# Broken packets: IP fragmentation is flawed

*18 Aug 2017 by Marek Majkowski.*

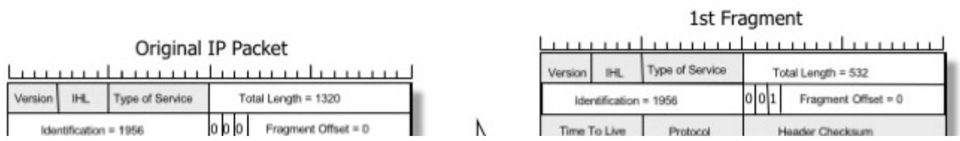
As opposed to the public telephone network (https://en.wikipedia.org/wiki/Public_switched_telephone_network) , the internet has a Packet Switched (https://en.wikipedia.org/wiki/Packet_switching) design. But just how big can these packets be?

CC BY 2.0 image (https://www.flickr.com/photos/ajmexico/8093997590) by ajmexico (https://www.flickr.com/photos/ajmexico/) , inspired by (https://blog.apnic.net/2016/05/19/fragmenting-ipv6/)

This is an old question and the IPv4 RFCs answer it pretty clearly. The idea was to split the problem into two separate concerns:

- What is the maximum packet size that can be handled by operating systems on both ends?
- What is the maximum permitted datagram size that can be safely pushed through the physical connections between the hosts?

When a packet is too big for a physical link, an intermediate router might chop it into multiple smaller datagrams in order to make it fit. This process is called "forward" IP fragmentation and the smaller datagrams are called IP fragments1 (#fn:1) .



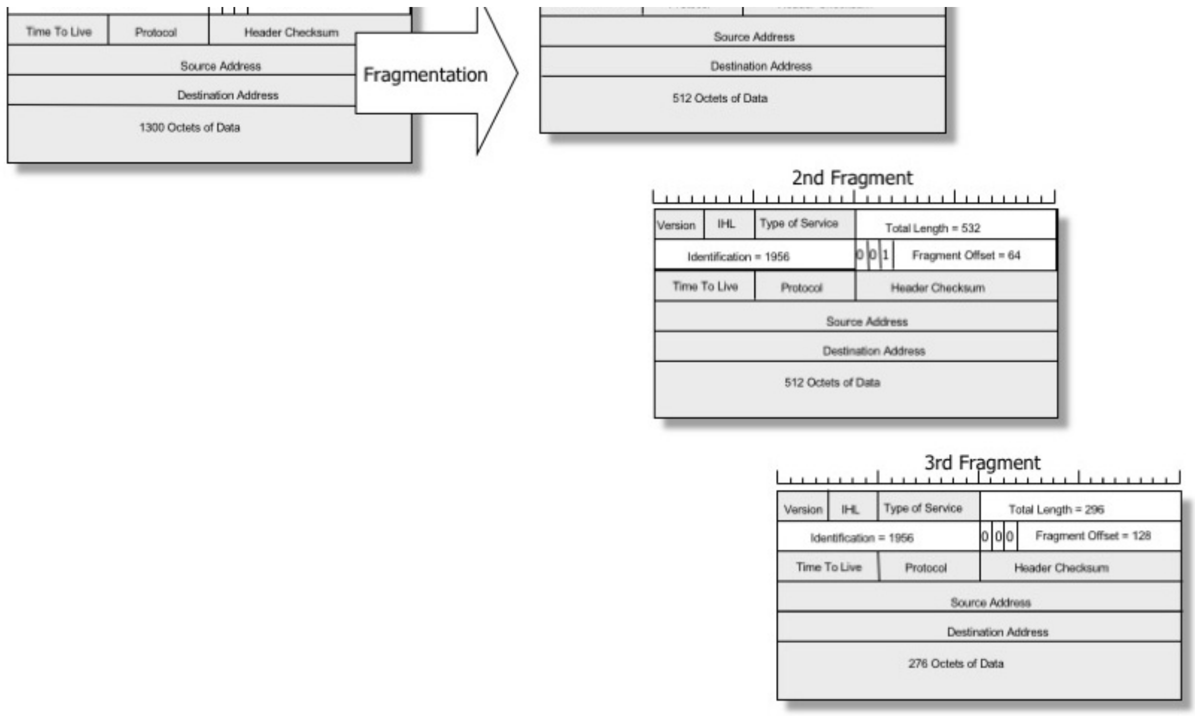


Image by Geoff Huston (https://blog.apnic.net/2016/01/28/evaluating-ipv4-and-ipv6-packet-frangmentation/) , reproduced with permission

The IPv4 specification defines the minimal requirements. From the RFC791 (https://tools.ietf.org/html/rfc791) :

```
Every internet destination must be able to receive a datagram
of 576 octets either in one piece or in fragments to
be reassembled. [...]

Every internet module must be able to forward a datagram of 68
octets without further fragmentation. [...]
```

The first value - permitted reassembled packet size - is typically not problematic. IPv4 defines the minimum as 576 bytes, but popular operating systems can cope with very big packets, typically up to 65KiB.

The second one is more troublesome. All physical connections have inherent datagram size limits, depending on the specific medium they use. For example Frame Relay can send datagrams between 46 and 4,470 bytes. ATM uses fixed 53 bytes, classical Ethernet can do between 64 and 1500 bytes.

The spec defines the minimal requirement - each physical link must be able to transmit datagrams of at least 68 bytes. For IPv6 that minimal value has been bumped up to 1,280 bytes (see RFC2460 (https://tools.ietf.org/html/rfc2460#section-5) ).

On the other hand, the maximum datagram size that can be transmitted without fragmentation is not defined by any specification and varies by link type. This value is called the MTU (Maximum Transmission Unit (https://en.wikipedia.org/wiki/Maximum_transmission_unit) )2 (#fn:2) .

The MTU defines a maximum datagram size on a local physical link. The internet is created from non-homogeneous networks, and on the path between two hosts there might be links with shorter MTU values. The maximum packet size that can be transmitted without fragmentation between two remote hosts is called a Path MTU (https://en.wikipedia.org/wiki/Path_MTU_Discovery) , and can potentially be different for every connection.

## Avoid fragmentation

One might think that it's fine to build applications that transmit very big packets and rely on routers to perform the IP fragmentation. This is not a good idea. The problems with this approach were first discussed by Kent and Mogul (http://www.hpl.hp.com/techreports/Compaq-DEC/WRL-87-3.pdf) in 1987. Here are a couple of highlights:

- To successfully reassemble a packet, all fragments must be delivered. No fragment can become corrupt or get lost in-flight. There simply is no way to notify the other party about missing fragments!
- The last fragment will almost never have the optimal size. For large transfers this means a significant part of the traffic will be composed of suboptimal short datagrams - a waste of precious router resources.
- Before the re-assembly a host must hold partial, fragment datagrams in memory. This opens an opportunity for memory exhaustion attacks.
- Subsequent fragments lack the higher-layer header. TCP or UDP header is only present in the first fragment. This makes it impossible for firewalls to filter fragment datagrams based on criteria like source or destination ports.

A more elaborate description of IP fragmentation problems can be found in these articles by Geoff Huston:

- Evaluating IPv4 and IPv6 packet fragmentation (https://blog.apnic.net/2016/01/28/evaluating-ipv4-and-ipv6-packet-frangmentation/)
- Fragmenting IPv6 (https://blog.apnic.net/2016/05/19/fragmenting-ipv6/)

## Don't fragment - ICMP Packet too big

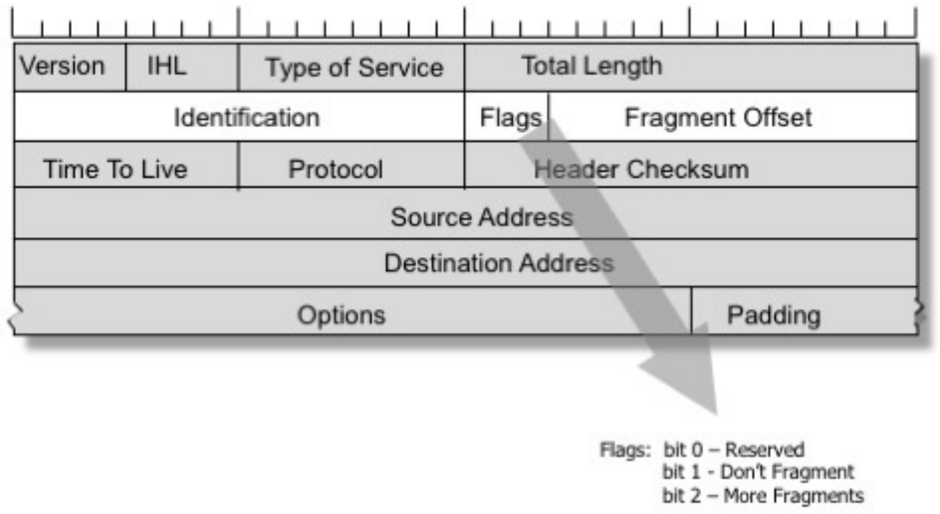


Image by Geoff Huston (https://blog.apnic.net/2016/01/28/evaluating-ipv4-and-ipv6-packet-frangmentation/) , reproduced with permission

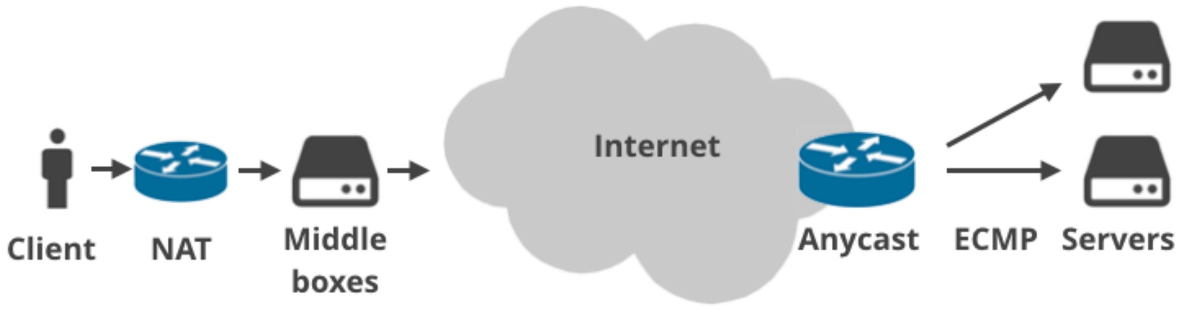
A solution to these problems was included in the IPv4 protocol. A sender can set the DF (Don't Fragment) flag in the IP header, asking intermediate routers never to perform fragmentation of a packet. Instead a router with a link having a smaller MTU will send an ICMP message "backward" and inform the sender to reduce the MTU for this connection.

The TCP protocol always sets the DF flag. The network stack looks carefully for incoming "Packet too big"3 (#fn:3) ICMP messages and keeps track of the "path MTU" characteristic for every connection4 (#fn:4) . This technique is called "path MTU discovery", and it is mostly commonly used for TCP, although it can also be applied to other IP-based protocols. Being able to deliver the ICMP "Packet too big" messages is critical in keeping the TCP stack working optimally.

## How the internet actually works

In a perfect world, internet connected devices would cooperate and correctly handle fragment datagrams and the associated ICMP packets. In reality though, IP fragments and ICMP packets are very often filtered out.

This is because the modern internet is much more complex than anticipated 36 years ago. Today, basically nobody is plugged directly into the public internet.

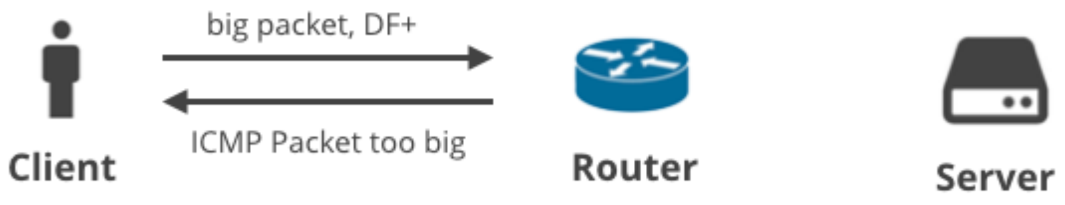


Customer devices connect through home routers which do NAT (Network Address Translation (https://en.wikipedia.org/wiki/Network_address_translation) ) and usually enforce firewall rules. Increasingly often there is more than one NAT installation on the packet path (e.g. carrier-grade NAT (https://en.wikipedia.org/wiki/Carrier-grade_NAT) ). Then, the packets hit the ISP infrastructure where there are ISP "middle boxes". They perform all manner of weird things on the traffic: enforce plan caps, throttle connections, perform logging, hijack DNS requests, implement government-mandated web site bans, force transparent caching or arguably "optimize" the traffic in some other magical way. The middle boxes are used especially by mobile telcos.

Similarly, there are often multiple layers between a server and the public internet. Service providers sometimes use Anycast BGP routing (https://en.wikipedia.org/wiki/Anycast#Internet_Protocol_Version_4) . That is: they handle the same IP ranges from multiple physical locations around the world. Within a datacenter on the other hand it's increasingly popular to use ECMP Equal Cost Multi Path (https://en.wikipedia.org/wiki/Equal-cost_multi-path_routing) for load balancing.

Each of these layers between a client and server can cause a Path MTU problem. Allow me to illustrate this with four scenarios.

## 1. Client -> Server DF+ / ICMP

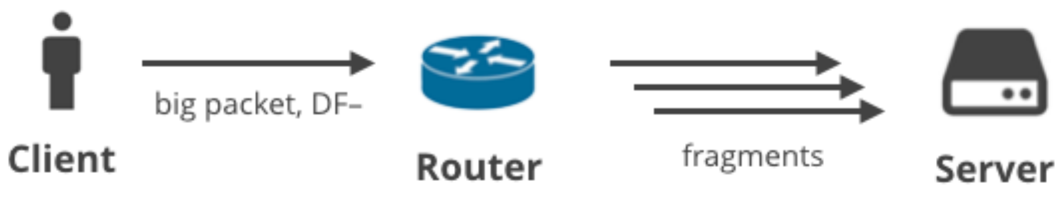


In the first scenario, a client uploads some data to the server using TCP so the DF flag is set on all of the packets. If the client fails to predict an appropriate MTU, an intermediate router will drop the big packets and send an ICMP “Packet too big” notification back to the client. These ICMP packets might get dropped by misconfigured customer NAT devices or ISP middle boxes.

According to the paper by Maikel de Boer and Jeffrey Bosma (https://www.nlnetlabs.nl/downloads/publications/pmtu-black-holes-msc-thesis.pdf) from 2012 around 5% of IPv4 and 1% of IPv6 hosts block inbound ICMP packets.

My experience confirms this. ICMP messages are indeed often dropped for perceived security advantages, but this is relatively easy to fix. A bigger issue is with certain mobile ISPs with weird middle boxes. These often completely ignore ICMP and perform very aggressive connection rewriting. For example Orange Polska not only ignores inbound "Packet too big" ICMP messages, but also rewrites the connection state and clamps the MSS (http://lartc.org/howto/lartc.cookbook.mtu-mss.html) to a non-negotiable 1344 bytes.

## 2. Client -> Server DF- / fragmentation



In next scenario, a client uploads some data with a protocol other than TCP, which has the DF flag cleared. For example, this might be a user playing a game using UDP, or having a voice call. The big outbound packets might get fragmented at some point in the path.

We can emulate this by launching `ping` with a large payload size:

```
$ ping -s 2048 facebook.com
```

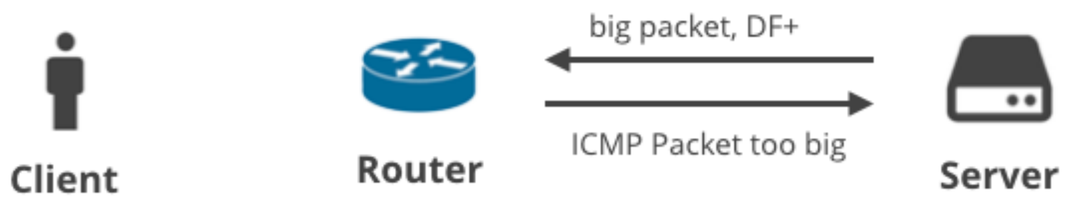
This particular `ping` will fail with payloads bigger than 1472 bytes. Any larger size will get fragmented and won’t get delivered properly. There are multiple reasons why servers might mishandle fragments, but one of a popular problems is the use of ECMP load balancing. Due to the ECMP hashing, the first datagram containing a protocol header is likely to be load-balanced to a different server than the rest of the fragments, preventing the reassembly.

For a more detailed discussion of this issue, see:

- Our previous write up on ECMP (https://blog.cloudflare.com/path-mtu-discovery-in-practice/) .
- How Google attempts to solve ECMP fragmentation issues with Maglev L4 Load balancer (https://research.google.com/pubs/archive/44824.pdf) .

Furthermore, server and router misconfiguration is a significant issue. According to RFC7852 (https://tools.ietf.org/html/rfc7872) between 30% and 55% of servers drop IPv6 datagrams containing fragmentation header.

### 3. Server -> Client DF+ / ICMP



The next scenario is about a client downloading some data over TCP. When the server fails to predict the correct MTU, it should receive an ICMP “Packet too big” message. Easy, right?

Sadly, it's not, again due to ECMP routing. The ICMP message will most likely get delivered to the wrong server - the 5-tuple hash of ICMP packet will not match the 5-tuple hash of the problematic connection. We wrote about this in the past (https://blog.cloudflare.com/path-mtu-discovery-in-practice/) , and developed a simple userspace daemon to solve it. It works by broadcasting the inbound ICMP “Packet too big” notification to all the ECMP servers, hoping that the one with the problematic connection will see it.

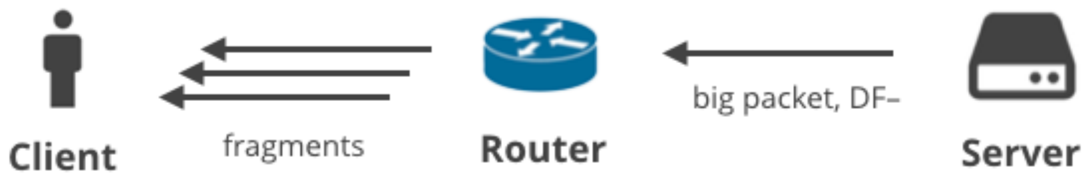
Additionally due to Anycast routing, the ICMP might be delivered to the wrong datacenter altogether! Internet routing is often asymmetric and the best path from an intermediate router might direct the ICMP packets to the wrong place.

Missing ICMP “Packet too big” notifications can result in connections stalling and timing out. This is often called a PMTU blackhole (https://en.wikipedia.org/wiki/Path_MTU_Discovery#Problems_with_PMTUD) . To aid this pessimistic case Linux implements a workaround - MTU Probing RFC4821 (http://www.ietf.org/rfc/rfc4821.txt) . MTU Probing tries to automatically identify packets dropped due to the wrong MTU, and uses heuristics to tune it. This feature is controlled via a sysctl:

```
$ echo 1 > /proc/sys/net/ipv4/tcp_mtu_probing
```

But MTU probing is not without its own issues. First, it tends to miscategorize congestion-related packet loss as MTU issues. Long running connections tend to end up with a reduced MTU. Secondly, Linux does not implement MTU Probing for IPv6.

### 4. Server -> Client DF- / fragmentation



Finally, there is a situation where the server sends big packets using a non-TCP protocol with the DF bit clear. In this scenario, the big packets will get fragmented on the path to the client. This situation is best illustrated with big DNS responses. Here are two DNS requests that will generate large responses and be delivered to the client as multiple IP fragments:

```
$ dig +notcp +dnssec DNSKEY org @199.19.56.1
$ dig +notcp +dnssec DNSKEY org @2001:500:f::1
```

These requests might fail due to already mentioned the misconfigured home router (https://en.wikipedia.org/wiki/Customer-premises_equipment) , broken NAT, broken ISP installations, or too restrictive firewall settings.

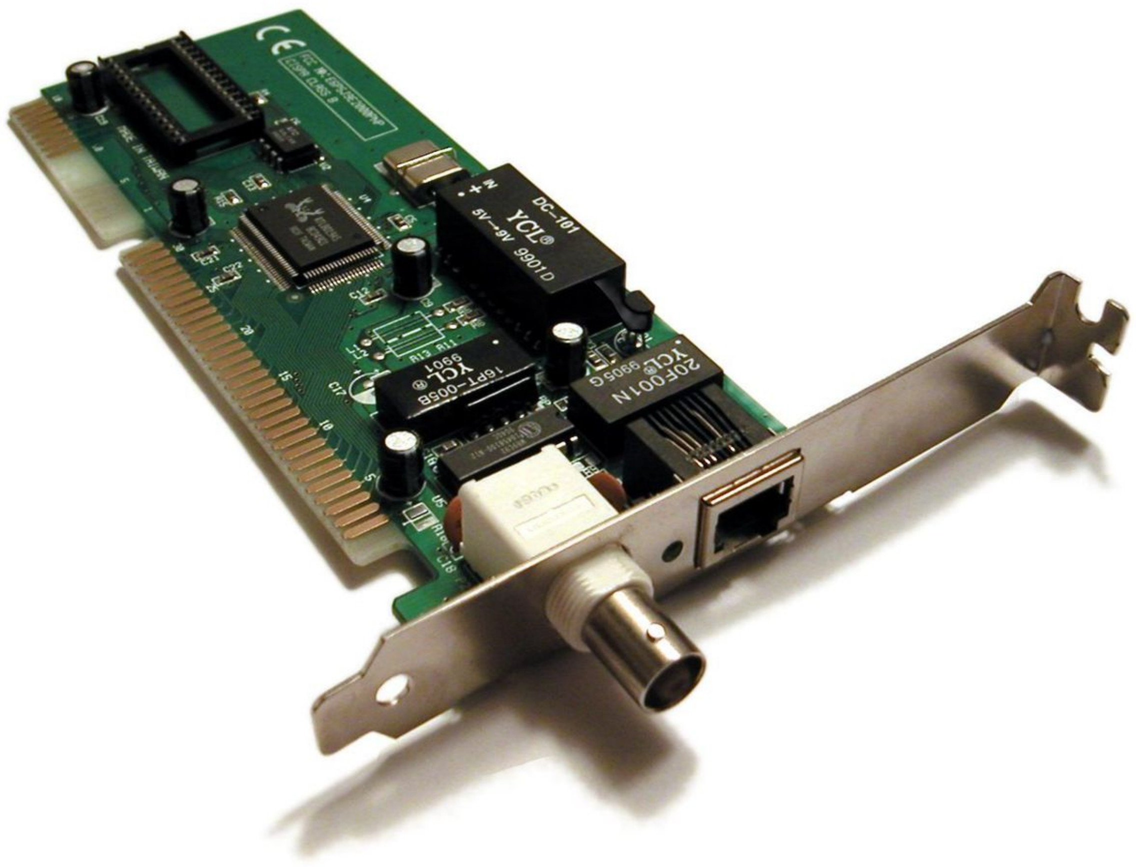
According to Boer and Bosma (https://www.nlnetlabs.nl/downloads/publications/pmtu-black-holes-msc-thesis.pdf) around 6% of IPv4 and 10% of IPv6 hosts block inbound fragment datagrams.

Here are some links with more information about the specific fragmentation issues affecting DNS:

- DNS-OARC Reply Size Test (https://www.dns-oarc.net/oarc/services/replysizetest)
- IPv6, Large UDP Packets and the DNS (http://www.potaroo.net/ispcol/2017-08/xtn-hdrs.html)

## Yet the internet still works!

With all these things going wrong, how does the internet still manage to work?



CC BY-SA 3.0, source: Wikipedia (https://commons.wikimedia.org/w/index.php?curid=122201)

This mainly due to the success of Ethernet (https://en.wikipedia.org/wiki/Ethernet_frame) . The great majority of the links in the public internet are Ethernet (or derived from it) and support the MTU of 1500 bytes.

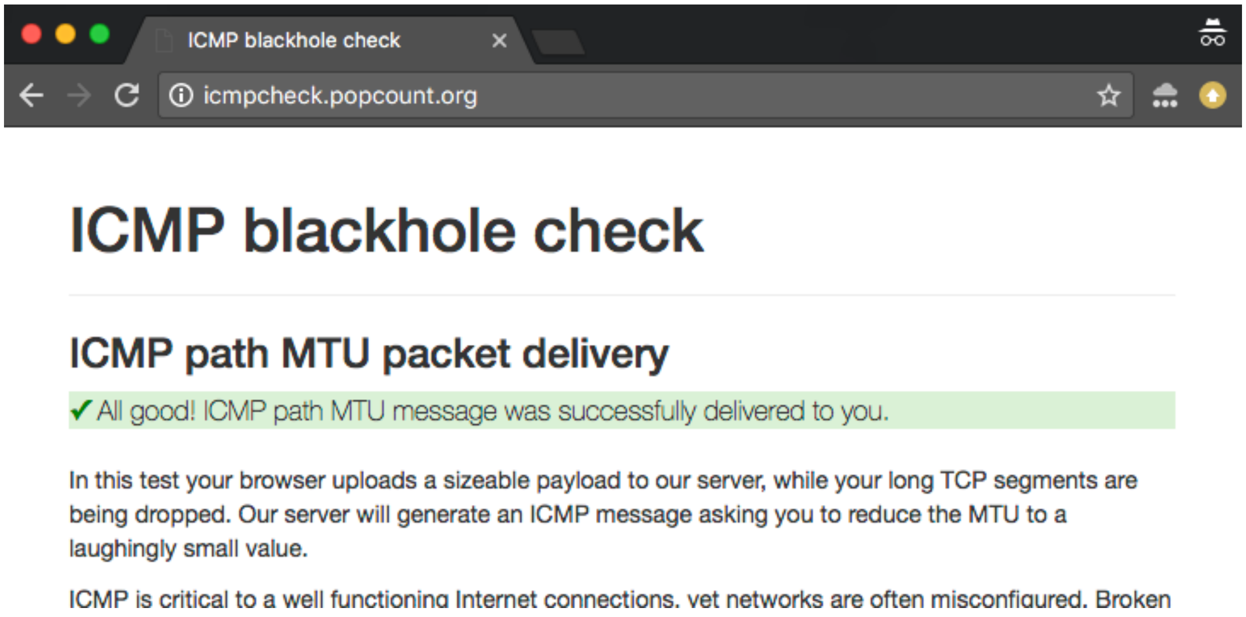
If you blindly assume the MTU of 1500[9] (#fn:9) , you will be surprised how often it will work just fine. The internet keeps on working mostly because we are all using an MTU of 1500 and rarely need to do IP fragmentation and send ICMP messages.

This stops working on an unusual setup with links having a non-standard MTU. VPNs and other network tunnel software must be careful to ensure that the fragments and ICMP messages are working fine.

This is especially visible in the IPv6 world, where many users connect through tunnels. Having a healthy passage of ICMP in both ways is very important, especially since fragmentation in IPv6 basically doesn't work (we cited two sources claiming between 10% and 50% of IPv6 hosts block IPv6 Fragment header).

Since the Path MTU issues in IPv6 are so common, many IPv6 servers clamp down Path MTU to the protocol mandated minimum of 1280 bytes. This approach trades a bit of performance for best reliability.

## Online ICMP blackhole checker



ICMP blackhole check

ICMP path MTU packet delivery

✓ All good! ICMP path MTU message was successfully delivered to you.

In this test your browser uploads a sizeable payload to our server, while your long TCP segments are being dropped. Our server will generate an ICMP message asking you to reduce the MTU to a laughingly small value.

ICMP is critical to a well functioning Internet connections, yet networks are often misconfigured. Broken

To help explore and debug these issues, we built an online checker. You can find two versions of the test:

- IPv4 version: http://icmpcheck.popcount.org (http://icmpcheck.popcount.org)
- IPv6 version: http://icmpcheckv6.popcount.org (http://icmpcheckv6.popcount.org)

These sites launch two tests:

- The first test will deliver ICMP messages to your computer, with the intention of reducing the Path MTU to a laughingly small value.
- The second test will send fragment datagrams back to you.

Receiving a "pass" in both these tests should give you a reasonable assurance that the internet on your side of the cable is behaving well.

It's also easy to run the tests from command line, in case you want to run it on the server:

```
perl -e "print 'packettoolongyaithuji6reeNab4XahChaeRah1diej4' x 180" > payloa
curl -v -s http://icmpcheck.popcount.org/icmp --data @payload.bin
curl -v -s http://icmpcheckv6.popcount.org/icmp --data @payload.bin
```

This should reduce the path MTU to our server to 905 bytes. You can to verify this by looking into the routing cache table. On Linux you do this with:

```
ip route get `dig +short icmpcheck.popcount.org`
```

It's possible to clear the routing cache on Linux:

```
ip route flush cache to `dig +short icmpcheck.popcount.org`
```

The second test verifies if fragments are properly delivered to the client:

```
curl -v -s http://icmpcheck.popcount.org/frag -o /dev/null
curl -v -s http://icmpcheckv6.popcount.org/frag -o /dev/null
```

## Summary

In this blog post we described the problems with detecting Path MTU values in the internet. ICMP and fragment datagrams are often blocked on both sides of the connections. Clients can encounter misconfigured firewalls, NAT devices or use ISPs which aggressively intercept connections. Clients also often use VPN's or IPv6 tunnels which, misconfigured, can cause path MTU issues.

Servers on the other hand increasingly often rely on Anycast or ECMP. Both of these things, as well as router and firewall misconfiguration are often a cause for ICMP and fragment datagrams being dropped.

Finally, we hope the online test is useful and can give you more insight into the inner workings of your networks. The test has useful examples of tcpdump syntax, useful to gain more insight. Happy network debugging!

*Is fixing fragmentation issues for 10% of the internet exciting? We are hiring system engineers of all stripes, Golang programmers, C wranglers, and interns in multiple locations! Join us (https://www.cloudflare.com/careers) in San Francisco, London, Austin, Champaign and Warsaw.*

1. In IPv6 the "forward" fragmentation works slightly differently than in IPv4. The intermediate routers are prohibited from fragmenting the packets, but the source can still do it. This is often confusing - a host might be asked to fragment a packet that it transmitted in the past. This makes little sense for stateless protocols like DNS. ↩ (#fnref:1)
2. On a side note, there also exists a "minimum transmission unit"! In commonly used Ethernet framing, each

transmitted datagram must have at least 64 bytes on Layer 2. This translates to 22 bytes on UDP and 10 bytes on TCP layer. Multiple implementations used to leak uninitialized memory on shorter packets! ↩ (#fnref:2)

3. Strictly speaking in IPv4 the ICMP packet is named "Destination Unreachable, Fragmentation Needed and Don't Fragment was Set". But I find the IPv6 ICMP error description "Packet too big" much clearer. ↩ (#fnref:3)
4. As a hint, TCP stack also include a maximum allowed "MSS" value in SYN packets (MSS is basically an MTU value reduced by size of IP and TCP headers). This allows the hosts to know what is the MTU on *their* links. Notice: this doesn't say what is the MTU on the dozens internet links between the two hosts! ↩ (#fnref:4)
5. Let's err on the safe side. A better MTU is 1492, to accommodate for DSL and PPPoE connections. ↩ (#fnref:9)