July 2020
Geoff Huston

IPv6 and the DNS

These days it seems that whenever we start talking about the DNS the conversation immediately swings around to the subject of DNS over HTTPS (DoH) and the various implications of this technology in terms of changes in the way the DNS is used. It's true that DoH is a rich topic space and while much has been said and written already, there is still more to say (and write). But that's not my intention here. I'd like to look at a different, but still very familiar and somewhat related, topic relating to the DNS, namely how IPv6 is being used as a transport protocol for DNS queries.

Before I set out the questions that I want to consider here, it's useful to remember why we are deploying IPv6 in the first place. IPv6 is not intended to sit alongside IPv4 in a dual-stack situation as the end objective of this transition process. A fully deployed dual-stack world is not the goal here. We need to push this transition process one step further, and the objective is to get to the point where IPv4 is not only no longer necessary but no longer used at all. In other words, we are attempting to get to the point where IPv6-only services are not only a viable way of using the Internet but are the only way of using the Internet.

One of the key elements in the overall transition is the way in which we manage a dual-stack networked environment. The basic approach is to bias the choice of IP protocol towards IPv6 when both protocols are viable for a connection. In the context of browser behaviour, we call this preference to use IPv6 "Happy Eyeballs". Where the client and the server are both operating in a dual stack mode, they should prefer to communicate using IPv6. The theory is that the combination of this preference and the increasing adoption of dual-stacked environments is that the use of IPv4 automatically diminishes as the dual stacked world becomes more pervasive. We won't need to "turn off" IPv4 at any particular point as this preferential process will cause the use of IPv4 to disappear quite naturally. But while this preferential protocol behaviour is used in a number of browsers, it's not necessarily used in other applications and infrastructure services. The larger question of IPv6 transition is more than browsers, and we should look more broadly at the use of IPv6. Obviously, this includes an examination of the DNS and its use of IPv6 as the transport protocol for UDP and TCP.

To answer the question of how well IPv6 is supported in the DNS today, there are three aspects to this question that need to be examined:

- How does the DNS handle dual-stacked authoritative servers?
  Is there a "happy eyeballs" version of DNS server selection?
  Or is there a reverse bias to use IPv4?
- If you placed authoritative servers on an IPv6-only service how many users would be able to reach you?
- And what about DNSSEC? How well does IPv6 support large UDP packets?

Challenges in Measuring the DNS

Before looking at the measurements that can answer these questions it might be useful to remember that the DNS is actually very challenging to measure. When we say that some proportion of the DNS exhibits some particular behaviour, are we talking about DNS resolvers? Or maybe we are referring to a

proportion of observed queries? Or perhaps to an inferred set of end systems, or even end users? Each of these present some issues in definition and interpretation.

Resolvers?

We really don't understand what a *resolver* is in the context of a measurement.

A DNS resolver might be a single instance of resolver software running on a single host platform. But it also might be a single front-end client-facing system that accepts DNS queries and uses a query distributor to pass the query onto one (or more) 'back-end' resolvers that perform the actual DNS resolution function. Each of these DNS resolver engines could operate autonomously, or they could be loosely coupled with a shared cache. The system might also use a collection of these server "farms" all configured behind a single anycast front end service address.

There are various types of resolvers, including *stub resolvers* that are typically part of end host systems, there are *forwarding resolvers* that pass queries on to other resolvers, as well as *recursive resolvers* that attempt to perform a full resolution of a DNS name.

A resolver might have a single user as a client, or at the other end of the range it may have millions or even billions of end users as clients, and of course there are many resolvers in between these two extremes.

Claims such as "30% of resolvers show some particular behaviour" are essentially meaningless statements given that there is no clear concept of which resolvers are being referred to, and how many users are impacted by whatever behaviour is being referenced.

When we talk about *resolvers* in the context of measurement it is unclear what we might be referring to!

Queries?

Again, it might sound odd, but we really don't know what a *query* is.

An end client might generate a single DNS query and pass it to a resolver, but that query may cause the resolver to generate further queries to establish where the zone cuts exists corresponding to delegations and the addresses of the authoritative name servers for the DNS zone, and possibly further queries if the resolver performs DNSSEC validation. Prior to *qname minimisation* a resolver would pass the full query name to each resolver when performing this top down zone cut discovery, and there is no essential difference between these discovery queries and the ultimate query. The essential characteristic of the DNS is that a single DNS query may generate a cascade of queries in order to amass sufficient information to generate a response to the original query. Viewed from the 'inside' of the DNS it is impossible to distinguish between what could be called 'discovery' queries and the 'original' query, whatever that might be.

Resolvers all use their own timers, and if they do not receive a response within the local timer period the resolver may generate further queries, both to the same resolver and to other resolvers. There is also a DNS error code that may cause a re-query, namely the SERVFAIL response.

DNS queries have no hop count attribute nor any resolution path attribute, so queries have no context that will provide the reason why a resolver has generated a DNS query, nor provide any indication of what the *root cause* original query may have been.

It appears that queries have a life of their own, and while there is some relation to user and application activity it is by no means an exclusive relationship, and DNS queries are often made without any such triggering event.

Statements such as "30% of queries have some particular property" are again somewhat meaningless as a measurement statement given that there are many forms of queries and many potential causes for such queries to be generated.

Once more it appears to be the case that when we talk about *queries* in the context of measurement it is unclear what we actually mean.

APNIC Labs Measurements

At APNIC we use a measurement system that is based on a measurement of end user behaviours rather than infrastructure behaviours. The measurement system is based on the use of online advertisements to distribute a client-side measurement script. The platform is capable of 'seeding' the DNS with queries that use unique labels, and we observe the queries that are placed with the measurement platform's authoritative servers in response. We can then correlate end user activities and authoritative server query activity and measure the DNS in terms of users and behaviours (Figure 1).

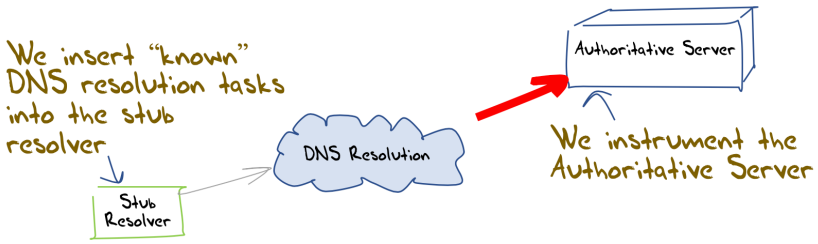

Figure 1 – APNIC Labs DNS Measurement Framework

The intent of this measurement system is to measure a particular behaviour or property by measuring its visibility to users. We want to understand to what extent users are impacted by a particular behaviour or property of the network infrastructure.

In this exercise of measuring IPv6 in the DNS we will be looking at the DNS and measuring the population of users that generate DNS activity over IPv6.

DNS Happy Eyeballs?

The question we are looking to answer here is: How does the DNS handle a dual stack environment? Do we observe a "happy eyeballs" preference to use IPv6? Or is there a reverse bias to prefer to use IPv4?

In this case we are looking at the interaction between the user's recursive resolvers and authoritative servers. In this measurement experiment the authoritative server was configured to respond to DNS queries over both IPv4 and IPv6 and the address records for the delegated authoritative server had both IPv4 and IPv6.

For each execution of the measurement script on a end-user client device we examine the collection of DNS queries and look at the protocol used to perform the query. Some experiments will generate a single query, while others generate multiple queries. We will sort the experiments into three categories:

- experiments where all the queries between the visible recursive resolver and the authoritative server were made using IPv6
- experiments where all the queries were made using IPv4
- experiments where we observed queries for the DNS name made over both protocols.

The data collected over the last week in April 2020 shows that 15% of tested end users used recursive resolvers that only used IPv6 to query the experiment's authoritative servers. A further 18% used multiple queries that used both IPv4 and IPv6 to make these queries. The remaining 67% used only IPv4 in their queries as seen at the authoritative server.

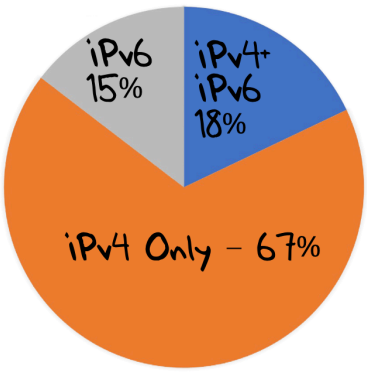

Figure 2 – Breakdown of protocols used by experiment

We can look at this data over time, looking the same data for the past 9 months on a day-by-day basis.

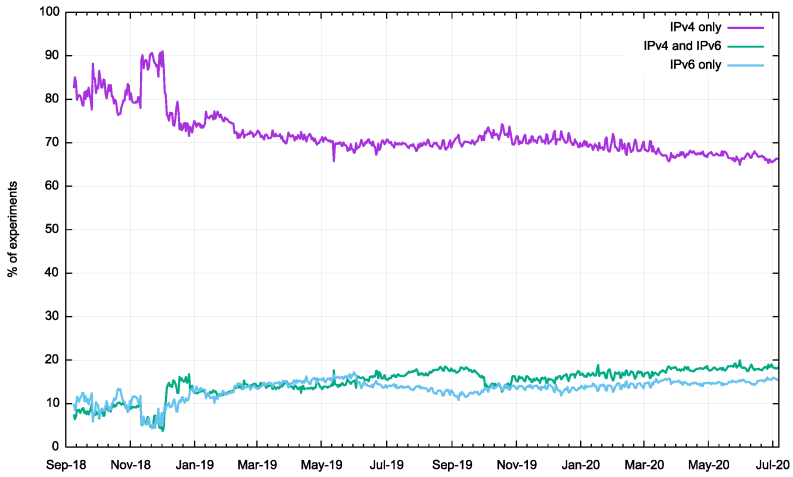

Figure 3 – Breakdown of protocols used by experiment for the past 22 months

Over this period the proportion of users behind resolvers using only IPv4 has fallen from 80% to 67%. At the same time the proposition of IPv6-only activity rose from 10% to 15%.

Where we see both IPv4 and IPv6 queries in the same experiment we've split them to show which protocol was used for the first query.

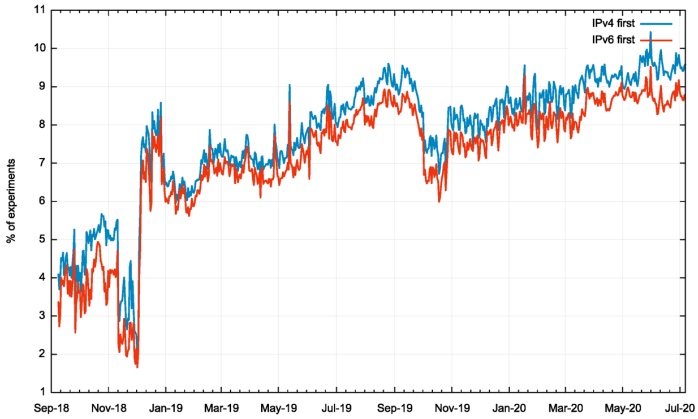

Figure 4 – First query protocol for the past 22 months

What this shows is that where we see cases of both IPv4 and IPv6 queries, there is a constant but relatively small bias to use IPv4 for the initial query.

We can now provide an answer to the first of our questions. It's clear that the DNS does not prefer to use IPv6 in dual stack situations. There is no "Happy Eyeballs" for the DNS. There is also a small but visible reverse bias where IPv4 is favoured over IPv6 in a small number of cases where both protocols are observed in queries.

IPv6-only DNS Servers

The data in Figure 3 indicates that 67% of users sit behind DNS resolvers that send queries to authoritative servers using IPv4 only. That might lead to an inference that if we were to set up a server on an IPv6-only platform then only some 43% of users would be able to resolve DNS names that are served from this server.

However, when we set up an IPv6-only experiment in parallel to the dual stack experiment the results are somewhat different. Using the same measurement framework and the same set of end users for the measurement, we can also direct users to a URL where the only way to access the server of the DNS label is using IPv6. Of all the end clients who are presented with such a URL, in this case some 45M such clients over a 5-day period at the start of July 2020, some 55% of users are using DNS resolvers that can make a DNS query using IPv6 (Figure 5).

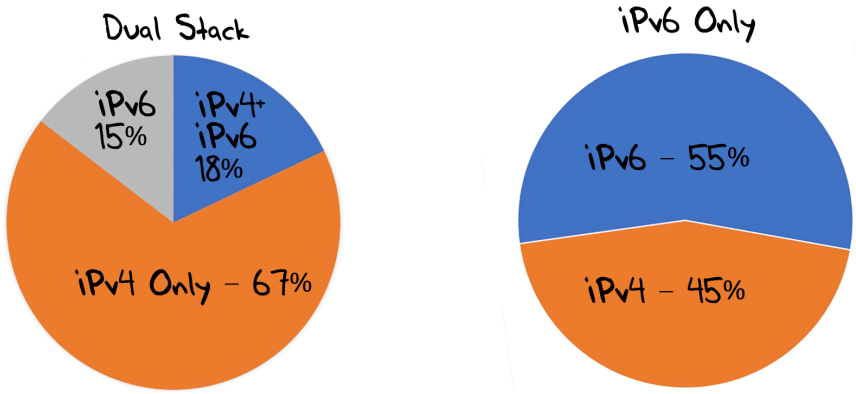

Figure 5 – Dual Stack vs IPv6 only in the DNS

Given that the level of IPv6 capability in the end user population as a whole in July 2020 is some 25% (https://stats.labs.apnic.net/ipv6/XA), the DNS IPv6 measurement of 55% is certainly an encouraging level of progress in the adoption of IPv6 capability in DNS infrastructure.

The current answer to our second question, namely: "If you placed authoritative servers on an IPv6-only service how many users would be able to reach you?" is 55% of users.

Large DNS packets and IPv6

The final question concerns large packets in the DNS. This is relevant when considering the use of DNSSEC, where a DNS response may contain the digital signature of the DNS data as well as the data itself, and this can have an impact on DNS responses sizes.

By default, the DNS is a UDP-based protocol. The transactional nature of the DNS protocol seemed to be a perfect fit for UDP. However, there are limitations in datagram-based services, particularly as they relate to larger payloads. IP packet fragmentation is an unreliable approach. The lack of transport protocol headers in trailing fragments creates a quandary for firewalls. Admitting trailing fragments admits the possibility of a number of attack vectors, while discarding them may disrupt a service. The DNS adopted a very conservative position in UDP. A DNS response over UDP is no longer than 512 octets. If the response is larger than this size, the DNS response packet is truncated such that it is no larger than 512 octets, and the truncation bit is set in the response to flag the fact that the response has been truncated. A DNS resolver should treat this truncation bit as a signal to re-query the server using TCP, so that the larger response can be handled by TCP.

> This 512-octet limit for DNS packets over UDP seems absurdly small today, but it comes from the original, and still extant, standard specification of IPv4. A standards-compliant host must be capable of accepting an IPv4 packet of at least 576 octets in length. The packet may be fragmented to smaller fragments, so this 576 octet lower limit applies to packet reassembly as well.
>
> What this implies is that an IPv4 packet larger than 576 octets may not necessarily be accepted by a host. Given this, the DNS opted for an approach of explicit signalling for larger payloads rather than the slower and indeterminant process of using timeouts to infer size-based packet loss, Payloads larger than 512 octets are truncated in DNS over UDP to explicitly signal to the other end that TCP is required to send the complete response.

If we had stuck to this original approach to handing DNS over UDP then large UDP packets wouldn't present a problem for the DNS. But the extended time needed to discover that TCP is necessary and then setting up a TCP session to re-query the server imposes a time penalty on the client and a load penalty on the server. If the major motivation for supporting large DNS responses was concentrated in the need to support DNSSEC, then it's likely that DNSSEC would never have been deployed under such conditions.

What we did instead was to include an extension option in the DNS query to allow the client to inform the server that a large UDP response can be handled by the client. The Extension Mechanism for DNS (EDNS(0), RFC 6891) includes a UDP buffer size option that informs the server of the size of the client's maximum UDP receiver buffer. This option allows the server to send UDP responses up to this size without truncation. Of course, the problem is that neither the client nor the server necessarily know the characteristics of the path from the server to the client. In IPv4 it is possible for the packet to the fragmented while in flight so this issue of path MTU mismatch is not a forbidding concern. But IPv6 does not have fragmentation on the fly. The router that cannot forward the large IPv6 packet is supposed to send an ICMPv6 PacketTooBig message back to the packet's source address. But this is UDP, so the source has no retained record of the packet and cannot resend the packet in smaller fragments. The transport layer of the host can do nothing to salvage the situation. It's largely left to the application to try and climb out of this hole though application timeouts. It's an unsatisfying solution.

But the IPv6 problems with packet fragmentation don't stop there. The IPv6 protocol design uses the concept of variable extension headers that are placed between the IP header and the UDP header. This means that the UDP transport header is pushed back deeper into the packet. Now the theory says that this is irrelevant to the routers in the network as the network devices should not be looking at the transport header in any case. The practice says something entirely different. Many paths through the Internet today use load distribution techniques where traffic is spread across multiple carriage paths. In order to preserve the order of packets within each UDP or TCP flow the load distributor typically looks at the transport header to keep packets from the same flow in the same path. Extension headers and packet fragmentation make this a far more challenging task to perform at speed as the transport header is no longer at a fixed offset from the start of the header, but at a variable location based on unravelling the extension header chain. Some network devices push such packets away from the ASIC-based fast path and queue them up for CPU processing. Other devices simply discard IPv6 packets with extension headers.

It appears that UDP packet fragmentation and IPv6 don't go well together. So perhaps we should just avoid large fragmented DNS packets altogether. However, it's easy to find large DNS responses in UDP.

In the past, I always liked to use the **.org** domain as the classic illustration of a fragmented UDP packet. Whenever you queried the **.org** servers for the DNSKEY record of the domain, with the DNSSEC OK flag turned on, the signed response was 1,625 octets:

```
$ dig +dnssec DNSKEY org
; <<>> DiG 9.8.3-P1 <<>> +dnssec DNSKEY org
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 21353
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 7, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 512;
; QUESTION SECTION:
;org.       IN          DNSKEY

;; ANSWER SECTION:
org.        861         IN          DNSKEY      256 3 7 AwEAAXxsMmN/JgpEE9Y4uFNRJm7Q9GBwmEYUCsCxuKlg
org.        861         IN          DNSKEY      256 3 7 AwEAAayiVbuM+ehlsKsuAL1CI3mA+5JM7ti3VeY8ysmo
org.        861         IN          DNSKEY      257 3 7 AwEAAZTjbIO5kIpxWUtyXc8avsKyHIIZ+LjC2Dv8naO+
org.        861         IN          DNSKEY      257 3 7 AwEAAcMnWBKLuvG/LwnPVykcmpvnntwxfshHlHRhlY0F

org.        861         IN          RRSIG       DNSKEY 7 1 900 20170815152632 20170725142632 3947
org.        861         IN          RRSIG       DNSKEY 7 1 900 20170815152632 20170725142632 9795
org.        861         IN          RRSIG       DNSKEY 7 1 900 20170815152632 20170725142632 17883

;; Query time: 134 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Mon Jul 31 12:07:1[illegible] 2017
;; MSG SIZE  rcvd: 1625
```

The response to a DNSKEY query for .org used a response of 1,625 octets!

However, that was back in 2017, and these days a similar query generates a response of just 1,058 octets. With my favourite example now 'fixed' I've had to look further afield for large DNS responses that force UDP fragmentation to happen. Thankfully, the DNS root zone is still full of such domains. Here's the largest of the DNSSEC-signed responses to a query for the DNSSEC field of entries in the root zone:

.booking	2,932
.winners	2,932
.watches	2,932
.ferrero	2,932
.lincoln	2,932
.chintai	2,932
.citadel	2,932
.oldnavy	2,932
.banamex	2,932
.farmers	2,932
.athleta	2,932
.jpmorgan	2,937
.discover	2,937
.homegoods	2,942
.marshalls	2,942
.analytics	2,942
.homesense	2,942
.statefarm	2,942
.swiftcover	2,947
.xn--kpu716f	2,952
.weatherchannel	2,967
.bananarepublic	2,967
.americanexpress	2,972
.gdn	3,033

Some 1,420 labels in the root zone (which has a little under 1,350 DNSSEC-signed entries) generate UDP responses larger than 1,500 octets, which inevitably will be fragmented (Figure 6).

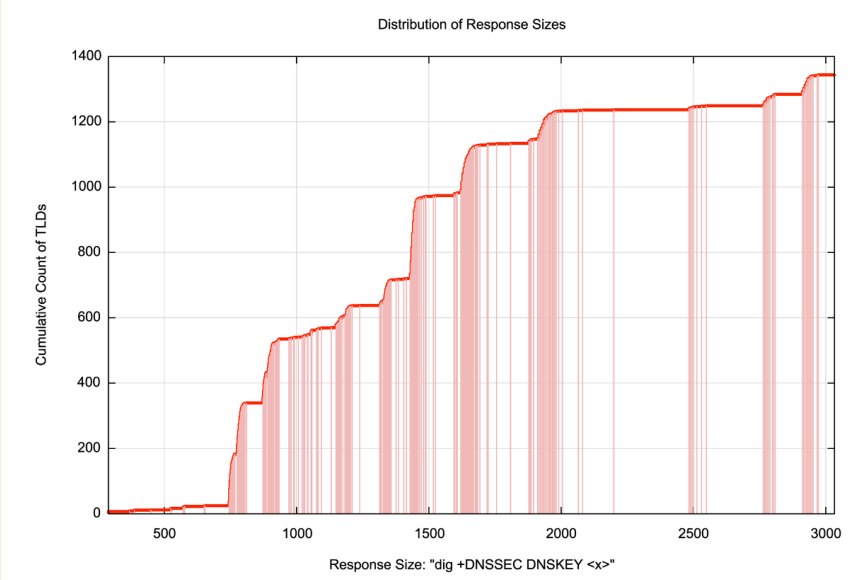

Figure 6 – Distribution of DNS responses sizes in the DNS Root Zone

It is evident that large DNS responses, UDP and IPv6 may encounter problems when used together. Can we quantify these problems? If an authoritative name server is only accessible using IPv6 and the server's responses are all larger than 1,500 octets, so they will all be fragmented then what proportion of user's will be impacted?

How can we tell whether a DNS response was received or not?

There are many ways to do this is you have full control of the end systems, but in our context we have a very limited repertoire of instructions. We can request that the end client fetches a URL and we can get the client to report on the success or otherwise of this operation. And that's about it! We can get the client to use one of "our" domain names, and if the name is unique then we will see the query at our authoritative name server, but we can't tell directly if the client received the response.

The way we perform this measurement is to create a short sequence of DNS queries, where within the sequence the next query is made only if it received a response from the previous query. The DNS behaviour we leverage for this is the DNS discovery process. This process starts with a query to the root servers, and the response lists the servers for the relevant top-level domain, and their IP addresses. It then asks one of these servers and the response will indicate the servers for the next level down, and so on. (Thankfully, caching means that a resolver will not actually perform most of these queries most of the time.) If the IP addresses are not included in the responses to these discovery queries, then the resolver will have to pause the original discovery process and commence a new discovery process to discover the IP address of one of these servers. If this task completes then the resolver will recommence the original discovery task.

This allows up to determine if a particular DNS response is received or not (Figure 7).

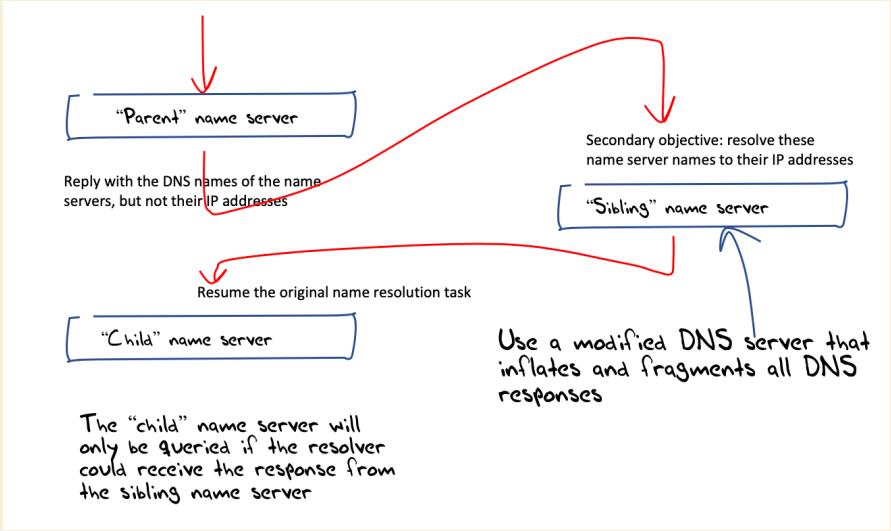

Figure 7 – Glueless Delegation

In a measurement performed at the end of April 2020 we performed this experiment some 27M times and observed that in 11M cases the client's DNS systems did not receive a response. That's a failure rate of 41%.

And that's the answer to our third question. How well does IPv6 support large DNS responses? Not well at all, with a failure rate of 41% of user experiments.

What to Do?

If we accept the prospect of an IPv6-only Internet, we are going to have to take DNS over IPv6 far more seriously than we are doing now. Today the dual-stack Internet comes to the rescue and what does not or cannot happen in IPv6 is seamlessly fixed using IPv4, but that's not a course of action that will be sustainable forever. We really need to address this appalling packet drop rate for fragmented IPv6 packets in the DNS, and on all end-to-end IPv6 paths in the Internet. Our choices are to either try and fix this problem in the switches in the network, or we can alter end systems and applications to simply work around the problem.

As ever, it's always amusing to consult the RFC document series to see what advice others are offering. In March 2017 the IETF published RFC 8085, with the grandiose title of "UDP Usage Guidelines":

> "Applications that do not follow the recommendations to do PMTU/PLPMTUD discovery SHOULD still avoid sending UDP datagrams that would result in IP packets that exceed the path MTU. Because the actual path MTU is unknown, such applications SHOULD fall back to sending messages that are shorted that the default effective MTU for sending (EMTU_S in RFC1122). EMTU-S is the samller of 576 bytes and the first-hop MTU [RFC1122]. For IPv6 EMTU-S is 1280 bytes [RFC2460]." RFC 8085

Well that seems to be clear – for the DNS over UDP, this RFC is suggesting that the DNS should not use EDNS(0) UDP buffer size, and DNS implementation should keep all UDP DNS packets below 512 octets if it wants to operate consistently over both IPv4 and IPv6.

But there is still some residual uncertainty here. Is this about the IPv6 manner of IP packet fragmentation, or about the more general matter of the use of Extension Headers in IPv6? The lingering suspicion is that while packet fragmentation has issues, IPv6 Extension Headers are their own problem, and we might want to avoid IPv6 Extension Headers completely!

What can we do about this?

It seems to be too late to change the protocol specification of IPv6 and re-think the way extension headers are handled in the protocol header of IPv6 packets.

It seems too challenging to change the hardware design process and process packets with extension headers at high speed inside dedicated ASICs in network equipment. Variable length fields in packet headers ask too much of the deployed packet processing equipment. If the trade-off here is between high capacity and high speed on the one hand and support of IPv6 extension headers on the other, then extension headers have to go the same way as the IPv4 header options. They're gone.

Unfortunately, by writing off IPv6 extension headers as unsupportable we've also written off IP level packet fragmentation in IPv6. The pragmatic observation is that the maximal packet size that can be reliably passed in IPv6 is 1,280 octets, or a UDP payload of 1,232 octets. The implication for the DNS is that if we can't stuff arbitrarily large payloads into UDP DNS packets in IPv6, and we don't want to pay the time penalty of sending the query over UDP and waiting for a truncated UDP response to trigger a subsequent query over TCP than what other options are open to us?

We could drop DNS over UDP completely and shift DNS to use TCP only. When this option has been aired in networking circles reactions to this proposition are mixed. Some observe the current profile of small TCP transactions in the web and pose the question of why the DNS folk should think that what the web folk are already doing is so hard! Others look at the additional time and load overheads of maintaining TCP state in resolvers and confidently predict the collapse of the DNS if we shifted the entirety of the DNS to TCP.

Variants of this approach include DNS over TLS over TCP (DoT), or DNS over HTTPS (DoH) (which itself is TLS over TCP) even DNS over QUIC (which is in effect DNS over HTTPS over TLS over TCP over payload encrypted UDP) and the issues with all of these approaches are largely the same as with DNS over TCP. Some folk believe that the shifting patterns of web usage point to the viability of this as an option for the DNS while others equally confidently predict the demise of the DNS if we head down this path.

Is there a middle ground here? Could we add path MTU discovery to UDP? It's certainly possible, but at that point the time penalties for the client are potentially worse than just using TCP, so there seems to be little value in that approach.

I always liked ATR (Additional Truncated Response) as a hybrid response. The idea is simple: when a server sends a fragmented response over UDP, it sends an additional UDP packet following the response, namely a response with an empty response section and the truncation bit set. If the fragmented response is lost the unfragmented truncated answer will follow. We measured the effectiveness of this approach in 2018 (https://www.potaroo.net/ispcol/2018-04/atr.html). The conclusion was that in 15% of cases the "fast" signal to use TCP would improve the client experience. However, ATR has not gained popular acceptance in the DNS implementors' world so far, and at the same time as ATR was floated the "DNS Camel" conversation had taken hold in the DNS community. That conversation asserts that the DNS is now so heavily ornamented with hacks, exceptions, special cases and features that the result is all but unsustainable. Invoking the "DNS Camel" is the same as saying that we should avoid adding further bells and whistles to the DNS.

The problem is that we can't have it all. If the underlying IP and transport protocol cannot manage packetization effectively (and packet fragmentation falls into this category) then its left to the application to patch up the problem for itself. The DNS can't just throw the problem of substandard support for large packets back to the network and transport layers and just simply wish it away. IPv6 is not going to get any better in this respect, so the DNS itself has to adapt to this reality, camels or not.

Where Now?

It seems that we now have a pretty decent idea of the problem space with DNS over IPv6 and we know what we would like to solve. How to solve these issues is the question. Poor experiences with orchestrated changes and flag days leads to a strong preference for a plan that allows every service operator to work independently in an uncoordinated manner.

We pretty sure that it's not feasible to clear out the various ICMPv6 packet filters, the IPv6 EH packet droppers and the aggressive fragmentation filters in today's network. It's also out of the question to contemplate changes to the IPv6 protocol and the format of the packet header.

All that's left for us if we want to make DNS work effectively and efficiently over IPv6 is to change the way the DNS behaves.

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