Internet of Things – A.Y. 2020/2021

Home Challenge #1 – Diego Savoia (Personal Code: 10535515)

Note: the only tool used for this challenge is Wireshark. In particular, the version included in the Virtual Machine provided by this course. The only setting that was changed is in "Capture -> Options… -> Options -> Name Resolution" in which all the three boxes have been enabled, especially to resolve network names so to simplify the lookup for the MQTT broker. Additionally, only the useful packets have been considered, hence discarding the ones highlighted as "Malformed Packet" by Wireshark.

Question 1 - What's the difference between the message with MID: 3978 and the one with MID: 22636?

Using the filter "coap.mid == 3978", we find a DELETE confirmable message (frame 6701) from 10.0.2.15 to californium.eclipse.org. The same filter also finds the corresponding ACK message (frame 6702 with the same token 8d:68:40:a0) from californium.eclipse.org to 10.0.2.15 (since the first was a confirmable message): this acknowledgement contains an error code "4.05 Method Not Allowed" signalling that the DELETE requested by 10.0.2.15 was not allowed. Instead, using the filter "coap.mid == 22636", we find a DELETE non-confirmable message (frame 6943 with token 55:20:eb:62) from 10.0.2.15 to californium.eclipse.org. The difference between the two messages is that the first one is a confirmable message, and therefore an acknowledgment in response is expected (and actually received); instead, the second is a non-confirmable message, and therefore no acknowledgement is expected (and indeed it is not present).

Question 2 - Does the client receive the response of message No. 6949?

Using the filter "frame.number == 6949", we find a GET confirmable message from 10.0.2.15 to californium.eclipse.org with ID=28357 and token=6f:b6:3c:18. To find the corresponding response, we need to apply the filter "coap.token == 6f:b6:3c:18", which returns both the above confirmable message and the acknowledgement received in response. Therefore, the ACK is correctly received by 10.0.2.15. However, we see that the code of this ACK is "4.05 Method Not Allowed", so we could say that the ACK is correctly received, but it does not include the actual resource requested by 10.0.2.15, since apparently a GET method was not allowed on that resource.

Question 3 - How many replies of type confirmable and result code "Content" are received by the server "localhost"?

The expression "replies of type confirmable" in the question obviously refers to ACKs related to confirmable messages. Therefore, we are looking for the ACKs that contain a result code "2.05 Content" and are received by the server "localhost". To do this, we apply the filter "coap.code == 69 and ip.dst_host == localhost": we find 8 packets (frames 90, 1047, 1337, 2124, 2537, 2673, 2921, 3055).

Question 4 - How many messages containing the topic "factory/department*/+" are published by a client with username: "jane"? Where * replaces the dep. number, e.g. factory/department1/+, factory/department2/+ and so on.

We can use the filter "mqtt.username == jane" to find all the connect messages containing that username: 4 packets are returned by the filter (frames 1554, 1623, 3708, 5816). Using the "Follow -> TCP Stream" tool in Wireshark, we can follow the stream for each of these four packets. Even if different publish messages by the client with username "jane" are found, none of them contains the topic specified in the question. This is because, according to the protocol, the "+" wildcard in the topic represents only one level. Instead, in the publish messages found, there is always more than one level after the department level (for example "factory/department2/section4/plc" which does not respect the pattern asked by this question). Therefore, there are 0 messages containing the topic "factory/department*/+" published by a client with username "jane".

Question 5 - How many clients connected to the broker "hivemq" have specified a will message?

We can use the filter "mqtt.willmsg and ip.dst_host == broker.hivemq.com" to find all the connect messages sent to the broker "hivemq" in which a will message has been specified. We find 10 messages (frames 544, 1540, 2304, 4681, 4774, 4803, 5373, 6042, 6485, 7408), and therefore 10 clients that connected to the broker "hivemq" and have specified a will message. We notice that some of them have the same value in the "Client ID" field, and therefore we can suppose that the same client connected to the broker, then disconnected, and then connected again. However, since the question does not specify "distinct" clients, we might count the actual connect messages, obtaining a total of 10 clients as said before. Alternatively, assuming that the empty "Client ID" field in the connect messages refers to different clients, we would have a total of 9 clients since the one with ID "6jhtWRl3sl3MTQ75FVhIVx" is present twice (frames 4803 and 7408).

Question 6 - How many publishes with QoS 1 don't receive the ACK?

Using the filter "mqtt.msgtype == 3 and mqtt.qos == 1 and !_ws.malformed" we can find all the 118 publish messages with QoS 1 (which are not malformed). Instead, using the filter "mqtt.msgtype == 4 and !_ws.malformed" we find all the 50 ACKs (which are not malformed). Therefore, there are 118 – 50 = 68 publishes messages with QoS 1 that don't receive the ACK. Notice that no filter on the source or destination host is applied, since the question does not specify any additional constraint.

Question 7 - How many last will messages with QoS set to 0 are actually delivered?

If the question is to be interpreted as "delivered last will messages whose QoS level was set to 0 in the initial connect message", then we can use the filter "mqtt.willmsg and mqtt.conflag.qos == 0" to find all the connect messages containing the last will with QoS set to 0. The filter returns 14 packets, and we can easily see that the last will message (in this pcap scenario) always contains the word "error". Therefore, we apply the filter "mqtt.msg contains "error"" to obtain the last will messages that are delivered: they are 5 frames (4164, 4178, 4195, 5504, 5557). We can check that none of these delivered last will messages correspond to the ones obtained using the initial filter: indeed, none of these delivered last will messages had a QoS set to 0 in the initial connect message. Therefore, we can

say that 0 last will messages with QoS set to 0 (in the initial connect message) are actually delivered. Instead, if the question is to be interpreted as “last will messages sent using QoS 0” we would use the filter “mqtt.msg contains “error” and mqtt.qos == 0”, finding that only one last will message is delivered using QoS 0. So, it could be 0 or 1 packet, depending on how we interpret the question.

Question 8 - Are all the messages with QoS > 0 published by the client “4m3DWYzWr40pce6OaBQAfk” correctly delivered to the subscribers?

We can use the filter “mqtt.clientid == 4m3DWYzWr40pce6OaBQAfk” to find the connect message sent from the client with that ID to the broker test.mosquitto.org (frame 964). Then, using the “Follow -> TCP Stream” tool in Wireshark, we can follow the stream for that packet. We discover that the client sends, apart from that connect message: a subscribe request (frame 968), a publish message with QoS 0 (frame 972), and a publish message with QoS 2 (frame 2423). According to the question, we are interested only in this last publish message, which has ID=3 and topic=factory/department1/section1/deposit. Analysing further the TCP stream above, we see that the broker correctly answers with a PUBREC (frame 2425 and of course same ID=3) but then the PUBREL message from the client and the consequent PUBCOMP message from the broker are missing. Additionally, we could double check this result using the filter “mqtt.msg == factory/department1/section1/deposit”. We discover that no message with that content is ever delivered to any subscriber, since this filter returns only the publish message found above (frame 2423).
In conclusion, we can say that no message with QoS > 0 published by the client “4m3DWYzWr40pce6OaBQAfk” is correctly delivered to the subscribers. Indeed, there is only one message that satisfies these conditions, and the exchange of messages between the client and the broker is not done correctly according to the protocol (with respect to the QoS 2 rules).

Question 9 - What is the average message length of a connect msg using mqttv5 protocol? Why messages have different size?

We can use the filter “mqtt.ver == 5 and !_w.malformed” to visualize only the MQTT connect messages which use the MQTTv5 protocol (and are not malformed): we get a total of 48 packets. The message length of each MQTT packet is reported in the “Msg len” field: by doing some simple computation, the average length is 813 / 48 = 16.9375 (instead, considering the length of the whole frame we would get an average of 3729 / 48 = 77.6875).
We can see that most of them have length 13: these contains just basic information, like the KeepAlive value, the QoS level and the CleanSession flag. Instead, longer packets also contain information about the Client ID, or even the username and password used by the client, in case these fields are necessary for that specific connection. This is the reason why the considered messages have different size.

Question 10 - Why there aren’t any REQ/RESP pings in the pcap?

In this pcap, clients remain active by sending other control packets. They disconnect by sending an explicit DISCONNECT request (found using the filter “mqtt.msgtype == 14”) or by hard disconnection (e.g. crashes). We know the existence of this latter case since last will messages are sent (as seen in Question 7) and, according to the protocol, they are sent in case of hard disconnections. Apparently, the clients in the pcap follow these three cases and therefore REQ/RESP pings are not present.