**The ISP Column**
*A monthly column on things Internet*

February 2023
Geoff Huston

# The Root Zone of the DNS Revisited

The DNS is a remarkably simple system. You send it queries and you get back answers. Within the system you see exactly the same simplicity: The DNS resolver that receives your query may not know the answer, so it, in turn, will send queries deeper into the system and collects the answers. The query/response process is the same, applied recursively. Simple.

However, the DNS is simple in the same way that Chess or Go are simple. They are all constrained environments governed by a small set of rigid rules, but they all possess astonishing complexity.

Simple systems can have very deep complexity. This is a major theme in the study of Formal Systems.

The study of mathematics in the 19th century moved into to dizzying heights of self-reflection. The question they were trying to answer was: What are the formal assumptions upon which the entire superstructure of mathematics was constructed? The *Peano Axioms* for natural numbers are a good example here. If you took these axioms and only applied operations from a constrained and well-defined set, then was it possible to derive every known truth (provably correct) statement in maths? This question motivated Whitehead and Russell to labour on the landmark three volume work Principia Mathematica in the early 20th century, that was intended to build the entire edifice of mathematics using only first principles and symbolic logic. Their work was in part brought about by an interest in logicism, the view on which all mathematical truths are logical truths. Mathematics was seen as a "pure" form of philosophical study, whose truths were independent of any observer or any natural system. This study led to work by Kurt Gödel that probed the limits of this approach. His *Incompleteness Theorems* are two theorems of mathematical logic that demonstrate the inherent limitations of every formal axiomatic system capable of modelling basic arithmetic. These results are important both in mathematical logic and in the philosophy of mathematics. The first incompleteness theorem states that no consistent system of axioms whose theorems can be listed by an effective procedure is capable of proving all truths about the arithmetic of natural numbers. For any such consistent formal system, there will always be statements about natural numbers that are true, but that are unprovable within the system. The second incompleteness theorem, an extension of the first, shows that the system cannot demonstrate its own consistency.

With all our faith in rule-based automated systems that largely operate today's digital world its sobering to realize that the limitations of such a world view were clearly stated by Kurt Gödel 1931.

Why am I talking about this? Well, the informal expression of Gödel's work is that any formal system that is powerful enough to be useful is also powerful enough to express paradoxes. Or more informally, any constrained simple system that is sufficiently useful to express compound concepts is also capable of impenetrable complexity. And this is where the DNS comes in!

## The Root Zone

The DNS is not a dictionary of any natural language, although these days when we use DNS names in our spoken language, we might be excused from getting the two concepts confused! The DNS is a hierarchical name space. Individual domain names are constructed using an ordered sequence of labels. This ordered sequence of labels serves a number of functions, but perhaps most usefully it can be used as an implicit procedure to translate a domain name into an associated attribute value through the DNS name resolution protocol.

For example, I operate a web server that is accessed using the DNS name *www.potaroo.net*. If you direct your browser to load the contents of this DNS name then your system firstly needs to resolve this DNS name to an IP address, so that your browser knows where to send the IP packets to perform a transaction with my server. This is where the structure of the name is used. In this case the DNS system will query a root server to translate this name to a corresponding IP address and the response from any root server to such a query will be the set of resolvers that are authoritative for the *.net* zone. Ask any of these *.net* servers for this same name and the response will be the servers that are authoritative for the *potaroo.net* zone. Ask any of these *potaroo.net* servers for the same name and you will get back the IP address you are looking for. Every DNS name can be decomposed in the same way. The name itself defines the order of name resolution processing.

There is one starting point for every DNS resolution operation: the root zone.

There is a school of thought that decries any exceptional treatment given to the root zone of the DNS. It's just another zone, like any other. It's a set of authoritative servers that receive queries and answer them, like any other zone. There's no magic in the root zone and all this attention on the root zone as *special* is entirely unwarranted.

However, I think this view understates the importance of the root zone in the DNS. The DNS is a massive distributed database. Indeed, it's so massive that there is no single static map that identifies every authoritative source of information and the collection of data points about which it is authoritative. Instead we use a process of *dynamic discovery* where the resolution of a DNS name firstly is directed to locating the authoritative server that has the data relating to the name we want resolved, and then querying this server for the data. The beauty of this system is that these discovery queries and the ultimate query are precisely the same query in every case.

But everyone has to start somewhere. A DNS recursive resolver does not know all the DNS authoritative servers in advance and never will. But it does know one thing. It knows the IP address of at least one of the root servers. From this starting point everything can be constructed on the fly. The resolver can ask a root server for the names and IP addresses of all other root servers (the so-called *priming query*), and it can store that answer in a local cache. When the resolver is given a name to resolve it can then start with

a query to a root server to find the next point in the name delegation hierarchy and go on from there in a recursive manner.

If this was how the DNS actually worked, then it's pretty obvious that the DNS system would've melted down years ago. What makes this approach viable is *local caching*. A DNS resolver will store the answers in a local cache and use this locally held information to answer subsequent queries for the life of the cached entry. So perhaps a more refined statement of the role of the root servers is that every DNS resolution operation starts with a query to the cached state of the root zone. If the query cannot be answered from the local cache, then a root server is queried.

However, behind this statement lurks an uncomfortable observation. If all of the root servers are inaccessible, then the entire DNS ceases to function. This is perhaps a dramatic overstatement in some respects, as there would be no sudden collapse of the DNS and the Internet along with it. In the hypothetical situation where all the instances of the root servers were inaccessible then DNS resolvers would continue to work using locally cached information. However, as these cached entries time out, they would be discarded from these local resolvers (as they could not be refreshed by re-querying the root servers). The light of the DNS would fade to black bit by bit as these cached entries time out and are removed. The DNS root zone is the master lookup for every other zone. That's why it deserves particular attention. For that reason, the DNS root zone is uniquely different from every other zone.

Due to local caching, root zone servers are not used for every DNS lookup. The theory is that the root servers will only see queries as a result of cache misses. With a relatively small root zone and a relatively small set of DNS recursive resolvers, then the root zone query load should be small. Even as the Internet expands its user base the query load at the root servers does not necessarily rise in direct proportion. It's the number of DNS resolvers that supposedly determines root server query load if we believe in this model of the root's function in the DNS.

However, the model may not hold up under operational experience. The total volume of queries per day recorded by the eroot servers is shown in Figure 1.

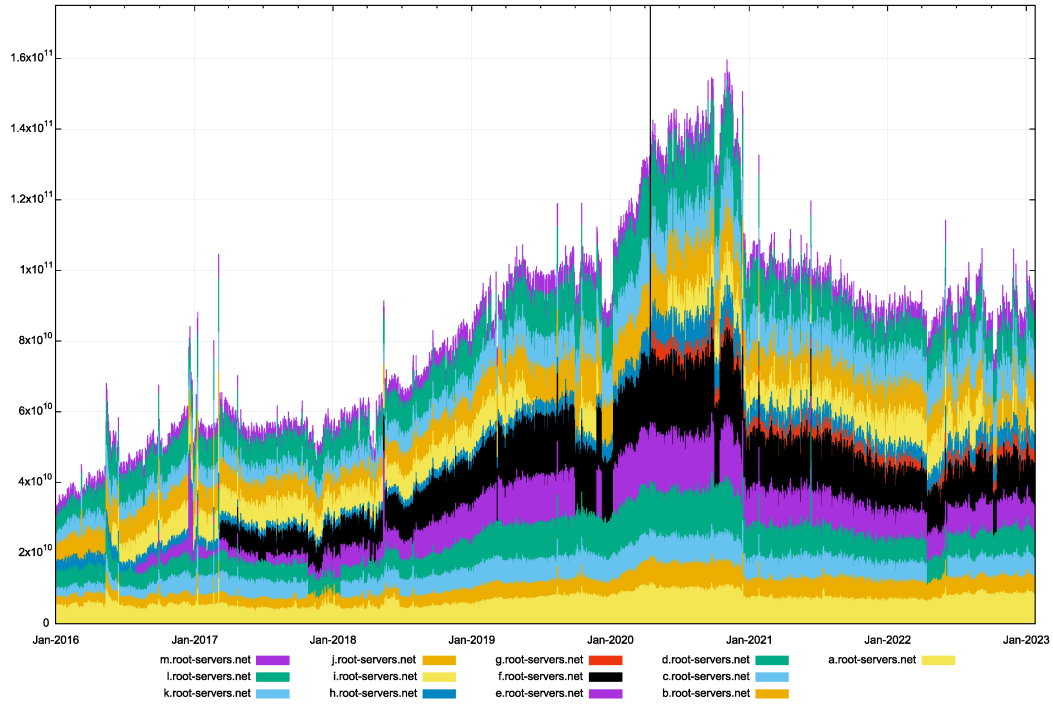


*Figure 1 – Total Root Servers Queries per Day (RSSAC002 data)*

Over the period from 2016 to 2020 the volume of queries seen by the collection of root servers has tripled. Happily, the root zone query volume has come down in 2021 and stabilised over 2022. It is likely that changes to the behaviour of the Chrome browser may explain this. Chrome used to probe the local

DNS environment by making a sequence of queries to non-existent names upon startup and because the query names referred to undelegated top-level domains, these queries were a significant component of the queries seen at the root servers. Changing this behaviour in Chrome t the end of 2020 appears to have resulted in a dramatic change to the DNS query profile as seen by the root servers. However, the relative stability of query volumes at the root servers that has been observed over the past two years may not be sustained indefinitely.

What are we doing in response to this? How are we ensuring that the root zone service can continue to grow in capacity in response to a resumption in the growth of query rates?

## Root Zone Scaling

The original model of authoritative servers in the DNS was based on the concept of unicast routing. A server name had a single IP address, and this single server was located at a single point in the network. Augmenting server capacity entailed using a larger server and adding network capacity. However, such a model does not address the issues of a single point of vulnerability, nor does it provide an optimal service for distant clients.

The DNS approach to this is to use multiple name server records. A DNS resolver was expected to retry its query with a different server if its original query did not elicit a response. That way, a collection of servers could provide a framework of mutual backup. To address the model of optimal choice, DNS resolvers were expected to maintain a record of the query/response delay for each of the root servers and prefer to direct their queries to the fastest server.

Why not use multiple address records for a single common server name? The two approaches (multiple server names and multiple address records for a name) look similar, but typical DNS resolver behaviour apparently differs between these two cases.

A DNS resolver is expected to prefer to use a particular server name based on query delay. On the other hand, a DNS resolver is expected to rotate its queries in round-robin order across all IP addresses for the same server name.

I must admit that I'm not overly happy this this explanation, as it seems to be somewhat of an artifice. In the original model of IP, hosts with multiple network interfaces have multiple IP addresses as the IP address of a host is the address of its point of attachment to the network, and not the host itself. So, the situation of multiple addresses for a server is interpreted as a collection of addresses that represent some form of path diversity to reach the same unique host. Multiple names were supposedly used to denote different servers, I guess. But such scenarios do not seem to match today's experience, where single platforms may have multiple names and IP addresses, and multiple platforms may share a single name and a single IP address. The implication of this observation is that once a resolver has assembled a collection of IP addresses that represent the nameservers for a domain, then it seems to me that a resolver could be justified for treating the list of IP addresses consistently irrespective of whether the list was assembled from multiple IP addresses associated with a single name, or from multiple names.

So why do we have multiple server names that each have one IPv4 address and one IPv6 address for the root zone? I'm not sure I have a

good explanation, other than: "That's the way we've always enumerated the servers for the root zone."

The scaling issue with multiple servers is the question of completeness and the size of the name server response to the *priming query*. The question here is: If a resolver asks for the name servers of the root zone, should the resolver necessarily be informed of *all* such servers in the response? The size of the response will increase with the number of servers, and the size of the response may exceed the default maximal DNS over UDP payload size of 512 bytes. The choice of the number of server names for the root zone, 13, was based on the calculation that this was the largest list of a server list that could fit into a DNS response that was under 512 bytes in size. This assumed that only the IPv4 address records were being used in the response, and with the addition of the IPv6 AAAA records the response size has expanded. The size of the *priming response* for the root zone with 13 dual stack authoritative servers is 823 bytes, or 1,097 bytes if the DNSSEC signature is included, and slightly larger if DNS cookies are added. In today's DNS environment, if the query does not include an EDNS(0) indication that they can accept a DNS response over UDP larger than 512 bytes, then the root servers will provide a partial response in any case, generally listing all 13 names, but truncating the list addresses of these services in the Additional Section to fit with a 512 byte payload.

If we can't, or don't want to, just keep on adding more root servers to name server set in the root zone, then what are the other scaling options for serving the root zone?

The first set of responses to these scaling issues was in building root servers that have greater network capacity and greater processing throughput. But with just 13 servers to work with, then this was never going to scale at the pace of the Internet. We needed something more. The next scaling step has been in the conversion from *unicast* to *anycast* services. There may be 26 unique IP addresses for root servers (13 in IPv4 and 13 in IPv6) but each of these service operators now use *anycast* to replicate the root service in different locations. The current number of root server sites is described at root-servers.org (Table 1). Now the routing system is used to optimise the choice of the "closest" location for each root server.

| Root | Anycast Sites |
|---|---|
| A | 58 |
| B | 6 |
| C | 12 |
| D | 181 |
| E | 254 |
| F | 336 |
| G | 6 |
| H | 12 |
| I | 68 |
| J | 163 |
| K | 97 |
| L | 192 |
| M | 11 |
| Total | 1,396 |

*Table 1 – Anycast Site Counts for Root Servers, January 2023*

The root server system has enthusiastically embraced *anycast*. That's a total of 1,396 sites where there are instances of root servers. Some 28 months earlier, in September 2020, the root server site count was 1,098, so that's a 30% increase in the number of sites in a little over two years!

The number of authoritative server engines is larger than that count of the number of sites, as its common these days to use multiple server engines within a site and use some form of query distribution front-end to distribute the incoming query load across multiple back-end engines at each site.

However even this form of expanding the distributed service may not be enough. If the growth profile of 2016-2020 resumes, then in two years from now we may need double the root service capacity from the current levels, and in a further two years we'd need to double it again. And again, and again, and again. Exponential growth is a very harsh master. Can this anycast model of replicated root servers expand indefinitely? Or should we look elsewhere for scaling solutions? To attempt to answer this question we should look at the root service in a little more detail.

There have been many studies of the root service and the behavior of the DNS over the past few decades. If the root servers were simply meant to collect the cache misses of DNS resolvers, then whatever is happening at the root is not entirely consistent with a model of resolver cache misses. Indeed, it's not clear what going on at the root!

It has been reported that the majority of queries to the root servers result in NXDOMAIN responses. In looking at the published response code data, it appears that some 55% of root zone queries result in NXDOMAIN responses (Figure 2). The NXDOMAIN response rate was as high as 75% in 2020, and dropped presumably when the default behaviour of the Chrome browser changed. In theory these queries are all cache misses at the recursive resolver level, so the issue is that the DNS is not all that effective in handling cases where the name itself does not exist.

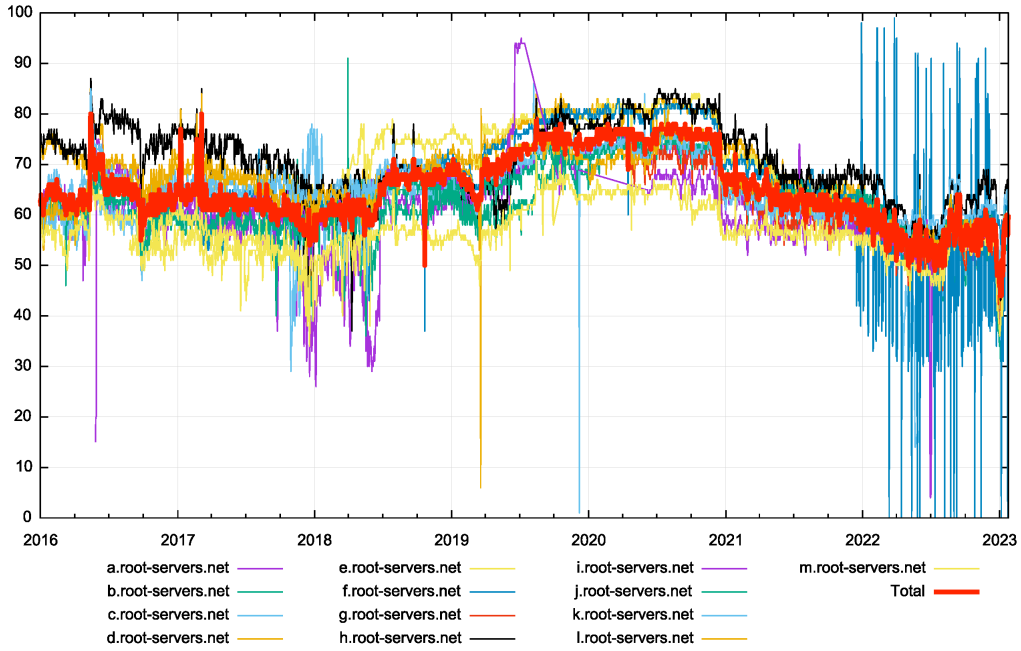


*Figure 2 – Proportion of Root Zone NXDOMAIN responses per Day (RSSAC002 data)*

## Query Deflection

If we want to reduce the query pressure on the root servers it appears that we need to handle the case of queries for non-existent names, and in particular names where top-level label in the name is not delegated in the root zone. How else can we deflect these queries away from the root server system?

There are two approaches that may help.

### NSEC Caching

The first is described in RFC8198, *aggressive NSEC caching*. When a top level label does not exist in a DNSSEC-signed zone, and the query has the DNSSEC EDNS(0) flag enabled, the NXDOMAIN response from a root server includes a signed NSEC record that gives the two labels that exist in the root zone that "surround" the non-existent label. NSEC records say more than "this label is not in this zone". It says that every label that is lexicographically between these two labels does not exist. If the recursive

resolver caches this NSEC record it can use this same cached record to respond to all subsequent queries for names in this label range, in the same way that it conventionally uses "positive" cache records.

```
$ dig +dnssec this.is.not.a.tld @a.root-servers.net

; <<>> DiG 9.18.10 <<>> +dnssec this.is.not.a.tld @a.root-servers.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 18390
;; flags: qr aa rd; QUERY: 1, ANSWER: 0, AUTHORITY: 6, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; QUESTION SECTION:
;this.is.not.a.tld.     IN          A

;; AUTHORITY SECTION:
.           86400       IN          SOA         a.root-servers.net. nstld.verisign-grs.com. 2023020601 1800 900 604800 86400
.           86400       IN          RRSIG       SOA 8 0 86400 20230219170000 20230206160000 951 . wkTq6wKu2J5w […] STwcw==
tl.         86400       IN          NSEC        tm. NS DS RRSIG NSEC
tl.         86400       IN          RRSIG       NSEC 8 1 86400 20230219170000 20230206160000 951 . kgQXGN5Vo […] vQHw==
.           86400       IN          NSEC        aaa. NS SOA RRSIG NSEC DNSKEY
.           86400       IN          RRSIG       NSEC 8 0 86400 20230219170000 20230206160000 951 . LVRI […] DiIQ==

;; Query time: 127 msec
;; SERVER: 198.41.0.4#53(a.root-servers.net) (UDP)
;; WHEN: Tue Feb 07 10:12:17 AEDT 2023
;; MSG SIZE  rcvd: 1031
```

*Figure 3 – A NXDOMAIN response from a root server*

In the example in Figure 3, a DNSSEC-enabled query for the name `this.is.not.a.tld` elicits a response which includes a signed NSEC record that asserts there are no delegated TLDs between `.tl` and `.tm`, and any query for a name that lies between these two labels would elicit exactly the same response. If a recursive resolver cached both the 1,481 top level delegated labels and the 1,481 NSEC records in the root zone, then the resolver would not need to pass any queries to a root server for the lifetime of the cached entries. If all recursive resolvers performed this form of NSEC caching of the root zone, then the query volumes seen at the root from recursive resolvers would fall significantly for non-existent labels.

How many TLDS are in the Root Zone?

There are 1,481 top level domains in the root zone of the DNS in February 2023. It has not always been this size. The root zone started with a small set of generic labels, and in the late 1980's expanded to include the set of two-letter country codes. There were some tentative steps to augment the number of generic top level domain names, and then in the 2010's ICANN embarked on a larger program of generic TLD expansion. Figure 4 shows the daily count of TLDs in the root zone since 2014.

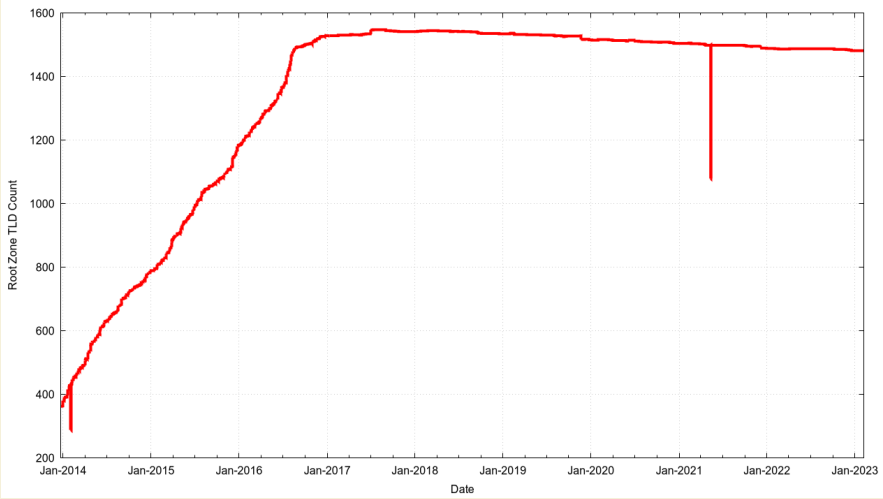


*Figure 4 Daily count of root zone TLDs*

What was surprising to me when I generated this data set was that top level domains are not necessarily permanent. The largest TLD count occurred in August 2017 with 1,547 TLDs, and since then the number of TLDs have been gently declining.

NSEC caching in recursive resolvers could play a valuable role in scaling the root zone. Bind supports this as of release 9.12. Unbound supports this as of release 1.7.0. Knot resolver supports this as of 2.0.0. But the queries at the root zone keep growing.

However, NSEC caching is a tactical response to root zone scaling concerns, as distinct from a strategic response. It's still dependent on the root server infrastructure and uses a query-based method of promulgating the contents of the root zone. Nothing really changes in the root service model. What NSEC caching does is allow the resolver to make full use of the information in the NSEC response. Nothing else changes.

### Local Root and ZONEMD

Another option is to jump out of the query/response model of learning the contents of the root zone and simply load the entire root zone into recursive resolvers. The idea is that if a recursive resolver is loaded with a copy of the root zone then it can operate autonomously with respect to the root servers for the period of validity of the local copy of the root zone contents. It will send no further queries to the root servers. The procedures to follow to load a local root zone are well documented in RFC8806, and I should also note here the existence of the LocalRoot service (https://localroot.isi.edu/) that offers DNS NOTIFY messages when the root zone changes.

The root zone is not a big data set. A signed, uncompressed plain text copy of the root zone as at the start of February 2023 is 2,258,235 bytes in size.

However, this approach has its drawbacks. How do you know that the zone you might have received via some form of zone transfer or otherwise is the current genuine root zone? Yes, the zone is signed, but not every element in the zone is signed (NS records for delegated zones are unsigned). The client is left with the task of performing a validation of every digital signature in the zone, and at present there are some 2,847 RRSIG records in the root zone. Even then the client cannot confirm that its local copy of the root zone is complete and authentic, because of these unsigned NS delegation records.

The IETF published RFC 8976, the specification of a *message digest* record for DNS zones, in February 2021. This RFC defines the ZONEMD record.

What's a Message Digest?

A *message digest* is a form a condensed digital signature of a digital artefact. If the digital artefact has changed in any way, then the digest will necessarily change in value as well. If a receiver of this artefact is given the data object and its digest value then the receiver can be assured, to some extent, that the contents of the file have been unaltered since the digest was generated.

These digital signatures are typically generated using a cryptographic hash function. These functions have several useful properties. They are normally a fixed length function, so that the function value is a fixed size irrespective of the size of the data for which the hash has been generated. They are a unidirectional function, in that knowledge of the hash function value will not provide any assistance in trying to recreate the

original data. They are deterministic, in that the same hash function applied to the same data will always product the same hash value. Any form of change to the data should generate a different hash value. Hash functions do not necessarily produce a unique value for each possible data collection, but it should be exhaustively challenging (unfeasible) to synthesise or discover a data set that produces a given hash value (pre-image resistance), and equally challenging to find or generate two different data sets that have the same hash function value (collision resistance).

This means that an adversary, malicious or otherwise, cannot replace or modify the data set without changing its digest value. Thus, if two data sets have the same digest, one can be relatively confident that they are identical. Second pre-image resistance prevents an attacker from crafting a data set with the same hash as a document the attacker cannot control. Collision resistance prevents an attacker from creating two distinct documents with the same hash.

What if the root zone included a ZONEMD record, signed with the Zone Signing Key of the root zone? If a client received such a root zone that included this record, then it could validate the RRSIG of the ZONEMD record in the same way that it DNSSEC-validates any other RRSIG entry in the root zone, then use the value of this record and compare it with a locally calculated message digest value of the root zone. If the digest values match, then the client has a high level of assurance that this is an authentic copy of the root zone and has not been altered in any way.

The dates in the DNSSEC signatures can indicate some level of currency of the data, but further assurance at a finer level of granularity than the built-in key validity dates that the local copy of the root zone data is indeed the current value of the root zone is a little more challenging in this context. DNSSEC does not provide any explicit concept of revocation of prior versions of data, so all 'snapshots' of the root zone within the DNSSEC key validity times are equally valid for a client. The root zone uses a two-week signature validity period (Figure 5).

```
. 86400 IN SOA a.root-servers.net. nstld.verisign-grs.com. 2023020501 1800 900 604800 86400
. 86400 IN RRSIG SOA 8 0 86400 20230218170000 20230205160000 951 .  dtm […] SgQ==
```

*Figure 5 – Root Zone SOA signature*

Despite this, the approach of a whole-of-zone signature has some real potential in terms of the distribution of the root zone to DNS resolvers, and thereby reduce the dependency on the availability and responsiveness of the root zone servers. The use of the ZONEMD record would allow any client to use a copy of the root zone irrespective on the way in which the zone file was obtained. Within the limits of the authenticated currency of the zone file as already noted, any party can redistribute a copy of the root zone and clients of such a redistributed zone can answer queries using this data with some level of confidence that the responses so generated are authentic.

While a local zone administrator can add a ZONEMD record to the zone or zones they administer at any time (assuming that the zone is already DNSSEC-signed), the story is somewhat different for the root zone. The risk is that if the inclusion of this new resource record in the root zone causes resolvers to reject the zone, then the consequences could be dire if this rejection is widespread. This is not unlike the issues with the roll of the KSK, although the likelihood of zone file rejection with the addition of this ZONEMD record is apparently considered to be a low-level risk at this stage. In any case the addition of a new record to the root zone involves many interested parties, including the IANA Functions Operator, the Root Zone Maintainer, and the Root Server Operators. The approach to introduce this record into the root zone follows a practice previously used to introduce DNSSEC keys into the zone,

starting with a "dummy" record that presents the resource record but in a format that cannot be used to validate the data by using a private-use hash algorithm number in the first instance. The second phase of the process is then to cut over to a value that can be used to verify the zone data.

Recursive resolvers on the Internet are not impacted by plans to add the ZONEMD record if they interact with the root zone and it's servers via standard query mechanisms. There is no expectation that resolvers will issue queries for the ZONEMD record, and there is no harm should they do so in any case.

For those resolvers who elect to use a locally managed copy of the root zone, the zoneMD record, they will benefit from a resolver implementation that support ZONEMD and can use it to verify a received zone. Resolver implementations that perform this verification include Unbound (from v1.13.23) and PowerDNS Recursor (from v4.7.04) and Bind (v9.17.13).

Notification mechanisms that could prompt a resolver to work from a new copy of the root zone are not addressed in this framework. To me that's the last piece of the framework that could promote every recursive resolver into a peer root server. We've tried a number of approaches to scalable distribution mechanisms over the years. There is the structured push mechanism where clients sign up to a distributor and the distributor pushes updated copies of the data to clients. Routing protocols use this mechanism. There is the pull approach where the client probes its feed point to see if the data has changed and pulls a new copy if it has changed. This mechanism has some scaling issues in that aggressive probing by clients may overwhelm the distributor, and we've also seen hybrid approaches where a change indication signal is pushed to the client, and it is up to the client to determine when to pull the new data.

This model of local root zone distribution has the potential to change the nature of the DNS root service, unlike NSEC caching. If there is one thing that we've learned to do astonishingly well in recent times its distribution of content. Indeed, we've concentrated on this activity to such an extent that it appears that the entire Internet is nothing more than a small set of Content Distribution Networks. If the root zone is signed in its entirety with zone signatures that allow a recursive resolver to confirm its validity and currency and submitted into these distribution systems as just another digital object, then the CDN infrastructure is perfectly capable of feeding this zone to the entire collection of recursive resolvers with ease. Perhaps if we changed the management regime of the root zone to generate a new zone file every 24 hours according to a strict schedule, we could eliminate the entire notification superstructure. Each iteration of the root zone contents would be published 2 hours in advance and is valid for precisely 24 hours, for example. At that point the root zone will be served by millions of recursive resolvers rather than the twelve operators we use today.

## Options? Pick them all!

We operate the root service in its current framework because it's been functionally adequate so far. That is to say the predominate query-based approach to root zone distribution hasn't visibly collapsed in a screaming heap of broken DNS yet! But we don't have to continue relying only on this query-based approach just because it hasn't broken so far. Our need to further scale this function is still a current need, and it makes a whole lot of sense to take a broader view of available options. We have some choices as to how the root service can evolve and scale.

By deflecting some of the current load to delegation points lower in the domain hierarchy we can make a massive change in the current root query load, as we've already seen with the changes to the Chrome browser.

By having resolvers make better use of signed NSEC records we can stave off some of the more pressing immediate issues about further scaling of the root system.

But that's probably not enough. We can either wait for the system to collapse and then try and salvage the DNS from the broken mess, or perhaps we could explore some alternatives now, and look at how we can break out of a query-based root content promulgation model and view the root zone as just

another content "blob" in the larger ecosystem of content distribution. If we can cost efficiently load every recursive resolver with a current copy of the root zone, and these days that's not even a remotely challenging target, then perhaps we can put aside the issues of how to scale the root server system to serve ever greater quantities of "NO!" to ever more demanding clients!

## Disclaimer

The above views do not necessarily represent the views or positions of the Asia Pacific Network Information Centre.

## Author

*Geoff Huston* B.Sc., M.Sc., is the Chief Scientist at APNIC, the Regional Internet Registry serving the Asia Pacific region.

*www.potaroo.net*