The ISP Column
A monthly column on things Internet

February 2019
Geoff Huston

No!

What part of "**No**!" doesn't the DNS understand?

One effective form of attack on the authoritative DNS server infrastructure, including the root servers, is the so-called random name attack. If you want to target the online availability of a particular domain name, then a random name attack will attempt to saturate the domain name's authoritative name server (or servers) with queries to resolve names in that zone, putting the server (or servers) under such a level of load that 'legitimate' queries are no longer answered. The result is that the name goes dark and the denial of service attack is successful.

This attack is far easier to mount than many other forms of attack. It does not require any advanced form of crafting of query packets and does not require a sophisticated level of control of the attack bots. All that's needed from each co-opted host is the ability to generate random names within the targeted domain and then attempt to fetch URLs that use this random name. This is typically achieved within a simple script with no need for privileged access on the host.

The attack is effective in targeting the authoritative servers because of the way in which the DNS handles "no such domain name" (`NXDOMAIN`) responses. The aim of the attacker is to bypass any local DNS resolver caches, as the intention is to place the authoritative server under pressure. If the distributed attack uses a name that exists within the target name space, then as the query level for the name increases from a diverse set of orchestrated attackers, the resolution response will be cached in the attackers' local recursive resolvers and these recursive resolvers will essentially protect the authoritative name servers. Even if the attack uses a non-existent but constant query name, then resolver caching will still be effective, as the recursive resolver will cache the fact that a DNS name does not exist and use this cached information to respond to subsequent queries for the same name. If the attack uses randomly generated names, then the local caches are ineffective as they only cache the specific query name, and each query for a new random name is passed to an authoritative server to resolve.

Understanding how the DNS handles requests to resolve non-existent names is an important part of understanding how we can make the DNS more resilient to this form of random name attack. To focus on this aspect of the DNS at APNIC Labs we've been experimenting with seeding the DNS with queries to non-existent domain names that are part of domains that we operate.

The methodology is very simple. We use a script embedded in an online ad to perform a measurement experiment and collect the query data on our server infrastructure. The script generates a DNS resolution query for a random non-existent name, and we then see DNS recursive resolvers query our authoritative name servers for these same DNS names. (Figure 1).

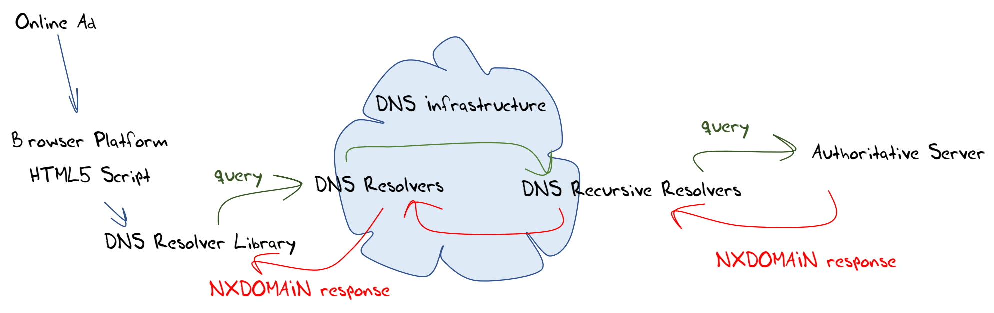

Figure 1 – NXDOMAIN Experiment

By directing the ad to be impressed on millions of endpoints per day we can quickly gain a view that encompasses much of the Internet's breadth both in terms of geographic diversity and platform diversity.

Predicted Results

We had a theory about what we expected to see from this simple experiment.

To explain the theory, we need some background of how DNS resolvers are configured. Clients are loaded with the addresses of one or more DNS resolvers. For example, my ISP provides two such resolver addresses, which are the passed to my devices by DHCP-based auto-configuration. When an application generates a DNS query the application host's name resolver library routines will send the query to one of these resolver addresses. Because the DNS is a UDP protocol then there is no assurance that the end host will receive a response, so the end host starts a timer when it launches the query. If the timer expires and no response has been received, then the query will be repeated. If there are multiple resolvers configured in the end host, then the repeat query may be directed to a different resolver. Successive timeouts may cause the client to issue further repeat queries to the set of configured resolvers. The interval between successive queries may lengthen until a total query period has elapsed, at which point in time the query is abandoned.

The timeout period between queries, and the total time that a query will remain active in a resolver varies according to the DNS resolver implementation and local configuration settings. An example of one such approach, Microsoft's client resolver library, the query repeat procedure is documented at:
https://support.microsoft.com/en-au/help/2834226/net-dns-dns-client-resolution-timeouts

It's not just the client resolver libraries that perform re-queries. A similar timeout mechanism is used in recursive resolvers, and these resolvers may also emit further queries when they do not receive a response. Where two or more authoritative nameservers are configured for a name, the repeat queries may be directed across the nameserver set. Typically, we see a far more conservative approach to re-query from recursive resolvers than from end hosts. If both end hosts and forwarding resolvers use a very aggressive set of re-query timers then a single query could potentially trigger a query storm on a loaded server, with the potential of turning a transient load event into an unintentional denial of service attack!

The general rule of behaviour is that unresponsive servers will elicit further queries from resolvers, but these re-queries should not impose a significant additional load on the server.

As well as the queries, let's look at the response. When a name does not exist in a zone there will be a DNS response sent back to the querier. The DNS response uses the response code 3, `NXDOMAIN`. The response indicates that the name is not contained in the zone, and resolvers should interpret this response as a terminating response to their query. This is a locally cacheable response, so repeated queries for the same name, irrespective of the query type, can be answered by the caching resolver from its local cache for the cache lifetime.

We can now answer the question: What were we expecting to see when we answer queries with `NXDOMAIN`?

We expected to see a single query from each client, assuming that the authoritative server is answering every query and the delay in the combined network delay and the server processing delay is under the client's retransmit timer values.

We ascertained that we answered every query by performing a packet capture of the servers used in the experiment, confirming that every query for these names generated a matching response.

For those clients with a long delay to the server we may see a repeat query, but on the assumption that the outstanding query is still 'open' with the client, it is likely that the initial response will arrive at the client within the timeframe of the timeout of the second query.

For these reasons we anticipated that the exceptional delay conditions would be somewhat rare, so that the average number of queries for each non-existent unique name, as seen by the authoritative server for the domain would be far closer to a value of 1 than 2.

Observed Results

We used the data from a random sample of experiments, and only looked at queries for non-existent names within the first 30 seconds of the name being presented to the authoritative server. Some 6,269,472 measurements were included in this data set.

The number of DNS queries seen at the authoritative servers for these non-existent domains was 14,003,675 or an average of 2.23 queries per queried unique name. This is a surprisingly high value, particularly so given our expectations described above

We operate four servers for the experiment, locating the servers in Singapore, Frankfurt, Dallas and Sao Paulo and geo-locate the experiment end hosts so that the name the end hosts are given to query will direct their DNS queries to the server that is closest to their location. We observe round trip times to end hosts are well within half a second for almost all measurement endpoints, and if client software uses a 1 second re-query timer then we expected well less than roughly 10% of endpoints would experience a query timeout and perform a re-query. This average of over 2 queries per unique name is well outside the anticipated result.

Average values occlude a large amount of information, so it is useful to understand the distribution of individual experiments. Are we seeing a small number of experiments with excessively large queries? Or are we seeing a more 'normal' distribution.

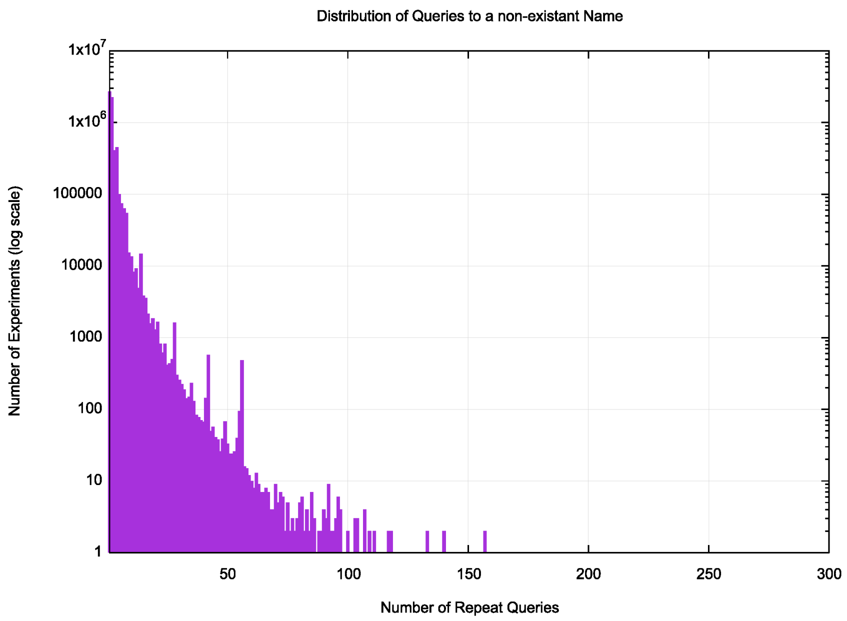

Figure 2 – Distribution of Query Counts per unique non-existent name

Figure 2 shows the distribution of the number of queries for each experiment.

Surprisingly only 43% of experiments used a single query to establish that the name was not defined. That figure implies that more than half the experiments did not accept an initial `NXDOMAIN` response! A further 36% of resolver systems used 2 queries to establish the non-existence of the domain name, while a further 21% took 3 or more queries. The range of queries over the 30 second window from the initial load of the experiment in the end hosts was from 1 query to 297 queries!

A closer look at the lower range of queries (1 to 100) shows a roughly linear drop in the log-scaled plot (Figure 3), which correlates to an exponential decline in the unscaled data.

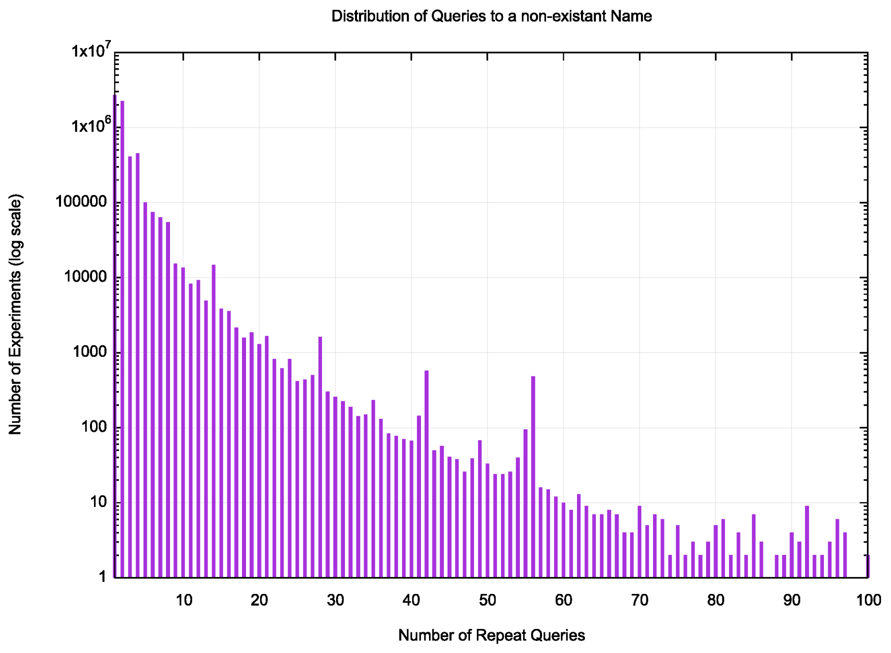

Figure 3 – Lower Range of Distribution of Query Counts per unique non-existent name

What about the time interval between successive queries? Is there any discernible time signature? If we take the time of the initial query as it is seen at the server as the start time, and then record all successive queries with the difference in time between the initial time and the query time then we can look at the distribution of these time differences. They are shown in Figure 4.

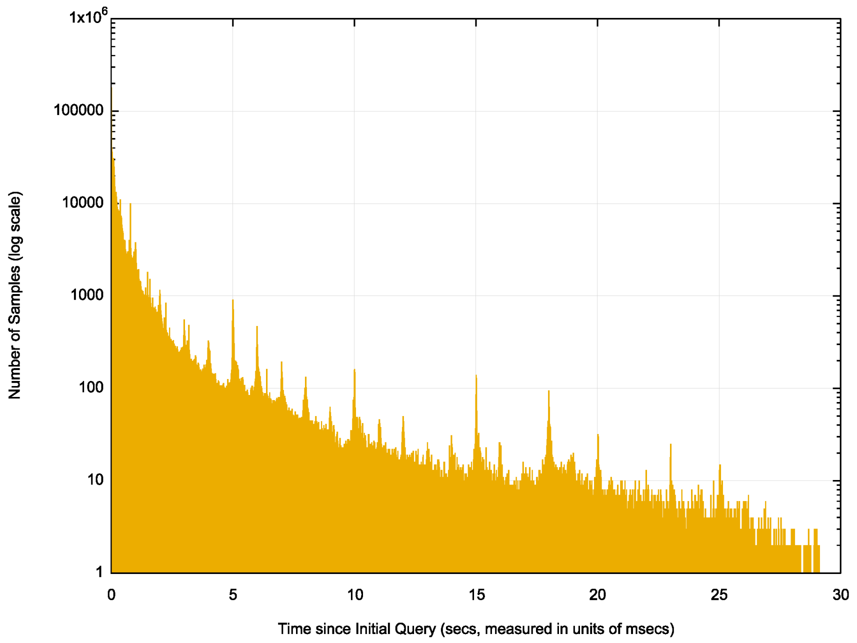

Figure 4 – Distribution of elapsed time from initial query to re-query

There is a discernible 1 second interval between local peaks, particularly in the period between 4 and 12 seconds after the initial query. The peaks at 5, 10 and 20 seconds are also far higher than the background. However, such regular peaks are not the dominant signal.

There are probably a number of factors that contribute to this re-query pattern.

Happy Eyeballs

The very high number of instances where there are exactly two queries spaced closely together in Figures 3 and 4 points to a likely side-effect of dual stack name resolution using the *happy eyeballs* technique (RFC8305). A *happy eyeballs* application running on a dual stack hosts will generate queries for the A and AAAA records of a name essentially in parallel, irrespective of whether the query name exists or not.

Let's split out these query records and look at the query counts for A and AAAA query types records separately (Figure 5).

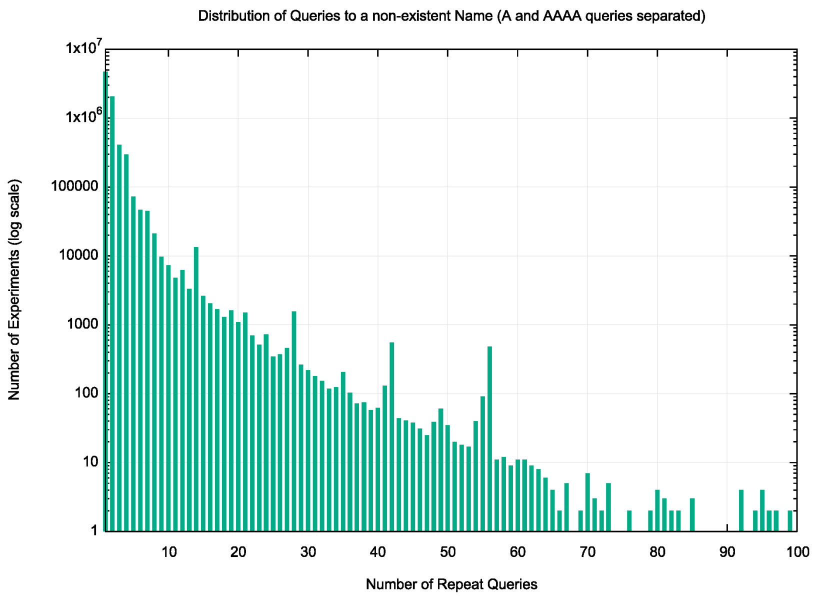

Figure 5 – Lower Range of Distribution of Query Counts per unique non-existent name with A and AAAA queries separated

In this revised view of the data some 3,689,593 experiments out of 6,269,472 (or 58% of experiments) used a 'single' query to establish that the name was not defined, where 'single' is either a single A query, a single AAAA query or both one A and one AAAA query. That still means that for slightly more than 40% of the time we see multiple A and/or multiple AAAA queries for the same query name.

Given that approximately 20% of end users are on dual-stack platforms, it appears that the *happy eyeballs* algorithm adds a similar additional query load to the DNS when attempting to resolve non-existent names.

While the *happy eyeballs* approach explains some additional queries for a non-existent name, as we can now account for 4,560,076 queries as instances of some form of single query, but that still leaves us searching for an explanation of the remaining 9,443,599 queries where we have repeat queries for the same non-existent domain name.

Resolver Re-Queries

Could lost responses be a possible factor here? A reasonable assumption that once a resolver receives an `NXDOMAIN` response it will cease further queries if the resolver is performing 'normally', and it would only re-query if it did not receive a response (or if it had entered an error state of uncontrolled re-queries). The distribution of re-query intervals from the same resolver address for the same query type is shown in Figure 6.

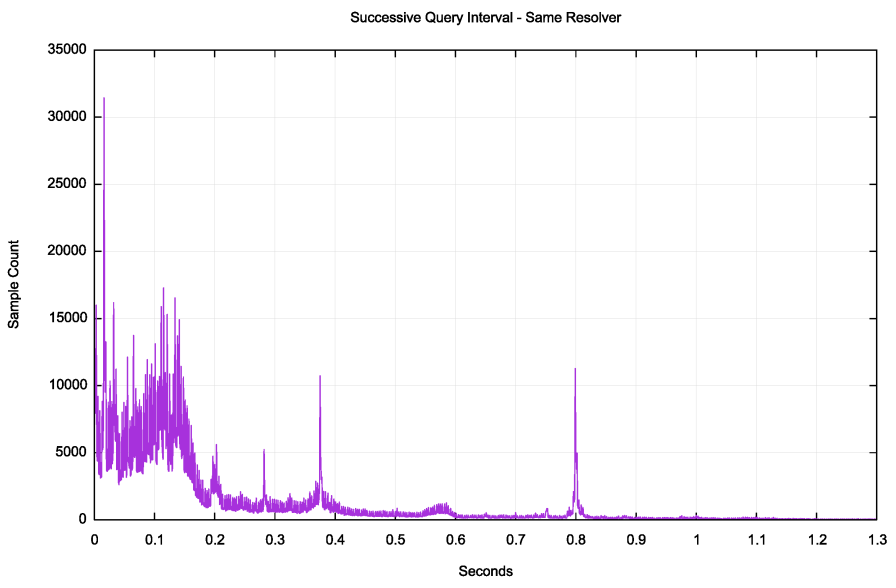

Figure 6 – Distribution of re-query intervals for the same resolver

The same resolver will repeat the same query within 4ms some 1% of the time. Even this relatively low number is unanticipated given that the resolver is not expected to use a re-query timer of millisecond duration, and a recursive resolver should keep a list of outstanding queries and not generate a new query for the same name as an outstanding query.

There is a clear peak of query intervals at 0.2 seconds, 0.28 seconds, 0.38 seconds and 0.8 seconds. It is feasible to relate the 0.8 second peak to the re-query interval of some DNS resolver implementations. There may be some quite aggressive re-query timers in certain resolvers at 0.2, 0.28 and 0.38 seconds. It is unclear why there is such a high immediate re-query load under 0.15 seconds. The interval is probably too short to be a conventional re-query interval, but sufficiently long enough not to be the result of network-level packet duplication.

A cumulative distribution of the data (Figure 7) shows that some 12% of the total query load are repeat queries from the same resolver, and the re-query interval appears to be within 1.6 seconds. Some 8% of resolver re-queries with 200ms of the previous query. This appears to be a very fast re-transmit and it unlikely that it is based on a local re-query timer value.

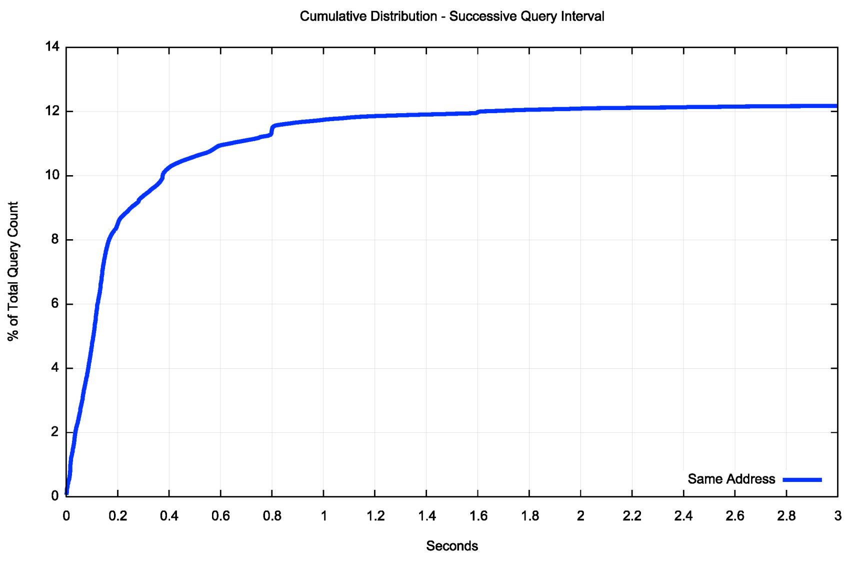

Figure 7 – Cumulative Distribution of re-query intervals for the same resolver

A conventional assumption of the cause of repeat queries from the same resolver is that the resolver has not received the earlier response, either because of network path congestion or local resolver congestion. However, the large number of instances of repeated queries within such short intervals does not seem to be consistent with that assumption.

A possible explanation is concentration of recursive resolvers. If a client is configured with two forwarding resolvers, but queries to either of these resolvers are passed to the same recursive resolver, then it's possible for queries to the recursive resolver to arrive while a previous query is still awaiting a response. Depending on the way the recursive resolver manages its task queue these duplicate queries may remain in the processing queue and be resolved in turn despite the still-active previous query.

Priming Resolver Farms

Beyond queries from the same resolver IP address, there is another potential cause of repeated queries, and that is poorly configured *resolver farms*. A resolver farm is used in many large-scale DNS resolvers. Rather than pass all queries to a single DNS resolver instance, there are a collection of *slave resolvers* and incoming queries are scheduled to a slave instance for resolution.

What we see is a number of instances where a cluster of resolver IP addresses that share a common address prefix making repeated queries for the same query name. An example of this behaviour is shown in Table 1.

Resolver	**Query Time**
7x.xxx.0.178	0.752
6z.zzz.161.146	0.865
7x.xxx.0.230	0.980
6z.zzz.161.220	1.094
7x.xxx.0.188	1.201
6z.zzz.161.182	1.319
7x.xxx.0.180	1.430
6z.zzz.161.144	1.542
7x.xxx.0.226	1.650
7x.xxx.0.138	1.654
6z.zzz.161.134	1.762
6z.zzz.161.222	1.775

Table 1 – Resolver Farm repeat queries

We can make some highly approximate assumptions about how to identify a set of resolvers that collectively form a *resolver farm*. The easiest method is to group all resolver addresses that share a common /24 IPv4 address prefix, or a common /48 IPv6 prefix, and consider them to be members of the same resolver set.

The distribution of interval time between repeated queries from resolvers that share a common address prefix is very similar to the previous case of queries from the same resolver IP address (Figure 8). The overall volume of queries from resolver farms appears to account for some 29% of the total query load.

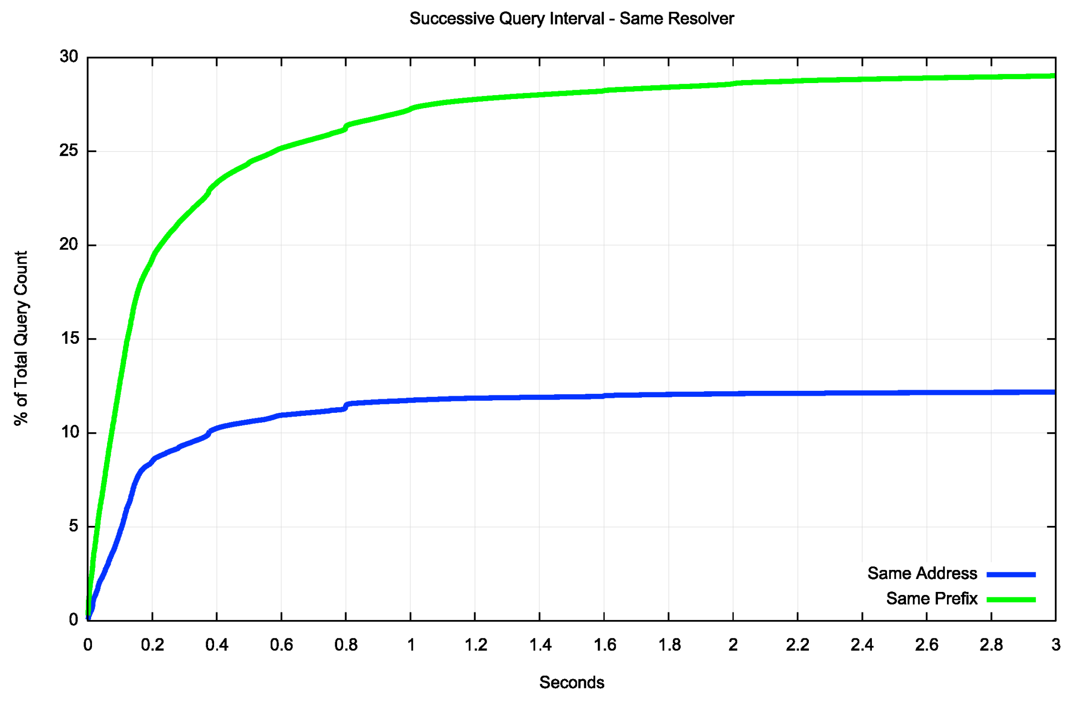

Figure 8 – Cumulative Distribution of re-query intervals for a common address prefix resolver sets up to 30 seconds

The incidence of repeated queries using this classification of a common /24 (or /48 for IPv6) is significant. The same resolver accounts for some 12% of the total query load, and a further 16% are from resolvers that are located in the same subnet.

The time interval between resolver farm queries is shown in Figure 9.

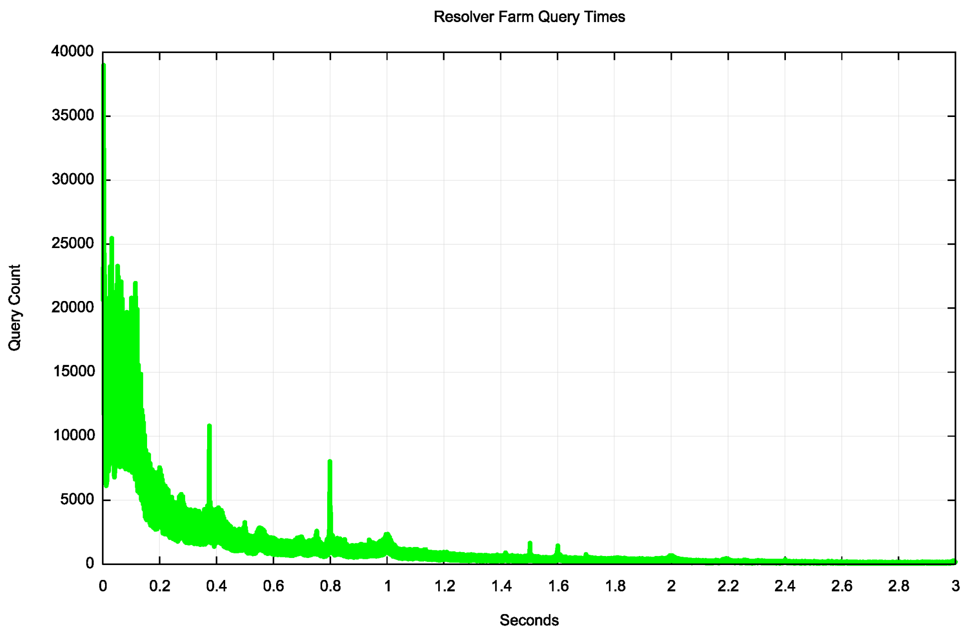

Figure 9 – Distribution of re-query intervals for resolver farms

Its evident that there is both a 'front load' signature where other elements of the resolver farm perform the same query operation so as to, and then there are re-query peak times at 0.38 and 0.8 seconds which is consistent with the same resolver re-query time interval.

The cumulative time distribution to load the resolver query farm is shown in Figure 10. One half of the query times across the resolver farm are less than 0.25 seconds

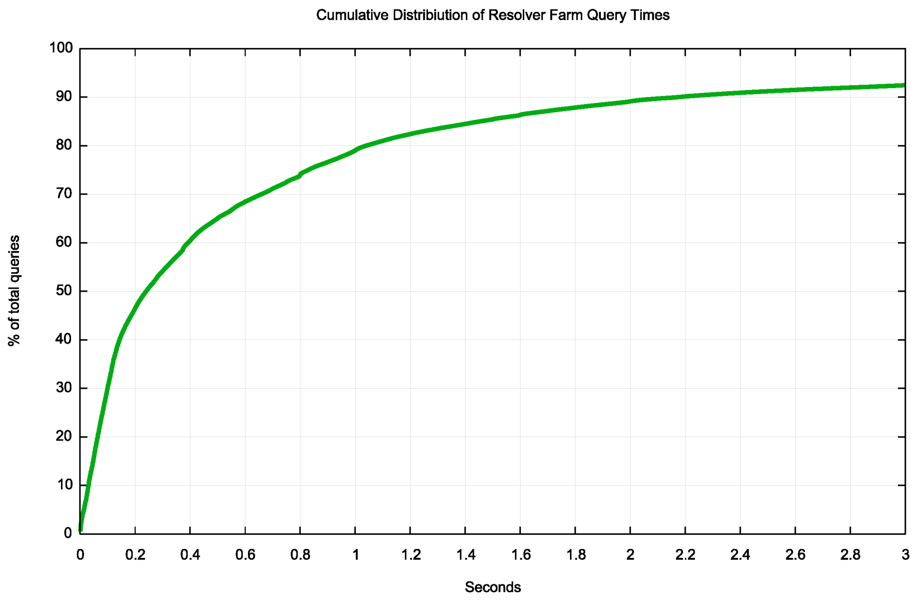

Figure 10 – Cumulative Distribution of re-query intervals for resolver farms

So exactly what part of "NO!" can't the DNS understand?

Part of the explanation of the DNS's apparent inability to efficiently handle a 'no' answer lies in the IPv6 transition, where *happy eyeballs* has had the side effect of increasing DNS load. The use of two back-to-back DNS queries for A and AAAA records means that all dual stack end hosts will generate twice the DNS query load for simple name resolution functions. Given that the population of dual stack endpoints is now around 20% of the total we would expect to see an average of some 1.2 queries for non-existent names in the DNS. In some ways this is an unavoidable by product of the dual stack world. As the level of dual stack deployment increases, we can expect to see the use of simultaneous A and AAAA queries in the DNS increase.

But we see a far higher average query rate for non-existent domain names than just the pool of dual stack hosts can generate. What else is going on?

The choice of UDP for the DNS has some interesting compromises. The use of a lightweight datagram protocol is a highly efficient way of supporting a simple query response application model, and the scaling ability of DNS is due in some part to this choice of UDP as the DNS transport protocol. There is the ever-present likelihood of packet loss in these UDP-based transactions, and to some extent the incidence of re-queries is due to packet loss. But to achieve the observed DNS re-query result through packet loss alone we be experiencing a packet loss rate of some 50% across the Internet. We probably would've noticed that rather catastrophic level of packet loss!

The datagram model poses some further difficulties for DNS clients. A client cannot make a clear and unambiguous decision that either the query or the response has been dropped. All the client can do is wait for some interval of time and assume at that point that the lack of a response is indicative of a lost transaction. A very short re-transmit timer may well provide a faster resolution service during times of congestion. However, short re-query timers may make DNS transient congestion incidents worse. So it's possible that aggressive re-query timers are contributing to the DNS query load.

At this point we head into areas of DNS resolver misbehaviour to explain the residual high query rate for non-existent domain names. There are observed *resolver farm* behaviours where every resolver in a cluster that shares a common address prefix generates a query for the same name at much the same time. There are also observed resolver behaviours that drop into a high frequency query rate mode which appear to completely ignore the `NXDOMAIN` response and continue the high query rate irrespective of any responses it may receive.

Taking *happy eyeballs* into account, the good news is that 61% of resolution efforts for non-existent names are resolved in a single DNS query to the authoritative name server. It could be worse.

The not-so-good (bad) news is that 39% of resolution efforts for non-existent names are not resolved in a simple DNS query to the authoritative name server. This is not an impressive outcome.

We've been able to perform some categorisation of the query load for this simple test, and this is shown in Figure 11. We've been able to account for some 60% of the total query volume in this manner, while the remaining 40% are still unclear at this stage.

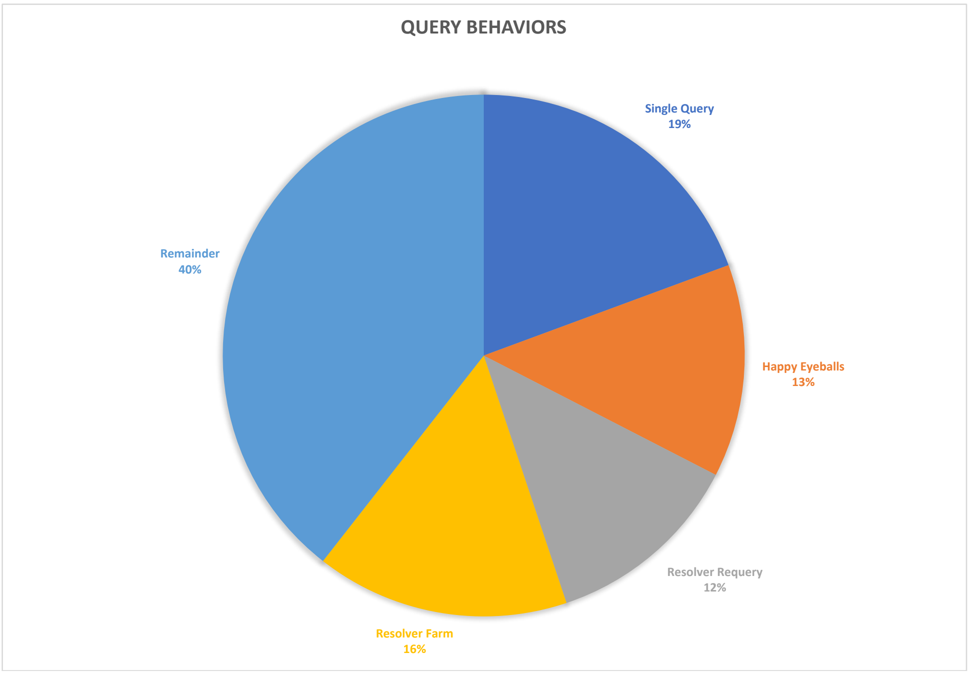

Figure 11 – Query Behaviour and relative Query Volume

What about the case where a name does exist and can be resolved?

Does the DNS understand a "YES" any better than "NO?"

It's better, but not by much. A name is resolved in a single query 73% of the time. But the other 27% of the time we observed two or more queries to complete the name resolution. On average it takes the DNS 1.65 queries to resolve a resolvable name.

It's an interesting question as to why a NO response generates more follow-up DNS queries than a YES. So far, we have no firm ideas as to what the answer might be.

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