Geoff Huston
May 2016

Fragmenting IPv6

The design of IPv6 represented a relatively conservative evolutionary step of the Internet protocol. Mostly, it's just IPv4 with significantly larger address fields. Mostly, but not completely, as there were some changes. IPv6 changed the boot process to use auto-configuration and multicast to perform functions that were performed by ARP and DHCP in IPv4. IPv6 added a 20-bit Flow Identifier to the packet header. IPv6 replaced IP header options with an optional chain of extension headers. IPv6 also changed the behaviour of packet fragmentation. Which is what we will look at here.

We've looked at IP packet fragmentation earlier (http://www.potaroo.net/ispcol/2016-01/frags.html), so I won't repeat myself here, but that article provides some useful background, so it may be useful to read before embarking on this.

IPv4 did not specify a minimum packet size that had to be passed unfragmented through the network, although by implication it makes a lot of sense to ensure that the IP header itself is not fragmented, so by implication the minimum unfragmented size is 40 bytes as presumably you would require any IP header options to remain with the common IPv4 header that is replicated in each fragment packet. The other limit defined by IPv4 was that a host had to be able to reassemble an IP packet that was up to at least 576 bytes in total.

The change implemented in IPv6 was that the IPv6 specification effectively cemented the IPv4 "DONT FRAGMENT" bit to ON. IPv6 packets could not be fragmented on the fly during transit across the network, so each router could either forward on an IPv6 packet or discard it. An early IPv6 specification, RFC 1883, published in 1995, required that IPv6 be able to pass any IPv6 packet across an IPv6 network that was up to 576 bytes in size without triggering packet fragmentation. This is a consistent change with the IPv4 behaviour, which effectively says that IP packets of up to 576 octets have a high probability of successful delivery, as they will not trigger any structural size limitations, and all hosts had to accept packets up to 576 bytes in size. By removing fragmentation on the fly within the network, the original IPv6 specification consistently translated this to a requirement to be able to pass packet of up to 576 bytes in size across an IPv6 network without triggering fragmentation. This size was altered during the refinement of the protocol specification, and RFC 2460, published in 1998, raised this minimum size from 576 bytes to 1,280 bytes.

This raises two questions: Why set this Don't Fragment bit to always ON? Why use 1,280 bytes as the critical minimum packet size?

Why Set "Don't Fragment"?

This was a topic of considerable debate in the late 1980's in packet networking, and surfaced once more in the design of IPv6 a few years later.

Fragmentation was seen as being highly inefficient. When a fragment is lost there is no ability to signal that just the missing fragment needs to be resent. Instead, the sender is required to resend the entire packet once more. Fragmentation also represents a window of opportunity in terms of exploiting

potential vulnerabilities. Trailing fragments have no upper layer transport protocol header, so firewalls have a problem in determining whether or not the fragment should be admitted. Packet reassembly consumes resources at the destination, so an attacker can generate packet fragments and force the intended victim to reserve for a period reassembly resources to await the remainder of the original packet's fragments that will never arrive.

This and more was written up in a 1987 paper, "Fragmentation Considered Harmful" by Kent and Mogul (http://www.hpl.hp.com/techreports/Compaq-DEC/WRL-87-3.pdf), and the conclusions from the paper are to avoid fragmentation wherever possible.

This paper recommended the use of the Don't Fragment flag as a means of supporting the communications hosts discovering the path MTU. The intended functionality was that a router that could not forward a packet as it was too large for the next hop link would return the leading bytes of the packet together with the value of the MTU of the next hop link back to the packet's sender. The sender could assemble this information and discern the path MTU for each of the destinations that the local sender communicates with.

Why 1,280?

The online digital archives of the discussions at the time appear to relatively incomplete, and I cannot find anything in the way of a design discussion about the selection of this value for IPv6. One view is that if not 576 then surely the next logical choice would be 1,500. The rationale for 1,500 was the predominance of Ethernet as an almost ubiquitous layer 2 media framework for digital transmission systems, and even if they were not Ethernet networks, Ethernet packet framing was extremely common.

It's possible that considerations of encapsulation and the related area of tunnelling come into play here. The use of "shim" packet headers for PPP and MPLS would tend to imply that as a universal constant 1,492 was probably a safe choice, allowing 8 bytes for local shim headers. However, if you admit this form of packet encapsulation, then why not allow for IPv6-in-IPv4 (20 bytes), or even IPv6-in-IPv6 (40 bytes)? If you allow for this possibility, then perhaps it might also make sense to also add a further 8 bytes for a UDP header, or 20 bytes to permit a TCP header. Even in this case of encapsulating IPv6-in-TCP-in-IPv6, the resultant payload packet size is a maximum of 1,440 octets.

It appears that the minimum unfragmented size of 1,280 bytes as specified in RFC 2460 appears to be the result of considering the extremely unlikely case of a 1,500 bytes packet that carries 220 octets of encapsulation headers. Our intuition tends to the view that the Internet supports a 1,500 byte packet almost ubiquitously, and most forms of tunnelling would be happily accommodated by allowing for much less than 220 bytes of tunnel encapsulation headers per packet.

So why was the value of 1,280 chosen? I'm afraid that I simply don't know!

IPv6 Fragmentation Behaviour

How do we cope with variable packet size limits in the IPv6 network? When a router is passed a packet that is too large to be forwarded along the next hop, then the router is supposed to extract the source IPv6 address of the packet and generate an ICMP message addressed to this source address. The ICMP message includes the MTU value of the next hop, and includes as a payload the leading bytes of the original packet.

What should an IPv6 host do when receiving this ICMP message?

RFC1981 proposed that hosts reduced the size of packets: "The node MUST reduce the size of the packets it is sending along the path."

In the context of a TCP session this can be achieved through an interaction between the upper level protocol and the ICMP receiver on the sending host. The ICMP Packet Too Big message should cause the TCP session to drop its estimate of the maximum unfragmented packet size that is supported on the path between this host and the remote end (the "PathMTU") to the value provided in the ICMP message. This way the TCP session should not generate fragmented packets, but dynamically adjust its packetization size to match what is supported on the network path. In other words, for TCP sessions, the preferred behaviour is not to use IP fragmentation as a response, but instead to push this information into the TCP session and adjust the TCP Maximum Segment Size appropriately. When this occurs as par the plan, one should not see fragmented TCP packets at all.

If we won't (or shouldn't) see TCP fragments in IPv6, should we expect to see UDP fragments? As with many questions about UDP behaviour the answer is that "it depends!" What it depends on is the behaviour of the application that is emitting the UDP packets.

Assuming that the UDP application is a stateless application that operates in a simply query/response model, such as a DNS server for example, then the application has no remembered connection state and no buffer of previously sent messages. This implies that the ICMP message is largely useless in any case!

So what value should an IPv6 host use for its local MTU?

If you set it low, such as a setting of 1,280 bytes, then attempts to send larger payloads in UDP will result in fragmentation of the payload.

If you set it higher, and 1,500 bytes is also common, then attempts to send a large payload, such as 1,400 bytes may encounter path constraints and may generate ICMP Packet Too Big ICMP messages. Assuming that the ICMP message makes it all the way back to the sender (which is by no means an assured outcome) then a TCP application can react by adjusting its session MSS. On the other hand, a UDP-based application may be simply unable to react.

This would tend to suggest that a conservative approach is to use 1,280 bytes as a local MTU, as this would minimise, or hopefully eliminate the issues of UDP and ICMP PTB messages. However, relying on packet fragmentation, which is the inevitable consequence of using a small MTU with larger UDP payloads, may not necessarily be a good idea either. A soon-to-be Informational RFC, currently called "draft-ietf-v6ops-ipv6-ehs-in-real-world" by Fernando Gont and colleagues, indicated a packet drop rate of up to 55% when passing IPv6 packets with Fragmentation Extension headers through the network.

Experimenting in the DNS

Given that this issue of packet fragmentation is one that primarily concerns UDP, and the major user of UDP in the Internet today appears to be the DNS, we set up an experiment that tested the ability to pass variously sized IPv6 UDP DNS responses through the network.

We tested in three size ranges for packets: small (below 1,280 bytes), medium (between 1,280 and 1,500 bytes) and large (above 1,500 bytes). The first size is the control point, and we do not expect packets of this size to encounter any network delivery issues. The second size lies between 1280 and 1500 bytes, and we expect to see some form of interaction between packets in this size range and network paths that generate ICMP PTB messages that have MTUs below the particular packet size. The third size is 1700 octets. The sending hosts use a local outbound MTU setting of 1500, so the outbound packet is fragmented at the source into an initial segment and a trailing fragment. Both of these packets have a fragmentation header.

We used a modified DNS name server that ignored EDNS0 UDP buffer sizes, and did not respond to TCP connection requests. The system was configured with a local MTU of 1,500 bytes, and in this case listened exclusively on IPv6. The Linux kernel (running Debian GNU/Linux 8 (Jesse)) we used includes support for ICMPv6 Packet Too Big Messages, and will hold in its FIB a cache of the recent Path MTU values. In the case of outgoing UDP messages, the kernel will perform fragmentation down to 1,280 bytes.

We expected to see a result where all IPv6 resolvers were capable of receiving packets up to 1,280 bytes in length. It's unclear how many resolvers sit behind network paths that do not admit 1500 byte packets, so we are unsure what the drop rate might be for packets that are sized between 1,280 and 1,500 bytes. The interaction is a little more complex here, as while the local server will adjust its local cache of Path MTU in response to received ICMP messages, this will only be effective if the remote end uses the same interface identifier value a subsequent query, and if the remote end performs a re-query within the local cache lifetime.

For packets larger than 1,500 octets, the response will be fragmented at source, and there are three potential reasons why the network will discard the packet. The initial fragment will be 1,500 octets in length, which implies that this leading fragment will encounter the same path MTU issues as the slightly smaller packets. Secondly, firewalls may reject trailing fragments, as there is no transport level port information in the trailing fragment. And finally there is the Extension Header drop issue where other observations report a drop rate of approximately 50% when an IPv6 packet has a fragmentation Extension Header.

We would expect to see that if a resolver is able to follow the glueless delegation path for small packets then we would expect to see some level of packet drop for packets larger than 1280 bytes due to Path MTU mismatch issues where the remote end does not perform re-queries within the cache lifetime. For the packet test larger than 1,500 bytes the would be further compounded by a further level of packet drop due to firewall filtering of trailing fragments, and in addition there is the extension header packet drop problem as noted above.

How can you tell if a resolver has received a response?

The technique we used in this experiment is one of a combination of dynamic generation of DNS labels and the synthesis of "glueless delegation".

When a resolver attempts to resolve a unique terminal name, the "parent" will send a response to indicate that the terminal name lies in a uniquely named delegated zone, and provides the DNS name of the authoritative name server for this delegated zone, but it deliberately omits the conventional inclusion of the IP address values of this name server (the "glue" records that are conventionally loaded into the Additional section of a DNS response). This means that the resolver will have to separately resolve this DNS name to an IP address before it can resume its task to resolve the original name. We have made this name a synthetic unique name so that the resolver cannot use a cached value, and must perform the name resolution task each time.

In our case we have deliberately inflated the DNS response to the query for the address record of the authoritative name server, and the visible confirmation that the DNS resolver has successfully received this inflated response is in the subsequent query to the delegated domain for the original name.

What we observed from this experiment is shown in Table 1.

Size	Count	Always Fetched	Sometime Fetched	Never Fetched
151	5,707	5,603 (98%)	104 (2%)	0
1,159	489	487 (99%)	2 (1%)	0
1,280				
1,400	5,075	4,942 (97%)	95 (2%)	38 (1%)
1,425	480	467 (97%)	3 (1%)	10 (2%)
1,455	481	456 (94%)	3 (1%)	22 (5%)
1,500				
1,700	5,086	3,857 (76%)	65 (1%)	1,164 (23%)

Table 1 – UDP Fetch Rate vs Packet Size

Table 1 shows that the packet loss rates increase as the packet size increases. Of course the reasons for the packet loss rates differ above and below the 1,500 byte size point.

Below 1500 octets we can expect to see Path MTU issues as being the major loss factor for packets, where the packet is larger than a particular link on the path to the destination. This is commonly encountered in IP-in-IP tunnels, although in this case the common IPv6-in-IPv4 case would not impact these three packet sizes, as they all fit below the 1,480 IPv6-in-IPv6 tunnel MTU. When an outbound packet encounters such a link, the router should return a Packet Too Big ICMPv6 message. The distribution of sizes reported in these ICMP messages is shown in Table 2.

Size	PTB	%	Cumulative Count	%
1280	36	8.76%	36	8.76%
1400	5	1.22%	41	9.98%
1408	1	0.24%	42	10.22%
1418	1	0.24%	43	10.46%
1428	2	0.49%	45	10.95%
1432	1	0.24%	46	11.19%
1434	2	0.49%	48	11.68%
1436	1	0.24%	49	11.92%
1445	1	0.24%	50	12.17%
1452	3	0.73%	53	12.90%
1454	9	2.19%	62	15.09%
1456	1	0.24%	63	15.33%
1460	4	0.97%	67	16.30%
1462	2	0.49%	69	16.79%
1472	12	2.92%	81	19.71%
1476	3	0.73%	84	20.44%
1480	283	68.86%	367	89.29%
1492	28	6.81%	395	96.11%
1500	15	3.65%	410	99.76%
1586	1	0.24%	411	100.00%

Table 2 – Distribution of MTU sizes in Packet Too Big messages per /64 prefix

While the most common constrained MTU value is 1,480 bytes, corresponding to a simple IPv6-in-IPv4 tunnel arrangement, there are also a range of smaller MTU values spread across 1,400 to 1,480 bytes. Some appear to be related to IPv6-in-IPv6 tunnels (1460 bytes), or the use of UDP in IPv6 and IPv6 as the outer tunnel wrapper (1,472 and 1,452 bytes). Other values are not as readily explained, and the peak at 1,400 bytes appears to be as contrived as 1,280 bytes! The 1,500 byte MTU is likely to be a known problem with a particular vendor's firewall implementation, which reassembles fragments to

determine whether to admit the packet, but then does not perform any subsequent fragmentation, so the reassembled packet is too large for the next hop! The 1,586 MTU is also likely to be a similar form of implementation bug. Approximately 90% of the observed IPv6 /64 prefixes appear to be reachable using a 1,500 byte Path MTU.

Above 1,500 bytes in size there are three potential problems that a UDP packet may encounter, as noted above, namely the initial 1,500 byte fragment may trigger MTU mismatch problems, the trailing fragment may encounter filtering firewall rules that will cause the destination to timeout on packet reassembly, or both the initial and trailing fragments will be dropped in transit because of the presence of the Fragmentation Extension Header. The first two cases are meant to generate ICMPv6 messages back to the sender, while the Extension Header discard is a silent discard.

We saw consistent problems in sending the 1,700 byte UDP packet to 1,164 distinct /64 IPv6 subnets. Some 331 of these failing subnets generated ICMPv6 messages indicating a timeout in fragment reassembly, and the payload indicated that the receiver had received the initial fragment and was waiting for the trailing fragment. This is consistent with a front end firewall that filters trailing fragments. A further 61 failing subnets generated ICMP Packet Too Big messages, and evidently did not perform re-queries from the same /128 IPv6 address within the local cache lifetime window. It is unclear what is going on in the remaining 772 cases. It may be that there are additional fragmentation reassembly or path MTU issues, but that the diagnostic ICMPv6 messages are either not being generated, or themselves are being filtered in the network. It is also possible that we may be encountering the same behaviour reported by Gont et al. where parts of the IPv6 network appear to silently drop IPv6 packets that use an Extension Header, including the Fragmentation Extension Header. If this is the case, then from the limited set of IPv6 addresses that are used for DNS resolvers, the extent of this Extended Header packet drop is somewhere around 15%, a considerably lower value that the 50% level reported by Gont et al.

Conclusions

Do Big Packets work in IPv6?

The overall picture for IPv6, UDP and large packets does not appear, in the first instance, to be all that reassuring.

The 5% drop rate for 1,455 byte IPv6 packets is somewhat higher than one would expect from a server that appropriately responds to ICMPv6 Packet Too Big messages. To what extent this is due to the ICMP messages not being generated, or being filtered in flight, and to what extent this is due to the application not performing re-queries is an open question, but there is certainly a problem here.

The larger issue is a 23% drop rate for fragmented IPv6 packets. Because the network cannot forward-fragment the initial fragment, the initial fragment may encounter Path MTU issues and we can anticipate a certain level of packet drop from this. Secondly, the presence of filtering firewalls that discard tailing fragments is a problem here (as it is in IPv4). And there is the additional issue in IPv6 that there are reports of routers that perform a silent discard of IPv6 packets that contain an Extension Header, including the Fragmentation Extension Header. The combination of these three factors points to a consistent loss rate of one in four packets in the IPv6 network. So, no, big packets don't work all that well in IPv6.

But is this really as bad as it sounds?

If the aim of the exercise is for an application to get the date from the server to the client, then it's not just the IP level behaviour that we must take into account, but also the application level behaviour.

In this experiment we deliberately turned off the DNS level EDNS0 UDP buffer size behaviour in the server. We sent the same large response to all clients irrespective of the EDNS0 settings they tried to

pass to our server in their query. Normally, and quite correctly, this would not happen. If the client resolver did not include an EDNS0 buffer size we should've constrained the response to no more than 512 bytes, and in the case of these larger responses we should've sent a truncated DNS response. This would trigger the DNS resolver to re-attempt the query using TCP. If the client resolver provided an EDNS0 UDP buffer size we should've honoured that maximum size, and truncated the response if it could not fit into the specified buffer size. Again, truncation would trigger the client resolver to re-attempt the query using TCP.

The DNS client resolver also has a simple test to distinguish between path MTU issues and unresponsive servers. If it does not receive a response to its original query, it will repeat the query using an EDNS0 UDP buffer size of 512 bytes. If this query does not elicit a response then it can conclude that it is an unresponsive server, as a 512 byte response should not encounter Path MTU problems.

By ignoring EDNS0 and not truncating any responses in this experiment we deliberately set out to expose the underlying IP level failure rate and did not allow the DNS to use its conventional responses that would've mitigated this issue to a significant extent.

The observation is that the DNS already uses large packets, and we have no solid evidence that supports large scale structural damage when large DNS responses are used. For example, a DNSSEC-validating resolver will at some point need to validate a name within the .org domain, and to do so it will need to query for the DNSKEY RRset of .org. This domain is served by dual stack name servers, and they will serve a 1,625 byte response to this query. In the case of an IPv4 UDP query the response is a 1,500 initial fragment and a 173 byte trailing fragment. In the case of IPv6 the servers provide a 1,280 byte initial fragment and a 449 byte trailing fragment. As far as I am aware, there are no complaints of widescale problems in resolving .org domain names by DNSSEC validating resolvers.

While there is an underlying issue with the viability of large IP packets and fragmentation in the IPv6 network, as long as the application is capable of resolving the distinction between network packet loss and server availability, then this need not be a significant impediment.

MTU choice: 1,280 vs 1500?

The second question is whether it is preferable to use a 1,280 byte local MTU for IPv6, or use a 1,500 byte MTU (assuming that you are connecting to an underlying local network that supports a 1,500 byte MTU, of course).

A lower local MTU setting will avoid Path MTU issues, which will be particularly useful for TCP, and this will also ensure that TCP packets are not fragmented. MTU issues. As long as all ICMPv6 messages can reach the sender, and TCP can adjust its session MSS accordingly then TCP need not be adversely impacted by a higher MTU. But practical experience suggests that ICMP filtering is widespread and larger MTUs in TCP expose the risk of wedging the TCP session in an irretrievable manner. So a smaller MTU for TCP in IPv6 will avoid the relatively dramatic loss rates associated with IPv6 and fragmentation, while still allowing TCP to perform path MTU discovery if it so desires. So for TCP and IPv6 it makes sense to start the max segment size low and potentially probe upward if desired.

But this lower MTU setting, if applied to UDP over IPv6, will cause fragmented UDP packets to start at the lower threshold size, and the not inconsiderable fragment filtering and Fragmentation Extension Header loss factors then come into play. This, in turn, may require an increased level of application level intervention and in the case of DNS may involve higher query levels and a higher TCP query rate as a consequence. For UDP the larger MTU setting of 1,500 bytes allows packets in the range of 1,280 to 1,500 bytes to be sent without gratuitous fragmentation. As we saw in this experiment there will still be remote systems using smaller UDP MTU sizes, and the flow of ICMP Packet Too Big messages will still trigger packet fragmentation, but rather than occurring for all packets over the lower threshold, this

fragmentation will occur only if the packet is larger than the local interface MTU or in response to an incoming ICMP message.

So for TCP over IPv6 an MTU choice of 1,280, or an initial MSS default value of 1,220, is a very rational design choice. For UDP over IPv6 it's not so straightforward, but it appears that there are marginal advantages with using the larger MTU size, as it avoids gratuitous fragmentation and the associated fragmentation loss issues.

If you had to pick one, and only one, setting then I would tend towards optimising the system for TCP over UDP.

Can we set different MTU values for TCP and UDP? This does not appear to be a common feature of most operating systems, however Apple's MacOSX platform, and the related FreeBSD operating system both have a very interesting system configuration control (sysctl) parameter of net.inet.tcp.v6mssdflt, which appears to be a default local setting for the TCP MSS size for IPv6. In theory this setting could be used to implement these differing TCP and UDP MTU settings for IPv6. But perhaps a story about my experiences in playing with these system settings is best left to another time.

A Postscript

Above, I posed the question on whether it is preferable to use a 1,280 byte local MTU for IPv6, or use a 1,500 byte MTU (assuming that you are connecting to an underlying local network that can supports a 1,500 byte MTU, of course).

The loss rate for packets of 1,400 bytes in size was 1%, which appears to be due to Path MTU issues for those end sites that site behind MTU-constrained tunnels. The far higher loss rate for 1,700 byte packets points to a set of problems with fragmented packets in IPv6 networks, including the dropping of trailing fragments and the silent dropping of packets that have Extension Headers.

If the local host uses a lower MTU value, such as 1280, then it will cause fragmented UDP packets to start at the lower MTU size, and the not inconsiderable fragment filtering and Fragmentation Extension Header loss factors then come into play for our 1,400 byte test packet.

This has now been measured in an experiment whose conditions are the same as the previous experiment with the single change that the MTU size has been dropped from 1,500 bytes to 1,280 bytes. The loss rate for 1,400 byte UDP packets (which are now all fragmented) has risen from 1% to 20%, as shown in the following table.

Size	Count	Always Fetched	Sometime Fetched	Never Fetched
151	4,211	4,067 (97%)	144 (3%)	0
1,400	4,112	3,203 (78%)	75 (2%)	834 (20%)
1,700	4,100	2,954 (72%)	84 (2%)	1,062 (26%)

For the 1,400 byte packet, of these 834 failing IPv6 /64's, 288 (35%) of these V6 prefixes generated a Fragmentation Reassembly ICMP error, indicating some form of packet firewall filtering system that is dropping trailing fragments. The remainder of the drops were silent drops.

The message seems pretty clear that for UDP in IPv6 it's best for a sender to use a large MTU if they can, in order to avoid gratuitous fragmentation-caused packet drop.

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