
91. Zirngibl, J. *et al.*: Domain Parking: Largely Present, Rarely Considered! In: TMA (2022)

A Ethical Considerations

Before starting our measurement, we considered ethical implications, following our institution's guidelines and the Menlo Report [42]. Given that we collect generally public data, and analyze only technical aspects, we consider rate-limiting and reducing harm towards other networks as the main ethical objective.

As our measurements may exceed the query load of normal DNS resolution, we took precautions not to concentrate load on individual authoritative nameservers, i.e., by limiting query rates and queries in-flight, reusing TCP connections, not sending malformed DNS packets, and not querying unresponsive servers, see Section 4.1. We seed our measurements from public sources or receive them under an agreement that allows their use for measurements (zone files).

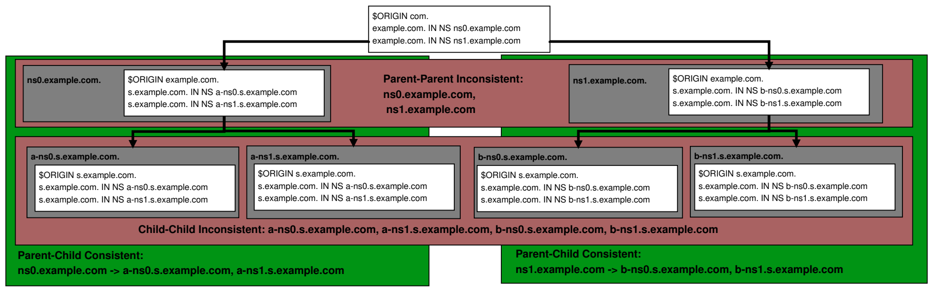
We ensure that our measurements can be attributed to us by (a) hosting a Web page with a project description and contact details on all scan machines; (b) using informative reverse DNS entries; (c) dedicating a network segment to the scans, with associated WHOIS information describing the project and contact details; (d) `YoDNS` enables opt-out; yet we received no requests for this.

In addition, we submit our study design to our institution's ethical review board, which attested no concerns in response to our application No. 23-09-2.

B Hidden Child-Child Inconsistency

Seemingly, the number of child-child inconsistencies in Table 4 is "inconsistent" with the number of worst-case parent-child inconsistencies. Intuitively, one would assume that the number of child-child inconsistencies should always be *lesser or equal* to the number of parent-child inconsistencies. However, we find that some child-child inconsistencies cannot be detected by following a resolution path, if, at the same time, a parent-parent inconsistency exists, see Figure 8.

Here, we have two authoritative NS for a parent zone, `ns0.example.com` and `ns1.example.com`, delegating `s.example.com`. We have a parent-parent inconsistency since they both return different NS sets. Here, `ns0.example.com` could return an NS set for `s.example.com`, containing `a-ns0.s.example.com` and `a-ns1.s.example.com`, along with valid glue for both. At the same time, `ns1.example.com` could return an NS set for `s.example.com`, containing `b-ns0` and `b-ns1.s.example.com`, again, along with glue for both.


```
$ORIGIN com.
example.com. IN NS ns0.example.com
example.com. IN NS ns1.example.com
```

```
$ORIGIN example.com.
s.example.com. IN NS a-ns0.s.example.com
s.example.com. IN NS a-ns1.s.example.com
```

```
$ORIGIN example.com.
s.example.com. IN NS b-ns0.s.example.com
s.example.com. IN NS b-ns1.s.example.com
```

```
$ORIGIN s.example.com.
s.example.com. IN NS a-ns0.s.example.com
s.example.com. IN NS a-ns1.s.example.com
```

```
$ORIGIN s.example.com.
s.example.com. IN NS a-ns0.s.example.com
s.example.com. IN NS a-ns1.s.example.com
```

```
$ORIGIN s.example.com.
s.example.com. IN NS b-ns0.s.example.com
s.example.com. IN NS b-ns1.s.example.com
```

```
$ORIGIN s.example.com.
s.example.com. IN NS b-ns0.s.example.com
s.example.com. IN NS b-ns1.s.example.com
```
ns0.example.com.
Parent-Parent Inconsistent: ns0.example.com, ns1.example.com
ns1.example.com.
a-ns0.s.example.com.
a-ns1.s.example.com.
b-ns0.s.example.com.
b-ns1.s.example.com.
Child-Child Inconsistent: a-ns0.s.example.com, a-ns1.s.example.com, b-ns0.s.example.com, b-ns1.s.example.com
Parent-Child Consistent: ns0.example.com -> a-ns0.s.example.com, a-ns1.s.example.com
Parent-Child Consistent: ns1.example.com -> b-ns0.s.example.com, b-ns1.s.example.com

Fig. 8: Example of a zone with two auth. NSs serving different NS sets, leading to parent-parent and child-child mismatches. The latter only become visible when all parts of the DNS tree are resolved, i.e., all four children are detected.

If we now query the NS set for `s.example.com` on `a-ns0` and `a-ns1.s.example.com`, we receive two consistent NS sets containing only `a-ns0` and `a-ns1.s.example.com`. Similarly, if we query `b-ns0` and `b-ns1.s.example.com`, we would receive an NS set containing only `b-ns0` and `b-ns1.s.example.com`.

However, when we evaluate the *complete* zone tree, there is not only an obvious inconsistency between the parents but also between the NSs authoritative for `s.example.com`, the child. All *four* children are inconsistent.

However, our three synthesized perspectives follow a DNS resolution path: If we traverse the tree, regardless of whether we do it randomly (rnd.), or in an attempt to minimize (resp. maximize) paths with parent-child mismatches, there is no path down the tree that lets us receive the parent NS set from `ns0.example.com` while also receiving responses from `b-ns0` or `b-ns1`. We can only evaluate the NS sets from `a-ns0` and `a-ns1`, that are consistent to each other.

Hence, numbers in Table 4 may *look* odd but are correct, as *no* standard DNS resolution strategy for zones would identify inconsistent parents returning internally-consistent NS sets. Only exploring the full DNS tree will find such cases, which are especially prevalent in $>2^{nd}$ level zones.