February 2025
Geoff Huston

Notes from OARC 44

The DNS Operations, Analysis, and Research Center (DNS-OARC) brings together DNS service operators, DNS software implementors, and researchers together to share concerns, information and learn together about the operation and evolution of the DNS. They meet between two or three times a year in a workshops format. The most recent workshop was held in Atlanta, in February 2025. Here are my thoughts on some of the material that was presented and discussed at this workshop, where too much DNS is barely enough!

DNS Traceroute

The two *goto* tools in identifying unusual network forwarding behaviour are *ping* and *traceroute*. It's not just in exposing packet forwarding paths that *traceroute* can be useful, but it has its uses in the DNS as well, as Davey Song from Alibaba Cloud described.

Conventional *traceroute* uses a technique of sending a sequence of *ICMP Ping* packets with increasing Time To Live (TTL) values in the IP header. Each time a packet is processed by a router, the TTL value in the IP header is decreased. When the packet's TTL reaches a value of zero, the router will stop forwarding the packet and generate an *ICMP Time Exceeded* message and pass it back to the originator. By sending packets with TTL values of 1, 2, 3 and so on, and looking at the addresses of the routers who are responding with these *ICMP Time Exceeded* messages it's possible to construct a hop-by-hop path to a given IP destination.

This technique can be applied to any form of IP packet, including TCP and UDP packets. Indeed, the payload can be a UDP packet containing a well-formed DNS query. This is particularly relevant to detection of various forms of on-path DNS blocking, where an intermediate system responds to a DNS query with either a synthetic NXDOMAIN response, or responds with a substituted response.

It's helpful to start with an ICMP *traceroute* to the intended server or resolver being queried, which provides the number of hops from the test point to destination IP address and the path to the destination. Then a sequence of DNS queries with an increasing TTL can be sent used to the same destination. If there is a case of an active intermediary performing selecting query blocking or substitution, then this test will reveal a DNS response coming back from a query with a lower TTL than is otherwise required to reach the destination, assuming that the queried name is a name that is being blocked by the intermediate system.

A convenient way to construct such DNS packets is by using *scapy* (https://scapy.net/) to construct the DNS traceroute packets.

This *DNS traceroute* approach can detect instances of DNS interception but cannot prevent it. If you are looking for tools that provide greater levels of protection against such forms of interception then encryption and authentication techniques, such as DNS over TLS, DNS over HTTPS and Obfuscated DNS, can hide the DNS traffic from the network (and from any intercepting intermediaries) and authenticate the identity of the remote nameserver.

Presentation: Who Forged my DNS Answers?

DNS Servers: Virtual vs Bare Metal

At the upper levels of the DNS namespace hierarchy there is an increasing use of *anycast* as a means of supporting a set of nameservers. But how should you set up this constellation of anycast servers? One approach is to use Virtual Servers where multiple nameserver processes run within their own container in a shared host platform. There is also a "Bare Metal" approach where multiple nameserver processes run within mutually isolated filesystems on a dedicated platform. A presentation by .DE's Benjamin Schönbach looked at the performance implications of this choice of platforms.

Virtual systems can be set up extremely efficiently, administered with ease, and are highly portable. Bare metal hosts have superior performance and scalability, but this comes at a cost of a higher administrative effort and limited portability (Figure 1). While the presentation did not extend into pricing, I suspect that pricing reflects relative levels of demand and virtualised cloud-based servers are the mainstream of the cloud environment and are priced highly competitively in most markets. Bare metal services are provided as a more specialised service, and this may have a price premium associated with it. A summary of these relative trade-offs is shown in Figure 1.

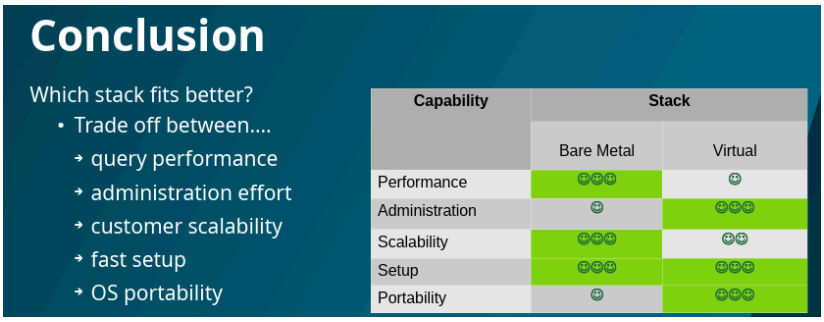

Figure 1 – Relative merits of Bare Metal va Virtual server platforms from "DNS Anycast Stack" at OARC 44

Presentation: DNS Anycast Stack

The Answer is "NO"

There are a number of ways that the DNS can respond to indicate that queried name, or a queried record type, does not exist. For non-existent names, the normal response is a NXDOMAIN response code (RCODE). Where the record type does not exist, the DNS returns an empty Answer Section and a NOERROR RCODE. This is referred to as a NODATA pseudo-RCODE. There are two other DNSSEC-signed responses where there is a wildcard match for the queried name and type, and no closer match than the wildcard (Wildcard), and where this is a wildcard, but the queried type does not exist (Wildcard NODATA). If the zone is not DNSSEC-signed there is no indication provided to the querier that the response has been generated due to the presence of a match to the wildcard entry in the zone.

When designing the addition of DNSSEC to the DNS, there was the challenge of how to provide an authenticated response to indicate that a name or a record type does not exist. The chosen response (eventually) was to use "spanning records", or NSEC records. All the zone's labels are sorted into a canonical order and each label entry has an NSEC record that enumerates the defined record types for the name and the next name in the zone (in canonical order). That way the DNSSEC credentials for the NXDOMAIN response can be generated in advance without knowing what non-existent name is being queried.

For example, given a zone file with the following records:

```
black.example.com.  6400   IN A      192.0.2.1
                    6400   IN RRSIG  A 13 2 6400 (…)
                    6400   IN NSEC   blue.example.com A RRSIG NSEC
                    6400   IN RRSIG  NSEC 13 2 6400 (…)
```

```
` blue.example.com.    6400   IN A      192.0.2.2
                       6400   IN RRSIG  A 13 2 6400 (…)
```

then a query for the non-existent name `blot.example.com` (and any name that falls between the labels "`black`" and "`blue`") will have the following response:

```
$ dig +dnssec blot.example.com

…
;; AUTHORITY SECTION:
example.com.       6400 IN  SOA   example.com. admin.example.com. 2025010101 …
example.com.       6400 IN  RRSIG SOA 15 3 6400 …
example.com.       6400 IN  NSEC  acme.example.com. NS SOA RRSIG NSEC DNSKEY
example.com.       6400 IN  RRSIG NSEC SOA 15 3 6400 …
black.example.com 6400 IN  NSEC  blue.example.com A RRSIG NSEC
black.example.com 6400 IN  RRSIG NSEC 13 2 6400 …
…
```

This response has two parts. The first four records (the SOA, an NSEC record and two signature records) provide an authenticatable account that there is no wildcard record in the zone. The remaining two records (NSEC and a signature) provide an authenticatable account that there is no name in the zone between "black" and "blue".

The intention in this design is to allow DNSSEC responses for non-existent names and record types to be prepared in advance, and the nameservers for a zone can generate answers from essentially a static zone file. The authentication "hooks" are based on the labels and record types that are defined in the zone and the NSEC record spans are used to match all non-existent names.

There are a couple of limitations in this approach. The first is that it is easy to enumerate a zone's contents by striding the NSEC records. For some zone operators this was (and is) a significant problem. The adopted solution was to order the hashes of the labels in the zone, rather than the labels themselves. While the SHA-1 hashes of the zone can be decoded, it's not a completely trivial exercise. This approach was termed "NSEC3". The issue now is that any changes to the zone require a pass to generate an ordered collection of hashes, which can be an imposition for very large sparsely-signed zones. NSEC3 also introduced an *Opt-Out* flag to reduce this span calculation overhead. Addition or removal of insecure delegations does not require a recalculation of the NSEC3 chains for the entire zone when using Opt-Out.

There was a proposal for NSEC5 in 2017 that was provably secure against zone enumeration, but it was sufficiently complex and costly to correctly deploy and use that it gathered no momentum in the IETF at the time.

The next area of consideration was that of the use of pre-computed signatures as compared to the emerging practice of online signing on-demand. Pre-computed signatures are more secure, and the signing keys can be stored offline. A signed zone generated in this manner can then be treated in the same way as an unsigned zone. The downside is that these signed zone data sets are larger and changes to the zone (the addition and removal of labels in the zone) require a signing computation across the zone to correctly configure the NSEC (or NSEC3 records). Online signing, in contrast, adds the signature records as part of the response generation process. Changes to the zone can be handled without and re-generation of NSEC or NSEC3 records. However, the online signer needs access to the entire zone to generate the correct NSEC or NSEC3 records (which is needed for the NXDOMAIN response) and all the RRsets for a given label in a zone (needed for the NSEC record used in the NODATA response). The challenge was then to reduce the overheads for the online signer for these types of negative responses.

RFC 4470 (and RFC 4471) described an approach that generated synthetic minimally covering NSEC records. Rather than consult the zone file to find the two labels in the ordered zone that span the queried name, it's possible to generate predecessor and successor names that lie effectively one byte "before" and "after" the queried non-existent name.

> This all sounds a bit odd to me. The whole idea behind NSEC and NSEC3 records was that if you are pre-computing the signed response to queries for, at the time of signing, unknown labels, then it's not possible to generate in advance a signed response of the form "the label *x* does not exist in this zone". So NSEC generates a signature using as hooks labels that exist in the zone and adds signed records to the zone that state "all labels between *x* and *y* do not exist."
>
> But if you are using an online signer that has possession of the zone signing key then it can generate a direct signed response that simply asserts that the queried label does not exist in the zone. However, this was a path not taken. Instead, the chosen path was to persist in using a spanning record NSEC(3) response, but to use the absolute minimal span that encompasses the queried label.
>
> I suspect that because the original concept was developed by Cloudflare outside of the IETF, then they were loath to rely on the existence of new Resource Records and new behaviours. The minimally spanning NSEC records were intended to operate as an authenticable denial of existence without requiring any modifications of the DNS standards and modifications to DNS validating code in resolvers.
>
> Somewhat ironically, now that this approach has entered into the purview of the IETF, new standard specifications are being drafted to describe this behaviour! If you are back in the IETF standards process, then why not just define a new signed response that cuts to the chase and simply asserts that "No, this label is not in this zone (either explicitly or as a wildcard match)."

However, the minimally spanning DNSSEC response is still large, with two NSEC records (one for the queried label and one to prove that a wildcard does not exist), and three RRSIG records (one for the SOA record and two NSEC records).

Large responses are a problem for the DNS. If we use a UDP transport then the large response may trigger UDP fragmentation, where lost packet fragments cause extended delays due to the use of timers in the absence of reliable explicit lost fragment signalling. Alternatively, if current operational practices are followed, the sender will not attempt a large UDP response and instead provide a truncated UDP response, triggering a re-query using a TCP transport, which costs delay in terms of the initial UDP transaction and the TCP handshake as well as the TCP control overhead in both the client and the server. There is no "best" approach here for large DNS responses, other than to try to avoid them!

The question then is: How to make this negative response smaller? The approach being proposed these days is called *Compact Denial of Existence*. This approach answers queries for a non-existent name by claiming that the name exists but there is no data associated with the queried record type. Such NODATA DNS responses require just one NSEC record and one signature record, which minimizes both message size and computational cost. The downside here is the loss of visibility of the NXDOMAIN signal and the ambiguity between non-existent names and so-called "*empty non-terminal*" names (which contain NS records but no other, and as the NS record is only authoritative on the child-side of a delegation the

parent asserts that it has no authoritative data for this label). The solution to this was to assert that the non-existent name exists in the zone, and has just three resource records: a RRSIG signature, an NSEC record and a NXNAME pseudo RRType.

> If you think that all this sounds a lot like piling incremental hack upon incremental hack, then you would not be alone in having that thought!

The NSEC record is constructed on the fly using a span of one byte beyond the non-existent name.

```
blot.example.com 3600 IN NSEC \000.blot.example.com RRSIG NSEC NXNAME
```

This approach is now being standardized in the IETF.

There is a trade-off going on here in terms of the ability of recursive resolvers to deflect random name attacks on authoritative nameservers. RFC 8198 describes the use of Aggressive NSEC Caching, where a recursive resolver can reuse a cached NSEC (or NSEC3) response and provide this same response for a query for any name that falls within this NSEC name span. That way, the recursive resolver effectively absorbs the load of a random name attack without passing the query load on to the authoritative nameservers. Minimally covering NSEC responses revert to the same properties as an unsigned zone and provide no such assistance in absorbing the load of a random name attack on a zone's authoritative servers.

As Salesforce's Shumon Huque points out, Authenticated Denial of Existence can be complex. There are many varieties with differing features and trade-offs, such as NSEC vs NSEC3, Pre-Computed vs Online Signing, White Lies, and Compact Denial of Existence. A given authoritative DNS system is likely to offer only specific modes of negative responses. It is a murky area if a DNSSEC-signed zone is served by multiple authoritative service operators, each of which use a different approach to Authenticated Denial of Existence. Online Signing with minimal NSEC records is popular for domains that are below the TLDs, while NSEC3 and *Opt-Out* remains popular in the TLD space.

The trade-offs include trying to keep the response size as small as possible, allowing caching resolvers to absorb random name query attacks, reducing the computation load per query on the authoritative servers and ensuring that both servers and resolvers share a common understanding of the various RCODES used in these negative responses.

Presentation: A Survey of Authenticated Denial of Existence in DNSSEC

Served Stale

The DNS is a loosely coupled system. In general, each component cannot necessarily force another party to behave in a certain way. They zone publisher (the zone's "authority") certainly provides a collection of hints, but they are at best hints and are not firm directives. A good example here is that of a resource record's cache lifetime. The domain's authority can propose a time duration to a recursive resolver to hold a resource record in its local cache, but that is more of a guideline than a mandatory direction. The purpose of such a cache hold time is to signal to the broader DNS environment about the anticipated longevity of the DNS data, or its anticipated longevity. Longer cache times relieve load from authoritative servers and provide faster name resolution to clients but hinder the process of making changes to the zone.

A resolver is expected to keep a copy of the data records it obtained from a zone's authoritative nameservers, and when subsequent queries are received, then the resolver can simply use the locally cached response and not query the authoritative nameserver. This local caching is the essential behaviour that makes the DNS viable and has allowed it to scale to the scope of today's network. Without it the DNS would not function in any useful manner. Caching is what allows DNS data to be pushed off to

the edge of the network very close to the user devices making the queries. Looking at the queries seen at the root servers, or at authoritative nameservers is only a small part of the picture of DNS activity, as like the tip of an iceberg, the bulk of the query volume lies below the surface, absorbed by local caches.

RFC 8767 describes an approach termed *serve stale*, where a recursive resolver may use "stale" DNS data to avoid outages when authoritative nameservers cannot be reached to refresh expired data in the local cache. One of the motivations for serve-stale is to make the DNS more resilient to DoS attacks and thereby make them less attractive as an attack vector, as the RFC asserts.

In using *stale* data, the local conditions are that the local cache entry has been held for more than the cache retention time interval, and the authoritative nameservers are not providing responses.to the local resolver. This still has some vague aspects, such as how long should a resolve retain and use notionally expired cached content. Hours? Days? Years? How much effort should a resolver expend in terms of time and queries before deciding that the authoritative nameservers are unresponsive and a stale answer used instead? Should the resolver retry to contact these unresponsive nameservers upon every query, or should the serve-stale state be cached and reused immediately for subsequent queries? What is the appropriate retry interval to re-query these unresponsive nameservers? At what point should the resolver stop serving stale RRsets. While Extended DNS Errors (RFC 8914) has the option to mark the response as "stale" what response code should be used otherwise? If a response is marked as "stale" what should the querier do in response, if anything?

There are a number of timers in a recursive resolver that have some relevance to the decision to serve a stale cached response.

The first is a *client response timer*, which determines the maximum amount of time that a resolver will spend in trying to assemble a response. When this timer expires should the resolver:

- Drop the query on the floor unanswered? Probably a poor choice, as the client may well assume some form of network loss and re-query in any case.
- Respond with a SERVFAIL RCODE? This indicates that the resolver has been unable to assemble a response, and hint to the client to try some other resolver if they are desperate for an answer.
- Serve a stale response from the local cache?

The second is the *query resolution timer*, which bounds the total amount of time that a recursive resolver will spend in attempting to assemble a response. An "unresponsive" authoritative nameserver in UDP is not sending the resolver an answer of the form: "I'm not going to answer you!" There is simply no answer at all in some given time interval which allows the recursive resolver to determine that the server is indeed "unresponsive".

The third is the *failure recheck timer*, which limits the frequency at which a failed lookup will be attempted again, which implies that the stale response will continue to be used for the lifetime of this failure recheck timer.

Finally, there is the *maximum stale timer*, which limits the amount of time that records will be kept past their nominal expiration, limiting the total "staleness" of such stale responses.

The presentation from ISC's Cathy Almond then explained how *serve stale* can be configured in a number of popular open source name server implementations, including Bind, Unbound, Knot and PowerDNS. I won't repeat that material here, as it can be found in her OARC 44 presentation.

The question Cathy poses at the end of her presentation is particularly relevant: "Is the serve stale feature genuinely useful?" Are old (and potentially outdated) responses better than no response at all? Or to put it another way: Are there reports from operational environments where serve stale has allowed name persistence through a period of extended outage of all authoritative nameservers? When a cache entry

has timed out out it is certainly faster to serve a query by returning a stale response and then refresh the stale cache entry from the authoritative server in the background, but is this approach just too risky?

The DNS is not a protocol that maintains tight synchronisation. It's a loosely coupled system where the data that is served is "approximately recent" rather than "instantaneously current". By relaxing this time window of loose synchronisation even further by allowing resolvers to serve cache timed-out data are we crossing a line between maintaining timely data coherency and trying to further improve aspects of DNS resilience?

Presentation: Thinking about serve-stale?

DNS Packet Sizes

When dealing with large DNS responses, The DNS steers a careful path between a couple of potential problem areas. The first is the handling of large responses using DNS over UDP. Once the response packet is greater than 1,500 bytes then IP fragmentation typically occurs, and fragment packet loss is qall too common. The DNS response is to use limit the size of DNS over UDP packets to the lesser of the EDNS(0) buffer size and around 1,400 octets. If the response is greater than this size, then the responder sets the truncated flag in its response. The DNS Answer section is not used in a truncated response, so it can be left empty. The querier is supposed to re-query using TCP.

AS IBM's Shane Kerr points out in his presentation on Packet Sizes, this requery using TCP is not universally supported within the Internet, and resolvers who do not use TCP are prone to enter a loop of UDP re-queries. Maybe the DNS 2020 Flag Day advice to use a 1,232-octet buffer size is too small and is triggering a requery using TCP unnecessarily.

This topic of the inability to perform a requery over TCP after receiving a truncated response was studied a year ago by APNIC Labs. Our study found that the TCP failure rate affected some 0.77% of users across the Internet. Higher than average TCP failure rates were observed in Bharti Airtel in India, China Unicom Guandong, and, interestingly, in a number of ISPs in Chile. There is no clear reason why this problem occurs over a number of Chilean ISPs.

Presentation: Kobayashi Maru: Packet Sizes

HTTPS Queries

QUIC is one of the more recent additions to the Internet's set of transport protocols. By "more recent" I mean "within the last 15 years or so!" which is not exactly recent. And it's not exactly a new transport protocol, as QUIC uses UDP as its transport protocol. QUIC is in some ways a re-packaging of TCP as an encrypted payload within a UDP stream.

How can a web client and server reveal their mutual interest in running QUIC as a session protocol? The original approach was to embed the server's capability to use QUIC via an Alternative Service directive embedded in the in the web content header, namely the addition of a content header: `Alt-Svc: h3=":443"`. The intended response by the client was to cache the association of the object's URL and the server's identity with the capability to use QUIC, and on the subsequent use of this URL by the client, it would use QUIC to perform the access to the server.

A more recent trigger has been the use of a domain name with a defined HTTPS RR type whose value includes the field: `alpn="h3"`. If the service can publish its QUIC capabilities in the DNS, then the client can query for this HTTPS RR type and if the `alpn` field includes "h3" then the client can use QUIC at the outset.

These days a dual-stacked, QUIC-capable client will launch three DNS queries simultaneously, for the A, AAAA and HTTPS RRtypes. For example, in the DNS query record below the three queries are launched within 0.108 seconds of each other. The A query is delayed presumably due to the Happy Eyeballs behaviour.

```
1740355205.111693 client 172.70.x.x#53408: query: 0du-xx-0.ap.dotnxdomain.net. IN AAAA -ED
1740355205.111896 client 172.70.x.x#29891: query: 0du-xx-0.ap.dotnxdomain.net. IN HTTPS -ED
1740355205.122545 client 172.70.x.x#19959: query: 0du-xx-0.ap.dotnxdomain.net. IN A -ED
```

As a result, one way to look at the uptake of QUIC as a client-side capability is to look at the logs of DNS queries and look at the occurrence of queries for A, AAAA and HTTPS RRtypes. And this is what Akamai's Ralph Weber reported on (Figure 2). The data shows a steady increase in the number of HTTPS queries. In this data set if A queries are 100%, then 32% of these names were also queries for their AAAA record and 5.5% of these names were queries for their HTTPS record.

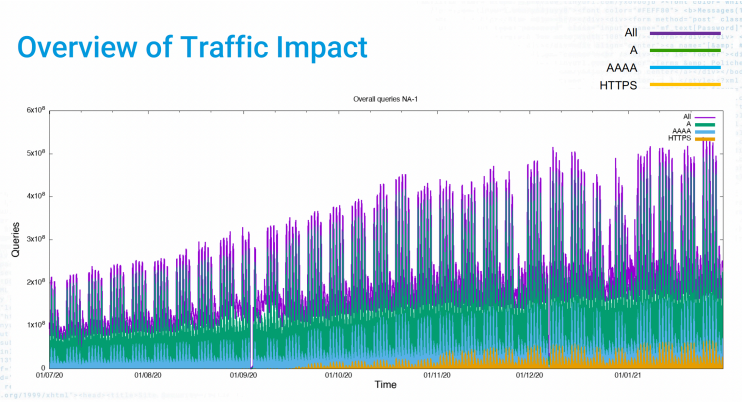

Figure 2 – A, AAAA and HTTPS queries, seen by Akamai, from "ECH from a DNS (data) perspective" at OARC 44

The other interesting data table is the use of the fields of this HTTPS record. This is shown in Table 1.

SVCB Parameter	Count	%
mandatory	33	0.00%
apln	75,521,983	99.58%
no-default-alpn	170	0.00%
port	1,806	0.00%
ipv4hint	75,531,330	99.59%
ipv6hint	73,437,342	96.83%
ech	64,966,919	85.66%
dfohpath	9	0.00%

Table 1 – HTTPS SVCB Parameters used in Tranco top 1M names, from "ECH from a DNS (data) perspective" at OARC 44

It is interesting to note that the dominant use of the HTTPS record, aside from the primary purpose of signalling QUIC capability at the server, is to provide the `ipv4hints` and `ipv6hints` address fields. RFC 9460 suggests that if A and/or AAAA records are also available, then clients should ignore these hint values. It is also interesting to note the prevalent use of the `ech` field, used to signal the key used to encrypt the SNI field in the TLS *client hello*. It is evident the majority of the use of these fields is related to the use of DNS and web hosting services being provided by Cloudflare hosting.

I must admit to being somewhat surprised by the use of three separate queries, namely HTTPS, A and AAAA as these A and AAAA queries are not necessarily essential queries given the use of the hints fields in the HTTPS record. However, the querier has no a priori knowledge of the presence of these fields in the HTTPS record. Maybe we could adopt a convention that always places the addresses in HTTPS record. Maybe we should reverse the relative preference in RFC 9460, with the underlying objective that a single DNS query for the HTTPS record should suffice in terms of a assembling a preferred connection profile

such that the next packet from the client should be one to initiate a connection to the HTTP service!

Presentation: ECH from a DNS (data) perspective

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