The ISP Column
A column on things Internet

Geoff Huston
August 2017

IPv6 Fragmentation Extension Headers, Part 2

The first part of this article looked at what happens when an authoritative DNS server delivers fragmented UDP responses to DNS resolvers using IPv6.

The result from this experiment was that:

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

As concerning as this high loss rate may be, it is not a complete picture of the brokenness of IPv6 Fragmentation Extension Headers in today's IPv6 Internet. This number only refers to DNS traffic, when fragmented UDP responses are sent from a DNS server to "visible" DNS resolvers. It would be useful to understand the larger picture of IPv6 Extension Header drop rate. What would be interesting to measure is the packet drop rate when sending fragmented packets to IPv6 end hosts.

The measurements reported in RFC7872 pointed to drop rates of approximately 30% when sending fragmented packets towards the Alexa top 1M web servers. However, that's moving packets in the opposite direction to that we are interested in. We would like to send fragmented IPv6 packets in the opposite direction, from a server towards end clients.

Constructing the Measurement Environment

Here we are using online Ad framework to enrol end hosts to conduct the experiment. This means that we have a limited repertoire of techniques that we can use at the end host, namely only those techniques we can incorporate into a scripted retrieval of a web object. The only end-to-end component of this experiment are the HTTP(S) sessions used for retrieval of the web objects, so we need to fragment the TCP packets from the server towards the end host in order to undertake this measurement. We would also like to perform this fragmentation without requiring a customised platform, so a solution using raw socket interfaces would be preferred. The approach used here was to set up a front end unit to the web server, and have this front end perform packet fragmentation on outbound packets as required.

To allow the greatest level of flexibility, this front end was programmed as a IPv6 NAT. Incoming IPv6 packets addressed to TCP port 80 or port 443 of the front end had their IPv6 and TCP headers rewritten as they were passed towards the back end web serve. The packet's source address became the IPv6 address of the front end server, and the destination address was that of the back end web server. The a locally generated port number used as the source port. This port number is also used as the lookup in a table of active connections, so that packets received from the back end addressed to this NAT can have their addresses and port values translated back into packets that are to be destined to the original remote endpoint.

In addition to this NAT function, the front end performs one further task. All TCP packets passed across the unit from the back end to the Internet that contain a TCP payload larger than 15 octets that are fragmented.

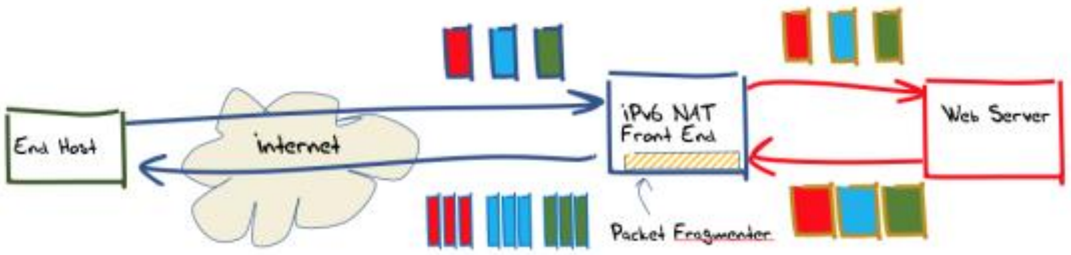

Figure 1 – Experiment Configuration

Unfortunately, there are many aspects of today's platforms that don't make this easy. We can't use a regular TCP socket connection on this front end, as we are relying on a packet level interface to perform the necessary processing of the packet headers.

The *Berkley Packet Filter* (BPF) drivers and the associated *libpcap* routines allow us the necessary flexibility to pull in all incoming packets into the front end process at this "raw" packet level, but it is not quite as easy as it may sound. Many operating systems respond to incoming TCP packets that are addressed to port numbers that have no associated active listener with a TCP reset (RST) packet. This has to be turned off. Also, many physical interface cards are now "smart" interfaces, and rather than sending to the BPF driver exactly the packets as received on the wire, they may join a number of successive packets together and present a synthetic TCP packet that is the join of these successive packets. This is not always possible to turn off, unfortunately. In order to ensure that TLS functions correctly, the DNS name of the service resolves to the IPv6 address of the front end, while the web server at the back end is loaded with the associated domain name certificate.

Once the spurious TCP resets and the possibility of gratuitous TCP segment joins are accommodated in the code we can now move on to the experiment itself.

The IPv6 specification requires that a conformant IP network path be capable of passing an IPv6 packet of up to 1,280 bytes without requiring packet fragmentation. What it fails to specify is the minimum fragmented packet size that an end host can receive. It appears that we can fragment almost any packet, irrespective of its size, and that implies we can fragment small packets as well as large ones. Conveniently, over at the receiver, the TCP stack will be unaware of any packet fragmentation that may have occurred. Packet fragmentation and reassembly is an IP layer function, and TCP is a byte streaming protocol. This means that our NAT unit can perform both packet fragmentation and TCP re-sequencing as required and the packet transforms applied by this unit will be invisible to the remote TCP process.

The function of this NAT unit can be described in a small set of rules:

- For TCP packets received from the Internet, if a translation table exists that maps the triplet of the source address, source port and destination port in the packet to a local port value, then replace the source address of the packet with the local host source address, replace the destination address with that of the back end, replace the source port with the local port address and send the packet out to the back end. If no translation table exists, and the packet contains a TCP SYN flag, take the oldest translation table entry and re-use the local port value. Otherwise drop the packet.

- For TCP packets received from the back end, the processing is similar. The source and destination port is used to look up the translation table. If a translation table entry is found, then the packet's destination address and destination port is replaced with those contained in this entry, and the source address is replaced with the local address. If the packet length is greater than the interface MTU then the received TCP packet is broken into a number of outbound TCP packets, each with a size less than or equal to the interface MTU setting. If an outbound TCP packet has a payload length is greater than 8, then the packet is fragmented as well. The sequence of fragmented packets is then sent.

The subsequent data analysis phase unit can detect if the end host has received and successfully reassembled the set of fragments by looking at the front end's packet capture log. Where an incoming TCP ACK number has an ACK sequence number that encompasses the sending sequence number of outbound fragments within the same TCP session, then we have evidence that the remote end has successfully reassembled the fragmented packet.

How "real" is this Experiment?

Before looking at the results, it may be useful to ask whether this experiment represents a "real" scenario that is commonly encountered on the Internet.

It's certainly the case that in TCP over IPv6 we do not expect to see packet fragmentation.

A TCP sender should ensure that all outbound TCP segments fit within the local interface MSS size, so in the absence of network path MTU issues a sender should not be fragmenting outbound TCP packets before sending them.

What about the case where the path MTU is smaller than the local interface MTU? When a packet encounters a network path next hop where the packet is larger than the next hop MTU, then the IPv6 router constructs an ICMPv6 Packet Too Big message, noting the size of the next hop, and also including the original packet headers as the payload of this ICMPv6 message. It sends this ICMPv6 diagnostic message back to the original sender, and discards the original packet. When a sending host receives this ICMPv6 message it also has the TCP packet header. This information can be used to find the TCP control entry for this session, and the outbound MSS value of this TCP session can be updated with the new value. In addition to the updated size information, the TCP header in the ICMPv6 message payload also contains the sequence number of the lost packet. The sending TCP process can interpret the ICMPv6 message as an implicit NACK of the lost data, and resend the discarded data, using the updated MSS size. Again, no packet fragmentation is required.

All this sounds like a blatant case of "layer violation" and we should call in the protocol police. But before you do so, maybe you should think about the hypothetical situation where the host did not pass the Packet Too Big message to the TCP control block.

This is analogous to the case where the ICMPv6 Packet Too Big message is not passed to the host at all, where, for example, some unhelpful piece of network filtering middleware is filtering out all ICMPv6 messages.

In this case, the sending TCP session has sent a TCP segment and is waiting to receive an ACK. The receiver will not get this packet, so it cannot ACK it. The sender might have a retransmission timer and it might try to resend the offending large packet, but that too will get lost, so it will never get the ACK.

This results in a wedged TCP state, or a Path MTU Black Hole condition.

Hiding ICMPv6 Packet Too Big messages from the TCP controller, either because of local processing rules within the host or because some network element has decided to drop them, is invariably harmful!

In that sense, we have constructed a somewhat “unreal” experiment, and we should not expect to see applications that critically depend on the correct working of packet fragmentation in TCP experiencing the same network conditioned as those we’ve set up here.

On the other hand, fragmentation is an IP function, not a function performed by an end-to-end transport protocol, and the question of whether a host can receive a fragmented UDP packet is essentially the same question as whether a host can receive a fragmented TCP packet, at least from the perspective of the host itself. In both cases the real question is whether the IPv6 process on the host can receive fragmented IPv6 packets.

So while the experiment itself uses conditions that are essentially an artifice, the result, namely the extent to which IPv6 Extension Header drop occurs when passing fragmented IPv6 packets towards end hosts, is nevertheless a useful and informative result.

Results

Over the period from the 11-22 August 2017, this experiment presented fragmented TCP packets to 1,702,949 unique IPv6 addresses. The results are summarized in Table 1.

	Count	% of Total	% of Frags
IPv6 Addresses	1,720,949		
Did not Complete TCP Handshake	42,406	2.46%	
Did not proceed with HTTP(s)	2,645	0.15%	
Sent Fragmented TCP Packets	1,675,898	97.38%	
Acknowledge Fragmented TCP Packets	1,324,834	76.96%	79.03%
Failed to Acknowledge Fragmented TCP Packets	351,514	**20.43%**	**20.97%**

Table 1- Results of Fragmentation Test

Compared to the earlier DNS packet fragmentation result, namely that some 37% of endpoints who used IPv6-capable DNS resolvers used resolvers that were incapable of receiving IPv6 Fragmentation Extension Headers, the overall failure rate observed here of some 20% looks somewhat better.

However, what it does indicate is that some one fifth of IPv6-capable endpoints are unable to receive a fragmented IPv6 packet. In TCP this may not be a major issue, but for UDP-based applications, and the DNS sites first and foremost as a UDP application where large packet responses are common, having one fifth of the end user population being incapable of receiving fragmented large responses over IPv6 is indeed a serious problem.

Of course, whenever you talk about an “address” in IPv6 it is often hard to map such an address into a unique end point. Many end points use IPv6 privacy addresses, and, over the 11 day period of this experiment a single end point may have been seen using a number of /128 addresses.

It may be a little coarse, but what if we look instead at the number of /64 unique prefixes seen by this experiment?

	Count	% of Total	% of Frags
IPv6 /64 prefixes	351,122		
Did not Complete TCP Handshake	10,615	3.02%	
Did not proceed with HTTP(s)	501	0.14%	
Sent Fragmented TCP Packets	340,006	96.83%	
Acknowledge Fragmented TCP Packets	258,247	73.62%	76.02%
Failed to Acknowledge Fragmented TCP Packets	81,527	**23.22%**	**23.98%**

Table 2- Results of Fragmentation Test per unique /64 prefix

This does make that much of a change to the observation. The failure rate rises to some 24% of the /64 prefixes that are sent fragmented packets.

The next question is whether this failure behaviour is even spread across the network, or whether there are higher rates seen in certain networks than others. To generate this data we filtered out the Teredo and 6to4 prefixes where the end user is behind a tunnel, and looked at the remainder.

But before we do let's look briefly at Teredo and 6to4 themselves, as in the IPv6 Internet these two auto-tunnelled IPv6 bridging technologies just don't seem to want to die!

	Teredo	%	**6to4**	%
IPv6 prefixes	59,923		27,246	
Did not Complete TCP Handshake	5,944	9.9%	2,785	9.9%
Did not proceed with HTTP(s)	199	0.3%	77	0.3%
Sent Fragmented TCP Packets	53,780	89.7%	24,384	89.5%
Acknowledge Fragmented TCP Packets	263	0.4%	1,486	5.5%
Failed to Acknowledge Fragmented TCP Packets	53,517	**89.3%**	22,898	**84.0%**

Table 3- Results of Fragmentation Test for Teredo and 6to4 prefixes

Both of these auto-tunnelling services are atrocious in this respect! 10% of end points cannot even complete a TCP handshake. Of those that do, almost no Teredo end points can handle IPv6 fragmentation, and the 6to4 failure rate is not much better. Having no IPv6 at all is far better than having such a terrible service, and I can think of few better justifications for turning off the remaining Teredo and 6to4 gateways than these figures! What is even more depressing that these two auto-tunnelling technologies represent one quarter of the count of unique /64 prefixes seen in this experiment.

The remaining 263,953 /64 prefixes are advertised from just 321 networks (using BGP's originating AS to associate a prefix with a network). In 186 cases we saw only a single sample point from the network, and if we rank these originating AS's by the unique /64 sample rate, across the 11 days of the experiment we only 25 originating AS's had 28 or more samples. This is partially due to the ad network placement algorithm, and partially due to the relatively small number of networks which have significant levels of IPv6 deployment.

In any case, these 25 largest IPv6 networks in terms of sample count are shown in Table 4 below.

Rank	**AS**	**Samples**	**Failure**	**Fail Rate**	**AS Name**
1	55836	246,302	3,461	**1.40%**	Reliance Jio, IN
2	55644	4,680	35	**0.70%**	Idea Cellular, IN

3	45271	2,769	30	**1.10%**	ICLNET-AS-AP, Idea Cellular, IN
4	7922	1,593	417	**26.20%**	Comcast Cable, US
5	4818	1,165	25	**2.10%**	DiGi Telecommunications, MY
6	18101	696	29	**4.20%**	Reliance Communications, IN
7	9644	683	0	**0.00%**	SKTELECOM-NET-AS, SK, Telecom, KR
8	8708	340	183	**53.80%**	RCS-RDS, RO
9	29247	305	7	**2.30%**	Cosmote, Mobile, GR
10	55441	215	1	**0.50%**	TATA-DOCOMO-AS-AP, IN
11	21928	168	1	**0.60%**	T-Mobile, US
12	15169	161	161	**100.00%**	Google, US
13	20057	99	4	**4.00%**	AT&T Mobility, US
14	7018	94	15	**16.00%**	AT&T Internet Services, US
15	19782	76	70	**92.10%**	Indiana, University, US
16	25820	66	0	**0.00%**	IT7 Networks, CA
17	22394	60	47	**78.30%**	Cellco, Verizon Wireless, US
18	13124	58	46	**79.30%**	IBGC, BG
19	16276	57	6	**10.50%**	OVH, FR
20	38466	55	3	**5.50%**	U, Mobile, MY
21	18881	54	19	**35.20%**	TELEFONICA, BR
22	1257	45	2	**4.40%**	TELE2, SE
23	24940	45	19	**42.20%**	HETZNER-AS, DE
24	109	30	17	**56.70%**	Cisco Systems, US
25	23910	28	19	**67.90%**	Next Generation Internet, CERNET2, CN

Table 4 - Results of Fragmentation Test by Origin AS

There is a considerable level of variation in the extent to which networks support the delivery of IPv6 Fragmentation Extension Headers to hosts. In some cases, it appears that the choice of customer premises equipment, or the configuration of IPv6 firewalls, may be a factor. Where the failure rate is very high it would point to the drop point being part of the behaviour of the provider network rather than the behaviour of the customer premises equipment.

Conclusion

Whatever the reasons, the conclusion is here is unavoidable: IPv6 fragmentation is just not a viable component of the IPv6 Internet.

We need to adjust our protocols to avoid fragmentation.

For TCP, this should not be a major issue. Of course, this assertion relies on ICMPv6 Packet Too Big messages getting back to the sender's TCP process, but that is a major topic in its own right, so we won't delve deeper into this right now.

However, for UDP, this should be cause for some major re-thinking of the way the DNS works, as the combination of DNSSEC, UDP and IPv6 is really not going to work very well. However, it has implications for other UDP-based protocols as well, particularly where the protocol can generate large payloads.

QUIC has taken the pragmatic position of using a maximum packet size of 1,350 octets as a universal base, and does not expect to encounter fragmentation issues given this somewhat conservative choice. If the DNS over IPv6 used a similar ceiling UDP size, and always sent back truncated responses for larger answers we could probably avoid many of the packet loss problems that we encounter today. Of course, the larger use of TCP has its own implications in terms of query processing capacity for DNS resolvers and servers, so there are no free points here.

As I observed in respect to the analysis of IPv6 fragmentation loss in the DNS, maybe we should bow to the inevitable and recognise that, in IPv6, fragmentation is an unfixable problem.

This is not a new thought, and it is best described in recent years in a now-neglected four year old Internet draft "IPv6 Fragment Header Deprecated".

Perhaps this draft was just slightly ahead of its time, but our experience in conducting these measurement experiments indicate that like it or not, our operational IPv6 Internet has effectively deprecated IPv6 Fragment Extension headers.

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