April 2022
Geoff Huston

Hop by Hop

It is a rare situation when you can create an outcome from two somewhat broken technologies where the outcome is not also broken. Unless each of the components can precisely complement each other such that a weakness in one can be covered by a strength in the other, then the most likely outcome is that the result is even worse than either technology.

No, I'm not talking about the ill-fated Next Generation Network (NGN) proposals of a distant past (although the same observation applies to the NGN work!). I'm referring to a more recent effort to try and salvage something from the debacle that is IPv6 packet fragmentation support by taking another piece of operationally broken IPv6, namely Hop-by-Hop (HBH) extension headers and trying to use that to solve the IPv6 Path Maximum Transfer Unit Discovery (PMTUD) problem.

To provide some background to this it's probably worth going back in time to the late 1980s and look at the discussion about packet size and on-the-fly 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 frames of data, 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.

This change allowed for more efficient and cheaper networks. The network could dispense with a fully synchronized highly-stable clock, which is a win in terms of costs, and each individual data flow can take advantage of idle network capacity, which is a huge win in terms of overall network efficiency.

There are also other potential optimizations when a network accepts variably-sized packets. 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 turn 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. In a mixed-load carriage network there is no single best answer here, and the optimal response is to let the application pick its own packet sizes to optimize its behaviour.

Even then, it's not a completely unrestricted choice. Various network designs adopted various limiting parameters for packet size. Ethernet, invented in the early 1970s, adopted a variable packet size, with supported packet sizes of between 64 and 1,500 octets. Fibre Distributed Data Interface (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 digitized voice, may prefer to use a stream of smaller packets to minimize jitter effects, 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 minimizes the impact of sporadic errors within the data stream, which may increase throughput in such environments.

In designing a common 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 the Internet Protocol (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.

Packet fragmentation

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. It does this by copying the original IP header field into each of these fragments, then forwarding each of these fragments as independent IP packets. The destination host is responsible for reassembling these fragments back into the original IP packet and passing the result, namely the IP packet payload as it was originally sent, back to the local instance of the end-to-end transport protocol (Figure 1).

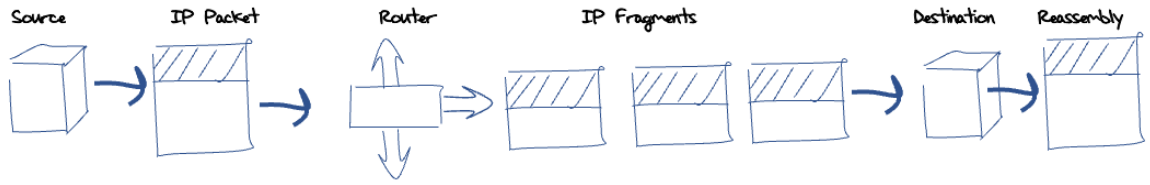

Figure 1 – IPv4 forward fragmentation.

This 'on-the-fly' packet fragmentation is a clever approach, as it hides the entire network-level packet size issues from the upper-level protocols, including TCP and UDP, but it also attracted its share of criticism. Packet fragmentation was seen as being a source of inefficiency, a security vulnerability and even posed an upper limit on maximal delay bandwidth product on data flows across networks. An influential paper by Kent and Mogul, "Fragmentation Considered Harmful", published in ACM SIGCOMM in 1987, called the entire concept of IP packet fragmentation into question, and this influenced some aspects of the design of the IPv6 protocol.

While the IPv6 protocol was intended to closely resemble IPv4 in many respects, one major point of difference was in the protocol's treatment of packet fragmentation. Firstly, IPv6 removes the always present fragmentation controls from the common IPv4 packet header and placed these control elements into an optional extension header. This IPv6 extension header was only present in fragmented packets. Secondly, IPv6 does not permit fragmentation to be performed when the packet is in transit within the network. All fragmentation is performed by the packet source prior to transmission.

When a router cannot forward an IPv6 packet because the packet is too large for the next hop network, it is supposed to pass a control message back to the packet's source address. This Internet Control Message Protocol version 6 (ICMPv6) Packet Too Big (PTB) message contains the maximum packet size of the link that caused the forwarding failure (the Maximum Transmission Unit size, or MTU) and as much of the original packet as possible, without having the ICMPv6 packet exceed the minimum unfragmented MTU size of 1,280 octets. When the sending host receives this ICMPv6 PTB message

then it should store this association of the destination address and the updated path MTU in the host's forwarding table. This should ensure that any further packets that are directed to this destination are fragmented by the host prior to passing them into the network. Through this mechanism, IPv6 packet fragmentation is still permitted but not within the network itself (Figure 2).

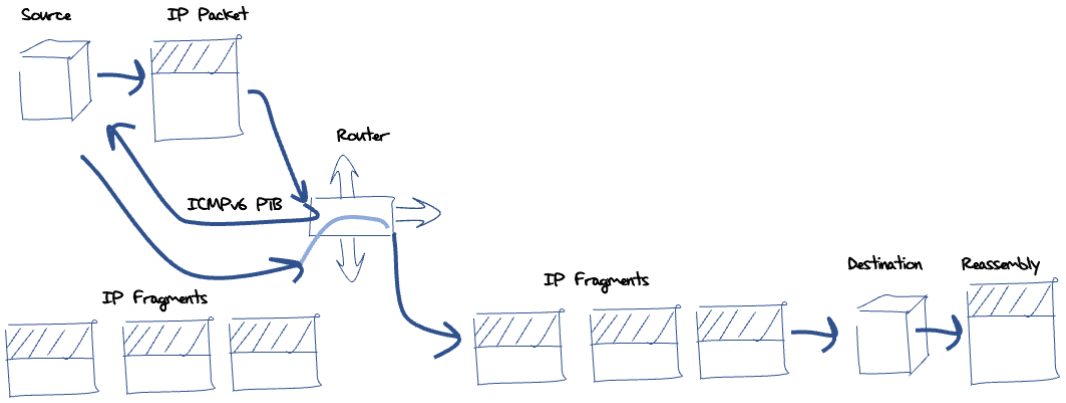

Figure 2 – IPv6 reverse fragmentation.

In the case of TCP, a small amount of layer violation goes a long way. If the TCP driver consults the forwarding table for an updated MTU value prior to sending each packet, the TCP session can update its Maximum Segment Size (MSS) setting in accordance with the updated destination MTU size. This dynamic adjustment of the TCP MSS value implies that TCP need not emit fragmented packets.

There are some issues with this ICMPv6 approach:

- ICMPv6 messages are unverified, and while its reasonable for a host to assume that all received ICMPv6 messages are genuine, it can be a somewhat foolhardy assumption. Many security firewalls discard all incoming ICMPv6 messages.

- These *reverse* ICMPv6 messages assume that the interior of a network uses the same addressing realm as the edge. This may not be the case. A good illustration of this is an IP-in-IP tunnel, where the source address of the packet is the tunnel ingress point, not the original sender.

- The approach assumes that the sender's IP address identifies the host that actually sent the packet. For anycast networks this is not necessarily the case.

- How long should a host maintain this routing table entry? There is no clear "right" answer here.

One response to these issues is to set the sender's MTU size to a setting where fragmentation will not be encountered. All IPv6 networks are supposed to pass all packets up to 1,280 octets in size without fragmentation, so 1,280 octets might be a good value. One could be a little more adventurous and try larger settings, such as 1,340, 1,380 or 1,400 octets. In each case, the trade-off is one of a slightly greater carriage efficiency (ratio of payload to packet header overhead) as compared to the risk of encountering poorly handled packet fragmentation.

Another approach is to be a little more diligent in discovering the path MTU.

Path MTU Discovery

This is not a new problem for IP and following on from the "Fragmentation Considered Harmful" paper, RFC 1063 "IP MTU Discovery Options" was published in 1988. This RFC proposed two new IP options that could be placed in the IP header. One was used by routers to record the minimum path MTU by updating the value in the probe option with the local next hop MTU if this is smaller than the IP packet's

option value, and the second was used by the destination host to reply to the sender with the received value.

Even in 1988, the prospect of adding new IP options was evidently not viewed with enthusiasm, and while this approach preserved the packet flow by passing the MTU discovery signal forward to the destination and then embedding the result in an IP options field in the reverse destination-to-source traffic flow, there simply wasn't the appetite at the time to add to the repertoire of IP options. A little over a year later, in November 1990, RFC 1063 was obsoleted by RFC 1191, which proposed the modification of ICMP Datagram Too Big messages to include the next hop MTU value in these ICMP messages. This was intended to inform the sender of what size packet would be passed by this router if the Don't Fragment bit was set in the packet.

These IPv4 ICMP Datagram Too Big messages had an IPv6 equivalent, the ICMPv6 PTB message (RFC 8201), even though the RFC 1191 practice of adding the next hop MTU value into these ICMP Datagram Too Big messages took many years to be commonly adopted. The IPv6 designers opted to avoid this slow adoption process by requiring the inclusion of the next hop MTU value from the outset in the IPv6 specification.

The IP level approach to path MTU discovery had its issues with ICMP processing, as already noted, and the option of using the TCP transport protocol for path MTU discovery was specified in March 2007's RFC 4821. A generalization of this approach using a generic packetization layer was described in RFC 8899. The basic approach is to probe the path with various-sized packets and get back a report of those packets that were successfully received.

The common issue with these probing approaches is that they can take a considerable amount to time to complete, and they only offer a net benefit to the end points if the ensuing data flow is sufficiently large that the small relative gains in carriage efficiency becomes a net positive benefit compared to the time taken to undertake these discovery tests. In other words, the application needs to be moving a very large volume of data for the marginal efficiency gains offset the time taken to perform the probe.

If you want to avoid this probing overhead, then your choices are limited. Either all IPv6 packet flows should use a packetization process that ensures that all packets are not fragmented by using a maximum packet size of 1,280 octets for every packet, or the sender must rely on the integrity of ICMPv6 PTB messages being passed back from inside the network to the sender to ensure that the sender can adjust their outgoing MTU to match that of the path to the destination.

Path MTU Discovery as an HBH function

This has prompted the proposal documented in draft-ietf-6man-mtu-option. This approach proposes to return to the now thirty-year old model first described in RFC 1063. The proposal uses a HBH extension header field that directs the routers along the path to update the header's MTU field if the next hop MTU is small than the value contained in the option. A control flag directs the receiver to copy the received MTU value into a return field and direct the option back to the original sender (Figure 3).

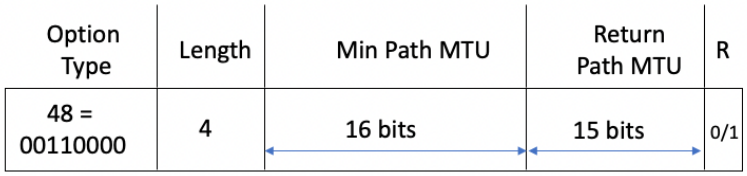

Figure 3 – Path MTU HBH header.

There are some necessary preconditions for this kind of approach to be viable. All hosts, routers, and all forms of sundry middleware need to support this option. By 'support' I mean that:

- IPv6 packets carrying this HBH option are not arbitrarily discarded by devices on the path or by the destination host.
- All forwarding devices recognise this HBH option and will update the Min Path MTU field when forwarding IPv6 packets that contain this HBH extension header.
- All IPv6 hosts recognise this HBH option and receivers will echo all received path MTU HBH options values back to the sender as determined by the control flag.
- All IPv6 hosts will make local adjustments to their stored value of path MTU in accordance with this received MTU information.
- All IPv6 protocol implementations need to support a socket option to allow upper layer protocols to request this path MTU probe function.

That's a pretty forbidding set of preconditions and based on experience so far in the diverse networking environment of the public Internet, these preconditions are highly unlikely to be met. Our experience with HBH extension options in the IPv6 Internet are not exactly encouraging so far.

At APNIC Labs we have been measuring the drop rate of IPv6 packets that contain the most benign form of HBH extension header we could find, namely the use of an 8-octet padding option. We added this option into TCP data streams and then looked to see if the receiver subsequently acknowledged receipt of this packet in the TCP sequence number flow. Measuring 65 days in 2022 with some 5,000 measurements per day on the path from a set of IPv6 servers to a diverse collection of IPv6 client hosts, the average HBH option drop rate was measured across these 314,000 IPv6 end hosts to be an astounding 92%.

APNIC's measurement cannot discern between network drop and host drop, and some manual tests have shown that many host implementations of IPv6 discard incoming packets with unexpected HBH extension headers.

A group at the University of Liège has set up a slightly different test method. Instead of testing the combination of network and host to see if IPv6 HBH packets were delivered, this set of tests looks only a selected network paths in the network. They used 13 virtual hosts and performed a pairwise test of crafted IPv6 packets to see if the network passed these packets between each pair of VMs. The results for HBH options were similar to the APNIC results, and in the small sample of networks tested in this manner the drop rate for HBH headers was also best summarized as comprehensive!.

While the approach of using end-to-end signalling inside the HBH IPv6 extension header to record the path MTU in the IPv6 HBH options sounds like a fine approach, the practical experience with HBH headers in the public IPv6 Internet makes the prospects for this approach pretty dismal.

What's wrong with HBH extension headers?

Within a packet forwarding device packet throughput is of paramount importance, so the default path through the device is conventionally based on a very simple inspection of the packet's destination address, decrementing the packet's hop limit and once a forwarding decision has been made checking the network hop MTU against the packet size. Any other steps would most likely invoke the packet being passed into the device's control plane for processing. Advances in ASIC design may well change this picture in coming years, but to date the pressures of constructing devices with ever higher packet processing capacity does not also readily allow for constructing devices with greater breadth of processing functions, such as processing packets with HBH headers. What this implies is that the packet throughput for packets with HBH options is a far lower capacity than the unit's default packet throughput, and this continues to be the case even as processing capability increases.

As was noted in a recent draft: "In hindsight, HBH options were still not practical to be used widely in the Internet. Many operational routers are configured to drop all packets containing a HBH option header." Incidentally, this draft was written by the same authors as the HBH MTU option proposal!

It seems odd to me to propose a response to the packet fragmentation problem that was considered and rejected some 30 years ago and relies on a technology that clearly is impractical to use.

What's the point of this specification?

This HBH MTU draft is proposing a "solution" to IPv6 packet fragmentation that relies on a technology that is simply not supported in today's network, and highly likely not to be supported in next week's network either. So, if the entire concept of this approach is doomed from the start, then why is this draft in the final steps leading to publication as an RFC?

Many years ago, the IETF tried to distinguish itself from other technology standards bodies, notably those associated with the Open Systems Interconnection (OSI) effort, by claiming that the IETF specifications were grounded in specifying what worked, as distinct from documenting some ephemeral wish list of what some folk would dearly like to work!

The IETF's mantra of "Running Code" implied that the underlying specifications were not only implementable, but deployable, and the IETF's Standards Track emphasized that progress towards a 'Full IETF Standard' was contingent on widespread adoption. The IETF claimed that it was specifying not only technologies that could be constructed would work as intended and would interoperate with other implementations of the same specification, but also that these technologies were both useful and were used. At the time these were worthy aspirations, and they certainly distinguished the IETF from a number of other standards efforts that appeared to be little more than printing factories of vague and unworkable concepts.

It seems to me that the ensuing years have eroded these IETF aspirations. It appears that the IETF has decided that volume is far easier to achieve than quality. These days, what the IETF is generating as RFCs is pretty much what the IETF accused the OSI folk of producing back then: Nothing more than voluminous paperware about vapourware!

Disclaimer

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

Author

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

www.potaroo.net