Challenge 1

Group:

Reggiani Riccardo, 10670577

Tucci Francesco, 10607818

Question 1

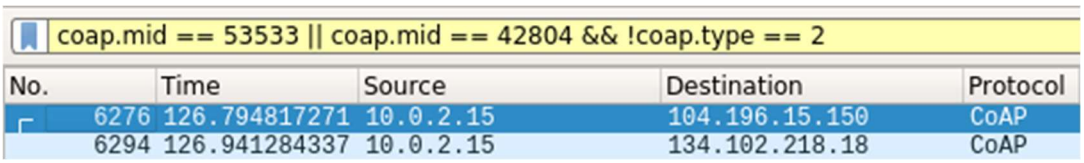
The main difference between the two requests with MID 53533 and MID 42804 is that the first one is a confirmable message, so it will need to be acknowledged and eventually retransmitted to guarantee a reliable exchange. While the second is not, consequently it will be unreliable and won't receive an ack. The ACK for the request with MID 53533 is sent right after (frame 6295), so in addition to the same token they will also have the same MID.

The filter used to obtain the requested frames is the following:

```
coap.mid == 53533 || coap.mid == 42804 && !coap.type == 2
```

Through it we obtained the frames below.

Frame	MID	Type	Code	Token
6276	42804	NON confirmable	DELETE	6dbdd020
6294	53533	confirmable	GET	242f92f0



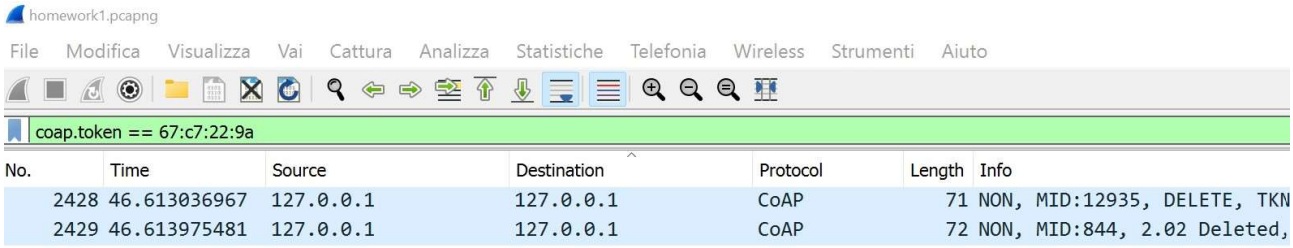
Question 2

Frame 2428 is a coap message with code 4 (delete) whose intent is to delete the resource /living_room/door.

In this case the filter we used was:

```
frame.number == 2428
```

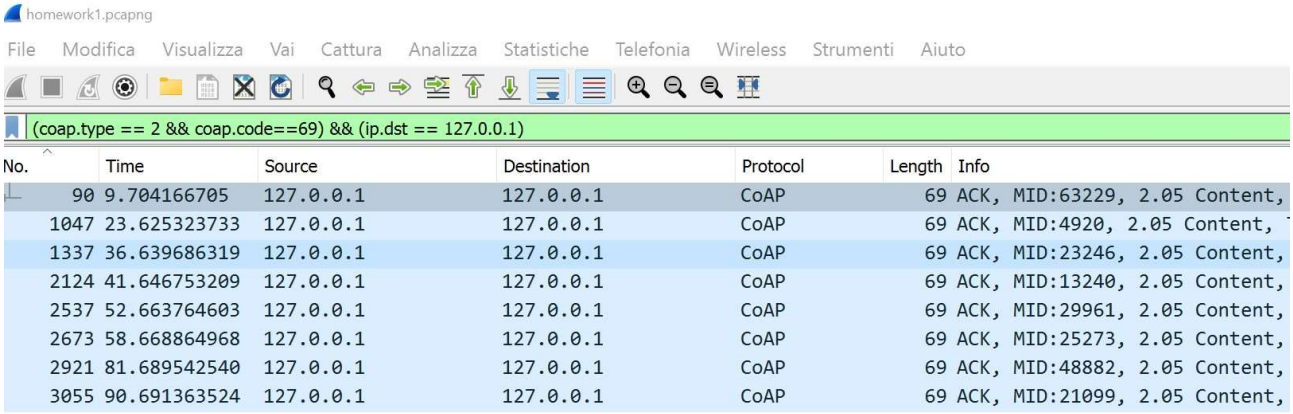
The DELETE request is fulfilled even though it was sent in an unreliable way. To find the response, we have looked for a message with the same token of the request and in the image below we can see the results.



The linked response is contained in frame 2429, a NON confirmable coap message with code 2.02 (deleted). This means that resource indicated in frame 2428 was effectively deleted.

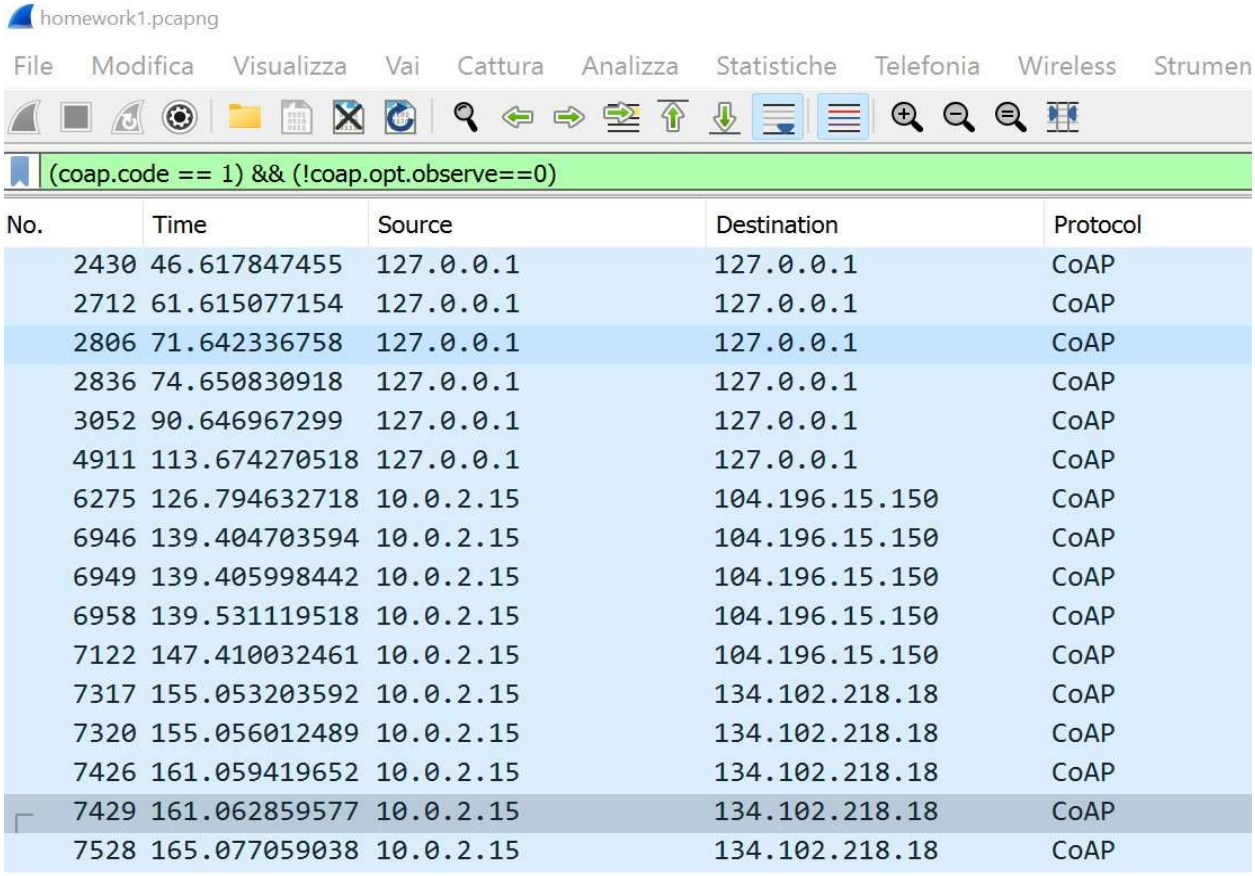
Question 3

After applying the filter shown in the picture below, we obtained 8 frames.



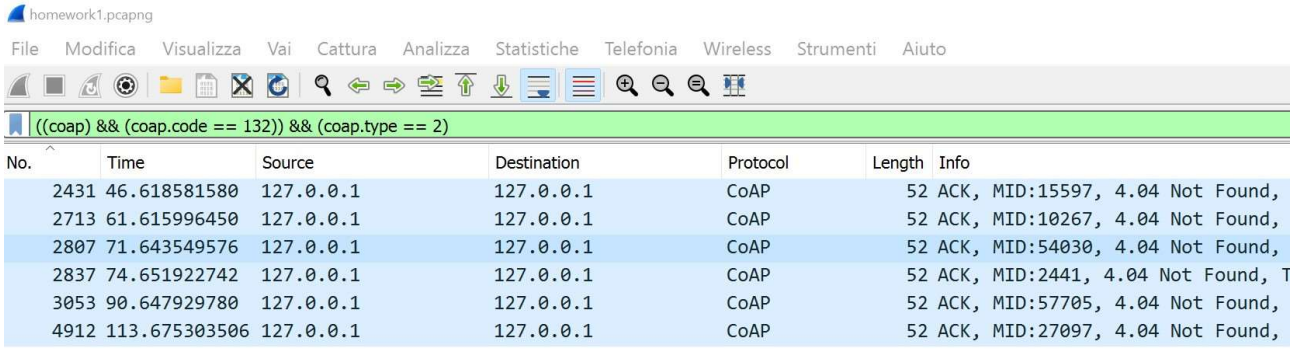
Question 4

First, we have filtered the messages to find the GET requests excluding the OBSERVE requests.



No.	Time	Source	Destination	Protocol
2430	46.617847455	127.0.0.1	127.0.0.1	CoAP
2712	61.615077154	127.0.0.1	127.0.0.1	CoAP
2806	71.642336758	127.0.0.1	127.0.0.1	CoAP
2836	74.650830918	127.0.0.1	127.0.0.1	CoAP
3052	90.646967299	127.0.0.1	127.0.0.1	CoAP
4911	113.674270518	127.0.0.1	127.0.0.1	CoAP
6275	126.794632718	10.0.2.15	104.196.15.150	CoAP
6946	139.404703594	10.0.2.15	104.196.15.150	CoAP
6949	139.405998442	10.0.2.15	104.196.15.150	CoAP
6958	139.531119518	10.0.2.15	104.196.15.150	CoAP
7122	147.410032461	10.0.2.15	104.196.15.150	CoAP
7317	155.053203592	10.0.2.15	134.102.218.18	CoAP
7320	155.056012489	10.0.2.15	134.102.218.18	CoAP
7426	161.059419652	10.0.2.15	134.102.218.18	CoAP
7429	161.062859577	10.0.2.15	134.102.218.18	CoAP
7528	165.077059038	10.0.2.15	134.102.218.18	CoAP

Requests that have been directed to non-existing resources must have been replied with an acknowledgement, and that acknowledgement must contain the error '404 not found', so we filtered with the below criteria.



No.	Time	Source	Destination	Protocol	Length	Info
2431	46.618581580	127.0.0.1	127.0.0.1	CoAP	52	ACK, MID:15597, 4.04 Not Found,
2713	61.615996450	127.0.0.1	127.0.0.1	CoAP	52	ACK, MID:10267, 4.04 Not Found,
2807	71.643549576	127.0.0.1	127.0.0.1	CoAP	52	ACK, MID:54030, 4.04 Not Found,
2837	74.651922742	127.0.0.1	127.0.0.1	CoAP	52	ACK, MID:2441, 4.04 Not Found, T
3053	90.647929780	127.0.0.1	127.0.0.1	CoAP	52	ACK, MID:57705, 4.04 Not Found,
4912	113.675303506	127.0.0.1	127.0.0.1	CoAP	52	ACK, MID:27097, 4.04 Not Found,

Then we have looked for the same tokens of the acknowledgements, in the GET requests excluding the observe requests and six of them matched.

Frame	2430	2712	2806	2836	3052	4911

Question 5

In this case it was first necessary to obtain the IP address and port of all the clients that transmitted a connect message with password admin. To do this it was used the following filter:

```
mqtt.msgtype == 1 && mqtt.passwd == "admin"
```

The IP address linked to all the resulting messages was 127.0.0.1 while the ports were different, and they are listed below:

Ports	51565	41869	60395	40989	47315	44429	60419	55953

At this point we had to apply the following filter by changing each time the source port:

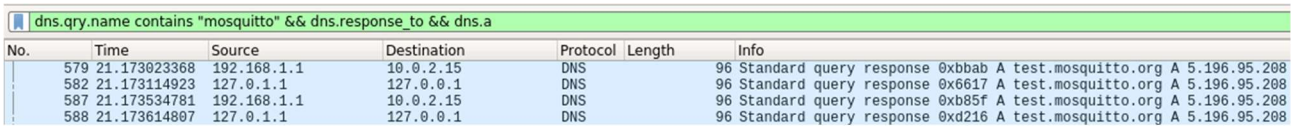
```
mqtt.topic contains "factory/department" && ip.src == 127.0.0.1 && mqtt.msgtype == 3 && tcp.srcport == 51565
```

Ports from which clients with psw admin connects:	Number of messages
51565	2
41869	1
60395	2
40989	3
47135	1
44429	1
60419	2
55953	1

In total the number of publishing messages sent by clients with password “admin” are 13.

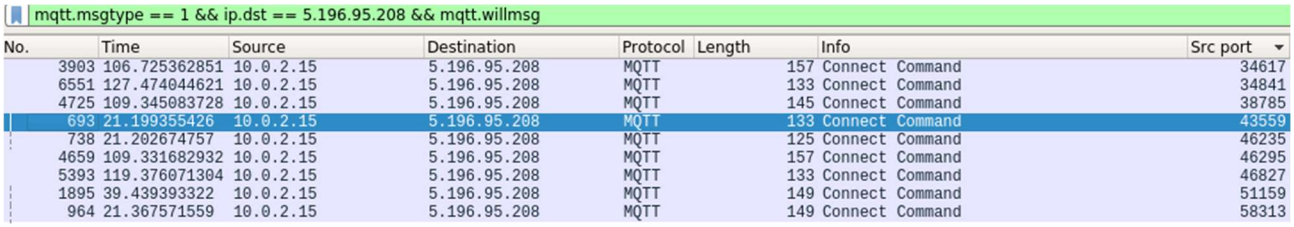
Question 6

The first step in answering this question was to find the address of the broker with name “mosquitto”. To do this we filtered the DNS response and got its IP address: 5.196.95.208.



After that we obtained the answer by specifying the following filters:

- mqtt.msgtype == 1: this filter let us obtain all the frames corresponding to connect messages
- ip.dest == 5.196.95.208: with this we obtained all the messages with destination the address of the desired broker
- mqtt.willmsg: this filter gives us all the frames containing a will message

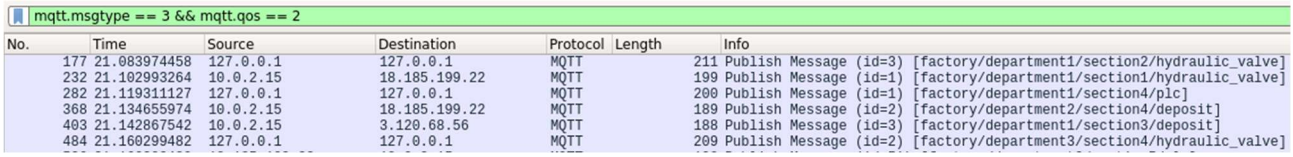


All the source ports are different so we can assume that the answer to the original question is 9 clients.

Question 7

To obtain this answer we used the following filters:

- mqtt.msgtype == 3: with this we obtained all the publish messages
- mqtt.qos == 2: through this we obtained all the mqtt messages with quality of service equal to 2.



Through the filters we obtained 94 packets that satisfied the request. After this we filtered all the packets that corresponded to a PUBREL message through the filter: mqtt.msgtype == 6. Given that no frame turned out we assumed that none (94 packets) of the original messages got a PUBREL.

Question 8

In this case the filter that must be used is the following:

```
mqtt.msgtype == 1 && mqtt.clientid_len == 0 && !_ws.malformed && mqtt.willtopic_len >= 0
```

The first condition allows us to obtain all the packages corresponding to connect requests whilst the second one filters all the packets with a client id different from the empty string. The third filter removes all the malformed packets and the fourth condition filters all the packages with a will topic length greater or equal to 0.

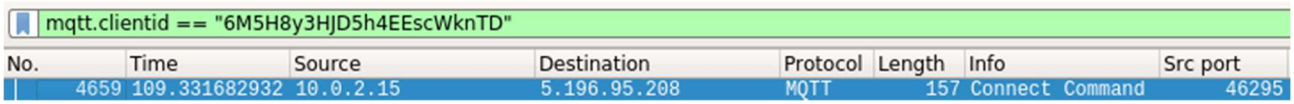
After the application of the filters above we obtained 22 frames. By collecting all the will topic lengths and averaging them we obtain an average will topic length of 37.77273.

Question 9

To answer this question, it is necessary to first discover the ip and port used by the client whose id is "6M5H8y3HJD5h4EEscWknTD". To do that we used the following filter:

```
mqtt.clientid == "6M5H8y3HJD5h4EEscWknTD"
```

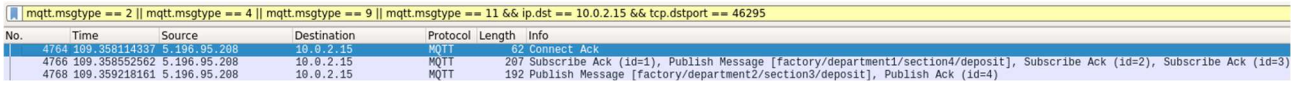
The filtering got us a single packet from which it is possible to obtain IP and port used by the client.



The next step was to filter the entire set of frames based on the following filter:

```
mqtt.msgtype == 2 || mqtt.msgtype == 4 || mqtt.msgtype == 9 || mqtt.msgtype == 11 && ip.dst == 10.0.2.15 && tcp.dstport == 46295
```

Through this passage we obtained 3 frames corresponding to all the filters, which contain a total of 5 ACKs.



Question 10

The filter to use in this case is the following:

```
mqtt.msgtype == 1 && mqtt.ver == 3
```

The first condition limits the set of packets to all those corresponding to connect requests while the second filters all the packets produced with a version of MQTT different from the 3.1. By collecting the data of all the messages, it is possible to obtain an average length of 63.59574.

The packets vary in size because of the optionality of some fields in MQTT connection requests.