IT'S ALL ABOUT THE MTU'S

Contact Point: Hans Addleman, Eli Dart, Jason Zurawski, Doug Southworth, Ken Miller

Audience: Network Engineers

Last updated: May 13, 2021

ABOUT EPOC

Over the last decade, the scientific community has experienced an unprecedented shift in the way research is performed and how discoveries are made. Highly sophisticated experimental instruments are creating massive datasets for diverse scientific communities and hold the potential for new insights that will have long-lasting impacts on society. However, scientists cannot make effective use of this data if they are unable to move, store, and analyze it. The Engagement and Performance Operations Center was established in 2018 as a collaborative focal point for operational expertise and analysis and is jointly led by Indiana University (IU) and the Energy Sciences Network (ESnet). EPOC provides researchers with a holistic set of tools and services needed to debug performance issues and enable reliable and robust data transfers. By considering the full end-to-end data movement pipeline, EPOC is uniquely able to support collaborative science, allowing researchers to make the most effective use of shared data, computing, and storage resources to accelerate the discovery process.

EPOC supports six main activities:

- *Roadside Assistance and Consultations* via a coordinated Operations Center to resolve network performance problems with end-to-end data transfers;
- *Application Deep Dives* to work more closely with application communities and understand full workflows for diverse research teams in order to evaluate bottlenecks and potential capacity issues;
- *Network Analysis enabled by the NetSage* monitoring suite to proactively discover and resolve performance issues;
- *Data Transfer Testing/ Data Mobility Exhibition* to check transfer times against known good end points;
- *Provision of managed services* via support through the IU GlobalNOC and our Network Partners;
- *Coordinated Training* to ensure effective use of network tools and science support.

JUMBO FRAMES, PATH MTU DISCOVERY, AND MTU TROUBLESHOOTING

MTU Basics

The Maximum Transmission Unit (MTU) setting on a network device or host interface defines the largest packet that can be sent or received by the interface. The packet normally has 2 parts; the header and data portions. The header contains the source address, destination address, flags, and other important information necessary to get a packet from A to Z. The data portion is

This material is based upon work supported by the National Science Foundation under Grant No. 1826994

info@epoc.global

variable in size up to the MTU setting and carries the payload or information being transmitted. Higher MTU values generally allow for more efficient and faster data transfers, especially across long distances. Fewer overall packets mean less processing overhead by the DTNs and network devices involved in the data transfer and allows for faster recovery during a packet loss event. In many cases, however, higher MTU values can lead to data transfer performance issues as it needs either a fully capable path configured for jumbo frames or the ability to run path MTU discovery (PMTUD) end to end. One misconfigured device in the network path can cause packet fragmentation or loss.

TCP Maximum Segment Size (MSS)

When a TCP connection is established, the SYN packets contain the maximum segment size (MSS) for each side. The TCP connection then uses the highest common MSS, which determines the largest packet that will be used (TCP data + TCP header + IP header determines the IP packet size). In many cases, the host interface MTU matches the path MTU, and MSS negotiation results in the correct value. However, there are some cases where some interfaces in the path between the two hosts have a smaller MTU than either host. In this case, the hosts will choose a MSS value that works for the hosts but not for the network path. Fortunately, there is an on-by-default mechanism to address this issue - Path MTU Discovery.

Path MTU Discovery

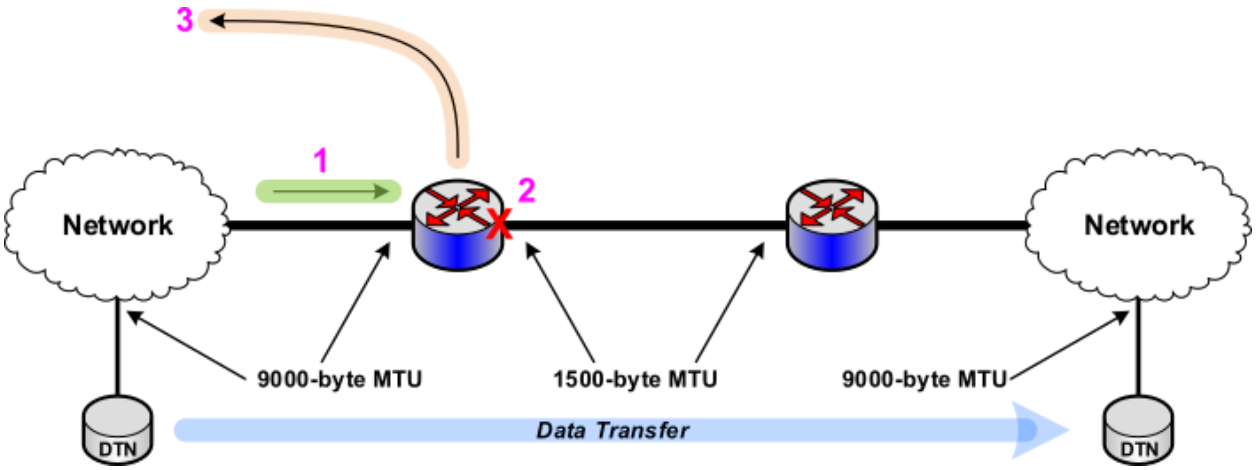
There are two primary Ethernet MTU sizes in use today, 1500 bytes (the standard Ethernet MTU) and 9000 bytes ("Jumbo Frames"). There are other MTUs in use on the Internet, for example because of specific technologies (e.g. the legacy SONET MTU of 4470 bytes), tunnels (including virtual private networks), or a number of other reasons. Because the smallest MTU of the set of interfaces in a network path between two hosts (the "path MTU") may be smaller than the MTU configured on the communicating hosts, the TCP/IP protocol suite needs a way to adjust a connection to conform to the path MTU. This mechanism is called Path MTU Discovery or PMTUD. RFC 1191[1] provides the full PMTUD specification.

An application or protocol (typically TCP) sets the DF (don't fragment) bit in the flags field of the IP packet header, instructing routers to drop the packet rather than fragment it if the packet is too large to forward. When forwarding a packet, the forwarding router checks the MTU of the outgoing interface, and if it is smaller than the packet it needs to send and DF is set, it drops the packet and sends the appropriate ICMP unreachable message (type 3, code 4 - fragmentation needed and DF set - along with the outgoing interface MTU) to the source address in the dropped packet. When it receives the ICMP message, the source host will then lower the size of

[1] https://tools.ietf.org/html/rfc1191

This material is based upon work supported by the National Science Foundation under Grant No. 1826994
info@epoc.global

the packet to fit inside this new MTU and resend. Note that the host must be able to receive the ICMP packets from PMTUD in order for PMTUD to function correctly - if the ICMP packets are blocked then PMTUD will fail. Because of this, over-zealous firewall configurations ("block all ICMP!") are a significant source of PMTUD failures.



The diagram above illustrates the correct operation of PMTUD. The incoming jumbo packet (1) sent by the source DTN cannot be forwarded out the router's standard-frame interface, and because DF is set the packet is dropped (2). The router then sends an ICMP unreachable message (3) to the source address (in this case, the address of the sending DTN).

If DF is not set, then the router can fragment the packet and send the fragments. However, this reduces performance and reduces reliability (especially because many firewalls block fragmented packets). TCP sets DF on all packets, so fragmentation typically happens in UDP-based applications. Even for UDP, fragmentation is typically undesirable for a variety of reasons, including performance and security.

PMTUD Black Holes

PMTUD is a Layer 3 construct which requires successful communication between routers and hosts in order to function correctly. When this communication fails due to lack of response from one of the routers, or from packets being dropped silently (e.g. because of blocking ICMP), then a so-called "Black Hole" situation occurs: the packets in question disappear from the network with no warnings or error messages. Users typically experience this as a timeout in a data transfer, or an SSH session hanging, or other similar non-responsive application behavior.

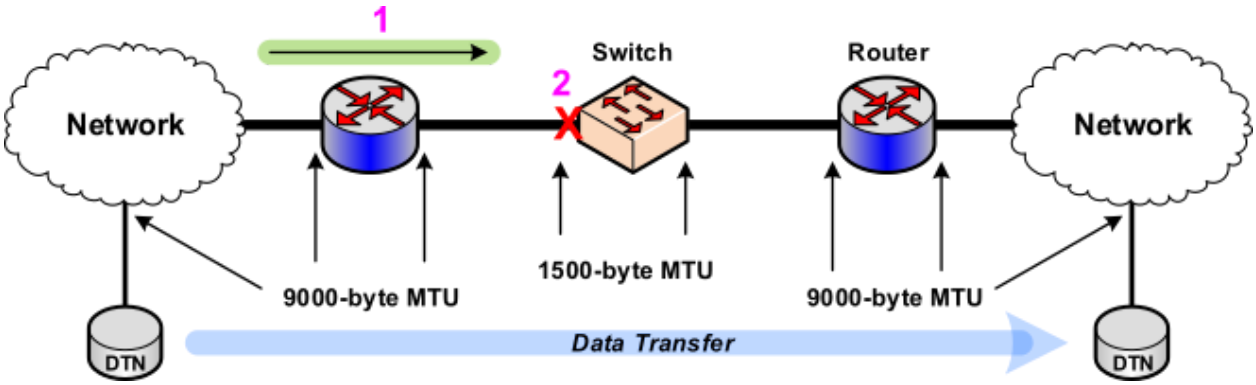
Three issues we see often as the primary cause of PMTUD Black Holes:

- A device in the path blocking ICMP

This material is based upon work supported by the National Science Foundation under Grant No. 1826994
info@epoc.global

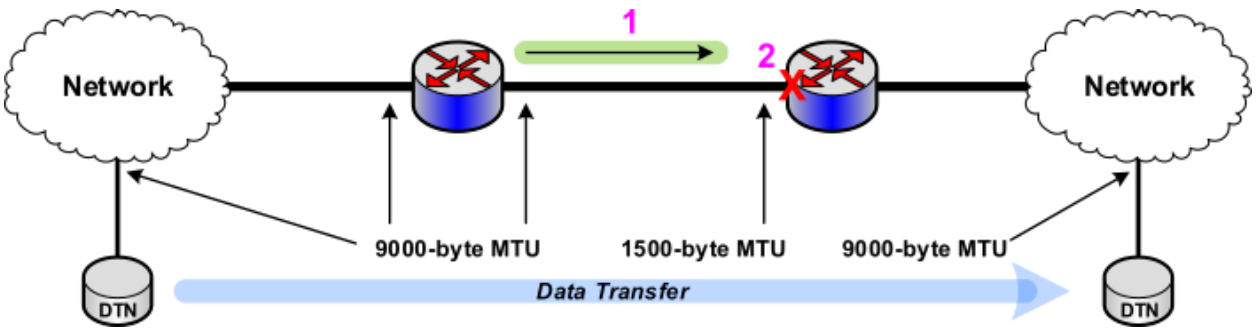
- Smaller frame MTU on a Layer 2 (switch) device between two 9000-byte jumbo frame routers in the data path
- One side of a link configured for standard frames and the other side configured for jumbo frames

If a Layer 2 device, configured with a lower MTU than the routers, is present in the path between the connected routers, the packet will be silently dropped or "black holed" because the layer2 switch will drop the jumbo frame packets as errors (these typically show up as "giants" in router and switch Ethernet interface statistics). This happens at the Ethernet layer, and so is invisible to IP. This is shown in the figure below, with the sending router forwarding the packet normally (1) and the switch dropping the packet as a giant (2). No ICMP message is sent, because the router is unaware of the problem, and the switch does not operate at layer 3. Note that this doesn't only occur with standard frame (1500 byte) Ethernet - some networks add a Q-in-Q (VLANs within VLANs) configuration to a jumbo frame link without first increasing the MTU to accommodate the additional 4-byte VLAN tag. This results in a Layer 2 MTU of 8996 bytes instead of 9000, and will cause the same problem.



A similar problem occurs if one side of a router-to-router link is configured with a 1500 byte MTU and the other side is configured with a 9000-byte MTU. Because the mismatch occurs outside of the routers, jumbo frame packets will be silently dropped in one direction. In this case, when forwarding a jumbo frame packet, the router on the jumbo frame side forwards the packet normally and the receiving (standard-frame) router drops the packet as a giant. The error counter on the standard-frame interface should increment, but there will be no communication back to the hosts or the data transfer application. The diagram below illustrates this version of the problem, with the packet being forwarded normally by the sending router (1) and being dropped as a giant by the receiving router (2). No ICMP message is sent, because the MTU change did not occur during packet forwarding on a router - the router with the standard frame interface behaved correctly and dropped the packet as an error (a giant in this case).

This material is based upon work supported by the National Science Foundation under Grant No. 1826994
info@epoc.global



Configuration Best Practices

Below are a few best practices that may help keep MTU issues from being a problem on your network.

Configuring your network hardware including firewalls to allow for path MTU discovery is critical to high throughput and packet loss free network operations. Allowing network devices to respond to ICMP requests and not blocking ICMP transiting your network help not only PMTUD, but also network troubleshooting tools such as traceroute and ping. Throttling the amount of ICMP traffic you transit or allow your network devices to respond to can be a good practice that may help avoid denial of service attacks.

If there is a security policy at your institution requiring ICMP to be blocked, we recommend working with your security office to acquire an exception for at LEAST the following type of ICMP requests. These are the minimum required ICMP messages for PMTUD to work properly:

- ICMP Unreachable
- ICMP time-exceeded - (needed for traceroute)

Verify consistent MTU values across network hardware and servers. Some network devices may have both Layer 2 and Layer 3 MTU settings that will need to be verified. Different segments of a network may have different MTU policies. The Science DMZ may have jumbo frames (9000 byte) enabled while the enterprise part may only be 1500 byte frame sizes to interface properly with firewalls or other security devices.

Path MTU Testing

You can test whether the path MTU between two hosts supports a 9000-byte MTU using the ping command with the DO NOT FRAGMENT (-M do) and packet size (-s 8972) flags set. The size

This material is based upon work supported by the National Science Foundation under Grant No. 1826994
info@epoc.global

of 8972 bytes specifies a 8972-byte payload for the ping packet, the ICMP header is 8 bytes, and the IP header is 20 bytes - this adds up to a 9000-byte IP packet. An example is below:

```
[user@host ~]$ ping -M do -s 8972 test.example.net
PING test.example.net (192.168.1.X) 8972(9000) bytes of data.
8980 bytes from test.example.net (192.168.1.X): icmp_seq=1
ttl=53 time=59.6 ms
8980 bytes from test.example.net (192.168.1.X): icmp_seq=2
ttl=53 time=59.6 ms
8980 bytes from test.example.net (192.168.1.X): icmp_seq=3
ttl=53 time=59.6 ms
8980 bytes from test.example.net (192.168.1.X): icmp_seq=4
ttl=53 time=59.6 ms
8980 bytes from test.example.net (192.168.1.X): icmp_seq=5
ttl=53 time=59.6 ms
--- test.example.net ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4002ms
rtt min/avg/max/mdev = 59.658/59.666/59.676/0.267 ms
```

When you ping a host with a packet that exceeds the MTU at any point in the network path you will get a response like the one below as long as ICMP is allowed:

```
[user@host ~]$ ping -M do -s 8972 X.X.X.X
PING X.X.X.X (192.186.0.1) 8972(9000) bytes of data.
From X.X.X.Y icmp_seq=1 Frag needed and DF set (mtu = 1500)
ping: local error: Message too long, mtu=1500
ping: local error: Message too long, mtu=1500
ping: local error: Message too long, mtu=1500
ping: local error: Message too long, mtu=1500
```

You can also use the tracepath tool to see where the MTU size changes on a path between 2 hosts. Tracepath differs from ping and traceroute in that it sends UDP packets instead of ICMP packets. It relies on ICMP responses from the network devices it queries along the path. In the example below you can see the MTU step down at hop 3. A network device can appear in a tracepath output twice (as shown in hop 3 below) when the MTU changes between inbound and outbound interface. perfSONAR (pS) also has tracepath as part of it's tool set if both end points have a pS node you can use pS in third party mode to test between them.

```
[user@host ~]# tracepath 192.168.0.1
1?: [LOCALHOST]                      pmtu 9000
1:  hop-1.example.net                4.007ms
2:  hop-2.example.net                2.737ms
3:  hop-3.example.net                2.680ms pmtu 1500
```


This material is based upon work supported by the National Science Foundation under Grant No. 1826994
info@epoc.global

```
3:  hop-3-1.example.net              4.999ms
4:  hop-2.example.net                48.826ms
5:  hop-2.example.net                47.683ms
6:  hop-2.example.net                57.498ms
7:  hop-2.example.net                56.160ms asymm  8
8:  hop-2.example.net                70.938ms asymm  7
9:  hop-2.example.net                71.480ms asymm  8
10: hop-2.example.net                80.652ms asymm  9
11: 192.168.0.1                      81.688ms reached
     Resume: pmtu 1500 hops 11 back 10
```

Links to more information

https://fasterdata.es.net/network-tuning/mtu-issues/
https://fasterdata.es.net/network-tuning/mtu-issues/debugging-mtu-problems/

Additional examples and perfSONAR commands:

For the different Operating Systems, here are a few examples:

Linux:

```
ping -M do -s 8972 [destination]
```

OSX (The OSX ping command only allows for a max size of 8184):

```
ping -D -s 8184 [destination]
```

Windows:

```
ping -f -l 9000 [destination]
```

In addition, a few perfsonar commands for troubleshooting MTU:

```
[localhost]# pscheduler troubleshoot [destination]
Performing basic troubleshooting of localhost and [destination].

localhost:

  Measuring MTU... 65535 (Local)
  Looking for pScheduler... OK.
  Fetching API level... 4
  Checking clock... Unsynchronized  (See check against [destination])
  Exercising API... Status... Tests... Tools... OK.
  Fetching service status... OK.
  Checking services... ticker... scheduler... runner... archiver... OK.
  Idle test.... 8 seconds.... Checking archiving... OK.

[destination]:

  Measuring MTU... Unsafe or unknown: MTU along path drops from 9000 to 1500
```


This material is based upon work supported by the National Science Foundation under Grant No. 1826994
info@epoc.global

```
  Looking for pScheduler... OK.
  Fetching API level... 3
  Checking clock... OK.
  Exercising API... Status... Tests... Tools... OK.
  Idle test.... Failed.

The server never scheduled a run for the task.
```

For example RTT No-fragment shows 100% loss as jumbo frames cannot pass between the two hosts:

```
[user@ps.example.net ~]$ pscheduler task rtt --no-fragment --source ps.example.net
--dest ps.example.edu  --count 2 --length 8972
       Submitting task...
       Task URL:
       https://ps.example.net/pscheduler/tasks/fffc90f2-7fec-45df-9b63-9679f1a12101
       Running with tool 'ping'
       Fetching first run...

       Next scheduled run:
       https://ps.example.net/pscheduler/tasks/fffc90f2-7fec-45df-9b63-9679f1a12101/ru
       ns/620234e2-2b0a-4a1b-9c94-f9020f8f4f22
       Starts 2021-04-28T12:26:07-07 (~2 seconds)
       Ends   2021-04-28T12:26:15-07 (~7 seconds)
       Waiting for result...

       100% Packet Loss  RTT Min/Mean/Max/StdDev = Unknown/Unknown/Unknown/Unknown ms

       No further runs scheduled.
```

***** RTT without the no-fragment flag*****

```
[user@ps.example.net ~]$ pscheduler task rtt  --source ps.example.net --dest
ps.example.edu --count 2 --length 8972
       Submitting task...
       Task URL:
       https://ps.example.net/pscheduler/tasks/53d0dc87-ce6c-4869-8a83-4ee881ae8cb1
       Running with tool 'ping'
       Fetching first run...

       Next scheduled run:
       https://ps.example.net/pscheduler/tasks/53d0dc87-ce6c-4869-8a83-4ee881ae8cb1/ru
       ns/c89006cd-caf1-4861-9a37-0512b34c3efc
       Starts 2021-04-28T12:27:15-07 (~3 seconds)
       Ends   2021-04-28T12:27:23-07 (~7 seconds)
       Waiting for result...

       1 ps.example.edu (xyz.xyz.xyz.xyz)  8980 Bytes  TTL 53  RTT  51.3000 ms
       2 ps.example.edu (xyz.xyz.xyz.xyz)  8980 Bytes  TTL 53  RTT  51.3000 ms

       0% Packet Loss  RTT Min/Mean/Max/StdDev =
       51.301000/51.322000/51.343000/0.021000 ms
```


This material is based upon work supported by the National Science Foundation under Grant No. 1826994

```
No further runs scheduled.
```

Another example from perfSONAR is the task --tool tracepath trace option
For example:

```
[user@localhost ~]$ pscheduler task --tool tracepath trace --dest
ps.example.edu
Submitting task...
Task URL:
https://localhost/pscheduler/tasks/2dd46cce-1ec2-4f69-9c66-258aa8dfddf7
Running with tool 'tracepath'
Fetching first run...

Next scheduled run:
https://localhost/pscheduler/tasks/2dd46cce-1ec2-4f69-9c66-258aa8dfddf7/runs/ff
2e4cf2-a9d4-4202-ab99-3d9fa251f33d
Starts 2021-04-28T12:28:49-07 (~3 seconds)
Ends   2021-04-28T12:30:30-07 (~100 seconds)
Waiting for result...

1 localhost (127.0.0.1) mtu 9000 bytes
2 hop1.example.net (xyz.xyz.xyz.xyz) ASxyz 14.711 ms mtu 9000 bytes
 EXAMPLENET, US
3 hop2.example.net (xyz.xyz.xyz.xyz) ASxyz 27.061 ms mtu 9000 bytes
 EXAMPLENET, US
4 hop3.example.net (xyz.xyz.xyz.xyz) ASxyz 39.756 ms mtu 9000 bytes
 EXAMPLENET, US
5 hop4.example.net (xyz.xyz.xyz.xyz) ASxyz 44.759 ms mtu 9000 bytes
 EXAMPLENET, US
6 hop5.example.net (xyz.xyz.xyz.xyz) ASxyz  44.888 ms mtu 9000 bytes
 EXAMPLENET, US
7 hop6.example.net (xyz.xyz.xyz.xyz) ASxyz 49.389 ms mtu 9000 bytes
 EXAMPLENET, US
8 hop7.example.net (xyz.xyz.xyz.xyz) ASxyz AS10578 49.861 ms mtu 9000 bytes
EXAMPLENET, US
9 hop1.example.edu (xyz.xyz.xyz.xyz) ASxyz 51.636 ms mtu 9000 bytes
 GIGAPOP-EXAMPLE, US
10 hop2.example.edu (xyz.xyz.xyz.xyz) ASxyz 50.942 ms mtu 8996 bytes
 GIGAPOP-EXAMPLE, US
11 hop3.example.edu (xyz.xyz.xyz.xyz) ASxyz 50.949 ms mtu 1500 bytes
 GIGAPOP-EXAMPLE, US
12 ps.example.edu (xyz.xyz.xyz.xyz) ASxyz 51.045 ms mtu 1500 bytes
 EXAMPLE-UNIV, US
```


This material is based upon work supported by the National Science Foundation under Grant No. 1826994
info@epoc.global