Practice Lab 2: IP Fragmentation and Reassembly

Preliminary Reading: *Read section* ***"IPv4 Datagram Fragmentation"*** *in the recommended course textbook (Kurose). Depending on the edition, it is located in different parts of Chapter 4. The* ***5th and 7th editions in Spanish*** *are recommended (available in the University library). In the* ***5th edition****, it is in* ***Section 4.4.1****, and in the* ***7th edition****, in* ***Section 4.3.2****. The* ***8th edition*** *does not cover this topic, as* ***IPv6 does not allow fragmentation in routers****, only at the source, which will fragment the original datagram if necessary, in a manner similar to what is described in this lab for IPv4.*

1. Introduction

In this lab, we will study the issue of **IPv4 datagram fragmentation**. As discussed in the theory sessions, although the theoretical maximum size of an IP datagram is **64 KB**, in practice, smaller datagrams are transmitted. This is because, for transmission, the datagram must be encapsulated within a **data link layer frame**, occupying its data field in the same way that application data is transported within the **data field of TCP segments**. Therefore, the size of the datagram is limited by the **maximum data field size** of the frame that will carry it. This value depends on the **network technology** in use. Most network technologies define maximum sizes, also known as **MTUs (Maximum Transfer Unit)**. For example, **Ethernet** defines an **MTU of 1,500 bytes**, **PPPoE** has an **MTU of 1,492 bytes**, and **FDDI** supports **4,470 bytes**

In **Unit 6** of our Computer Networks course, we will study the **Ethernet frame format** in detail. However, we can highlight the key aspects relevant to this lab. As shown in **Figure 1**, aside from **preambles and start/end delimiters**, the **Ethernet frame** has a maximum size of **1,518 bytes**. Subtracting the **12 bytes** occupied by the **source and destination physical addresses**, the **2-byte** field indicating the **embedded protocol type** (e.g., **0x800** means the data contains an **IP datagram**), and the **4-byte CRC**, we are left with:

1,518 – 18 = 1,500 bytes (maximum) available for data.

This **data field** will contain the **IP datagram** (see **Figure 2**). Therefore, the **maximum datagram length (MTU)** for an **Ethernet frame** cannot exceed **1,500 bytes**.

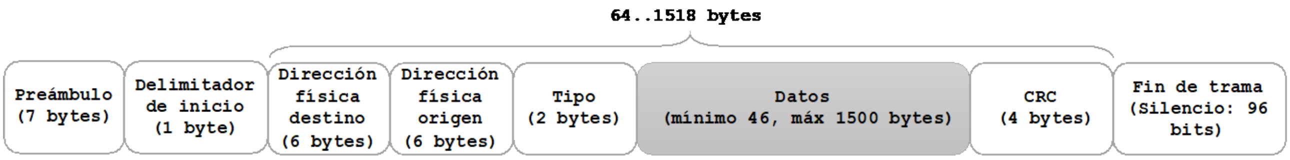

Figura 1. Ethernet frame format

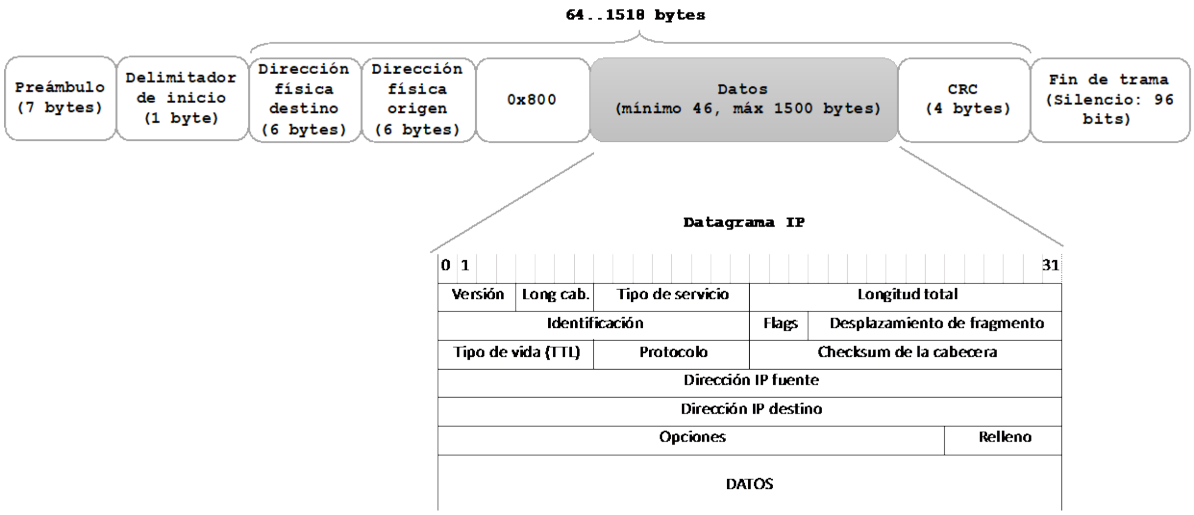

Figure 2. IP Datagram in Ethernet frame

The concept of **MTU (Maximum Transfer Unit)** is similar to the already known **MSS (Maximum Segment Size)** at the transport layer. Do you remember that, when we studied **TCP** and did lab exercises, we obtained an **MSS of 1460 bytes**? Do you now understand why? Think about this aspect, considering that both the **IP and TCP headers** (without options) occupy **20 bytes each**. Looking at **Figure 2**, reflect on what information is encapsulated in the **IP datagram's data field**.

When using **TCP**, the **maximum segment size** is chosen so that the resulting **IP datagram** fits within the **data field of the frame** in which it will be encapsulated. Unfortunately, even with this precaution, the **datagram** may still need to be **fragmented** into smaller pieces if it must pass through a network, on its way to the destination, with an **MTU smaller** than that of the original network. The **router** that separates the two networks will handle this task before forwarding the **datagram** to the **next network**. Subsequently, each fragment travels separately until it reaches the **destination device**, which must **reassemble the original datagram** once all fragments have been received.

These ideas are illustrated in **Figure 3**. The datagram that **Device A** wants to send to **Device B** is generated with a **size** that matches the **MTU** of the network it is on (**Network 1**). During its journey, when it reaches the **first router**, the router detects that **Network 2** has an **MTU smaller** than the **datagram size** it just received. At that moment, the router determines that it must **fragment** the **datagram** to comply with **Network 2's restrictions**.

For simplicity, we can assume that the original **datagram** will be **split into two** because its original size is around **900 or 1000 bytes**. (**If the original size were the maximum, it would need to be fragmented into three datagrams.**) In this way, a **first datagram** will be generated with up to **576 bytes** of the original data (we will specify this later), and the remaining data will be placed in a **second datagram**.

Each of these **datagrams** will have its **own header**, adjusted according to the information it carries, its **total length, checksum**, and the fields related to **fragmentation**. The **header colors** have been changed to emphasize that they are **not identical**.

As mentioned before, once a datagram is **fragmented**, it is **not reassembled** until all its fragments arrive at the **destination**. As shown in the **figure**, even though **Network 3** has a **larger MTU**, the **datagrams remain fragmented** until they reach **Device B**, where they are finally **reassembled**.

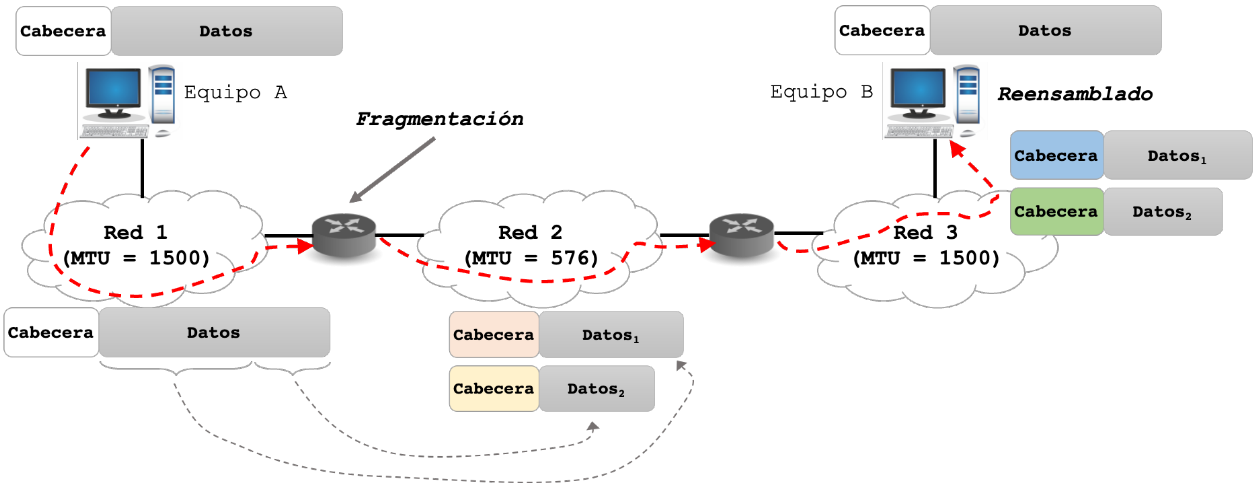

Figura 3. Datagram fragmentation when changing the MTU of Network 2.

When using protocols other than **TCP**, such as **UDP** or **ICMP**, fragmentation issues can arise **even at the source host** since **UDP and ICMP do not consider the MTU** when generating their data units.

All **IP implementations** must support **datagrams up to 576 bytes** (whether they arrive complete or fragmented). However, most can handle **larger values**, typically **above 8192 bytes or even higher**.

2. IPv4 Fragmentation

In **IPv4**, some fields in the **datagram header** are involved in the **fragmentation process**. These fields are highlighted in the **datagram format** shown in **Figure 4**.

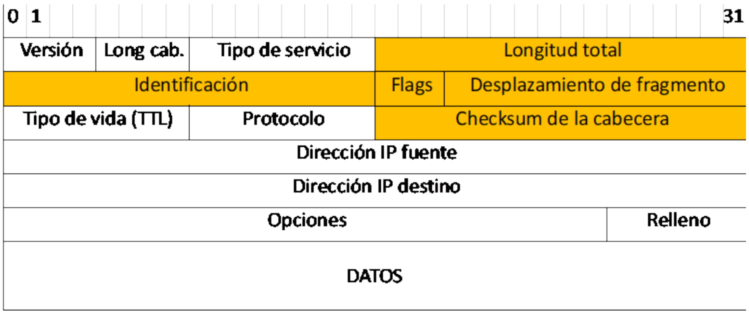

Figure 4.IPv4 datagram.

- **Total Length Field**: Defines the total size of the **datagram (header + data)** in bytes. If fragmentation is required, this field will indicate the **fragment size**.
- **Identification Field**: A **16-bit integer** that uniquely identifies each **datagram** transmitted by a host, tagging the **original datagram**. It allows fragments to be identified as part of the same datagram since all **fragments inherit the original datagram's identifier**.
- **Flags**: This field consists of **three bits**, though the most significant bit is not used. The remaining two bits specify conditions related to **packet fragmentation**:
  - **Do Not Fragment (DF)**: When set to **1**, it indicates that the **datagram cannot be fragmented**. If forwarding an IP packet with this bit enabled requires fragmentation, the packet will be **discarded** instead of being forwarded, and the **source** will be notified via an **ICMP message**.
  - **More Fragments (MF)**: When set to **1**, it indicates that **this fragment is not the last** in the sequence. This bit will be **set for all fragments except the last one,** and it is used at the **final destination** during **reassembly**.
- **Fragment Offset**: A **13-bit field** that indicates the **position of the fragment** within the **original datagram**. Since this field (13 bits) is **three bits shorter** than the **Total Length field (16 bits)**, the **data offset is expressed in multiples of 8 bytes** (i.e., **64-bit blocks**). This means that a **fragment's data field (except the last one) must be a multiple of 8 bytes** to ensure the correct offset of the next fragment. The **last fragment** does **not** need to meet this size restriction, as there are no subsequent fragments, and it is marked with **MF = 0**. The **first fragment** has an **offset of zero** and **MF = 1**.
- **Header Checksum**: Ensures **error detection** in the **datagram header**. It is recalculated whenever a **node modifies any fields** (e.g., **Time to Live (TTL)**). After **fragmentation**, multiple **header fields** are modified, requiring the **checksum** to be recalculated.

It may be necessary to **refragment an already fragmented datagram** if it traverses another network with a **smaller MTU**. In this case, the **offset of all fragments** refers to the **original datagram**.

Reassembly Process:

Reassembly always occurs at the receiver, requiring **all fragments to arrive within a limited time** before a **timer expires**. The **timer starts when the first fragment is received** (whichever arrives first, even if it is not the **zero-offset fragment**). If the **timer expires**, all received fragments are **discarded**. If necessary, the **upper-layer protocol** (e.g., **TCP**) may request a **retransmission**, requiring the **entire datagram to be resent**.

Fragmentation example:

Given the **network topology** shown in **Figure 5**, suppose **Host A** wants to send a **datagram** with a **total length of 1620 bytes**. Since **Network 1** supports **datagrams of this size (or larger)**, it generates a **single datagram** of that length (the so-called **"original" datagram**).

When this **datagram reaches Router 1**, the router must **fragment it**, as the **maximum frame size** allowed in **Network 2** is **smaller**. As a result, the **original datagram is divided into three fragments**. While **Network 1** carried a **single 1620-byte datagram**, **Network 2** will transport **three fragments** of that datagram:

- **Two fragments of 620 bytes each**
- **One last fragment of 420 bytes**

Next, we will analyse how to calculate the **length of each fragment**.

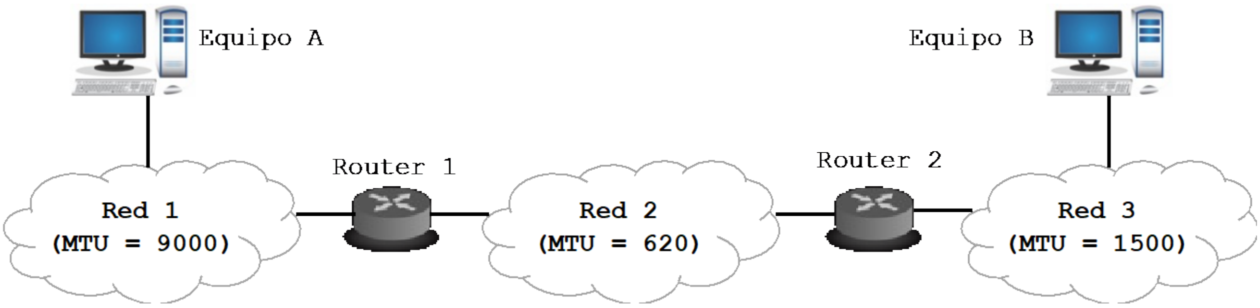

Figure 5. Fragmentation example

Original Datagram	Total Length	Identification	DF	MF	Fragment Offset	Data
	1620	32	0	0	0	1600 bytes

The fragmentation performed by **Router 1** generates the following **datagrams**:

	Total Length	Identification	DF	MF	Fragment Offset	Data
Fragment 1	**620**	32	0	**1**	0	**Del byte 0 al 599**
Fragment 2	**620**	32	0	**1**	**75** (75×8 = 600)	**Del byte 600 al 1199**
Fragment 3	**420**	32	0	0	**150** (150×8 = 1200)	**Del byte 1200 a 1599**

When calculating the amount of **IP data** that fits within a frame, the following must be considered:

a) The **IP header** occupies **20 bytes**, assuming no options are included, which is the usual case. The remaining portion of the **MTU**, in this case, **620 – 20 = 600**, is what is available for **IP data**. In our example, the **original datagram** carried **1,600 bytes** of **IP data**, which must be **distributed into fragments** carrying a **maximum of 600**

bytes of IP data, provided the condition analysed in **point (b)** allows it.
The amount of data included in **each fragment**, except for the last one, **must be divisible by 8**, due to how the **fragment offset** is expressed. In this case, **600 ÷ 8 = 75**. Since **600 is divisible by 8**, everything aligns perfectly. Additionally, the **offset** will be a **multiple of 600** in the different fragments. However, the values that actually appear in the **IP header** of the fragments will be **multiples of 75**.

It is important to note that, when using **other typical MTU sizes**, such as **576**, everything does not always align as well. In this case, **576 – 20 = 556**, and **556 ÷ 8 = 69.5**, meaning it **is not divisible by 8**. Therefore, in such cases, we must use the **closest multiple of 8** that fits within the **maximum available size**. In this example, only **552 bytes (69 × 8)** out of the **556 bytes available** in the **MTU** can be used, ensuring an exact division. In a sequence of fragments, the **actual offset** would be a **multiple of 552**, but in the **offset field**, it would be expressed in **multiples of 69** (**69 × 8 = 552**).

Exercise 2

3480 bytes of data

A **router** receives a **datagram of 3,500 bytes**. The **outgoing network**, which must be used to forward the **datagram** to its destination, has an **MTU of 1,500 bytes**, meaning the **router must fragment the datagram**.

- **Calculate** the **number of fragments** generated and the **size of each fragment**. Include the **calculations performed** in your response.
- **Determine** the **fragment offset field value** in the **IP header** for each generated fragment. (**Remember that the data field size for all fragments, except the last one, must be divisible by 8**).
- **Complete the following table** with the obtained values.

Number of fragments	*Total length /fragment*	*Fragment Offset*	*MF Bit*
0	1500	0	1
1	1500	1480 / 8 = 185	1
2	3480 - 1480 * 2 + 20 = 540	1480 * 2 / 8 = 370	0

3. Traffic Analysis

Although we cannot directly observe **fragmentation occurring in routers**, we can use a simple trick to generate **fragmentation** on our own device.

As mentioned in the **introduction**, **ICMP and UDP protocols** do **not** consider the **local MTU size** when generating their **data units**: **ICMP packets** or **UDP datagrams**, respectively. In the **previous lab**, we studied the **ping command**, which allows us to send an **ICMP echo request** packet to a destination with a specified **ICMP data size,** and wait for the **associated response**. If the **total size** of the **ICMP packet** (header + data field) to be sent, plus the **IP header size**, **exceeds the local MTU**, the **IP layer** of our device will be forced to **fragment the datagram** containing the **ICMP packet**.

3.1 ICMP Echo Request and Reply Message Format

The **ping command** uses **ICMP Echo Request** and **ICMP Echo Reply** messages. Although the **ICMP protocol** will be studied in detail in **Lab 4**, we will now highlight the **basic concepts** needed to understand its use in this lab. The **format of these messages**, which are encapsulated in the **data field of the IP datagram**, is shown in **Figure 6**.

As seen in the figure, these are **simple messages** consisting of a **header and data**. The **header** specifies:

ICMP Message Type (Echo Request or Reply).

The **Code field**, which is **not used** in these messages.

A **Checksum** to detect errors, similar to those used in other protocols.

In the next **32-bit word**, the **Identifier** and **Sequence Number** fields allow tracking of sent and received messages. The **Identifier field** can be thought of as a number that **identifies the request**, while the **Sequence Number** is **incremented** with each new request sent to a destination. This allows the **sender** to clearly identify **which request** a received response corresponds to.

Finally, in the **Data field**, the sender places information that should be **returned** by the recipient in the **echo reply** (hence the name **"echo" request**). This field is typically filled with a few dozen bytes (each **operating system** has a **default size**: **32 bytes in Windows**, **56 bytes in macOS**, and **64 bytes in Linux**).

In summary, the **total length** of the message is:

8 bytes (two **32-bit words** forming the **header**)

+ the size of the data field used for the **echo request**

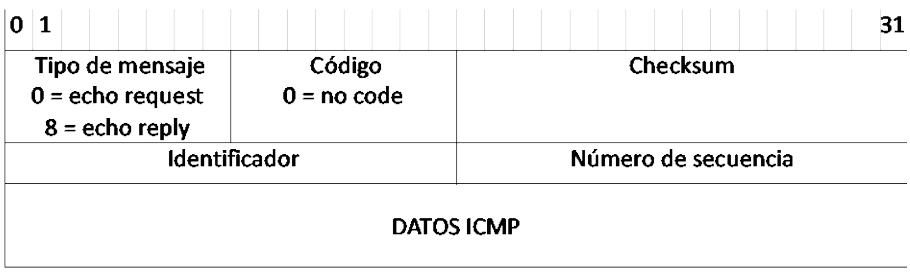

Figure 6. ICMP echo request and reply message format.

Encapsulation and Network Layers

Figure 7 shows the complete structure of the **different network layers involved in communication**.

Within the **Ethernet frame's data field** (which is later passed to the **physical layer** to be transmitted **bit by bit**), the **IP datagram** is encapsulated. The **Ethernet frame type field**, set to **0x800**, indicates that an **IP datagram** is encapsulated within the frame.

In this case, the **IP datagram** carries an **ICMP message**. As shown in the figure, the **IP header's Protocol field** is set to **1**, indicating that the **data field** contains an **ICMP message**. If the **protocol field** were set to **6**, it would indicate that the **data field** contains an embedded **TCP segment**.

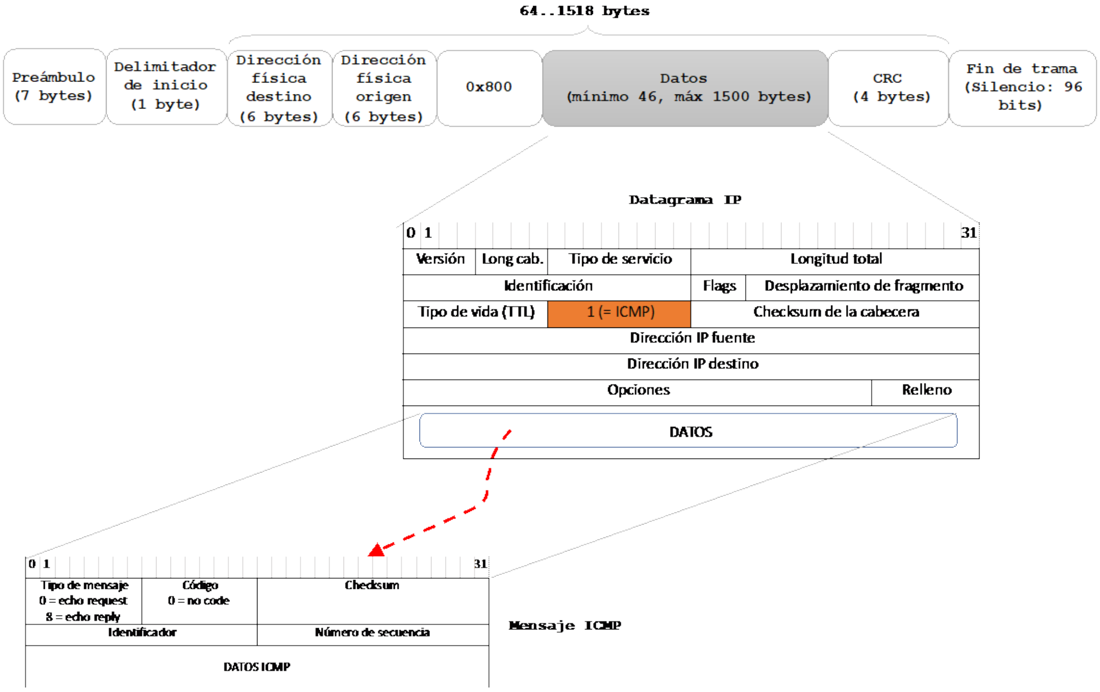

Figure 7. ICMP message – IP datagram – Ethernet frame

<u>Exercise 2</u>

We will prepare a **network traffic capture** using **Wireshark**.

1. Open **Wireshark** and apply a **filter** to display only **ICMP traffic** sent or received by your computer:

   Capture → Options → Capture filter for selected interfaces: icmp and host xx.xx.xx.xx

   You must replace *xx.xx.xx.xx* with your **machine's IP address**.

2. Remember that in the **previous lab**, we used the **ip address** command in **Linux** to find this information. Similarly, in **Windows**, you can use **ipconfig**

3. Once you have verified your **IP address**, start the **packet capture**.

4. Next, open a **Linux command prompt** and enter

`ping -c 1 -s 3972 www.rediris.es`

The equivalent command in **Windows** would be:

`C:\> ping -n 1 -l 3972 www.rediris.es`

5. After completing the **ping command**, **stop the packet capture** in **Wireshark**

You will obtain a **packet capture** similar to the one shown in Figure 8.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.00000…	158.42.180.23	130.206.13.20	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=d8f4) [Reassembled in #3]
2	0.00000…	158.42.180.23	130.206.13.20	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=d8f4) [Reassembled in #3]
3	0.00001…	158.42.180.23	130.206.13.20	ICMP	1054	Echo (ping) request id=0x0003, seq=1/256, ttl=64 (reply in 6)
4	0.00913…	130.206.13.20	158.42.180.23	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=03b6) [Reassembled in #6]
5	0.00913…	130.206.13.20	158.42.180.23	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=03b6) [Reassembled in #6]
6	0.00918…	130.206.13.20	158.42.180.23	ICMP	1054	Echo (ping) reply id=0x0003, seq=1/256, ttl=56 (request in 3)

Figure 8. echo ICMP message (pings) - Wireshark capture.

The option **-c 1** in **Linux and macOS** or **-n 1** in **Windows** ensures that only **one ICMP echo request** message is sent to the specified host. As will be seen in the **lab where the ICMP protocol is studied in detail**, the **ping command** by default sends packets **continuously**.

The option -s 3972 (Linux and macOS) or -l 3972 (Windows) indicates that the ICMP message includes a data field of 3972 bytes, whose content is not relevant. Keep in mind that the ICMP packet also has an 8-byte header, as mentioned earlier.

Next, the **ping command** displays information about the **response**, which will be an **ICMP echo reply** message.

Since we are connected to an Ethernet network (with an MTU of 1500 bytes), sending with -s 3972 will require the fragmentation of the packet into multiple IP packets.

Do not **misinterpret** what you see in **Wireshark**: there are **three fragments** in both the **request** and the **response**. **Wireshark** labels the **first two** as **"Fragmented IP protocol"**, and the **third (last one)** as the actual **"ping request"**, which was originally made up of **three fragments**. This is a **convenience feature** provided by **Wireshark** to make interpreting **captured frames** easier.

Analyse the three fragments in detail and answer the following questions:

a) For the **datagram sent by your computer**, compare the **headers of the generated fragments**, focusing especially on the **Total Length, Flags, and Fragment Offset fields** (shown as **Fragment Offset** in **Wireshark**). Use the following table to record these values.

Fragment Identifier	*Flag DF*	*Flag MF*	*Fragment Offset*	*Total Length*
18915	0	1	0	1500
18915	0	1	1480	1500
18915	0	0	2960	1040

b) What is the value of the **Protocol field** in the **header of all three fragments**? Should it be the **same for all fragments**? Justify your answer. 1 (ICMP), for all of them, as all are fragmented packets of an ICPM packet.

c) What is the value of the **Fragment Offset field** in the **IP header of the second fragment**? **Wireshark** displays the **calculated offset value**, not the actual **value sent**. Verify the **real value** sent by checking the **hexadecimal data** in the lower pane of **Wireshark**. Remember that the **Fragment Offset field** is **13 bits long**. Offset = 1480 bin(0 0000 1011 1001) = 185

d) **Calculate** the **message size** that should be sent to generate **four maximum-sized fragments**. For this calculation, take into account the **size of the ICMP header**. The **ICMP header length** can be determined by examining the size of each of its **fields** in the **lower pane** of the capture.

(1500 - 20) * 4 - 8 (ICMP header) = 5912

e) Verify that this **calculated message size** is **correct** by **capturing the generated traffic** after executing the **ping command** again, replacing **3972** with the **calculated message size**.

f) How many **IP data bytes** travel in **each packet**? What about **ICMP data bytes**? You can use the **"Header Length"** and **"Total Length"** fields of the **IP datagram header** to help with this calculation. IP data bytes / packet = 1480

ICMP data bytes for the first packet = 1480 - 8 (the first one contains the ICMP header),

= 1480 for the rest of the packets

Exercise 3

The **MTUs** of **Networks 1 and 2** are **4500 and 800 bytes**, respectively. On **Computer B** in **Network 2**, the following **IP datagrams** were received. The sender of these **datagrams** is **Computer A** from **Network 1**.

IP header fields				
Total length	***Identifier***	***DF***	***MF***	***Fragment offset***
796	16	0	0	194
40	28	0	0	194
796	16	0	1	0
796	28	0	1	0
780	63	0	0	0
796	16	0	1	97
796	95	0	1	291
796	28	0	1	97
54	95	0	0	388

a) Do the **received datagrams** have any **relation to each other**? Justify your answer.
b) Complete the **table** with the values of the **datagrams as they were sent by Computer A**.

Total length	***Identifier***	***Flag DF***	***Flag MF***	***Fragment offset***
796*3 - 20*2 = 2348	16	0	0	0
796*2 + 40 -20*2 = 1592	28	0	0	0
780	63	0	0	0
796*4 + 54 - 20*4 = 3158	95	0	0	0

For ID 95, the first 3 packets (with size 796) have been lost, but we have to take them into account here.

c) Will **all received datagrams** be delivered to the **upper layer**?

No, datagram 95 will not be delivered to the upper layer, as the first 3 packets have been lost.