The ISP Column
A monthly column on things Internet

April 2019
Geoff Huston

Expanding the DNS Root: Hyperlocal vs NSEC Caching

I guess that for me April has been DNS month, as here is another article on the Domain Nam e System. This time I would like to look at the efforts to expand the root zone service by enabling recursive resolvers.

The root zone of the DNS has been the focal point of many DNS conversations for decades. One set of conversations, which is a major preoccupation of ICANN meetings, concerns what labels are contained in the root zone. A separate set of conversations concern how this root zone is served in the context of the DNS resolution protocol. In this article I'd like to look at the second topic, and, in particular, two proposals to augment the way the root zone is served to the DNS.

Serving the Root Zone

For reasons that are buried in a long-gone history that relates to aspects of UDP, the IPv4 packet size specification, and IPv4 packet fragmentation and reassembly behaviours, there are 13 distinctly named nameservers for the root zone's nameserver set. At that time there were also just 13 distinct physical machines that served the root zone.

Why 13?

IPv4 defines a *minimum reassembly packet size*. All IP hosts and routers must be able to pass a packet up to this size without the intended recipient discarding the packet because it was too large. A host may accept or discard packets larger than this size, but if it is larger than this size then it is up to the host.

This *reassembly packet size* is subtly different from the *minimum transmission size*. The *minimum transmission size* refers to the packet size that hosts and network devices must be capable of processing without packet fragmentation.

In IPv4 the *minimum transmission size* is 68 octets, and the *minimum reassembly packet size* is 576 octets. To quote RFC 791: "Every internet module must be able to forward a datagram of 68 octets without further fragmentation. This is because an internet header may be up to 60 octets, and the minimum fragment is 8 octets. Every internet destination must be able to receive a datagram of 576 octets either in one piece or in fragments to be reassembled." While it seems ludicrously small these days the maximum IPv4 packet size that all IPv4 hosts must be capable of receiving is still 576 octets. (The reason for the selection of 576 bytes is unknown to me.)

IPv6 defines the *minimum reassembly packet size* of 1,500 octets and the *minimum transmission size* of 1,280 octets.

The DNS specifies (in RFC 1035) that the maximum size of any DNS packet that uses UDP is 512 octets (there is no such limit specified for TCP responses).

While the extension mechanisms for DNS (EDNS(0), RFC 6891) allow a DNS client to signal that it is capable of receiving a much larger UDP DNS packet, if a query does not have a UDP buffer size specified in the EDNS(0) extension, then the responder must fit the DNS response into 512 octets.

The DNS query to learn the servers for the root zone and their IP addresses is a critical query for DNS resolvers. While resolvers are normally configured with a static root hints file that lists the servers, there is no assurance that this is the current set. Therefore, the first action of a resolver is to ask a root server for the current set of servers. It requires a response to get past this initial step.

In order to increase the level of certainty that any DNS resolver would receive a response to this query, it is necessary to fit the response to this query in 512 octets. (Thus, fitting the response in a single unfragmented UDP packet that is no larger than 580 octets. The reason for this 4-octet discrepancy with IPv4's *minimum reassembly packet size* of 576 octets is also unknown to me.) This is possible if we use only 13 distinct name server names and only provide their IPv4 addresses in the response. Hence the reference to 13 distinctly named servers for the root zone.

These days there are IPv6 and IPv4 addresses for all 13 root servers, and it takes some 811 octets to pass the complete root server set in a DNS response. The relationship between 13 servers and 512 octets is no longer an overriding limitation, and if you specify a maximum of 512 octets in the response, a root server will either curtail the response to pass back a partial set that fits within a 512 octet response or truncate the response to signal that the client should re-query using TCP. Of course, once you use TCP the entire 512 octet size limit is irrelevant. Either way 512 octets and 13 distinctly named root servers is somewhat anachronistic these days.

In the same way that the notional IPv4 *minimum reassembly packet size* is 576 octets because 'history', there are 13 distinctly named root servers because 'history'.

These days the root zone's server infrastructure has grown significantly. Most of the root servers have adopted *anycast server collections*. The anycast concept is very simple: Rather than fielding one server with a unique IP addresses, the server operator may field a number of servers, reusing the same IP address for each of these servers. It is left to the routing system to ensure that a packet that is addressed to this anycast address is sent to the 'closest' location. Anycast has been enthusiastically adopted by the root servers. There are reported to be some 1,228 locations that host an anycast instance of the root zone (https://root-servers.org).

Why have we spent so much effort to make the root zone so available? No other zone in the DNS is the subject of so much engineering attention to ensure its resilience and speed. Why is the root zone so special? My take on the answer to these questions is that the DNS resolution protocol is actually incredibly simple-minded! Every name resolution query starts with a query of the root zone. Yes, that's every query!

Now if this statement was literally true then the 1,228 root servers would have long since turned to molten silicon under the query stress. What saves the root servers from fatal overload is that DNS resolvers cache the answers that they receive and use their local cache wherever possible. Most of the queries to the root zone are 'virtual' queries, as they are answered by the local resolver's cache of previous responses. What the root servers do is to periodically refresh the recursive resolvers' locally cached data.

This cache behaviour dramatically cuts down the query traffic volumes seen by root zone servers, but it doesn't alter the central role of the root zone to the entire DNS. If a recursive resolver cannot contact a root server for the lifetime of the locally cached data, then the local resolver will lose its cache of root data. At that point the resolver would be incapable of resolving any DNS name whatsoever. A consistently available root service is essential for the Internet.

Given this critical role of the root zone and the root zone's servers for the reliable operation of the Internet's name space, are some 1,228 instances of root servers enough? Should we further increase the number of root servers? Should we further increase the size of these anycast constellations? Or both? Anyone who has observed root server discussions in the past couple of decades would immediately appreciate that these are not necessarily engineering queries but have considerable political overtones. "[*insert country name here*] wants its own root service" is a constant refrain, and our inability to collectively accommodate all these demands and still maintain a coherent and efficient DNS has resulted in a level of paralysis in the root server system.

This paralysis is unhelpful. On the Internet it is always easier to attack than to defend. An ever-growing Internet of Trash makes launching an overwhelming attack on the DNS root server system a viable option. Populating the root server anycast sever sets with more and more servers is never going to keep pace with the increased capability of DDOS attacks. Our ability to appropriate ever-increasing resources to support an otherwise unfunded root service is a matter of continuing concern.

This thinking has directed us into a path of lateral thinking about ways to augment root service capacity. Are there other ways to make the DNS root zone more resilient?

In looking at this question this is where hyperlocal roots and NSEC caching provide alternative approaches.

Hyperlocal Roots

The first is a technique commonly called *hyperlocal*, described in RFC 7706. The underlying concept is that the DNS infrastructure is a loosely coupled system. While domain is served by a set of 'authoritative' servers, it can also be served, either in full or in part, by other servers outside this authoritative server set.

If a resolver responds to a query from its local cache, then this is an example of such a served response. The conventional way of managing this local cache of responses is on an on-demand basis where the responses to queries populate the local cache. But there is also another approach, namely by using a copy of a DNS zone and using this copy in a manner analogous to the local cache.

Why not just use caching? Why expend the effort to maintain a local copy of the entire zone? The utility of this 'full zone' local cache lies in the treatment of queries for non-existent domain names. There are infinitely many names that do not exist in a zone, and if an on-demand caching resolver receives a series of queries for non-existent names then the demand cache will not be useful, whereas a "full zone" local cache will equally serve queries for existing and non-existent names in the zone.

When applied to the approach of using resolvers to serve the root zone, this approach requires the resolver to fetch a copy of the entire root zone. The suggested method is to use the DNS AXFR command, which will transfer the zone in its entirety (`dig . AXFR @f.root-servers.net`, for example). The local resolver is then configured to serve this zone as an authoritative server. As long as the resolver is careful to manage the root zone in a restricted view that is only accessible to the resolver itself then the result is that the resolver behaves as if it was an authoritative for the entire root zone to its served client base.

How do you know that the copy of the root zone is the genuine copy? While each entry in the root zone is DNSSEC-signed, the AXFR command is unprotected, and the root zone itself is unsigned it its entirety. It would be useful if the root zone was signed with a digest (such as described in the internet draft: draft-wessels-dns-zone-digest), allowing the resolver to check that the zone it has just transferred is the genuine zone. If this approach were to be widely adopted zone digest signing for the root zone seems like an obvious requirement.

There is also the issue of currency. Those resolvers that use such a *hyperlocal* cloned copy of the root are not notified when changes are made to the root zone, so they need to regularly perform a consistency check against a root server to ensure that they are serving the current zone contents.

In terms of configuration, this *hyperlocal* configuration is not a simple setting, or at least not yet. Resolvers need to set up the slave secondary configuration, list the servers for the zone and enable a restricted view to the zone.

Interestingly, Bind 9.14 will simplify this into a single configuration line:

```
        zone "." { type mirror; };
```

You might think that this configuration command would permit any zone to be configured as type mirror, but that would be an incorrect assumption. Bind 9.14 will have a built-in list of AXFR server targets for the root zone. Serving other zones in this manner requires a more detailed configuration, or due to the lack of AXFR support, might not be an option at all.

The *hyperlocal* approach appears to have engendered a recent revival in interest, and RFC 7706 is being updated with some further configuration examples (https://tools.ietf.org/html/draft-ietf-dnsop-7706bis-03).

NSEC Caching

The second approach is aggressive NSEC caching, described in RFC 8198. The vulnerability that the DNS has in this space is a *random name* attack where the attack consists of posing an overwhelmingly large number of queries for non-existent names in the same zone. The attack is effective in the form of a widely distributed attack, where each element of the co-opted zombie set generates just a few random name queries per second, which sits well below a conventionally detectable query threshold. The attack becomes toxic at the point where the number of zombie attackers encompasses millions of systems.

If the query name was fixed, the caches in the recursive resolver population would be able to answer the queries, and therefore protect the authoritative name server from overload. The use of random name strings in the attack defeats these conventional caches as the recursive resolver will pass the random name queries to the authoritative name server.

Here is where DNSSEC can help. If the zone is DNSSEC-signed, as is the root zone, and the resolver both requests DNSSEC credentials and performs DNSSEC validation, then the response received from the authoritative name server for a non-existent domain name contains more information than was strictly requested, in order to facilitate the resolver to perform validation of this response. The NSEC response describes a range of labels that are not defined in the zone. It does so by listing in the signed response the two labels in the zone that minimally span the space encompassed by the non-existent label. If this label range is cached by the recursive resolver, rather than just the queried label, then the same NSEC response can be used to respond to subsequent queries that relate to any label that sits within the same span of labels within this zone. When handling queries related to a random name attack, a recursive resolver that caches these NSEC responses it will rapidly "learn" the entire contents of the target zone, and at that point will have no further need to consult the authoritative name server until these cached NSEC entries expire.

Both NSEC and NSEC3 signed zones can be cached in this matter. While NSEC uses a simple ascii ordering of the zone contents to determine the NSEC "gaps", NSEC3 requires the labels to be hashed prior to the ordering function. In all other respects they function in the same manner.

To understand just how dramatic the change could be for the DNS, it has been observed that some 70% of all queries to the root name servers relate to non-existent top-level names. If all recursive resolvers performed DNSSEC validation and aggressive NSEC caching of just the root zone by default, then we could drop the query rate to the root servers by most of this 70%, which is a dramatic change.

Some further advantages of this NSEC caching approach include:

- It can be used for all DNSSEC-signed zones, rather than only the root.
- It requires no special configuration of the recursive resolver.
- It requires no special support on the part of authoritative name servers.
- It uses conventional DNS queries and responses.
- DNSSEC validation is used to validate the NSEC responses.
- It assists in cases of random name attacks by both increasing the efficiency of the local cache and absorbing query traffic that would otherwise be directed to the authoritative name server.

Which?

Both of these approaches have the potential to improve the availability and resilience of the root zone in the DNS. Which is better?

Both approaches increase the availability and resiliency of the zones that are locally served, and both approaches can play a valuable role in the defusing of certain forms of random name attacks on the authoritative server infrastructure of the DNS. In these respects, the two approaches are equally effective.

However, there are some obvious differences between the two approaches.

I find the *hyperlocal* approach unduly fussy. It requires special configuration. It uses a special query to pull the zone. It requires another novel mechanism to check the integrity of the pulled zone. There is no automated control of cached zone refresh, nor is there any explicit notification of a zone update. The approach also requires explicit support from the zone, and also requires specific configuration for each zone that is served in this matter.

However, the cached copy of the root zone looks and smells like a locally served root zone, and I guess the proponents of this approach would like to think that this approach might calm down some of the more strident politically inspired calls for opening up the root server environment for new root servers. I am nowhere near as optimistic!

NSEC caching appears to me to be streamlined by comparison. It requires no new queries. It has the cache managed in a conventional manner using TTLs. It can be applied to all DNSSEC-signed zones, not just the root zones. It is demand based, in that the local cache of the zone is assembled as needed, rather than maintained continually just in case.

Personally, I strongly prefer the NSEC caching approach.

Disclaimer

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