The ISP Column
A column on things Internet

May 2022
Geoff Huston

The Path to Resolverless DNS

There is an intriguing mention of "Server Push" in the specification of DNS over HTTPS (DoH) (RFC 8484). The RFC is somewhat vague in the description of server push, apart from noting a caveat that "`extra care must be taken to ensure that the pushed URI is one that the client would have directed the same query to if the client had initiated the request (in addition to the other security checks normally needed for server push).`"

It's a somewhat elliptical reference to an intriguing possibility that a HTTPS server could deliver one or more DNS HTTP objects that contain both query and answer sections without the client ever making the DNS request/query to the server in the first place. It seems like an approach that is totally alien to the DNS as we know it, so it might be useful to ask: How did we get to this point where this resolverless form of DNS name resolution makes some sense? And, to whom does it make sense?

"Classic" DNS

Let's start with the "classic" DNS system architecture of name resolution.

We normally describe the DNS system architecture as having three distinct components, as shown in Figure 1.

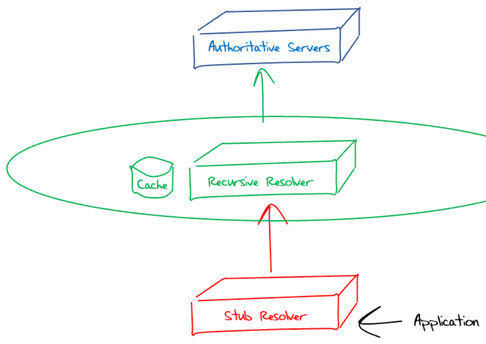

Figure 1 – "Classic" DNS System Architecture

In this model, an application resolves a DNS name by invoking a system function via a call to *gethostbyname()* or similar. This function call invokes the system's local DNS resolver function, which is commonly referred to as a *stub* resolver. The resolver may have a small local cache, and if the query can be answered from this local cache, then it will do so immediately. Otherwise, it will refer the query to a remote *recursive resolver*. In this classic model of the DNS the recursive resolution function is a service provided by the local Internet Service Provider (ISP), or in the case of an enterprise environment it may be provided by a server within the enterprise environment.

For public DNS recursive resolvers, the resolver is initially configured with information relating to the servers of the root zone (and the root zone Key Signing Key if this is a DNSSEC-validating resolver). When it receives a DNS resolution query it will first ask a root zone server to resolve this name. Unless the query is for a delegation entry of a top-level domain, the root server's response would be a *referral response*, providing the nameservers of the right-most (top) label in the query name. The recursive resolver would then follow this referral, querying one of these top-level domain authoritative servers with the same query name, and so on until the penultimate response is a referral to the authoritative servers of the domain name in question. The final query is to one of these authoritative servers, and the response it receives is passed back to the sub resolver as the answer to the query.

Of course, if this was the way the DNS really worked, then it would be so slow as to be completely useless! What makes all this tedious iterative discovery process viable is the cache used by recursive resolvers. The recursive resolver will only query an authoritative server if there is no current entry in its local cache. This applies equally to the authoritative server discovery process queries as it does to the final query. The more a resolver is used, the more valuable the role of the cache in terms of speeding up the name resolution process.

It's conventional for this DNS recursive resolution process to be operated by your ISP, as it's impractical for this function to be carried out by every host. Yes, it could be done, but the advantages of using a pooled cache of DNS responses would be lost and the DNS would revert to a slow and inefficient mode of operation. This consideration leads to the model of DNS operation as shown in Figure 2.

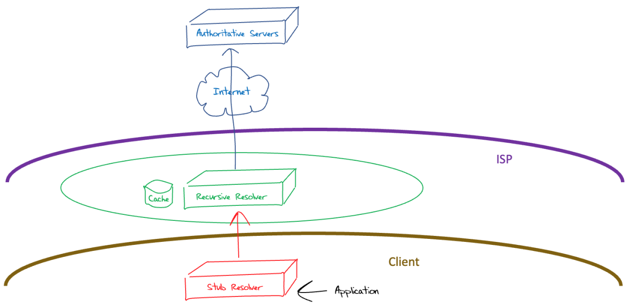

Figure 2 – "ISP" DNS System Architecture

The rise of Google's Open Resolver Service

There are some issues with this ISP-based approach to DNS resolution. Users do not directly pay the ISP per query, nor do they use DNS resolution performance as a criterion in selecting an ISP. If the DNS service is not a competitive distinguisher for an ISP, then the ISP can regard this provision of this DNS function as a direct cost, instead of being a revenue raising activity. Consequently, the ISP's DNS service may not be closely managed, and its performance as a DNS resolver may degrade over time.

It's not just the potential to sink into degraded modes of DNS service that is a problem with this model. From time to time, ISPs have tried to raise revenue from this recursive DNS service. They may look at selling their query logs to third parties. As the raw DNS query log data directly identifies the end clients and their current activities, these DNS logs are both incredibly valuable within the Internet's pervasive surveillance economy and represent incredibly sensitive data from a privacy perspective. Selling this query data puts the ISP into a conflicted position, in that while the additional revenue stream is tempting, the erosion of trust from the ISP's customers, if they were to be aware of this transaction, would probably be a significant liability for the ISP. In many regimes such an action is counter to regulatory provisions that apply to these service providers.

Browsers have also contributed to undermining this classic DNS model, although I suspect it was an unintended consequence. Early browsers used two input fields in their user interface. One was to enter URLs, which directed the browser to display the referenced content. The second was to enter free text into a search engine. Users were confused by this, and to simplify the browser interface, browsers all adopted a single input field. If the user entered a URL, or a domain name, then the browser interpreted this as a URL and displayed this content. Otherwise, the browser passed this input to a search engine, and displayed the search results. It shouldn't be a surprise to learn that this move created a major form of "cross-domain" leakage where search engines were seeing leaked malformed URLs and the DNS resolvers were seeing leaked search terms. It also contributed to a user perspective that melded the concepts of DNS names and search terms into a single more nebulous concept of identification on the Internet.

This has some further implications. If the user really cannot appreciate the difference between DNS names and search terms, then why not redirect a "no such name" response from the DNS into a search result page? Presumably, the search engine would pay the ISP some financial bounty for such referrals, and the ISP would be able to raise some revenue from operating a DNS resolver.

I've previously noted a case in Australia back in 2009 where a major ISP was observed performing this form of NXDOMAIN referral: https://www.potaroo.net/ispcol/2009-12/nxdomain.html

One significant party that was potentially impacted by this implicit merging of DNS resolution and search functions was Google, who have held a highly dominant position in search for many years, and for whom search remains the mainstay of their corporate asset base. Search provides the profile information to support more accurate ad placement algorithms which in turns provides the company with most of its revenue.

Google's response was rapid. Previously, open DNS resolvers were a fringe activity that was not very significant in the larger DNS world. These resolvers operated in a largely unfunded mode and thus were limited in the scale they could achieve before they reached the limits of donated effort and capital. Google bought the power of its infrastructure to bear onto DNS resolution and set up a large-scale open DNS resolver service that leveraged the entire distributed infrastructure of Google's network. Google made two public undertakings: firstly, that it faithfully responded with whatever was in the DNS (https://developers.google.com/speed/public-dns/faq), and secondly, Google would not use the query stream as a data source for ad placement, nor would they retain personally identifying information in such logs (https://developers.google.com/speed/public-dns/privacy).

Google's DNS resolution service was fast, well managed, accurate, didn't perform any form of filtering, and was free to use. Their service was an early adopter of DNSSEC validation. It performs query name minimisation and uses its own form of local root service. Given that Google only queries authoritative servers when the local cache does not already hold the DNS response, the greater the use of Google's service the more effective the local cache becomes, and the sparser the collection of queries that are sent to authoritative DNS servers. What this implies that Google's DNS service leaks little information out about the clients that send DNS queries to Google's service.

This service has proved to be very popular with many classes of Internet users. The lack of any form of DNS filtering is seen as a positive in same cases. The resolution environment is responsive due to a densely deployed service infrastructure. Google's service is part of the DNS resolution environment used by some 25% of users (Figure 3), although only 14% of users have placed Google's service at the head of their resolver query lists.

Perhaps the distinguishing factor here between Google and other open DNS resolver servers was that Google was able to back up its service with a business model that supported continued investment in the service. Further investment in this service was equivalent to investment in a model of DNS operation that clearly distinguished DNS resolution and search, which in turn was a direct benefit to Google's very dominant position in the search market. The business case for search within Google directly supported the business case for keeping the DNS as a function distinct from search, and Google's way of doing this was to operate a free-to-use, large scale and robust open recursive resolution service for the entire Internet.

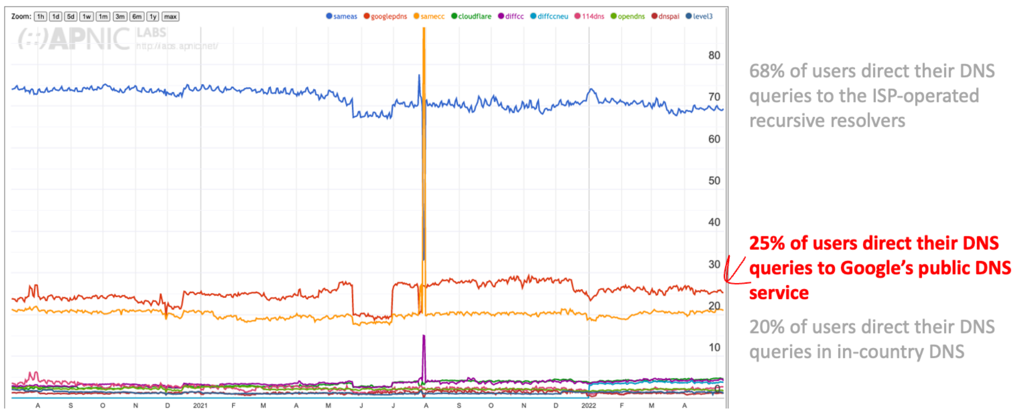

Figure 3 – Market share of Google's DNS service across all users

The Emergence of DNS Privacy

But at the same time, this model of remote DNS recursive resolvers poses its own additional privacy and integrity issues. By moving the DNS recursive resolver away from the ISP, the client traffic now traverses the Internet between the stub and recursive resolvers (Figure 4).

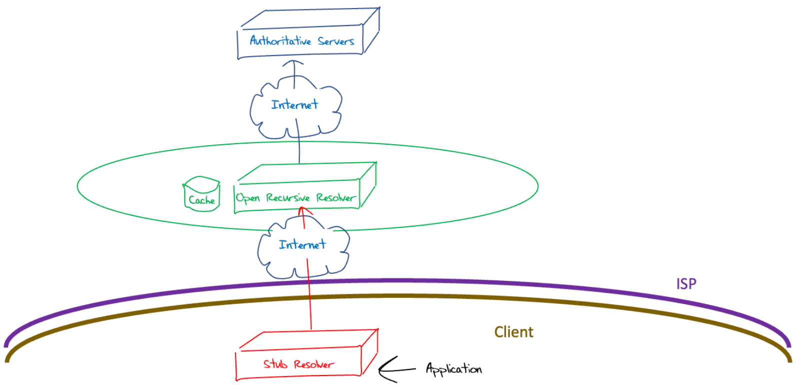

Figure 4 – Open resolver DNS System Architecture

The transaction flow between the stub resolver and the recursive resolver is a problem in any case, but particularly so when the traffic leaves the ISP and traverses the public Internet. These DNS transactions use unencrypted UDP, which allows for unconstrained third-party inspection and manipulation possibilities. Stub resolvers typically do not perform DNSSEC validation, so they are not well prepared to detect any such manipulation of DNS responses. DNS over UDP does not perform any form of server authentication, so the stub resolver is completely unaware when the service address of the open DNS resolver is redirected to another DNS resolver though some form of routing hijack. Moving DNS resolution away from the user opened a larger set of DNS issues around interference and surveillance.

The Snowden papers catalysed a significant reaction to digital surveillance in the technical community and the DNS has been the subject of much of this effort. The work to date has concentrated on the development of stub-to-recursive connections over encrypted transport session:

- **DNS over TLS** (DoT packages DNS transactions over a persistent TLS session over TCP (RFC 7858)
- **DNS over QUIC** (DoQ) packages DNS transactions over an encrypted QUIC session over UDP (RFC 9250)
- **DNS over HTTPS** (DoH) uses DNS over HTTP/3 (over QUIC) where supported, and DNS over HTTP2 (over TLS) otherwise (RFC 8484)

What is common to these approaches is the ability of the stub resolver to authenticate the identity of the recursive resolver, mitigating various forms of session interception. Obviously, it cannot prevent interception, but the client can detect an attempt to masquerade the intended resolver and will not bring up the session. Session encryption resists various forms of middle attacks that attempt to tamper with the DNS response. There are no UDP fragmentation and TCP failover issues in these encrypted session models. TCP and encryption present some unavoidable overheads, but these can be amortised through long held sessions and TCP Fast Open, so the higher session setup costs are amortised over the volume of subsequent queries. The Head of Line blocking issues associated with serialising multiple queries over TCP can be avoided by using DoQ or DoH.

DNS over HTTPS

These DNS session encryption measures can mitigate the additional risk factors when the stub and recursive resolvers are separated by an Internet transit. However, both DoT and DoQ are functionally adequate in this respect. So, why is DoH a thing?

One view is that DoH is completely unnecessary. Even though DoH effectively embeds the DNS queries and responses within the same network profile as all other HTTP traffic, the subsequent traffic flows would allow an eavesdropper to infer the original DNS transaction in any case. A more encompassing approach would be to place all traffic in an encrypted session, as in the model of a VPN, but if a VPN is being used then DoH adds very little to the security properties of the VPN setup.

However, not only does DoH use port 443, along with web traffic, it also can use generic HTTP caching controls to manage the local retention of DNS responses. HTTP/2 and HTTP/3 both include support for parallelism, reordering of responses and header compression. Applications do not need to use the local stub resolver and can direct its DNS queries to an HTTP server of its choice. And, importantly, HTTP/2 and HTTP/3 both include "Server Push":

> `HTTP/2 allows a server to pre-emptively send (or "push") responses (along with corresponding "promised" requests) to a client in association with a previous client-initiated request. This can be useful when the server knows the client will need to have those responses available in order to fully process the response to the original request.`
>
> RFC 7540, Hypertext Transfer Protocol Version 2 (HTTP/2)

A similar function exists in the HTTP/3 specification:

> `Server push is an interaction mode that permits a server to push a request-response exchange to a client in anticipation of the client making the indicated request. This trades off network usage against a potential latency gain. HTTP/3 server push is similar to what is described in Section 8.2 of [HTTP2], but uses different mechanisms.`
>
> `Each server push is assigned a unique Push ID by the server. The Push ID is used to refer to the push in various contexts throughout the lifetime of the HTTP/3 connection.`
>
> draft-ietf-quic-http, Hypertext Transfer Protocol Version 3 (HTTP/3)

This means that when a server sends a response to an HTTP request, the server can also package and push unrequested HTTP objects to the client. Conventionally, this would be expected to include HTML style sheets, images and other elements of a compound collection of related objects. Because of DOH, this can also include HTTP-packaged DNS responses. The client can use these DNS results immediately and bypass DNS resolution step and its associated delays and privacy implications. There is no DNS resolver call, and no DNS meta-data associated with this form of DNS name resolution.

The client has a problem, however. How does it know that the server is pushing an authentic DNS response? Securing the transport session means that the path between the server and client is resistant to third party tampering, but there is no assurance that the server is passing correct DNS information. Of course, this uncertainty is not unique to this form of resolverless DNS, and all clients have the same challenge of establishing the authenticity of DNS information.

The only answer we have that addresses this challenge is DNSSEC. If the name is not DNSSEC-signed then the client simply cannot verify the DNS data, whether it has been passed to the client as an HTTP object via server push or from a DNS resolver by a conventional query/response transaction. If the name is DNSSEC signed, then in conventional DNS the client sets the EDNS0 DNSSEC OK flag in the query to signal to the resolver to include the DNSSEC signatures in the response. The client then needs to authenticate this signature, by performing a series of DNS queries for the DNSKEY and DS records that allow a validation chain to be constructed from the original RRSIG record to the root zone KSK. These additional DNS queries consume additional time, and perhaps it's no surprise that few stub resolver clients perform DNSSEC validation themselves.

In the DoH server push model it's up to the server to include the RRSIG record in the pushed DNS object, and also up to the server push the sequence of DNSKEY and DS DNS objects that the client will need to construct the validation chain up to the root if it really wants the client to eschew all direct DNS queries when resolving this name.

This seems to be an unnecessarily long path to the validation destination. A more efficient approach is for the server to bundle all of these validation DNS records into an EDNS0 Chain Response (RFC 7901). While this approach is generally inefficient in DNS over UDP, as the large response would normally imply that the client should move to TCP to perform this query, the nature of DoH means that the DNS data is already packaged in a reliable, encrypted, streaming transport of TLS or QUIC. The larger volume of DNS response data is not a problem in this case and Chain Responses are viable within the framework of the DoH transport model.

DNSSEC validation provides the client with assurance that the DNS data is authentic, current and complete for each RRTYPE. If this is the case, then why should it matter how the client learned the data? If DNSSEC validation is successful, then its DNSSEC that is providing the necessary assurance that the DNS data is usable. Trust in the authenticity of the data is not based on whether a server pushed the data or the data was obtained as a result of a direct DNS query. Once a DNS record is validated with DNSSEC is does not matter how the data was learned.

If the DNS data is unsigned then all questions about the authenticity of the data are unanswerable. In a server push scenario, the client has no visibility as to how the server learned the DNS data in the first place, nor when. The client has no way of assuring itself that the server is not trying to deceive it, and if it elects to use the data it has no way of understanding in whom it is implicitly trusting. When the DNS data is unsigned the client is probably well advised to discard server push data, and the server is probably well advised not to push it in the first place.

Given all these caveats then why is Resolverless DNS interesting? I would suppose that the answer is that this approach gives HTTP-based applications and services far more control over the quality of the user experience. It allows the server to pre-provision the client with DNS data that is likely to be useful in the context of the application. It allows the client-side application to avoid the delays of conventional name resolution and also perform rapid DNSSEC validation without relying on the local stub resolver's capabilities and settings, and without making any on-demand DNS queries. Resolverless DNS can replace UDP-based timers, query retries, fragmentation and TCP switching with server-to-client provision. It operates over a secured connection with an authenticated server. All of this is achievable as long as the DNS data is DNSSEC-signed in the first place.

If the DNS data is unsigned then there is little to say to justify its use. The approach appears to open up more sources of vulnerability for clients with little in the way of reasonable mitigations.

Signalling DNSSEC

There is, however, one problem here, and that's DNSSEC.

DNSSEC is a tough problem for the Internet.

While measurement approaches are challenging in this space, what we have been able to measure has not been encouraging. Adoption rates of zone signing are low. Part of the issue here is the challenges of robust key management procedures. Provisioning authoritative Servers with pre-signed zones is stable for small zones with stable zone contents. Larger zones tend to use on-the-fly signing which increases the complexity and cost of serving data. Processing signed DNS data can be an issue with UDP-based resolution mechanisms because of the issues with handing large responses and the additional query overheads of DNSSEC validation. The prevalent position is that signing a zone with DNSSEC presents a set of costs and additional risks which are not offset against the assessment of the risk that DNSSEC is mitigating. And most stub clients do not generally validate DNS responses in any case, so if the risk model for tampering includes the stub-to-recursive path, as is the case with open DNS resolvers, this not generally addressed in any case.

Early in the days of DNSSEC deployment recursive resolver operators enabled the DNSSEC OK EDNS0 flag by default in all their queries, and this has remained the case through to today. Even if the resolver is not performing DNSSEC validation it is still pulling down RRSIG signature records when the name exists, and signed NSEC/NSEC3 records in the case when the name does not exist. While some 30% of Internet users sit behind recursive resolvers that perform DNSSEC validation a far higher rate of more that 70% of users sit behind recursive resolvers that set the DNSSEC OK flag.

What if we alter this resolver behaviour and push the workload of assembling the validation material to the server side of DNS? What would happen if we could use a slightly different flag to DNSEC OK? What if we had a ENDS0 DNSSEC Chained Response flag in a query that would request a RFC 7901 Chained response? It would be a useful setting if the client was using a TCP or QUIC transport, and would be ideal in the context of a server assembling DNS data to serve over Resolverless DNS. This approach pushes the overheads of collecting DNSSEC validation data back towards the server of the data rather than loading the client with additional delay through the imposition of additional DNS queries.

Why is Resolverless DNS Interesting?

Perhaps a small excursion into the changing nature of the economics of the Internet is useful at this stage to some of this into a larger context.

The "value" within the Internet is concentrated in the area of provision of content and services. The application layer appears to be the predominate generator of value. This gain in value appears to have occurred at the cost of the lower protocol layers of access and common infrastructure. Common infrastructure provision, including the DNS, is stabilising as a commodity-based activity that is resisting taking on incremental costs of further innovation. The pressures here are pressures to achieve further efficiencies and there is a pressure to achieve economies of scale by centralising service provision in infrastructure in fewer entities. The incentives to invest in further innovation lie at the application and service layers.

Resolverless DNS plays directly into this scenario. Resolverless DNS is not a change in the common infrastructure of the DNS itself. It's a change in the way applications can use the DNS to increase the level of control applications have over the user experience with the application. From this perspective, it appears to be a natural fit for applications.

We have spent a huge amount of effort over the last decade trying to make the Internet faster and more robust:

- We've been deploying CDNs to replicate content and services and bring them closer to users.

- We've been deploying non-blocking transport protocols (such as QUIC) to exploit parallelism.
- We've been tuning TCP and network behaviour to create more efficient and faster network transactions.
- We've been packing more information in the DNS to make service startup faster (SVC and HTTPS records).

Yet the DNS is dragging its feet in all this. The DNS remains a source of:
- time penalties,
- privacy leakage, and
- obscure modes of failure.

Resolverless DNS won't fix all of the DNS all at once. That was never going to be possible! But it can hand a significant amount of control over application and service quality back to HTTPS-based applications and services It can be engineered to be a whole lot faster than classic DNS! It can obscure the end client from the resolver-based DNS infrastructure. It can constrain the potential for failures. For those reasons it's a very interesting step in the potential evolution of the DNS.

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