Geoff Huston
January 2016

Fragmentation

One of the more difficult design exercises in packet switched network architectures is that of the design of packet fragmentation. In this article I'd like to examine IP packet fragmentation in detail and look at the design choices made by IP version 4, and then compare that with the design choices made by IP version 6.

Packet-switched networks dispensed with a constant time base, which, in turn allowed individual packets to be sized according to the needs of the application as well as the needs of the network. Smaller packets have a higher packet header to payload ratio, and are consequently less efficient in data carriage. On the other hand, within a packet switching system the smaller packet can be dispatched faster, reducing the level of *head-of-line blocking* in the internal queues within a packet switch and potentially reducing network-imposed jitter as a result. 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. But larger packets take longer to be dispatched and this can be a cause of increased jitter.

Various network designs adopted various parameters for packet size. Ethernet, standardized in the mid-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 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 to 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.

IPv4 and Packet Fragmentation

In designing a network protocol that is intended to operate over a wide variety of substrate carriage networks, the designers of IP could not rely on a single packet size for all transmissions. Instead the IP designers of the day provided a packet length field in the IP version 4 header [RFC791]. 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. So IP itself supports a variable size packet format. But which packet size should an implementation use?

The tempting answer is to use the maximum size permitted by the local device's network interface, with the caveat that an application may nominate the explicit use of smaller-sized packets. But there is a complication here. The Internet was designed as an "inter-network" network protocol, allowing an IP packet to undertake an end-to-end journey from source to destination across a number of difference networks. For example, consider a host connected to a FDDI network, which is connected to an Ethernet network. The FDDI-connected host may elect to send a 4,478 octet packet, which will fit into

a FDDI network, but the packet switch that is attempting to pass the packet into the Ethernet network will be unable to do so because it is too large.

The solution adopted by IPv4 was the use of *forward fragmentation*. The basic approach is that any IP router that is unable to forward an IP packet into the next network because the packet is too large for this network may split the packet into a set of smaller IP fragments, and forward each of these fragments. The fragments continue along the network path as autonomous packets, and the addressed destination host is responsible to re-assemble these fragments back into the original IP packet.

The behaviour is managed by a 32-bit field in the IPv4 header, which is subdivided into 3 sub fields (Figure 1).

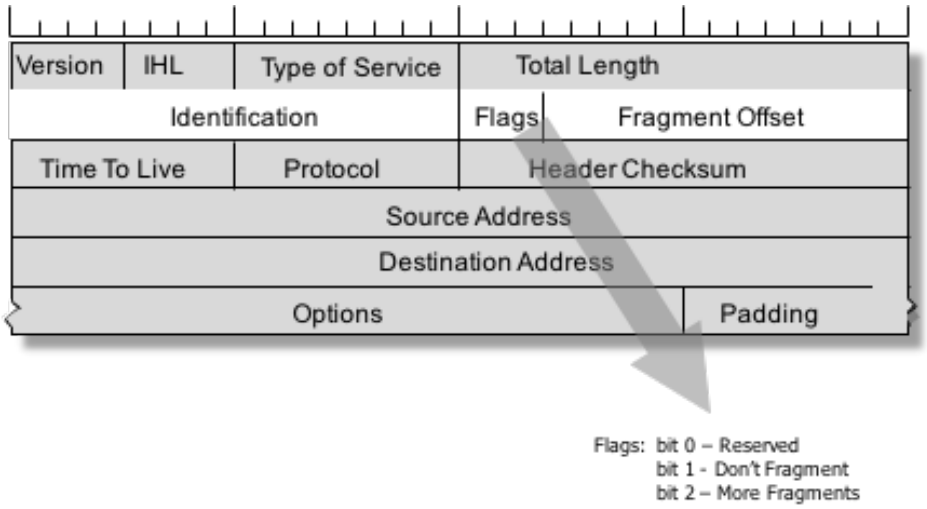

Figure 1 – IPv4 Packet Header Fragmentation Fields

The first sub-field is a 16-bit packet identifier, which allows fragments that share a common packet identifier value to be identified as fragments of the same original packet.

The second sub-field is a 3-bit vector of flags. The first bit is unused. The second is the *Don't Fragment* flag. If this flag is set the packet cannot be fragmented, and must be discarded when it cannot be forwarded. The third bit is the *More Fragments* field, and is set for all fragments bar the final fragment.

The third sub-field is the fragmentation offset value, that is the offset of this fragment from the start of the original packet's IP payload, measured in octawords (64 bit units).

For example, a router attempting to pass a 1320 octet IP packet into a network whose maximum packet size is 532 octets would need split the IP packet into three parts. The first packet would have a fragmentation offset of 0, and the *More Fragments* bit set. The total length would be 532 octets, and the IP payload would be 512 octets, making a total of 532 octets for the packet. The second packet would have a fragmentation offset value of 64, the *More Fragments* bit set, total length of 532 and an IP payload of 512 octets, making a total of 532 octets for the packet. The third packet would have a fragmentation offset value of 128, the *More Fragments* bit clear, total length of 296 and an IP payload of 276 octets, making a total of 296 octets for the packet. (Figure 2)

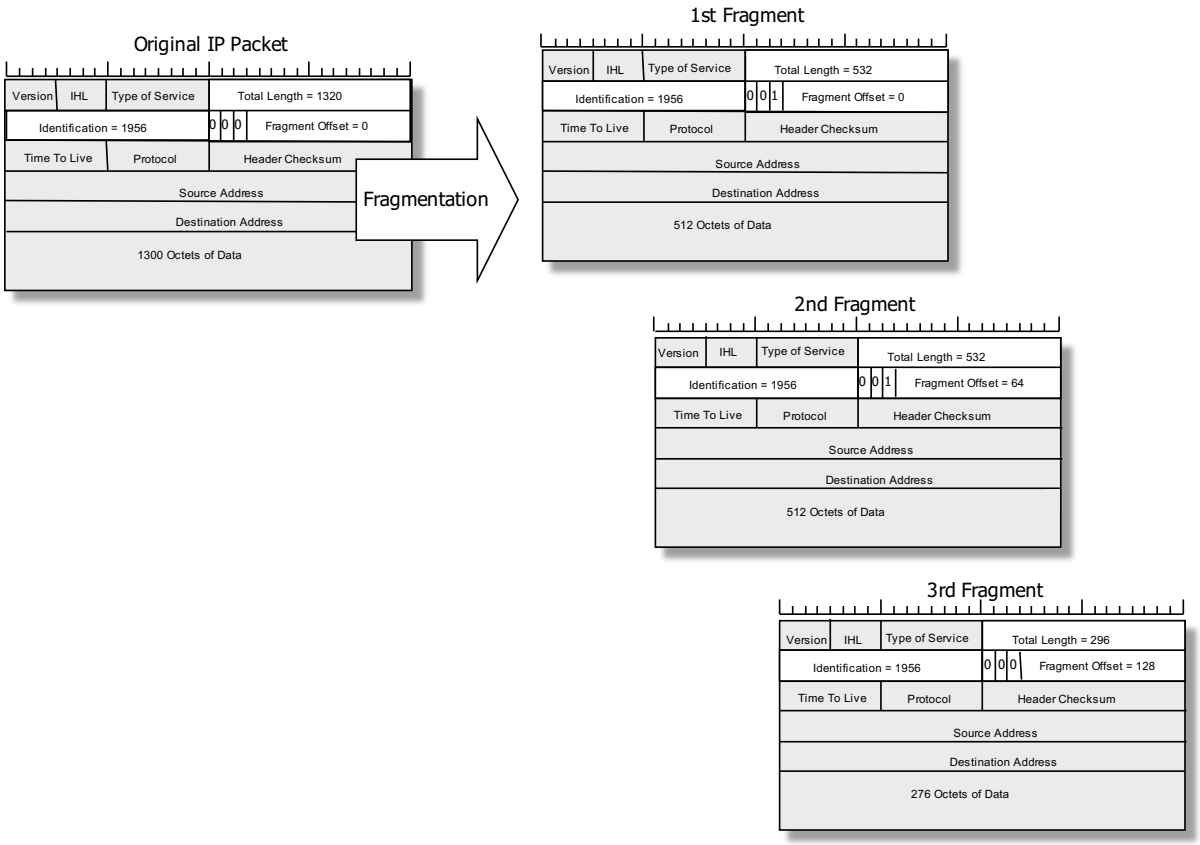

Figure 2 – Example of IPv4 Packet Fragmentation

The advantage of this approach is that as long as it is permissible to fragment the IP packet, all packet flows are "forward". By this it is meant that the sending host is unaware that packet fragmentation is occurring, and all the IP fragment packets continue to head towards the original destination where they are reassembled. Another advantage is that while the router performing the fragmentation has to expend resources to generate the packet fragments, the ensuing routers on the path to the destination have no additional processing overhead, assuming that they do not need to further fragment these IP fragments. Fragments can be delivered in any order, so the fragments may be passed along parallel paths to the destination.

To complete the IPv4 story its necessary to describe the IPv4 behaviour when the *Don't Fragment* bit is set. The router that is attempting to fragment such a packet is forced to discard it. Under these circumstances the router is expected to generate a ICMP Unreachable error (type 3, code 4), and in later versions of the IP specification it was expected to add the MTU of the next hop network into the ICMP packet. The original sender would react to receiving such an ICMP message by changing its local maximum packet size associated with that particular destination address, and thus it would 'learn' a viable packet size for the path between the source and destination.

Evaluating IPv4 Fragmentation

A case has been made that IP's approach to fragmentation contributed to IP's success. This design allowed transport protocols to operate without consideration of the exact nature of the underlying transmission networks, and avoid additional protocol overhead in negotiating an optimal packet size for each transaction. Large UDP packets could be transmitted and fragmented on the fly as required without requiring any form of packet size discovery. This approach allowed IP to be used on a wide variety of substrate networks without requiring extensive tailoring.

But it wasn't all good news.

Cracks in IP's fragmentation story were described in a 1987 paper by Kent and Mogul, 'Fragmentation Considered Harmful." [Kent 87]

TCP has always attempted to avoid IP fragmentation. The initial opening handshake of TCP exchanges the local Maximum Segment Sizes, and the sender will not send a TCP segment larger than that notified by the remote end at the start of the TCP session. The reason for TCP attempting to avoid fragmentation was that fragmentation was inefficient under conditions of packet loss in a TCP environment. Lost fragments can only be repaired by resending the entire packet, including resending all those fragments that were successfully transmitted in the first place. TCP will perform a data repair more efficiently if it were to limit its packet size to one that did not entail packet fragmentation.

This form of fragmentation also posed vulnerabilities for hosts. For example, an attacker could send a stream of fragments with a close to maximally sized fragment offset value, and random packet identifier values. If the receiving host believed that the fragments represented genuine incoming packets, then a credulous implementation might generate a reassembly buffer for each received fragment which may represent a memory buffer starvation attack. It is also possible, either through malicious attack or by poor network operation, that fragments may overlap or overrun, and the task of reassembly requires care and and attention in implementation of fragment reassembly.

Lost fragments represent a slightly more involved problem than lost packets. The receiver has a packet reassembly timer upon the receipt of the first fragment, and will continue to hold this reassembly state for the reassembly time. The reassembly timer is a factor in the maximal count of packets in flight, as the packet identifier cannot be recycled within period defined by the sender-received path delay plus the receiver's reassembly timer. For higher delay high capacity network paths this limit of 65,535 packets in flight can be a potential performance bottleneck [RFC 4963].

Fragmentation also consumes router processing time, forcing the processing of over-sized packets from a highly optimised fast path into a processor queue.

And then there is the middleware problem. Filters and firewalls perform their function by applying a set of policy rules to the packet stream. But these rules typically require the presence of the transport layer header. How can a firewall handle a fragment? One option is to pass all trailing fragments through without inspection, but this exposes the internal systems to potential attack [RFC 1858]. Another option is to have the firewall rebuild the original packet, apply the filter rules, and then refragment the packet and forward it on if the packet is accepted by the filter rules. However, by doing this the firewall is now exposed to various forms of memory starvation attack. NATs that use the transport level port addresses as part of its binding table have a similar problem with trailing fragments. The conservative approach is for the NAT to reassemble the IP packet at the NAT, apply the NAT address transform and then pass the pack onward, fragmenting as required.

IPv6 and Fragmentation

When it came time to think about the design of what was to become IPv6 the *forward fragmentation* approach was considered to be a liability, and while it was not possible to completely ditch IP packet fragmentation in IPv6, there was a strong desire to redefine its behaviour.

The essential change between IPv4 and IPv6 is that in IPv6 the *Don't Fragment* bit is always on, and because its always on, its not explicitly contained in the IPv6 packet header (Figure 3). There is only one fragmentation flag in the Fragmentation Header, the "More Fragments" bit, and the other two bits are reserved. The other change was that the packet identifier size was doubled in IPv6, using a 32-bit packet identifier field.

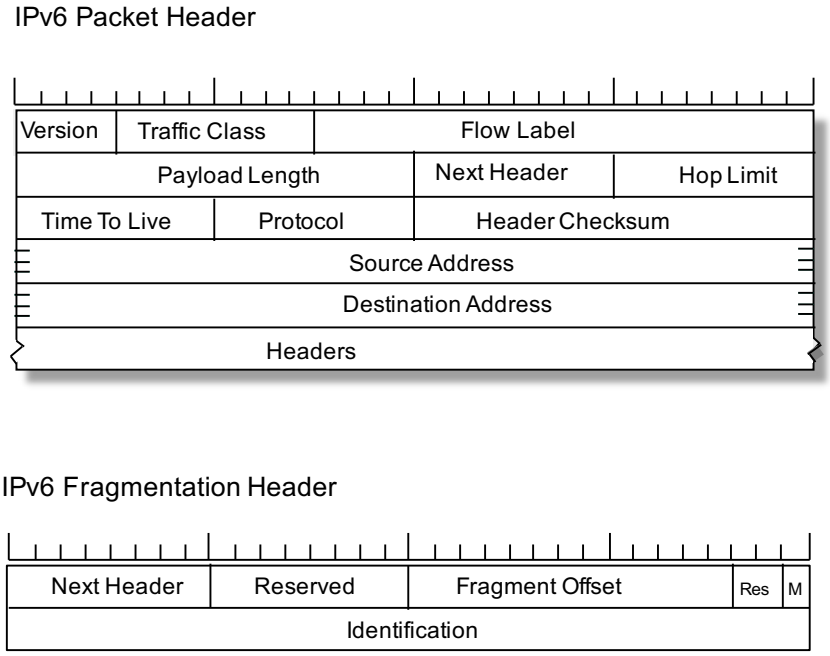

Figure 3 – IPv6 Packet Header and Fragmentation Header

An IPv6 router cannot fragment an IPv6 packet, so if the packet is too large for the next hop the router is required to generate an ICMP6 Type 2 packet, addressed to the source of the packet with a Packet Too Big (PTB) code, and also providing the MTU size of the next hop. While an IPv6 router cannot perform packet fragmentation, the IPv6 sender may fragment an IPv6 packet at the source.

Evaluating IPv6 Packet Fragmentation

The hope was that these IPv6 changes would fix the problems seen with IPv4 and fragmentation.

Our experience appears to point to a different conclusion.

The first problem is that there is widespread ICMP packet filtering in today's Internet. For IPv4 this was basically a reasonable defense tactic, and if you were willing to have a packet fragmented you cleared the *Don't Fragment* bit before sending the packet so that you were not reliant of receiving an ICMP message to indicate a Path sender MTU problem. But in IPv6 the equivalent *Don't Fragment* bit functionality is jammed in the "on" position, and the only way fragmentation will be performed is that if the original sender receives the ICMP6 PTB message and then resends the packet fragmented into a size that will meet the specified MTU size. But when ICMP6 PTB messages are filtered then the large packet in silently discarded within the network without any discernible trace. Attempts by the sender to time out and resend the large IPv6 packet will meet with the same fate, so this can lead to a wedged state.

This has been seen in the context of the HTTP protocol, where the path MTU is smaller than the MTU of the host systems at either end. The TCP handshake completes as none of the opening packets are large. The opening HTTP GET packet also makes it through as this is normally not a large packet. However, the first response may be a large packet. If it is silently discarded because of the combination of fragmentation required and ICMP6 filtering, then neither the client nor the server are capable or repairing the situation. The connection hangs.

The second problem is that the ICMP6 PTB message is sent backwards to the source from the interior of a network path. Oddly enough, the IPv6 ICMP PTB message is perhaps the one critical instance in

the entire IP architecture where the IP source address is interpreted by anything than the intended destination. The problems here include path asymmetry, in that the source address may be unreachable from the point of the ICMP packet's generation. There is also the case of tunneling IP-in-IP. Because IPv6 fragmentation can only be performed at the source, should the ICMP message be sent to the tunnel ingress point or to the original source? If the tunnel ingress is used that this assumes that the tunnel egress performs packet reassembly, which can burden the tunnel egress. This is further confounded in the cross protocol case of IPv6-in-IPv4 and IPv4-in-IPv6. [RFC 4459]

The third problem is the combination of IPv6 packet fragmentation and UDP. UDP is an unreliable datagram delivery service, so a sender of an UDP packet is not expected to cache the packet and be prepared to resend it. A UDP packet delivery error can only be effected at the level of the application, and not at the IP or UDP protocol level. So what should a host do upon receipt of an ICMP PTB message if resending the IP packet is not an option? Given that the sender does not cache sent UDP packets the packet header in the ICMP6 message is unhelpful. As the original packet was UDP, the sender does not necessarily have a connection state, so it is not clear how this information should be retained and how and when it should be used. How can a receiver even tell if an ICMP6 PTB packet is genuine? If the sender adds an entry into its local IPv6 forwarding table then it is exposing itself to a potential resource starvation problem. A high volume flow of synthetic PTB messages has the potential to bloat the local IPv6 forwarding table. If the sender ignores the PTB message than the application is left to attempt to recover the transaction.

If it makes little sense in the context of an attempt to fragment a UDP packet, it makes less sense to fragment a TCP packet. In the context of a TCP session a received ICMP6 PTB Message can be interpreted as a re-definition of the remote end MSS value, and the outgoing TCP segments can be re-framed to conform to this MSS.

Wither Fragmentation?

The basic problem here is that the network was supposed to operate at the IP level and be completely unaware of transport. This implies that IP level fragmentation was meant to work in a manner that does not involve transport protocol interaction. So much of today's network (firewalls, filters, etc.) is transport-aware and the trailing fragments have no transport context. That means that transport aware network middleware needs to re-assemble the packet, which could represent a problem and a DOS vulnerability in its own right.

So is fragmentation worth it at all?

I'd still say that's its more useful to have it than not. But the IPv4 model of *forward fragmentation* on the fly has proved to be more robust than the IPv6 model because the IPv4 model only requires traffic flows in one direction and is an IP level function. It has its problems, and no doubt the papers that warned that IP fragmentation was "harmful" were sincere in taking that view [Kent 87]. But it is possible to make it worse, and the IPv6 model requiring a *backward* ICMP6 message from the interior of the network was in retrospect a decision that made the issue worse!

So what should we do now?

It is probably not a realistic option to try and alter the way that IPv6 manages fragmentation. There was an effort in 2013 in one of the IETF's IPv6 Working Groups to deprecate the IPv6 Fragment Header [Bonica 2013]. That's possibly an over-reaction to the problem of packet fragmentation and IPv6, but there is no doubt that the upper level protocols simply should not assume that IPv6 fragmentation operates in the same manner as IPv4, or even operates in a reliable manner at all!

That implies that transport protocol implementations, and even applications, should try and manage their behaviour on the assumption that ICMP message filtering is sufficiently prevalent that it is prudent to assume that all ICMP messages are dropped. The result is a default assumption that large IPv6 packets that require fragmentation are silently dropped.

How can we work around this and operate a network that uses variable-sized packets, but cannot directly signal when a packet is too large? RFC4281 describes a Path MTU Discovery process that operates without relying on ICMP messages, and IPv6 TCP implementations should rely on this mechanism to establish and maintain a viable MTU size that can support packet delivery. In this way TCP can manage the path MTU and the application layer need not add explicit functionality to manage persistent silent drop of large segments.

Path MTU Discovery

Path MTU discovery was specified in RFC 1191. The approach was to send packets with the Don't Fragment bit set. Where a router on the path is unable to forward the packet because it is too large for the next hop, the Don't Fragment field directs the router to discard the packet and send a Destination Unreachable ICMP message with a code of "Fragmentation Required and DF set" (Type 3, Code 4). RFC1191 advocated the inclusion of the MTU of the next hop network in the next field of the ICMP message.

A host receiving this form of ICMP message should store the new MTU in the local forwarding table, with an associated time to allow the entry to time out. Also the host should identify all active TCP sessions that are connected to the same destination address as given in the IP packet header fragment of the ICMP message, and notify the TCP session of the revised path MTU value.

RFC1981 defined the much the same behavour for IPv6, relying on the MTU information conveyed in the ICMP6 Packet Too Big information in exactly the same manner as its IPv4 counterpart.

The problem of filtered ICMP messages is a difficult one, and attention has turned to path MTU Discovery ideas that do not rely on an ICMP message to operate correctly. RFC4821 describes a process refines the RFC 1191 ICMP-based process by adding an alternate process that is based on detection and reporting of packet loss as an inference of path MTU problems in those cases when there is no ICMP feedback. This uses a probe procedure that attempts to establish a working MTU size through probing the path with various sized packets to establish the upper bound MTU. The tradeoff here is the number of round trip intervals taken to perform the probes and the accuracy of the path MTU estimate.0

Because these probes take time, the entire exercise only tends to be of value in long held TCP and TCP-like flows. For shorter sessions the pragmatic advice is to clamp the local MTU to a conservative value (1,280 is a good first choice for IPv6, and RFC 4821 also suggests 1,024 for IPv4) and try to avoid the entire issue of fragmentation in the first place.

UDP is a different story. The lightweight UDP protocol shim does not admit much in the way of additional functionality, and one possible approach is to insist that UDP-based applications limit themselves to the local MTU size, or to be even more conservative, limit themselves to the 1,280 octet IPv6 minimum unfragmented packet size.

The major issue with such advice for UDP lies in the Domain Name System (DNS). Efforts to improve the security of the DNS (DNSSEC) have added additional data into DNS responses, and if you want to maintain the lightweight efficiency of the DNS then its not possible to keep DNSSEC responses under 1500 octets all of the time, let alone under 1,280 octets. One option here is to insist that larger DNS responses must use TCP, but this imposes some considerable cost overhead on the operation of the DNS. What the DNS has chosen to do appears to represent a reasonable compromise. The first part of the approach is that the management of the packet MTU is passed into the application layer. The application will conventionally operate with a maximum UDP payload size that assumes that UDP fragmentation is working, and a DNS query would normally offer an EDNS buffer size of 4,096 octets. The responder would use this to assemble its UDP response of up to 4,096 octets in length, which would conventionally cause the source to perform UDP packet fragmentation for large responses, and may invoke path fragmentation is the path MTU is lower than the responders local MTU. The EDNS buffer size is dropped back to a more conservative value that is not expected to trigger fragmentation after a number of unsuccessful attempts using a buffer size that would normally trigger fragmentation. The intended result is that if the network cannot complete a UDP transaction that entails a fragmented UDP response, the transaction is repeated using smaller maximum UDP packet size, and the truncated response explicitly signals to the client to re-try the query using TCP [RFC 6891]. This process is protocol agnostic, in that it will operate as intended in the case of IPv4 *forward fragmentation* where trailing fragments are filtered out by middleware, and in the case of IPv6, where there is no *forward fragmentation*, and it operates whether or not the responder receives any ICMP PTB messages.

Conclusion

What we have learned through all this is that packet fragmentation is extremely challenging, and is sensibly avoided, if at all possible.

Rather than trying to bury packet fragmentation to an IP level function performed invisibly at the lower levels of the protocol stack, a robust approach to packet fragmentation requires a more careful approach that lifts the management of Path MTU into the end-to-end transport protocol and even into the application.

IPv6 UDP-based applications that want a lightweight operation should look at keeping their UDP packets under the IPv6 1,280 octet unfragmented packet limit. And if that's not possible, then the application itself needs to explicitly manage Path MTU, and not rely on the lower levels of the protocol stack to manage this.

IPv6 TCP implementations should never assume that IPv6 PTB messages are reliably delivered. High volume flows should use RFC4821 Path MTU Discovery and management procedures to ensure that the TCP session can avoid Path MTU blackholing. For short flows MSS clamping still represents the most viable approach.

I'm not sure that we should go as far as deprecating IP fragmentation in IPv6. The situation is not that dire. But we should treat Path MTU with a lot more respect, and include explicit consideration of the trade-offs between lightweight design and robust behaviour in today's network.

References

[Bonica 2013] Bonica, R. W. Kumari, R. Bush, H. Pfeifer, "IPv6 Fragment Header Deprecated", work in progress, Internet draft: draft-bonica-6man-frag-deprecate, July 2013

[Kent87] Kent, C. and J. Mogul, "Fragmentation Considered Harmful", Proc. SIGCOMM '87 Workshop on Frontiers in Computer Communications Technology, August 1987.

[RFC 791] Postel, J., "Internet Protocol", September 1981.

[RFC 1191] Mogul, J., S. Deering, "Path MTU Discovery", November 1990.

[RFC 1858] Ziemba, G., Reed, D., and P. Traina, "Security Considerations for IP Fragment Filtering, October 1995.

[RFC 1981] McCann, J., Deering, S., and J. Mogul, "Path MTU Discovery for IP version 6", August 1996.

[RFC 4443] Conta, A., Deering, S., and M. Gupta, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", March 2006.

[RFC 4459] Savola, P., "MTU and Fragmentation Issues with In-the- Network Tunneling", April 2006.

[RFC 4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", March 2007.

[RFC 4963] Heffner, J., M. Mathis and B. Chandler, "IPv4 Reassembly Errors at High Data Rates", July 2007.

[RFC 5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", November 2008.

[RFC 6891] Damas, J. M. Graff, P. Vixie, "Extension Mechanisms for DNS (EDNS(0))", April 2013.

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