When parents and children disagree: Diving into DNS delegation inconsistency

Raffaele Sommese[1], Giovane C.M. Moura[2], Mattijs Jonker[1], Roland van Rijswijk-Deij[1,3], Alberto Dainotti[4], K.C. Claffy[4], and Anna Sperotto[1]

[1] University of Twente
[2] SIDN Labs
[3] NLnet Labs
[4] CAIDA

Abstract. The Domain Name System (DNS) is a hierarchical, decentralized, and distributed database. A key mechanism that enables the DNS to be hierarchical and distributed is *delegation* [7] of responsibility from parent to child *zones*—typically managed by different entities. RFC1034 [12] states that authoritative nameserver (NS) records at both parent and child should be "consistent and remain so", but we find inconsistencies for over 13M second-level domains. We classify the type of inconsistencies we observe, and the behavior of resolvers in the face of such inconsistencies, using RIPE Atlas to probe our experimental domain configured for different scenarios. Our results underline the risk such inconsistencies pose to the availability of misconfigured domains.

1 Introduction

The Domain Name System (DNS) [12] is one of the most critical components of the Internet, used by virtually every user and application. DNS is a distributed, hierarchical database that maps hosts, services and applications to IP addresses and various other types of records. A key mechanism that enables the DNS to be hierarchical and distributed is *delegation* [7]. In order for delegation to work, the DNS hierarchy is organized in parent and child *zones*—typically managed by different entities—that need to share common information (NS records) about which are the authoritative name servers for a given domain. While RFC1034 [12] states that the NS records at both parent and child should be "consistent and remain so", there is evidence that this is not always the case [10]. However, a full and systematic analysis of the extent of this problem is still missing.

In this paper, we analyze this issue by *(i)* providing a broad characterization of inconsistencies in DNS delegations, and *(ii)* investigating and shedding light on their practical consequences. Specifically, we first evaluate if there are inconsistencies between parent and child sets of NS records (NSSet) for all active second-level domain names of three large DNS zones: `.com`, `.net`, and `.org` (§3)—together comprising of more than 166M domain names (50% of the DNS namespace), as well as all top-level domains (TLDs) from the Root DNS zone [22]. We show that while 80% of these domain names exhibit consistency, 8% (i.e., 13 million domains) do not. These inconsistencies affect even large and popular organizations, including Twitter, Intel and AT&T. Overall we find that at least 50k `.com`, `.net`, and `.org` domains of the Alexa Top 1M list are affected.

We then classify these inconsistencies into four categories (§3): the cases (i) in which the parent and child NSSets are *disjoint* sets, (ii) the parent NSSet is a *subset* of the child NSSet, (iii) the parent NSSet is a *superset* of the child NSSet and (iv) the parent and child NSSet have a non-empty intersection but do not match (ii) or (iii). These inconsistencies are not without harm. Even in the case in which disjoint sets of NS records resolve to the same IP addresses, case (i) introduces fragility in the DNS infrastructure, since operators need to maintain different information at different levels of the DNS hierarchy, which are typically under separate administrative control. Case (ii) may lead to unresponsive name servers, while case (iii) points to a quite understandable error of modifying the child zone while forgetting the parent, but it offers a false sense of resilience and it results in improper load balancing among the name servers. Finally, case (iv), which we see happening in more than 10% of the cases in which parent and child have a non-empty intersection, suffers all the aforementioned risks.

To understand the practical consequences of such inconsistencies, we emulate all four categories (§4) by setting up a test domain name and issuing DNS queries from more than 15k vantage points. Our experiment highlights the consequences of delegation inconsistency on query load distribution in the wild. We then investigate how popular DNS resolvers from different vendors deal with such inconsistencies (§5), and find that some resolvers do not comply with RFC specifications.

Finally, we conclude the paper discussing our findings and offering recommendations for domain name operators to manage the inconsistencies we identified.

2 Background and Related Work

DNS uses a *hierarchical name space* [12], in which the root node is the *dot* (`.`). Zones under the root—the top-level domains such as `.org`— are referred to as *delegations* [7]. These delegations have second-level delegations of their own such as `example.org`. To create delegations for a *child* zone (such as `example.org`), DNS NS records [12] are added to the *parent* zone (`.org` in Figure 1). In this example, the NS records are `[a,b].iana-servers.net`, which, in practice, means that these records are the *authoritative* name servers for `example.org`, *i.e.,* servers that have definitive information about the `example.org` zone.

RFC1034 states that the NSSet should be consistent between parent and child authoritative servers. This, however, is far from trivial. Parent and child zones and servers are almost always maintained by different organizations across administrative boundaries. The most common case is where the parent is a TLD. Delegation changes in the parent go through the so-called Registry-Registrar-Registrant (RRR) channel for almost all TLDs. In this model, the Registry operates the TLD, the Registrar sells domain names under the TLD and the Registrant is the domain holder. If the domain holder wants to change the dele-

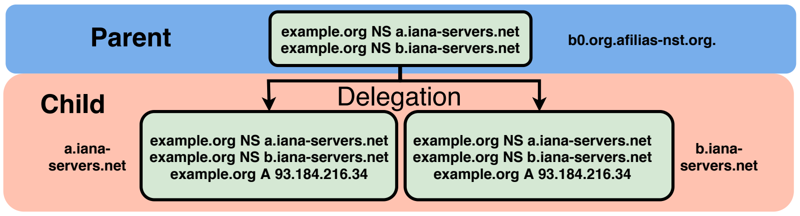

Parent
example.org NS a.iana-servers.net
example.org NS b.iana-servers.net
b0.org.afilias-nst.org.
Child
Delegation
a.iana-servers.net
example.org NS a.iana-servers.net
example.org NS b.iana-servers.net
example.org A 93.184.216.34
example.org NS a.iana-servers.net
example.org NS b.iana-servers.net
example.org A 93.184.216.34
b.iana-servers.net

Fig. 1: Domain name delegation: parent and child authoritative servers.

gation, they can make the change in their child zone, but need to file a request with the Registry through the Registrar. This process currently always happens via an out-of-band channel (not through the DNS) and in some cases may even require forms on paper. Add to this that domain holders may not always be aware of this complexity and the requirement to keep parent and child in sync, and it is clear to see that keeping the DNS consistent is prone to human errors.

The problem of Parent-Child consistency is addressed in RFC7477 [6], which introduces a method to automatically keep records in the parent in sync through a periodical polling of the child using SOA records and a new type of record (CSYNC). Unfortunately, RFC7477 lacks deployment.

Pappas *et al.* [17] analyzed divergence between parent and child delegations on sample domains ($\sim$ 6M) from multiple zones and found inconsistencies in 21% of the DNS zones evaluated, in three different years. Kristoff [10] analysed delegations in `.edu` and finds that 25% of `.edu` delegations suffer some form of inconsistency. In his work, he considers 3 types of inconsistency: superset, subset and disjoint-set. Our work significantly expands on both studies by considering both the largest generic TLDs `.com`, `.net` and `.org` and the root zone of the DNS ($\sim$ 166 million domains, §3) and evaluating implications for resolvers in the wild (§4).

Liu *et al.* show that dangling delegation records referring to expired resources (e.g., cloud IP addresses or names) left in the parent or child pose a significant risk [11]. An attacker can obtain control of these records through the same cloud services by randomly registering new services, and in this way take control of the domain. Finally, Moura *et al.* [14] have looked into the consistency of time-to-live values [12] of parent and child NS records.

3 Parent and Child NSSet: are they consistent?

DNS NS records must be configured at both parent and child zones [12,5].

We compare `NS` records at parents and children in the wild considering all second-level domains (SLDs) under `.com`, `.net`, and `.org`, on 2019-10-16. We also evaluate the records in the Root DNS zone on 2019-10-30. We make use of OpenINTEL, a large-scale DNS measurement platform [23]. OpenINTEL collects daily active measurements of over 60% of the global DNS namespace every day.

	`.com` SLD	`.org` SLD	`.net` SLD	Root TLD	`.com` Ratio	`.org` Ratio	`.net` Ratio
Total domains	142,302,090	9,998,488	13,181,091	1528			
Unresponsive	19,860,226	949,137	1,663,403	0	14.0%	9.5%	12.6%
$P = C$	111,077,299	8,291,257	10,443,314	1476	78.0%	82.9%	79.2%
$P \neq C$	11,364,565	758,094	1,074,374	52	8.0%	7.6%	8.2%
$P \cap C = \emptyset$	6,594,680	418,269	548,718	16	58.0%	55.2%	51.0%
$IP(P) = IP(C)$	3,046,075	216,130	245,936	16	48.2%	53.9%	46.7%
$IP(P) \neq IP(C)$	3,265,171	184,885	280,988	0	51.8%	46.1%	53.3%
$IP(P) \cap IP(C) = \emptyset$	1,415,838	83,720	137,913	0	43.3%	45.3%	49.1%
$IP(P) \cap IP(C) \neq \emptyset$	1,849,333	101,165	143,075	0	56.7%	54.7%	51.9%
$P \cap C \neq \emptyset$	4,769,885	339,825	525,656	36	42.0%	44.8%	49.0%
$P \subset C$	3,506,090	236,257	369,442	18	73.5%	69.5%	70.2%
$P \supset C$	681,082	64,161	98,345	10	14.3%	18.9%	18.7%
Rest	582,713	39,407	57,869	8	12.2%	11.6%	11.1%

Table 1: Parent (P) and Child (C) NSSet consistency results. "IP" refers to `A` records of the NSSet of P and C.

For each SLD, we extract the sets of `NS` records from the parent and child authoritative servers, respectively indicated as P and C.

Table 1 shows the results of our comparative analysis. The first row shows the total number of SLDs for each TLD zone on the date considered. For the three zones, ~80% of SLDs have a consistent set of `NS` records at both the parent and the child zones. However, ~8% of SLDs (~ 13M) do not. For comparison, consider that 13M is almost as many domain names as some of the largest country-code TLDs (Germany's `.de`, one of the largest, has 16M SLDs [3]). The remaining 12% of domains are unresponsive to our queries. This could happen for different reasons, i.e. misconfigurations, failure, etc., not addressed in this work. We even see that 52 TLDs in the Root zone have inconsistent NSSets. Out of these, 26 are country-code TLDs (ccTLDs). We are currently notifying these ccTLD operators, in order to resolve these non-conforming setups, since they can have an adverse effect, among others, on load balancing.

Inconsistent NSSets classification: We classify inconsistent domain names into four categories: the cases in which (i) the parent and child NSSets are *disjoint*, (ii) the parent NSSet is a *subset* of the child NSSet, (iii) the parent NSSet is a *superset* of the child NSSet and (iv) the parent and child NSSet have a non-empty intersection but do not match (ii) or (iii).

For case (i), we observe that 51–58% of domains have completely *disjoint NSSets* ($P \cap C = \emptyset$). Depending on if resolvers are parent or child-centric, in this case resolvers will trust different `NS` records.

Given the surprising results for disjoint sets, we investigate the IP addresses of the NS records (IP(P, C, lines 4-7 in Table 1).[5] We discover that in half of

[5] This covers 96% of names with disjoint NSSets, the remaining 4% are indeterminate due to unresolvable names in the NSSets.

the cases, domains have disjoint NSSets that point to the same addresses, *i.e.,* there is an inconsistency of names but addresses match. In the other half, there is inconsistency also in addresses. Of these, $\sim 45\%$ have completely disjoint sets of IP addresses, for the remaining 55% there is some sort of overlap.

Disjoint sets may increase the risk of human error even in the case of name servers resolving to the same IP address, since operators would need to maintain redundant information in the parent and child, thus introducing fragility in the DNS data. Disjoint sets also may lead to lame delegations [7], *i.e.,* pointing resolvers to servers that may no longer be authoritative for the domain name.

Finally disjoint sets can be related to another malpractice: *CNAME configured on the Apex* [1]. However, further analysis shows that only a negligible percentage of cases are related to this.

Considering partially matching SLDs ($P \cap C \neq \emptyset$), we observe that 69–73% belong to case (ii), where the parent NSSet is a *subset* of the child NSSet. This may be intentional, e.g. an operator may want to first update the child and observe traffic shifts, and then later update the parent. Alternatively, operators may forget to update the delegation at the parent after updating the child.

Case (iii) where the parent NSSet forms a *superset* of the child NSSet ($P \supset C$) occurs in 14-18% of cases. This situation may introduce latency in the resolution process due to unresponsive name servers. Finally, the *Rest* category is case (iv), where the NSSets form neither a superset nor a subset, yet they have a non-empty intersection. Between 11-12% of SLDs fall in this category, and are susceptible to the range of operational issue highlighted for the previous categories.

Note that the OpenINTEL platform performs the measurements choosing *one* of the child authoritative nameservers. To verify how often sibling name servers have different configurations (child-child delegation inconsistency), we execute a measurement on a random sample of $\sim 1\%$ of `.org` domains (10k domains). The measurement suggests that $\sim 2\%$ of total parent-child delegation inconsistency cases also have child-child delegation inconsistencies, meaning that our results give a lower bound for the problem of parent-child mismatch. In fact, the OpenINTEL resolver could randomly choose a server configured correctly, while the others are not.

4 Implications of NSSet differences in the wild

We observed that roughly 8% of studied domains have parent/child inconsistencies. In this section, we investigate the consequences of such inconsistencies, by emulating the four categories of NSSet mismatches. We configure parent and child authoritative servers in eight different configurations (Table 2), and explore the consequences in terms of query load distribution. Our goal is to study these consequences in a controlled environment, where the authoritative name servers are in the same network. In the real-world, the authoritative name servers are often distributed geographically and the query load can depend on external factors, e.g. nearest server, popularity of a domain in a certain region, etc.

We emulate an operator that (i) has full control over its child authoritative name servers and (ii) uses the same zone file on all authoritative name servers (zones are synchronized). We place all child authoritative servers in the same network, thus, having similar latencies. We expect this to result in querying resolvers distributing queries evenly among child authoritatives [15].

As vantage points, we use RIPE Atlas [20,21], measuring each unique resolver as seen from their probes physically distributed around the world (3.3k ASes). Many Atlas probes have multiple recursive resolvers, so we treat each combination of probe and unique recursive resolver as a vantage point (VP), since potentially each represents a different perspective. We therefore see about 15k VPs from about 9k Atlas probes, with the exact number varying by experiment due to small changes in probe and resolver availability.

	Disjoint		**Subset**		**Superset**		**Rest**	
Experiment	Min-Off	Min-On	Min-Off	Min-On	Min-Off	Min-On	Min-Off	Min-On
Measurement ID	*23020789*	*23019715*	*23113087*	*23113622*	*23114128*	*23115432*	*23117852*	*23116481*
Frequency	600s							
Duration	2h							
Query	`A` $probeid-$timestamp.`marigliano.xyz` with 30 seconds TTL							
NSSet Parent	[ns1, ns3]		[ns1, ns3]		[ns1, ns2, ns3, ns4]		[ns1, ns2, ns3, ns4]	
NSSet Child	[ns2, ns4]		[ns1, ns2, ns3, ns4]		[ns2, ns4]		[ns2, ns4, ns5, ns6]	
TTL NS Parent	3600 s							
TTL NS Child	3600 s							
Date	20191003	20191003	20191025	20191025	20191025	20191026	20191027	20191027
Probes	9028	9031	8888	8883	8892	8879	8875	8875
VPs	15956	15950	15639	15657	15647	15611	15557	15586
Queries	190434	190333	184364	185706	186960	185015	182992	186472
Answers	178428	178416	169224	175200	175080	174804	174288	174504
From ns1, ns3	109661	175124	132179	169482	52233	83607	53944	84709
From ns2, ns4	65527	322	31753	1557	118835	86804	83100	85739
From ns5, ns6	N/A	N/A	N/A	N/A	N/A	N/A	31740	1545
fail	3240	2970	5292	4161	4012	4393	5504	2511

Table 2: Experiments to compare differents in Parent/Child NSSet

4.1 Disjoint Parent and Child NSSet

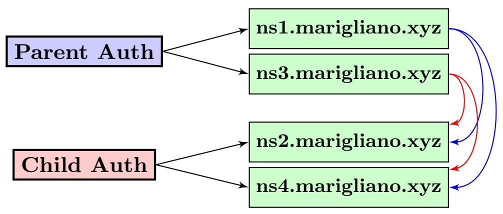


Fig. 2: Disjoint NSSset Experiment for marigliano.xyz

We have configured our test domain (`marigliano.xyz`) for the disjoint NSSet experiment as shown in Figure 2. For this experiment, we set the NSSet at the parent to [ns1, ns3].`marigliano.xyz`, while on the child authoritative servers, we set the NSSet to [ns2, ns4].`marigliano.xyz` (Table 2).

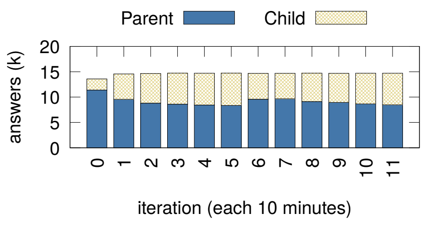
Zone files: we then configure the zone files of [ns1–ns4] to answer NS queries with [ns2,ns4], if explicitly asked, *i.e.,* the same records pointed to by the child authoritative servers. By doing that, we are able to single out resolvers that are *parent-centric*, since they will only contact [ns1,ns3].

As vantage points, we use ∼9k Atlas probes, and configure them to send `A` queries through each of their resolvers for $probeid-$timestamp.`marigliano.xyz`, which encodes the unique Atlas probe ID and query timestamp, thus avoiding queries of multiple probes interfering with each other. We also set the TTL value of the record to 30 s, and probe every 600 s, so resolver caches are expected to be empty for each round of measurements [13].

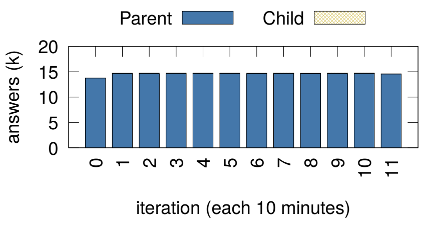
Our goal is to determine, *indirectly*, which NS records were used to answer the queries. To do that, we configure [ns1,ns3] to answer our `A` queries with the IP `42.42.42.42`, and [ns2,ns4] with the IP `43.43.43.43`. We use this approach instead of inspecting the query log on the server-side to speed up parsing and to avoid duplicated detection.

Figure 3a shows the results of the experiment. In round 0 of the measurements, we have a warm-up phase of RIPE Atlas probes, where not all the probes participate. Furthermore, we expect resolvers to have a cold cache and to use the NSSet provided by the parent. As the figure shows, this is mostly the case although 253 unique resolver IPs (different probes can share the same resolver) do contact the child name servers. This can be either due to them sending explicit NS queries (and thus learning about [ns2,ns4]) or because some probes share upstream caches. In subsequent rounds, we expect more traffic to go to the child name servers [ns2, ns4]. This is because resolvers learn about the child delegation from the "authority section" included in the response to the `A` query to ns1 or ns3. According to RFC2181 resolvers may prefer this information over the delegation provided by the parent. Indeed, in rounds [1–11] we see traffic also going to the child name servers. However, not all traffic goes to servers in the child NSSet, because not all resolvers trust data from the "authority section" due to mitigations against the so-called Kaminsky attack [8]. A key takeaway of this experiment is that domain owners may mistakenly assume traffic to go to the name servers in the child NSSet if they change it, whereas for this change to be effective, they must also update the parent NSSet.

The situation is even worse in our second experiment. Here, we configure [ns1–ns4] to answer with *minimal responses*, which prevents these servers from including "extra" records in the authority and additional sections of DNS answers. This means we do not expect resolvers to learn about the existence of [ns2,ns4] at all, since they are no longer present in the "authority section" of responses to the `A` queries. Only if resolvers perform explicit `NS` queries will they learn about [ns2,ns4]. As Figure 3b shows, as expected, almost all resolvers exclusively send their queries to the name servers in the NSSet of the parent.



(a) Results for normal responses



(b) Results with minimal responses

Fig. 3: Disjoint NSSet experiments

Only about 40 vantage points receive data from the name servers in the child NSSet, indicating their resolvers likely performed explicit `NS` queries. Authoritative name servers are increasingly configured to return minimal responses to dampen the effect of DNS amplification attacks, especially for DNSSEC-signed domains [19]. A key takeaway from this experiment is with this configuration becoming more and more prevalent, it becomes even more important to keep parent and child NSSets correctly synchronized.

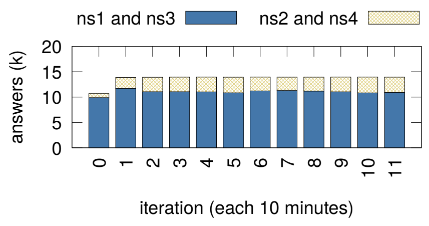
Real-world case: On 2019-10-30, we notified India's `.in`, given they had `ns[1-6].neustar.in` as NS records at the parent, and `[ns1-ns6].registry.in` at the child. However, altogether, both NSSets pointed to the same `A`/`AAAA` records and, as such, resolvers ended up reaching the same machines. After our notification, `.in` fixed this inconsistency on 2019-11-02 (we analyzed DNS OARC's root zone file repository [4]). Besides `.in`, 15 other internationalized ccTLDs run by India had the same issue with their NSset, and were also fixed.

4.2 Parent NSSet is a subset of Child

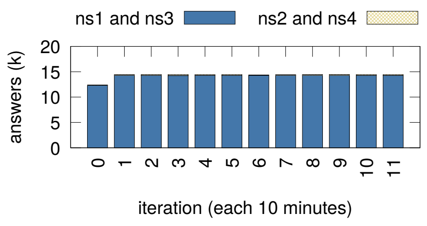
Recall from Table 1 that the majority (69-73%) of cases in which parent and child NSSets differ fall into the category where the child NSSets contains one or more additional `NS` records not present in the parent NSSet. A common reason to add additional `NS` records is to spread load over more name servers, and we assume this to be one of the reasons for this common misconfiguration.

We set up experiments to determine the consequences on query distribution if you have this setup. In other words: how many queries will eventually be answered by the extra `NS` record? We configure our test domain with [ns1, ns3] at the parent and [ns1, ns2, ns3, ns4] at the child. Like in the previous section, we configure [ns1, ns3] to give a different response to the `A` queries sent by the Atlas probes than [ns2, ns4], so we learn how many queries were answered by the name servers that are only in the child NSSet.

Figure 4a shows the results. Similary to the results shown in §4.1, most resolvers will use the `NS` records provided by the parent. Given that the child NSSet includes the NSSet at the parent, we see that the extra name servers receive only ∼24% of the queries. If in addition we configure the name servers



(a) Results for normal responses



(b) Results with minimal responses

Fig. 4: Subset NS sets experiments

to return minimal responses, we see that, just as in §4.1 virtually no resolvers contact the extra name servers in the child NSSet (Figure 4b). A key takeaway from these two experiments is that the, perhaps, expected even load distribution domain owners are hoping to see will not occur if only the child NSSet is updated. This again underlines the importance of keeping parent and child in sync.

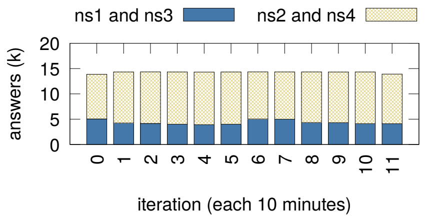
Real-world case: `att.com`: A real-world example that demonstrates that this type of misconfiguration also occurs for prominent domains is the case of `att.com`. We discovered that AT&T's main domain `att.com` had a parent NSSet containing `[ns1...ns3].attdns.com`, whereas the child had `[ns1...ns4].attdns.com`. We notified AT&T of this misconfiguration and on 2019-10-24 the issue was resolved when the fourth name server (`ns4.attdns.com`) was also added to the parent.

4.3 Parent NSSet is a superset of Child

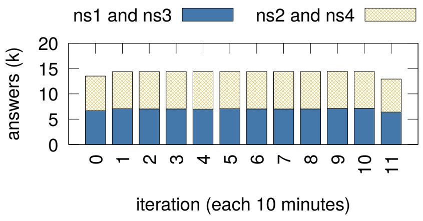
Roughly 14-18% of domain names that have different NSSet at parent and child have, one or more extra `NS` records at the *parent* ($P \supset C$ in Table 1). This could be due to operators forgetting to remove name servers that are no longer in use at the parent, but also the reverse case of the previous section in which a new name server is added at the parent but not added at the child.

To investigate the consequences of this for resolvers, we carry out experiments using Atlas VPs, setting four `NS` records at the parent ([ns1, ns2, ns3, ns4], as in Table 2) and only two at the child ([ns2, ns4]). Our goal is to identify the ratio of queries answered by the extra `NS` records at the parent.

Figure 5a shows the results for the experiment. As can be seen, the servers listed both in the parent and in the child ([ns2,ns4]) answer, on average, 68% of the queries. In case minimal responses are configured (Figure 5b), we see the queries being distributed evenly among the NS records in the parent. Consequently, having authoritative servers include an authority section in their answer to the `A` queries seems to cause *some* resolvers to prefer the child NSSet over the one in the parent. For example, Atlas VP (21448, 129.13.64.5) distributes queries only among ns2 and ns4, in the case of normal responses, instead it distributes queries among all name servers in case of minimal responses.

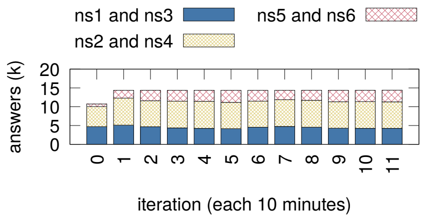


(a) Results for normal responses

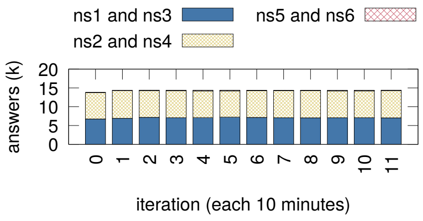


(b) Results with minimal responses

Fig. 5: Superset NS sets experiments



(a) Results for normal responses



(b) Results with minimal responses

Fig. 6: Rest NS sets experiments

These measurements then confirm that including "authority data" in the authoritative server responses will cause *some* resolvers to prefer only the child authoritative servers.

4.4 Mixed NSSets (Rest)

We have shown in Table 1 that in 11% of cases, the NSSet of the parent and child do not have a subset/superset relationship. Instead, some elements are present in both, but both parent and child have at least one NS that is not available in the other. To simulate this scenario, as shown in Table 2, we set four NS records at the parent: [*ns1*,ns2,*ns3*,ns4]. Then, at the child, we set [ns2,ns4,*ns5*,*ns6*], where the highlighted names show the ones not shared.

Figure 6a shows the experiment results. We see that [ns2,ns4], which are listed at both parent and child receive most queries. Then, records set only at the parent ([ns1,ns3]) are second to receive more queries. Finally, records set only at the child ([ns5,ns6]) receive the least amount of queries. In case of minimal responses (Figure 6b), the name servers only present at the child ([ns5,ns6]) receive virtually no traffic.

4.5 Discussion

Having inconsistent NSSets in parent and child authoritative servers impacts how queries are distributed among name servers, which plays an important role in DNS engineering. Overall, for all evaluated cases, queries will be unevenly distributed among authoritative servers – and the servers listed at the parent zone will receive more queries than then ones specified in the child.

5 Resolver software evaluation

The experiments carried out in §3 evaluates DNS resolver behavior in the wild. Since we use RIPE Atlas, we do not know what resolver software is used, if probes use DNS forwarders, or what kind of cache policies they use. We, however, see the aggregated behavior among a large set of configurations.

In this section, we focus on evaluating specific DNS resolver software instead, in a controlled environment, in order to understand how they behave towards DNS zones that are inconsistent with regards to their parent/child NSSet. Our goal is to identify which vendors conform to the standards. In particular, we pay attention as to whether resolvers follow RFC2181 [5], which specifies *how* resolvers should rank data in case of inconsistency: child authoritative data should be preferred.

We evaluate four popular DNS resolver implementations: *BIND* [9], *Unbound* [16], *Knot* [2], and *PowerDNS* [18]. We do this under popular Linux server distribution releases, using default packages and configurations. In addition, we evaluate resolvers shipped with various Windows server releases. Table 3 shows which vendors and versions we evaluate.

	Bind	Unbound	Knot	PowerDNS	Windows-DNS
Ubuntu-18-04	9.11.3-1	1.6.7	2.1.1	4.1.1	N/A
Ubuntu-16.04	9.10.3-P4	1.5.8	1.0.0	4.0.0	N/A
CentOS 7	9.9.4	1.6.6	2.4.1	4.1.9	N/A
CentOS 6	9.8.2rc1	1.4.20	N/C	3.7.4	N/A
Source	9.14.0	1.9.0	N/C	4.1.9	N/A
Windows	N/C	N/C	N/C	N/C	2008r2, 2012, 2016, 2019

Table 3: O.S. and resolver versions evaluated (N/Available, N/Covered)

Experiments: We configure the authoritative name servers for our test domain (`marigliano.xyz`) as a *disjoint* NSSet, as in §4.1. We configure the parent zone with [ns1,ns3]`.marigliano.xyz`, and the child with [ns2,ns4]`.marigliano.xyz`

Each experiment includes the four tests described in Table 4 (i–iv), in which we vary query types and query sequence. In (i), we ask the resolver for an `A` record of a subdomain in our test zone. In test (ii), we ask for the `NS` record of the zone. In (iii) we send first an `A` query followed by an `NS` query, to understand if resolvers use non-authoritative cached `NS` information to answer to the following query violating (§5.4.1 of RFC2181 [5]). In (iv) we invert this order to understand if authoritative record are overwritten by non-authoritative ones in the cache.

	(i) A Query	(ii) NS Query	(iii) A Query Then NS Query		(iv) NS Query Then A Query	
Query			First	Second	First	Second
Answer	C(A)	C(NS)	C(A)	C(NS)	C(NS)	C(A)
Cache	C(A); C(NS)	C(NS)	C(A); C(NS)	C(A); C(NS)	C(NS)	C(NS); C(A)
Minimal response enabled						
Answer	C(A)	C(NS)	C(A)	C(NS)	C(NS)	C(A)
Cache	C(A); P(NS)	C(NS)	C(A); P(NS)	C(A); C(NS)	C(NS)	C(NS); C(A)
Information provided by: C⇒ Child, P⇒ Parent						

Table 4: Expected Resolver Behavior

We dump the cache of the resolver after each query, and show which records are in cache and received by our client (we clear the cache after each query). Table 4 shows the expected NS usage by the resolvers, if they conform to the RFCs.

5.1 Results

We evaluate five resolver vendors and multiple versions. In total, we found that out of 22 resolvers/vendors evaluated, 13 conform to the RFCs. Next, we report the non-confirming resolver vendors/versions.

For experiment (i), in which we query for `A` records, we found that BIND packaged for Ubuntu did not conform to the standards: it caches only information from the parent and does not override it with information from the authoritative section provided by the child (which comes as additional section). This, in turn, could explain part of results of parent centricity observed in §4.

For experiment (i) and (iii), if we compile the latest *BIND* from source it also does not behave as expected: it sends the parent an explicit `NS` query before performing the `A` query. This is not a bad behavior, *i.e.,* it does not violate RFCs, instead it tries to retrieve more authoritative information. However, either if the name server information retrieved and used in the following query is the one provided by the child, *BIND* caches the data from the parent. This behavior of BIND could be one explanation of the small number of child-centric resolvers shown in §4 with Minimal Responses.

We are in the process of notifying BIND developers about this issue.

For experiment (iii), *PowerDNS* packaged for `CentOS 6` and `Ubuntu Xenial`, and Windows (all) use the cached non-authoritative information to answer the *NS* query in the test, not conforming to RFC2181.

PowerDNS notification We reached out to the developers of *PowerDNS*, who have confirmed the behavior. They do not maintain older versions anymore and the fix will not be backported due to the low severity of the problem. Our suggestion to the package maintainers of the distributions is to update the software to a newer version of the software.

6 Conclusions and Recommendations

Given a domain name, its NSSet in the parent and child DNS zones should be consistent [12]. This is the first study that shows, across the `.com`, `.net` and `org` zones (50% of the DNS namespace), that roughly 8% (13M) domains do not conform to that. We also show that DNS resolvers in the wild differ in behavior in returning information from the parent or child.

Inconsistency in parent and child NSSets have consequences for the operation of the DNS, such as improper load balancing among the name servers, increased resolution latency and unresponsive name servers. We strongly advise operators to verify their zones and follow RFC1034. To automate this process, we advise zone operators to consider supporting CSYNC DNS records (RFC7477) or other automated consistency checks, so the synchronization can be done in an automated fashion.

Finally, we also recommend that resolver vendors conform to the authoritative information ranking in RFC2181 (taking into account the recommendations to mitigate the Kaminsky attack as specified in RFC5452), and when possible, to *explicitly* ask for the child's NS records, similarly to what is done in DNSSEC, where signed records are only available at the child (§5).

Acknowledgments

We thank John Heidemann, Ólafur Guðmundsson and Ülrich Wisser for feedback provided in the early stages of this research. We also thank the PAM2020 anonymous reviewers, our shepherd, Steve Uhlig, and Philip Homburg, from RIPE NCC. This work uses measurements from RIPE Atlas (https://atlas.ripe.net), an open measurements platform operated by RIPE NCC.

This work is partially funded by the NWO-DHS MADDVIPR project (Grant Agreement 628.001.031/FA8750-19-2-0004), the PANDA project (NSF OAC-1724853) and the EU CONCORDIA project (Grant Agreement 830927). This material is based on research sponsored by Air Force Research Laboratory under agreement number FA8750-18-2-0049. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions in this paper are those of the authors and do not necessarily reflect the opinions of a sponsor, Air Force Research Laboratory or the U.S. Government.

References

1. Almond, C.: CNAME at the apex of a zone. https://www.isc.org/blogs/cname-at-the-apex-of-a-zone/
2. CZ.NIC: Knot Resolver. https://www.knot-resolver.cz
3. DENIC AG: Statistics of .de domains (Oct 22 2019), https://www.denic.de/en/know-how/statistics/1
4. DNS OARC: Root Zone Archive. https://www.dns-oarc.net/oarc/data/zfr/root (Jan 2020)
5. Elz, R., Bush, R.: Clarifications to the DNS Specification. RFC 2181, IETF (Jul 1997), http://tools.ietf.org/rfc/rfc2181.txt

6. Hardaker, W.: Child-to-Parent Synchronization in DNS. RFC 7477, IETF (Mar 2015), http://tools.ietf.org/rfc/rfc7477.txt
7. Hoffman, P., Sullivan, A., Fujiwara, K.: DNS Terminology. RFC 8499, IETF (Nov 2018), http://tools.ietf.org/rfc/rfc8499.txt
8. Hubert, A., Mook, R.v.: Measures for Making DNS More Resilient against Forged Answers. RFC 5452, IETF (Jan 2009), http://tools.ietf.org/rfc/rfc5452.txt
9. Internet Systems Consortium: BIND: Berkeley Internet Name Domain. https://www.isc.org/bind/
10. Kristoff, J.: DNS inconsistency. https://blog.apnic.net/2018/08/29/dns-inconsistency/ (2018)
11. Liu, D., Hao, S., Wang, H.: All Your DNS Records Point to Us: Understanding the Security Threats of Dangling DNS Records. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1414–1425. CCS '16, ACM, New York, NY, USA (2016). https://doi.org/10.1145/2976749.2978387
12. Mockapetris, P.: Domain names - concepts and facilities. RFC 1034, IETF (Nov 1987), http://tools.ietf.org/rfc/rfc1034.txt
13. Moura, G.C.M., Heidemann, J., Müller, M., de O. Schmidt, R., Davids, M.: When the dike breaks: Dissecting DNS defenses during DDoS. In: Proceedings of the ACM Internet Measurement Conference (Oct 2018). https://doi.org/https://doi.org/10.1145/3278532.3278534
14. Moura, G.C.M., Heidemann, J., de O. Schmidt, R., Hardaker, W.: Cache Me If You Can: Effects of DNS Time-to-Live (extended). In: Proceedings of the ACM Internet Measurement Conference. p. to appear. ACM, Amsterdam, the Netherlands (Oct 2019). https://doi.org/https://doi.org/10.1145/3355369.3355568
15. Müller, M., Moura, G.C.M., de O. Schmidt, R., Heidemann, J.: Recursives in the wild: Engineering authoritative DNS servers. In: Proceedings of the ACM Internet Measurement Conference. pp. 489–495. London, UK (2017). https://doi.org/https://doi.org/10.1145/3131365.3131366
16. NLnet Labs: Unbound. https://unbound.net/ (Mar 2019)
17. Pappas, V., Wessels, D., Massey, D., Lu, S., Terzis, A., Zhang, L.: Impact of configuration errors on DNS robustness. IEEE Journal on Selected Areas in Communications **27**(3), 275–290 (2009)
18. PowerDNS: PowerDNS Recursor. https://www.powerdns.com/recursor.html
19. van Rijswijk-Deij, R., Sperotto, A., Pras, A.: DNSSEC and Its Potential for DDoS Attacks: a comprehensive measurement study. In: Proceedings of the 2014 ACM Conference on Internet Measurement Conference. pp. 449–460. IMC, ACM (Nov 2014)
20. RIPE NCC Staff: RIPE Atlas: A Global Internet Measurement Network. Internet Protocol Journal (IPJ) **18**(3), 2–26 (Sep 2015)
21. RIPE Network Coordination Centre: RIPE Atlas. https://atlas.ripe.net (2015)
22. Root Zone file: Root (Feb 2019), http://www.internic.net/domain/root.zone
23. van Rijswijk-Deij, R., Jonker, M., Sperotto, A., Pras, A.: A High-Performance, Scalable Infrastructure for Large-Scale Active DNS Measurements. IEEE Journal on Selected Areas in Communications **34**(6), 1877–1888 (June 2016). https://doi.org/10.1109/JSAC.2016.2558918

A Longitudinal view on inconsistency

A.1 NS Inconsistency over time

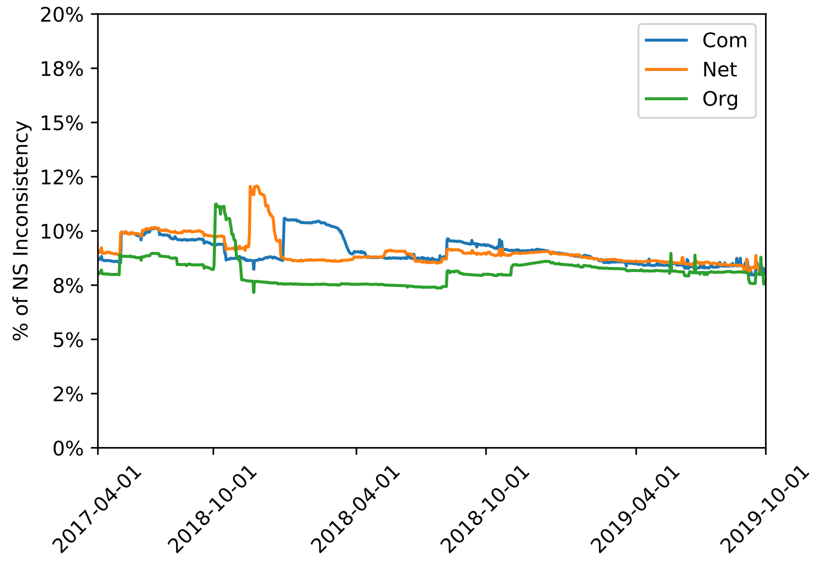


Fig. 7: NS inconsistency ($P \neq C$) from 2017-04-01 until 2019-10-01

The results presented in Table 1 show NS inconsistency for a single day. However, it is also interesting to understand how this misconfiguration evolves over time. We analyzed NS inconsistency for the case $P \neq C$ over the two and a half year-period preceding the date of the analysis presented in Table 1. Fig. 7 shows the results of this analysis. The figure clearly demonstrates that the fraction of domains affected by this misconfiguration remains similar over time. This result suggests that NS inconsistency is a long-term misconfiguration in the DNS ecosystem.