IoT 2023 CHALLENGE 1

(1) Name: Alessandrini Luca Person Code: 10569363

(2) Name: Renzi Flavio Person Code: 10659808

For more detailed answers and for complete code snippets, refer to the notebook (we didn't report the code here, since some long lines were not well readable within the pdf page):

1. ***How many CoAP GET requests are directed to non-existing resources in the <u>local</u> CoAP server? How many of these are of type Non confirmable? (0.2 pts)***

   Answer: 11 requests, 6 of which are non confirmable.
   We started finding the CoAP `GET` requests. Then, comparing the tokens or message ids, we found the responses of the server. By filtering the ip of the server (which needs to run on `localhost`), we found that the `GET` requests which returned a not found were 11. Adding the filter on the type of the message, we found that 6 were non confirmable.
   In order to see the actual code snippets, refer to the section of the notebook about this question.

2. ***How many CoAP DELETE requests directed to the "coap.me" server did not produce a successful result? How many of these are directed to the "/hello" resource? (0.2 pts)***

   Answer: 105 were unsuccessful, 5 of which were directed to the `hello` resource.
   As first thing, we need to find the ip of the `coap.me` server. To do this, it is enough to filter the dns on Wireshark. The response is:
   `coap.me: type A, class IN, addr 134.102.218.18`
   Then we need to find how many CoAP delete requests were directed to that server. In order to find the bad ones, it is enough to see which of them did not have a response with a `deleted` result (number of requests, minus the ones that answered `deleted`: in this way we are counting also the ones never received). They are 105.
   Then, we can save every bad response and check whether they were directed to the `hello` resource. The ones directed to that resource are 5.
   In order to see the actual code snippets, refer to the section of the notebook about this question.

3. ***How many different MQTT clients subscribe to the <u>public</u> broker mosquitto using single-level wildcards? How many of these clients WOULD receive a publish message issued to the topic "hospital/room2/area0" (0.2 pts)***

   Answer: Single clients subscribed with single layer wildcard: 3, 2 of which would receive messages issued to the topic `hospital/room2/area0`.
   We need to find the dns resolution on wireshark for the domain `test.mosquitto.org`. The resolution is:
   `test.mosquitto.org: type A, class IN, addr 91.121.93.94`

Then, we need to check the subscribe messages to that ip, whose topic contain a `+`.
At this point, we have the subscriptions to that server with a single level wildcard. We need to check the source port of the connection, to detect the single client. The single clients detected were 3.
Then, we need to check among those clients who made a subscription to that server which matches `hospital/room2/area0` (written as it is, or with single or multiple level wildcards). We implemented the check through a regex, and we obtained that 2 clients matched that topic. This is all contained in the `Interpretation 2` section of the notebook.
For completeness, we also implemented another version of the question (in the notebook the section called `Interpretation 1`), checking which of the subscription containing a `+` matched the topic (only 1).
In order to see the actual code snippets, refer to the section of the notebook about this question.

4. ***How many MQTT clients specify a last Will Message directed to a topic having as first level "university"? How many of these Will Messages are sent from the broker to the subscribers? (0.2 pts)***

Answer: 3 clients set it with as first topic level `university`, and 0 last will messages have been sent.
The last will is set on connection, hence we need to check among the mqtt connection who set the last will message. To see which have as first topic university, we need to check whether splitting the string on "`/`" and taking the first argument, we have `university`. They are 3. Since they are not a lot, we decided to inspect the messages. In python, we decoded the message, and every message starts with the word `error`. In this case, if we check quickly how many messages have been sent which starts with this word, the code returned 0, and it is enough to conclude that no last will message of this type have been sent.
In order to see the actual code snippets, refer to the section of the notebook about this question.

5. ***How many Publish messages with QoS = 1 are received by the MQTT clients connected to the HiveMQ broker with MQTT version 5? (0.1 pts)***

Answer: 60.
As before, we need to find the address of the broker. In this case, the dns resolve it with 2 ip addresses:
```
52.29.173.150
3.65.137.17
```
we need to find the list of single clients subscribed to the server with mqtt version 5. In order to check the actual number of messages received by those subscribers with QoS = 1 we checked the ack messages: we obtained a result of 60.
We did further investigations written on the notebook about the messages sent from the broker to those clients, and we noticed that the number of messages was smaller than the number of acks. This is because we have multiple mqtt packets in a single message.
In order to see the actual code snippets, refer to the section of the notebook about this question.

6. ***How many MQTT-SN (on port 1885) publish messages sent after the hour 3.16PM (Milan Time) are directed to topic 9? Are these messages handled by the server? (0.1 pts)***

Answer: 15, none of which is handled by the server.
We can start finding the publish messages directed to the topic id `9`, and then we can check the time constraint. The number of messages is 15.
In order to check whether these messages has been handled by the server, we tried two options (both lead to the same result).
We can either move to Wireshark, since it is really simple, or we can proceed with python.

Wireshark:
It is enough to filter the messages with:
```
mqttsn && mqttsn.topic.id == 9 && mqttsn.msg.type == 0xc
```
We can see that for each message sent, there it is an ICMP response saying `Destination Unreachable (Port Unreachable)`, which indicates that no one is handled by the server.
Python:
We can create a list of bad ICMP responses and check if among the messages handled within the constraints are among one of the bad responses. The result is still 0.

It may be due to the fact that the default port is `1884`, whereas the port used is `1885` and it may not have been changed on the server.
In order to see the actual code snippets, refer to the section of the notebook about this question.