ZDNS: A Fast DNS Toolkit for Internet Measurement

Liz Izhikevich
Stanford University

Gautam Akiwate
Stanford University

Briana Berger
Stanford University

Spencer Drakontaidis
Stanford University

Anna Ascheman
Stanford University

Paul Pearce
Georgia Institute of Technology

David Adrian
Stanford University

Zakir Durumeric
Stanford University

ABSTRACT

Active DNS measurement is fundamental to understanding and improving the DNS ecosystem. However, the absence of an extensible, high-performance, and easy-to-use DNS toolkit has limited both the reproducibility and coverage of DNS research. In this paper, we introduce ZDNS, a modular and open-source active DNS measurement framework optimized for large-scale research studies of DNS on the public Internet. We describe ZDNS's architecture, evaluate its performance, and present two case studies that highlight how the tool can be used to shed light on the operational complexities of DNS. We hope that ZDNS will enable researchers to better—and in a more reproducible manner—understand Internet behavior.

ACM Reference Format:
Liz Izhikevich, Gautam Akiwate, Briana Berger, Spencer Drakontaidis, Anna Ascheman, Paul Pearce, David Adrian, and Zakir Durumeric. 2022. ZDNS: A Fast DNS Toolkit for Internet Measurement. In *ACM Internet Measurement Conference (IMC '22), October 25–27, 2022, Nice, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3517745.3561434

1 INTRODUCTION

The Domain Name System (DNS) plays a critical role on the Internet, from acting as the phone book of the web to controlling traffic routes for major content delivery networks [3, 11] and authenticating services [42]. DNS has proven to be complex to manage, which, paired with its ubiquity, has led to major Internet outages [53, 65, 68] and the hijacking of prominent services [20, 46]. With billions of names, millions of resolvers, and dozens of types of records defined across hundreds of RFCs, DNS has become a massively complex and distributed ecosystem that is not fully visible to researchers. Further inhibiting visibility, DNS behavior is often hidden behind recursive and caching resolvers.

While there exist many tools for actively querying DNS, none expose internal DNS operations (e.g., responses from each step of the recursive process) while scaling to today's namespace and providing the extensibility needed for quickly answering new types of research questions. Consequently, researchers frequently resort to building their own specialized scanning solutions [21, 49, 62], which are expensive and error prone to develop. Further, most tools have remained closed source, hampering reproducibility.

In this work, we introduce ZDNS, an open source measurement toolkit for large-scale active DNS research. ZDNS is composed of: (1) a DNS library that exposes the internal characteristics of DNS operations, (2) a core framework that isolates and orchestrates non-DNS specific scanning logic, and (3) composable modules that allow researchers to easily add new DNS queries and records, including 65 already-implemented record types. Critically, ZDNS implements its own internal caching and recursion, which is key to exposing internal DNS operations and querying a large number of unique names. ZDNS provides a simple command line interface for scanning and outputs results in programmatically interpretable JSON.

We evaluate ZDNS and show that it successfully resolves 85 times more domains per second than prior work [7], performing 90K lookups per second when using an external recursive resolver. ZDNS resolves 50M domains in 10 minutes and queries the PTR records of the full public IPv4 address space in 12 hours. When performing its own internal recursion, ZDNS exposes all recursive details to the user and resolves 50M domains in 46 minutes and 100% of the public IPv4 address space in 116 hours. We supplement our evaluation with two case studies that explore (1) the customizability and extensibility of ZDNS when exposing internal DNS operations to analyze redundant nameserver deployment, and (2) the versatility of ZDNS and the importance of accounting for blind spots in the coverage of open data sets.

Since its open source release, ZDNS has been used by a series of Internet measurement papers [22, 26, 27, 32, 33, 50–52, 54, 56–59, 61, 67] and serves as the foundation for several open data sets [23, 28, 45]. Given its widespread use and now stabilized codebase, we formally introduce ZDNS to the community in order to promote an awareness and full understanding of the tool's motivation, architecture, performance, capabilities, and caveats. High-performance, open source, Internet-wide scanners (e.g., ZMap [29], ZGrab [28], Masscan [38]) have already helped break down the barrier of scalability and reproducibility for hundreds of research papers that study the Internet ecosystem; we hope that ZDNS will do the same for DNS. ZDNS is released under the Apache 2.0 licence at https://github.com/zmap/zdns.

2 RELATED WORK

Active DNS measurement has been critical to understanding the operation [33, 56, 59, 61], privacy [26, 32, 57], and security [50, 51, 58, 66] of the DNS ecosystem. There exist a wide range of tools from dig [7] to MassDNS [12]—a high-performance stub resolver that was developed in parallel to ZDNS. Often, these tools are paired with a public resolver like Cloudflare's `1.1.1.1` [25] or Google's

IMC '22, October 25–27, 2022, Nice, France

ACM ISBN 978-1-4503-9259-4/22/10…$15.00
https://doi.org/10.1145/3517745.3561434

`8.8.8.8` [37], or an external recursive resolver like PowerDNS [17], Unbound [14], or Bind [4]. We evaluate these tools in Section 4.

However, existing tools do not always extend to the research question at hand. For example, MassDNS is not engineered to perform iterative lookups and expose the DNS lookup chain. Unbound implements only features based on RFC standards [15], resulting in over 30 feature requests that have been unresolved for years [16]. Consequently, researchers regularly develop their own purpose-built measurement tools [21, 49, 62]. These independent scanning solutions are not always optimized for performance—frequently relying on sub-sampling the DNS/rDNS search-space [34, 60]—and are not always public [21, 49, 62], therefore hindering reproducibility and instigating an unnecessary reinvention of the same tool.

The measurement community has also produced datasets built on top of active DNS querying, including OpenINTEL [64], ActiveDNS [45], Rapid 7 [48], Farsight [10], and Censys [28]. More than 100 papers have relied on these datasets to understand [62, 63] and protect [24, 44] DNS infrastructure. While these open data sets have considerably benefited the community, we demonstrate in Section 6, along with prior work [35], that the selective coverage of data sets often requires supplemental active measurement (e.g., measuring additional zones or response inconsistencies). Notably, ActiveDNS and Censys are built on top of ZDNS.

3 ZDNS ARCHITECTURE

In this section, we describe ZDNS's architecture and guiding principles that inform its extensible and performant design. To support reproducibility, ZDNS is open source and can be found at https://github.com/zmap/zdns.

3.1 Requirements and Guiding Principles

We built ZDNS because there were no readily available tools for quickly performing millions of unique DNS queries. The following observations, goals, requirements, and lessons learned from other open source measurement tools, inform our architecture:

Internal Recursion. Recursive resolvers hide many characteristics of DNS operations (e.g., lame and dangling delegations [21]). Further, public recursive resolvers often rate limit queries [2]—cloud providers like Amazon have explicitly asked that we not use their infrastructure for large scale experiments—and open source resolvers like Unbound [14] are not easily extensible. ZDNS must be able to perform its own recursive resolution and expose lookup chains in addition to supporting external recursive resolvers.

High Performance. DNS experiments frequently require querying a large number of names. There are 1.5B unique FQDNs in public Certificate Transparency logs [5], 161M domains in the Verisign `.com` zone file [13], 6M recursive resolvers on the public Internet [5], and collecting reverse PTR records for all of IPv4 requires querying 3.7B publicly accessible IPv4 addresses. ZDNS needs to be performant enough to lookup PTR records for all IPs in a few days and known names in a few hours to keep up with regular Internet churn.

Safe. The DNS protocol is defined across at least 285 RFCs and there are more than 65 types of DNS records in 2022. Internet servers regularly respond with malformed responses due to misconfiguration and intentionally malicious operators. ZDNS needs to be written in a memory-safe language and support a modular interface where researchers can easily and safely implement new functionality.

Extensible. Complex, difficult-to-use tools lead to measurement error because there is no ground truth against which to compare results. Similarly, operators will not use tools with a high barrier to entry, instead opting to build other tools, which may not provide the same level of accuracy. ZDNS must be easy to understand and use. This precludes any complex, distributed setups and dictates that ZDNS be easy to install and collect correct results.

3.2 Architecture

ZDNS architecture is composed of three primary components: (1) a DNS library that fully exposes DNS lookup chains, (2) a framework for easy command-line interaction, and (3) composable modules that facilitate easy extensibility. We detail each below:

DNS Library. ZDNS implements its own caching recursive resolver library on top of Miek Gieben's DNS implementation [36]. The ZDNS library provides recursive lookups, caching, validation, exposed transcripts of exchanged packets and easy DNS question construction and answer parsing. It also supports using an external recursive resolver and various performance optimization (e.g., UDP socket reuse), which we discuss in Section 3.4. ZDNS modules are given direct access to the DNS library, eliminating the need to redundantly implement DNS query logic for varying DNS queries/records.

Framework. The core framework is responsible for facilitating command-line configuration, spawning lookup routines, delegating control to modules, encoding, decoding and routing data to/from routines, load balancing against upstream resolvers, aggregating run-time statistics, and ensuring consistent output. The framework is light-weight, accounts for only a quarter of the ZDNS code base, and is absent of most DNS-specific logic.

Modules. Modules are responsible for providing DNS query-specific logic, including logic specific to a DNS query or record, expected query response, additional command line flags, global and per-routine initialization. Developers add modules by implementing a Go interface that defines a `DoLookup` function that accepts two parameters: (1) a name to be queried and (2) a name server. `DoLookup` returns an `interface` with JSON export tags (i.e., a struct with tags that hint to our framework how to label the fields when converting the struct to JSON). As we later discuss, because the ZDNS framework orchestrates concurrency through light-weight routines, `DoLookup` functions can be simple (e.g., open a UDP socket, create and send packet, wait for response or socket timeout, close socket, and return any relevant data from the response packet). We show an example `DoLookup` function and a complete module in Appendix B.

By having an existing core framework and helper library, many simple modules (e.g., `CAA` query) are implemented in a few lines of code that simply set the query type and return the resulting DNS answers. ZDNS modules support both recursive and iterative lookups, and can be used to query a single resolver for a large number of names, a large number of servers for a single name, or a combination of both.

3.3 Implemented Modules

ZDNS currently includes three types of modules:

Raw DNS modules. Basic modules provide the raw DNS response from a server similar to dig, but as structured JSON records. There exist modules for most types of DNS records today.[1]

Lookup modules. Raw DNS responses frequently do not provide the data users need. For example, an `MX` response may not include the associated `A` records in the additionals section, requiring an additional lookup. To address this gap and provide a friendlier interface, we also provide several alternative lookup modules: alookup and mxlookup. mxlookup will additionally do an `A` lookup for the IP addresses that correspond with an exchange record. alookup acts similar to nslookup and will follow CNAME records.

Misc modules. Misc modules provide other alternative means of querying servers, such as extracting the version of resolvers (e.g., `bind.version`).

3.4 Performance Optimizations

While ZDNS's architecture facilitates extensibility, several optimizations are critical to its performance.

Parallelism. The majority of clock time expended making a DNS query is spent waiting for a response rather than constructing or parsing DNS packets. Thus, efficiently parallelizing a sufficiently large number of concurrent queries is crucial to achieve the performance we need. Inspired by ZGrab [28]'s success—a popular open source high-performance application layer scanner written in Go—, we build ZDNS in Go such that we can utilize the language's ability to efficiently manage thousands of concurrent queries using lightweight routines. Go is memory-safe and garbage collected, which facilitates providing a safe but extensible platform while remaining highly performant. While ZDNS's architecture is similar to ZGrab, the ratio of waiting to work is much higher than for application handshakes which often require expensive cryptographic operations or parsing large amounts of data. As we discuss in the next section, optimal performance requires around 50K concurrent queries—about 5–10 times more than ZGrab—which introduces new challenges.

UDP Socket Reuse. Creating a socket for every lookup is exorbitantly expensive because each socket is used to send and receive only two packets before being torn down and recreated. Instead, we establish and maintain a single long-lived raw UDP socket in each lightweight routine for the lifetime of the program execution. Raw UDP sockets bind to a static source port, and can be used to send UDP packets to arbitrary destination IP/ports. This eliminates per-connection socket overhead, without requiring us to manually construct IP and Ethernet headers for each request. To support more threads than the size of the OS ephemeral port range, we support binding to multiple IP addresses. This approach does not accommodate TCP queries (e.g., truncated records) but we find that this rarely occurs in practice (e.g., when querying the A records of a random sub-sample of 10 million domain names found in the Censys certificate transparency logs, 0.4% responses return truncated). Thus, ZDNS by default only establishes TCP connections as needed. Nevertheless, ZDNS can optionally be configured to scan using only TCP.

Selective Caching. While popular recursive resolvers like Unbound cache results in order to quickly respond to lookups for frequently queried names, we expect ZDNS users to frequently lookup unique names. Caching remains critical to avoid repeating initial queries in iterative resolutions (e.g., repeatedly querying root resolvers for `.com`'s name servers), but caching the results for the names queried causes unnecessary thrashing. Thus, we selectively cache only `NS` and glue records to help with future recursion, but do not cache any results for the leaf names being directly queried.

Increased Garbage Collection. Decreasing the frequency of garbage collection is typically associated with improved performance. However, the opposite is true for ZDNS. Quadrupling the frequency that garbage collection occurs increases the throughput, likely because short collections can be interspersed between other request processing and do not cause connections to timeout while waiting for garbage collection to finish.

As we detail in the next section, our optimizations allow ZDNS to consistently perform over 90K lookups per second across billions of consecutive lookups.

3.5 Ethics

Any open source scanner can be used by researchers to understand and protect the Internet, and abused by adversaries to find vulnerabilities. We design ZDNS with the ability to perform its own recursion internally in order to avoid overloading public recursive resolvers. ZDNS parameters allow the user to directly modulate the number of lookups-per-second, thereby minimizing packet drop and the unnecessary overloading of external networks and servers. We include an extensive README with ZDNS that recommends users to coordinate with their upstream provider before performing high-volume scans. When executing scans and developing ZDNS, we follow the community standards for good Internet citizenship outlined by Durumeric et al. [29].

4 EVALUATION

We evaluate ZDNS's scalability, execution time, and success rate when performing billions of queries and compare ZDNS to a set of existing tools. We show that ZDNS queries the PTR records of all IPv4 addresses in 12 hours using Google's public recursive resolver, and in 116 hours using ZDNS's own iterative resolver. ZDNS achieves 2.6–85 times more successful queries per second and up to 35% less packet drop compared to existing tools.

4.1 Performance and Scalability

ZDNS's performance is dependent on a variety of configuration parameters. We evaluate ZDNS's execution time and success rate (i.e., a `NOERROR` or `NXDOMAIN` response) performing `A` and `PTR` lookups while modulating available resources. We evaluate performance using Cloudflare and Google's public recursive resolvers as well as ZDNS's own iterative resolver.

[1] We support querying for and parsing these types of records: A, AAAA, AFSDB, ANY, ATMA, AVC, AXFR, CAA, CDNSKEY, CDS, CERT, CNAME, CSYNC, DHCID, DMARC, DNSKEY, DS, EID, EUI48, EUI64, GID, GPOS, HINFO, HIP, ISDN, KEY, KX, L32, L64, LOC, LP, MB, MD, MF, MG, MR, MX, NAPTR, NID, NINFO, NS, NSAPPTR, NSEC, NSEC3, NSEC3PARAM, NXT, OPENPGPKEY, PTR, PX, RP, RRSIG, RT, SMIMEA, SOA, SPF, SRV, SSHFP, TALINK, TKEY, TLSA, TXT, UID, UINFO, UNSPEC, and URI.

We perform all experiments with 24 virtual cores, 16GB of system memory, one process per CPU core (the Go default), and an available 45K ephemeral ports. While we do not vary the server specifications for benchmarking, we vary the amount of threads and cache-size that ZDNS is exposed to, which can be used as an approximation for measuring the computational resources ZDNS requires. We find that a single virtual core uses 100% of resources at approximately 2K ZDNS threads and RAM usage never exceeds 10GB across all cache sizes in our experiments. Domain names are drawn from a corpus of 234M fully qualified domain names found on unexpired, browser-trusted certificates in Censys [28]; we provide a breakdown of the domains in Appendix A. We do not overlap names or IPv4 addresses between consecutive trials to minimize the impact of resolver caching.

Results. ZDNS's performance is directly dependent on choosing the optimal number of threads (i.e., light-weight Go routines), the resolver, and number of client IPs. We modulate between 1K–100K threads and scanning from a /32, /29, or /28 sub-network of IPs while resolving 10M domains/IPs in Figure 1. Across resolvers, peak performance plateaus at roughly 50K threads, with ZDNS successfully completing 91.6K A lookups per second when using the Cloudflare resolver, and 102K PTR lookups per second when using Google's resolver. The number of available scanning IPs limits the number of threads ZDNS can use, as each thread requires its own socket and unique source port to send and receive traffic. Furthermore, we experience a Google per-client-IP rate-limiting [2] when using a single IP client IP address, decreasing the successes rate by a factor of six compared to using Cloudflare's resolver, which does not rate limit clients [1].

To respect resolver rate limiting, and to expose the internal characteristics of DNS operations, ZDNS performs its own recursive resolution at a peak performance of 18K lookups per second. This performance dip is due to (1) ZDNS's inability to leech off of a public resolver's full cache, and (2) the increased number of iterative queries that ZDNS must perform to obtain the final response. The number of queries that ZDNS's iterative resolver sends in order to receive the final record is nearly equivalent to the number of queries and successes/second when using the Google resolver; at 50K threads, ZDNS sends 67K queries per second when resolving PTR records for IPv4 addresses compared to Google's 71K successes per second.

ZDNS's iterative resolver relies on its own selective caching of responses in order to avoid repeating queries and achieve greater performance. We evaluate ZDNS's A and PTR lookup performance while modulating cache size between 50K–1M entries while using 50K threads. Figure 2 illustrates that while increasing ZDNS's cache size creates a marginal increase (less than 5%) in cache hit rate, it increases the number of successes per second greater than three-fold. Performance plateaus at a cache size of 600K entries.

ZDNS performance scales when performing large amounts of consecutive lookups (Table 1). For example, ZDNS resolves the entire IPv4 address space in 12.1 hours when using Google's public resolver and in 116.7 hours when using its own iterative resolver, while maintaining a success rate of 93% and 88.5%, respectively. ZDNS's success rate therefore drops less than 2% when scaling from millions to billions of lookups.

Lookup	Resolver	# Domains/IPs	% Successes	Time
A	Google	50M	96.4%	10.6m
A	Cloudflare	50M	97.0%	10.3m
A	Iterative	50M	96.7%	46.3m
PTR	Google	100% IPv4	93.0%	12.1h
PTR	Cloudflare	100% IPv4	93.5%	12.9h
PTR	Iterative	100% IPv4	88.5%	116.7h

Table 1: ZDNS Performance—ZDNS resolves 100% of the IPv4 address space in 12.1 hours using a public recursive resolver and in 116.7 hours using its own iterative resolver.

4.2 Alternative Approaches

While there are few systems specifically architected for large-scale DNS measurement research, ZDNS is not the first system to implement an exposed lookup chain, recursive/caching resolving, or stub resolving. We compare ZDNS performance against three popular tools—Dig [7], Unbound [14], MassDNS [12]—when resolving 10M random IPs/domains with the same server configuration from Section 4.1. We configure ZDNS to use 60K threads, a cache-size of 600K entries, and up to 5 retries per query.

Exposed Lookup Chain. Dig [7] is a command-line tool used for troubleshooting DNS and provides the ability to "trace"/expose the full DNS lookup chain by iteratively querying a domain starting from its root nameserver. Dig was never designed to be a high performance scanning engine and we find that its batch lookup performs an average 0.5 A/PTR traces per second. Forking individual dig processes is more efficient and achieves a peak performance of 120 successful A lookups per second when using Cloudflare's resolver to perform a trace query. Beyond its shy performance, Dig's output is not programmatically interpretable, requiring an additional tool for parsing. We compare ZDNS and Dig's outputs in Appendix C.

Recursive/Caching Resolving. Unbound [14] is a recursive resolver that is commonly used to provide recursive DNS services to clients. To fairly evaluate Unbound's performance using the same resources as ZDNS's iterative resolver, we install a performance-optimized [47] version of Unbound on ZDNS's server[2] and configure ZDNS to use the locally installed resolver. Unbound is less CPU efficient than ZDNS's iterative resolver, creating resource contention and capping ZDNS to a maximum of 10K and 5K threads when querying PTR and A records, respectively. While ZDNS and Unbound achieve the same number of successes, ZDNS's iterative resolver successfully resolves 2.6–3.6 times more names per second (Table 2).

Stub Resolver. MassDNS [12] is a high-performance DNS stub resolver that was developed concurrently to ZDNS. Unfortunately, during our evaluation, we find that its default behavior overwhelms DNS resolvers, which causes 35% of responses to either drop or instigate a SERVFAIL (Table 2). To overcome the high failure rate, MassDNS performs up to an additional 50 retries, which further overloads resolvers. We caution users to approach the tool carefully given the potential to overload servers.

[2]In practice, Unbound resolvers are often not co-located with the querying program.

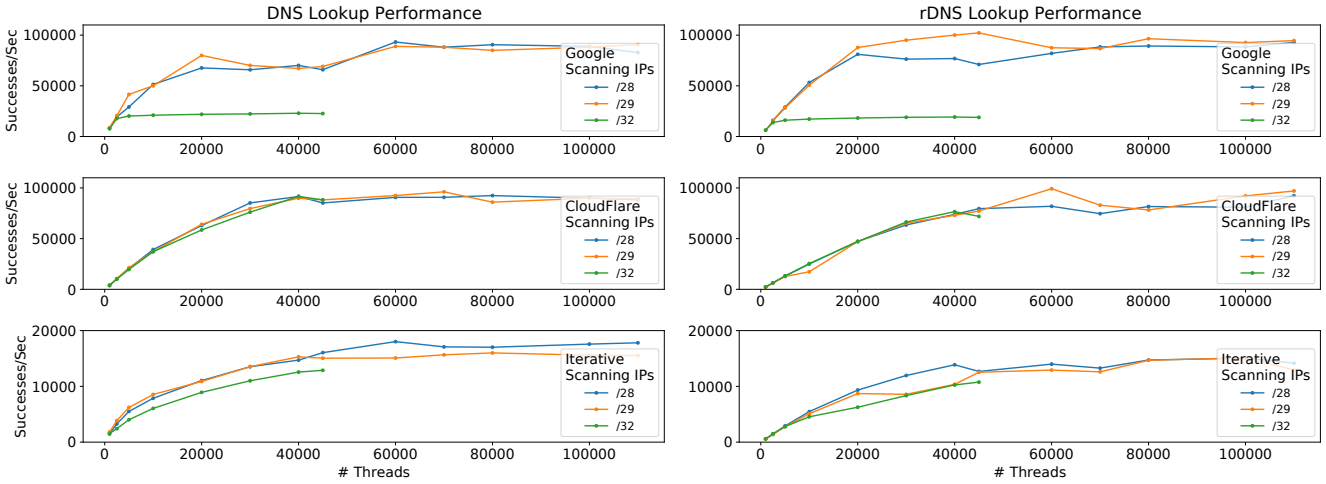

Figure 1: ZDNS Scalability— When using at least 45K threads, ZDNS's successfully performs 91.6K A lookups per second when using the Cloudflare resolver, and 102K PTR lookups per second when using Google's resolver. When ZDNS uses one IP address (i.e., a /32) to scan from, it encounters a socket limit—thereby not being able spawn more threads—and a Google rate limit that decreases the success rate by a factor of six.

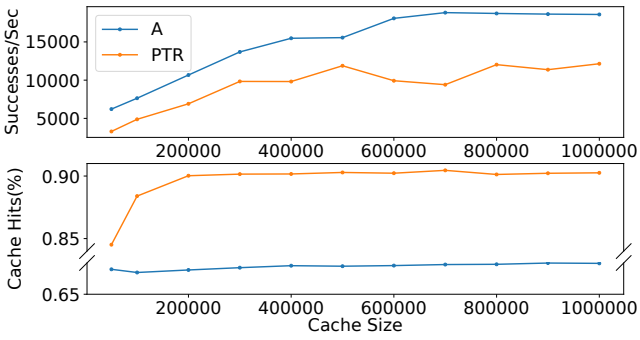

Figure 2: ZDNS Internal Resolver Cache Performance—Increasing the cache size results in over 3 times the amount of successes/second.

Tool	Lookup	Resolver	Success/Sec	% Total Success
MassDNS	A	Google	197K	65%
	PTR	Google	179K	61%
	A	Cloudflare	224K	67%
	PTR	Cloudflare	183K	63%
ZDNS	A	Unbound	4.9K	96%
	PTR	Unbound	4.5K	91%
	A	Iterative	18K	97%
	PTR	Iterative	11.8K	90%
	A	Google	93.1K	96%
	PTR	Google	88.8K	93%
	A	Cloudflare	92.5K	97%
	PTR	Cloudflare	99.1K	94%

Table 2: Alternatives vs ZDNS: ZDNS's iterative resolver performs 2.6–3.6 more successful queries per second than Unbound, and experiences roughly 30% less packet drop than MassDNS.

5 CASE STUDY: NAMESERVER (IN)CONSISTENCY

In this section, we demonstrate the extensibility and utility of ZDNS's iterative resolver by analyzing the availability and consistency of redundant nameserver deployment. To increase DNS reliability, RFC 1034 [55] and 2182 [30] require that a zone must have topologically distributed redundant nameservers. Prior work has found, however, that nameservers are not always redundant and vary within their ability to support TCP fallback [49], CAA records returned [62], and parent-child delegations [63].

To compare nameserver responses within the lookup chain, we use ZDNS's iterative resolver and add functionality to query and record the responses of all name servers when resolving a domain (i.e., an "*–all-nameservers*" flag). The functionality is implemented in 30 lines of code. Using the new functionality, ZDNS resolves all 234M fully qualified domain names from our evaluation set (Appendix A), allowing up to 10 retries for each query to minimize transient errors and approximate the availability of servers. ZDNS completes the scan in 18.5 hours.

Availability. We compare the availability of nameservers for each domain by comparing the number of queries retried across nameservers. Nameservers are remarkably consistent in availability; only 0.55% of resolvable domains have at least one nameserver that needs at least two retries. Interestingly, 0.01% of domains have at least one nameserver that requires 10 retries in order to elicit a response, 31% of which belong to the namebrightdns.com domain. Domains belonging to the Vietnam ccTLD and Nigerian ccTLD are also often unavailable, contributing to 11% and 7% of the inconsistent domains, respectively. We find no relationship between domain categories (e.g., medical, entertainment) and domain availability when categorizing domains using Cloudflare's DNS categories [9].

Upon follow-up probing, we find that the availability inconsistencies are likely not due to server overload, but rather a temporary probabilistic blocking, similar to SSH [66], in which single, yet consecutive, queries cause a temporary response timeout.

Response Consistency. Over 99.99% of domains return consistent sets of A records across nameservers, due to the high-centralization of domains being hosted by large response-consistent providers such as Cloudflare (12%), and GoDaddy (12%). Our results paint a much more consistent picture of DNS compared to prior work [63], likely due to the different source of domain names (i.e., CT logs vs. zone files) and the continued centralization of the Internet.

6 CASE STUDY: CAA RECORDS

In this section, we demonstrate ZDNS's versatility with alternative record types by analyzing the Certification Authority Authorization (CAA) ecosystem. CAA records allow a domain owner to specify the Certificate Authorities (CAs) authorized to issue certificates; the expectation is that a CA will validate the CAA record before certificate issuance [39, 40]. Notably, Scheitle et al., [62]—the first and primary work to analyze the CAA ecosystem—explicitly advocated for the community to develop an open-source tool capable of querying CAA records. Furthermore, while existing data sets [64] query CAA records on a continuous basis, they exclude the vast majority of ccTLDs due to lack of zone files, which we show contribute to nearly half of all CAA record holders.

ZDNS implements CAA records by changing less than five lines of code in the template module shown in Appendix B, and adding 15 lines of additional code tailored to CAA-records across two other framework-specific files. We use ZDNS to query the CAA records of all 93M base domains from our evaluation set (Appendix A), in which 55% of domains are legacy generic TLDs, 39% are country code TLDs, and 6% are new generic TLDs. ZDNS is able to follow CNAMEs for CAA validation, per RFC 8659 [40].[3]

CAA Deployment. Of the 64M domains with a NOERROR response, 1.08M domains (1.69%) respond to CAA queries, with 8,000 domains requiring the CNAME chain to be followed to obtain the CAA record. ccTLDs are 20% more likely to hold a CAA record than a gTLD and contribute to 48% of all CAA records. The `.pl` ccTLD—a ccTLD absent from most open data sets—alone accounts for 25% of all CAA-enabled ccTLD domains. The top 10 ccTLDs together account for 70% of all CAA-enabled ccTLD domains.

CAA Configuration. The vast majority of the CAA enabled domains configure the tags correctly: 99% of domains use either or both the `issue` (96.8%) and `issuewild` (55.27%) tags. While the use of `iodef` tags is generally limited (6.87%), 647 domains—many appearing to be affiliated with Visa Inc—use only the `iodef` tag. Further, we find that 459 domains (0.04%) are configured with invalid tags. We trace the root cause of the majority of these invalid tags to incorrect input validation by a large registrar. We report the issue to the registrar, prompting them to fix the issue.

CAA Issuers. In 2017, Scheitle et al., [62] found that Let's Encrypt [19] was present in roughly 60% of CAA records We find that, five years later—and when considering ccTLDs—Let's Encrypt is present nearly all (92.4% of `issue` and 93.48% of `issuewild`) CAA records. Further, Comodo [6] and Digicert [8] are now present in over 50% of domains.

[3] If `cname.example.com` is a CNAME to `example.net`, then on a request to issue certificate for `cname.example.com` ZDNS follows the CNAME chain to request the CAA records for `example.net` [31, 40].

7 CHALLENGES AND FUTURE WORK

While ZDNS has enabled efficiently querying large numbers of names, the DNS ecosystem is continually evolving and there are several avenues for future work. This includes extending ZDNS support for encrypted DNS lookups, including DNS over HTTPS (DoH) [41] and DNS over TLS (DoT) [43]. Unfortunately, encrypted DNS protocols require ZDNS to maintain a TCP connection, eschewing the UDP socket re-use optimization that contributes to ZDNS performance (Section 3). Furthermore, ZDNS will need to integrate a TLS library (e.g., ZCrypto [18]) which will cause additional latency when performing cryptographic computations and maintaining stateful TLS connections. To maintain ZDNS's fast performance, we will explore adding optimizations to ZDNS's core architecture such as the integrating the reuse of TLS connections across multiple resolutions. Other community-requested features include exposing the capabilities of ZDNS as a library, integrating additional tests, and adding more metadata to returned results.

8 CONCLUSION

ZDNS is a modular, extensible, fast, open-source DNS toolkit optimized for quickly and safely performing billions of recursive DNS queries within hours. As the domain name system continues to grow in search space, add more record types, and implement more complex functionality, ZDNS is built to effortlessly scale and to be easily extended. We hope that ZDNS helps researchers to better understand, build, and secure the DNS ecosystem. ZDNS is released under the Apache 2.0 licence at https://github.com/zmap/zdns.

ACKNOWLEDGEMENTS

We thank the dozens of open source contributors to the ZDNS project, as well as Stefan Savage, Geoffrey Voelker, Gerry Wan, Tatyana Izhikevich, Katherine Izhikevich, Vishal Mohanty, Deepak Kumar, Brian Dickson, Duane Wessels, members of the Stanford University security and networking groups, our shepherd, Fabián E. Bustamante, and the anonymous reviewers for providing insightful discussion and comments. This work was supported in part by the National Science Foundation under award CNS-1823192, Google., Inc., the NSF Graduate Fellowship DGE-1656518, the Office of Naval Research under award N00014-18-1-2662, and a Stanford Graduate Fellowship.

REFERENCES

[1] 2019. 1.1.1.1 Rate Limiting. https://community.cloudflare.com/t/is-there-any-rate-limiting-for-1-1-1-1-dns-queries/137206. (2019).
[2] 2021. Google Public DNS Rate-limiting queries. https://developers.google.com/speed/public-dns/docs/security#rate_limit. (2021).
[3] 2022. Amazon CloudFront. https://aws.amazon.com/cloudfront/. (2022).
[4] 2022. Bind 9. (2022). https://www.isc.org/bind/.
[5] 2022. Censys Search. https://search.censys.io/search. (2022).
[6] 2022. Comodo Certificate Authority. (2022). https://www.comodoca.com.
[7] 2022. dig - Linux Man Page. https://linux.die.net/man/1/dig. (2022).
[8] 2022. Digicert. (2022). https://www.digicert.com.
[9] 2022. DNS Categories. (2022). https://developers.cloudflare.com/cloudflare-one/policies/filtering/dns-policies/dns-categories/.

[10] 2022. Farsight Security. (2022). https://www.farsightsecurity.com.
[11] 2022. Global Traffic Management: How it works. https://techdocs.akamai.com/gtm/docs/how-it-works. (2022).
[12] 2022. MassDNS 0.3. https://github.com/blechschmidt/massdns. (2022).
[13] 2022. Top-Level Domain Zone File Information. https://www.verisign.com/en_US/channel-resources/domain-registry-products/zone-file/index.xhtml. (2022).
[14] 2022. Unbound. https://www.nlnetlabs.nl/projects/unbound/about/. (2022).
[15] 2022. Unbound - RFC Compliance. (2022). https://unbound.docs.nlnetlabs.nl/en/latest/reference/rfc-compliance.html.
[16] 2022. Unbound Issues. (2022). https://github.com/NLnetLabs/unbound/issues.
[17] 2022. Welcome to PowerDNS. (2022). https://www.powerdns.com.
[18] 2022. ZCrypto. (2022). https://github.com/zmap/zcrypto.
[19] Josh Aas, Richard Barnes, Benton Case, Zakir Durumeric, Peter Eckersley, Alan Flores-López, J Alex Halderman, Jacob Hoffman-Andrews, James Kasten, Eric Rescorla, et al. 2019. Let's Encrypt: an automated certificate authority to encrypt the entire web. In *ACM Conference on Computer and Communications Security*.
[20] Danny Adamitis, David Maynor, Warren Mercer, Matthew Olney, and Paul Rascagneres. 2019. DNS Hijacking Abuses Trust In Core Internet Service. (April 2019). https://blog.talosintelligence.com/2019/04/seaturtle.html.
[21] Gautam Akiwate, Mattijs Jonker, Raffaele Sommese, Ian Foster, Geoffrey M Voelker, Stefan Savage, and KC Claffy. 2020. Unresolved Issues: Prevalence, Persistence, and Perils of Lame Delegations. In *Proceedings of the ACM Internet Measurement Conference*. 281–294.
[22] Ludovic Barman, Sandra Siby, Christopher Wood, Marwan Fayed, Nick Sullivan, and Carmela Troncoso. 2022. This is not the padding you are looking for! On the ineffectiveness of QUIC PADDING against website fingerprinting. *arXiv preprint arXiv:2203.07806* (2022).
[23] CAIDA. [n. d.]. Complete Routed-Space DNS Lookups. ([n. d.]). https://www.caida.org/catalog/datasets/complete_dns_lookups_dataset/.
[24] Taejoong Chung, Roland van Rijswijk-Deij, David Choffnes, Dave Levin, Bruce M Maggs, Alan Mislove, and Christo Wilson. 2017. Understanding the role of registrars in DNSSEC deployment. In *ACM Internet Measurement Conference*.
[25] Cloudflare. [n. d.]. 1.1.1.1 - the Internet's fasters, privacy-first DNS resolver. ([n. d.]). https://1.1.1.1/dns/.
[26] Yana Dimova, Gunes Acar, Lukasz Olejnik, Wouter Joosen, and Tom Van Goethem. 2021. The cname of the game: Large-scale analysis of dns-based tracking evasion. *arXiv preprint arXiv:2102.09301* (2021).
[27] Kristen Dorey. 2017. An Internet-Wide Analysis of Diffie-Hellman Key Exchange and X. 509 Certificates in TLS. (2017).
[28] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J. Alex Halderman. 2015. A Search Engine Backed by Internet-Wide Scanning. In *ACM Computer and Communication Security*.
[29] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. 2013. ZMap: Fast Internet-Wide Scanning and its Security Applications. In *USENIX Security Symposium*.
[30] R. Elz, R. Bush, S. Bradner, and M. Patton. 1997. Selection and Operation of Secondary DNS Servers. IETTF RFC 2182. (1997).
[31] Let's Encrypt. 2017. Certificate Authority Authorization (CAA). (jul 2017). https://letsencrypt.org/docs/caa/.
[32] Rahel A Fainchtein, Adam J Aviv, Micah Sherr, Stephen Ribaudo, and Armaan Khullar. 2021. Holes in the Geofence: Privacy Vulnerabilities in" Smart" DNS Services. *Privacy Enhancing Technologies* (2021).
[33] Rodérick Fanou, Bradley Huffaker, Ricky Mok, and Kimberly C Claffy. 2020. Unintended consequences: Effects of submarine cable deployment on Internet routing. In *Conference on Passive and Active Network Measurement*.
[34] Tobias Fiebig, Kevin Borgolte, Shuang Hao, Christopher Kruegel, and Giovanni Vigna. 2017. Something from nothing (There): collecting global IPv6 datasets from DNS. In *Conf. on Passive and Active Network Measurement*.
[35] Tobias Fiebig, Kevin Borgolte, Shuang Hao, Christopher Kruegel, Giovanni Vigna, and Anja Feldmann. 2018. In rDNS we trust: revisiting a common data-source's reliability. In *Conference on Passive and Active Network Measurement*.
[36] Miek Gieben. [n. d.]. Alternative (more granular) approach to a DNS library. https://github.com/miekg/dns. ([n. d.]).
[37] Google. [n. d.]. Google public DNS. ([n. d.]). https://developers.google.com/speed/public-dns.
[38] Robert David Graham. 2014. MASSCAN: Mass IP port scanner. (2014). https://github.com/robertdavidgraham/masscan.
[39] Phillip Hallam-Baker and Rob Stradling. 2013. DNS Certification Authority Authorization (CAA) Resource Record. RFC 6844. (Jan. 2013). https://doi.org/10.17487/RFC6844
[40] Phillip Hallam-Baker, Rob Stradling, and Jacob Hoffman-Andrews. 2019. DNS Certification Authority Authorization (CAA) Resource Record. RFC 8659. (Nov. 2019). https://doi.org/10.17487/RFC8659
[41] P. Hoffman and P. McManus. 2018. DNS Queries over HTTPS (DoH). IETF RFC 8484. (2018).
[42] P. Hoffman and J. Schlyter. 2012. The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA. IETTF RFC 6698. (2012).
[43] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. Hoffman. [n. d.]. Specification for DNS over Transport Layer Security (TLS). IETF RFC 7858. ([n. d.]).
[44] Mattijs Jonker, Alistair King, Johannes Krupp, Christian Rossow, Anna Sperotto, and Alberto Dainotti. 2017. Millions of targets under attack: a macroscopic characterization of the DoS ecosystem. In *ACM Internet Measurement Conference*.
[45] Athanasios Kountouras, Panagiotis Kintis, Chaz Lever, Yizheng Chen, Yacin Nadji, David Dagon, Manos Antonakakis, and Rodney Joffe. 2016. Enabling network security through active DNS datasets. In *Intl. Symposium on Research in Attacks, Intrusions, and Defenses*.
[46] Brian Krebs. 2019. A Deep Dive on the Recent Widespread DNS Hijacking Attacks. (Feb. 2019). https://krebsonsecurity.com/2019/02/a-deep-dive-on-the-recent-widespread-dns-hijacking-attacks.
[47] NLnet Labs. [n. d.]. Performance Tuning. ([n. d.]). https://unbound.docs.nlnetlabs.nl/en/latest/topics/performance.html.
[48] Rapid 7 Labs. [n. d.]. Open Data. ([n. d.]). https://opendata.rapid7.com.
[49] Jiarun Mao, Michael Rabinovich, and Kyle Schomp. 2022. Assessing Support for DNS-over-TCP in the Wild. In *International Conference on Passive and Active Network Measurement*. Springer, 487–517.
[50] Sourena Maroofi, Maciej Korczynski, and Andrzej Duda. 2020. From defensive registration to subdomain protection: evaluation of email anti-spoofing schemes for high-profile domains. In *Proc. Network Traffic Measurement and Analysis Conference (TMA)*.
[51] Sourena Maroofi, Maciej Korczyński, Arnold Hölzel, and Andrzej Duda. 2021. Adoption of Email Anti-Spoofing Schemes: A Large Scale Analysis. *IEEE Transactions on Network and Service Management* (2021).
[52] Theodore Ian Martiny. 2022. *Privacy in Centralized Systems*. Ph.D. Dissertation. University of Colorado.
[53] Angelique Medina. 2021. Inside the Fastly Outage: Analysis and Lessons Learned. https://www.thousandeyes.com/blog/inside-the-fastly-outage-analysis-and-lessons-learned. (2021).
[54] Ariana Mirian, Christopher Thompson, Stefan Savage, Geoffrey M Voelker, and Adrienne Porter Felt. 2018. HTTPS Adoption in the Longtail. (2018).
[55] P.V Mockapetris. 1987. Domain names: concepts and facilities. IETF RFC 1034. (1987).
[56] Simran Patil and Nikita Borisov. 2019. What can you learn from an IP?. In *Proceedings of the Applied Networking Research Workshop*. 45–51.
[57] Simran Pramod Patil. 2020. *Privacy implications of information leakage from IP addresses-a web fingerprinting approach*. Ph.D. Dissertation.
[58] Paul Pearce, Ben Jones, Frank Li, Roya Ensafi, Nick Feamster, Nick Weaver, and Vern Paxson. 2017. Global measurement of DNS manipulation. In *26th USENIX Security Symposium*.
[59] Reethika Ramesh, Ram Sundara Raman, Matthew Bernhard, Victor Ongkowijaya, Leonid Evdokimov, Anne Edmundson, Steven Sprecher, Muhammad Ikram, and Roya Ensafi. 2020. Decentralized control: A case study of russia. In *Network and Distributed Systems Security (NDSS) Symposium 2020*.
[60] Philipp Richter, Georgios Smaragdakis, David Plonka, and Arthur Berger. 2016. Beyond counting: new perspectives on the active IPv4 address space. In *ACM Internet Measurement Conference*.
[61] Jan Rüth, Ingmar Poese, Christoph Dietzel, and Oliver Hohlfeld. 2018. A First Look at QUIC in the Wild. In *Conf. Passive and Active Network Measurement*.
[62] Quirin Scheitle, Taejoong Chung, Jens Hiller, Oliver Gasser, Johannes Naab, Roland van Rijswijk-Deij, Oliver Hohlfeld, Ralph Holz, Dave Choffnes, Alan Mislove, and Georg Carle. 2018. A First Look at Certification Authority Authorization (CAA). *SIGCOMM Comput. Commun. Rev.* 48, 2 (may 2018), 10–23. https://doi.org/10.1145/3213232.3213235
[63] Raffaele Sommese, Giovane Moura, Mattijs Jonker, Roland van Rijswijk-Deij, Alberto Dainotti, Kimberly C Claffy, and Anna Sperotto. 2020. When parents and children disagree: Diving into DNS delegation inconsistency. In *International Conference on Passive and Active Network Measurement*. Springer, 175–189.
[64] Roland van Rijswijk-Deij, Mattijs Jonker, Anna Sperotto, and Aiko Pras. 2016. A high-performance, scalable infrastructure for large-scale active DNS measurements. *IEEE journal on selected areas in communications* (2016).
[65] Chris Villemez. 2021. AWS Outage Analysis: December 7, 2021. https://www.thousandeyes.com/blog/aws-outage-analysis-dec-7-2021l. (2021).
[66] Gerry Wan, Liz Izhikevich, David Adrian, Katsunari Yoshioka, Ralph Holz, Christian Rossow, and Zakir Durumeric. 2020. On the origin of scanning: The impact of location on Internet-wide scans. In *Proceedings of the ACM Internet Measurement Conference*. 662–679.
[67] Kaishen Wang. 2019. Blacklist filtering for security research: bridging the gap between domain blacklists and malicious web content. (2019).
[68] Zack Whittaker. 2021. A DNS outage just took down a large chunk of the internet. https://techcrunch.com/2021/07/22/a-dns-outage-just-took-down-a-good-chunk-of-the-internet/. (2021).

A EVALUATION DATASET

To evaluate ZDNS, we extract all unique fully qualified domain names found on unexpired, browser-trusted certificates in Censys [28]. The 234M fully-qualified domains (fqdn) found in the Certificate Transparency logs present a good mix of different types of domains: 55% are legacy generic TLDs, 39% are country code TLDs, and 6% are new generic TLDs (Table 3). The 234M fqdns map to 93M base domains. When running a ZDNS scan with default parameters, roughly 70% of the domain names successfully resolve.

	fqdn	domain	tld
legacy gTLDs	129644044	45865899	5
ngTLDs	14228236	6094090	1211
ccTLDs	90659109	41574286	486
All Domains	234531389	93534275	1702

Table 3: Certificate Transparency Domains Dataset—We present the breakdown of domain types used for our evaluations in Sections 4 and 6

B EXAMPLE MODULE

Implementing DoLookup functions and modules is simple. We provide an example DoLookup function in Figure 3, and an example module that performs Sender Policy Framework (SPF) lookups in Figure 4.

```
func (s *Lookup) DoLookup(namestring, nameServer string) (interface{}, zdns.Trace, zdns.Status, error) {
        innerRes, trace, status, err := s.DoMiekgLookup(
                                            miekg.Question{Name: name, Type: s.DNSType, Class: s.DNSClass}
                                           , nameServer)
        resString, resStatus, err := s.CheckTxtRecords(innerRes, status, err)
        res := Result{Spf: resString}
        return res, trace, resStatus, err
}
```

Figure 3: Example DoLookup—Implementing DoLookup function is simple, such as this Sender Policy Framework lookup function.

```
// SPF Module ==================================================================
package spf

import (
    "github.com/zmap/dns"
    "github.com/zmap/zdns/pkg/miekg"
    "github.com/zmap/zdns/pkg/zdns"
    "regexp"
)

const spfPrefixRegexp = "(?i)^v=spf1"

// result to be returned by scan of host
type Result struct {
        Spf string `json:"spf,omitempty" groups:"short,normal,long,trace"`
}

// Per Connection Lookup =======================================================
type Lookup struct {
        Factory *RoutineLookupFactory
        miekg.Lookup
}

func (s *Lookup) DoLookup(name string, nameServer string) (interface{}, zdns.Trace, zdns.Status, error) {
        innerRes, trace, status, err := s.DoMiekgLookup(
                                            miekg.Question{Name: name, Type: s.DNSType, Class: s.DNSClass}
                                           , nameServer)
        resString, resStatus, err := s.CheckTxtRecords(innerRes, status, err)
        res := Result{Spf: resString}
        return res, trace, resStatus, err
}
```

```
// Per GoRoutine Factory ===================================================
type RoutineLookupFactory struct {
        miekg.RoutineLookupFactory
        Factory *GlobalLookupFactory
}

func (rlf *RoutineLookupFactory) MakeLookup() (zdns.Lookup, error) {
        lookup := Lookup{Factory: rlf}
        nameServer := rlf.Factory.RandomNameServer()
        lookup.Initialize(nameServer, dns.TypeTXT, dns.ClassINET, &rlf.RoutineLookupFactory)
        return &lookup, nil
}

func (rlf *RoutineLookupFactory) InitPrefixRegexp() {
        rlf.PrefixRegexp = regexp.MustCompile(spfPrefixRegexp)
}

// Global Factory ==========================================================
type GlobalLookupFactory struct {
        miekg.GlobalLookupFactory
}

func (s *GlobalLookupFactory) MakeRoutineFactory(threadID int) (zdns.RoutineLookupFactory, error) {
        rlf := new(RoutineLookupFactory)
        rlf.RoutineLookupFactory.Factory = &s.GlobalLookupFactory
        rlf.Factory = s
        rlf.InitPrefixRegexp()
        rlf.ThreadID = threadID
        rlf.Initialize(s.GlobalConf)
        return rlf, nil
}

// Global Registration =====================================================
func init() {
        s := new(GlobalLookupFactory)
        zdns.RegisterLookup("SPF", s)
}
```

Figure 4: Example Module—Implementing a module is simple, such as this Sender Policy Framework module.

C EXPOSED LOOKUP CHAIN: DIG VS ZDNS

Released prior to ZDNS, dig [7] is a command-line tool with the ability to "trace" (i.e., expose) the DNS lookup chain. We configure both dig and ZDNS to expose the lookup chain when querying the A record of google.com, and compare their outputs in Figure 5 and Figure 6. ZDNS's output is more programmatically interpretable compared to dig.

```
; <<>> DiG 9.11.3-1ubuntu1.16-Ubuntu <<>> google.com @1.1.1.1 +trace
;; global options: +cmd
.                       517042  IN      NS      a.root-servers.net.
.                       517042  IN      NS      b.root-servers.net.
.                       517042  IN      NS      c.root-servers.net.
.                       517042  IN      NS      d.root-servers.net.
.                       517042  IN      NS      e.root-servers.net.
.                       517042  IN      NS      f.root-servers.net.
.                       517042  IN      NS      g.root-servers.net.
.                       517042  IN      NS      h.root-servers.net.
.                       517042  IN      NS      i.root-servers.net.
.                       517042  IN      NS      j.root-servers.net.
.                       517042  IN      NS      k.root-servers.net.
.                       517042  IN      NS      l.root-servers.net.
.                       517042  IN      NS      m.root-servers.net.
.                       517042  IN      RRSIG   NS 8 0 518400 20220531170000 20220518160000 47671 . QaVW5itNWx...

;; Received 1097 bytes from 1.1.1.1#53(1.1.1.1) in 2 ms

com.                    172800  IN      NS      i.gtld-servers.net.
com.                    172800  IN      NS      f.gtld-servers.net.
com.                    172800  IN      NS      b.gtld-servers.net.
com.                    172800  IN      NS      l.gtld-servers.net.
com.                    172800  IN      NS      d.gtld-servers.net.
com.                    172800  IN      NS      c.gtld-servers.net.
com.                    172800  IN      NS      j.gtld-servers.net.
com.                    172800  IN      NS      e.gtld-servers.net.
com.                    172800  IN      NS      h.gtld-servers.net.
com.                    172800  IN      NS      g.gtld-servers.net.
com.                    172800  IN      NS      a.gtld-servers.net.
com.                    172800  IN      NS      k.gtld-servers.net.
com.                    172800  IN      NS      m.gtld-servers.net.
com.                    86400   IN      DS      30909 8 2 E2D3C916F6DEEAC73294E8268FB5885044A833FC5459588F4A9184CF C41A5766
com.                    86400   IN      RRSIG   DS 8 1 86400 20220531170000 20220518160000 47671 . HBLBAX50zT...

;; Received 1198 bytes from 199.9.14.201#53(b.root-servers.net) in 8 ms

google.com.             172800  IN      NS      ns2.google.com.
google.com.             172800  IN      NS      ns1.google.com.
google.com.             172800  IN      NS      ns3.google.com.
google.com.             172800  IN      NS      ns4.google.com.
CK0POJMG874LJREF7EFN8430QVIT8BSM.com. 86400 IN NSEC3 1 1 0 - CK0Q1GIN43N1ARRC9OSM6QPQR81H5M9A  NS SOA RRSIG DNSKEY NSEC3PARAM
CK0POJMG874LJREF7EFN8430QVIT8BSM.com. 86400 IN RRSIG NSEC3 8 2 86400 20220523042356 20220516031356 37269 com. tnQdPZZqo2...

S84BKCIBC38P58340AKVNFN5KR9O59QC.com. 86400 IN NSEC3 1 1 0 - S84BUO64GQCVN69RJFUO6LVC7FSLUNJ5  NS DS RRSIG
S84BKCIBC38P58340AKVNFN5KR9O59QC.com. 86400 IN RRSIG NSEC3 8 2 86400 20220524051956 20220517040956 37269 com. BQwb7CiufG...

;; Received 836 bytes from 192.35.51.30#53(f.gtld-servers.net) in 60 ms

google.com.             300     IN      A       142.250.188.14
;; Received 55 bytes from 216.239.38.10#53(ns4.google.com) in 48 ms
```

Figure 5: dig +trace Output—When exposing the lookup chain, dig's output is not programmatically interpretable, requiring an additional tool for parsing.

```
{ "class": "IN",
  "data": {
    "answers": [{ "answer": "216.58.195.78", "class": "IN", "name": "google.com", "ttl": 300, "type": "A" }],
    "flags": { "authenticated": false, "authoritative": true, "checking_disabled": false, "error_code": 0, "opcode": 0,
               "recursion_available": false, "recursion_desired": false, "response": true, "truncated": false },
    "protocol": "udp",
    "resolver": "216.239.34.10:53"
  },
  "name": "google.com",
  "status": "NOERROR",
  "timestamp": "2022-05-18T19:19:58Z",
  "trace":[{"cached": false, "class": 1, "depth": 1, "layer": ".", "name": "google.com", "name_server": "199.7.83.42:53",
            "results": {
                "additionals":[
                    {"answer":"192.55.83.30","class":"IN","name":"m.gtld-servers.net","ttl":172800,"type":"A"},
                    {"answer":"192.41.162.30","class":"IN","name":"l.gtld-servers.net","ttl":172800,"type":"A"},
                    {"answer":"192.52.178.30","class":"IN","name":"k.gtld-servers.net","ttl":172800,"type":"A"},
                    ...
                    {"answer":"192.26.92.30","class":"IN","name":"c.gtld-servers.net","ttl":172800,"type":"A"},
                    {"answer":"192.33.14.30","class":"IN","name":"b.gtld-servers.net","ttl":172800,"type":"A"},
                    {"answer":"192.5.6.30","class":"IN","name":"a.gtld-servers.net","ttl":172800,"type":"A"},
                    {"answer":"2001:501:b1f9::30","class":"IN","name":"m.gtld-servers.net","ttl":172800,"type":"AAAA"} ],
                "authorities":[
                    {"answer":"f.gtld-servers.net.","class":"IN","name":"com","ttl":172800,"type":"NS"},
                    {"answer":"d.gtld-servers.net.","class":"IN","name":"com","ttl":172800,"type":"NS"},
                    {"answer":"i.gtld-servers.net.","class":"IN","name":"com","ttl":172800,"type":"NS"},
                    ...
                    {"answer":"g.gtld-servers.net.","class":"IN","name":"com","ttl":172800,"type":"NS"},
                    {"answer":"c.gtld-servers.net.","class":"IN","name":"com","ttl":172800,"type":"NS"},
                    {"answer":"k.gtld-servers.net.","class":"IN","name":"com","ttl":172800,"type":"NS"} ]
                "flags": { "authenticated": false, "authoritative": false, "checking_disabled": false, "error_code": 0, "opcode": 0,
                           "recursion_available": false, "recursion_desired": false, "response": true, "truncated": false},
                "protocol": "udp",
                "resolver": "199.7.83.42:53"},
            "try": 1,
            "type": 1 },
           {"cached": false, "class": 1, "depth": 2, "layer": "com", "name": "google.com", "name_server": "192.5.6.30:53",
            "results": {
                "additionals":[
                    {"answer":"2001:4860:4802:34::a","class":"IN","name":"ns2.google.com","ttl":172800,"type":"AAAA"},
                    {"answer":"216.239.34.10","class":"IN","name":"ns2.google.com","ttl":172800,"type":"A"},
                    {"answer":"2001:4860:4802:32::a","class":"IN","name":"ns1.google.com","ttl":172800,"type":"AAAA"},
                    {"answer":"216.239.32.10","class":"IN","name":"ns1.google.com","ttl":172800,"type":"A"},
                    {"answer":"2001:4860:4802:36::a","class":"IN","name":"ns3.google.com","ttl":172800,"type":"AAAA"},
                    {"answer":"216.239.36.10","class":"IN","name":"ns3.google.com","ttl":172800,"type":"A"},
                    {"answer":"2001:4860:4802:38::a","class":"IN","name":"ns4.google.com","ttl":172800,"type":"AAAA"},
                    {"answer":"216.239.38.10","class":"IN","name":"ns4.google.com","ttl":172800,"type":"A"}],
                "authorities":[
                    {"answer":"ns2.google.com.","class":"IN","name":"google.com","ttl":172800,"type":"NS"},
                    {"answer":"ns1.google.com.","class":"IN","name":"google.com","ttl":172800,"type":"NS"},
                    {"answer":"ns3.google.com.","class":"IN","name":"google.com","ttl":172800,"type":"NS"},
                    {"answer":"ns4.google.com.","class":"IN","name":"google.com","ttl":172800,"type":"NS"}],
                "flags": { "authenticated": false, "authoritative": false, "checking_disabled": false, "error_code": 0, "opcode": 0,
                           "recursion_available": false, "recursion_desired": false,"response": true, "truncated": false },
                "protocol": "udp",
                "resolver": "192.5.6.30:53" },
            "try": 1,
            "type": 1 },
           {"cached": false, "class": 1, "depth": 3, "layer": "google.com", "name":   "google.com",  "name_server": "216.239.34.10:53",
            "results": {
            "answers": [{ "answer": "216.58.195.78", "class": "IN", "name": "google.com", "ttl": 300, "type": "A"}],
            "flags": { "authenticated": false, "authoritative": true, "checking_disabled": false, "error_code": 0, "opcode": 0,
                   "recursion_available": false, "recursion_desired": false, "response": true, "truncated": false },
            "protocol": "udp",
            "resolver": "216.239.34.10:53" },
            "try": 1,
            "type": 1
            }]
}
```


Figure 6: ZDNS `+trace` Output—When exposing the lookup chain, ZDNS' output is in JSON, which is programmatically interpretable.