The ISP Column
A column on things Internet

September 2023
Geoff Huston

Notes from OARC 41

OARC held a 2-day meeting in September in Danang, Vietnam, with a set of presentations on various DNS topics. Here's some observations that I picked up from the presentations that were made that meeting.

Deploying ZONEMD in the Root Zone

As a distributed database, the DNS works through the piecemeal distribution of data in a *just in time* methodology. The DNS name resolution protocol answers specific queries, and the answers contain information pertinent to the query and nothing more. The DNS tries not to overshare. But if that was all it did, then it would not still be running today. If every query started with a query to the root zone for the name servers for the relevant top-level domain, and then a sequence of queries as the querier traversed down through the delegation hierarchy, then each DNS resolution operation would take an impossibly long time and the query load at the servers of the apex of the DNS tree would be unsupportable. The DNS resolution protocol is implemented as a hybrid of on-demand response generation and *just in case* caching. As data is passed through the network of intermediaries (recursive resolvers), it is cached on an opportunistic basis. When successive queries are processed by these recursive resolves, the cache is consulted to see if the response can be provided directly. It is this caching that has allowed the DNS to scale.

Even so, the twin factors of the growth of the Internet and the concentration of queries at the upper parts of the name hierarchy has meant that the scaling pressures are felt most acutely at the apex of the DNS, the root zone.

Our response to these scaling pressures has been to exploit anycast to increase the number and capacity of servers that serve the root zone. There are currently some 1,754 distinct instances of root servers on the Internet (https://root-servers.org), serving in aggregate some 100 billion queries per day (https://rssac002.root-servers.org).

The growth of the network is not going to slow down anytime soon, so the question is: How do we continue to scale this system? Do we just keep on adding more and more root instances to the anycast clouds with no clear end in sight? Or can change course and enlist some help from DNS intermediaries, namely the millions of recursive resolvers out there? This latter approach is described in RFC7706, "Decreasing Access Time to Root Servers by Running One on Loopback". Rather than have the recursive resolver cache the responses to individual queries to the root zone, the resolver is configured to fetch the entire root zone and load this into the local cache. The true utility of this local cache of the entire root zone 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 on-demand cache will not be useful, whereas a local cache of the entire root zone 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.

But how can a recursive resolver be assured that this local copy of the root zone is a genuine copy of the most up-to-date version of the root zone? 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 entire root zone was signed with a digest, as described in RFC 8976. This would allow the resolver to check that the zone it has just transferred is the genuine zone.

Verisign's Duane Wessels presented to OARC the schedule of actions to introduce this ZONEMD record to the root zone. On 22 September the ZONEMD record was first seen in the root zone:

```
. 86400 IN ZONEMD 2023092502 1 241 4227BDE7C956255A3564A4D98F25D60DF3285F9FAA16100CC623B2E3BEC6593F
                                   1BF33B261A7C1CA7258813C695AF42EF

. 86400 IN RRSIG ZONEMD 8 0 86400 20231008210000 20230925200000 11019 . ps2qM5vIC8dlmzR+VyxPqrtm/R6YgK
                                   hPPFqcBx+OwzpFkr9F2CqxzrxDCBw4vUp2jkFL/pMGmBHGhWUngn5cXAVxNFGWmN
                                   UHbmxp9h6S4qiPSOEunhgtfTHc10cFHh31THEwyW1bgOIrFVijuQlsXUCLHq1zUs
                                   Xn4fZkoaAXZSeuk1dE2/g8ORRtDdaAGGNFrzyQZjiP4g/mt+Z0N3dgPp/TjXId6+
                                   1H14R78HCsZMqdLsT7+f2hU6cWUUEKLXdDs7tbbb5L7q2JEEjFpbrEPwO6YiSPRG
                                   ikB+oyvCXXLdAMijwzF85R898POGcdNm8zyT0h3whUaiGKl7TY39PrGQ==
```

This is a *false* record, which is deliberately unverifiable, using a private-use hash algorithm. The intention here is to ensure that those clients that make use of a full root zone file are not disrupted by the appearance of a new resource record type in the root zone file. Given that this is the first introduction of a new record type into the root zone in 13 years there is some understandable caution associated with this change.

On the 6th December 2023 it is planned to change the hash algorithm to SHA-384, which will then be an accurate hash of the current copy of the root zone file.

There is also the issue of currency. Those resolvers that use such a cloned copy of the root zone 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. However, it is noted that the signed SOA record in the root zone changes every 24 hours:

```
./20/root:. 86400 IN SOA a.root-servers.net. nstld.verisign-grs.com. 2023091903 1800 900 604800 86400
./21/root:. 86400 IN SOA a.root-servers.net. nstld.verisign-grs.com. 2023092002 1800 900 604800 86400
./22/root:. 86400 IN SOA a.root-servers.net. nstld.verisign-grs.com. 2023092103 1800 900 604800 86400
./23/root:. 86400 IN SOA a.root-servers.net. nstld.verisign-grs.com. 2023092203 1800 900 604800 86400
./24/root:. 86400 IN SOA a.root-servers.net. nstld.verisign-grs.com. 2023092301 1800 900 604800 86400
./25/root:. 86400 IN SOA a.root-servers.net. nstld.verisign-grs.com. 2023092401 1800 900 604800 86400
./26/root:. 86400 IN SOA a.root-servers.net. nstld.verisign-grs.com. 2023092502 1800 900 604800 86400
```

A resolver client can use this SOA data to provide some assurance that the zone file they are using it is using is a recent one.

DNSSEC and Cryptography

The addition of cryptographic information to the DNS to allow DNS clients to gain some additional assurance that the response that they have received is authentic, complete, and current, namely DNSSEC, is becoming a protracted story of proportions similar to that of IPv6 adoption. While the adoption of DNSSEC validation has been proceeding with some visible impetus over the years, the uptake of zone signing is a mixed story. (I explored this topic in a recent article, https://www.potaroo.net/ispcol/2023-09/dnssec-queries.html, so I won't repeat myself here!)

There are a number of reasons for the reluctance to sign DNS names, and part of this lies in the arcane area of cryptography. A very common cryptographic algorithm is RSA (named after the surnames of the original authors of the 1978 paper that described the algorithm, Ron Rivest, Adi Shamir and Leonard Adelman). RSA is a cryptographic algorithm that does both encryption and decryption using a variable key length. A shorter key is more efficient in terms of encryption and decryption but is not as robust. Longer keys are more expensive to use but offer greater robustness against efforts to break encoded data.

RSA is based on prime number operations using modular exponentiation. A basic principle behind RSA is the observation that it is practical to find three very large positive integers **e**, **d**, and **n**, such that with modular exponentiation for all integers **m** (with $\mathbf{0} \leq \mathbf{m} < \mathbf{n}$): $(\mathbf{m}^{\mathbf{e}})^{\mathbf{d}} \equiv \mathbf{m} \bmod \mathbf{n}$. Even if you know the values of **e** and **n**, and even **m**, it can be extremely difficult to find the value of **d**.

The underlying premise here is that prime number factorisation of a very large composite integer can be hard, and if the number is the product of two very large prime numbers, then that task can be exceptionally hard. We've not devised (as yet) any better approach other than brute force enumeration.

For example, the workload to factor a composite number that is the product of two prime numbers that are 512 bits long in binary notation was estimated in 1995 to take around 30,000 years using a 1 MIP (millions of instructions per second) computer. At that time, this scale of an infeasible challenge was a good match for that computing environment. (This threshold of feasibility was still considered reasonable some 8 years later, as I have a security practices publication from 2002 that observed that RSA using 512-bit keys was a practical profile for all but the most extreme security applications of 2002.) If you increase the computational capacity to 10 billion instructions per second then the problem is a 3-year computational problem. Obviously, this is a feasible task in today's computing environment.

To help understand the relative strength of cryptographic algorithms and keys there is the concept of a security level which the log base 2 of the number of operations to solve a cryptographic challenge. In other words, a security level of n implies that it will take 2^n operations to solve the cryptographic challenge. Table 1 shows the security level for various RSA key lengths.

Security Level	RSA Key Length (bits)
80	1,024
112	2,048
128	3,072
140	4,096
192	7,680

Table 1 – Security level of various RSA key sizes

Things change in this space, and these days estimates of a minimal acceptable security strength that will provide protection across the coming decade points to the use of SHA-256 in conjunction with RSA with 2,048-bit keys, or a security level of 112 bits. However, if we want to encrypt data today with a protected secure lifetime of greater than 10 years then it looks like we may be looking at SHA-384 and RSA with 4,096-bit keys, or in other words a security level of more than 128 bits.

Using larger keys in crypto has several implications when we are talking about the DNS. Larger keys mean larger DNS signatures and larger payloads, particularly for the DNSKEY records. A comparison of key sizes and DNSSEC signature record sizes is shown in Table 2.

Algorithm	Private Key	Public Key	Signature	Security Level
RSA-1024	1,102	438	259	80
RSA-2048	1,776	620	403	112
RSA-4096	3,312	967	744	140

Table 2 – Crypto Sizes (Bytes)

Larger key sizes also imply that it takes more time to both sign and validate signatures. Table 3 shows the elapsed time taken to sign a zone with 500K entries, using OpenSSL 1.1.1k libraries on a FreeBSD 12.2 host with the DNSSEC toolset supplied with Bind 9.16.16. Validation time is elapsed time for completing 50K queries with DNSSEC validation, and for comparison I've included the time taken for the same set of queries into an unsigned zone. The absolute time intervals are not that important here, but the relative differences in time when using the different crypto algorithms and key sizes is important. RSA adds to the signing time at a rate that rises at a higher rate than the key size (i.e. double the key size in RSA takes more than double the time). (Table 3).

Algorithm	Signing Time (secs)	Validation Time (secs)
Unsigned		905
RSA-1024	52	1,168
RSA-2048	126	1,173
RSA-4096	830	1,176

Table 3 – DNSSEC timings (seconds)

Using RSA with 4,096-bit keys implies taking more time to sign and validate these signatures (although the other overheads associated with fetching the validation records tend to swamp the crypto processing time in this simple experiment). It also implies that DNS responses will be larger. Larger DNS records in UDP means dancing around the issues of IP fragmentation, UDP buffer size settings, truncation of responses and fallback requests via TCP, all of which create a far slower DNS resolution operation.

An obvious answer is to avoid using large RSA keys. In this case RSA with 4,096-bit keys encounters a visible level of resolution failure, due to the inconsistent handling of the larger DNS responses for the DNSKEY record, and the current practice appears to point to the use of 2,048-bit keys as a suitable security choice for the moment. The problem is that the crypto environment is a moving target and over time smaller RSA keys will be more vulnerable as we develop more capable computers. Another answer is to use a "denser" crypto algorithm that has a high security level with a far smaller key size than RSA. Here the Elliptical Curve algorithm, ECDSA P-256, is an obvious contender.

Briefly, ECDSA is a digital signing algorithm that is based on a form of cryptography termed "Elliptical Curve Cryptography". This form of cryptography is based on the algebraic structure of elliptic curves over finite fields. The security of ECC depends on the ability to compute a point multiplication and the inability to compute the multiplicand given the original and product points. This is phrased as a discrete logarithm problem, solving the equation $\boldsymbol{b^k = g}$ for an integer **k** when $\boldsymbol{b}$ and $\boldsymbol{g}$ are members of a finite group. Computing a solution for certain discrete logarithm problems is believed to be difficult, to the extent that no efficient general method for computing discrete logarithms on conventional computers is known. The size of the elliptic curve determines the difficulty of the problem.

Both **.com** and **.net** zones are signed with a 2048-bit Key-Signing Key (KSK) and a 1280-bit Zone-Signing Key (ZSK). Shifting to ECDSA P-256 will reduce the signature size in the signed records, which will help both in fitting signed responses into UDP messages and reducing the memory footprint of cached signed responses. At the same time the security level of ECDSA P-256 is the equivalent of a 3072-bit RSA key.

Algorithm	Private Key	Public Key	Signature	Security Level
RSA-2048	1,776	620	403	112
ECDSA P-256	187	353	146	128

Table 4 – ECDSA Crypto Sizes (Bytes)

Transitioning DNSSEC algorithms can be challenging. The approach being following by Verisign here is a conservative one, using a dual-algorithm approach (as distinct from an abrupt cutover). Their plan is to populate the domain with a second set of RRSIG records that are generated using the ECDSA ZSK value, then adding the ECDSA ZSK and KSK records to the zone DNSKEY set. The root zone then

published as new DS record, which is the hash of the ECDSA KSK value. The old RSA DNSKEY values can then be removed, followed by removal of the RSA-signed RRSIG records.

The timetable is shown in Table 5.

Zone	start	roll	end
Edu	6 Sep	12-15 Sep	22 Sep
Net	25 Oct	31 Oct - 3 Nov	10 Nov
Com	29 Nov	5-8 Dec	15 Dec

Table 5 – Schedule for Algorithm Rool for .edu, .net and .com

At the time of writing this article, the **.edu** algorithm roll has completed.

Of course, these estimates of algorithm strength don't factor in the impact of any future use of quantum computing. If one views the onset of quantum as a "when", rather than an "if" question then if would be prudent to prepare our cryptographic tools for a world that includes quantum computing. A number of groups are working one this, including the US through the PQC project at NIST's Computer Security Resource Centre (https://csrc.nist.gov/projects/post-quantum-cryptography).

A better form of Client Subnet?

The DNS is used extensively as a replacement for BGP to perform content steering (see DNS is the new BGP for a longer description of this). The problem is that DNS-based content steering assumes that an end user is located close to the recursive resolver that handlers their DNS queries, and when this is not the case (such as with the use of open recursive resolvers) then the content steering can make some pretty sub-optimal decisions.

The answer that the resolver and content folk can up with was to attach the IP address of the client to the query, so that when the recursive resolver made the query to the authoritative server the client's identity was attached to the query. From a privacy perspective this is a horrifying breach of user privacy. The specification made some attempt to mitigate the potential problems by using a subnet rather than the cull client address, but this is a relatively poor mitigation. The RFC describing this procedure, EDNS Client Subnet (ECS), RFC 7871, has some disparaging commentary on this process: "We recommend that the feature be turned off by default in all nameserver software." Pretty stern stuff for a technical specification!

The presentation by AdGuard's Andrey Meshkov proposes a different approach to what is encoded in a DNS query's Client Subnet field that is intended to improve the privacy position of this signalling mechanism while still providing the authoritative service with sufficient information to perform content steering via the DNS.

Despite the obvious privacy concerns with the use of Client Subnet in DNS queries, AdGuard report that about two thirds of the queries received on AdGuard's open resolver are for domains that support ECS.

How can a recursive resolver still provide some geolocation information to the authoritative DNS server for domain names that wish to receive such information to inform optimal content steering outcomes, but not reveal the querier's network?

The folk at NextDNS come up with an innovative idea back in 2019. They adopted a solution to these ECS issues by substituting client's subnet with another subnet that would be shared by all clients from the same approximate location and ISP. That way they were not providing any data that was related to the querier but did provide the same geolocation information. They also observed that many of the domain names that use ECS are in fact providing exactly the same response no matter what ECS data is provided, which serves no purpose in terms of content steering but cripples the resolver's cache

performance. In this case they determined to drop the ECS data for domains that did not use it. They noted that these efforts achieved an outcome of a 75% cache hit rate for their resolver service.

AdGuard are using a similar approach in their resolver. In their case they use the BGP routing table to map the client IP address to the origin AS, and then selecting a subnet from all the address prefixes originated from that origin AS as being representative of all such users. This approach restored a cache hit rate of around 76%, but the cache itself was some 5 times larger than a non-ECS cache.

The next refinement was to group together the geolocation data for the same country and use a single subnet for all users in that country. AdGuard struck the same issue as the earlier NextDNS effort where of the 1,000 most popularly queried domains 50% indicates support for ECS, but 15% returned an invariant response irrespective of the ECS information provided in the query.

The interim approach AdGuard are using at the moment is to apply a finer-granularity of subdivision for the most populous countries, and use a country level granularity for all others. This gives then a good cache efficiency, a contained cache size and an equivalent content steering outcome.

The DNS TTL Field

The DNS depends on extensive caching of DNS responses for good performance. Every DNS zone administrator sets a TTL value for each record in the zone, which guides a caching resolver as to how long a DNS response should be held in the resolver's local cache. There are a number of conflicting requirements as to what value to use for a cache lifetime. Extended TTL values offload the name resolution workload to the recursive resolvers, making name resolution faster, while shorter TTL values assist in the agility of configuration and reduce the time to repair a zone configuration problem.

The TTL value is not a strict value but is intended to act as an upper bound to a caching resolver. However, there is conflicting advice to caching resolvers to retain a cached value beyond its TTL lite in RFC 8767, "Serving Stale Data to Improve Resiliency". Experimental observations also show that a significant set of recursive resolvers retain cached data beyond the TTL (https://indico.dns-oarc.net/event/46/contributions/987/attachments/941/1743/oarc40.pdf). Perhaps the most surprising observation has been the case when zone administrators have lowered their TTL, anticipating a proportionate rise in the query volume, yet no such change was observed (https://ripe84.ripe.net/wp-content/uploads/presentations/85-RIPE84_DNS_Update_bd.pdf, https://www.root.cz/clanky/jak-se-projevilo-snizeni-ttl-v-zone-cz/).

These observations support the hypothesis that extended TTL values do not appear to be strictly necessary in many cases, and reducing TTL values, such as from 1 hour to 15 minutes may not negatively impact the resolution performance of the name, yet the shorter TTL values have a benefit in permitting facter recovery from various configuration errors in published zones.

So yes, it still is the case that caching benefits DNS performance, but the relationship between changes in cache TTLs and changes in DNS resolution times and query loads are by no means simple in today's DNS.

Polling does not Scale

If there is one factor that lies behind every issue, we see on the Internet its scale. The inexorable impact of Moore's Law with the production of silicon chips means that computing continues along a path of lower cost and greater capability. We use these devices in ever increasing volume, and this impacts the demands we place on the network infrastructure. It means increased pressure on network address consumption, increased pressure on the scale of the Internet's routing system, increased volumes of DNS queries, and an increased number of DNS names out there.

The DNS was built at a time where there was considerable focus on resilience and availability in the face of unreliable connectivity infrastructure. The DNS reflects this though the use of multiple authoritative

servers and multiple recursive resolvers. But does this scale? This question was asked by nic.at's Klaus Darilion in the context of primary and secondary DNS servers.

Primary and Secondary servers are loosely coupled in the DNS. The secondary is meant to periodically query the primary at a regular interval (the maximum interval is defined by the REFRESH value in the zone's SOA record) for the SOA record's serial value. If the secondary holds a lower SOA value in its local copy of the zone, then it initiates a zone transfer (XFR) to get the primary server's version of the zone. This zone refresh can also be triggered if the primary sends a NOTIFY to the secondary. It is left to the secondary to ensure that the zone refresh is successful, and the primary never checks to ensure that its secondaries are in sync with the primary version.

All this has worked tolerably well for decades, but there are scaling issues lurking here. The scenario explored in this presentation is the case where a secondary server is undertaking that role for a million zones. If the secondary wants to be moderately responsive to changes in any of the served zones, then it might use a 10 second SOA query interval. This is 100,000 queries per second, which is a challenge for a single DNS platform.

Some attention is being given to alternative approaches to detecting changes in the primary zones that do not rely on SOA polling. A secondary could use the SERIAL property in catalogue zone or add the ZONEMD value into the catalogue zone data.

It seems to me that polling is never a scalable solution. Thrashing away at the primary source just to see if anything has changed can work at small scale but imposes significant loads at higher scale. If you want a responsive system at scale, then you need to head into the world of explicit notification and use a low frequency polling mechanism to catch the case of missed notifications.

OARC 41

The full workshop program, and all the presentations from the workshop can be found at the OARC 41 web page.

Disclaimer

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

Author

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

www.potaroo.net