December 2020
Geoff Huston,
Joao Damas

DNS Flag Day 2020

The architecture of the Internet took a highly radical step in the evolution of wide area communications protocols. Rather than placing much of the functionality into the network infrastructure and using network functions to emulate reliable edge-to-edge circuitry, the Internet Protocol used a network service model that was minimal and provide nothing beyond unreliable packet delivery. Everything else was left to the connected hosts.

The true workhorse of the Internet has always been the TCP protocol. This protocol takes the underlying unreliable datagram service and constructs reliable application-to-application circuitry. Almost all of the widely used Internet applications are constructed using TCP or variations of this TCP model. The Internet Protocol suite also included an abstraction of the IP datagram model, which added application-level port addresses to the IP datagram and little else. This datagram-based transport service, UDP, was not so widely used, and even less so once the potential for widespread and intense abuse using spoofed address fields was noted. These days it seems that the only mainstream applications still using UDP transport in the public network are the network Time Protocol (NTP) and the Domain Name System (DNS).

The DNS appears to be an ideal match to UDP. The query/response transactional model of the DNS where both the queries and the responses were expected to be small seemed like a good fit to a network model of a simple packet exchange. In the cases where packet drop claimed either the query or the response then the loss situation could be recovered by the use of timeouts, re-queries and broadening the circle of servers being queried. As long as the packet loss incidence is low, the overheads of loss detection and recovery are more than offset by the inherent efficiencies of UDP transport in the normal case. This implies that one of the constraining factors in DNS over UDP is to use UDP in a manner that minimizes the possibility of packet loss.

There are some measures that the DNS application may take in order to reduce the likelihood of DNS over UDP packet loss:

- Firstly, a sender should not generate UDP packets larger than the capability of the receiver to receive. In IPv4 the assured packet size that all hosts will receive is 576 octets. This means that a sender that has no additional information concerning the capabilities should restrict its UDP packet size to no larger than this size. In the case of the DNS this was implemented by an application limit that restricted the size of any DNS response to at most 512 octets of DNS payload. Larger responses were to be truncated and the receiver should interpret the truncated response as a signal to re-query using TCP if they wanted to receive the entire response.

- Secondly, for a UDP-based protocol to operate efficiently it really should avoid IP level fragmentation. In IPv4 the maximum packet size that will pass across IPv4 networks without

packet fragmentation is 68 octets. The DNS did not attempt to fit its transactions within a 40-octet payload limit (assuming that the IP packet had no options fields). Instead, it assumed that IP fragmentation was sufficiently robust to allow the application to work efficiently over UDP, and fragmentation of IP packets of up to 576 octets in size was very much an exceptional case.

This 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

The design assumption in the DNS application was that IP packet fragmentation was accepted as an unavoidable occurrence. However, the assumption that all IP hosts would need to participate in the DNS meant that it could not be assumed that all hosts would receive UDP packets larger than 576 octets in the IPv4 Internet of the time. Hence the DNS had a limit of 512 octets for the DNS payload when using UDP transport.

Since then, we've changed IP and we've changed the DNS. The major DNS change in terms of this consideration of transport options is the use of adding digital signatures to DNS responses. DNSSEC signatures add size to the response and keeping the overall majority of DNS responses under this UDP limit of 512 octets was increasingly challenging. At the same time the world appeared to coalesce on the original Ethernet models as the common denominator of packet carriage and there is a de facto universal unfragmented IP packet size maximum of 1,500 octets.

We could've elected to keep the specifications for DNS unchanged and use truncation and switch over to TCP for responses larger than 512 octets. However, this may be an acceptable behaviour if the incidence of larger responses is rare, but it can become inefficient if a significant proportion of DNS responses are larger than 512 octets. A client would need to query using UDP, receive the truncated response and then open up a TCP session to re-query. Without particular effort related to TCP session reuse, this two-step process will take an additional 2 round trip times for each such query and impose a TCP process overhead on the responding server.

The way we responded to the need to provide for larger DNS response while still retaining UDP as the default DNS transport protocol for larger responses was the use of a receiver buffer size indication in DNS queries using the EDNS(0) extension mechanism (RFC 6891).

Don't forget that the reason why DNS UDP responses were originally limited to 512 octets was not to avoid UDP fragmentation, but to be assured that the receiver can receive the DNS response at all. In the IPv4 Internet the minimum unfragmented IP packet is just 68 octets in size. It is quite permissible to fragment IP packets of 576 octets or smaller. All the EDNS(0) extension mechanism was intended to achieve was to allow the client to signal to the server that it could handle a DNS response passed over UDP that was larger than the default maximum payload size of 512 octets, irrespective of whether the response may be fragmented or not.

So, if the primary objective of this payload size parameter was not to avoid UDP fragmentation but simply to avoid the additional overheads associated with the premature onset of TCP when it is not strictly necessary, then why not use simply a larger default value for this parameter? If hosts had packet size limitations then they could simply not use an EDNS(0) extension in their queries or use a smaller buffer size parameter. It is this line of thinking that appears to lie behind the original default value of 4,096 octets for the buffer size that was proposed in the EDNS(0) specification in RFC 6891.

In short, the design trade-off implicit in this line of thought appeared to be that while IP fragmentation loss may be bad, the severity of the scenarios of IP packet fragmentation loss was believed to be lower than the service impact of prematurely switching to TCP.

However, we continue to appreciate the subtleties within the topic of network robustness. Packet fragmentation has both performance and security implications, and these days we work by a general principle that applications and platforms should really strive to avoid relying on IP fragmentation. In the best case, applications should avoid triggering IP packet fragmentation at all!

Along this line of thinking, we appear to have apparently decided to re-purpose the role of this EDNS(0) size parameter. As well as signalling the maximum packet size that the receiver can accept for incoming UDP packets, we're adding a further qualification that this size should be less than or equal to the maximum UDP payload size that can be passed through the network without triggering IP packet fragmentation. It is this thinking that lies behind the DNS Flag Day 2020 measures.

DNS Flag Day 2020

The DNS is a very heterogenous environment with many moving parts and many vendors. Coordinating change in the DNS has proved to be very challenging, and lately we've used the approach of "Flag Days" to coordinate collective actions. This allows DNS software vendors to make changes to their code base on the understanding that the changes are common to all code bases at the same time. This approach was first tried in 2019 when DNS resolver software stopped attempting to work around non-standard behaviours relating to EDNS processing. In 2020 a second Flag Day was adopted where the intent was to modify DNS protocol behaviour to avoid relying on fragmented UDP packets. Details of the 2020 Flag Day program can be found at https://dnsflagday.net/2020/.

On the Flag Day (1st October 2020) DNS client behaviour should use a default EDNS payload size value of 1,232 octets, and DNS server behaviour should not attempt to send DNS responses over UDP where the DNS message size is larger than 1,232 octets.

Naturally, this has raised a number of questions:

- Why choose 1,232 octets here? If the idea is to avoid IP packet fragmentation then why not chose a larger value, such as 1,452 octets.

- Just how bad is packet fragmentation loss in the DNS?

- How 'bad' is TCP? Is it better or worse than fragmented UDP?

DNS Measurements

We set out to measure this situation. The measurement system we used has its limitations and issues, and these should be noted at the outset.

The first limitation is that the DNS is actually a collection of loosely coupled elements and it is extremely challenging to look at the entire set of transactions all at once. There are the interactions between the stub resolver systems at the edge and one or more recursive resolvers. Each of these recursive resolvers perform name resolution tasks on behalf of these stub resolvers. And then there are authoritative servers that contain DNS information for a set of zones. Each zone may be published using a number of these authoritative servers. Resolvers and servers may be individual systems, or these days are commonly implemented using a distributed set of engines, either clustered in a service "farm" using a query distribution front end or distributed across the Internet using an anycast service address. Any measurement has to necessarily look at some constrained subset of the system. In our case we are constrained in looking at the DNS resolution performance between the recursive resolver and the

authoritative name server components. Our measurement technique cannot directly observe the interactions between the stub resolver and the recursive resolver.

The second limitation is that we are using a technique of DNS measurement that relies on the resolution of the name of a name server. A number of recursive resolvers alter their query settings when resolving such names and use smaller buffer size values or even none at all.

The third matter is an issue about DNS measurement that is common to many such exercises. This concerns the units of such a measurement. "What is a DNS resolver?" is a surprisingly difficult question to answer, and there is also the factor that some resolvers are more critical than others. A recursive resolver with a few hundred million stub clients is far more critical in overall terms than a personal use resolver. In this measurement exercise we use counts of end users as the unit of measurement. This approach essentially measures the DNS by measuring the percentage of tested users that sit behind recursive resolvers that exhibit a certain property. If a measurement reports that 1.8% of users sit behind DNS resolvers that cannot complete a resolution task using TCP, then this does not indicate how many resolvers have this behaviour. This user-based measurement is a measure of impact of a particular behaviour.

What's "Normal" these days?

To establish a baseline 'control measurement we used DNS responses that were generated using an authoritative server that performed a random selection from 11 different padding sizes in the response. The results are shown in Table 2.

Size	Tests	Passed	Failed	Rate
1230	4,303,845	4,282,457	21,388	0.50%
1270	4,308,667	4,287,046	21,621	0.50%
1310	4,307,456	4,286,064	21,392	0.50%
1350	4,304,230	4,282,752	21,478	0.50%
1390	4,310,182	4,288,413	21,769	0.51%
1430	4,303,906	4,281,858	22,048	0.51%
1470	4,308,722	4,269,785	38,937	0.90%
1510	4,303,923	4,197,910	106,013	2.46%
1550	4,306,824	4,194,465	112,359	2.61%
1590	4,300,559	4,187,575	112,984	2.63%
1630	4,305,525	4,191,994	113,531	2.64%

Table 2 – Baseline Results

Each of the individual tests require the recursive resolver to successfully resolve a control case of a DNS response of 175 octets, and they are then presented with a case where the response size is one of those listed in this table. Surprisingly, the DNS is not all that successful when attempting to resolve names where the DNS response is 1,230 octets in size. some 0.5% of cases, or 1 in 200, do not successfully complete the name resolution process. This failure rate is consistent for sizes up to 1,470 octets. For this size the corresponding IPv4 UDP packet is 1,498 octets, or just under 1,500 octets and the corresponding IPv6 UDP packet would be 1,518 packets, which will require UDP fragmentation. This size has a failure rate of 0.9%, which is consistent with a lower failure rate for IPv4 packets and a higher failure rate for IPv6 packets. For larger DNS responses the failure rate jumps to 2.6%, or around 1 in 40 users.

TCP

To understand this situation in a little more detail its necessary to understand the interaction between the resolver client and the authoritative server through the use of the payload size parameter.

Prior to the specification of the EDNS(0) UDP DNS payload size the default maximum UDP DNS response size was 512 octets. This was the maximum size of a DNS response that could be passed back to the resolver client. If the DNS message was larger than 512 octets, then the server was allowed to put as much of the answer as could fit and still stay within this 512 octet ceiling (or nothing if it wanted) and

set the truncate bit in the response. The client could either just use the provided answer (or the bits that could fit), or it could use the truncate bit as a signal to open a TCP connection with the server and ask the question over TCP. There is no comparable size limitation in TCP, and it is assumed that TCP will successfully transmit the complete answer.

EDNS(0) allows a client to specify a UDP DNS payload size that is larger than 512 octets. If the response is greater than this query-specified size, then once more the answer is truncated, and the truncate bit is set if the query was passed to the server over UDP.

In addition, the server may have its own maximum UDP DNS buffer size, and in this case the lower of the client-specified size in the query and the locally specified size is used to determine the UDP response truncation point.

While TCP queries could be initiated by a client without the trigger of a truncated UDP response, this is rarely the case. We observe that the overwhelming majority of TCP queries are made only after a truncated UDP response has been passed back to the resolver client.

The use of TCP in our measurement context is shown in Table 3.

Size	TCP Use		Pass	Fail		NO TCP	NO ACK	TCP OK
1230	9%		98.7%	1.3%		11.4%	28.3%	60.3%
1270	13%		99.0%	1.0%		12.2%	27.7%	60.1%
1310	13%		99.0%	1.0%		13.2%	26.4%	60.4%
1350	13%		99.0%	1.0%		13.0%	27.9%	59.0%
1390	14%		99.0%	1.0%		15.2%	27.1%	57.7%
1430	14%		99.1%	0.9%		15.7%	25.9%	58.5%
1470	30%		98.5%	1.5%		9.2%	58.3%	32.5%
1510	36%		98.1%	1.9%		22.7%	47.2%	30.1%
1550	36%		98.1%	1.9%		23.2%	46.8%	30.0%
1590	36%		98.1%	1.9%		24.5%	45.7%	29.8%
1630	36%		98.1%	1.9%		25.6%	45.5%	28.9%

Table 3 – TCP Use

Once the response size exceeds 1,470 octets then in a little over one third of cases the response is greater than the DNS UDP buffer size and a truncated response is passed back to the resolver client, with the intention of triggering a TCP re-query ("TCP Use").

TCP is relatively successful with a failure rate of some 1% of cases. Where the response is larger than 1,500 octets the failure rate increases by a further 0.9% ("Fail")

We classify the TCP failure into three cases. The first is where the TCP session does not get established ("NO TCP"). The second case is where the TCP session is established, the query is asked but the TCP session hangs once the response is passed back to the resolver client, and no ACK is seen for the response segments. This occurs in around a quarter of the failure cases for smaller packets and rises to a little under half of the cases for multi-segment larger responses ("NO ACK"). For these larger responses the most likely cause is a TCP Path MTU mismatch and the larger TCP segment cannot make it through to the client. The third case is where the TCP session appears to complete normally, but the overall resolution task fails in any case ("TCP OK").

It is noted that we used in this experiment a local interface MTU setting of 1,500 octets., so that we would encounter TCP Path issues, if they existed.

UDP

In a little over half of the cases the UDP DNS buffer size is 4,096 octets. This is lower than "normal" in that a number of resolvers change their behaviour when resolving name server names. The "normal"

value uses 4,096 octets some 82% of cases. This means that the TCP use in this experiment is around double the rate one would normally expect.

Size	UDP Use	Pass	Fail
1230	91%	99.6%	0.4%
1270	87%	99.6%	0.4%
1310	87%	99.6%	0.4%
1350	87%	99.6%	0.4%
1390	86%	99.6%	0.4%
1430	86%	99.6%	0.4%
1470	70%	99.4%	0.6%
1510	64%	97.2%	2.8%
1550	64%	97.0%	3.0%
1590	64%	97.0%	3.0%
1630	64%	97.0%	3.0%

Table 4 – UDP Use

Table 4 shows the failure rate for UDP-only resolutions. The failure rate for unfragmented UDP is less than half the failure rate for TCP, with a failure rate of 0.4% of cases. When the UDP response is fragmented, then the failure rate rises to 3.0% of cases, which is worse than the comparable failure rate for TCP. It appears that there are a number of resolvers that advertise a large UDP DNS buffer size but are not capable of receiving large UDP responses. This is most likely due to IP packet filters on the path where fragmentation drop is a common security setting in such filters.

Observations

What we've observed in these experiments is that once the DNS response exceeds 1,500 octets the failure rate rises five-fold from 0.5% to 2.5% of individual cases. Some 75% of these failures are failures where the server responds using fragmented UDP. In other words, the resolver client is advertising a large buffer size to the server, but efforts to respond using fragmented UDP based on this larger buffer size fail in these cases.

In many ways the use of the EDNS(0) buffer size parameter to control server behaviour to attempt to avoid UDP fragmentation is always going to be based on very crude guesses. A client cannot tell in advance what network path will be used by a server and the various server-to-client path MTU discovery techniques are based on server-side probes, not client-side probes. The packet size is determined by the server, and only the server is informed of any Path MTU mismatch via ICMP messages. The client is never in a position to understand the Path MTU characteristics of an Internet path. So the client EDNS(0) payload size setting is at best a crude substitute for a Path MTU setting.

If the intent is to reduce the inefficiencies and delays when attempting to recover from lost UDP packets, then setting a low buffer size, as in the Flag Day setting of 1,232 octets makes more sense than 4,096 octets.

But this relatively low buffer size setting has its own problems as it invokes TCP unnecessarily early, particularly in the context of recursive resolver-to-authoritative paths. TCP has its own issues. It's far slower than unfragmented UDP and certainly not as reliable. Not all resolver clients can use TCP, probably due to over-enthusiastic local packet filtering rules.

At this point we are now trying to position the DNS between three considerations. Unfragmented UDP is fast, efficient and very reliable. TCP is slower, less efficient and not as reliable, but with careful management of the TCP MSS settings to avoid TCP black holes can be tuned to be reasonably reliable. Fragmented UDP is the worst option of the three. It has a high failure rate and has to rely on local timers to expire to detect packet loss.

What this means is that UDP should be used for as long as it will not encounter fragmentation, and then the DNS should shift to TCP.

How can this be achieved? It is unreasonable to expect that a lightweight UDP-based packet exchange should perform a path MTU discovery operation for each and every transaction. This implies that both the client and the server should use conservative settings for transport parameters that avoid path MTU issues.

What should a DNS client (resolver) do?

The DNS Flag Day 2020 settings are a good start, but I think that they don't quite catch the entirety of the space. In the context of recursive resolver to authoritative server a recursive resolver client should use a EDNS(0) payload size setting equal to or only slightly less than 1,452 octets in IPv6 (accounting for a 40 octet IPv6 header and an 8 octet UDP header), and 1,472 in IPv4 (accounting for a 20 octet IPv4 header and an 8 octet UDP header).

For TCP, a recursive resolver client should use a TCP MSS setting less than 1440 octets in IPv6 (accounting for a 40 octet IPv6 header and a 20 octet TCP header) and 1460 octets in IPv4 (accounting for a 20 octet IPv4 header and an 20 octet TCP header).

What should a DNS server do?

The server should also avoid fragmentation, and it can do this by setting a maximum payload size value no larger than 1,452 in IPv6 and 1,472 in IPv4. It should also impose a ceiling on the size of outgoing TCP packets of 1,440 packets in IPv6 and 1,460 in IPv4.

Summary

Specific circumstances vary, and there is a difference between measurements at the edge of the Internet and within the infrastructure of the network. Our measurements of the behaviour of the inner infrastructure of the Internet between recursive resolvers and authoritative servers indicate that the network behaviour is relatively uniform with IP packet sizes up to 1,500 octets. If we restrict ourselves to settings that relate only to the transactions between recursive resolvers and authoritative servers then the DNS Flag Day 2020 setting of 1,232 octets are too low. The result is that the transaction will invoke TCP too early. A more efficient outcome can be achieved by pushing the UDP packet size to 1,500 octets including the IP header before triggering TCP.

At the same time, it is prudent to pull the TCP segment size down. The incremental performance cost of using a 1,200 octet MSS value is extremely small when looking at DNS transactions and such a small setting has a high level of assurance that TCP MTU black hole issues can be avoided.

This leads to some recommendations for transport parameter values for DNS clients and servers, shown in Table 24. The intent of these settings is to use UDP all the way to 1,500 octets of IP packet size, then use TCP with a more conservative MSS setting that increases the reliability of TCP sessions.

	IPv4	**IPv6**
Client EDNS(0) Buffer Size	1,472	1,452
Client TCP MSS	1,200	1,200
Server Max Buffer Size	1,472	1,452
Server max TCP MSS	1,200	1,200

It must be noted that these settings apply only to the "inside" of the Internet in the path between recursive resolvers and authoritative servers. The edge of the Internet shows greater levels of variability and it is probably prudent to use a lower UDP upper bound, although this is an aspect of the DNS where our measurement technique cannot gain a direct insight, so we've refrained from making any particular recommendations for the edge stub-to-recursive resolver scenario.

Disclaimer

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

Authors

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

Joao Luis Silva Damas is the Senior Researcher at APNIC.

www.potaroo.net