Geoff Huston
August 2017

IPv6, Large UDP Packets and the DNS

The IPv6 protocol introduced very few changes to its IPv4 predecessor. The major change was of course the expansion of the size of the IP source and destination address fields in the packet header from 32-bits to 128-bits. There were, however, some other changes that apparently were intended to subtly alter IP behaviour. One of these was the change in treatment of packet fragmentation.

It appears that rather than effecting a slight improvement from IPv4, the manner of fragmentation handling in IPv6 appears to be significantly worse than IPv4. Little wonder that there have been calls from time to time to completely dispense with packet fragmentation in IPv6, as the current situation with IPv6 appears to be worse than either no fragmentation or the IPv4-style of fragmentation.

One of the more difficult design exercises in packet switched network architectures is that of the design of packet fragmentation.

In time-switched networks, developed to support a common bearer model for telephony, each 'unit' of information passed through the network occurred within a fixed timeframe, which resulted in fixed size packets, all clocked off a common time base. Packet-switched networks dispensed with such a constant common time base, which, in turn allowed individual packets to be sized according to the needs of the application as well as the needs and limitations of the network substrate.

For example, smaller packets have a higher packet header to payload ratio, and are consequently less efficient in data carriage and impose a higher processing load as a function of effective data throughput. On the other hand, within a packet switching system the smaller packet can be dispatched faster, reducing head-of-line blocking in the internal queues within a packet switch and potentially reducing network-imposed jitter as a result. This can make it easier to use the network for real time applications such as voice or video. Larger packets allow larger data payloads which in turns allows greater carriage efficiency. Larger payload per packet also allows a higher internal switch capacity when measured in terms of data throughput, which, in turn, facilitates higher capacity and higher speed network systems.

Various network designs adopted various parameters for packet size. Ethernet, invented in the early 1970's adopted a variable packet size, with supported packet sizes of between 64 and 1,500 octets. FDDI, a fibre ring local network, used a variable packet size of up to 4,478 octets. Frame Relay used a variable packet size of between 46 and 4,470 octets. The choice of a variable-sized packets allows to applications to refine their behaviour. Jitter and delay-sensitive

applications, such as digitised voice, may prefer to use a stream of smaller packets in an attempt to minimise jitter, while reliable bulk data transfer may choose a larger packet size to increase the carriage efficiency. The nature of the medium may also have a bearing on this choice. If there is a high bit error rate (BER) probability, then reducing the packet size minimises the impact of sporadic errors within the data stream, which may increase throughput in such environments.

In designing a network protocol that is intended to operate over a wide variety of substrate networking media and support as wide a variety of applications as possible, the designers of IP could not rely on a single packet size for all transmissions. Instead, the designers of IPv4 provided a packet length field in the packet header. This field was a 16-bit octet count, allowing for an IP packet to be anywhere from the minimum size of 20 octets (corresponding to an IP header without any payload) to a maximum of 65,535 octets.

Obviously not all packets can fit into all substrate media. If the packet is too small for the minimum payload size then it can be readily padded. But if it's too big for the media's maximum packet size, then the problem is a little more challenging. IPv4 solved this using "forward fragmentation." The basic approach is that any IPv4 router that is unable to forward an IPv4 packet into the next network because the packet is too large for the next hop network may split the packet into a set of smaller "fragments," copying the original IP header fields into each of these fragments, then forwarding each of these fragments instead. The fragments continue along the network path as autonomous IP packets, and the destination host is responsible for re-assembling these fragments back into the original IP packet and pass the result, namely the packet as it was originally sent, back up to the local instance of the end-to-end transport protocol.

It's a clever approach, as it hides the entire network-level fragmentation issue from the upper level protocols, including TCP and UDP, but it has accreted a lot of negative feedback. Packet fragmentation was seen as being a source of inefficiency, a security vulnerability and even posed a cap on maximal delay bandwidth product on data flows across networks.

IPv6 removed the fragmentation controls from the common IPv4 packet header, and placed them into an "Extension Header". This additional packet header was only present in fragmented packets. Secondly, IPv6 did not permit fragmentation to be performed when the packet was in transit within the network: all fragmentation was to be performed by the packet source prior to transmission. This too has resulted in an uncomfortable compromise, where an unforeseen need for fragmentation relies on ICMP signalling and retransmission.

In the case of TCP a small amount of layer violation goes a long way, and if the sending host is permitted to pass IPv6's Packet Too Big ICMPv6 diagnostic message up to the TCP session that generated the original packet, then it's possible for the TCP driver to adjust its sending Maximum Segment Size to the new smaller value and carry on. In this case, no fragmentation is required.

UDP is different, and in UDP a functional response to path message size issues inevitably relies on interaction with the upper level application protocol.

It appears that when we consider fragmentation in IPv6 we have to consider the treatment of IPv6 Extension Headers and UDP.

The DNS is the major user of UDP, and as a consequence of the increasing use of DNSSEC, coupled with the increasing use of IPv6 as the IP protocol transition gathers momentum, and it's time to look once more at the interaction of larger DNS payloads over IPv6.

To illustrate this situation, here are two DNS queries, both made by a recursive resolver to an authoritative name server, both using UDP over IPv6.

Query 1

$ dig +bufsize=4096 +dnssec 000-4a4-000a-000a-0000-b9ec853b-241-1498607999-2a72134a.ap2.dotnxdomain.net. @8.8.8.8

; <<>> DiG 9.9.5-9+deb8u10-Debian <<>> +bufsize=4096 +dnssec 000-4a4-000a-000a-0000-b9ec853b
-241-1498607999-2a72134a.ap2.dotnxdomain.net. @8.8.8.8
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 43601
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 512
;; QUESTION SECTION:
;000-4a4-000a-000a-0000-b9ec853b-241-1498607999-2a72134a.ap2.dotnxdomain.net. IN A

;; ANSWER SECTION:
000-4a4-000a-000a-0000-b9ec853b-241-1498607999-2a72134a.ap2.dotnxdomain.net. 0 IN A 139.162.21.135
000-4a4-000a-000a-0000-b9ec853b-241-1498607999-2a72134a.ap2.dotnxdomain.net. 0 IN RRSIG A 5 4 60 20170803045714
20170706035714 2997 000-4a4-000a-000a-0000-b9ec853b-241-1498607999-2a72134a.ap2.dotnxdomain.net.
FpuBXVfZ9KXzizaJhQkk1TZF3f26pbYhIBjeZ51euEuY/zMxLgXmGfSh
cqPJ6zAPdBc+RTT5z0k7nw+ZcPsnj2qdhIXZQRysnxdTCCfqsrmO1yVY
zWy0hAAOzS3T6e2E4tv5w3L28M6Ie8d2Me4QNKDuT9n/JQLxndJKwAmz hUk=
000-4a4-000a-000a-0000-b9ec853b-241-1498607999-2a72134a.ap2.dotnxdomain.net. 0 IN RRSIG A 42 4 60
20170803045714 20170706035714 47541 000-4a4-000a-000a-0000-b9ec853b-241-1498607999-
2a72134a.ap2.dotnxdomain.net.
lHLMBbT+oOj1FVl2W7Bv8AnowHGUuJeUqukha8akRloWEmDLEACzBhUM
fHC862DF1CA3aNnO/lO4He5+wjTZ2Ec5o1c5Vl1OYtq+HsUe5Jk+GwlX
6f6boRyJIN6++bYFMlpca7C6uROUdzFZlRXz0zD16xrzhrsPD9vtzdSk
0gb+L3Gu6SrBfaHz1jYlyQo5vvVTsnsOwYrqr1i+UyrFUVk2/0Jhwb8C
tJY5vF9D9R44SNCzV5E9QUCV/5PAxOQZ++RcXKUbXlmxlxUR2gsvEIP/
xaqQA+vRmOtkWK9JcqotzgbS6WUrm/xArNdL2+mf2q9JarI1O0ogoKPP
6RV6FuOA6MzlE2fiUxO5n+6iPshRhzMDvG5O3A7xrPcGJg3ppvW1jAgd
blwwJ/sfyTnnG8AaHn2JbFmXXQWPYyucTNKSAl6aH8z2T/PxbrwqVtfr
cPZo+WLBkcDHICPyvHDETnIi6ZHu3+Dh0U+e+6V15hVrTg/OEKCO18Cx

```
yXwhjsuTsLQkn5MFgGRUHmD5lEYO/5UdzaW9W3x9DUX6LtPFwoR55iMM
66NxP8LROFYXR1WsZCNRln7Nn4sTmbrmXnxq12KN5E4xVY3zJsZJPQ2e
6nHBO5NACTPLHMyFAisBbQJk+uayKzs/HmEFZ58SBomEx8QXB81K0+kX
WOxCWllEvlMrcH9mr53ltQVnxwvwS9K0Y9qCra1rxAVXBl+wSx0Edo2D
3D0gpPlIC7kw+wUDsGjdMhWKndqP9eDvpSsMqaGaLH7XTSLJci6CoymH
ptnvgwsFDanfnJ6/i0PrmO2MMhkKCWYt0tlbVHyE3CJey6Vp0LISr06w
b9r0WnLh5qT68a1hHn86edO2/a/YW3t9xUsv1/t9iGpXfMTJXaptV5sa
uLmZ8jJtqDAcgIuX/VDjLCjeqrBIASwMy4m2OOBSU5kL7Is+WTRudrT5
DbJK8N5yzogiFopOIlU=

;; Query time: 3728 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Thu Jul 06 04:57:16 UTC 2017
;; MSG SIZE  rcvd: 1190
```

Query 2

```
$ dig +bufsize=4096 +dnssec 000-510-000a-000a-0000-b9ec853b-241-1498607999-2a72134a.ap2.
dotnxdomain.net. @8.8.8.8

; <<>> DiG 9.9.5-9+deb8u10-Debian <<>> +bufsize=4096 +dnssec 000-510-000a-000a-0000-b9ec853b-241-1498607999-
2a72134a.ap2.dotnxdomain.net. @8.8.8.8
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: SERVFAIL, id: 34058
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 512
;; QUESTION SECTION:
;000-510-000a-000a-0000-b9ec853b-241-1498607999-2a72134a.ap2.dotnxdomain.net. IN    A

;; Query time: 3477 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Thu Jul 06 04:57:41 UTC 2017
;; MSG SIZE  rcvd: 104
```

What we see here are two almost identical DNS queries that have been passed to Google's Public DNS service to resolve.

The queries differ in a sub-field in the query which is '**4a4'** in the first query and '**510'** in the second. The name server used here is a highly modified name server, and it constructs a response by interpreting the value of this hexadecimal sub-field as a size parameter. The DNS response is constructed to include additional padding of the requested size. In the first case the DNS response is 1,190 octets in length, and in the second case the response is 1,346 octets in length. The DNS server is a IPv6-only server, and the underlying host of this name server is configured with a local maximum packet size of 1,280 octets. This means that in the first case the response being sent to the Google resolver is a single unfragmented IPv6 UDP packet, and the second case the response is broken into two fragmented IPv6 UDP packets. And it is this single change that triggers the Google Public DNS Server to provide the intended answer in the first case, but to return a SERVFAIL failure notice in

response to the fragmented IPv6 response. When the local MTU on the server is lifted from 1,280 octets to 1,500 octets the Google resolver returns the server's DNS response in both cases.

What's going on?

The only difference in the two responses is IPv6 fragmentation, but there is perhaps more to it than that.

IP fragmentation in both IPv4 and IPv6 raises the eyebrows of firewalls. Firewalls typically use the information provided in the transport protocol header of the IP packet to decide whether to admit or deny the packet. For example, you may see firewall rules admitting packets using TCP port 80 and 443 as a way of allowing web traffic through the firewall filter. For this to work the inspected packet needs to contain a TCP header and the fields in the header are used to match against the filter set. Fragmentation in IP duplicates the IP portion of the packet header, but the inner IP payload, including the transport protocol header, is not duplicated in every ensuring packet fragment. This means that trailing fragments pose a conundrum to the firewall. Either all trailing fragments are admitted, which has its own set of consequent risks, or all trailing fragments are discarded, which also poses connection issues.

> These issues are discussed in an Internet Draft "Why Operators Filter Fragments and What It Implies" (https://tools.ietf.org/html/draft-taylor-v6ops-fragdrop-02)

IPv6 adds a further factor to the picture. In IPv4 every IP packet, fragmented or not, contains IP fragmentation control fields. In IPv6 these same fragmentation control fields are included in an IPv6 Extension Header that is only attached to packets that are fragmented. This 8 octet extension header is placed immediately after the IPv6 packet header in all fragmented packets. This means that a fragmented IPv6 packet does not contain the Upper Level Protocol header starting at octet offset 40 from the start of the IP packet header, but in the first packet of this set of fragmented packets, the upper level protocol header is chained off the fragmentation header, at byte offset 48, assuming that there is only a Fragmentation Extension Header in the packet. The implications of this are quite significant. Instead of always looking at a fixed point in a packet to determine its upper level protocol, you need to unravel the extension header chain. This raises two rather tough questions. Firstly, how long are you prepared to spend unravelling this chain? Secondly, would you be prepared to pass on a packet with an extension header that you don't recognise?

In some cases implementers of IPv6 equipment have found it simpler to just drop all IPv6 packets that contain Extension Headers. Some measurements of this behaviour are reported in RFC7872 (https://tools.ietf.org/html/rfc7872). This RFC reports a 38% packet drop rate when sending fragmented IPv6 packets to DNS Name servers.

But the example provided above is in fact the opposite case in the DNS, and illustrates a more conventional case. It's not the queries in the DNS that can grow to sizes that required packet fragmentation, but the responses. The relevant question here is what is the anticipated probability of packet drop when sending fragmented UDP IPv6 packets as responses to DNS queries? To rephrase the question slightly, how do DNS recursive resolvers fare when the IPv6 response from the server is fragmented?

For a start, it appears from this above example that Google's Public DNS resolvers experience some packet drop problem when passed a fragmented IPv6 response. But is this a problem that is limited to

Google's Public DNS service, or do other DNS resolvers experience a similar packet drop issue? How widespread is this problem?

We tested this question using three approaches.

I. Repairing Missing "Glue" with Large DNS packets

The measurement technique we are using is based on scripting inside online ads. This allows us to instrument a server and get the endpoints who are executing the measurement script to interact with the server. However, we cannot see what the endpoint is doing. For example, we can see from the server when we deliver a DNS response to a client, but we have no clear way to confirm that the client received the response. Normally the mechanisms are indirect, such as looking at whether or not the client then retrieved a web object that was the target of the DNS name. This measurement approach has some degree of uncertainty, as there are a number of reasons for a missing web object fetch, and the inability to resolve the DNS name is just one of these reasons. Is there a better way to measure how DNS resolvers behave?

The first approach we've used here is so-called "glueless" delegation, and use of dynamically named DNS name servers. The basic approach is to remove the additional section from the "parent" DNS response that lists the IP address of the authoritative name servers for the delegated "child" domain. A resolver, when provided with this answer must suspend its effort to resolve the original DNS name and instead resolve the name server name. Only when it has completed its task can it resume the original name resolution task. We can manipulate the characteristics of the DNS response from the name server name, and we can confirm if the resolver received the response by observing whether it was then able to resume the original resolution task and query the child name server.

We tested the system using an IPv6-only name server address response that used three response sizes:

Small	169 octets
Medium	1,428 octets
Large	1,886 octets

The local MTU on the server was set to 1,280 octets, so both the medium and large responses were fragmented.

This test was loaded into an online advertising campaign.

Results - I

68,229,946 experiments

35,602,243 experiments used IPv6-capable resolvers

Small: 34,901,983 / 35,602,243 = 98.03% = 1.97% Drop

Medium: 34,715,933 / 35,666,726 = 97.335 = 2.67% Drop

Large: 34,514,927 / 35,787,975 = 96.44% = 3.56% Drop

The first outcome from this data is somewhat surprising. While the overall penetration of IPv6 in the larger Internet is currently some 15% of users, the DNS resolver result is far higher. Some 52% of these 68M experiments directed their DNS queries to recursive resolvers that were capable of posing their DNS queries over a IPv6 network substrate.

That's an interesting result:

Some 52% of tested endpoints use DNS resolvers that are capable of using IPv6

Interpreting the packet drop probabilities for the three sizes of DNS responses is not so straight forward. There is certainty an increased probability of drop for the larger DNS responses, but this is far lower than the 40% drop rate reported in RFC 7872.

It seems that we should question the experimental conditions that we used here. Are these responses actually using fragmentation in IPv6?

We observed that a number of recursive resolvers use different query options when resolving the addresses of name servers, as distinct from resolving names. In particular, a number of resolvers, including Google's public DNS resolvers, strip all EDNS(0) query options from these name server address resolution queries.

When the query has no EDNS(0) options, and in particular when there is no UDP Buffer size option in the query, then the name server responds with what will fit in 512 octets. If the response is larger, and in our case this includes the Medium and Large tests, the name server sets the Truncated Response flag in its response to indicate that the provided response is incomplete. This Truncated Response flag is a signal to the resolver that it should query again, using TCP this time.

In the case of this experiment we saw some 18,571,561 medium-size records resolved using TCP and 19,363,818 large-size records resolved using TCP. This means that the observed rate of failure to resolve the name is not necessarily attributable to an inability to receive fragmented IPv6 UDP packets.

Perhaps if we remove all those instances that use TCP to retrieve the large DNS response, then what do we have left?

UDP-only queries:

Small: 34,901,983 / 35,602,243 = 98.03% = 1.97% Drop

Medium: 16,238,433 / 17,095,165 = 92.21% = 5.01% Drop

Large: 15,257,853 / 16,424,157 = 93.90% = 7.10% Drop

There is certainly a clearer signal here in this data - some 5% to 7% of experiments used DNS resolvers that appeared to be incapable of retrieving a fragmented IPv6 UDP DNS response, as compared to the "base" loss rate as experienced by the control small response of 2%. Tentatively, we can propose that a minimum of 3% of clients use DNS resolvers that fail to receive fragmented IPv6 packets.

However, in doing this we have filtered out more than one half of the tests, and perhaps we have filtered out those resolvers that cannot receive IPv6 fragmented packets.

II. Large DNS Packets and Web Fetch

Our second approach was to use a large response for the 'final' response for the requested name.

The way in which this has been done is to pad the response using bogus DNSSEC signature records (RRSIG). These DNSSEC signature records are bogus in the sense that the name itself is not DNSSEC-signed, so the content of the digital signature will never be checked, but as long as the resolver is using EDNS(0) and has turned on the DNSSEC OK bit, which occurs in some 70% of all DNS queries to authoritative name servers, then the DNSSEC signature records will be added to the response.

We are now looking at the web fetch rate, and looking for a variance between the web fetch rates when the DNS responses involve UDP IPv6 fragmentation. We filtered out all experiments that did not fetch the small DNS web object, all experiments that did not set the DO bit in their query, and all experiments that used TCP for the medium and large experiments. In this case, we are looking for those experiments where a fragmented UDP IPv6 response was passed and testing whether or not the endpoint retrieved the web object.

Results - II

68,229,946 experiments

25,096,961 experiments used UDP IPv6-capable resolvers

and had the DO bit set in the query

Medium: 13,648,884 / 25,096,961 = 54.38% = 45.62% Drop

Large: 13,476,800 / 24,969,527 = 53.97% = 46.03% Drop

This is a result that is more consistent with the drop rate reported in RFC 7872, but there are a number of factors at play here, and it is not clear exactly how much of this drop rate can be directly attributed to the issue of packet fragmentation in IPv6 and the network's handling of IPv6 packets with Extension Headers. Again, there is also the consideration that in only looking at a subset of resolvers, namely those resolvers that can use IPv6, use EDNS(0) options and set the DO bit in these queries

III. Fragmented Small DNS Packets

Let's return to the first experiment, as this form of experiment has far less potential sources of noise in the measurement. We are wanting to test whether a fragmented IPv6 packet can be received by recursive DNS resolvers, and our use of a large fragmented response is being frustrated by DNS truncation.

What if we use a customised DNS name server arrangement that gratuitously fragments the small DNS response itself? While the IPv6 specification specifies that network Path MTU sizes should be no smaller than 1,280 octets, it does not specify a minimum size of fragmented IPv6 packets.

The approach we've taken in this experiment is to use a user level packet processing system that listens on UDP port 53 and passes all incoming DNS queries to a back-end DNS server. When it receives a response from this back-end server it generates a sequence of IPv6 packets that fragments the DNS payload and uses a raw device socket to pass these packets directly to the device interface.

We are relying on the observation that IPv6 packet fragmentation occurs at the IP level in the protocol stack, so the IPv6 driver at the remote end will reassemble the fragments and pass the UDP payload to the DNS application, and if the payload packets are received by the resolver, there will be no trace that the IPv6 packets were fragmented.

As we are manipulating the response to the query for the address of the name server, we can tell if the recursive resolver has received the fragmented packets if the resolver resumes its original query sequence and queries for the terminal name.

Results - III

10,851,323 experiments used IPv6 queries for the name server address

6,786,967 experiments queried for the terminal DNS name

Fragmented Response: 6,786,967 / 10,851,323 = 62.54% = 37.45% Drop

This is our second result:

> **Some 37% of endpoints used IPv6-capable DNS resolvers that were incapable of receiving a fragmented IPv6 response.**

We used three servers for this experiment, on serving Asia Pacific, a second serving the America and the third serving Eurasia and Africa. There are some visible differences in the drop rate:

Asia Pacific:	31% Drop
Americas:	37% Drop
Eurasia & Africa:	47% Drop

Given that this experiment occurs completely in the DNS, we can track each individual DNS resolver as they query for the name server record then, depending on if they receive the fragmented response, query for the terminal name. There are approximately 2 million recursive resolvers in today's Internet, but only some 15,000 individual resolvers appear to serve some 90% of all users. This implies that the behaviour of the most intensively used resolvers has a noticeable impact on the overall picture of capabilities if DNS infrastructure for the Internet.

We saw 10,115 individual IPv6 addresses used by IPv6-capable recursive resolvers. Of this set of resolvers, we saw 3,592 resolvers that consistently behaved in a manner that was consistent with being unable to receive a fragmented IPv6 packet, The most intensively used recursive resolvers which exhibit this problem are shown in the following table.

Resolver	Hits	AS	AS Name	CC
2405:200:1606:672::5	4,178,119	55836	RELIANCEJIO-IN Reliance Jio Infocomm Limited	IN
2402:8100:c::8	1,352,024	55644	IDEANET1-IN Idea Cellular Limited	IN
2402:8100:c::7	1,238,764	55644	IDEANET1-IN Idea Cellular Limited	IN
2407:0:0:2b::5	938,584	4761	INDOSAT-INP-AP INDOSAT Internet Network Provider	ID
2407:0:0:2a::3	936,883	4761	INDOSAT-INP-AP INDOSAT Internet Network Provider	ID
2407:0:0:2a::6	885,322	4761	INDOSAT-INP-AP INDOSAT Internet Network Provider	ID
2407:0:0:2b::6	882,687	4761	INDOSAT-INP-AP INDOSAT Internet Network Provider	ID
2407:0:0:2b::2	882,305	4761	INDOSAT-INP-AP INDOSAT Internet Network Provider	ID
2407:0:0:2a::4	881,604	4761	INDOSAT-INP-AP INDOSAT Internet Network Provider	ID
2407:0:0:2a::5	880,870	4761	INDOSAT-INP-AP INDOSAT Internet Network Provider	ID
2407:0:0:2a::2	877,329	4761	INDOSAT-INP-AP INDOSAT Internet Network Provider	ID
2407:0:0:2b::4	876,723	4761	INDOSAT-INP-AP INDOSAT Internet Network Provider	ID
2407:0:0:2b::3	876,150	4761	INDOSAT-INP-AP INDOSAT Internet Network Provider	ID
2402:8100:d::8	616,037	55644	IDEANET1-IN Idea Cellular Limited	IN
2402:8100:d::7	426,648	55644	IDEANET1-IN Idea Cellular Limited	IN
2407:0:0:9::2	417,184	4761	INDOSAT-INP-AP INDOSAT Internet Network Provider	ID
2407:0:0:8::2	415,375	4761	INDOSAT-INP-AP INDOSAT Internet Network Provider	ID
2407:0:0:8::4	414,410	4761	INDOSAT-INP-AP INDOSAT Internet Network Provider	ID
2407:0:0:9::4	414,226	4761	INDOSAT-INP-AP INDOSAT Internet Network Provider	ID

2407:0:0:9::6	411,993	4761	INDOSAT-INP-AP INDOSAT Internet Network Provider	ID

This table is slightly misleading in so far as very large recursive resolvers use resolver "farms" and the queries are managed by a collection of query 'slaves'. We can group these individual resolver IPv6 addresses to their common Origin AS, and look at which networks use resolvers that show this problem with IPv6 Extension Header drops.

The second table (below) now shows the preeminent position of Google's Public DNS service as the most heavily used recursive resolver, and its Extension Header drop issues, as shown in the example at the start of this article, is consistent with its position at the head of the list of networks that have DNS resolvers with this problem.

AS	Hits	% of Total	AS Name	CC
15169	7,952,272	17.3%	GOOGLE - Google Inc.	US
4761	6,521,674	14.2%	INDOSAT-INP-AP INDOSAT Internet Network Provider	ID
55644	4,313,225	9.4%	IDEANET1-IN Idea Cellular Limited	IN
22394	4,217,285	9.2%	CELLCO - Cellco Partnership DBA Verizon Wireless	US
55836	4,179,921	9.1%	RELIANCEJIO-IN Reliance Jio Infocomm Limited	IN
10507	2,939,364	6.4%	SPCS - Sprint Personal Communications Systems	US
5650	2,005,583	4.4%	FRONTIER-FRTR - Frontier Communications of America	US
2516	1,322,228	2.9%	KDDI KDDI CORPORATION	JP
6128	1,275,278	2.8%	CABLE-NET-1 - Cablevision Systems Corp.	US
32934	1,128,751	2.5%	FACEBOOK - Facebook	US
20115	984,165	2.1%	CHARTER-NET-HKY-NC - Charter Communications	US
9498	779,603	1.7%	BBIL-AP BHARTI Airtel Ltd.	IN
20057	438,137	1.0%	ATT-MOBILITY-LLC-AS20057 - AT&T Mobility LLC	US
17813	398,404	0.9%	MTNL-AP Mahanagar Telephone Nigam Ltd.	IN
2527	397,832	0.9%	SO-NET So-net Entertainment Corporation	JP
45458	276,963	0.6%	SBN-AWN-AS-02-AP SBN-ISP/AWN-ISP and SBN-NIX/AWN-NIX	TH
6167	263,583	0.6%	CELLCO-PART - Cellco Partnership DBA Verizon Wireless	US
8708	255,958	0.6%	RCS-RDS 73-75 Dr. Staicovici	RO
38091	255,930	0.6%	HELLONET-AS-KR CJ-HELLOVISION	KR
18101	168,164	0.4%	Reliance Communications DAKC MUMBAI	IN

What's the Problem Here?

IPv6 Extension Headers require that any transport protocol-sensitive functions in network switches need to unravel the packet header's extension header chain. This takes a variable number of cycles for the device, and furthermore requires that the switch should recognise all the extension headers encountered on the header chain. This is an anathema to a switch in so far as it entails a variable amount of time to process. And passing through extension headers that the switch does not understand or not prepared to check is a security risk.

It's easier to drop all packets with extension headers!

Which is what a lot of deployed equipment evidently does.

What can we do about it?

It's easy to say "Well, we should just fix all this errant equipment" but it may be far more challenging to actually do so.

There is a cost in discovering which parts of the inventory of network equipment have this behaviour, and a cost in obtaining and deploying replacement equipment that corrects this problem. Undeniably, for as long as we are operating a dual stack network and for as long as services can revert to using IPv4 when IPv6 fails, then the case to spend this money is not exactly solid. Dual stack networks avoid showing any evidence of the issue because IPv4 simply heals up the problem in a seamless manner.

The result is that there could well be no clear business case to underwrite the costs of correcting this problem in today's networks for as long as the DNS operates within a dual-stack Internet.

But if we can't generate the momentum to actually fix this by modifying all this deployed equipment to pass IPv6 packets with Fragmentation Extension Headers, then maybe we should look a little deeper at the underlying issue in the IPv6 specification?

What is wrong with allowing network equipment to perform forward fragmenting on IPv6 packets in the same manner as IPv4? As far as I can see, there is no intrinsic problem at all with allowing this behaviour as long as we are also prepared to admit the reality that IPSEC in IPv6 is a failure. The upside is that we eliminate another painful issue in today's IPv6 internet, namely that of network filters discarding ICMPv6 Packet Too Big messages. The underlying issue is that these ICMPv6 diagnostic messages are essentially unverified, and it is possible to generate spurious messages of this form and attempt to mount some form of DDOS attack on a host.

However, that still does not address the substance of the problem, namely that Extension Headers appear to present intractable problems to IPv6 network equipment. One approach could be to fold in the Fragmentation Extension header back into the IPv6 header, and use a permanently present set of fragmentation control fields in the IPv6 packet header in a manner that is exactly the same as used in IPv4. Tempting as this sounds superficially, the case for making fundamental changes to the IPv6 specification at this time just cannot withstand more critical scrutiny. IPv6 is not such a software behaviour, as its baked into the firmware and potentially even the hardware of a large proportion of deployed IPv6 equipment. If the prospect of correcting the inventory of equipment that does not handle Extension Headers is daunting, the degree of difficulty of changing the behaviour of all deployed IPv6 equipment to meet some new packet header specification would be on a new level entirely.

Maybe we should bow to the inevitable and recognise that in IPv6 fragmentation is an unfixable problem.

> This is not a new thought, and is best described in recent years in an Internet draft "IPv6 Fragment Header Deprecated" (https://tools.ietf.org/html/draft-bonica-6man-frag-deprecate-01).

What would the Internet environment look like if we could not perform packet fragmentation at the IP level?

QUIC is an illustration of one approach to this problem. In QUIC the maximum packet size is set to 1,350 octets, and fragmentation is no longer exposed as an IP-layer behaviour, but instead the task of payload segmentation and reassembly is an application task inside the QUIC protocol. Logically it appears that the task of payload quantisation into packets has been moved up the protocol stack, and is no longer part of IP and no longer part of the end-to-end protocol. In the QUIC architecture it appears that packet fragmentation is managed as a session level task, sitting above the common UDP substrate.

What that means is that it could be that shifting the DNS to perform its queries over QUIC could help us to envisage a viable all-IPv6 DNS. It's not the only answer, and we could contemplate DNS over

TCP, DNS over secure sockets using TLS over TCP, or even DNS over HTTP or HTTPs. Or, like QUIC we might device some new DNS session level framing protocol and eschew IP level fragmentation.

However, one conclusion looks starkly clear to me from these results. We can't just assume that the DNS as we know it today will just work in an all IPv6 future Internet. We must make some changes in some parts of the protocol design to get around this current widespread problem of IPv6 Extension Header packet loss in the DNS, assuming that we want to have a DNS at all in this all-IPv6 future Internet.

Author

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

www.potaroo.net

Disclaimer

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