Google Cloud

A Brief Look at Path MTU Discovery

"DRINK ME" To Get Through the Network

Rick Jones

Disclaimer: In no way, shape, or form should the results presented in this document be construed as defining an SLA, SLI, SLO, or any other TLA. The author's sole intent is to offer helpful examples to facilitate a deeper understanding of the subject matter.

Introduction

Historically, Google Compute Engine's networking has specified an uncommon guest instance vNIC Internet Protocol (IP) Maximum Transmission Unit (MTU) of 1460 bytes. Many other clouds, most private/corporate/enterprise networks, and the broader Internet have the standard, albeit still-constraining IP MTU of 1500 bytes as defined from the dawn of Ethernet. From time to time this difference has caused issues and confusion. While one can now configure an "industry standard" IP MTU of 1500 bytes for Compute Engine guest instances' vNICs, and even an MTU up to 8896 bytes, the default MTU remains 1460 bytes and so there may still be value in a description of how Path MTU discovery is intended to function and where/how it can go wrong.

The Short Version

IP Path MTU Discovery works by a sender setting the "Don't Fragment" (DF) bit in the IP datagram header. When an IP datagram with the DF bit set arrives at a router which would otherwise have to fragment the datagram to forward it, the datagram is dropped and the router sends a message back to the datagram's original sender informing the sender of the maximum size it may send without needing fragmentation. The sender then adjusts its outbound packet size accordingly.

This can go wrong when the messages informing senders of maximum sizes are blocked for administrative reasons, in which case the recourses are to either disable Path MTU discovery and live with fragmentation, reduce the IP MTU of the sender(s), or if the traffic is TCP, enable TCP's Path MTU Black Hole detection heuristics on the sender(s). Of those three, the best is option four - stop blocking the message informing senders of maximum sizes.

The Long Version

MSS Ain't MTU

Before we discuss Path MTU Discovery, there is one thing which is very important to get clear. So let's do that now.

When this document's author was taking Physics in high-school, the very wise teacher wanted to impress upon the students that while there was a relationship between the Mass of an object and its Weight, they were not the same thing. A similar confusion arises in networking between MTU (Maximum Transmission Unit) and MSS (Maximum Segment Size).

The choice of 1460 bytes for the default IP MTU of Cloud Engine guest instances' vNICs can add to that confusion.

Maximum Transmission Unit

Networks have both minimum and maximum packet sizes they can support. When Metcalfe et al. first defined Ethernet back in the mists of time, an Ethernet network would have a maximum frame size of 1518 bytes[1]. That consists of a 14 byte Ethernet header, 1500 bytes of Ethernet payload/data and 4 bytes of Frame Check Sequence (FCS). So the payload size of an Ethernet frame[2] is 1500 bytes. You can't send anything larger in a single, standard Ethernet frame.

Eth. Header	Ethernet Payload	FCS
14 Bytes	Up to 1500 Bytes	4 B

When one goes to send IP datagrams over Ethernet the IP datagram[3] is the Ethernet frame's payload. So the IP datagram, including the IP header, may be no larger than 1500 bytes[4]. This is then often expressed as the MTU or the IP MTU of the network interface.

Eth. Header	IPv4 Header	IPv4 Payload	FCS
14 Bytes	20 Bytes	Up to 1480 Bytes	4 B

The dominance of Ethernet for the data-link/physical network layers has cemented an IP MTU of 1500 bytes as "the standard." While other networking technologies with different maximum frame sizes (e.g. FDDI and Token Ring et al.) have tried to replace Ethernet, and in so doing change "the standard" IP MTU; to date, none have been successful. Perhaps someday something will.

[1] We are ignoring things like preambles, Start Frame Delimiter, and inter-frame gaps, and later developments like IEEE 802.3 and VLAN tagging. We are also ignoring de facto standards such as Jumbo Frames arising from standards body inertia.
[2] Packets at the Ethernet level are formally referred-to as Frames.
[3] Packets at the IP level are formally referred-to as Datagrams.
[4] We are ignoring IP fragmentation for the moment. Patience.

Maximum Segment Size

The Transmission Control Protocol (TCP) provides a byte-stream service to its users, but operates over a message-oriented network layer[5], e.g. IP. So, TCP takes the user's data, bundles ("segments") it into segments[6], which TCP then hands to IP to encapsulate in IP datagrams, which then (usually) hands off the IP datagram to Ethernet to be put into an Ethernet frame. And for reasons which are beyond the scope of this document to discuss in detail, TCP recognizes the undesirability of IP Fragmentation. So, when TCP establishes a connection, each end will look at information it has, such as the IP MTU of their respective network interfaces and use that to compute the maximum quantity of user data each believes can go into a TCP segment before the IP datagram carrying that TCP segment must be fragmented. Each TCP communicates this to the other via a Maximum Segment Size (MSS) TCP Option carried in their respective SYN(chronize) segments.

Remember the IP MTU includes the IP header and the IP payload. An IPv4 header is (usually) 20 bytes[7]. That means for a network with an "industry-standard" IP MTU of 1500 bytes there can be 1480 bytes of what IP considers payload. The entire TCP segment, including TCP header, is payload to IP, and must fit into that 1480 bytes. A TCP header was (generally) also 20 bytes[8]. So, on a network with an "industry-standard" 1500 byte IP MTU, the TCP MSS Option will be 1460 bytes.

Eth. Header	IPv4 Header	TCP Header	TCP Payload	FCS
14 Bytes	20 Bytes	20 Bytes	Up to 1460 Bytes	4 B

1460 bytes. Sound familiar? Indeed, that is also the default IP MTU of a Compute Engine guest instance's vNIC. So we have the unfortunate coincidence that the IP MTU of a guest instance in Compute Engine has the same numeric value as the typical TCP MSS on an industry-standard 1500 byte MTU network...

[5] See the Evi Nemeth reference at https://en.wikipedia.org/wiki/Layer_8
[6] Packets at the TCP level are formally referred-to as Segments.
[7] An IPv6 header is 40 bytes so adjust the math accordingly. Well, the first/fixed one anyway, and optional headers are beyond the scope of this write-up.
[8] "What about TCP Timestamps?" you may ask. That is also a discussion for another write-up.

Remember it!

So, what about the Physics teacher and impressing upon the students the difference between Mass and Weight? Well, to ensure the students would remember decades later, at the beginning of term the teacher had all the students, dressed in their jackets and ties, stand by their desks and start jumping up and down shouting:

Mass is NOT Weight!
Mass is NOT Weight!
Mass is NOT Weight![9]

In that vein, and circumstances permitting, if you would please stand where you are and shout:

MSS ain't MTU!
MSS ain't MTU!
MSS ain't MTU!

Then we can continue :)

Why We Avoid IP Fragmentation

When IP is handed data from above (e.g. from TCP), or receives an IP datagram to forward which is larger than can fit in the IP MTU of the interface out which IP wishes to send the datagram, one of its options is to fragment the too-large datagram into two or more IP datagram fragments. These fragments will then flow through the network(s) and, in theory, arrive at their final destination. IP at the final destination will reassemble those fragments into a complete IP datagram of the original size and hand the IP payload up to the higher layer protocol (e.g. TCP).

Sounds peachy. Ten IP Datagram Fragments. What's so bad about that?

[9] Whether this incident goes any distance in explaining how the author ended-up like he did is left to the imagination of the reader. It is fortunate that the Physics classroom and lab was in the basement of the building...

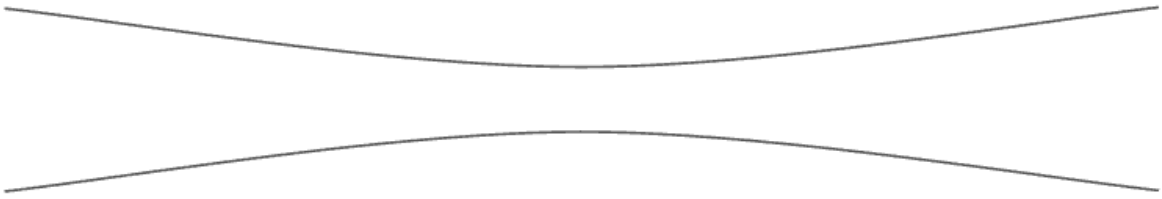
IP is just a “best effort” (aka one-shot) service and does not retransmit. IP datagrams and thus datagram fragments can be lost, and all the fragments of an IP datagram must arrive at the destination to be able to reassemble the original IP datagram.

So, if one datagram fragment is lost, the entire, original IP datagram is effectively lost and by extension the TCP segment is lost, and the bandwidth used to transmit the not-lost fragments wasted. TCP has to retransmit its segment, perhaps after a retransmission timeout. An underlying packet loss rate on the network is “amplified” into a much higher datagram/segment loss rate.

How We Avoid IP Fragmentation

This write-up has already mentioned one of the mechanisms used to avoid IP Fragmentation - namely the exchange of Maximum Segment Size (MSS) options in the TCP SYN(chronize) segments used to establish a TCP connection. This method works fine when the traffic is TCP and both endpoints are on the same network with the same IP MTU. Or when they are on different networks with the same IP MTU, and joined one network with the other via networks with the same or larger IP MTU. Or even when one endpoint is on a network with an MTU smaller than the other.

However, this method isn’t sufficient when the two endpoints are on different networks, joined one to the other along the way by a network with a smaller MTU. This aggregation of networks might be referred to as a “dumbbell network” because it is wide (large MTU) on either end and narrow (small MTU) in the middle. Like the weights on either end of the bar of a dumbbell.



Endpoint1<->BIGMTUNET1<->Router1<>smallmtunetwork<->Router2<->BIGMTUNET2<->Endpoint2

In this situation, each TCP computes and exchanges an MSS option based on its local IP MTU, and one side or the other starts sending TCP segments with that much data in them. Let’s assume Endpoint1 is doing the sending, and the BIGMTUNETs have an IP MTU of 1500 bytes and the smallmtunetwork has an IP MTU of 1460 bytes.

To Fragment, or Not To Fragment? That's some question.

Endpoint1 and Endpoint2 have exchanged TCP MSS options of 1460 bytes while establishing a TCP connection between them. Endpoint1 now sends a full-sized TCP segment containing 1460 bytes of user data. The 20 byte TCP header means TCP hands a 1480 byte segment to IP. That segment goes out Endpoint1's interface to BIGMTUNET1 in a 1500 byte IP datagram (1480 bytes of IP payload and a 20 byte IP header) in a 1518 byte Ethernet frame (14 bytes of Ethernet header, 1500 bytes of Ethernet data, and 4 bytes of Frame Check Sequence) and is received by Router1. The network interface on Router1 hands the frame to the driver, the driver hands the datagram to IP.

IP in Router1 consults its routing tables and determines the IP datagram carrying Endpoint1's TCP segment should be routed through smallmtunet. It also notices that smallmtunet has an IP MTU of 1460 bytes. But the IP datagram carrying Endpoint1's TCP's segment is 1500 bytes in size.

Dilemma. What should IP on Router1 do?

In short, it looks at a flag field in the IP header[10]. Within that part of the header is a single-bit flag known as the "Don't Fragment" flag or "DF bit". When that bit is clear (has a value of 0 - zero) IP on Router1 is supposed to fragment the too-large-to-forward-otherwise IP datagram into two or more IP datagram fragments and send them along their way on smallmtunetwork. As we've already seen, that has issues.

But TCP in Endpoint1 has another option. Rather than having IP in Endpoint1 send IP datagrams with the DF bit clear in the IP header, it can have IP set the DF bit in the IP header. Now when the 1500 byte IP datagram from Endpoint1 arrives at Router1, Router1 sees the DF bit is set. Instead of fragmenting the IP datagram, Router1 drops it. Wait! Drops it? What good does that do? That sounds just as bad as losing a fragment. Well, Router1 does something else as well. When Router1 drops the datagram, it sends a message back to Endpoint1 informing it of this. It does this with an `Internet Control Message Protocol (ICMP) Destination Unreachable, Datagram Too Big` message. And within that message is the maximum IP datagram size Router1 could have forwarded through smallmtunetwork without fragmentation (e.g. 1460 bytes - as the IP datagram size/MTU, not the TCP MSS!).

When this ICMP message arrives at Endpoint1, TCP is informed, and adjusts its effective MSS to result in TCP segments carried in IP datagrams no larger than the value carried in the ICMP message. It will immediately retransmit the data in the segment carried in the datagram

[10] "Wait!" you ask, "That's an IPv4 header. What about IPv6?" IPv6 does not include fragmentation by routers. All fragmentation is required to be performed by senders. Think of it as part of the Router Folks' Revenge. In essence it is as if there is a DF bit in the IPv6 header which is *always* set.

Router1 had to drop, and will ensure that any further segments TCP sends from Endpoint1 are sized to avoid fragmentation.

That is Path MTU Discovery. Setting the DF bit in the IP header to force routers along the path to inform the sender of the maximum IP datagram size the sender may send along that path without triggering IP fragmentation.

End of story, right? Well...not entirely.

Where Things Go Wrong

Network and other administrators are often concerned about security. Or perhaps more accurately, how to increase the cost of messing about with their networks to the point nefarious network ne'er-do-wells won't bother. We've already seen that Path MTU Discovery relies upon a particular message, an `ICMP Destination Unreachable, Datagram Too Big`, being sent by an intermediate device, a router, to the original sender, and that sender then taking appropriate action.

There are many other sorts of ICMP messages and some of them, such as ICMP Echo Request/Reply, are considered useful to attackers who want to map-out a target network. So, it has become somewhat common for network administrators to block ICMP messages. Either because they heard simply "Block ICMP for security!" and didn't think it through completely, or because the block mechanisms available to them are blunt instruments and can block only all ICMP messages rather than selected ones.

When this happens, it is referred to as an ICMP Black Hole or PMTU (Path MTU) Black Hole. When ICMP messages are blocked, the sending TCP will not be informed of the Path MTU, and since the DF bit means "Drop the datagram and send destination unreachable" it means traffic larger than the Path MTU appears to vanish - a black hole if you will into which (large) packets disappear never to be seen again.

Mitigations

There are (at least?) three possible mitigations when faced with a PMTU Black Hole which cannot be properly fixed. The first is to disable Path MTU Discovery by no longer setting the DF bit in the IP datagram headers. When the traffic needs to be forwarded across the smaller-MTU link, IP will fragment it. We trust luck and low packet loss rates to enable forward progress with fragmented traffic.

The second is to reduce the IP MTU of the endpoint(s) to that of the otherwise-smaller-MTU link along the path. The downside to this is it can become a race to the bottom, needing to configure ever-smaller MTUs as other, small MTU paths are encountered. Further, it can reduce performance when TCP has to send smaller segments, even in those cases where the potentially-but-not-sure-when small MTU path(s) aren't used. A variation on this theme can be to set a destination-specific MTU in the senders' routing tables. This will be considered along with the interface MTUs on the two endpoints when computing an MSS value for the exchange of SYNs for TCP.

The third is to enable PMTU Black Hole Detection in TCP. When this is enabled, a sending TCP will assume that after some number of retransmissions, the packet loss is perhaps the result of PMTU Black Hole, and will start trying smaller effective MSS values even without seeing an `ICMP Destination Unreachable, Datagram Too Big` message. The downside to this is it can take a few retransmissions to kick-in, and that will mean delays in data transfer.

The fourth of the three things to do... is to fix the PMTU Black Hole in the first place. Consider that IPv6 doesn't even specify routers fragmenting and so behaves as if there is a "DF bit" always set in the IPv6 header. Sometimes it will be suggested that the router simply ignore the DF bit and fragment anyway. That isn't a mitigation. It is a kludge.

Where Things Can Still Go Wrong

You may have heard people say things like "All hosts in a subnet" or "All stations in a broadcast domain[11]" "must have the same MTU." There is a reason. Remember earlier where we described Endpoint1 sending a TCP Segment containing 1460 bytes of user data in a 1500 byte IP Datagram across BIGMTUNET1 and that was received by Router1. Well, suppose for a moment that for some reason, Router1 didn't have a 1500 byte IP MTU configured on its interface to BIGMTUNET1. What could happen? Well, if this were a typical Ethernet setup, IP in Router1 wouldn't see the 1500 byte IP datagram in the first place, because it wouldn't have come-in to Router1 at the "Ethernet" layer in the first place. Router1's NIC on BIGMTUNET1 wouldn't have accepted the frame, because the frame was too big for what it had configured as a maximum frame size. And if IP in Router1 doesn't see the IP datagram, it has no way to send an `ICMP Destination Unreachable, Datagram Too Big` message. And it won't matter whether the DF bit is set or not in the IP header.

[11] Datalink-layer-speak for "All systems on the same side of a router."

Acknowledgements

The author would like to thank David Wetherall for his review. He would also like to thank Derek Phanekham for editing and updating this document for public release.