The ISP Column
A monthly column on things Internet

March 2022
Geoff Huston

IETF 113 – DNS Topics

The IETF met in a hybrid format in March 2022. Here are my impressions from the DNS-related Working Group sessions during the week.

Handling Negative Caching of DNS Resolution Failure

It has been a feature of the DNS that whenever there is a failure in serving DNS data, the query rate seen at the authoritative servers associated with the point of failure invariably increases. Failures can expose a whole range of implementation behaviours that are not normally visible when everything is working as expected. When an DNS client encounters an unexpected break in the communication flow how long should it wait before giving up and returning an error condition? While waiting for a response how many times should it resend the query? Should the querier perform an exhaustive search through all the nameservers? If it's a failure in a DNSSEC validation chain, should it search across all possible server permutations of the chain?

There is the view that it's better for the user experience (and better for avoiding becoming a DOS cannon!) to fail quickly with minimal retries. And there is the view, more prevalent in earlier DNS days, that patience and persistence in attempting to discover a response is often rewarded. In TCP most of this behaviour is governed by TCP session timers, with the one area of the initial SYN exchange being where we see variations in behaviours. In DNS over UDP this is a relevant question as the variation in behaviour is broader. What should a client of a DNS recursive resolver do when that recursive resolver fails to answer after some (unspecified) time? And what should a DNS recursive resolver do when an authoritative server fails to answer after some (unspecified) time? Are the two cases much the same? Or should recursive resolvers use a different wait-and-requery strategy? How persistent should they be in seeking a response? What are the impacts on the remainder of the DNS infrastructure when this happens?

When Facebook's authoritative servers failed to respond for six hours in October 2021, the queries seen on the .com nameservers increased from a 'normal' 7K queries per second (qps) to a peak of 900Kqps, an increase of 128 times over the normal query load! Other experiments show a larger 1200 times increase in query traffic when a botnet domain is intentionally disrupted by altering the DNS to return the SERVFAIL response code. A large increase in query load was observed at A and J root when the old key was published with the revocation bit, which abruptly disappeared when the revoked key was removed from the root zone.

The problem appears to have got worse as we turn to large scale DNS resolver "farms" as our way of constructing high-capacity resolvers. These services are constructed by using a collection of semi-autonomous resolvers that may (or may not) share the outcomes of successful resolution, but they do not block each other when one resolver starts to work on resolving a name. Failure implies delays and the extended resolution time can result in every resolver in the farm attempting to independently resolve the same name, particularly when the name is commonly used.

DNS specifications have been progressively updated to provide guidance on how to handle the prolonged silence that follows a UDP query to an unresponsive server. RFC 8767 advises that "Attempts to refresh from non-responsive or otherwise failing authoritative nameservers are recommended to be done no more frequently than every 30 seconds.". However, this is perhaps a very coarse response, and it leaves open the prospect of shutting down queries to a server in cases where the problem is context specific and related to the query, not necessarily the server.

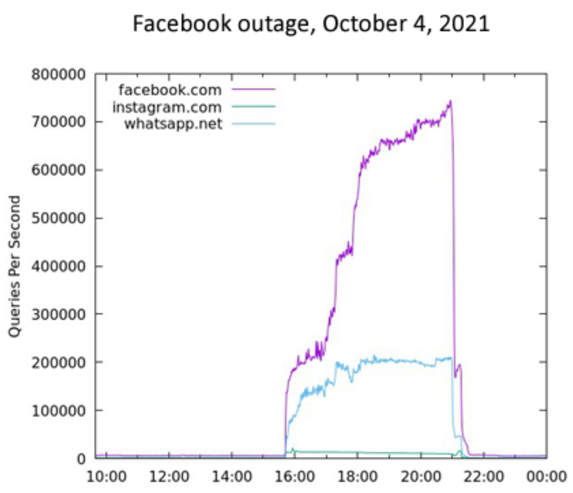

Figure 1 – Query Rate at com/net servers during Facebook Outage – From "Negative Caching of DNS Resolver Failures"

The draft, 'draft-dwmtwc-dnsop-caching-resolution-failures' proposes to update RFC 2308 by explicitly requiring the resolver to cache the non-responsive state of the server against the query profile. Further queries for this query should not be performed until the cached information expires. It also provides some guidance as to what is unresponsive, and here it's a sequence of three queries. It is not as specific as to how 'unresponsive' is defined but suggests a local timeout of between 3 and 30 seconds. What about the timer to be applied to the cache entry? The document proposes a 5 second initial cache time, and an increasing cache time for subsequent resolution failures to a maximum of 300 seconds. Constraints should apply to querying the parent zone servers, and RFC 4697 describes such constraints, although is not as specific with suggested timers and retry limits.

I can't help feeling that this is well intentioned, but likely to be misdirected effort. RFCs describe many aspects of DNS resolution behaviours, but they are mute on the normative behaviour of DNS resolver 'farms' and mute on how individual resolver engines should be coupled within the farm. It is likely that the basic cause of the amplification factor when DNS failure is encountered is the multiplicative behaviour of loosely coordinated large scale resolver farms. The resolver query constraints described in the draft may ameliorate this query amplification to some extent, but I suspect that it may be useful to look a little deeper into the behaviours of large-scale resolver farms and how they react to failures.

Glue

The DNS is a loosely coupled distributed database and resolving a domain name firstly involves a discovery phase to discover the authoritative servers for a domain name and then querying one of those servers for the name to be resolved. This discovery process is assisted through the use of *glue records* in DNS responses, where the resolver adds the name and IP address of the zone servers in a referral response in the Additional Section of a DNS response. Servers are expected to return all glue records in a referral response. However, what should a server do when assembling a UDP referral response and the total size of the response would exceed the working UDP size limit for this response?

The server has a number of options. It could add as many as it can fit into the response and leave it at that. After all, the querier is after any server, not all servers, and in the normal course of events it may be a reasonable guess that all servers are working and answering queries, so any server is as good as any other. The server could set the TC flag in this response to inform the client that the response is incomplete, and that the client should use TCP or its equivalent to retrieve the full response.

There is a question that is raised by this seemingly innocuous truncation directive. If there are some resolvers listed in a truncated referral response, should the resolver try these resolvers before re-querying over TCP? Is setting the truncated bit in a response a mandatory directive to the client to re-query in all cases? Or it could be just an indication that there is more information that is available, and the client can use TCP to retrieve it, but if the client can proceed with what is available in a truncated response, then it should do so. The former seems like a more pragmatic action, but there is always the risk of stalling a resolution with partial information. Of course, the prudent action would be to always re-query when the UDP response has the TC bit set.

Secondly, should a server differentiate between in-domain servers and other servers? For in-domain servers, glue records are essential because the circularity implies that the IP addresses of these servers cannot be otherwise discovered. Other servers are independently resolvable and the including of these servers' addresses as glue records in a response is not essential. This leads to a slightly difference directive, namely that if the message size constraints prevent the inclusion of all in-domain referral glue records, the server must set the TC flag to inform the client that the response is incomplete, and that the client should use TCP or its equivalent to retrieve the full response.

The Working Group conversation headed into registry issues and some consideration of management of DS records, illustrating that these days that contemplating changes in any DNS practices tends to ripple across all parts of the DNS infrastructure, making any such changes challenging to get right.

Service Bindings and DANE

There has been much work in recent times on service records in the DNS. These SVCB records allows a client to query the DNS with a service query name (in the form of *_port._transport_.basename* query names). This query is expected to follow CNAMES in the usual way.

In the process of setting up a TLS session a client needs to resolve the IP address of the server, and then perform a TLS handshake with that IP address. In that TLS handshake the client needs to provide the name of the intended server (SNI) in the clear, allowing the server to select the appropriate server public key certificate to kick off the TLS session. This in-the-clear SNI field is a huge problem in terms of privacy leakage. How can this be encrypted to protect this information from onlookers? The adopted approach is to turn to DANE's TLSA records, and publish SNI encryption keys as TLSA records.

An example may help here. This is how one would configure the DNS to allow a name server to be queried via DoT or DoQ.

```
_dns.dns.example.com. SVCB 0 dns.my-dns-host.net.

dns.my-dns-host.net.  SVCB 1 . alpn=dot,doq
dns.my-dns-host.net.  SVCB 2 . alpn=dot port=443

_853._tcp.dns.my-dns-host.net.  TLSA ...
_853._quic.dns.my-dns-host.net. TLSA ...
_443._tcp.dns.my-dns-host.net.  TLSA ...
```

The flexibility here is that different SNI encryption keys can be provided for each of the different transports, although why this is a feature and not a bug is a question I just can't answer!

Dry run DNSSEC

DNSSEC slip-ups happen all the time. In March the `.au` domain name published an iteration where the DNSKEY record was missing its RRSIG signature. Around the same time one of the servers for usa.gov was serving expired RRSIG records for its DNSKEY records. Then there was Fiji earlier in the month (https://blog.cloudflare.com/dnssec-issues-fiji/).

As we found in the KSK rollover some years back, performing comprehensive test of the deployment of a signed zone before using it can be challenging. While it is straightforward to check for the obvious problems, such as missing, expired or inconsistent signature records, some of the more subtle issues appear to manifest themselves only when other resolvers perform validation over the zone. However, DNSSEC fails *hard*. If there is a validation problem the resolver returns an error.

What if there was a way to sign a domain with a *testing* flag, so that validation failures would fail *soft*, return the DNS response in any case, and pass a failure diagnostic back to the authoritative server. There is a hint in the DNSSEC specifications that if a domain is signed with an unknown DNSKEY algorithm or with an unsupported message digest algorithm, then the resolver should treat this as an *unsigned* domain. This proposal is for a reserved DRY-RUN message digest type in the DNS record to signal that a DRY-RUN aware resolver should process the data as normal, but in the event of a validation failure treat the zone as unsigned and pass on an effort indicator.

Would this help?

Perhaps, but not in the cases above, and probably not in general.

The problem is not in the initial switch to DNSSEC-signing, but in the ongoing operational support of signatures. What we would like, of course, is that any third-party efforts to manipulate the DNS should result in a *hard fail* in validation, while any configuration errors DNSSEC should cause a *soft fail*. Yes, this is another case where the *evil bit* (RFC3514) would come in very handy!

Stateful Hash-based DNSSEC Signatures

Sometimes the IETF gets into a pattern where it obsesses over various solutions and loses sight of the original problem in the process.

Quantum computing, if it ever becomes viable at scale, poses some challenges to current thinking in cryptography. It's not as if our current approaches to cryptography create *impossible* problems. They don't. They rely on *difficult* problems, or computationally infeasible problems. What is *infeasible* is not the same for everyone. If I want to keep a secret for 20 years, then an attacker has 20 years to work on the problem, and the attacker will take advantage in increases in computational capacity over that period. If I want to keep a secret for just one day, then the attacker has just 24 hours from first use to work on the problem.

When looking at post-quantum cryptographic algorithms the mainstream effort is looking at the general issue and the objective of maintaining the integrity of some encrypted information for more than a decade. The does not translate to DNSSEC requirements in general. Most folk should roll their keys far more regularly than once a decade, and yes, this includes the KSK of the root zone. As a crude rule of thumb, the shorter the operational lifetime of the DNSSEC keys, the less the pressure to use extremely strong (and often very large) signatures.

One proposed approach is stateful hash-based signature schemes that generate a finite sequence of one-time signing keys. It is essential for the signer to keep state here, as the re-use of the same key value breaks the security of the scheme, so this is a more fragile arrangement than conventional multi-use keys. The signatures are large (more than 2.5Kb) so it's unlikely that this would be a preferred candidate for a robust quantum-computing resistant. The proponents of this approach argue that it should not be the preferred approach but should be kept on as a safe fallback option.

I suspect that DNSSEC would benefit not on working on larger and more potent cryptographic algorithms, but on greater key agility, including at the root, reducing the lifetime of operational keys down from year and months to perhaps one or two days. Is this feasible? Hard to say right now, but it may be useful to find out.

Expressing QoS in DNS queries and Responses

There seem to be two significant themes at work on the Internet today. One is to lift functions into the application layer and regard the lower layers of the protocol stack, including the network layer as an undistinguished commodity. To ensure that the application flows are not altered by the lower layers, encryption protects the integrity of these flows. We've seen this in QUIC, and DNS over HTTPS. The other is to increase the functionality of the network layer and make it responsive to the service requirements of applications. This approach is typified by "NewIP" today but can be traced back to a whole collection of Quality of Service (QoS) mechanisms over some decades. In the public Internet these QoS-styled networks have been an abject commercial failure, gathering no traction whatsoever.

But of course, that does not stop the proposal factory from churning out more. This proposal, draft-eastlake-dnsop-expressing-qos-requirements, encodes an application's desired service properties in the DNS Query name, and somehow has the IP address provided in the response as being associated with a particular network service profile. I simply can't see this proposal faring any better than previous proposals.

Just Say No!

The DNS protocol has a number of ways to say "no". There is NXDOMAIN reponse to indicate the domain name simply does not exist. There is NODATA to say that this query type does not exist. Then there is REFUSED to indicate that this server has determined not to answer the query, possibly due to some policy setting, and SERVFAIL to indicate that the service is unable to provide a response. And of course there is the absence of any response at all.

There has been a long-running debate in DNS circles as to whether there should be more error responses, what such additional codes may be, and who is the intended consumer of such codes. RFC 8914 defines Extended DNS Errors (EDE) as a collection of 24 additional codes that are intended to augment with original set of error codes.

This can be very helpful in a number of scenarios, particularly in handling DNSSEC validation errors. At present such errors cause a SERVFAIL response code, and many resolver implementations interpret this response as an invitation to re-query using an alternate server, and this takes time and causes further unnecessary queries.

However, these EDE response codes are intended to be used by resolvers, not by human users.

There has been a parallel effort to provide a textual description of the resolution failure, and draft-wing-dnsop-structured-dns-error-page is the current proposal in this direction. One view is that the DNS is not a user-visible protocol and readable diagnostic messages serve no useful purpose in this context and may cause a new set of vulnerabilities by having unauthenticated DNS middleware direct user applications to arbitrary destinations. I'm not sure what the positive view is on having such long-form error messages might be, but hopefully the draft's authors have some idea of why they are proposing this!

When to Encrypt

Encrypted sessions in the DNS is a recent thing. There is no explicit form of signalling that a server, whether it's a recursive resolver or an authoritative server, is capable of supporting DNS queries over TLS or QUIC.

For the stub-to-recursive scenario there is the Adaptive DNS Discovery work (ADD Working Group), which is attempting to answer this through adding discovery of this capability either by provisioning or by discovery through probing. Probing has an overhead in terms of time and resources, but this can be amortised over a number of subsequent queries. However, in the recursive-to-authoritative scenario this is not the case. In this case the recursive resolver has to probe an authoritative server to use if it responds over QUIC or to a TLS client hello, then establish the secure connection, then pass the query over this connection.

It's an interesting position. Previously when we've been confronted with the conflicting aims of security and speed we've opted for speed. Checking for certificate revocation (or not checking in the general case of Chrome) is a good example of this preference. This approach, draft-ietf-dprive-unilateral-probing, is proposing to go in the opposite direction and slow down the recursive-to-authoritative query process.

Frankly, I can't see this happening. There is no client information in the recursive-to-authoritative query (notwithstanding the fiasco over client subnet), and if query minimization is being used then the information leak is that authoritative server is being queried about names within the immediate scope of the zone for with the service is authoritative. I've noted already that sometimes the IETF gets into a pattern where it obsesses over various solutions and loses sight of the original problem in the process. I suspect this the recursive-to-authoritative privacy issue question is provoking a similar response.

IETF 113 Materials

The meeting material for this meeting can be found at https://datatracker.ietf.org/meeting/113. Video records of Working Group sessions can be found at on the IETF YouTube channel.

Disclaimer

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