November 2020
Geoff Huston,
Joao Damas

DNS XL

We've written a number of times about the issues of managing packet sizes in packet-switched networks. It's an interesting space that is an essential part of the design of packet-switched networks, and a space where we still seem to be searching for a robust design.

This work has been prompted by the DNS Flag Day 2020 (https://dnsflagday.net/2020/), where a number of DNS resolver implementations were altered to set their default EDNS(0) UDP Buffer Size down from 4,096 octets to 1,232 octets. That's a big change, and it raises many questions. How bad was the problem when with the larger default value of this parameter? Does this new parameter setting improve the robustness and efficiency of DNS transactions? Is the setting of 1,232 too conservative? Are there larger values for this parameter that could offer better DNS performance?

This is a technical report on a detailed exploration of the way the Internet's Domain Name System (DNS) interacts with the network when the size of the application transactions exceeds the underlying packet size limitations of hosts and networks.

IP Packet Sizes

The choice of permitted packet sizes in a packet-switched network is often the result of a set of design trade-offs, where issues of carriage efficiency, capacity, speed, jitter, reliability and transmission noise tolerance all come to bear. We've seen fixed length 53-octet ATM cells and variably sized Ethernet, Token Ring and FDDI LAN designs. The Internet Protocol was intended to straddle all of these variants in underlying packet transmission media as its name implies, but even this protocol has some minimum and maximum packet size thresholds.

We've had a couple of rounds of defining packet sizes in IP network, namely IPv4 and IPv6. There are four threshold values here of interest: the smallest valid packet size, the maximum size of a packet that is to be passed across any underlying network without IP level fragmentation, the largest packet size that all IP hosts must be capable of processing and the largest packet size that can be represented in the IP protocol itself.

In IPv4 the minimum IP packet size is the IPv4 packet header without any options and no upper level protocol payload, which is 20 octets in length. The maximum size of a packet that is to be passed across any network without IP fragmentation is 68 octets (RFC 791). The largest packet that all hosts must be capable of processing is 576 octets. The maximum IP packet size in this protocol is 65,535 octets, as the IP header uses a 16-bit IP packet length field.

> The choice of 576 octets is unexpected. Unlike ATM's choice of 53 octets this value was not the result of an uncomfortable compromise within a standards committee, although it certainly looks like it could've been! It's motivated by a desire to support a universal application transaction model in IP that was able to use a 512-octet data payload between any two IP hosts as an assured transaction. In addition to the payload there was an allowance of 64 octets for the IP and transport level headers (which is larger than the minimum required and shorter than the largest possible headers with options included). The specification of the DNS that uses a maximum of 512 octets for DNS payloads in UDP is of course related to this underlying IP definition.

In IPv6 the minimum IP packet size is 40 octets. The maximum size of a packet that needs to be passed across any network without IP fragmentation is 1,280 octets (RFC 8200). The largest packet that all hosts must be capable of processing is 1,500 octets (RFC 8200). The maximum packet size is 65,575, based on the redefinition of the 16-bit packet length field to not include the 40 octet IPv6 packet header. There is also a Jumbo Payload option (RFC 2675) that uses a 32-bit packet length field, allowing for packet sizes with a 32-bit payload length (which excludes the IPv6 packet header). This jumbo gram extension header allows the IPv6 protocol to support packets up to 4,294,967,336 octets in length.

> Similarly, 1,280 is an unexpected choice. In the same way that IPv4 used 576 as the sum of 512 octets of payload and an allowance of 64 octets of payload headers, IPv6 derived 1,280 as the sum of 1,024 and 256, so that it could be reasonably assumed that all IPv6 hosts could process a 1,024-octet packet payload with allowance of 256 octets of payload headers. Alternatively, it could be explained by assuming an Internet-wide packet size of 1,500 octets and allowing up to 220 octets of numerous levels of IP-in-IP encapsulations simultaneously. Given that the protocol needs to define some value here, 1,280 is probably as good as any other, but 1,280 as a value reflects no limitation of any underlying media layer in the transmission system.

> The value of 1,500 octets deserves some further explanation as this has become the default maximum packet size of the Internet for more than two decades. This value has a different derivation, based on the original 10Mbps Ethernet LAN specification. It appears that the available chip sets for the original Ethernet implementation had a maximum of 1,518 octets in their per-packet buffer. While larger packet sizes of some 4,000 octets were subsequently used in Token Ring and FDDI LAN standards, the lowest common denominator of 1,500 became a defacto industry standard. It should also be remembered that Ethernet used self-clocking packets, so a 1,500-octet packet required the data clock to maintain stability at a 20Mhz oscillation rate for 1.2ms. A larger maximum packet size would call for higher levels of clock stability as well as larger data buffers on the chip, both of which would've increased the cost of Ethernet chips at the time. The larger packet size range would've also increased the network's jitter characteristics.

This is set of capabilities and constraints for IP is summarized in the following table:

	IPv4	IPv6	
Minimum IP packet size	20	40	
Maximum Unfragmented IP packet size	68	1,280	
Assured Host Packet Size	<= 576	<= 1,500	
Maximum Packet Size	65,535	65,575	4,294,967,336 (Jumbogram)

Table 1 – IP Packet Sizes

Today the public Internet largely supports a maximum unfragmented IP packet size of 1,500 octets. There are minor variations where some forms of encapsulation are used, but in what we might call the core of the network 1,500 octets is the general rule.

DNS and Packet Sizes

Where this topic of packet sizes matters is with the DNS.

The DNS operates in a very conservative way when it uses UDP. In order to avoid the issues with packet loss and fragmentation, the original DNS specification (RFC 1035) mandated that the maximum size of a DNS response was 512 octets. This was not an onerous imposition, as it was challenging to generate a larger response for any query other than a response to an 'ANY' query, and this matched the IPv4 host requirements to accept any packet up to 576 octets in length (RFC 791). Even if the server had much it could add to a response in additional sections in a response, such additional sections are optional. A client could explicitly ask for this data in any case if it needed to use the data. If the DNS response was larger than 512 octets when packaged as a DNS message, it was possible to truncate the DNS response at 512 octets or less, mark it as truncated, and send it back. The client should interpret a truncated response as a directive to switch to use TCP, where larger responses could be placed into a TCP stream. TCP's reliable stream control system and the interaction between TCP MSS settings, host MTU settings, Path MTU and ICMP messages was seen to provide TCP with sufficient capability to carry a DNS response across a diversity of hosts and paths. This DNS packet handling framework was considered to be adequately robust for many years.

The situation changed with the introduction of DNSSEC and the adoption of IPv6. To convey the reason why the DNS response is trustworthy we've added digital signatures to responses. These signatures can add a significant volume of data, and this impacts on the design decisions relating to DNS' use of UDP. Also adding IPv6 addresses into the DNS response can also cause larger responses. For example, the so-called "priming query" to query for the servers that serve the root zone of the DNS require an 811-octet response when all the servers' IPv4 and IPv6 addresses are loaded into the additional section of the response.

There was an emerging concern that the DNS was triggering the switch to use TCP too early. The Internet certainly/ appears to support IP packets of sizes up to 1,500 without major issues, so pushing the DNS to use TCP for responses larger than 512 octets seems like a case of self-inflicted unnecessary damage. TCP requires more server state and therefore increases the overheads of the DNS. Invoking TCP too early is a case of incurring additional cost without benefit. If 512 octets is too early, then what's a good threshold value to switch to TCP? What packet size can the Internet handle with some level of reasonable assurance that the packet can be delivered? Certainly, a setting of 1,500 octets is a good guess, but can it be even larger? Even if the IP packet has to be fragmented, is UDP still a relatively reliable way of passing DNS information through the Internet? If so, then what value might we use here as the threshold point to shift to TCP? We want it to be high enough to reflect some degree of reluctance to use TCP and increase the overall utility of UDP, but not so large as to push the DNS into the overhead of recovery from lost responses which involved timeouts and retransmission. What's the "right" threshold?

The first such suggestion can be found in the specification of EDNS(0) DNS extensions (RFC 6891):

```
6.2.5 Payload Size Selection

Due to transaction overhead, it is not recommended to advertise an
architectural limit as a maximum UDP payload size.  Even on system
stacks capable of reassembling 64 KB datagrams, memory usage at low
levels in the system will be a concern.  A good compromise may be the
use of an EDNS maximum payload size of 4096 octets as a starting
point.

A requestor MAY choose to implement a fallback to smaller advertised
sizes to work around firewall or other network limitations.  A
requestor SHOULD choose to use a fallback mechanism that begins with
a large size, such as 4096.  If that fails, a fallback around the
range of 1280-1410 bytes SHOULD be tried, as it has a reasonable
chance to fit within a single Ethernet frame.  Failing that, a
requestor MAY choose a 512-byte packet, which with large answers may
cause a TCP retry.
```

> It seems that nomenclature has drifted since this RFC was written, and while the RFC refers to "UDP payload size", common use today appears to use the terms "buffer size" or "UDP buffer size". I'll use this same common use convention here and refer to this parameter as a "UDP buffer size" (or just "buffer size" when I get tired of typing "UDP" all the time!).

The first guess at a working size for an upper bound on UDP buffer sizes in UDP responses was 4,096 octets, as specified in RFC 6891. A query with a buffer size value of any value greater than 1,472 (or 1,452 when accounting for an IPv6 packet header) is saying to the server is actually two things:

1. The client's IP stack can reassemble a received fragmented IP datagram of up to the UDP buffer size plus the UDP and IP headers in length.

2. The path between the server and the client will allow a fragmented UDP datagram.

It may be an over-simplification, but in fact any UDP buffer size value more than 1,452 is saying much the same thing: the resolver client (or "requestor using the terminology of RFC 6891) believes that it is acceptable for a server to send it fragmented UDP packets. This is of course a bit of a fabrication on the part of the resolver client, as the client has absolutely no idea of the path characteristics of the full incoming path from the server to the resolver client. Even if a large UDP packet generates an ICMP message relating to a Path MTU problem it's the server who will receive that diagnostic packet relating to the path problem, not the resolver client. Which means that even if the client has over-estimated the path characteristics, it will be none the wiser if it got it wrong!

There appears to be some uncertainties, at least for me, with the precise interpretation of the intended role of UDP buffer size signalling.

- Is this signal related to UDP fragmentation avoidance and therefore relates to the Path MTU from the server to the resolver client, then how can the resolver client know the Path MTU for this server? If this is a unilateral announcement by the resolver client of the effective MTU for all paths from all servers to this resolver client, as a UDP fragmentation avoidance device, then the resolver client is not in an informed position to replace path-based Path MTU discovery with a unilateral declaration.

- Is this signal related to the host's fragmented IP packet reassembly capability? Hosts are not necessarily required to reassemble packets greater than 576 octets in IPv4 and 1,500 octets in IPv6, and the UDP Buffer Size could be strictly interpreted to indicate to the server that it is capable of reassembling a larger IP packet. But does the local DNS application really interrogate the IP drivers of the host software stack to find this IP driver configuration setting? How does this occur? Is it the product of looking at the maximum number of fragments per packet times

the interface MTU for the interface over which the query is sent? The client may not know the actual UDP buffer size in advance, as it is actually the product of the Path MTU multiplied by the maximum number of fragments per packet that the host accepts. Without knowing the incoming path MTU the maximum UDP buffer size cannot be calculated by the host. A reading of RFC6891 suggests that this parameter is not automatically set from the host's IP fragmentation reassembly capability in any case.

- Does this signal relate to the application behaviour and describes the maximal size of a DNS response that can be processed by the application? If so, then this setting should apply equally to TCP responses as to UDP responses. But that's apparently not the case.

It appears to me that it's the second objective from this list, namely the host's fragmented IP packet reassembly process that was intended to be described in this setting. However, I suspect that it is being used in a slightly different manner, namely in a manner similar to the MSS setting in TCP.

What UDP buffer sizes do we see in Queries?

Figure 1 shows the distribution of declared buffer sizes seen in DNS queries taken from DNS data relating to queries seen at our authoritative servers for an ad-based measurement run across the month of September 2020. Over this period, we saw some 887M queries where just 760,000 (or 0.9%) did not use a UDP Buffer size setting. Some 83% of queries used a setting of 4,096, presumably due to the advice in RFC6891. A further 6.5% of queries used a size of 8,932 octets. It may be that this setting relates to a 9,000-octet jumbo gram with allowance for an IPv6 plus UDP packet header. Of the 887M queries some 6,612 queries, or 0.0007% of all queries used a UDP buffer size of 65,535 octets.

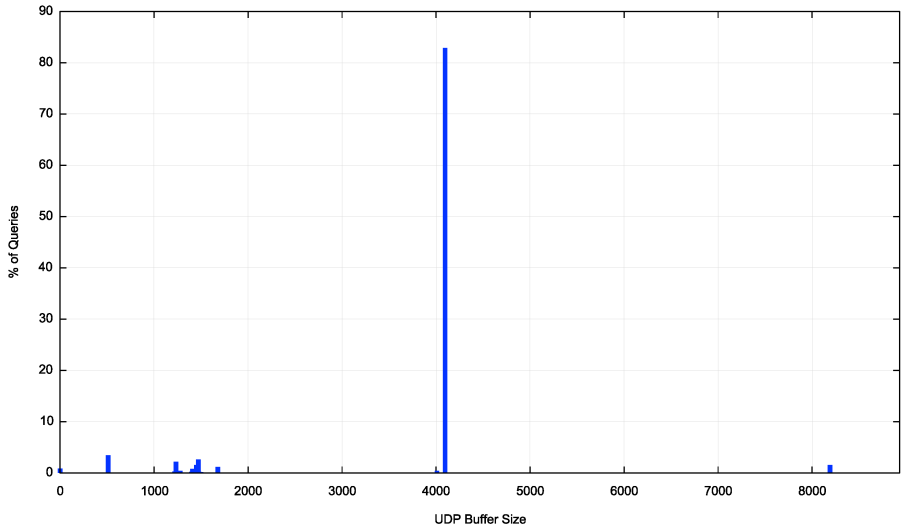

Figure 1 – UDP Buffer size distribution

At the other end of the range some 3.4% of queries used a setting of 512 octets. Given that this is the same as no UDP Buffer Size setting in the query, it is not clear why this value is used at all.

Figure 2 shows the detail of UDP Buffer sizes in the range between 1,200 and 1,500 octets. The use of the values 1,280 and 1,500 octets show that some DNS resolver admins have confused IP packet size with DNS payload size. In various ways the other values appear to indicate that the UDP Buffer size is being used not as a reassembly size limit, but as a surrogate path MTU signal. Perhaps this setting is being used as a surrogate signal to the server not to send fragmented UDP responses to this client.

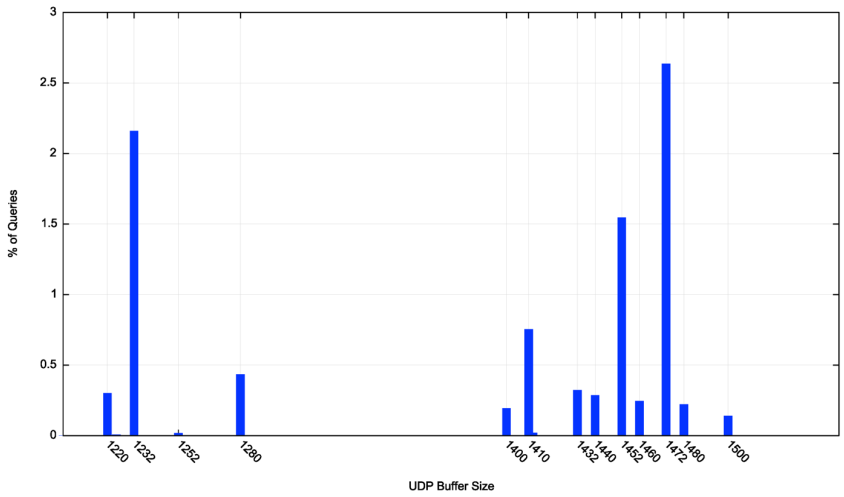

Figure 2 – UDP Buffer size distribution between 1,200 and 1,600 octets

DNS Flag Day 2020

The use of the UDP Buffer size appears to be changing in its role, and increasingly we are seeing it being used as a surrogate to the IPv4 DON'T FRAGMENT setting. IPv6 has no such surrogate, as the network itself cannot fragment IPv6 packets. Instead of being an IP-level signal, this buffer size query attribute is commonly used as an application directive, to direct the DNS server application not to generate UDP packets that are likely to be fragmented in its response.

This has been taken a step further of course, as we just can't resist taking most things one step further! We've now had DNS Flag Day 2020 (https://dnsflagday.net/2020/). This flag day was intended to commence a change the default operation of DNS clients (resolvers) to use the UDP Buffer Size setting in queries in a way that causes servers not to emit large UDP responses that may result in fragmented UDP responses.

The rationale for this initiative is that there appears to a high incidence of IP packet loss when the packet is fragmented. This is due to both firewall settings where fragments are often seen to pose a security threat, and IPv6 itself, where the IPv6 use of an Extended Header for IP fragmentation control can cause packet drop in some commonly used network switching equipment. Packet loss is a problem, in that it is also a loss in signalling. How long should a client wait before either resending, trying elsewhere or even giving up? It's a tough call, and the client software risks giving up too quickly and erroneously reporting resolution failure or being too persistent and causing the application to hang waiting for name resolution, and at the same time launching a large set of packets. In an isolated case the difference between two or twenty DNS queries over a few seconds is hardly noticeable, but if everyone does it, then we are talking about adding billions of additional queries and possibly turning the DNS into a gigantic denial of service weapon. If we want to be responsive, but not cause packet floods, then it's advisable to avoid signal loss.

One possible way to respond to this situation is by using the approach of an additional truncated response (https://www.potaroo.net/ispcol/2018-04/atr.html). This approach proposing sending an additional DNS response trailing a fragmented UDP response, which consists of just the query section with the truncate bit set. If the client does not receive the fragmented UDP response due to fragmented packet loss, the trailing response will be received, cause the client to immediately re-query using TCP. This concept did not gain any traction with DNS software developers and has not been taken up to any noticeable level.

Another response is that being advocated in the DNS Flag Day 2020. Here the UDP Buffer Size is re-interpreted as a DON'T FRAGMENT directive from the client to the server. In the absence of better information, the client should use a UDP Buffer Size setting of 1,232 octets in its UDP queries. This is intended to ensure that the server will not send UDP responses that are either fragmented at the server or fragmented in flight. If this is the intended outcome, then why not use a setting of 1,452 octets? Or if

you want to be a little more pedantic about this, why not use 1,452 octets if the query is sent using IPv6 and 1,472 octets if it's sent using IPv4? Or even just use a buffer size of 1,440 octets as a lower common unit. After all, if the default MTU of today's Internet is 1,500 octets, as appears to be the case, then shouldn't we allow for DNS responses where the IP packet is up to 1,500 octets in size before resorting to a second query over TCP?

It appears that the DNS Flag Day 2020 proponents took a highly conservative approach in this case. They appeared to be taking the position that fragmented UDP packet loss is more of an issue than the additional time and overheads to shift to TCP in a DNS re-query.

Were they correct in their assumptions? By forcing DNS transactions to be conducted over TCP for responses in the range of 1,232 octets to 1,440 octets, is this improving the behaviour of the DNS? Or is this actually imposing a potential penalty on DNS performance? Should we use a UDP Buffer Size setting of 1,440 octets rather than 1,232 octets and allow UDP to service responses in this size range as a first effort?

We can go a step further and ask a more general question about the current behaviours of the DNS when handling large UDP responses. That should provide some useful data to consider how to answer this and related questions about protocol behaviour in the DNS for large responses.

Measuring the DNS and Large Responses

To provide some further insights into the DNS and handling of large responses we set up an experiment to measure the behaviour of the DNS when handing large responses. The details of this online ad-based measurement technique have been described a number of times and I won't repeat that material here. However, there are a number of aspects to this measurement that should be noted, as they impact on the generality of the interpretation of the results of the measurement.

This is a measurement of the DNS resolution behaviour by those recursive resolvers that directly query authoritative servers. It is not a measurement of the stub-to-recursive query and response behaviour, nor is it a measure of the behaviour of the internal infrastructure of the DNS including load distribution and query engine farms.

This measurement technique uses "missing glue" as the mechanism to ascertain that the DNS response has been received by the resolver.

> The reason why we need to adopt this approach is based around the conditions of the advertisement system that we are using to perform this measurement. We need to use dynamic labels to bypass caching in the DNS, but the ad system only allows the ad recipient to resolve names using a single dynamic label as the terminal left-most label. To pose a condition in DNS resolution and then use the DNS itself to see if the DNS was able to resolve that condition would normally use multiple levels of dynamic labels in the delegation chain. As we are constrained in the names used in the ad script, we turned to the resolution of nameserver names top meet the constraints of the advertising system and achieve a relatively reliable measurement.

The approach is that the large DNS response is provided during the discovery process of finding the address of a name server for a domain. The original name resolution task can only proceed once the resolver has established the address of the name server. This resolution process is the same as all other name resolution exercises with just one difference: the result of this DNS resolution exercise does not need to be DNSSEC-validated, and so it is not strictly necessary to use the EDNS(0) DNSSEC-OK flag, or even to use EDNS(0) extensions at all. There is no need for DNSSEC signatures here. This

consideration implies that there may be a higher proportion of queries that use no UDP Buffer Size at all when resolving the name server names.

This is what we see in the measurement data. We observed that 27% of queries for nameserver resolution use no UDP buffer size setting and a further 6% of queries use a buffer size of 512 octets when resolving missing glue for nameservers. This 33% of queries that restrict the UDP response to 512 octets in name server resolution compared to the 11% of queries that use this same 512 octet restriction in conventional name resolution queries (Table 2).

Buffer Size	Resolution Queries	Cum	Name server Queries	Cum
none	2.3%	2%	27.5%	27%
512	8.8%	11%	6.0%	33%
1220	0.4%	11%	0.3%	34%
1232	3.2%	15%	3.0%	37%
1280	0.2%	15%	0.2%	37%
1400	0.5%	15%	0.3%	37%
1410	1.2%	17%	1.2%	39%
1432	1.3%	18%	2.1%	41%
1440	0.5%	18%	0.5%	41%
1452	1.2%	20%	1.3%	43%
1460	0.4%	20%	0.3%	43%
1472	0.8%	21%	0.9%	44%
1480	0.1%	21%	0.1%	44%
1500	0.4%	21%	0.2%	44%
1680	2.5%	24%	1.9%	46%
2048	0.1%	24%	0.1%	46%
3072	0.0%	24%	0.0%	46%
3500	0.1%	24%	0.0%	46%
4000	0.2%	24%	0.1%	46%
4010	0.2%	24%	0.2%	47%
4096	73.1%	97%	51.5%	98%
8192	2.6%	100%	1.8%	100%

Table 2 – Distribution of commonly used UDP Buffer Sizes per Query Task

The resolvers using the smaller UDP Buffer sizes appear to be the same resolvers that use a UDP Buffer size of 4,096 octets for conventional resolution queries (Figure 3). It should be noted that Google's DNS service is a major presence when looking at the behaviour of recursive resolvers from the user's perspective, with up to one quarter of all users using Google's service either directly or indirectly. This implies that Google's recursive resolver behaviour has a major impact on these numbers, and their DNS resolver performs nameserver resolution with no UDP buffer size specified in queries relating to resolving nameserver names.

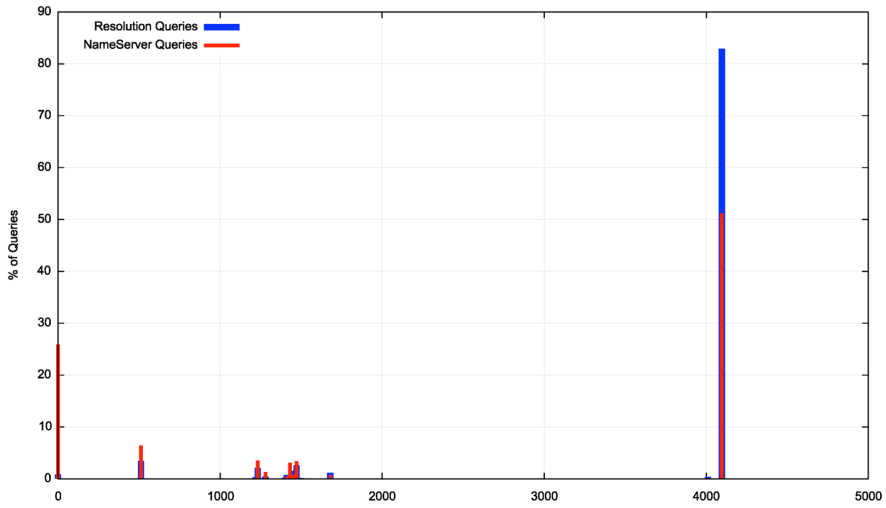

Figure 3 – Comparison of UDP Buffer Sizes for primary resolution queries and missing glue queries

At a finer level of detail there are a number of buffer sizes that are used for name server resolution but are not used as commonly for normal resolution (Figure 4).

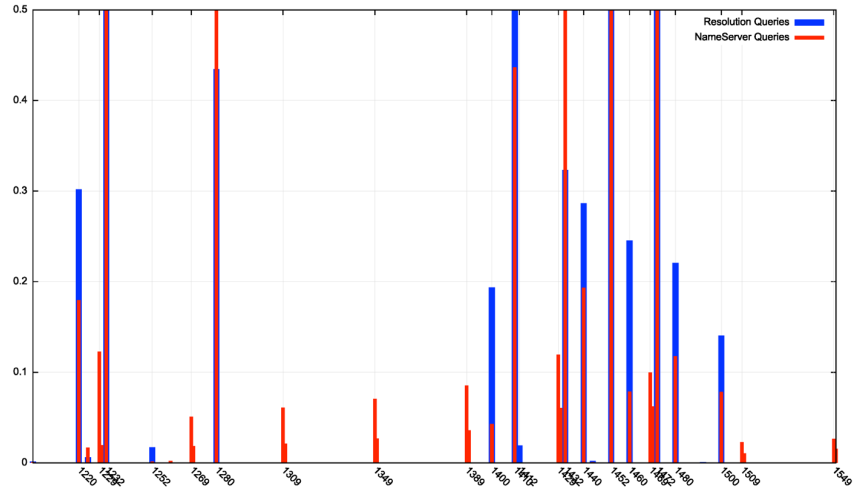

Figure 4 – Comparison of UDP Buffer Sizes for primary resolution queries and missing glue queries (detail)

It has been observed that some DNS resolvers are thrown by missing glue when the response is a referral to delegated nameservers and they abandon name resolution. To avoid this factor causing additional noise in these measurements we are used a second DNS query which generated an unpadded response to the name server query of some 175 octets as a control, and we look at only those experiments that have been successful in also resolving the control experiment.

We have generated the larger DNS responses using a padded DNS response. The padding contains an additional but irrelevant resource record. Where the response is to be truncated, we omit the answer section from the response, so the client cannot glean any information at all from the truncated response at all. This is intended to force the resolver client to use TCP in response to a received truncation signal.

Dual Stack Measurements

In this measurement we are operating the server in dual stack mode, and honouring the query's UDP buffer size setting, if provided, and if there is no UDP Buffer size we use a maximum DNS response size of 512 octets.

The failure rate for DNS responses that are up to 1,430 octets (corresponding to an IP packet size of 1,458 octets in size in IPv4 and 1478 octets in IPv6) are at a consistent level of some 0.5% of all such tests. The next larger DNS response is either 1,498 octets in IPv4, or a fragmented UDP response in IPv6. The failure rate for this size response is 0.9%. For larger DNS responses, where either UDP fragmentation or truncation is used in the initial response, the failure rate is a consistent 2.6% of all such tests (Table 3).

DNS Response Size	Tests	Pass	Fail	Failure Rate
1,230	4,303,625	4,282,457	21,168	**0.49%**
1,270	4,308,430	4,287,046	21,384	**0.50%**
1,310	4,307,172	4,286,064	21,108	**0.49%**
1,350	4,303,963	4,282,752	21,211	**0.49%**
1,390	4,309,937	4,288,413	21,524	**0.50%**
1,430	4,303,654	4,281,858	21,796	**0.51%**
1,470	4,308,472	4,269,785	38,687	**0.90%**
1,510	4,303,571	4,197,910	105,661	**2.46%**
1,550	4,306,455	4,194,465	111,990	**2.60%**
1,590	4,300,218	4,187,575	112,643	**2.62%**
1,630	4,305,191	4,191,994	113,197	**2.63%**

Table 3 – Dual Stack DNS Response

Note that in all test cases we also used an unpadded small response of 175 octets as a control, and the individual test results were only used in those cases where the resolver was also able to resolve the control case. This implies that the failure rates for larger responses are likely to be associated with truncated responses and the requirement to support TCP rather than the requirement to resolve a glueless delegation.

This 2.6% failure rate for larger DNS responses is a significant failure rate, as it precludes the use of large DNS responses as a universal measure without first changing either the behaviour of DNS resolvers, DNS servers, or both.

An analysis of the various cases of failure are shown in Table 4.

FAILURE ANALYSIS		A		B		C		D		E	
DNS Response Size	Count	UDP only		Fragmented UDP		Truncated UDP, NO TCP		Truncated UDP, Failed TCP		TCP	
1,230	21,168	15,846	74.9%	-	0.0%	605	2.9%	1,507	7.1%	3,206	15.1%
1,270	21,384	15,839	74.1%	-	0.0%	675	3.2%	1,532	7.2%	3,331	15.6%
1,310	21,108	15,482	73.3%	-	0.0%	735	3.5%	1,486	7.0%	3,399	16.1%
1,350	21,211	15,663	73.8%	-	0.0%	719	3.4%	1,541	7.3%	3,279	15.5%
1,390	21,524	15,770	73.3%	-	0.0%	871	4.0%	1,560	7.2%	3,322	15.4%
1,430	21,796	15,931	73.1%	-	0.0%	916	4.2%	1,516	7.0%	3,426	15.7%
1,470	38,687	16,906	43.7%	1,649	4.3%	1,859	4.8%	11,776	30.4%	6,495	16.8%
1,510	105,661	-	0.0%	76,136	72.1%	6,722	6.4%	13,975	13.2%	8,821	8.3%
1,550	111,990	-	0.0%	82,145	73.4%	6,948	6.2%	14,017	12.5%	8,872	7.9%
1,590	112,643	-	0.0%	82,334	73.1%	7,451	6.6%	13,910	12.3%	8,937	7.9%
1,630	113,197	-	0.0%	82,761	73.1%	7,802	6.9%	13,906	12.3%	8,721	7.7%

Table 4 – Dual Stack Failure Cases

Cases A and B are where there are only UDP responses. In both cases the UDP Buffer Size specified in the queries was greater than the response size, so the DNS responses were not truncated. This is the most common failure condition, accounting for three quarters of the failure conditions for small and large UDP response sizes. In the case of the larger responses it appears that the large UDP Buffer Size provided by the resolver is optimistic and the resolver does not receive a fragmented UDP response (Case B). It is not clear why the large but unfragmented UDP responses are being lost (Case A). These resolvers received the smaller unfragmented control response and were using a UDP Buffer Size value greater than the response size, as the response was not truncated. It is noted that there is no difference in the DNS response size in the failure rate of unfragmented UDP. The failure rate of DNS payloads of 1,230 octets is the same as the failure rate of DNS payload of 1,430 octets, and of all DNS payloads in between these two sizes. The higher failure rate of the 1,470-octet response is likely to be due to the fact that in IPv6 this packet will be fragmented and then UDP fragmentation loss issues come into play.

There are three more failure cases related to TCP.

Case C describes the case where the UDP Buffer size is either not specified (limiting the DNS UDP response to 512 octets, thereby forcing truncation) or is smaller than the response size. A truncated response is generated by the server, but there is no subsequent TCP session for this DNS name that gets to the point of passing the server a TCP segment that contains a DNS query. What is interesting here is that this failure rate rises as the DNS payload size increases between 1,230 octets through to 1,630 octets. What appears to be the happening here is the increasing number of UDP responses that trigger truncation as the number of queries that have a buffer size less than the size of the DNS response rises as the DNS payload size increases (see Table 2 for the incidence of these intermediate UDP buffer sizes in queries).

Case D describes a TCP condition where the response is passed back to the querier using TCP, but the server never sees a TCP ACK for the data segments that contain the response. The TCP session is left hanging. The larger responses require multiple TCP segments, while the smaller responses can be loaded into a single TCP segment. To get to the point of passing the response over TCP the TCP three-way handshake has been completed, so we can assume that the far end is reachable from the server for smaller TCP packets. The elevated loss rate for larger responses in Case D could possibly be explained by some form of path MTU mismatch that has resulted in a silent discard of the full-sized TCP segment that contains the response (or a "TCP Black Hole" condition). The client appears to be offering a TCP MSS size that generates IP packets in the TCP session that exceed the path capabilities. In IPv4 the path router that is encountering this MTU problem will fragment the packet, but the fragments may be filtered by a security firewall closer to the intended destination. In IPv6 the resolution is more complex, involving an ICMPv6 message being passed back to the server. The server needs to inform the TCP context of the problem and the sender should adjust the session MSS setting downward and resend the segment using a smaller TCP framing. The elevated failure rates point to some issues in this process.

Case E describes the condition where there is a TCP ACK response for the data, yet the resolver does not revert back to the primary query chain and make the closing query to complete the resolution process. Case E implies that there has not been a Path MTU problem, but the resolver at the far end of the TCP connection appears to have dropped the name resolution task completely. Most resolvers use a master query resolution timer and when this time is exceeded the name resolution context is abandoned.

It should be noted that there is nothing unusual in the server behaviour in this measurement exercise. The experiment's server honours the UDP buffer size parameter if specified or truncates the response if no UDP buffer size was specified and the DNS response is larger than 512 octets. The server is accessible over TCP and responds in all cases. And in all test cases used to compile these result tables the control experiment, namely of a small DNS response to resolve the nameserver name, was successfully completed. The server is configured as a dual stack server and uses a 1,500 octet MTU on its network interface, which while it may not be the most robust setting for a DNS server is certainly a conventional setting in the DNS environment.

Resolution Performance – Query Count

The way in which this glueless domain name structure was contrived we would expect to see at our servers up to 6 queries.

As already noted, and shown in Figure 5, the experiment's DNS structure uses three zones and with the use of dynamically generated DNS labels we can ensure that these three zone labels are unique for every measurement. A minimal path through for a resolver is to ask the "parent" zone server for the domain name, and because the response is a delegation response without any associated glue records the resolver must pause its primary resolution task and resolve the name of the name server. This resolution process will generate a second visible query to the "sibling" zone server for the nameserver name. The response will provide the resolver with an IP address which will then allow the

resolver to complete the task with a query to the "child" zone server. If the resolver supports only IPv4 then a minimal query set is three queries. A dual stack resolver may query for both the A and AAAA records for both the terminal zone and the so-called "sibling" zone, so a dual stack query set may use A and AAAA queries in all three cases, making 6 visible queries.

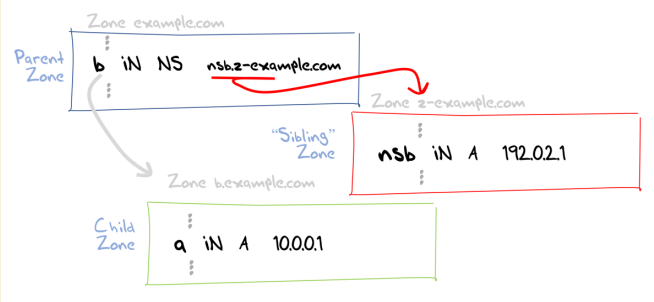

Figure 5 – "Glueless" delegation in the DNS

However, such a low query count assumes that the resolver and the server have complete knowledge about each other's state. UDP is not a protocol that supports tight state synchronisation between the two end parties, so a DNS resolver will generally re-query in a relatively aggressive manner when it has not received a response in the expected time, as it has no other way of determining the fate of its earlier query.

There is a considerable difference between success and failure here, as is shown in the average number of queries to either complete the resolution or fail (the failure point is 60 seconds from the first query) (Table 5).

DNS Response	Pass Queries	Fail Queries
1,230	4.8	12.3
1,270	4.9	12.0
1,310	4.9	12.0
1,350	4.9	12.5
1,390	4.9	12.7
1,430	4.9	12.6
1,470	5.2	27.5
1,510	5.8	46.1
1,550	5.8	43.7
1,590	5.8	43.6
1,630	5.9	44.2

Table 5 – Average Query Count vs DNS Response Size

A cumulative distribution of the successful resolution distribution is shown in Figure 6. All DNS sizes below 1,470 octets behave in a very similar manner. In some 70% of cases the name is resolved within 4 queries, and in 90% of cases the name is resolved within 7 queries. The case of 1,470 octet DNS payload requires fragmentation in IPv6 but not in IPv4 which adds to the query count. For DNS payloads that always required UDP fragmentation or truncation, namely 1,510 octets of DNS payload or higher, there is no visible difference between the various tested payload sizes. The larger packets add approximately 2 further queries to the the query count.

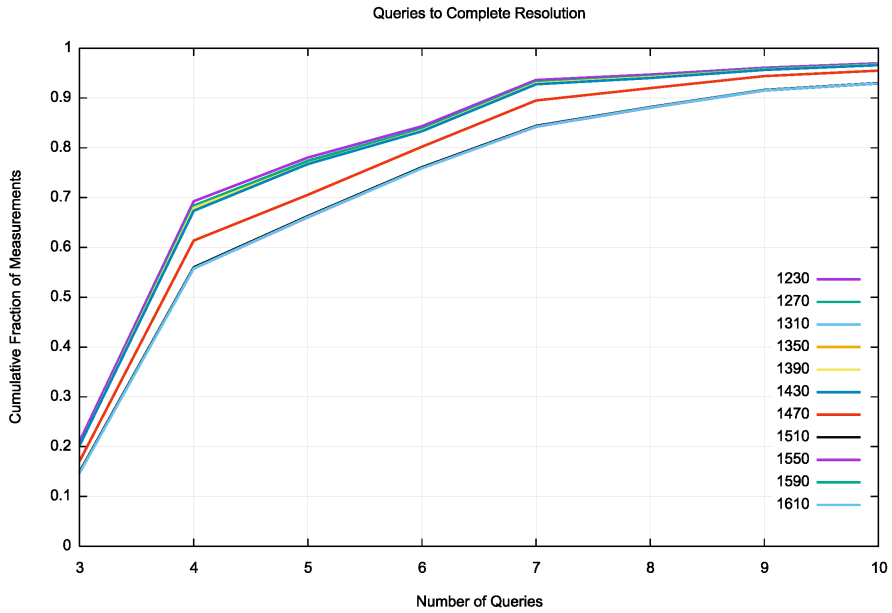

Figure 6 – Cumulative Distribution of Queries to Resolve the Target DNS Name

This query count is perhaps the one aspect of the measurement where there is a small but discernible difference between the various DNS payload sizes. Figure 7 shows a magnified detail of Figure 6, showing the cumulative proportion of experiments that complete in 4 queries or less. The DNS payload size of 1,230 octets shows that 69.3% of cases complete in 3 or 4 queries, while a 1,270-octet payload shows a slightly lower proportion of 68.4% of cases. The next three sizes of 1,310, 1,350 and 1,390 octets show a similar completion proportion of 67.9% of cases for each size. The next size of 1,530 shows a lower proportion of 67.4% of cases.

The difference in the query count is likely to be due to the distribution of UDP buffer sizes for name server name resolution (Figure 4). The relatively high proposition of queries with a UDP buffer size of 1,280 octets is the most probable factor, causing a higher rate of response truncation and re-try over TCP for DNS responses of more than 1,280 octets in size. there is a smaller peak in the use of a UDP buffer size of 1,269 octets, explaining why there is an observed difference here between responses of 1,230 octets and 1,270 octets in size.

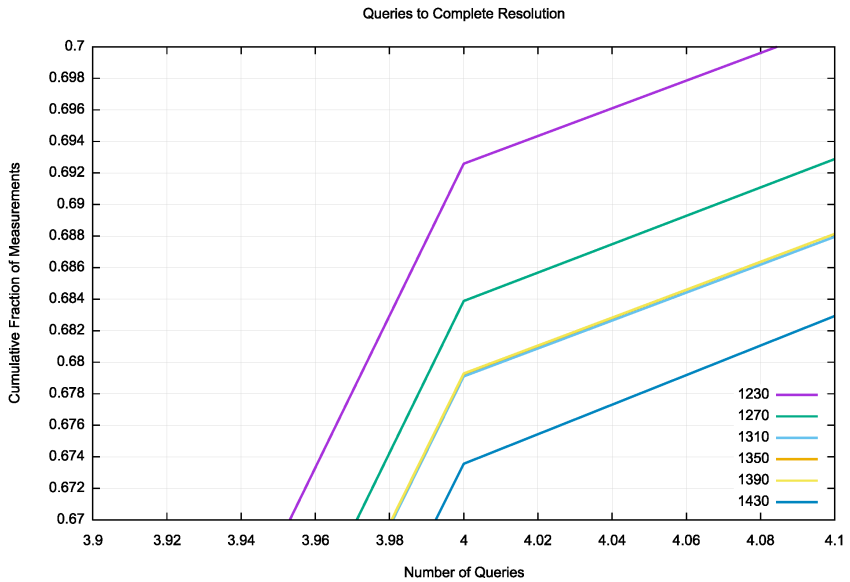

Figure 7 – Cumulative Distribution of Queries (magnified)

The DNS is extremely persistent when failure occurs. For smaller payloads 90% of cases stop the name resolution process within 20 queries. However, for the packet sizes larger than 1,500 octets 90% of cases perform up to 120 queries before either abandoning the query or after 60 seconds has elapsed since the initial query (Figure 8). The long tail of queries not shown in Figure 8 is extremely long, and we observed extreme cases of 3,843 queries, 4,347 queries and 6,467 queries within this 60 second interval. It appears that there are still a number of pathological situations in some parts of the DNS where query storms can occur.

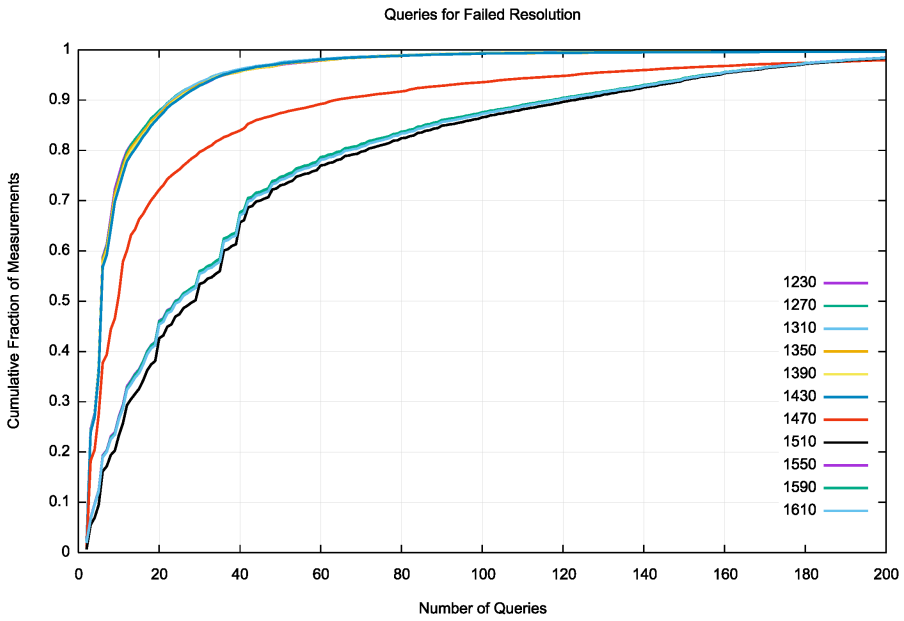

Figure 8 – Cumulative Distribution of Queries to Fail to Resolve the Target DNS Name

Resolution Performance – Resolution Time

When we look at the elapsed time to resolve this name, or abandon the resolution, we see a distribution pattern similar to the query time distribution. It is notable that there is a dramatic time difference between smaller DNS responses that are resolved using UDP and the time to set up a TCP session and requery (Table 6)

DNS Response	Pass Time (ms)	Fail time (ms)
1,230	66	5,801
1,270	68	5,829
1,310	68	5,970
1,350	71	6,107
1,390	72	6,163
1,430	73	6,190
1,470	247	12,385
1,510	1,221	20,842
1,550	1,230	19,708
1,590	1,232	19,605
1,630	1,250	19,799

Table 6 – Average DNS Resolution time vs DNS Response Size

Where the resolution is successful the elapsed time is very short. For small responses (smaller than 1,470 octets of DNS payload) 80% of experiments complete in under 200ms. For larger responses where the payload is greater than 1,500 octets this figure drops to 70% of clients completing the resolution in 200ms or less (Figure 9).

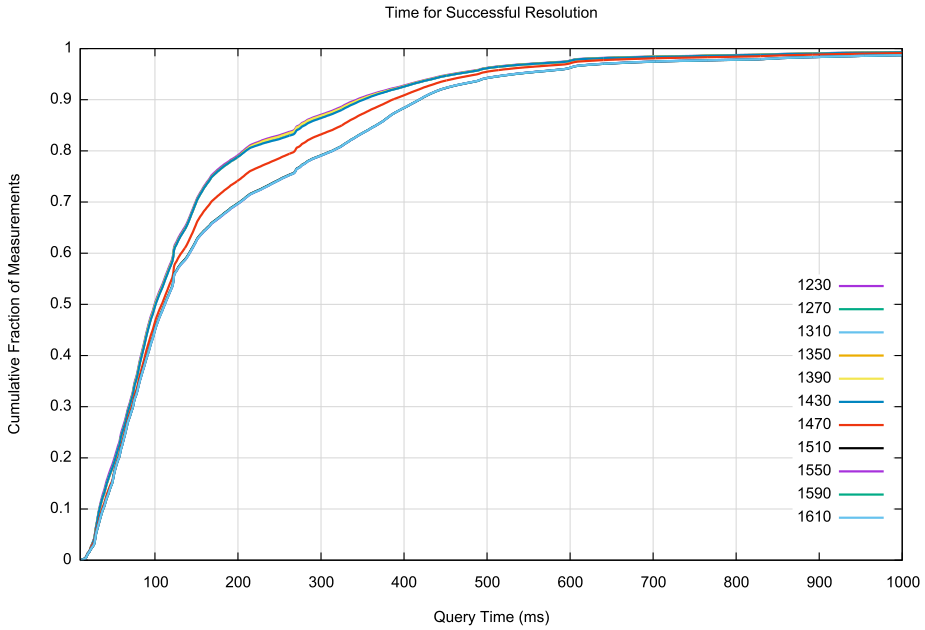

Figure 9 – Cumulative Distribution of Elapsed Time to Resolve the Target DNS Name

The situation changes somewhat when the resolution is not successful and the effort is abandoned. There is now a far more obvious difference between smaller and larger DNS payloads, and while 80% of the resolution experiments stop within 10 seconds for smaller DNS payloads, the larger payloads take longer. 80% of experiments with larger payloads stop within 30 seconds (Figure 10). This figure also shows that 5% of cases are still seeing queries at the time when the 60 second timer has expired.

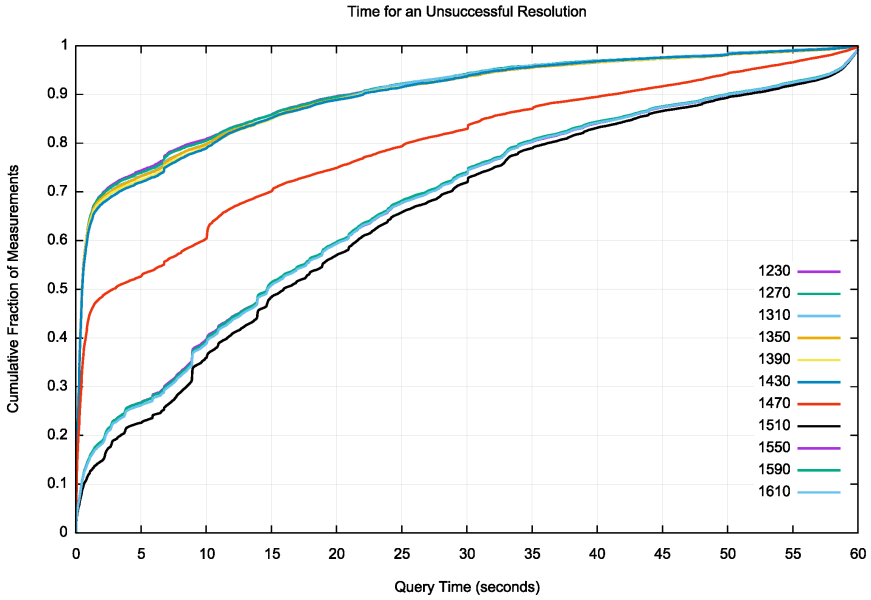

Figure 10 – Cumulative Distribution of Queries to Fail to Resolve the Target DNS Name

Resolution Performance – TCP

In this experiment there is a high level of reliance on TCP to complete the DNS resolution task. Some 43% of queries are received from resolvers that use a buffer size of less than 1,452 octets, which means that the server will truncate larger responses and not even attempt to respond with UDP despite the fact that the UDP packet is still smaller than the onset of packet fragmentation. In some 13% of cases of this behaviour is attributable to a resolver change when resolving name server names, where some resolvers use a smaller UDP buffer size for name server queries, but the remaining 20% of cases are also seen in conventional name resolution queries (Table 2). The overall reliance on TCP is shown in Table 7. What

is evident in Table 7 is that while there is widespread use of smaller buffer sizes in queries, the resolvers are at times unable to successfully manage the consequent TCP session when the UDP response is truncated. Where the buffer size is less than 1,452 octets, we see that one quarter of the failure cases are related to a failure where TCP is being used.

DNS Size	Total TCP use Ratio	TCP use when Pass	TCP use when Fail
1230	9%	9%	22%
1270	12%	12%	23%
1310	13%	13%	23%
1350	13%	13%	23%
1390	13%	13%	23%
1430	14%	14%	23%
1470	31%	30%	41%
1510	36%	36%	21%
1550	36%	37%	20%
1590	36%	37%	19%
1630	36%	37%	19%

Table 7 – TCP use Profile

The high TCP failure rate in Case D of Table 3 is also of interest. Some 12% of the failure cases are those where we see no subsequent ACK to the outgoing TCP segment that contained the response. Given that the first segment of the TCP response will be a full-sized segment, it will be sent using the minimum of the remote resolver's TCP MSS value and 1440 or 1460 octets, depending on whether IPv6 or IPv4 is used as the IP protocol (i.e. a 1,500 octet packet, corresponding to the server's MTU setting).

If there is any form of packet encapsulation on the path between the server and the resolver then this packet setting will be too large, and either fragmentation of the TCP packet will need to occur or the TCP MSS setting for this session will need to be revised. In IPv4 fragmentation can be performed on the fly by the router that is attempting the encapsulation, but this then assumes that the receiver will receive and reassemble the IP packet fragments, which according to the comparable results for fragmented UDP is not a good assumption. In IPv6 this path MTU issue is further complicated by the need to receive and process an ICMPv6 Packet Too Big message at the server. The best response to these kinds of issues is to adopt a conservative position and for resolver clients to use a TCP MSS value that is lower than maximum allowed by the interface MTU. Given that the DNS is (so far) not an intensive user of TCP, the additional inefficiencies of a smaller TCP MSS value would be outweighed by the increased likelihood of TCP segment delivery.

What MSS value to DNS resolvers use? This is shown in Figure 11. Some 80% of TCP sessions over IPv4 and 57% of TCP sessions over IPv6 use an MSS session that assumes a 1,500-octet path MTU.

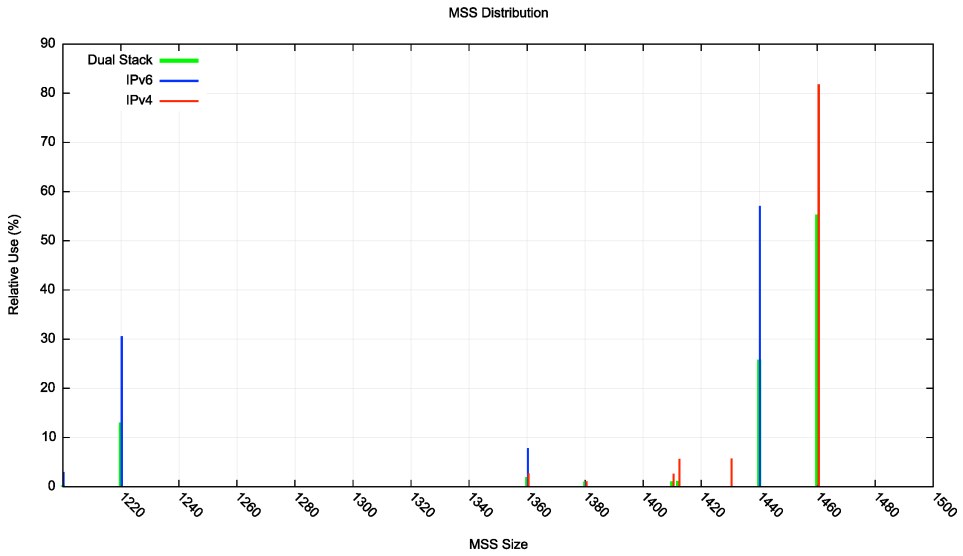

Figure 11 – Distribution of Resolver TCP MSS values

IPv6-only Measurements

UDP fragmentation behaviour differs between IPv6 and IPv4. In a second run of this measurement experiment the name server resolution was limited to IPv6-only. When IPv6 is tested with fragmented UDP we see significant loss rates, so what can we expect in the context of a managed DNS framework where there is the potential to specify a maximum UDP size in responses and shift to TCP? The results are shown in Table 8.

DNS Response Size	Tests	Pass	Fail	Failure Rate
1,230	5,933,991	5,929,015	4,976	**0.08%**
1,270	5,933,841	5,928,634	5,207	**0.09%**
1,310	5,938,874	5,933,491	5,383	**0.09%**
1,350	5,937,875	5,932,333	5,542	**0.09%**
1,390	5,938,285	5,932,563	5,722	**0.10%**
1,430	5,935,928	5,929,715	6,213	**0.10%**
1,470	5,941,800	5,816,543	125,257	**2.11%**
1,510	5,936,136	5,810,425	125,711	**2.12%**
1,550	5,938,521	5,812,568	125,953	**2.12%**
1,590	5,939,611	5,812,356	127,255	**2.14%**
1,630	5,941,612	5,813,309	128,303	**2.16%**

Table 8 – IPv6 DNS Response

This is a surprising result, in that an IPv6-only DNS resolution framework is better better (i.e. has a lower failure rate) than the comparable dual stack case, for all the DNS response sizes being tested. Obviously, the case of the 1,470-octet payload now behaves in a similar manner as larger payloads, as the addition of the 40-octet IPv6 packet header takes this response to 1,518 octets.

FAILURE ANALYSIS		A		B		C		D		E	
DNS Response Size	Count	UDP only		Fragmented UDP		Truncated UDP, NO TCP		Truncated UDP, Failed TCP		TCP	
1,230	4,976	2,228	44.8%	-	0.0%	1,587	31.9%	760	15.3%	401	8.1%
1,270	5,207	1,988	38.2%	1	0.0%	1,961	37.7%	796	15.3%	459	8.8%
1,310	5,383	1,967	36.5%	-	0.0%	2,192	40.7%	736	13.7%	474	8.8%
1,350	5,542	2,045	36.9%	1	0.0%	2,250	40.6%	764	13.8%	472	8.5%
1,390	5,722	1,972	34.5%	-	0.0%	2,533	44.3%	761	13.3%	444	7.8%
1,430	6,213	2,072	33.3%	1	0.0%	2,719	43.8%	824	13.3%	581	9.4%
1,470	125,257	616	0.5%	114,270	91.2%	7,847	6.3%	1,180	0.9%	1,324	1.1%
1,510	125,711	585	0.5%	114,060	90.7%	8,542	6.8%	1,139	0.9%	1,347	1.1%
1,550	125,953	592	0.5%	114,096	90.6%	8,738	6.9%	1,148	0.9%	1,352	1.1%
1,590	127,255	597	0.5%	114,600	90.1%	9,514	7.5%	1,158	0.9%	1,360	1.1%
1,630	128,303	646	0.5%	114,547	89.3%	10,493	8.2%	1,189	0.9%	1,399	1.1%

Table 9 – IPv6 Failure Cases

There are some differences in failure cases between Dual Stack and IPv6-only. Where the UDP Buffer size allows the server to send fragmented UDP packets, these fragmented UDP packets form the majority of the loss case (Case B). If the TCP session is completed, then the consequent failure rate is far lower (Case E). It may be the case that the extended time taken in a dual stack environment may trigger a resolver's query timer, and even through the TCP session completes it has taken too long to get to this point and the resolution is abandoned.

Before we get too far in conjectures here it is useful to compare this IPv6-only result with a comparable IPv4-only experiment.

IPv4-only Measurements

The third measurement pass used IPv4 only. For DNS responses less than 1,500 octets in size the IPv4 failure rate is significantly higher than IPv6 (slightly over three times the relative failure rate), which is an unexpected result. This does not appear to be related to the carriage of unfragmented UDP packets, so it must be related to those queries with a small buffer size or no buffer size at all. For responses larger than 1,500 octets IPv4 appears to operate more reliably than IPv6, and oddly enough, more reliably than the dual stack configuration (Table 10).

DNS Response Size	Tests	Pass	Fail	Failure Rate
1,230	1,165,962	1,162,612	3,350	0.29%
1,270	1,167,025	1,163,625	3,400	0.29%
1,310	1,169,710	1,166,336	3,374	0.29%
1,350	1,166,812	1,163,507	3,305	0.28%
1,390	1,168,637	1,165,209	3,428	0.29%
1,430	1,167,532	1,164,058	3,474	0.30%
1,470	1,166,895	1,163,069	3,826	0.33%
1,510	1,167,065	1,142,948	24,117	2.07%
1,550	1,167,403	1,143,182	24,221	2.07%
1,590	1,167,238	1,143,064	24,174	2.07%
1,630	1,167,685	1,143,158	24,527	2.10%

Table 10 – IPv4 DNS Response

The analysis of failure cases, shown in Table 11, shows some differences in failure diagnostics between IPv4 and IPv6. Case B shows that fragmented UDP packets fare better in IPv4 and IPv6, which corresponds with results from other studies of fragmented UDP. It also appears to be the case that there is a higher failure rate in not re-querying in TCP following the reception of a truncated response. Where the TCP transaction appear to be successfully completed there is a higher failure level in IPv4 as compared to IPv6.

FAILURE ANALYSIS		A		B		C		D		E	
DNS Response Size	Count	UDP only		Fragmented UDP		Truncated UDP, NO TCP		Truncated UDP, Failed TCP		TCP	
1,230	3,350	2,506	74.8%	-	0.0%	209	6.2%	281	8.4%	354	10.6%
1,270	3,400	2,553	75.1%	-	0.0%	210	6.2%	256	7.5%	381	11.2%
1,310	3,374	2,508	74.3%	-	0.0%	225	6.7%	256	7.6%	379	11.2%
1,350	3,305	2,476	74.9%	-	0.0%	227	6.9%	255	7.7%	342	10.3%
1,390	3,428	2,529	73.8%	-	0.0%	231	6.7%	269	7.8%	392	11.4%
1,430	3,474	2,492	71.7%	-	0.0%	270	7.8%	275	7.9%	427	12.3%
1,470	3,826	2,428	63.5%	262	6.8%	304	7.9%	299	7.8%	528	13.8%
1,510	24,117	700	2.9%	18,828	78.1%	2,568	10.6%	363	1.5%	1,640	6.8%
1,550	24,221	603	2.5%	19,055	78.7%	2,625	10.8%	373	1.5%	1,556	6.4%
1,590	24,174	652	2.7%	19,018	78.7%	2,490	10.3%	405	1.7%	1,595	6.6%
1,630	24,527	642	2.6%	19,185	78.2%	2,591	10.6%	394	1.6%	1,707	7.0%

Table 11 – IPv4 Failure Cases

Observations

There are a number of interesting observations from this data.

Poor Dual Stack Resolution Behaviour

Firstly, there the observation that dual stack DNS environments are marginally less reliable than either dedicated IPv4 or IPv6 environments (Table 9). This is a surprising outcome, in that a conventional expectation is that the dual stack environment should be able to use the "better" of the two protocols. A good case in point is the failure rates for the DNS response of 1,470 octets. In IPv4 the IP packet will be 1,498 octets in size, and the failure rate should be comparable to other small sized packets, which is the case. For IPv6 the higher failure rate for this response is comparable to all larger packets.

What is surprising is that DNS resolution failure in IPv6 does not immediately get repaired by repeating the query using IPv4. There is no "Happy Eyeballs" support in the DNS and failure in one protocol does not cause the other protocol to be invoked. This is a shortcoming of most DNS resolver implementations. It appears that many, or maybe all, resolver implementations assemble a list of name server IP addresses and query them serially, irrespective of which IP protocol family the addresses are associated with. A failure with an IPv4 address, for example, does not appear to cause a re-query immediately to the IPv6 of the same server, or, preferably the reverse. This is a shortcoming of most DNS resolver implementations. It is also unclear why in all cases except the 1,470-octet payload case the Dual Stack environment has a higher failure rather than exclusively using either IP protocol.

DNS Response Size	Dual Stack	IPv4	IPv6
1230	0.49%	0.29%	0.08%
1270	0.50%	0.30%	0.09%
1310	0.49%	0.30%	0.09%
1350	0.49%	0.29%	0.09%
1390	0.50%	0.30%	0.10%
1430	0.51%	0.30%	0.10%
1470	0.90%	0.34%	2.11%
1510	2.46%	2.13%	2.12%
1550	2.60%	2.14%	2.12%
1590	2.62%	2.13%	2.14%
1630	2.63%	2.16%	2.16%

Table 12 – DNS Failure Rates by Protocol

Uncertainty over the Role of UDP Buffer Size

Secondly, there is little evidence from the figures to suggest that there are systematic issues with the carriage of IP packets of sizes between 1,280 and 1,500 octets in the network itself when we look at the response paths leading to the recursive resolvers. There is no clear evidence that there are Path MTU issues at play here in the infrastructure parts of the network.

What appears to be the major issue here is the use of smaller UDP buffer sizes below the IPv4 level of 1,472 and the IPv6 level of 1,452 octets. The smaller buffer size causes DNS response truncation and requires a re-query using TCP.

There is some uncertainty of the precise operational role of the UDP Buffer Size field. It's commonly interpreted role is as a proxy for a Path MTU value to guide the DNS to avoid packet fragmentation. This interpretation appears to be the motivation between the DNS Flag Day 2020.

The problem with this form of use of the buffer size is that the receiver (the querier who set the value in the DNS query) has no clear idea of the incoming Path MTU value, and no way of revising its value even

in the event of wayward packet fragmentation! Any ICMP messages will head in the opposite direction, toward the sender of the packet, so the client is working in the dark here. If this value is set too low, then it adds the burden of doubling up on queries and the additional overhead of TCP session establishment for resolution. If it's too high then UDP packet fragmentation may occur which, in turn, may exercise firewall filters, which commonly interpret UDP fragments as inherently unsafe and discard them. While a low buffer size value does not cause a loss of signal, a too high a value that triggers fragmentation packet drop does cause signal loss, and this then requires the client to rely on packet timeouts to amend its query strategy to work around the issue.

Context of these Measurements

And finally, a small note of caution about the interpretation of these results. This is a measurement between recursive resolvers and authoritative nameservers. In fact, it's a measurement between recursive resolvers and just three authoritative name servers. It's not a measurement of the edge environment from the stub resolver to the recursive resolver. There are packet issues at the edge of the network that are not encountered in the "middle". It's also the case that larger DNS responses are often associated with DNSSEC signatures. DNSSEC validation at the stub resolver is still a rarity these days, so the entire issue of why we should have reliable mechanisms to carry large DNS responses all the way to the stub resolver is not clear to me.

Also, as already noted, this measurement is based on an exercise in resolving the name of a name server, and some resolvers use different settings when performing this task.

What If…

So far, these measurements are based on respecting the packet size conventions related to DNS. These are that no UDP DNS response should exceed 512 octets unless there is an EDNS(0) extension with a UDP buffer size in the query, and the value of this field is greater than 512. When there is a UDP buffer size in the query the response should be no larger than this size. In such cases where this is not possible the server will respond with a truncated packet. In this measurement the truncated packet has an empty answer section, so the resolver making the query cannot use the truncated response to form an answer, and the truncated response should trigger the resolver to repeat the query over a TCP session with the server.

What if we break these conventions?

In particular we are interested in understanding the likely changes to DNS resolution behaviour of fragmented UDP responses, the behaviour of TCP responses and the behaviour of the DNS as a whole if all recursive resolvers were use the DNS Flag Day 2020 setting of 1,232 as a buffer size in their queries. In the second part of this report we will look at the behaviour of the DNS when we rewrite the queries as if they all had an EDNS(0) extension and there was a buffer size in this extension that was set to a particular value. Yes, this server-based rewriting of queries is cheating, and it's not what resolvers may be expecting, but it allows us to gain some further insights into the capabilities of the resolver to authoritative part of the DNS.

More to come in the second part of this report.

Disclaimer

The above views do not necessarily represent the views or positions of the Asia Pacific Network Information Centre.

Author

Geoff Huston is the Chief Scientist at APNIC, the Regional Internet Registry serving the Asia Pacific region.

www.potaroo.net