Geoff Huston
December 2016

Scoring the DNS Root Server System, Pt 2 – A Sixth Star?

In November I wrote about some simple tests that I had undertaken on the DNS Root nameservers. The tests looked at the way the various servers responded when they presented a UDP DNS response that was larger than 1,280 octets. I awarded each of the name servers up to five stars depending on how that managed to serve such large responses in IPv4 and IPv6. I'd like to return to this topic by looking at one further aspect of DNS server behaviour, namely the way in which servers handle large UDP responses over IPv6.

What is "large" in this context?

Don't forget that 1,280 is a "special" number for IPv6 – it's the size of the largest IPv6 packet that standards-confirming networks must the able to support without requiring IP packet fragmentation. Any IPv6 packet larger than 1,280 octets can be considered "large" for our purposes here.

In an IPv6 environment there is no assurance for end hosts that a packet larger than 1,280 octets can be passed through an IPv6 network. This implies that IPv6 hosts must be able to successfully negotiate transmission of larger IPv6 packets using IPv6 fragmentation when the network path is constrained such that packets larger than 1,280 octets are dropped, or find another approach using a transport protocol that avoids large packets altogether. The way an IPv6 host is informed of network elements that impose a packet size constraint is that the network element is meant to generate an ICMPv6 Packet Too Big message (RFC4443) when a large IPv6 packet is discarded, and pass this error notice back to the packet's sender. This ICMPv6 packet contains a conventional IPv6 header and an ICMP payload that contains the Packet Too Big (PTB) code, the Maximum Transmission Unit value of the next hop link that is the constraining factor, and as much of the original IPv6 packet as can be contained without exceeding this 1280 octet minimum IPv6 MTU. (Figure 1)

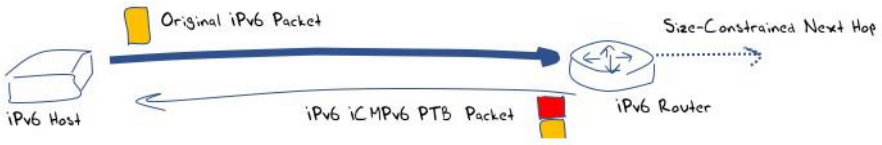

Figure 1 – IPv6 ICMPv6 Packet Too Big Behaviour

When a sending host receives such a ICMPv6 PTB message the IP processing element is required to inspect the inner IPv6 payload. To quote from RFC443, this control message "MUST be passed to the upper layer process if the relevant process can be identified."

Now all this is fine for TCP. The ICMPv6 PTB message is supposed to be passed to the TCP process and the MTU value is intended to be used to reduce the local host's value of the path Maximum Segment Size (MSS), so that the TCP session will adjust itself to ensure that it fits within the new path MSS value as per the ICMPv6 PTB message, Presumably, the TCP session will also resend any

outstanding unacknowledged data using this new MSS value to reframe the unacknowledged data stream. In the case of TCP, the sending host should never actually attempt to fragment an outgoing TCP packet, as the session's TCP MSS value should always be equal to or less that the host's Path MTU estimate.

But what about UDP? UDP transactions can be stateless transactions, and it's entirely unclear that there is an upper layer process or state left in the sending host once the UDP packet is sent. This is certainly the case for the DNS, where the UDP part of the server operates as a set of simple transactions where each UDP query generates a single UDP response. What is an IPv6 host meant to do with a received ICMPv6 PTB message when the message is a consequence of sending a UDP packet?

It would be helpful if IPv6 hosts "remembered" in some manner what the new MTU might be for destinations when they receive such an ICMPv6 message.

But do they?

One of the more critical applications that rely upon stateless UDP transactions is the Domain Name System (DNS). So we can focus our question a little more by asking: Do DNS nameservers adapt their behaviour in response to a received ICMPv6 PTB message? And if we want to be specific here, what do the DNS Root Servers do when they receive an ICMPv6 PTB message relating to a UDP response that they sent?

> The reason why I am focussing on the DNS root servers is due to the planned roll of the Key-Signing Key (KSK) of the DNS Root zone. There will be a period when two KSKs (old and new) are in the root zone at the same time, and the zone's DNSKEY record will be signed by both of these KSKs. The signed response to a query for the DNSKEY record will inflate from 864 octets in size to 1,425 octets at that point in time. In the context of the current keyroll plan this larger response will occur on the 11th January 2018, and last for 20 days (http://www.slideshare.net/apnic/rolling-the-root-zone-dnssec-key-signing-key, slide 28)
>
> As far as I am aware, this is the first time a 'normal' DNS response from the root servers will exceed 1,232 octets in length, and the IPv6 UDP packet will exceed 1,280 octets in length.
>
> It's critical in so far that if a DNSSEC-validating resolver is unable to retrieve the root zone keys, it will be unable to answer any further queries – it will go "black" and will need manual intervention to bring it back into server once more.

We'll try and answer this question using a simple test routine. The test involves sending each of the DNS Root name servers a query over UDP in IPv6 that will generate a response where the packet is over 1,280 octets in size.

> The code used to perform these tests can be found at https://github.com/gih900/icmpv6-ptb

The query used in this test related to a non-existent domain name, and the EDNS(0) extension is used by the query with the DNSSEC OK flag set, and the EDNS(0) UDP buffer size set to 4,096 octets. The *dig* utility was used to perform the query, so the utility performed UDP fragmentation reassembly, and also performed a fallback to TCP in response to a truncated UDP response. Each root name server was directly queried, bypassing any local DNS resolution infrastructure. A unique name was generated for each query, using the longest possible DNS name query string. The root servers generated a DNS response of 1,268 octets in size, which corresponded to a IPv6 UDP packet of 1,316 octets, 36 octets greater than the 1,280 unfragmented IPv6 packet size.

The query process was active for 10 hours, performing one query every two seconds. Each root server was queried in rotation, so each server was queried every 26 seconds. Each hour the ICMPv6 PTB responder was executed for a period of 3 minutes. The PTB responder generated a synthetic ICMPv6 PTB message by using a source address that contained a randomised Interface Identifier value and used a 1,280 octet MTU value, simulating a last hop router with a size-constrained next hop.

> These days the root servers operate using anycast constellations, so it is a challenging exercise to perform this experiment to query each and every individual instance of the anycast server set for all 13 root server constellations. In this experiment I used a server located close to an exchange point in London, and queried the "closest" anycast instance of each of the root servers. The assumption is that each root server letter has deployed every element of their constellation in exactly the same manner, with the same packet handling behaviour.

The results of this test are as follows:

Root	Behaviour
A	Always responds with a packet that is truncated so as to fit within a 1,280 octet limit - the test case response is 955 octets in size.
B	Always responds with a packet that is truncated so as to fit within a 1,280 octet limit - this server echoes only the query, so they response is 330 octets in size.
C	Appears to react to an ICMP PTB message, by sending a fragmented response for 10 minutes after receipt of the first ICMP PTB message.
D	Appears to react to an ICMP PTB message, by sending a fragmented response for 10 minutes after receipt of the first ICMP PTB message.
E	Appears to react to an ICMP PTB message, by sending a fragmented response for 10 minutes after receipt of the first ICMP PTB message.
F	Always fragments the response to fit within a 1280 MTU size.
G	Always responds with a packet that is truncated so as to fit within a 1,280 octet limit - this server echoes only the query, so they response is 330 octets in size.
H	Always sends a large UDP response and does not react to any incoming ICMPv6 PTB messages.
I	Appears to react to an ICMP PTB message, by sending a fragmented packet for 10 minutes after receipt of the first ICMP PTB message.
J	Always responds with a packet that is truncated so as to fit within a 1,280 octet limit - the test case response is 955 octets in size.
K	Always sends a large UDP response and does not react to any incoming ICMPv6 PTB messages.

L Appears to react to an ICMP PTB message, by sending a fragmented packet for 10 minutes after receipt of the first ICMP PTB message.

M Always fragments the response to fit within a 1280 MTU

It appears that the servers **C**, **D**, **E**, **I** and **L** act in a manner that appears to be consistent with the intent of the IPv6 ICMPv6 PTB semantics: this ICMPv6 error message generates a state change on the root server that appears to add an entry to the server's local route cache that associates the new MTU value with this IPv6 destination address. This route cache appears to use a lifetime of 10 minutes.

Servers **A**, **B**, **G** and **J** always truncate large responses, offering a UDP response no larger than 1,280 octets at all times.

Servers **F** and **M** always fragment in a manner that is consistent with a locally defined constraint of a UDP MTU of 1280 octets, irrespective of the UDP buffer size offered in the EDNS(0) settings in the query.

Servers **H** and **K** never adjust the packet size from their local MTU setting, and always sends the large UDP response irrespective of any ICMPv6 PTB messages. It appears that they are using platforms that have disabled the local IPv6 MTU route cache.

That's a wealth of variation in behaviour, but which of these four classes of behaviour is better here?

Each of these four behaviours appear to be the result of efforts to steer between known obstacles:

- Fragmented packets have a lower delivery reliability level than unfragmented packets because of common firewall filtering practices. Because the trailing fragment does not contain any transport level information a simple firewall's response is to either accept all fragments or deny all fragments. It's common to deny all packet fragments under such circumstances. Servers should avoid gratuitous fragmentation to minimise this problem.

- IPv6 has a further problem with packet fragmentation: fragmentation in IPv6 uses Extension Headers, and there are a number of network switches that are ill-equipped to handle IPv6 packets that contain IPv6 Extension Headers. These units simply drop IPv6 packets that contain Extension Headers, including fragmentation control Extension Headers. Previous measurements (reported at http://www.potaroo.net/ispcol/2016-05/v6frags.html) suggest that some 35% of visible DNS resolvers lie behind such devices and they cannot receive fragmented IPv6 packets. In the case of IPv6 the desire to avoid packet fragmentation is a serious consideration.

- ICMPv6 PTB packets are also commonly filtered. A server that relies on the successful reception of ICMPv6 PTB messages is also going to encounter cases where the control message is filtered, and the server has no backup method of adjusting its sending MTU in the case of a UDP transaction.

- An IPv6 server also has to defend itself against ICMPv6 PTB attacks. Because of the stateless nature of UDP it is readily possible to send an avalanche of ICMPv6PTB messages at a server. The host has no inherent ability to differentiate between a genuine ICMPv6 PTB message and a synthetic message. To defend itself, it will need to limit the size of the local MTU route cache, and not push these messages into the route cache once the size of the cache exceeds some system-defined upper limit, or drop the cache lifetime. In either case an overwhelming number of such ICMPv6 PTB messages would effectively invalidate the utility of this route cache. The other aspect of the handling of these messages is the CPU processing time taken to examine the

message, enter it into the MTU route cache and maintain the cache itself, exposing the host to a DDOS vector through the processing overhead associated with such messages.

- Using TCP as a fallback when attempting to send a large packet makes a lot of sense, except in the case of the DNS. While the use of TCP in the DNS is part of the DNS specification, it's not universally supported and more than 10% of the visible resolver population appear to be incapable of making a query over TCP. It is possible that this is the result of overly zealous firewall filters that only open UDP port 53 for DNS traffic and exclude TCP.

- Where TCP is used, the combination of lost ICMPv6 PTB messages and TCP also exposes the DNS TCP session to a risk of PTMU Blackholes. If the server pushes out a response that is larger than the Path MTU, and the ICMPv6 PTB message is lost, then the server will be waiting for an ACK that will never arrive and the client TCP session has no unacknowledged sent data, so it too will hang.

The use of a truncated response by servers **A**, **B**, **G** and **J** force the resolver into using TCP to complete their query. Earlier measurements (http://www.potaroo.net/ispcol/2013-09/dnstcp.html) show that some 17% of visible resolvers are unable to perform a DNS query using TCP.

Servers **C**, **D**, **E**, **I** and **L** also have a problem on those network paths that have a constrained MTU - the use of fragmentation will incur packet loss due to IPv6 Extension filtering and firewall filtering. This appears to affect some 35% of visible DNS resolvers using IPv6 (http://www.potaroo.net/ispcol/2016-10/dnsipv6.html).

Servers **F** and **M** have a similar issue, and due to problems with the transmission of fragments they also will be unable reach about 35% of visible DNS resolvers using IPv6.

Servers **H** and **K** will be unable to reach any resolvers that lie behind paths where the path MTU is smaller than the response they are sending.

There is no perfect answer here.

Perhaps the best compromise that can be achieved is to use a large MTU for UDP and react to ICMPv6 PTB, as is done by **C**, **D**, **E**, **I** and **L**.

The approach adopted by **A**, **B**, **G** and **J** to avoid sending large IPv6 UDP packets at all and instead push the resolver to re-query using TCP also has some merit.

Part of the issue here is attempting to assess which has the greater failure rate: failure of the server to receive an ICMP PTB packet, failure of the resolver to receive a fragmented IPv6 UDP packet or failure of the resolver client to perform DNS queries over TCP.

Starting with a large UDP packet response and then reacting to ICMPv6 PTB messages appears to offer a reasonable compromise between efficiency and maximising the chances that the first response will be used by the querying DNS resolver. Truncating the initial UDP response and forcing the resolver to use TCP to complete the query would appear to have a higher probability of ultimate success at the cost of efficiency, were it not for the problem of widespread filtering of DNS over TCP.

So I'll nominate only the root servers **C**, **D**, **E**, **I** and **L** for a sixth star here, as their approach appears to strike a reasonable balance between speed and reliability for large DNS responses over IPv6, despite the observation that it's by no means the optimal form of response. The problem here is that there is

no clear path that avoids all of these obstacles, so every form of response represents a compromise of sorts.

If this problem is so bad then why does the DNS work at all?

Firstly, the root servers do not normally send large packets. The exercise here uses an artificially constructed query that pushes the root server into constructing a large response. In any case the response is of the form NXDOMAIN - i.e. not only is this a rare case, the answer is basically a negative one! However, this will not always be the case - for a period in January 2018, from the 11th of January until the end of that month, the response to a query for the root zone's DNSKEY resource record will be 1,425 octets, so DNSSEC-resolving resolvers will encounter this large packet issue at that time. So while this is a largely esoteric issue for the moment, for a brief two week period in about a year from now it will be an issue that will be critical to the stable operation of the DNSSEC security framework of the DNS!

Secondly, there are thirteen root name server clusters using four distinct behaviours for sending large responses. There is a good chance that if a resolver cannot receive a response from one server it will be able to negotiate a response from another server.

Thirdly, and perhaps most importantly here, there is still IPv4 to help us out of this mess! Only some 11% of queries to a dual stack name server use IPv6, and the issues encountered here with IPv6 and packet fragmentation do not spill over into IPv4. For as long as the Dual Stack world persists for the DNS this IPv6 behaviour is not a major issue for us.

But a Dual Stack Internet is not the long term outcome planned for the Internet. We need to be considering an environment where IPv4 is no longer available, and in such an IPv6-only environment IPv6 and the DNS present some particular challenges. Much of these challenges would be addressed were we able to clean up just two practices: Extension Headers need to be treated in the same manner as all other packets, and not dropped, and ICMPv6 PTB messages need to be passed back to the source. Of course another option is to drop UDP completely and move the DNS on to a TLS platform that is layered on a TCP end-to-end transport. The downside here lies in the higher overheads imposed by adding session maintenance to what was a simple stateless query/response protocol.

Even in a TCP world there are still Path MTU issues. And here the ICMP PTB and packet fragmentation behaviour is less tractable. Even if all ICMP V6 PTB messages were to be passed through the issue of firewall policies and the vulnerability of exposing the host's MTU cache to external attack makes this a more challenging issue. But that does not imply that it is insoluble. One viable approach is to drop the IPv6 TCP MSS for the DNS to a default of 1,220 octets. This results in a slightly lower transmission efficiency for DNS, but the upside is the

prospect of greater resiliency of the TCP session in the face of arbitrary filtering actions.

If this lower MSS makes you uncomfortable, then there are few options left One possible approach is to revisit the concept of tunnelling in the Internet. Rather than view fragmentation as an IP layer function, one response is to push this one layer down in the protocol stack. What if we used tunnels that took it upon themselves to fragment packets as necessary on tunnel ingress and reassemble the tunnel payload fragments on tunnel egress? In terms of end-to-end path MTU such tunnels would be transparent, and we could remove the rather uncomfortable and error-prone interactions between the host and the network that are associated with the current model of MTU discovery. But I suspect that this is just a step too far to contemplate for the Internet.

If we are serious about an all-IPv6 network at some point in the future we need to revisit IPv6 packet fragmentation and make some changes to the way we currently operate IPv6 hosts, networks and applications. Relying on IPv4 as an ever-present plan B to fix up the gaping holes in the IPv6 end-to-end story is certainly a pragmatic short term tactical response. But it's a pretty lousy long term strategy.

Author

Geoff Huston B.Sc., M.Sc., is the Chief Scientist at APNIC, the Regional Internet Registry serving the Asia Pacific region. He has been closely involved with the development of the Internet for many years, particularly within Australia, where he was responsible for building the Internet within the Australian academic and research sector in the early 1990's. He is author of a number of Internet-related books, and was a member of the Internet Architecture Board from 1999 until 2005, and served on the Board of Trustees of the Internet Society from 1992 until 2001 and chaired a number of IETF Working Groups. He has worked as an Internet researcher, as an ISP systems architect and a network operator at various times.

www.potaroo.net

Disclaimer