Internet Performance and Troubleshooting Lab

Fragmentation of an IP packet

Report 1

Group 9

Giuseppe Bruno s319892

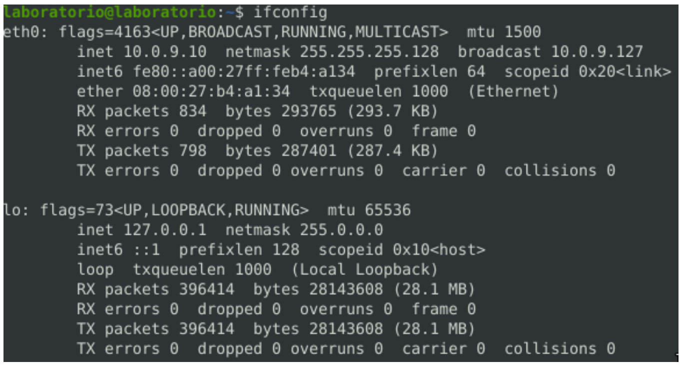
Salvatore Giarracca s314701

October 29, 2023

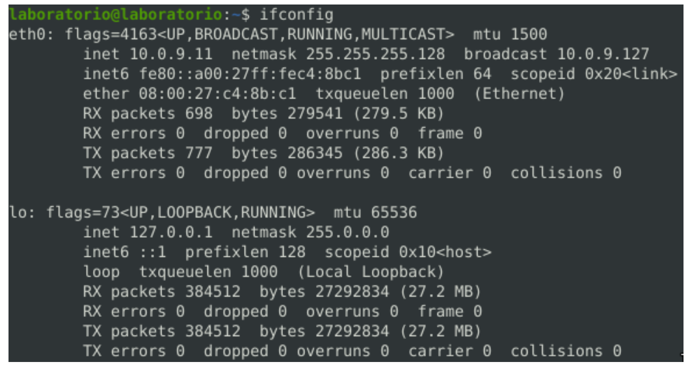
1 Introduction

1.1 Environment setup

Using VirtualBox, we have set up two virtual machines. The first one is named **Alice** with the IP address **10.0.9.10** (MTU 1500B), and the second one is **Bob** with the IP address **10.0.9.11** (MTU 1500B).



(a) Alice's configuration



(b) Bob's configuration

Figure 1: Environment

1.2 Maximum data before fragmentation

The Maximum Transmission Unit (MTU) represents the largest amount of data that can be accommodated in an Ethernet payload. Each Ethernet frame encapsulates an IPv4 packet, which consists of a 20−byte header ($IPv4_H$) and its corresponding payload. This payload contains the ICMP datagram, which is made up of an 8−byte header ($ICMP_H$) and its own payload that includes data from the ping application. The maximum amount of data that the ping application can send without causing fragmentation is calculated as follows:

$$\mathbf{MaxData = MTU - IPv4_H - ICMP_H}$$

In a default configuration where the MTU is set to 1500 bytes, **MaxData** equals to **1472 bytes**.

2 IP Fragmentation analysis

2.1 Fragments management

Fragmentation is performed by IP protocol, so there must be a way to rebuild all fragments into a single datagram that will be sent to upwards levels. To manage fragmentation, the following IPv4 header fields are required:

- **Identification**: Let the receiver to understand that each fragment belongs to the same datagram, it is 16-bit long and it is set by the sender
- **Total length**: The size of the carried data plus the IPv4 header (in Bytes)
- **MF**: (More Fragments) Set to 1 if there are more fragments or set to 0 if the fragment is the last of the sequence (or the unique packet)
- **DF**: (Don't Fragment) Set to 1 if it is allowed to split the datagram in more fragments, otherwise it is set to 0
- **Fragment Offset**: Let the receiver to understand the position of the data inside the whole IPv4 datagram, it is 13-bit long and is represented in octets

The sender splits the datagram in more fragments and copies the **Identification** of the original datagram into all the generated fragments, he sets the $\mathbf{MF = 1}$ into all the fragments but the last one, then he sets **Total length** that corresponds to the IPv4 header plus the payload of the fragment and also the **Fragment offset** that corresponds to the position of the fragments data into the original datagram.
When a router receives a packet can discard or forward it, depending on the value of **DF** (Don't fragment bit): if it is set to 0, then a packet bigger than MTU can be split in more fragments, otherwise, if is set to 1, it means that the packet cannot be fragmented and if it is bigger than the MTU, it will be discarded.

2.1.1 Note:

The intermediate routers can only forward packets, so they not reassemble them, but it is possible a situation in which they can fragment a packet (that is a fragment itself) if the next MTU is lower than the Total Length of the the processed packet. We will analyze this case in another section.

2.2 Fragments reassembly

When the receiver receives a packet, he can understand if it is part of a fragmented datagram or not looking at **MF**, **DF** and **Fragment offset**:

- $[\mathbf{MF = 0, DF = 1, FO = 0}]$: identifies an entire (not fragmented) datagram
- $[\mathbf{MF = 1, DF = 0, FO = 0}]$: identifies the first fragment of a larger datagram
- $[\mathbf{MF = 0, DF = 0, FO! = 0}]$: identifies the last fragment of a larger datagram

When fragmentation occurs, the receiver collects all the fragments into a buffer identified by **Identification** field common to all the headers. Because of the fact that they could not arrive in the same order as they were sent, it is crucial the role of **Total Length** and **Fragment Offset**. With those two fields, the receiver could build the original datagram putting each fragment in the right position (Figure 2). When the last fragment arrives, he could compute the length of "to be received" data. Here, the receiver can understand if all the fragment are arrived or if someone was lost during the transmission. If so, all the resources to perform the reassembly will be released.

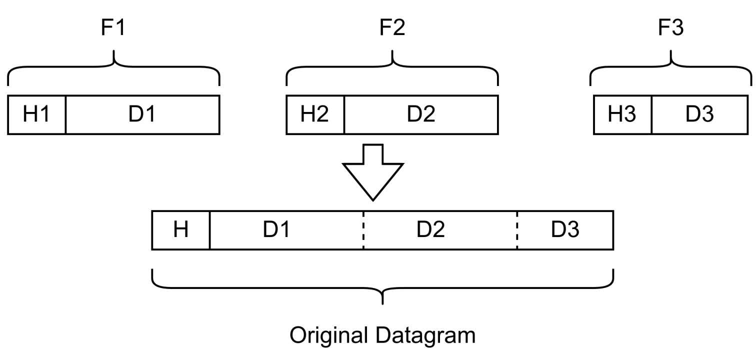


Figure 2: Fragment Reassembly

2.3 Fragmentation of an ICMP datagram: Both hosts MTUs set to 1500B

2.3.1 Alice pings Bob [`ping -c 1 -s 3500 10.0.9.11`]

Alice generates an ICMP echo request with a payload size of **3500B** and append the ICMP header (**8B**). As expected, IP split the original datagram into multiple fragments: the first two having **20B** of IPv4 header and a payload of **1480B**, and the last one having **20B** of IPv4 header and a payload of **548B**.

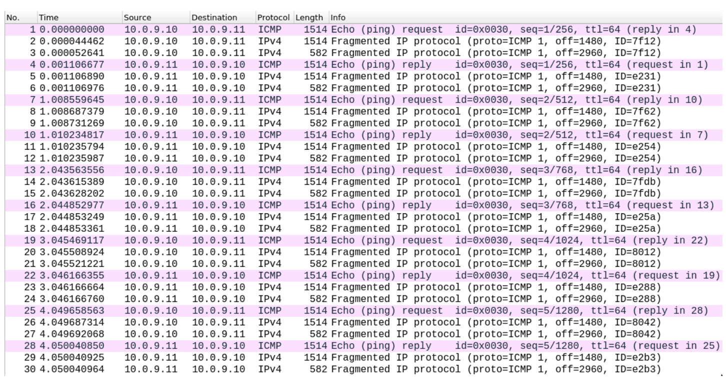
Packet	Identification	Total length	MF	Fragment offset
1	0x8545	1500	1	0
2	0x8545	1500	1	**185**(1480)
3	0x8545	568	0	**370**(2960)

2.3.2 Bob receives the ICMP echo request

When Bob receive the three fragments, he understand that are part of a bigger datagram and perform the reassembly. After the reconstruction, Bob knows that it is an ICMP echo request reading at the ICMP header and prepare the response for Alice. In this case both MTU's of sender and receiver are the same, so the ICMP echo reply will be encapsulated into an IPv4 packet and fragmented in the same way as Alice did with the ICMP echo request.

3 Fragments order

Using **Wireshark** tool we can see in which order the fragments of a big datagram are sent. In our configuration, the two hosts are connected directly and it is possible to observe the result of [`ping 10.0.9.11 -c 5 -s 3500`] performed by Alice to Bob. Looking at the second column (figure 3) we can observe the time in seconds since beginning the capture. The time value is in ascending order, according the number of packets and the fragments for each ICMP Echo request/reply.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.9.10	10.0.9.11	ICMP	1514	Echo (ping) request id=0x0030, seq=1/256, ttl=64 (reply in 4)
2	0.000044462	10.0.9.10	10.0.9.11	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=7f12)
3	0.000052641	10.0.9.10	10.0.9.11	IPv4	582	Fragmented IP protocol (proto=ICMP 1, off=2960, ID=7f12)
4	0.001106677	10.0.9.11	10.0.9.10	ICMP	1514	Echo (ping) reply id=0x0030, seq=1/256, ttl=64 (request in 1)
5	0.001106890	10.0.9.11	10.0.9.10	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=e231)
6	0.001106976	10.0.9.11	10.0.9.10	IPv4	582	Fragmented IP protocol (proto=ICMP 1, off=2960, ID=e231)
7	1.008559645	10.0.9.10	10.0.9.11	ICMP	1514	Echo (ping) request id=0x0030, seq=2/512, ttl=64 (reply in 10)
8	1.008687379	10.0.9.10	10.0.9.11	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=7f62)
9	1.008731269	10.0.9.10	10.0.9.11	IPv4	582	Fragmented IP protocol (proto=ICMP 1, off=2960, ID=7f62)
10	1.010234817	10.0.9.11	10.0.9.10	ICMP	1514	Echo (ping) reply id=0x0030, seq=2/512, ttl=64 (request in 7)
11	1.010235794	10.0.9.11	10.0.9.10	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=e254)
12	1.010235987	10.0.9.11	10.0.9.10	IPv4	582	Fragmented IP protocol (proto=ICMP 1, off=2960, ID=e254)
13	2.043563556	10.0.9.10	10.0.9.11	ICMP	1514	Echo (ping) request id=0x0030, seq=3/768, ttl=64 (reply in 16)
14	2.043615389	10.0.9.10	10.0.9.11	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=7fdb)
15	2.043628202	10.0.9.10	10.0.9.11	IPv4	582	Fragmented IP protocol (proto=ICMP 1, off=2960, ID=7fdb)
16	2.044852977	10.0.9.11	10.0.9.10	ICMP	1514	Echo (ping) reply id=0x0030, seq=3/768, ttl=64 (request in 13)
17	2.044853249	10.0.9.11	10.0.9.10	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=e25a)
18	2.044853361	10.0.9.11	10.0.9.10	IPv4	582	Fragmented IP protocol (proto=ICMP 1, off=2960, ID=e25a)
19	3.045469117	10.0.9.10	10.0.9.11	ICMP	1514	Echo (ping) request id=0x0030, seq=4/1024, ttl=64 (reply in 22)
20	3.045508924	10.0.9.10	10.0.9.11	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=8012)
21	3.045521221	10.0.9.10	10.0.9.11	IPv4	582	Fragmented IP protocol (proto=ICMP 1, off=2960, ID=8012)
22	3.046166355	10.0.9.11	10.0.9.10	ICMP	1514	Echo (ping) reply id=0x0030, seq=4/1024, ttl=64 (request in 19)
23	3.046166664	10.0.9.11	10.0.9.10	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=e288)
24	3.046166760	10.0.9.11	10.0.9.10	IPv4	582	Fragmented IP protocol (proto=ICMP 1, off=2960, ID=e288)
25	4.049658563	10.0.9.10	10.0.9.11	ICMP	1514	Echo (ping) request id=0x0030, seq=5/1280, ttl=64 (reply in 28)
26	4.049687314	10.0.9.10	10.0.9.11	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=8042)
27	4.049692068	10.0.9.10	10.0.9.11	IPv4	582	Fragmented IP protocol (proto=ICMP 1, off=2960, ID=8042)
28	4.050040850	10.0.9.11	10.0.9.10	ICMP	1514	Echo (ping) reply id=0x0030, seq=5/1280, ttl=64 (request in 25)
29	4.050040925	10.0.9.11	10.0.9.10	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=e2b3)
30	4.050040964	10.0.9.11	10.0.9.10	IPv4	582	Fragmented IP protocol (proto=ICMP 1, off=2960, ID=e2b3)

Figure 3: Fragments order (Sender)

4 Example algorithm for reassembly a datagram

```
def all_set(lower, upper, byte_table):
    all_unos = bytearray([0xFF] * upper)
    if (byte_table[lower:upper] == all_unos)
        return true
    return false

for f in fragments:
    bufid = f.source|f.dest|f.protocol|f.identfication
    # only one fragment
    if (f.header.FO==0 && f.header.MF==0):
        # if resources have been allocated
        if(resources(bufid)):
            flush_resources(bufid)
            submit_to_L4(f)
            break
    # if resources have not been allocated
    if (!resources(bufid)):
        (data_buf, header_buf, received_bmp, TL, timer) = allocate_resources(bufid)
        timer_lower_bound = 0
        timer = timer_lower_bound
    data_lower = f.header.FO * 8
    data_upper = f.header.TL - (len(f.header)) + f.header.FO * 8
    data_buf[data_lower:data_upper] = f.payload
    octet_lower = f.header.FO
    octet_upper = f.header.FO + ((f.header.TL - (len(f.header)) + 7) // 8)
    set(received_bmp[octet_lower:octet_upper])
    # if f is last fragment -> set total_len
    if (f.header.MF == 0): total_data_len = f.header.TL - (len(f.header)) + f.header.FO * 8
    # if f is first fragment -> put f.header into header_buf
    if (f.header.FO == 0): header_buf = f.header
    bmp_lower = 0
    bmp_upper = (total_data_len + 7) // 8
    # if all the fragments have been received
    if (total_data_len != 0 && all_set(bmp_lower, bmp_upper, received_bmp)):
        TL = total_data_len + (len(f.header))
        submit_to_L4(f)
        flush_resources(bufid)
        break
    timer = max(timer, f.header_TTL)
    # wait for next fragment or timer expiration
    wait(next() || timer)
    # free memory if timer expire
    if (timer.expired()):
        flush_resources(bufid)
```

In the pseudo-code snippet above it is shown a possible algorithm to perform the reassembly. The fragments could arrive out-of-order, for this reason they are buffered to allow reordering before reconstruct the original datagram. By reordering the fragments we mean that each fragment is put into the right place inside the **data_buf** by using the correspondent **TL** and **FO**, so no sorting algorithm are used due to the cost of the (sort) operation. When the first fragment is available, its header is put into the **header_buf**, while when last fragment is received the **total_data_len** is set. To test if all fragment are arrived **received_bmp** is used: this is a byte map that says if each octet of data is arrived. If each byte is set to one inside this map and the total_data_len is not 0, then all the fragments are arrived and the reconstructed datagram is sent to the upper layer. The **timer** allow to wait a fixed amount of time (usually 30 seconds for UNIX systems) before drop the packet and release all the allocated resources.

4.1 Possible attacks against IP fragmentation

The IP (alone) is not a very secure protocol because it relies only to IP addresses to authenticate the two peers that are communicating. This means that is very easy to impersonate a malicious person in an IP communication, for example by IP spoofing. An easy attack to IP fragmentation could be a Denial of Service, caused by an active Man In The Middle that could intercepts the traffic of two hosts and intentionally delay some of the fragments. Another possible attack could exploit the resource allocation at the receiver: the attacker can send many fragmented datagrams, but the final fragment and eventually slowing down the send rate in order to make the timer (at the receiver) almost expire. This could have a huge impact to the target host that could easily saturate its resources.

5 Custom experiments

5.1 Fragmentation of an ICMP datagram: hosts use different MTU's

5.1.1 Alice pings Bob [`ping -c 1 -s 2800 10.0.9.11`]

Alice (**MTU = 1500**) generates an ICMP echo request with a payload size of **2800B** and append the ICMP header (**8B**). As expected, IP split the original datagram into 2 fragments: the first one having **20B** of IPv4 header and a payload of **1480B**, and the last one having **20B** of IPv4 header and a payload of **1328B**.

Packet	Identification	Total length	MF	Fragment offset
1	0x83ab	1500	1	0
2	0x83ab	1348	1	**185**(1480)

5.1.2 Bob receives the ICMP echo request

Bob (**MTU = 1000**) generates an ICMP echo reply with a payload size of **2800B** and append the ICMP header (**8B**). IP splits the original datagram into 3 fragments: the first one having **20B** of IPv4 header and a payload of **976**, the second one having **20B** of IPv4 header and a payload of **976B** and the last one having **20B** of IPv4 header and a payload of **856B**.

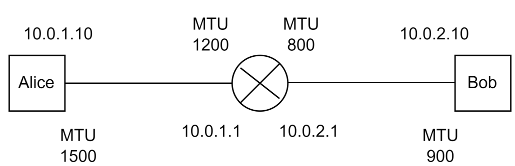
Packet	Identification	Total length	MF	Fragment offset
1	0x430b	996	1	0
2	0x430b	996	1	**122**(976)
3	0x430b	876	0	**244**(1952)

Important note: As a consequence of Bob's MTU (= **1000**), we expect a **Total Length** equal to 1000 in the first two fragments, but we observe that the real **TL** is equal to **996**. This is due to the fact that the **Fragment Offset** should be a multiple of **8**. This because every time an host fragments a datagram, it puts into the FO the offset divided by 8 and, when another host receive the fragment, it multiply the value inside the F0 by 8. This operation is performed to reduce the size of FO field (13 bytes) but without loosing the possibility to have the max number of fragments allowed by IP protocol ($2^{16} = 2^{13} * 2^{8}$).

5.1.3 Maximum supported MTU

With our configuration, the maximum MTU supported by our virtual network card is **16110B**

5.2 Router with two interfaces



5.2.1 Path with different MTU's

In this configuration we set up two hosts in two different LANs that are connected by a router having two interfaces. Alice can reach Bob's lan through router's **eth0** interface (**10.0.1.1**) and Bob can reach Alice through router's **eth1** interface (**10.0.2.1**). In the figure 4 we can see the result of [`ping -c 2 -s 2000 10.0.2.10`].

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.1.10	10.0.2.10	ICMP	1514	Echo (ping) request id=0x0066, seq=1/256, ttl=64 (reply in 4)
2	0.000043775	10.0.1.10	10.0.2.10	IPv4	562	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=9414)
4	0.012961418	10.0.2.10	10.0.1.10	ICMP	914	Echo (ping) reply id=0x0066, seq=1/256, ttl=63 (request in 1)
5	0.012962160	10.0.2.10	10.0.1.10	IPv4	914	Fragmented IP protocol (proto=ICMP 1, off=880, ID=5438)
6	0.012962274	10.0.2.10	10.0.1.10	IPv4	282	Fragmented IP protocol (proto=ICMP 1, off=1760, ID=5438)
7	1.001545425	10.0.1.10	10.0.2.10	ICMP	1514	Echo (ping) request id=0x0066, seq=2/512, ttl=64 (reply in 9)
8	1.001595920	10.0.1.10	10.0.2.10	IPv4	562	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=94b1)
9	1.003468716	10.0.2.10	10.0.1.10	ICMP	914	Echo (ping) reply id=0x0066, seq=2/512, ttl=63 (request in 7)
	1.003468946	10.0.2.10	10.0.1.10	IPv4	914	Fragmented IP protocol (proto=ICMP 1, off=880, ID=54e6)
	1.003469028	10.0.2.10	10.0.1.10	IPv4	282	Fragmented IP protocol (proto=ICMP 1, off=1760, ID=54e6)

(a) Alice

No.	Time	Source	Destination	Protocol	Length	Info
3	0.000822926	10.0.1.10	10.0.2.10	ICMP	810	Echo (ping) request id=0x0066, seq=1/256, ttl=63 (reply in 6)
4	0.010879677	10.0.1.10	10.0.2.10	IPv4	738	Fragmented IP protocol (proto=ICMP 1, off=776, ID=9414)
5	0.010880320	10.0.1.10	10.0.2.10	IPv4	562	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=9414)
6	0.010949108	10.0.2.10	10.0.1.10	ICMP	914	Echo (ping) reply id=0x0066, seq=1/256, ttl=64 (request in 3)
7	0.010979491	10.0.2.10	10.0.1.10	IPv4	914	Fragmented IP protocol (proto=ICMP 1, off=880, ID=5438)
8	0.010994255	10.0.2.10	10.0.1.10	IPv4	282	Fragmented IP protocol (proto=ICMP 1, off=1760, ID=5438)
9	1.001458403	10.0.1.10	10.0.2.10	ICMP	810	Echo (ping) request id=0x0066, seq=2/512, ttl=63 (reply in 12)
	1.001458768	10.0.1.10	10.0.2.10	IPv4	738	Fragmented IP protocol (proto=ICMP 1, off=776, ID=94b1)
	1.001458880	10.0.1.10	10.0.2.10	IPv4	562	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=94b1)
	1.001526474	10.0.2.10	10.0.1.10	ICMP	914	Echo (ping) reply id=0x0066, seq=2/512, ttl=64 (request in 9)
	1.001553126	10.0.2.10	10.0.1.10	IPv4	914	Fragmented IP protocol (proto=ICMP 1, off=880, ID=54e6)
	1.001563897	10.0.2.10	10.0.1.10	IPv4	282	Fragmented IP protocol (proto=ICMP 1, off=1760, ID=54e6)

(b) Bob

Figure 4: Alice pings Bob

Linux command "ping" option -M

Breaking a packet into smaller fragments can indeed lead to increased network congestion, as each fragment must be processed individually by the network devices. Not all devices along the route may have the necessary resources to manage a large number of packets effectively. Therefore, the optimal solution is to minimize the number of fragments for each packet. The option **-M** of ping command allow the host to choose the path MTU discovery strategy to know the minimum MTU in a path. This command supports three possible "hints": **dont**, **want**, **do**.

DONT

By choosing it, the **DF** (Don't Fragment) flag in the packet is not set, allowing it to be fragmented (if necessary) without any optimization. The packets still reach their destination, but resource optimization is not prioritized. Suppose a router receives two packets, one with a size of **1500B** and the other with a size of **200B**, while its MTU is set to **1000B**. In the outgoing traffic, you would have 3 packets (the first one divided into 2), when resource optimization could reduce it to only 2 packets (approximately **1000B** and **700B** in size).

DO

By choosing it, it the **DF** flag is set, so the packet can't be fragmented. If there is an MTU smaller than it's size, the packet is dropped and to the host is sent an ICMP Fragmentation Needed (Type 3, Code 4). The scope of this solution is to let the host to know the minimum MTU in a specified Path in a statistic way. Every time there is a Fragmentation Needed the sender must change manually the dimension of the IP payload until it can communicate correctly with the receiver.

```
laboratorio@laboratorio:~$ ping -c 1 -s 1400 -M do 10.0.2.10
PING 10.0.2.10 (10.0.2.10) 1400(1428) bytes of data.
From 10.0.1.1 icmp_seq=1 Frag needed and DF set (mtu = 800)

--- 10.0.2.10 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms
```

Figure 5: Do option

WANT

By choosing it, the **DF** flag is set, so the packet can't be fragmented. If there is an MTU smaller than it's size, the packet is dropped and the host receives an ICMP Fragmentation Needed. The difference with the previous one is that in this case the sender changes itself the size of the IP payload, such that next time has the correct dimension.
The image above shows [`ping 10.0.2.10 -c 3 -s 1400 -M want`] between two hosts connected by a router, where the outgoing MTU to the receiver is set to **700B**. The first request is blocked because there is a MTU smaller than **1400B**. However, the subsequent requests already have packets of maximum size equal to the minimum known MTU up to that point.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.1.10	224.0.0.…	MDNS	183	Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR _ftp._tcp.loc…
5	12.977087366	10.0.1.10	10.0.2.10	ICMP	1442	Echo (ping) request id=0x0064, seq=1/256, ttl=64 (no response found!)
6	12.977689312	10.0.1.1	10.0.1.10	ICMP	590	Destination unreachable (Fragmentation needed)
7	13.977400852	10.0.1.10	10.0.2.10	ICMP	714	Echo (ping) request id=0x0064, seq=2/512, ttl=64 (reply in 10)
8	13.977465455	10.0.1.10	10.0.2.10	IPv4	714	Fragmented IP protocol (proto=ICMP 1, off=680, ID=432e)
9	13.977476767	10.0.1.10	10.0.2.10	IPv4	82	Fragmented IP protocol (proto=ICMP 1, off=1360, ID=432e)
	13.980144527	10.0.2.10	10.0.1.10	ICMP	914	Echo (ping) reply id=0x0064, seq=2/512, ttl=63 (request in 7)
	13.980144899	10.0.2.10	10.0.1.10	IPv4	562	Fragmented IP protocol (proto=ICMP 1, off=880, ID=b1c5)
	14.978937793	10.0.1.10	10.0.2.10	ICMP	714	Echo (ping) request id=0x0064, seq=3/768, ttl=64 (reply in 15)
	14.978985451	10.0.1.10	10.0.2.10	IPv4	714	Fragmented IP protocol (proto=ICMP 1, off=680, ID=4384)
	14.978993976	10.0.1.10	10.0.2.10	IPv4	82	Fragmented IP protocol (proto=ICMP 1, off=1360, ID=4384)
	14.980897307	10.0.2.10	10.0.1.10	ICMP	914	Echo (ping) reply id=0x0064, seq=3/768, ttl=63 (request in 12)
	14.980897535	10.0.2.10	10.0.1.10	IPv4	562	Fragmented IP protocol (proto=ICMP 1, off=880, ID=b1e3)

Figure 6: Want option