The ISP Column
A column on things Internet

Geoff Huston
February 2017

The Root of the DNS

Few parts of the Domain Name System are filled with such levels of mythology as its root server system. Here I'd like to try and explain what it is all about and ask the question whether the system we have is still adequate, or if it's time to think about some further changes.

The namespace of the DNS is a hierarchically structured label space. Each label can have an arbitrary number of immediately descendant labels, and only one immediate parent label. Domain names are expressed as an ordered sequence of labels in left-to-right order starting at the terminal label and then each successive parent label until the root label is reached. When expressing a domain name the ascii period character denotes a label delimiter. Fully qualified domain names (FQDNs) are names that express a label sequence from the terminal label through to the apex (or "root") label. In FQDNs this root is expressed as a trailing period character at the end of the label sequence. But there is a little more than that, and that's where the hierarchal structure comes in. The sequence of labels, as read from right to left, describes a series of name delegations in the DNS. If we take an example DNA name, such as `www.example.com.`, then `com` is the label of a delegated zone in the root (Here we'll call a *zone* the collection of all defined labels at a particular delegation point in the name hierarchy.). `example` is the label of a delegated zone in the `com.` zone. And `www` is a terminal label in the `example.com.` zone.

Now before you think that's all there is to the DNS, then that's just not the case! There are many more subtitles and possibilities for variation, but as we want to look specifically at the root zone we're going to conveniently ignore all these other matters here. If you are interested, RFC1034 from November 1987 is still a good description of the way the DNS was intended to operate and the recently published RFC7719 provides a good compendium of DNS jargon.

The most common operation performed on DNS names is to *resolve* the name, which is an operation to translate a DNS name to a different form that is related to the name. This most common form of name resolution is to translate a name to an associated IP address, although many other forms of resolution are also possible. The resolution function is performed by agents termed resolvers and they function by passing queries to, and receiving results from, so-called *name servers*. In its simplest form, a name server can answer queries about a particular zone. The name itself defines a search algorithm that mirrors the same right-to-left delegation hierarchy. Continuing with our simple example, to resolve the name `www.example.com.`, we may not know the IP addresses of the authoritative name servers for `example.com.`, or even `com.` for that matter. To resolve this name a resolver would start by asking one of the root zone name servers to tell it the resolution outcome of the name `www.example.com.` The root name server will be unable to answer this query, but it will refer the resolver to the `com.` zone, and the root server will list the servers for this delegated zone, as this delegation information is part of the DNS root zone file for all delegated zones. The resolver will repeat this query to one of the servers for the `com.` zone, and the response is likely to be the list of servers for `example.com.` Assuming `www` is a terminal label in the `example.com.` zone, the third query, this time to a server for the `example.com.` zone will provide the response we are seeking.

In theory, as per our example, every resolution function starts with a query to one to the servers for the root zone. But how does a resolver know where to start? What are the IP addresses of the servers for the root zone?

Common DNS resolver packages include a local configuration fragment that provides the DNS names and IP addresses of the authoritative name servers for the root zone. Another way is to pull down the current root hints file from https://www.internic.net/domain/named.root.

But it may have been some time between the generation of this list and the reality of the IP addresses of the authoritative root servers today, so the first actions of a resolver on startup will be to query one of these addresses for the name servers of the root zone, and use these name servers instead. This is the so-called *priming query* (https://tools.ietf.org/html/draft-ietf-dnsop-resolver-priming).

This implies that the set of root server functions includes supporting the initial bootstrap of recursive DNS resolvers into the DNS framework by responding to resolver's priming queries, and anchoring the process of top-name name resolution by responding to specific name queries with the name server details of the next level delegated zone. It is a critical role in so far as if none of the root servers can respond to resolvers' queries then at some point thereafter, as resolvers' caches expire, resolvers will be unable to respond to any DNS queries for public names. So these root servers are important in that you may not know that they exist, or where they may be located in the net, but their absence, if that ever could occur, would definitely be noticed by all of us!

Moderating all considerations of the DNS is the issue of local caching of responses. For example, once a local resolver has queried a root server for the name `www.example.com.`, it will have received a response listing the delegated name servers for the `com.` zone. It this resolver were to subsequently attempt to resolve a different name in the `com.` zone then, for as long as the `com.` name servers are still held in the resolver's cache, the resolver will use the cached information and not query any root server. Given that the number of delegated zones in the root zone is relatively small (1,528 zones as of the start of February 2017), then a busy recursive resolver is likely to assemble in its local cache the name servers of many of the top level domain names and one would expect that it would have no further need to query the root nameservers. except as required from time to time to refresh its local cache, assuming that it is answering queries about DNS names that exist in the DNS.

In that respect, the root servers would not appear to be that critically important in terms of the resolution of names, and certainly not so for large recursive name servers that have a large client population and therefore have a well populated local cache. But this would not be a good conclusion. If a recursive resolver's cached information for a zone has expired, it will need to refresh the cache with a query to a root server. At some point, all of the locally cached information will time out of the cache, and then the resolver is no longer able to respond to any DNS query. To keep the DNS operating, recursive resolvers need to be able to query the root zone, so there is a requirement that collectively the root servers need to be always available.

In this respect the root servers "anchor" the entire DNS system. They do not participate in every name resolution query, but without their feed of root zone information into the caches of recursive resolvers the DNS would grind to a halt. So these servers are important to the Internet, and it might be reasonable to expect a role of such importance to be performed by hundreds or thousands of such servers. But there are 13 such root server systems.

Why 13?

The primary reason behind the use of multiple servers was diversity and availability. The root servers were each located in different parts of the network, within different service provider networks. The intended objective was that in the case where a DNS resolver was incapable of contacting one, two or even three root name servers, unless the resolver's site was itself isolated from the Internet, then the likelihood that it could not reach any of the root name servers was considered to be acceptably low.

Why not just use anycast and put as many root servers as we want behind a single IP address?

For a considerable time anycast was viewed with some caution and trepidation, particularly in the pre-DNSSEC time of a signed root zone. What would stop a hostile actor standing up a pseudo root server and publishing incorrect DNS information if the IP addresses used by the root servers could be announced from any arbitrary location? There was also some doubt that TCP would be adequately robust in such anycast scenarios. The original conservative line of thinking was that we needed multiple unitary DNS root zone servers, each with its own unique IP address announced from known points in the network fabric.

But needing "multiple" DNS Root Zone Servers and coming up with the number 13 appears to be somewhat curious. It seems such an odd limitation in the number of root servers given that a common rule of thumb in computer software design is Willem van der Poel's Zero, One or Infinity Rule, which states a principle that either an action or resource should not be permitted (zero), should happen uniquely (one) or have no arbitrary limit at all (infinity). For root servers, it appears that we would like more than one root server. But why set that number to 13?

The reason why may not be immediately obvious these days, but when the DNS system was being designed, the size limit of DNS responses using UDP was set to 512 bytes (Section 2.3.5 of RFC 1035). It seems a ludicrously small limit these days, but you have to also take into account that the requirement for IPv4 hosts was (and still is) that it accepts IPv4 packets up to 576 bytes in length (RFC791). Working backwards that would imply that if one takes into account a 20 octet IPv4 packet header and an 8 byte UDP header, then the UDP payload could be up to 548 octets in length, but no longer if you wanted some degree of assurance that the packet would be accepted by the remote host. If one also allows for up to 40 bytes of IP options, then this would mean that to ensure UDP packet acceptance under all circumstances then the maximal UDP payload size should be 508 octets. The DNS use of a maximum payload of 512 bytes is not completely inconsistent with this, but it is off by 4 bytes in this corner case!

This 512 byte size limit of DNS packets still holds, in that a query without any additional signal (i.e. a query that contains no EDNS(0) extension that provides a receiver UDP buffer size) is supposed to be answered by a response with a DNS payload no greater than 512 octets in length. If the actual response would be greater than 512 octets then the DNS server is supposed to truncate the response to fit within 512 octets, and mark this partial response as *truncated*. If a client receives a truncated response then the client may repeat the query to the server, but use TCP instead of UDP, so that it could be assured of receiving the larger response.

The desire in the design of the DNS priming query and response was to provide the longest possible list of root name servers and addresses in the priming response, but at the same time ensure that the response was capable of being passed in the DNS using UDP. The largest possible set of names that could be packed in a 512 octet DNS response in this manner was 13 such names and their IPv4 addresses - so there are at most 13 distinct root name servers to comply with this limit.

These days every root name server has an IPv6 address as well as an IPv4 address, so the DNS response that lists all these root servers and their IPv4 and IPv6 addresses is now 811 octets in size. If you also request the response to include the DNSSEC signatures, then this expands the response to 1,097 bytes in size. But if you pose a simple priming query to a root server without an EDNS(0) buffer size extension then you will still receive no more than 512 octets in response. Some root name servers provide the IPv6 addresses of root servers A through J in a 508 byte response, others give all 13 IPv4 addresses and add the IPv6 addresses of A and B in a 492 byte response. The remainder provide the IPv4 and IPv6 addresses for A through F and the IPv4 address of G in a 508 byte response. I suppose that the message here is that recursive resolvers should be supporting EDNS(0) these days and offering a UDP buffer size that is no less than 1,097 bytes if they want a complete signed response to a root zone priming query.

We are also entirely comfortable with anycast these days, and the root server system is now an enthusiastic adopter of anycast, where most of the root servers are replicated in many locations, and the overall result is that hundreds of locations host at least one instance of one of the root server anycast constellations, and often more. Part of the reason that our comfort level with anycast has increased is the use of a DNSSEC-signed zone, and recursive resolvers should be protecting their clients by validating the response they receive to ensure that they are using the genuine root zone data.

Should we do more? Do Root Servers make everything else better? Should we contemplate further expanding these anycast constellations into thousands of even tens of thousands of root servers? Should we open up the root server letter set to more letters? Is there a limit to “more” or “many”? Where might that limit be, and why?

These days the response recursive resolvers receive in 512 bytes or less is a partial view of the root name server system. From that perspective, 13 is not a practical protocol-derived ceiling on the number of distinct root server letters. Whether the partial response in 512 bytes reflects 6, 10 or 13 root nameservers out of a pool of 13 or 14 or any larger number is largely no longer relevant. The topic has moved beyond a conversation about any numeric ceiling on the letter count into a consideration of whether more root server letters would offer any incremental benefit to the Internet, as distinct from the current practice of enlarging the root server anycast constellations. Indeed, rather than more root name servers we should consider approaches to the DNS that can scale and improve resilience under attack through answering root queries but not directly involving these root name servers at all! In other words can we look at DNS structures that see the root servers as a distribution mechanism for the root zone data and use the existing recursive resolver infrastructure to directly answer all queries that relate to data in the root zone.

It's not clear that more Root Server letters or more Root Server anycast instances, or even both measures, make everything else better. Reducing the latency in querying a root name server has only a minimal impact for end users. The design objective of the DNS system is to push the data as close to the user as possible in the first place, so that every effort is made to provide an answer from a local resolver cache. It is only when there is a cache miss will the resolver query head back into the authoritative DNS server infrastructure, which would normally impact only a very small proportion of queries over time. The DNS derives its performance and efficiency through resolver caches, so that the overall intention is to query these root name servers to the minimal level possible. Secondly, a local root name server may not necessarily provide any additional name resolution resilience in the case of network isolation. Secondary root name servers also have an expiry time on the data they serve, and in the case of extended isolation the server will also time out a case to be able to respond. This is as true for the root zone as it is for any other zone. In many ways, the net impact of a local root name server on the internet experience of local users is minimal, and could well pass completely unnoticed in many cases.

In terms of the primary objective of the root name server system, diversity and availability, there is little to be gained by adding additional root name letters. It makes a complete priming response exceed 512 bytes, which means either forcing all priming queries into TCP, or dropping some named root servers from a non-EDNS(0) priming query response. But rather than butt our collective heads against the hard limits imposed by protocol specifications in some early RFCs, perhaps we are asking the wrong question. Rather than try to figure out how to field even more instances of root servers and keep them all current there is perhaps a different question: Why do we need these special dedicated root zone servers at all?

If the only distinguishing feature of these root servers is the proposition that any response with a source address of any of these 26 distinguished IP addresses is by simple unfounded assertion the absolute truth, then this is laughably implausible. Anyone who has experienced DNS interceptors would have to agree that that DNS lies are commonplace and lying in the DNS practiced by nation states as well as service providers across the entire Internet.

Enter DNSSEC.

The DNSSEC-signing of the root zone of the DNS introduced further possibilities to the root zone service to resolvers. If a resolver has a validated local copy of the current Key Signing Key, then it can independently validate any response provided to it from any signed zone that has a chain of signing back to this KSK, including of course any signed response about the root zone itself. A validating resolver no longer needs to obsess that it is querying a genuine root name server, and no longer needs to place a certain level of blind faith in the belief that its DNS queries are not being intercepted and faked responses being substituted for the actual response. With DNSSEC it simply does not matter in the slightest how you get the response. What matters is that you can validate these responses with your local copy of the root zone key. If you can perform this validation successfully, then the answer is much more likely to be genuine!

The use of DNSSEC casts the root server system in an entirely different light and the relationship between recursive resolvers and the root servers can change significantly.

One measure here is for resolvers to use so called "aggressive NSEC caching". This approach uses the NSEC records provided in the responses relating to the non-existence of a name to allow recursive resolvers to synthesise an authoritative NXDOMAIN responses for matching queries. Rather than caching a root zone NXDOMAIN answer for a single query, caching the NSEC response allows the recursive resolver to cache a common signed response for the entire span of query names as described in the NSEC response.

The observation here is that some 75% of responses from the root zone are NXDOMAIN responses (for example, http://stats.dns.icann.org/rssac/2017/01/rcode-volume/l-root-20170130-rcode-volume.yaml). Recursive resolvers could absorb much of the root server query load and answer these queries directly with NXDOMAIN responses were they to use this form of response synthesis.

Another approach is the use of local secondaries for the root zone. This approach is not an architectural change to the DNS, or at least not intentionally so. For recursive resolvers that implement this approach, this is a form of change in query behaviour in so far as a recursive resolver configured in this manner will no longer query the root servers for queries it would normally direct to an instance of the root, but instead direct these queries to a local instance of a slave server that is listening on the recursive resolver's loopback address. This slave server is serving a locally held instance of the root zone, and the recursive resolver would perform DNSSEC validation of responses from this local slave to ensure the integrity of responses received in this manner. In effect, this technique loads a recursive resolver with the entire root zone into what is functionally similar to a local secondary root zone server cache. For users of this recursive resolver there is no apparent change to the DNS or to their local configurations. Obviously, there is no change to the root zone either.

This proposal provides integrity in the local root server through the mechanism of having the recursive resolver perform DNSSEC validation against the responses received from the local root slave. If the recursive resolver is configured as a DNSSEC-validating resolver then this is configurable on current implementations of DNS recursive resolvers.

The advantage here is that the decision to set up a local slave root server or to use aggressive NSEC caching is a decision that is entirely local to the recursive resolver, and the impacts of this decision affects only the clients of this recursive resolver. No coordination with the root server operators is required, nor any explicit notification. The outcomes are only indirectly visible to the clients of this recursive resolver and no other.

Where does this leave the Root Server system? What's its future?

In the light of increasing use of DNSSEC the root server system is declining in relevance as a unique source of authoritative responses for the root zone, and we care forecast a time when their role would be largely anachronistic. A validated response can be considered a genuine response regarding the contents of the root zone regardless of how the recursive resolver learned this response. It is no longer necessary to have a dedicated set of name servers running on a known set of IP addresses as the only means to protect the integrity of the root zone.

It is also the case that the root servers are no longer being used as cache refresh for recursive resolvers for delegated domains. Today we see much of the time, effort and energy, and cost of root server operation being spent to ensure that NXDOMAIN answers are provided promptly and reliably. This really does not make any sense these days. The use of local secondary root servers and the use of NSEC caching can remove all of these resolvers' specific queries to the root servers, and what would be left is the cache priming queries. If all recursive resolvers were able to use either of these measures, the then the residual true role of the root server system would not be to respond to individual queries, but to distribute current root zone data into the resolver infrastructure.

But if the intention is to distribute signed root zone data to recursive resolvers then perhaps we could look more widely for potential approaches. Regularising the times that changes are made to the root zone would help to reduce opportunistic polling of the root servers to detect when a change might have occurred. Or using an approach based on incremental zone transfer (IXFR) that would allow recursive resolvers to request incremental changes to the root zone based on differences between zone SOA numbers may be more efficient. Maybe we can look further afield for additional ways to distribute the root zone contents. Social networks appear to be remarkably adept in their ability to distribute updates and a thought is that the small set of incremental changes to the signed root zone would be highly amenable to similar techniques or even using the same social networks. One can readily imagine a feed of incremental root zone updates on media such as Twitter, for example!

I also can't help but wonder about the wisdom of the root zone servers being promiscuous with respect to who they answer. Root zone query data points to some 75% of queries seen at the root zone servers generating NXDOMAIN responses. This means that three quarters of the responses from root servers are nonsensical questions in the context of the root zone. It's not clear to what extent the other 25% of queries reflect actual user activity. In an APNIC measurement exercise using synthetic domain names that included a time component, it was evident that more than 30% of the queries seen at the measurement's authoritative servers reflected 'old' queries, generated by query log replay or other DNS forms of stalking activities.

One way to respond to this is to farm out the query volume currently seen at the root servers into the existing recursive resolver infrastructure, so that all root zone responses are generated by these recursive resolvers, rather than passing queries onward to the root servers. If the root servers exclusively served some form of incremental zone transfer, and did not answer any other query type directly, then we would see a shift in query traffic away from the root servers as a crucial DNS query attractor, leaving only a lower profile role as a server to recursive resolvers.

There is much to learn about the DNS and there is still much we can do in trying to optimise the DNS infrastructure to continue to be robust, scalable and accurate – all essential attributes to underpin the continued growth pressures of the Internet.

Further Reading

Domain Names – Concepts and Facilities: https://tools.ietf.org/html/rfc1034

History of the Root Server System: https://www.icann.org/en/system/files/files/rssac-023-04nov16-en.pdf

RFC7720, DNS Root Name Service Protocol and Deployment Requirements:
https://tools.ietf.org/html/rfc7720

RFC7706, Decreasing Access Time to Root Servers by Running One on Loopback:
https://tools.ietf.org/html/rfc7706

Aggressive use of DNSSEC-validated Cache: https://tools.ietf.org/html/draft-ietf-dnsop-nsec-aggressiveuse

Workshop on DNS Future Root Service: http://www.potaroo.net/ispcol/2014-12/futureroots.html

DNS Terminology: https://tools.ietf.org/html/rfc7719

DNS RFCs: https://www.isc.org/community/rfcs/dns/

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

Disclaimer