

## Practical Exercises 2

**Surname:** Hejhal

**Name:** Jakub

**Degree** (Ing. del Software/Ing. Informática/Ing. Computadores): Erasmus

**Group** (A/B/C): B

**PC Label in the lab** (e.g. 012): personal laptop

**Exercise 1:** Observe the IP headers of the different datagrams. What is the protocol of the field “protocol” (from the IP header) of the ICMP, FTP and HTTP? Fill the following table with this information. What is the meaning of this field?

Protocol	Protocol field	Value (HEX)	Frame number
ICMP	ICMP	01	107680
HTTP	TCP	06	90600
FTP	TCP	06	94941

The IPv4 protocol field tells us the name of the encapsulated protocol

**Exercise 2.** Select an ICMP request (message *echo request*) and fill in the table with the IP destination address and the MAC destination address (i.e., the Ethernet header). Complete the same information for a FTP request (i.e., the info field contains "request"). Why are the addresses the same for one protocol and different for the other one?

	ICMP	FTP
IP Destination address (IP header)	150.214.54.249	150.214.40.67
MAC Destination Address (Ethernet header)	02:10:18:37:91:9c	02:10:18:37:91:9c
Frame number	107680	94957

Ethernet frames go through the same WiFi router at my home, so the WiFi router MAC is the same in both cases. The IP address is different because according to DNS, `informatica.cv.uma.es` and [ftp.uma.es](http://ftp.uma.es) have different IPs.

**Exercise 3.** What are the type and code of the ICMP messages (request/reply)? Focus on ICMP requests for the rest of the questions. What filter can you put to see only the fragments of a concrete IP datagram? Indicate in the following table the value of each flag of the fragment, its identification and offset (write these values for each packet size). Note: put the information of each fragment in a separate row, so add new rows when necessary.

Size	Fragments	Identifiers	Flags	Fragment Offsets
1200	1	4846	Reserved bit: 0, Don't fragment: 1, More fragments: 0	0
3100	3	4927	Reserved bit: 0, Don't fragment: 0, More fragments: 1 Reserved bit: 0, Don't fragment: 0, More fragments: 1 Reserved bit: 0, Don't fragment: 0, More fragments: 0	0,1480,2960

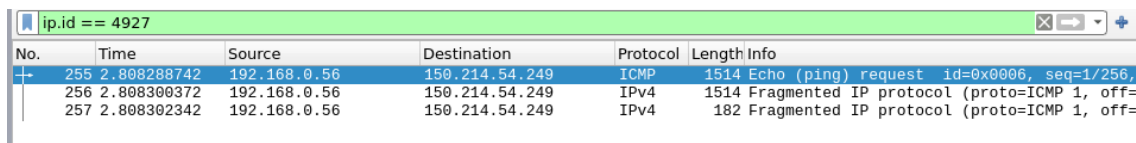
ICMP request type: 8

ICMP reply type: 0

ICMP request code: 0

ICMP reply code: 0

If I want to see only fragments of a IP datagram with id=4927, I will put in a filter ip.id == 4927



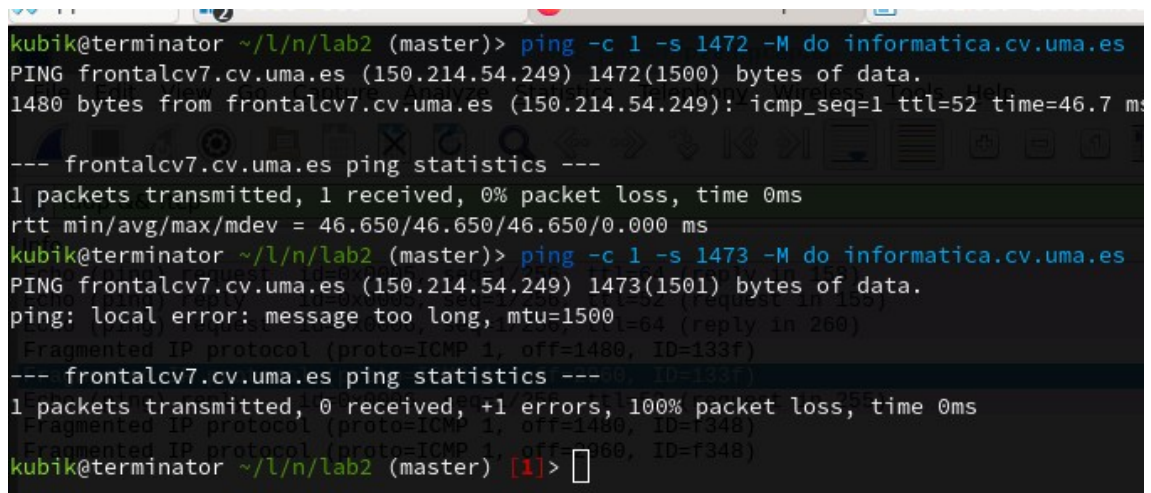
No.	Time	Source	Destination	Protocol	Length	Info
255	2.808288742	192.168.0.56	150.214.54.249	ICMP	1514	Echo (ping) request id=0x0006, seq=1/256.
256	2.808300372	192.168.0.56	150.214.54.249	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=
257	2.808302342	192.168.0.56	150.214.54.249	IPv4	182	Fragmented IP protocol (proto=ICMP 1, off=

**Exercise 4.** Calculate the maximum amount of user data that you can transmit in the network of the laboratory. Being MAX the maximum amount of user data that you have calculated, execute the ping command two times with the D flag set to 1 (**-f** option), one with a packet of length MAX and another with a packet of length MAX+1 (option **-l**). What is the value of MAX? Why? Does the second ping appear in the packet trace? Save the traffic capture as **p2e4.png** and take screenshots to show how the first ping succeed and the second one fail.

My WiFi card interface has MTU=1500

```
2: wlp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 40:a3:cc:2a:a6:d9 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.56/24 brd 192.168.0.255 scope global dynamic noprefixroute wlp2s0
        valid_lft 3219sec preferred_lft 3219sec
    inet6 fe80::7eb9:ad17:ad9e:4849/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
```

MAX = 1472 B, because together with 8 byte ICMP header + 20 byte IP header,  $1472 + 8 + 20 = 1500$



```
kubik@terminator ~/l/n/lab2 (master)> ping -c 1 -s 1472 -M do informatica.cv.uma.es
PING frontalcv7.cv.uma.es (150.214.54.249) 1472(1500) bytes of data.
1480 bytes from frontalcv7.cv.uma.es (150.214.54.249): icmp_seq=1 ttl=52 time=46.7 ms

--- frontalcv7.cv.uma.es ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 46.650/46.650/46.650/0.000 ms
kubik@terminator ~/l/n/lab2 (master)> ping -c 1 -s 1473 -M do informatica.cv.uma.es
PING frontalcv7.cv.uma.es (150.214.54.249) 1473(1501) bytes of data.
ping: local error: message too long, mtu=1500
Fragmented IP protocol (proto=ICMP 1, off=1480, ID=133f)
--- frontalcv7.cv.uma.es ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms
Fragmented IP protocol (proto=ICMP 1, off=1480, ID=f348)
Fragmented IP protocol (proto=ICMP 1, off=1480, ID=f348)
kubik@terminator ~/l/n/lab2 (master) [1]>
```

Second ping fails and doesn't appear in the packet trace, because the packet size exceeds MTU.

Btw, why should I save the traffic capture as PNG image (**p2e4.png**) ?

**Exercise 5.** Make successive pings to [informatica.cv.uma.es](http://informatica.cv.uma.es) starting with a TTL value of 1 and stopping as soon as a reply from the server arrives. Observe in Wireshark the message exchange that takes place during the execution of the pings. What type of ICMP message arrives? What type and code values does this ICMP message have? What is the additional information encapsulated in this ICMP message?

When ping fails, ICMP Time-to-live exceeded message is received.

Type: 11

Code: 0

After that, the packet contains the original IPv4 packet before just before exceeding the TTL (without the data field)

```

on_gateway (192.168.0.1) icmp_seq=1 Time to live exceeded

- frontalcv7.cv.una.es ping statistics ---
packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

bik@terminator ~/l/n/lab2 (master) (1)> ping informatica.cv.una.es -c 1 -t 2
WG frontalcv7.cv.una.es (150.214.54.249) 56(84) bytes of data.
on 10.195.135.1 (10.195.135.1) icmp_seq=1 Time to live exceeded

- frontalcv7.cv.una.es ping statistics ---
packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

bik@terminator ~/l/n/lab2 (master) (1)> ping informatica.cv.una.es -c 1 -t 3
WG frontalcv7.cv.una.es (150.214.54.249) 56(84) bytes of data.
on 10.254.34.13 (10.254.34.13) icmp_seq=1 Time to live exceeded

- frontalcv7.cv.una.es ping statistics ---
packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

bik@terminator ~/l/n/lab2 (master) (1)> ping informatica.cv.una.es -c 1 -t 4
WG frontalcv7.cv.una.es (150.214.54.249) 56(84) bytes of data.
on 10.179.37.129 (10.179.37.129) icmp_seq=1 Time to live exceeded

- frontalcv7.cv.una.es ping statistics ---
packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

bik@terminator ~/l/n/lab2 (master) (1)> ping informatica.cv.una.es -c 1 -t 5
WG frontalcv7.cv.una.es (150.214.54.249) 56(84) bytes of data.
on rediris.alta.espanix.net (185.79.175.154) icmp_seq=1 Time to live exceeded

- frontalcv7.cv.una.es ping statistics ---
packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

bik@terminator ~/l/n/lab2 (master) (1)> ping informatica.cv.una.es -c 1 -t 6
WG frontalcv7.cv.una.es (150.214.54.249) 56(84) bytes of data.
on 130.206.245.122 (130.206.245.122) icmp_seq=1 Time to live exceeded

- frontalcv7.cv.una.es ping statistics ---
packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

bik@terminator ~/l/n/lab2 (master) (1)> ping informatica.cv.una.es -c 1 -t 7
WG frontalcv7.cv.una.es (150.214.54.249) 56(84) bytes of data.
on cica-router-backup.red.rediris.es (130.206.211.42) icmp_seq=1 Time to live exceeded

- frontalcv7.cv.una.es ping statistics ---
packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

bik@terminator ~/l/n/lab2 (master) (1)> ping informatica.cv.una.es -c 1 -t 8
WG frontalcv7.cv.una.es (150.214.54.249) 56(84) bytes of data.
on una-router.red.cica.es (150.214.231.179) icmp_seq=1 Time to live exceeded

- frontalcv7.cv.una.es ping statistics ---
packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

bik@terminator ~/l/n/lab2 (master) (1)> ping informatica.cv.una.es -c 1 -t 9
WG frontalcv7.cv.una.es (150.214.54.249) 56(84) bytes of data.
on tuneles.una.es (150.214.47.249) icmp_seq=1 Time to live exceeded

- frontalcv7.cv.una.es ping statistics ---
packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

bik@terminator ~/l/n/lab2 (master) (1)> ping informatica.cv.una.es -c 1 -t 10
WG frontalcv7.cv.una.es (150.214.54.249) 56(84) bytes of data.
on te609dixie.una.una.es (150.214.41.238) icmp_seq=1 Time to live exceeded

- frontalcv7.cv.una.es ping statistics ---
packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

[Frame is ignored: False]
bik@terminator ~/l/n/lab2 (master) (1)> ping informatica.cv.una.es -c 1 -t 11
WG frontalcv7.cv.una.es (150.214.54.249) 56(84) bytes of data.

- frontalcv7.cv.una.es ping statistics --- icmp 11 icmpv6]
packets transmitted, 0 received, 100% packet loss, time 0ms

bik@terminator ~/l/n/lab2 (master) (1)> ping informatica.cv.una.es -c 1 -t 11:2a:a0
WG frontalcv7.cv.una.es (150.214.54.249) 56(84) bytes of data.

- frontalcv7.cv.una.es ping statistics ---
packets transmitted, 0 received, 100% packet loss, time 0ms

bik@terminator ~/l/n/lab2 (master) (1)> ping informatica.cv.una.es -c 1 -t 12
WG frontalcv7.cv.una.es (150.214.54.249) 56(84) bytes of data.
bytes from frontalcv7.cv.una.es (150.214.54.249): icmp_seq=1 ttl=52 time=45.5 ms

- frontalcv7.cv.una.es ping statistics ---
packets transmitted, 1 received, 0% packet loss, time 0ms
t min/avg/max/ndev = 45.488/45.488/45.488/0.000 ms

bik@terminator ~/l/n/lab2 (master) (1)>

```

**Exercise 6.** What protocol does the **tracert** use? List the protocols of each network layer. What are the protocols involved in the reply? What is the strategy used by **tracert** to find out the IP address of each of the hops?

What protocol does the **tracert** use?

Tricky question. Linux **tracert** implements various methods. Linux manpage says:

### List Of Available Methods

In general, a particular **tracert** method may have to be chosen by **-M name**, but most of the methods have their simple cmdline switches (you can see them after the method name, if present).

#### default

The traditional, ancient method of tracerouting. Used by default.

Probe packets are udp datagrams with so-called "unlikely" destination ports. The "unlikely" port of the first probe is 33434, then for each next probe it is incremented by one. Since the ports are expected to be unused, the destination host normally returns "icmp unreachable" as a final response. (Nobody knows what happens when some application listens for such ports, though).

This method is allowed for unprivileged users.

#### icmp -I

Most usual method for now, which uses icmp echo packets for probes.

If you can [ping](#)(8) the destination host, icmp tracerouting is applicable as well.

#### tcp -T

Well-known modern method, intended to bypass firewalls.

Uses the constant destination port (default is 80, http).

If some filters are present in the network path, then most probably any "unlikely" udp ports (as for *default* method) or even icmp echoes (as for *icmp*) are filtered, and whole tracerouting will just stop at such a firewall. To bypass a network filter, we have to use only allowed protocol/port combinations. If we trace for some, say, mailserver, then more likely **-T -p 25** can reach it, even when **-I** can not.

This method uses well-known "half-open technique", which prevents applications on the destination host from seeing our probes at all. Normally, a tcp syn is sent. For non-listened ports we receive tcp reset, and all is done. For active listening ports we receive tcp syn+ack, but answer by tcp reset (instead of expected tcp ack), this way the remote tcp session is dropped even without the application ever taking notice.

There is a couple of options for *tcp* method:

**syn,ack,fin,rst,psh,urg,ece,cwr**

Sets specified tcp flags for probe packet, in any combination.

**flags=num**

Sets the flags field in the tcp header exactly to *num*.

**ecn**

Send syn packet with tcp flags ECE and CWR (for Explicit Congestion Notification, rfc3168)

**sack,timestamps>window\_scaling**

Use the corresponding tcp header option in the outgoing probe packet.

**sysctl**

Use current sysctl (*/proc/sys/net/\**) setting for the tcp header options above and **ecn**. Always set by default, if nothing else specified.

**mss=num**

Use value of *num* for maxseg tcp header option (when **syn**).

Default options is **syn,sysctl**.

**tcpconn**

An initial implementation of tcp method, simple using [connect\(2\)](#) call, which does full tcp session opening. Not recommended for normal use, because a destination application is always affected (and can be confused).

**udp -U**

Use udp datagram with constant destination port (default 53, dns). Intended to bypass firewall as well.

Note, that unlike in *tcp* method, the correspond application on the destination host **always** receive our probes (with random data), and most can easily be confused by them. Most cases it will not respond to our packets though, so we will never see the final hop in the trace. (Fortunately, it seems that at least dns servers replies with something angry).

This method is allowed for unprivileged users.

**udplite -UL**

Use udplite datagram for probes (with constant destination port, default 53).

This method is allowed for unprivileged users.

Options:

**coverage=num**

Set udplite send coverage to *num*.

**raw -P proto**



Send raw packet of protocol *proto*.

No protocol-specific headers are used, just IP header only.

Implies **-N 1**.

Options:

**protocol=***proto*

Use IP protocol *proto* (default 253).

So the answer is: it really depends on which tracing method we choose. But by default it uses UDP to some random ports (33000-ish in my case) with increasing TTL, and checks, whether it receives ICMP TTL exceeded, so something similar to what we did in the previous exercise manually. So in my case the protocol stack is:

Query: WiFi Radio → Ethernet II → IPv4 → UDP

Response: WiFi Radio → Ethernet II → IPv4 → ICMP

The IPv4 header in the response contains the IP of the hop on the route.