**USEVALAD MILASHEUSKI/10816982**

**1)** (coap.mid == 53533 || coap.mid == 42804) && not coap.type ==2

Using filter coap.mid == MID, we can see that MID 53533 corresponds to a confirmable REQ message + an acknowledgement which we filter out by using not coap.type ==2, while MID 42804 corresponds to a non-confirmable one.

*Main differences:*

* MID 53533 – a) Code GET; b) Token 242f92f0; c) Confirmable; d) Source port 50593
* MID 42804 – a) Code DELETE; b) Token 6dbdd020; c) Non-confirmable; d) Source port 5683



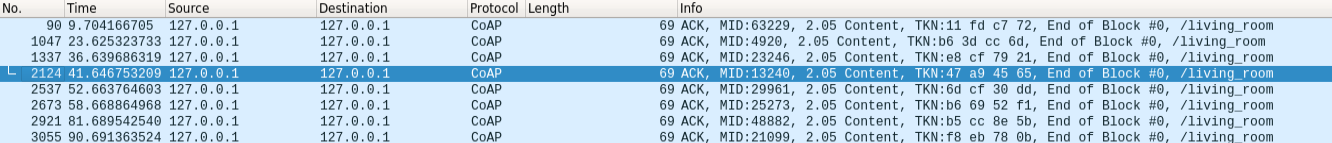
2) coap.response\_to == 2428

Firstly we apply frame.number == 2428 to check the request. It is a delete request, which is a non-confirmable, but since it is a DEL request, the user is supposed to receive a response. Indeed, if we apply ) coap.response\_to == 2428 we will see that it is true, the message contains code 2.02 (successfully deleted) and the tokens of the two messages coincide( coap.token == 67:c7:22:9a. will output two messages: from the user and from the server).



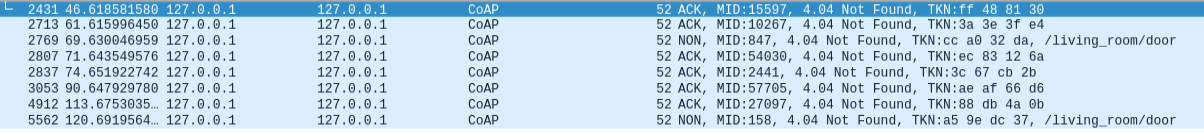
3) coap.code== 69 && ip.dst==127.0.0.1 -> 8 replies

Since the IP of the localhost is 127.0.0.1 and the code corresponding to content is 69, by combining the two filters we obtain eight received ACKs to confirmable messages.

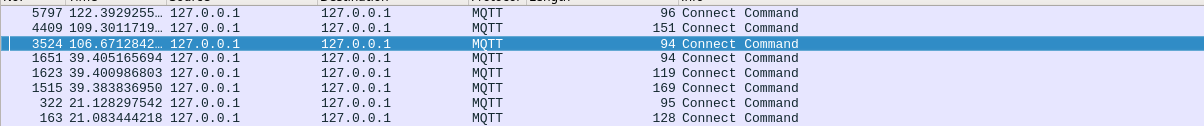


4) !coap.opt.observe && ( coap.code == 132) ->8 requests

Excluding the observable message at first, we apply the filter which corresponds to a 4.04 Not Found code, so we apply 132 code and observe 8 messages, 6 of which are ACKs and 2 of which are NON. We assume that number of requests equal to number of the replies (none was lost).



5) tcp.srcport in {51565 41869 60395 40989 47135 44429 60419 55953} && ip.src == 127.0.0.1 && mqtt.topic matches "factory/department./+" && mqtt.msgtype == 3 ->13 messages



Firstly we use mqtt.passwd == admin to check the users having the password “admin”.

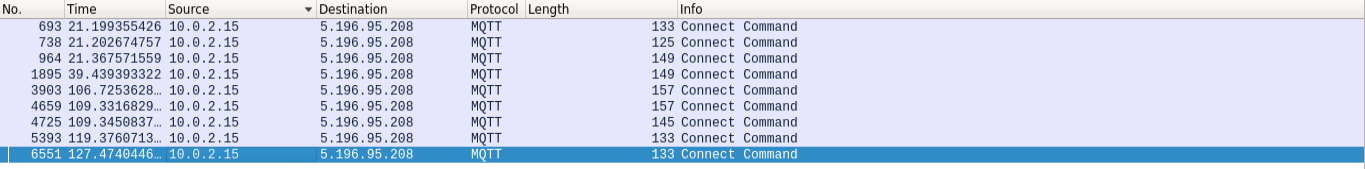


Than to each of the users we check the TCP source and IP in order to track only the needed users. Since we are interested in the publications, we specify it with mqtt.msgtype == 3. For this solution it is assumed that we are looking for any hierarchy lower than department\*/, where \* is a number. By using mqtt.topic matches "factory/department./+" we specify the lower hierarchy which is similar to the one of interest.

6) *in the terminal:* ping test.mosquitto.org

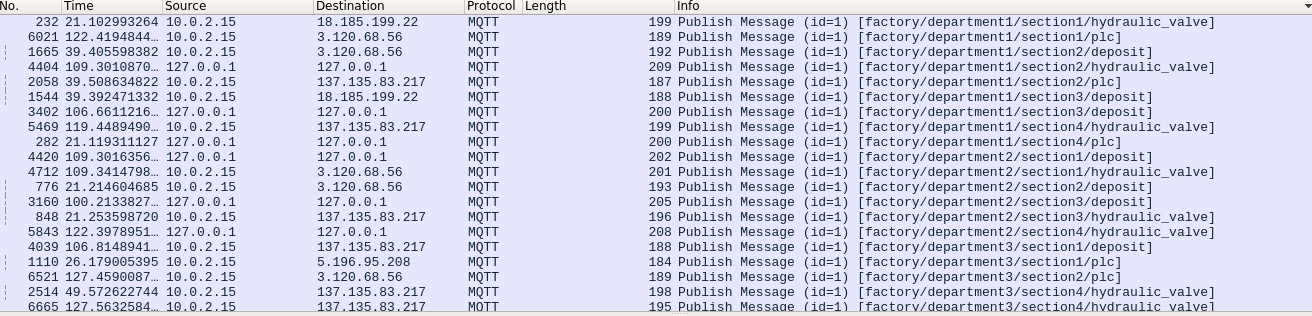
mqtt.conflag.willflag == 1 && ip.dst == 5.196.95.208 -> 9 clients

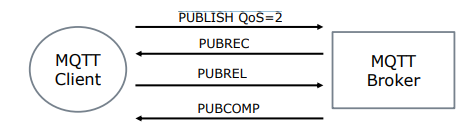




First in the comand window we *ping* mosquitto broker in order to obtain the IP. Then we use the adress as the destination and search for messages containing a last will message( checking via the flag) and observe that there are nine packages. Since the last will can be specified only in a connect message, specefication of the code is unnecessary.

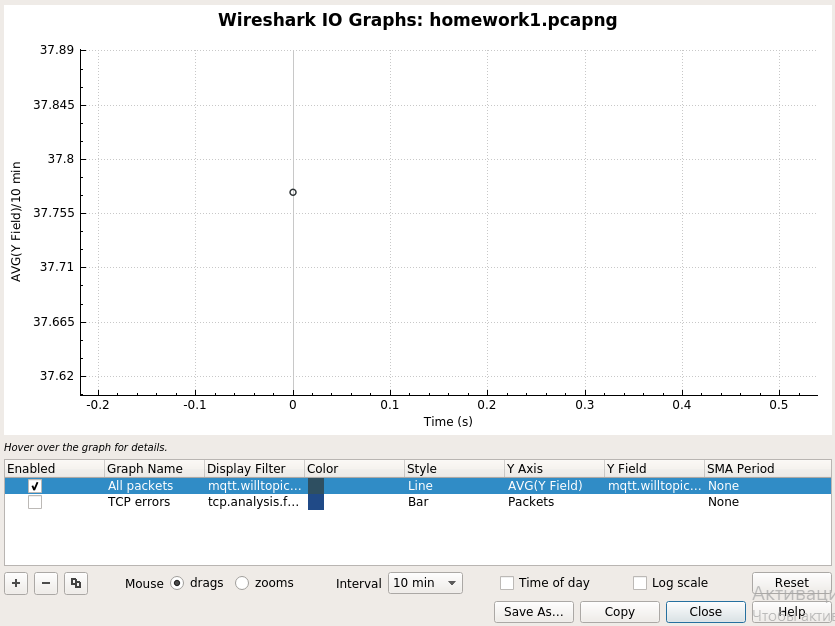
7) mqtt.msgtype == 6 || (mqtt.qos ==2 && mqtt.msgtype == 3) -> 95 publishes/94 frames





With the reference to the diagram above, we see that PUBREL is a message the client is supposed to sends to the broker after receiving PUBREC for every publish message of the QoS 2. However, if we check the number of PUBRELs using mqtt.msgtype == 6, we see none. So, for this solution it is assumed that thus none of the QoS 2 publishes has received it and, thus the number of messages which have not received PUBREL is equal to the number of QoS 2 PUBLISHes. To count the frames we can use Statistics->Capture Filter Properties and see the number of displayed packages 94, but it is important to notice, that frame 1152 has two PUBLISHes, leading to the final answer 95.

8) mqtt.willtopic\_len && !\_ws.malformed && mqtt.clientid=="" ->37.76

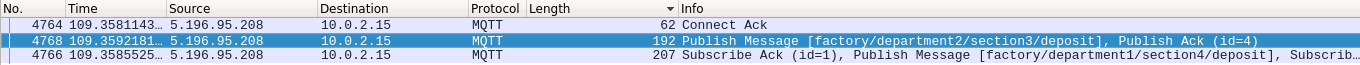


Firstly, if we apply the mqtt.willtopic\_len filter in the wireshark we can see that there are any malformed packages. In order to avoid them !\_ws.malformed filter was applied. Then since we care only about empty client IDs we use mqtt.clientid=="" filter. In order to find the average length, we went to Statistics->I/O Graph. There we inserted the final filter in the “Display Filter” field, changed the Y axis to AVG, inserted mqtt.willtopic\_len in the Y Field, increased the interval to capture everything and observed that the average is approximately equal to 37.76.

9) tcp.dstport == 46295 && ip.dst == 10.0.2.15 && (mqtt.msgtype == 2 || mqtt.msgtype == 4 || mqtt.msgtype == 9 || mqtt.msgtype == 11)->5 ACKs

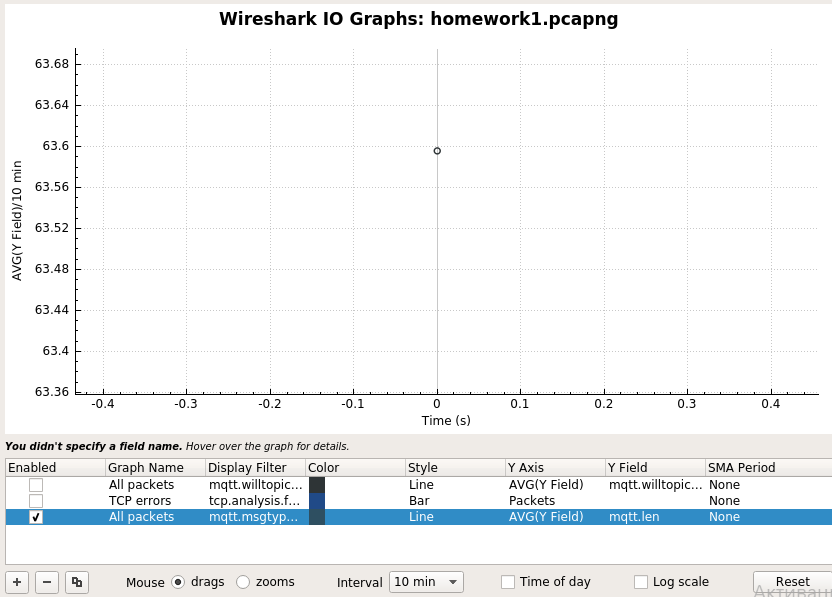


At the beginning we apply mqtt.clientid == 6M5H8y3HJD5h4EEscWknTD to discover the message with the given ID.



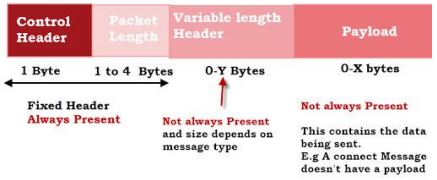
Than we apply filters specifying destination port and destination ip of the use client. Then we add only the acknowledgment codes (2, 4, 9 and 11). As a result, we can observe 3 frames. If we open the packets and count the ACKs, in total we have a Connection acknowledgment in frame 4764, Publish acknowledgment in frame 4768 and 3 Subscribe acknowledgments in frame 4766, which leads to 5 in total.

10) mqtt.ver == 3 && mqtt.msgtype == 1-> 63.59



Filter corresponding to CONNEC messages is mqtt.msgtype == 1. Since we are interested in the mqttv3.1 protocol, we apply mqtt.ver == 3, where 3 corresponds to the version of interest. Similar to Q8 we open Statistics->I/O Graph. There in the “Display Filter” field we insert our filter, change the Y axis to AVG, inserted mqtt.len in the Y Field, increase the interval and observe, that the average message length is approximately equal to 63.59.

In order to explain the variance in the msg length, we should take a look at the structure of the packet.



We can see that there are fixed parts as well as the variable length parts inside. The maximum length of a packet in this scenario is defined be the broker. If we take a look at the packet with the smallest(right) and the largest(left) length among the given ones, we can more information specified inside the larger one, indicating that there are optional fields in the mqttv3.1 messages (ex. user name, password, will message and etc. Their presence depends on the presence of the corresponding flags in the message). Below illustrated the package with the smallest and with the largest lengths.

