Gotta Query 'Em All, Again! Repeatable Name Resolution with Full Dependency Provenance

Johannes Naab
naab@net.in.tum.de
Technical University of Munich
Germany

Patrick Sattler
sattler@net.in.tum.de
Technical University of Munich
Germany

Johannes Zirngibl
zirngibl@net.in.tum.de
Technical University of Munich
Germany

Stephan Günther
guenther@tum.de
Technical University of Munich
Germany

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

ABSTRACT

Common DNS resolvers are optimized for query latency but are not designed to expose the internal dependencies and structures within the DNS. This makes it difficult to investigate DNS setups, detect errors and misconfigurations, and determine their impact on users.

In order to reliably track the internal, potentially cyclic dependencies within the DNS, we propose to split the dependency graph into strongly connected components. By querying all authoritative servers and considering differences in order and timing for repeated runs, we are able to resolve domain names in a repeatable and traceable manner. We validate this approach by introducing a test methodology that allows re-running the resolver against previously recorded data. This data can be used to further study various aspects of global DNS deployments. We provide an example scan with 1.6 M domains on https://tcb-resolve.github.io/.

CCS CONCEPTS

• **Networks** → **Naming and addressing**; **Network measurement**; *Logical / virtual topologies.*

KEYWORDS

DNS, Domain Name System, Resolver, Dependency Graph, Internet Measurement

ACM Reference Format:
Johannes Naab, Patrick Sattler, Johannes Zirngibl, Stephan Günther, and Georg Carle. 2023. Gotta Query 'Em All, Again! Repeatable Name Resolution with Full Dependency Provenance. In *Applied Networking Research Workshop (ANRW '23), July 22–28, 2023, San Francisco, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3606464.3606478

ANRW '23, July 22–28, 2023, San Francisco, CA, USA

ACM ISBN 979-8-4007-0274-7/23/07…$15.00
https://doi.org/10.1145/3606464.3606478

1 INTRODUCTION

The DNS is essential to the functioning of the Internet. Users mostly rely on standard resolvers providing an easy interface to resolve domain names. The full inner workings of the DNS and their potential impact on the name resolution are hidden behind this interface. Resolvers commonly try to optimize the end-to-end latency for queries and assume properly configured zones. Caching and multiple authoritative servers help to ensure availability.

However, the same features complicate debugging DNS setups and errors. A large part of the complexity in investigating those setups stems from the configuration of authoritative servers as FQDNs, which themselves need to be resolved using DNS. Resolvers need to rely on the root hints, glue data, and caches to overcome this chicken-and-egg problem. Multiple authoritative servers and the need to synchronize a zone's `NS` records with its parent introduce multiple data copies. While those copies should match each other, in practice there will be some deviations. To reliably investigate DNS setups, errors, misconfiguration, and their impact on users, it is necessary to consider the entire data set.

In this paper, we present an approach to resolve domain names that takes their entire dependency graph into account. By re-running the resolution against previously recorded data, we can ensure that we achieve a consistent result and do not omit any results that potentially influence the resolution. Our proposed approach achieves the following goals:

- *(i)* It discovers all reasonable resolution paths for a domain. By providing provenance of the resolution, we can later investigate why certain queries were made.
- *(ii)* It checks the different copies of the (same) data within the DNS. Therefore, we always query all authoritative servers within each zone and resolve glue records with authoritative data.
- *(iii)* This allows the resolution process to be verified using a deterministic and repeatable set of queries.
- *(iv)* The approach is designed to be fair and efficient. We only issue queries that are likely going to be answered, use rate limiting, and take special care to avoid impacting central services.

The paper is structured as follows: Section 2 gives a motivating example and shows how strongly connected components can be used to structure the resolution. The name resolution process is described in Section 3. The testing and validation of our approach

is shown in Section 4. We conclude by investigating related work and use cases in Section 5.

2 BACKGROUND

We follow the naming conventions of RFC 8499. DNS data is structured into zones hosted on authoritative servers. As an example, we consider the zone configurations necessary to resolve `tum.de` as shown in Figure 1. A zone configures its authoritative servers with `NS` records pointing to domain names (1). These `NS` records add a dependency to the zone authoritative for that name. A zone is reached by a delegation configured in its parent (2), which is in turn another dependency of the zone. Those FQDNs need to be resolved in their respective zones (3). If the NS records in a delegation point into their own zone (4), glue records are necessary (5).

If we keep following those dependencies, we end up with the dependency graph shown in Figure 2. The entire set of dependencies (including the individual servers which are omitted in the figure) is called the trusted computing base (TCB). In the dependency graph we find certain sets of (indirectly) interdependent zones. One such set in our example is `lrz.de`, `lrz.bayern`, and `lrz.eu`. A set of interdependent zones represent a strongly connected component (SCC).

The meaning of an SCC within the DNS is that specific query results within one zone context can (indirectly) impact the results for any other zone within the SCC, including itself. Queries that impact the SCC are the discovery of authoritative servers in `NS` records and the corresponding address resolution records. For instance, by resolving the `A` records for all known authoritative servers in a zone, we potentially discover a new IP address that needs to be added to the set of authoritative servers in those zones. If this new authoritative server has, for instance, an outdated copy of the zone data, it can introduce further dependencies into the graph. Using these SCCs, we can structure the resolution process so that no dependency remains hidden.

Beyond the direct dependencies for `tum.de`, Figure 2 shows that the SCC for the root zone additionally contains the `net` and `com` zone as well as `root-servers.net`, `gtld-servers.net` and `nstld.com` hosting the respective name servers. The `de` zone has `NS` dependencies to `nic.de` and `de.net`. These are hosted by `denic.de` and `denic.net`, with DENIC being the registrar for `de`. The `bayern` TLD invokes a longer dependency chain with its direct name servers in `irondns.net`. The name servers for `irondns.net` are within their own zone as well as `irondns.de`. The name servers for `irondns.de` are hosted by `knipp.de` and `knipp.net`. With IronDNS being a product of Knipp[1], there appears some organizational overlap that is not visible within the SCC graph. Compared to, *e.g.* `eu`, which hosts its name servers directly, `bayern` pulls in additional indirect dependencies that can potentially influence the name resolution. The SCCs provide a way to identify which zones belong together and help to pinpoint the responsible dependencies.

3 APPROACH

To investigate how to reliably resolve a zone's TCB, we developed a custom resolver. This resolver creates an internal representation of the discovered zone tree. Each zone has two primary purposes. In the bootstrap process, the resolver reliably identifies the addresses of the authoritative servers. After the authoritative servers have been discovered, the zone processes queries as necessitated by the inputs or by internal resolution requests. To ensure a consistent view of the discovered zones, we delay queries within each zone until the bootstrap process has been completed and all authoritative servers have been discovered and validated. Queries are only resolved during the bootstrap process if they can directly or indirectly affect the setup of the zone.

The resolver uses the following inputs: the preprocessed input list, the root hints, and an exclusion list containing domains and networks. For each input domain, the resolver queries all authoritative servers and saves the responses. Multiple domains are processed concurrently. Queries necessary for the resolution itself (`NS` FQDNs) are resolved internally while considering cyclic dependencies. The queries and dependencies used by the internal resolution steps are stored for further analysis.

We structure the resolution process as follows. Domains to be queried are collected and preprocessed. The root zone file is used to filter domains that are known not to exist. Zone file copies can be used to enrich the query list. This can help to reduce the query load against central zones (cf. Section 3.5.2).

3.1 Setup of a Zone within the Resolver

The candidates for authoritative servers are learned from

- glue records in the delegation discovered in the parent zone,
- internal resolution of the address records for `NS` names in the delegation,
- internal resolution of the address records for `NS` records queried against the authoritative servers,
- *inherited* authoritative servers of the parent if the delegation query provided an authoritative answer, and
- for the root zone, the root hints as the safety belt.

For each candidate address, the `SOA` and `NS` records for the zone apex are queried. `SOA` records *should* exist in common setups and are recorded for later analysis. `NS` records are necessary to discover all reachable authoritative server names and side-way delegations. The candidate is accepted as *authoritative* server address for the zone if any of these queries returns an authoritative answer. The candidate server address is considered *broken* within the zone's context if no answer or a non-authoritative answer is returned. It will no longer be queried within the zone's context in this case. Candidates can be learned from multiple sources and are processed independently of how they were learned.

The internal address resolution of `NS` FQDNs can introduce cyclic dependencies. The zone tracks the issued address resolution queries in order to help determine the cyclic dependences. It keeps track of the query progress and in which zone the query is processed.

Query requests received during the bootstrap process are queued. The requesting zone is notified (*scc wait*) that the query is currently on hold pending the completion of the zone setup. This is necessary to facilitate the search for SCCs.

Once the `SOA` and `NS` records for the candidate name servers have been processed and all tracked queries of the `NS` FQDNs are stuck in the *scc wait* state or completed, a search for the cyclic dependencies

[1] https://www.knipp.de/it-es/irondns?set-language=en

```
$ORIGIN tum.de.
@ NS dns1.lrz.de.     # (1)
@ NS dns2.lrz.bayern. # (1)
@ NS dns3.lrz.eu.     # (1)
```

(a) tum.de config

```
$ORIGIN de.
tum NS dns1.lrz.de.     # (2)
tum NS dns2.lrz.bayern. # (2)
tum NS dns3.lrz.eu.     # (2)
lrz NS dns1.lrz.de.     # (4)
lrz NS dns2.lrz.bayern.
lrz NS dns3.lrz.eu.
dns1.lrz A 129.187.19.183 # (5)
```

(b) .de config

```
$ORIGIN lrz.de.
@ NS dns1.lrz.de.
@ NS dns2.lrz.bayern.
@ NS dns3.lrz.eu.
dns1 A 129.187.19.183 # (3)
```

(c) lrz.de config

Figure 1: Simplified configuration of zones to resolve `tum.de`.

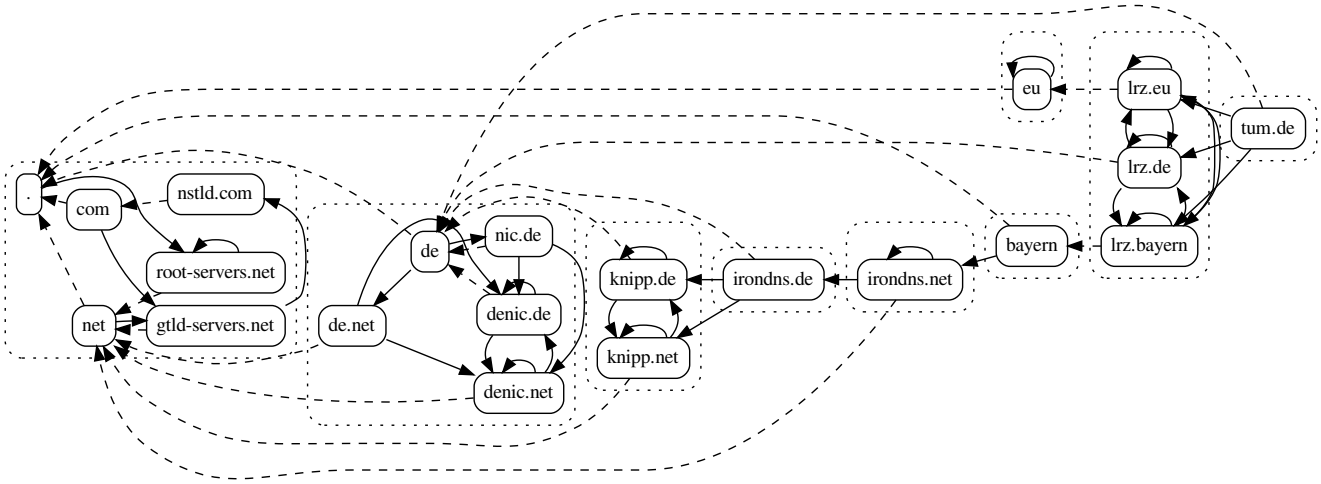

Figure 2: Dependency graph for `tum.de` as of 2023-05-10. Solid lines show `NS` dependencies, dashed lines the dependency on the parent zone, dotted boxes the resulting SCCs. The figure omits the specific authoritative servers and address records.

is triggered. For this, we consider the dependency graph within the known zones.

3.2 Strongly Connected Component Search

In our graph representation, zones are vertices. Internal queries to resolve `NS` FQDNs pointing to the zone that it is processed in and the dependency on the parent zone are directed edges. To detect cyclic dependencies, we start a strongly connected component search. The vertices (zones) considered during that search can be in the following states:

- *actively running*: waiting on queries for `SOA` or `NS` against candidate name servers or waiting for status updates of address resolution records.
- *scc wait*: address queries for `NS` FQDNs are stuck in another zone.
- *locally done*: `SOA` and `NS` for the candidate name servers are completed, the address queries for the `NS` FQDNs are provisionally marked as done. The zone will not gain any new authoritative server unless an address query gets updated. Changes (falling back to the *actively running* state) are possible if new results for address queries are received.
- *scc done*: the zone is already completely set up and has been assigned to an SCC.

Using Tarjan's algorithm [13], SCCs are discovered in their reverse topological sort order. The discovered components always start at the root zone. The SCC search is improved by pruning zones that are already completely set up (*scc done*). Those are part of previously discovered and finalized SCCs. The search is aborted if a visited zone is in the *actively running* state. The first discovered component (possibly a single zone) within this search iteration contains either all *locally done* zones or at least one zone stuck in the *scc wait* state. In case all zones in the discovered component are *locally done*, we discovered a new SCC. The zones in the component are notified, they switch into the *scc done* state, and start processing queries.

If any zone in the discovered component is in the *scc wait* state, a new cyclic dependency has been discovered. The queries spanning this dependency cycle need to be processed even if the responsible zone has not yet reached the *scc done* state. The zones involved in the identified component are searched for stuck *scc wait* queries originally requested by zones within the found SCC. These queries are then processed even though the response zone is in the *scc wait* or *locally done* state. Once such a query is resolved against the known authoritative servers, the requesting zones are notified with the query results. If a zone received the results for all `NS` FQDNs, it enters the *locally done* state and triggers a new search. Once all

necessary dependencies have been discovered and dealt with, the SCC search finalizes the component.

3.3 Deterministic Zone Cut Detection

In order to capture comprehensive and consistent information about the DNS infrastructure and to reliably perform the SCC search, we need to identify individual zones. We detect the zones by querying the potential zone cuts. Every label in a FQDN is the location of a potential zone cut. QNAME minimization [2, 3] offers an approach to discover zone cuts. We adopt the QNAME minimization approach with minor modifications.

We query `SOA` records because using `A` records [3] would hide the zone cut in case the authoritative server is also authoritative for the child zone. A `NOERROR` response is not enough to differentiate between a child zone and a name within the zone itself. The existence of a `SOA` record at the child name is a strong indicator for a separate child zone. Positive `SOA` responses can be reused during the zone's bootstrap process. `NS` queries [2] would also indicate a child zone. During initial attempts we observed broken authoritative servers that, when queried for `NS`, provided the delegation records in the answer section (while it should have been in the authority section).

The left most label is assumed to be within the enclosing zone (*e.g.* `www.example.org` is assumed to be within the `example.org` zone and does not trigger explicit delegation discovery). In case the answers to a query indicate a zone cut by a delegation response or by existence of a matching `SOA` record in the authority section (`NODATA`, the authoritative server is authoritative for the child zone as well), the zone is explicitly set up. An exception is made for delegation-centric zones, *i.e.*, TLDs and country code second level domains such as `co.uk`. These trigger `SOA` queries to discover potential delegations first, even for full domains such as `example.co.uk`.

All authoritative servers in the respective zone are queried. For the root zone and TLD zones this is further optimized to only query a subset of authoritative servers as described in Section 3.5.1. If any response indicates a delegation, an explicit child zone is set up by the resolver.

To ensure a consistent view on the discovered zone cuts, two additional details are taken into account. Different query types are grouped together. If `www.example.org` is queried for `A` and `AAAA`, a `NODATA` response for one of them could indicate a separate child zone hosted on the same authoritative servers. In this case, all query types for this name (here `A` and `AAAA`) are moved into the newly discovered child zone. Furthermore, the decision on whether to (initially) query directly within a given zone or to query for zone cuts is made independently of the already known child zones. `foo.example.org` will always be queried within `example.org` and potentially be moved to the child zone even if due to a previous query for `bar.foo.example.org` the child zone has already been discovered. Both considerations are necessary to allow for differences in order and timing of queries.

3.4 Network Interaction

Since our resolver queries all authoritative servers, it potentially creates significantly more load than regular resolvers. We try to minimize the impact this resolver has on the authoritative server infrastructure. By employing a per IP address rate limit (50 queries/s) and limiting the number of outstanding queries (100 queries), we reduce the chance of negatively impacting any authoritative server. Queries are retried up to 2 times (3 queries in total) with increasing timeouts. Should an address be unresponsive for 250 consecutive queries (excluding retries, at 50 queries/s this matches only 5 batches), the address is marked as unresponsive and will no longer be queried during this run. Additionally, we rely on global rate limits (24 k queries/s) and a limit for outstanding queries (12 k) to reduce the impact on the local infrastructure. The work queue, *i.e.*, all pending domains to be resolved, is limited to 128 k queries. In combination with randomizing the query list itself, this should avoid any hot spots on specific servers and smooths out the queries against individual servers over the entire run. The resolver does not send queries to any non-public IP addresses by default, and allows to exclude specific domains, IP addresses, or networks by use of a filter list. Queries are cached in the individual zones. Zones with their queries expire after 5 min of inactivity. The resolver sends specific queries only to addresses for which a public resolution path exists. It does not try to guess name servers or brute force any queries. Name server operators should not see any strange or misdirected queries.

3.5 Optimization for Query Efficiency

When setting up a child, the resolver queries all authoritative servers in the enclosing zone to determine the delegation status. Scans targeting distinct 2nd level domains trigger a `SOA` query for each domain against every authoritative server in the TLD zone. The `.com` zone is served by 26 authoritative servers. Each new 2nd level domain would thus trigger 26 queries. This can lead to the following problems: due to applying a per address query rate limit, the entire resolution run would be delayed by these centralized authoritative servers. A rate limit of 50 queries/s would prolong the scanning of 1 M `.com` domains to 5.5 h. Increasing the rate limit for specific addresses increases the risk of causing load spikes or additional work load for the operators to investigate those queries. We mitigate this problem in two ways: first by querying only a subset of authoritative servers in the root and TLD zones, second by using upstream zone data to skip querying those authoritative servers for most domains.

3.5.1 Querying a subset of authoritative servers. We opportunistically assume that the servers authoritative for the root and TLD zones are consistent within their zone and can thus reduce the query load by using only a subset of authoritative servers. To select this subset, the following requirements must be met: the queries should be evenly distributed among the authoritative servers, and the selection of the subset must be repeatable in order to support deterministically re-running the resolver. For each child to be processed, we order the authoritative servers by hashing the domain name concatenated with the IP address of the authoritative server (optionally including a per-run key). We use the first three authoritative servers. Deterministically picking the authoritative servers allows us to re-run against the recorded data and make the same decision the second time around. By picking at least two authoritative servers, we can (over enough domains) discover inconsistencies among the authoritative servers. This reduces the query load against

the .com authoritative servers to 3/26, or 38 min for 1 M .com domains at 50 queries/s. If the three responses are inconsistent or indicate that at least one authoritative server is authoritative for the child as well, all authoritative servers are queried.

This optimization is skipped if the delegation is queried during zone bootstrap to complete a SCC. This is done to gain independence on the order of when and how the authoritative servers are discovered and to guarantee that the discovered SCCs are complete.

3.5.2 Injecting Zone Data into the Resolver. The query load against large TLDs can be further reduced by injecting the zone data, *e.g.* from CZDS [5], directly into the resolution process. This is achieved by enriching the query list with the delegation and glue records as given by the zone file. This assumes that the zone data is recent and (in general) changes infrequently. If the zone apex records differ from the injected data or otherwise indicate failures, the injected data might be outdated. In such a case, the resolver additionally queries the delegation using the parent zone's authoritative servers (while being subject to the subset selection in Section 3.5.1[2]). Zones set up using injected data are only used to resolve queries from the input list. If the same query needs to be resolved within the context of the resolver itself, *i.e.*, address records to resolve NS FQDNs, the zone set up by injected data is ignored. A copy of the zone is set up using only actively queried data. This ensures that the internal queries are explicitly available even if order, timings, or inputs change. It also ensures that undetected but outdated data only affects the queries as given by the input list and cannot impact the SCC setup or indirectly dependent queries.

4 TESTING AND EVALUATION

To test the consistency and reliability of our implementation, we need to execute it multiple times and compare the results. However, the global DNS is not static and changes over time. Changes in network conditions can impact repeated runs as well. Running in-development versions repeatedly against the same infrastructure increases the burden on servers.

We avoid these problems by building a tool called speedbag. Figure 3 gives an architectural overview. Speedbag simulates authoritative name servers by echoing previously saved responses upon request. It relies on Linux network namespaces to emulate name servers for an unmodified resolver. UDP responses are injected by use of a tun device, TCP connections are handled by use of the iptables DNAT target. On startup, speedbag reads the queries recorded in a previous resolution run. It creates a lookup structure mapping (server ip, protocol, qname, qclass, qtype) to (status, response data, response delay).

Upon receiving a query, the request is decoded and the response is looked up. If a response is found, it is injected after the given delay in a way that the resolver sees it as coming from the original address. Should the previously recorded response indicate a timeout, network error, or if no response is found at all, no response is injected. Independent of the lookup outcome, the occurrence of the query is recorded for later analysis.

During shutdown, the observed queries and known responses are analyzed. In the optimal case, all observed queries are known,

[2]An inconsistent apex does not imply an inconsistent/outdated delegation.

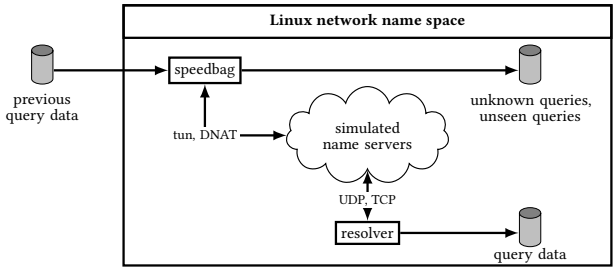

Figure 3: Stand-alone name server simulation and resolver testing architecture speedbag.

and all known responses are queried. If that is not the case, two categories of discrepancies can be found. Seen queries without a known response are *unknown queries*, known responses without corresponding query are *unqueried*. Some discrepancies can be accounted for if the resolution run used a different query list, deployed an updated resolver, or changed the filter list. Other discrepancies can be explained by inconsistent input data: if a query was executed twice in the original dataset, *i.e.*, due to cache expiration, and the subsequent response differs from the initial response, speedbag will always respond with the initial query data. This can have a knock on effect on downstream name resolution.

In principle, speedbag can be applied to other resolver implementations. The data set used must contain the queries that can be issued by the specific resolver. Some extensions can be made, *e.g.* to synthesize delegation within a zone's scope using the available data.

4.1 Repeatability Verification

Investigating individual discrepancies can be time-consuming. Therefore, we extended the testing procedure to automatically filter out repeatable discrepancies. For this, we run the resolver thrice in the speedbag environment as shown in Figure 4 — twice against the Internet resolved data set and once against a speedbag resolved data set. The verification of the resolution process uses the *unknown query* and *unqueried* lists generated by speedbag. Both runs against the Internet resolved data set (run #1, run #2) must be consistent, *i.e.*, result in the same *unknown queries* and *unqueried* queries. This filters out explainable inconsistencies. The follow-up resolution run against the speedbag resolved dataset (run #3) must now be consistent with the input, *i.e.*, it does not result in any *unknown queries* or *unqueried* queries. The consistency goals can be achieved even if non-functional parameters such as order of requests, rate limits, or logging, are modified or updated.

4.2 Timeouts and Unresponsive Servers

The resolver detects unresponsive name servers based on consecutive timeouts. If an unresponsive server is detected, it is no longer queried. The order in which the queries are sent is not fixed. Speedbag cannot see the queries that the resolver completes internally due to unresponsiveness. Without specifically considering timeouts and unresponsive name servers, the consistency checks would always show some discrepancies based on differences in order and timing.

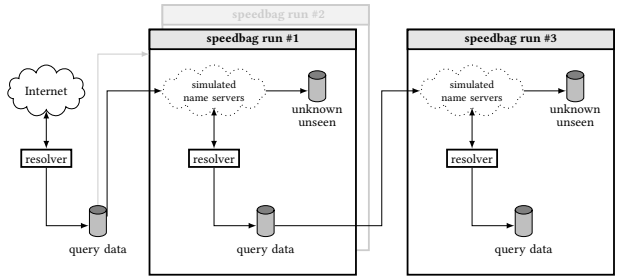

Figure 4: Consistency validation of the resolution process with three speedbag runs.

To mitigate these problems, we adapt both the resolver and speedbag. In speedbag, the known queries (these include those seen on the network as well as those internally generated by the resolver) for each IP address are analyzed. If the address was, at any point, marked as unresponsive during the original resolution, speedbag marks it as such. In order to gain independence from ordering, it does not respond to queries on this address even if they might exist. Additionally, if the original resolution observed many intermittent timeouts but never consistently enough to mark an address as unresponsive, speedbag still marks it as such and does not respond to any query at all. Queries seen for such servers are neither accounted for in *unknown queries* nor as *unqueried*. The resolver is updated to omit marking addresses as unresponsive during the speedbag runs if it receives at least one non-timeout answer. This allows the resolver to query *unknown queries*, *e.g.* for functional updates, without impacting known queries by the responsiveness check. To account for those *unknown queries* during run #3, the filtering of addresses with many intermittent timeouts is omitted. An address is considered responsive if at least one non-timeout query is known. Unresponsive addresses were already filtered in run #1.

5 RELATED WORK

Previous use cases: Ramasubramanian and Sirer [10] introduced the concept of TCBs in the realm of DNS. They found numerous authoritative servers inside the scanned domains' TCB and showed that 30 % have transitive relationships to vulnerable name servers. However, these results from 2005 are not applicable to today's ecosystem. As their scanning methodology is not published, a comparison to our approach is not possible. Osterweil et al. [8] posed the question of balancing between resilience, overhead and attack surface. The availability and robustness implications in the dependency graph are analyzed by Pappas et al. [9] and Deccio et al. [4]. Vissers et al. [16] analyzed the possibility of typo- and bitsquatting cases. The TCB is not fully resolved, as we suggest. Instead, they rely on information contained in zone files. Our approach enables a structured analysis on the full TCB. Obtaining information on the full TCB is also useful for determining whether a domain can be resolved using only IPv6. Streibelt et al. [12] implemented a version of this full resolution, but it took more than four days to query 476 k zones. We provide detailed explanations of our approach and are able to resolve 1.67 M zones in 2.1 h. Recent studies by Li et al. [7], Akiwate et al. [1] and Sommese et al. [11] are examples where existing problems in the zone configuration and protocol deployment have been found. Therefore, a consistent and reliable approach to monitoring and scanning the entire TCB is a valuable tool for detecting and analyzing such problems.

Existing Tools: Existing tools do not provide the full feature set needed to fulfill our requirements. `dig +trace` shows the resolution path a resolver might take, but without examining the full TCB. Similar, large-scale DNS resolution tools, such as ZDNS [6], only mimic a typical resolver behavior. ZDNS does not yet support querying IPv6 name servers. Transitive Trust Checker [15] uncovers the entire TCB but is limited to individual domains. Zhu and Heidemann [17] provided a role model for our speedbag approach. They implemented a DNS resolver suitable for large scale measurements and a replay tool that provides correct timing and inter-arrival times. Existing DNS data collection efforts, *e.g.* OpenIntel [14], only store the collected resource records but do not collect the information to build a name's TCB.

6 CONCLUSION

In this paper, we presented our approach to querying domain names in a repeatable and consistent manner while uncovering their entire dependency graph. This allows tracking internal dependencies and structures of the DNS as well as detecting errors and misconfigurations. By identifying strongly connected components during the resolution, we ensure that no hidden dependencies remain. Rerunning our implementation against recorded query data enables us to validate our implementation and show that it is deterministic and repeatable.

We want to further develop this approach to reliably study and classify DNS setups. The data will allow us to investigate and quantify specific misconfigurations.

EXAMPLE DATA SET

On https://tcb-resolve.github.io/ we provide the results for a scan of 1.6 M with documentation of the result format. It includes the speedbag validation runs.

The scan was executed in May 2023. The domains are sourced from the Alexa Top 1M and the Majestic Million lists. The scan queried the `A`, `AAAA`, `CAA`, `MX` and `TXT` records. Plausible `www` subdomains have been queried for `A` and `AAAA`. The resolution took 2.1 h and issued 118 M queries in 1.67 M zones against 254 k addresses.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their valuable feedback. This work was partially funded by the German Federal Ministry of Education and Research under the project PRIMEnet (16KIS1370), 6G-life (16KISK002) and 6G-ANNA (16KISK107) as well as the German Research Foundation (HyperNIC, grant no. CA595/13-1). Additionally, we received funding by the Bavarian Ministry of Economic Affairs, Regional Development and Energy as part of the project 6G Future Lab Bavaria and the European Union's Horizon 2020 research and innovation program (grant agreement no. 101008468 and 101079774).

REFERENCES

[1] 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 *Proc. ACM Int. Measurement Conference (IMC)* (Virtual Event, USA) *(IMC '20)*. Association for Computing Machinery. https://doi.org/10.1145/3419394.3423623

[2] Stéphane Bortzmeyer. 2016. DNS Query Name Minimisation to Improve Privacy. RFC 7816. https://doi.org/10.17487/RFC7816

[3] Stéphane Bortzmeyer, Ralph Dolmans, and Paul E. Hoffman. 2021. DNS Query Name Minimisation to Improve Privacy. RFC 9156. https://doi.org/10.17487/RFC9156

[4] Casey Deccio, Jeff Sedayao, Krishna Kant, and Prasant Mohapatra. 2010. Measuring Availability in the Domain Name System. In *Proc. IEEE Int. Conference on Computer Communications (INFOCOM)*. IEEE.

[5] Internet Corporation for Assigned Names and Numbers. [n.d.]. Centralized Zone Data Service. https://czds.icann.org/home

[6] 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 *Proc. ACM Int. Measurement Conference (IMC)* (Nice, France) *(IMC '22)*. Association for Computing Machinery. https://doi.org/10.1145/3517745.3561434

[7] Xiang Li, Baojun Liu, Xuesong Bai, Mingming Zhang, Qifan Zhang, Zhou Li, Haixin Duan, and Qi Li. 2023. Ghost Domain Reloaded: Vulnerable Links in Domain Name Delegation and Revocation. In *Proc. Network and Distributed System Security Symposium (NDSS)*. Internet Society, San Diego, CA, USA. https://doi.org/10.14722/ndss.2023.23005

[8] Eric Osterweil, Danny McPherson, and Lixia Zhang. 2011. Operational Implications of the DNS Control Plane. *IEEE Reliability Society Newsletter* (2011).

[9] Vasileios Pappas, Zhiguo Xu, Songwu Lu, Daniel Massey, Andreas Terzis, and Lixia Zhang. 2004. Impact of Configuration Errors on DNS Robustness. In *Proc. ACM SIGCOMM*.

[10] Venugopalan Ramasubramanian and Emin Gün Sirer. 2005. Perils of Transitive Trust in the Domain Name System. In *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement* (Berkeley, CA) *(IMC '05)*. USENIX Association, USA.

[11] Raffaele Sommese, Giovane C. M. Moura, Mattijs Jonker, Roland van Rijswijk-Deij, Alberto Dainotti, K. C. Claffy, and Anna Sperotto. 2020. When Parents and Children Disagree: Diving into DNS Delegation Inconsistency. In *Proc. Passive and Active Measurement (PAM)*.

[12] Florian Streibelt, Patrick Sattler, Franziska Lichtblau, Carlos H. Gañán, Anja Feldmann, Oliver Gasser, and Tobias Fiebig. 2023. How Ready is DNS for an IPv6-Only World?. In *Proc. Passive and Active Measurement (PAM)*, Anna Brunstrom, Marcel Flores, and Marco Fiore (Eds.). Springer Nature Switzerland.

[13] Robert Tarjan. 1972. Depth-First Search and Linear Graph Algorithms. *SIAM J. Comput.* 1, 2 (1972). https://doi.org/10.1137/0201010

[14] 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* 34, 6 (2016). https://doi.org/10.1109/JSAC.2016.2558918

[15] Verisign. [n.d.]. Transitive Trust Checker. https://trans-trust.verisignlabs.com/

[16] Thomas Vissers, Timothy Barron, Tom Van Goethem, Wouter Joosen, and Nick Nikiforakis. 2017. The Wolf of Name Street: Hijacking Domains Through Their Nameservers. In *Proc. ACM SIGSAC Conference on Computer and Communications Security (CCS)* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery. https://doi.org/10.1145/3133956.3133988

[17] Liang Zhu and John Heidemann. 2018. LDplayer: DNS Experimentation at Scale. In *Proc. ACM Int. Measurement Conference (IMC)* (Boston, MA, USA) *(IMC '18)*. Association for Computing Machinery. https://doi.org/10.1145/3278532.3278544