POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Course: Internet of Things - 5 CFU

Professor: Matteo Cesana
Students: Bruno Arena, Matteo Cavalleri
Personal IDs: 10806605, 10768765

Challenge 2 - Questions

Abstract

This section focuses on the empirical analysis of IoT network traffic captured in two PCAP files (A.pcapng and B.pcapng). The study is divided into three main areas of investigation:

- CoAP Protocol Evaluation: Identification of specific Non-Confirmable DELETE transactions and their success rates on the coap.me server. It further examines resource handling by quantifying Confirmable POST and PUT failures across local server resources.

- Observe Mechanism and Redundancy: A detailed audit of the CoAP Observe extension for the /dining_room/temperature resource, specifically filtering for unique notifications and identifying traffic inefficiencies or redundant data transmissions.

- MQTT and MQTT-SN Broker Interaction: An assessment of messaging patterns across local and public brokers. This includes measuring MQTT-SN traffic on port 1885, analysing subscriber behaviour regarding wildcards and Last Will messages, and identifying client attempts to clear retained values.

- Comparative Topic Depth Analysis: A comparative statistical study between the two captures, focusing on the structural complexity of MQTT topics (number of layers) and the overall volume of PUBLISH messages directed to the local broker.

POLITECNICO
MILANO 1863

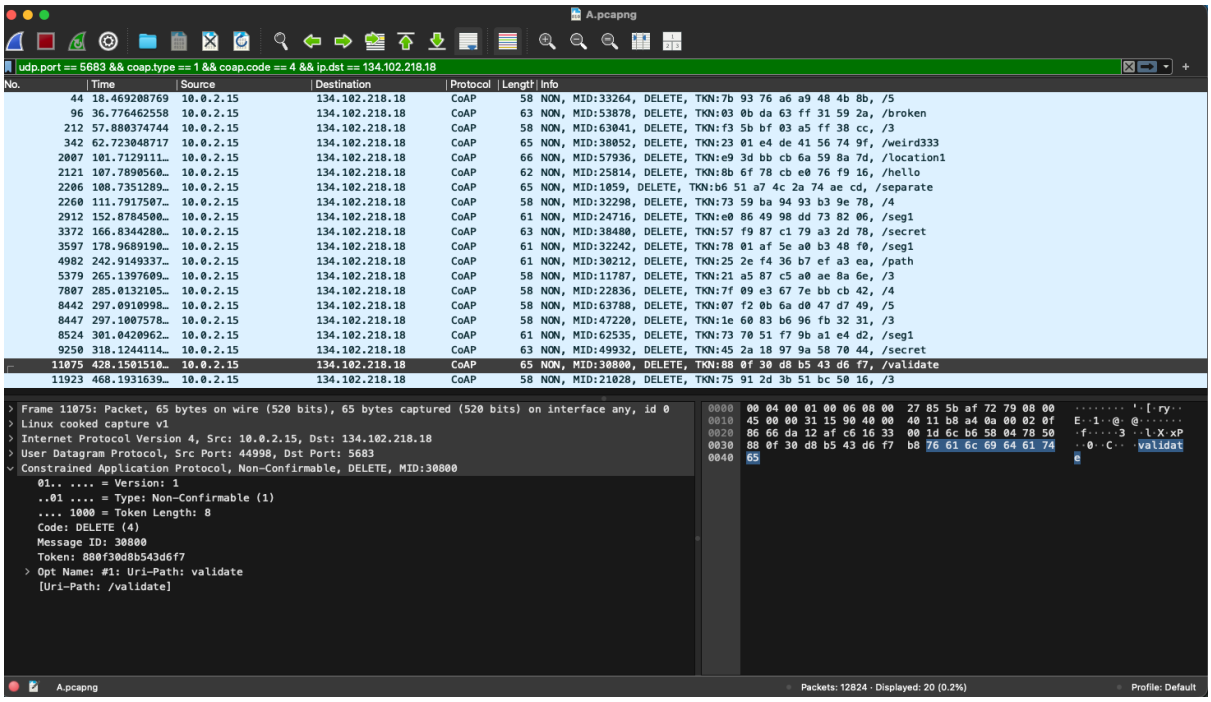
SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Question #1 – Analysis of Non-Confirmable DELETE Requests to coap.me

Answer CQ1a

CoAP messages are transported over UDP on the port 5683. A NON-Confirmable DELETE has Type 1 and code 4 (Delete). The CoAP.me server IP is 134.102.218.18.
The way used to identify a successful response is the code class. The code class for a successful response is 2.xx.
By using the following filter is possible to search every message:

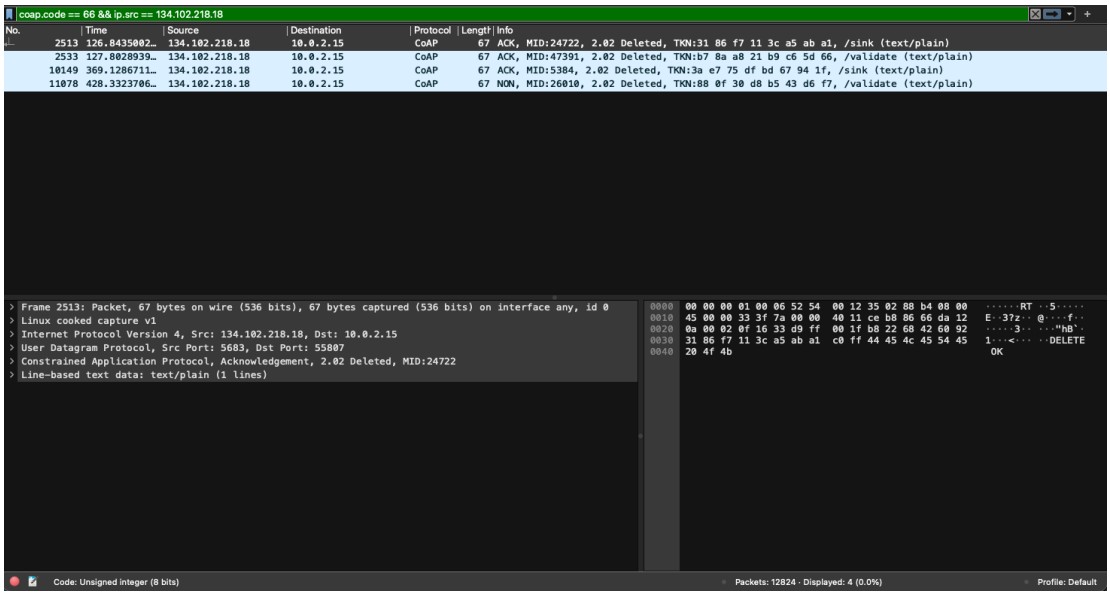
Filter: udp.port == 5683 && coap.type == 1 && coap.code == 4 && ip.dst == 134.102.218.18



The filter shows all the 20 NON DELETE requests.

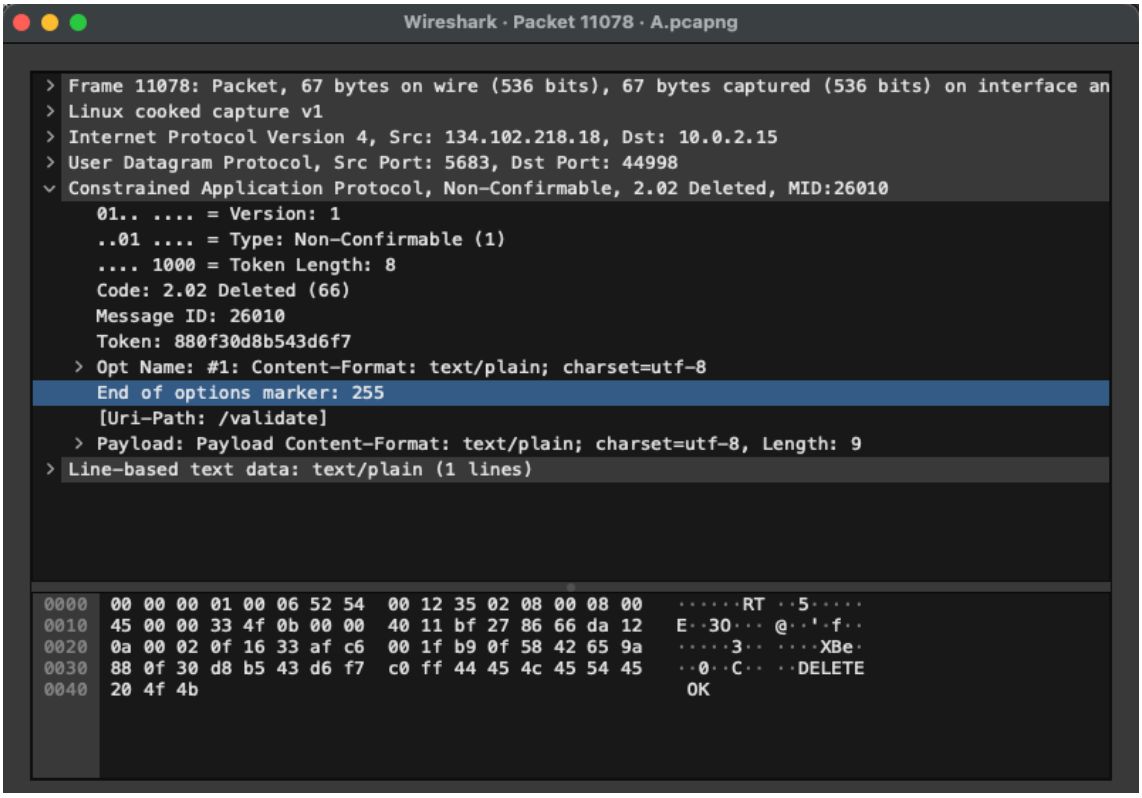
To identify the successful response of a delete requests is used this filter:

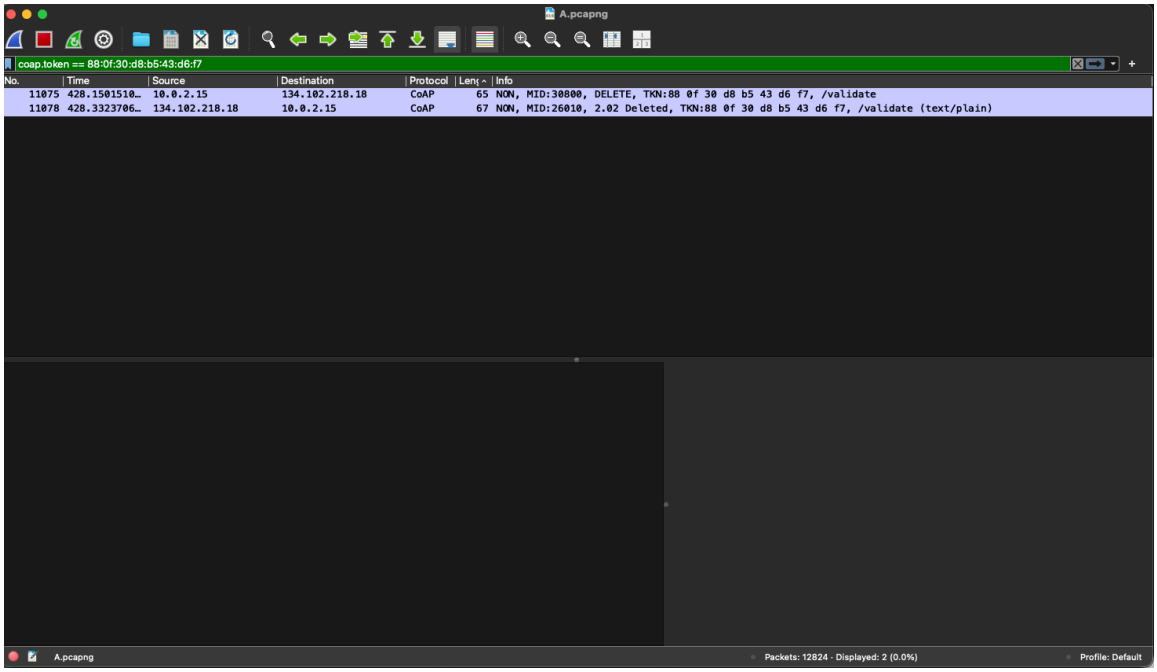
Filter: udp.port == 5683 && coap.code == 66 && ip.src == 134.102.218.18



By analysing these four requests from their token we have associated the request and the response, identifying the correct answer.

The check shows that the only request that received a successful response is the request with MID = 30800.





The request receives the 2.02 code which shows the success of the message.

Answer CQ1b

The desired outcome of a DELETE request is the resource deletion.
The confirm of the operation is verified from the request's response with code "2.02 Deleted".
All the other 20 request received a "4.05 Method Not Allowed" because the deletion is not permitted on those resources on coap.me.

Filter: udp.port == 5683 && coap.code == 66 && ip.src == 134.102.218.18

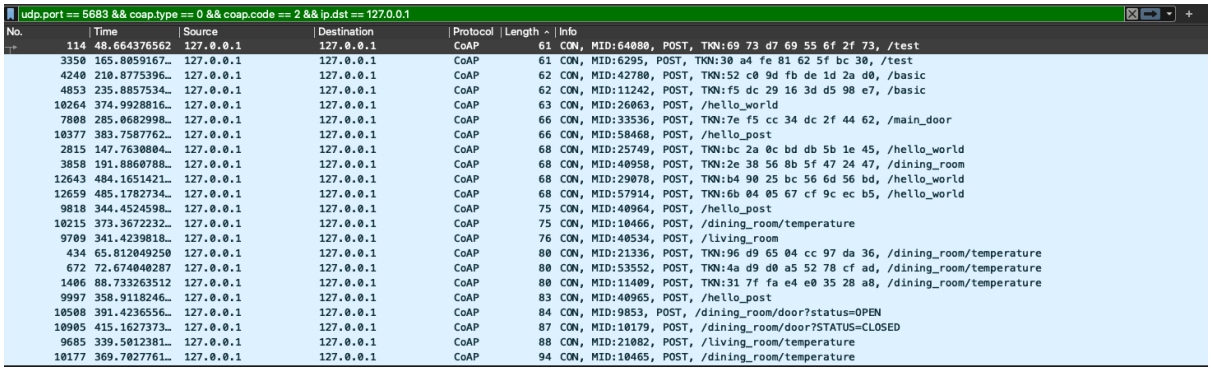
The correlation is by the Token: 880f30d8b543d6f7.

Answer CQ1
CQ1a: ***One request***, with MID = 30800 and Token: 880f30d8b543d6f7
CQ1b: ***One request*** obtained the desired outcome, the same as CQ1a with Token: 880f30d8b543d6f7

Question #2 - Analysis of Resources with Equal Counts of Unsuccessful Confirmable POST and PUT Requests

Retransmissions are detected by deduplicating on src_ip, dst_ip, MID.
An "Unsuccessful" request means the server sent a 4.xx/5.xx response.
Malformed packets are discard by hypothesis. Packets with token = 0 are not considered because, even if they are not malformed, is impossible to distinguish.
First we used a filter to select only the CON POST:

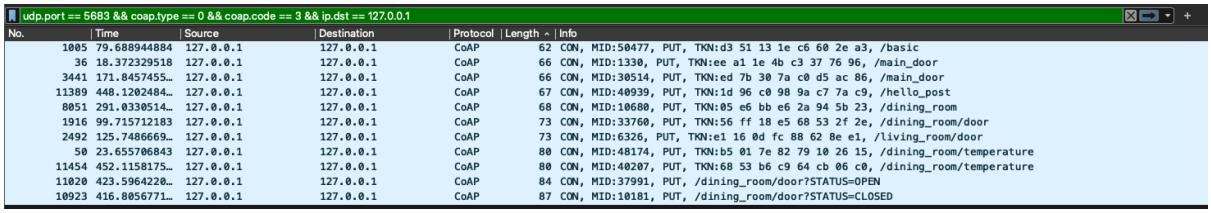
Filter: udp.port == 5683 && coap.type == 0 && coap.code == 2 && ip.dst == 127.0.0.1



udp.port == 5683 && coap.type == 0 && coap.code == 2 && ip.dst == 127.0.0.1

No.	Time	Source	Destination	Protocol	Length	Info
114	48.664376562	127.0.0.1	127.0.0.1	CoAP	61	CON, MID:64080, POST, TKN:69 73 d7 69 55 6f 2f 73, /test
3350	165.8059167…	127.0.0.1	127.0.0.1	CoAP	61	CON, MID:6295, POST, TKN:30 a4 fe 81 62 5f bc 30, /test
4240	210.8775396…	127.0.0.1	127.0.0.1	CoAP	62	CON, MID:42780, POST, TKN:52 c0 9d fb de 1d 2a d0, /basic
4853	235.8857534…	127.0.0.1	127.0.0.1	CoAP	62	CON, MID:11242, POST, TKN:f5 dc 29 16 3d d5 98 e7, /basic
10264	374.9928816…	127.0.0.1	127.0.0.1	CoAP	63	CON, MID:26063, POST, /hello_world
7808	285.0682998…	127.0.0.1	127.0.0.1	CoAP	66	CON, MID:33536, POST, TKN:7e f5 cc 34 dc 2f 44 62, /main_door
10377	383.7587762…	127.0.0.1	127.0.0.1	CoAP	66	CON, MID:58468, POST, /hello_post
2815	147.7630804…	127.0.0.1	127.0.0.1	CoAP	68	CON, MID:25749, POST, TKN:bc 2a 0c bd db 5b 1e 45, /hello_world
3858	191.8860788…	127.0.0.1	127.0.0.1	CoAP	68	CON, MID:40958, POST, TKN:2e 38 56 8b 5f 47 24 47, /dining_room
12643	484.1651421…	127.0.0.1	127.0.0.1	CoAP	68	CON, MID:29078, POST, TKN:b4 90 25 bc 56 6d 56 bd, /hello_world
12659	485.1782734…	127.0.0.1	127.0.0.1	CoAP	68	CON, MID:57914, POST, TKN:6b 04 05 67 cf 9c ec b5, /hello_world
9818	344.4524598…	127.0.0.1	127.0.0.1	CoAP	75	CON, MID:40964, POST, /hello_post
10215	373.3672232…	127.0.0.1	127.0.0.1	CoAP	75	CON, MID:10466, POST, /dining_room/temperature
9709	341.4239818…	127.0.0.1	127.0.0.1	CoAP	76	CON, MID:40534, POST, /living_room
434	65.812049250	127.0.0.1	127.0.0.1	CoAP	80	CON, MID:21336, POST, TKN:96 d9 65 04 cc 97 da 36, /dining_room/temperature
672	72.674040287	127.0.0.1	127.0.0.1	CoAP	80	CON, MID:53552, POST, TKN:4a d9 d0 a5 52 78 cf ad, /dining_room/temperature
1406	88.733263512	127.0.0.1	127.0.0.1	CoAP	80	CON, MID:11409, POST, TKN:31 7f fa e4 e0 35 28 a8, /dining_room/temperature
9997	358.9118246…	127.0.0.1	127.0.0.1	CoAP	83	CON, MID:40965, POST, /hello_post
10508	391.4236556…	127.0.0.1	127.0.0.1	CoAP	84	CON, MID:9853, POST, /dining_room/door?status=OPEN
10905	415.1627373…	127.0.0.1	127.0.0.1	CoAP	87	CON, MID:10179, POST, /dining_room/door?STATUS=CLOSED
9685	339.5012381…	127.0.0.1	127.0.0.1	CoAP	88	CON, MID:21082, POST, /living_room/temperature
10177	369.7027761…	127.0.0.1	127.0.0.1	CoAP	94	CON, MID:10465, POST, /dining_room/temperature

And the CON PUT:
Filter: udp.port == 5683 && coap.type == 0 && coap.code == 3 && ip.dst == 127.0.0.1



udp.port == 5683 && coap.type == 0 && coap.code == 3 && ip.dst == 127.0.0.1

No.	Time	Source	Destination	Protocol	Length	Info
1005	79.688944884	127.0.0.1	127.0.0.1	CoAP	62	CON, MID:50477, PUT, TKN:d3 51 13 1e c6 60 2e a3, /basic
36	18.372329518	127.0.0.1	127.0.0.1	CoAP	66	CON, MID:1330, PUT, TKN:ee a1 1e 4b c3 37 76 96, /main_door
3441	171.8457455…	127.0.0.1	127.0.0.1	CoAP	66	CON, MID:30514, PUT, TKN:ed 7b 30 7a c0 d5 ac 86, /main_door
11389	448.1202484…	127.0.0.1	127.0.0.1	CoAP	67	CON, MID:40939, PUT, TKN:1d 96 c0 98 9a c7 7a c9, /hello_post
8051	291.0330514…	127.0.0.1	127.0.0.1	CoAP	68	CON, MID:10680, PUT, TKN:05 e6 bb e6 2a 94 5b 23, /dining_room
1916	99.715712183	127.0.0.1	127.0.0.1	CoAP	73	CON, MID:33760, PUT, TKN:56 ff 18 e5 68 53 2f 2e, /dining_room/door
2492	125.7486669…	127.0.0.1	127.0.0.1	CoAP	73	CON, MID:6326, PUT, TKN:e1 16 0d fc 88 62 8e e1, /living_room/door
50	23.655706843	127.0.0.1	127.0.0.1	CoAP	80	CON, MID:48174, PUT, TKN:b5 01 7e 82 79 10 26 15, /dining_room/temperature
11454	452.1158175…	127.0.0.1	127.0.0.1	CoAP	80	CON, MID:40207, PUT, TKN:68 53 b6 c9 64 cb 06 c0, /dining_room/temperature
11020	423.5964220…	127.0.0.1	127.0.0.1	CoAP	84	CON, MID:37991, PUT, /dining_room/door?STATUS=OPEN
10923	416.8056771…	127.0.0.1	127.0.0.1	CoAP	87	CON, MID:10181, PUT, /dining_room/door?STATUS=CLOSED

Looking at the CON PUT is possible to identify the unsuccessful requests:

Resource	***CON POST***	***CON PUT***
/basic	2	1
/main_door	1	2
/hello_post	0	1
/dining_room	1	1
/dining_room/door	0	1
/living_room/door	0	1
/dining_room/temperature	3	2

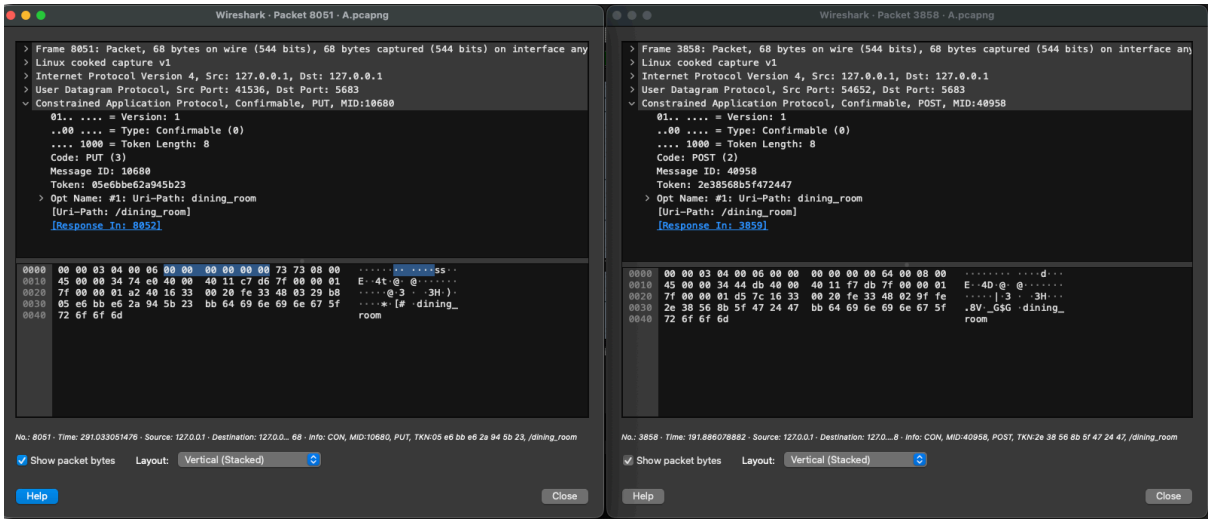


Figure 1 - Token match

Answer CQ2

The only resource with X (unique CON POST unsuccessful) = Y (unique CON PUT unsuccessful) > 0 is ***one*** and is the /dining_room resource.

POLITECNICO
MILANO 1863

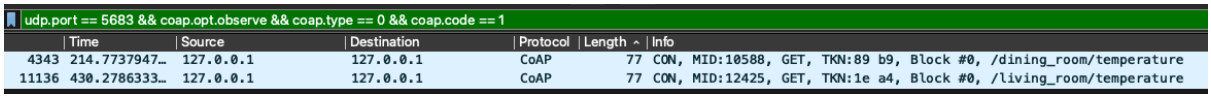
SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Question #3 - Analysis of Unique Observe Notifications and Redundancy for the /dining_room/temperature Resource

Answer CQ3a

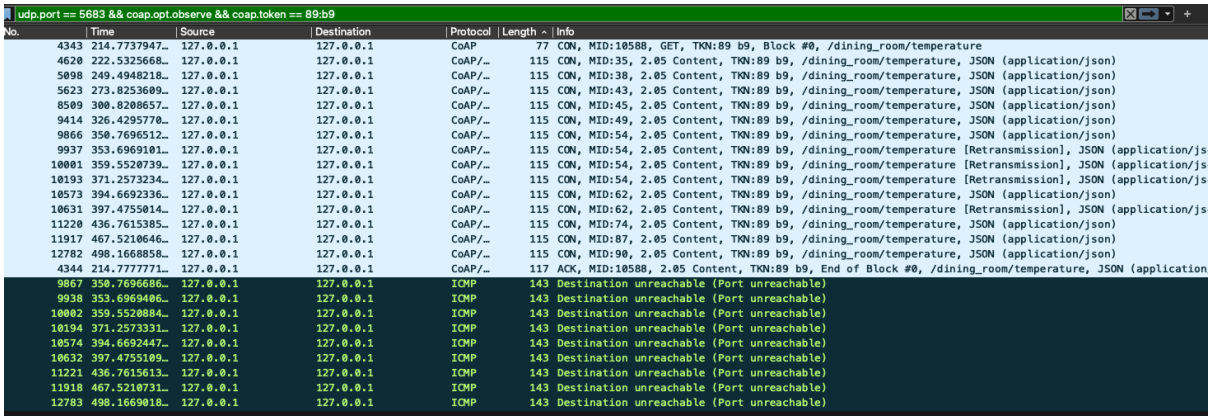
First we locate the Observe registration to the resource /dining_room/temperature using this filter:

Filter: udp.port == 5683 && coap.opt.observe && coap.type == 0 && coap.code == 1



The resource requested has Token = 89b9 so this token is used to find all the CON/NON responses with the observe option set.
Retransmissions aren't considered.

Filter: udp.port == 5683 && coap.opt.observe && coap.token == 89:b9



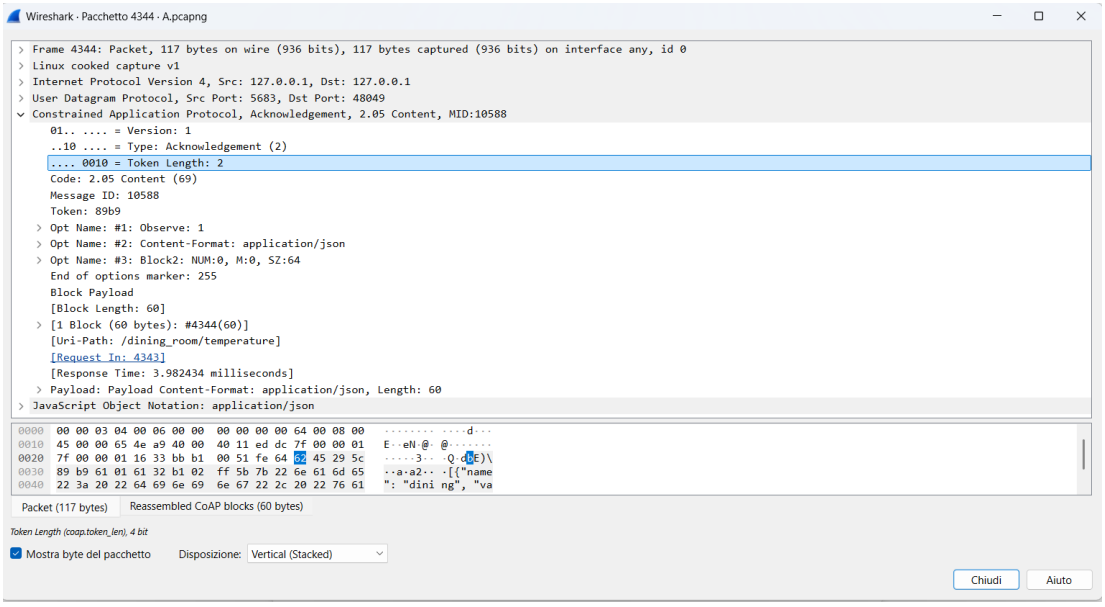
In the table are resumed all the transmissions:

Number	***MID***	***Type***	***Value***
1	10588	ACK	19.97
2	35	CON	22.32
3	38	CON	18.85
4	43	CON	25.98
5	45	CON	15.59
6	49	CON	25.35
7	54	CON	26.21
8	62	CON	27.53
9	74	CON	26.98
10	87	CON	25.32
11	90	CON	19.57

There are:

- 11 unique notifications
- 4 retransmissions

The ACK packet qualifies as a valid observe notification because it features a "2.05 Content" response code and includes the "Observe" option, indicating it successfully transmits the updated resource data. It is distinct from a simple acknowledgment due to its payload content, thus satisfying the criteria for separate notifications while excluding retransmissions as instructed.



Answer CQ3b
A notification is considered useless if it delivers the same value as the previous one, so there is no new information.
Comparing all the notifications, all the temperature values are distinct so there are no useless notifications.

Answer CQ3
CQ3a: ***Eleven unique observe notifications***
CQ3b: ***No useless notifications***

Question #4 - Analysis of MQTT-SN Message Reception from the Local Broker on Port 1885

Inspecting all 219 UDP packets on port 1885: everyone has dst_port = 1885, that is client to broker direction.
Filtering all UDP packets with source port 1885 (broker → client direction) yields zero results. No packet in has srcport=1885, confirming the broker sent no messages to any client.

Filter: udp.srcport == 1885

The formula counts the number of "/" characters (separators). The number of topic layers equals separators + 1. Topics with depth=2 contain one separator (e.g., "factory/room2"), depth=3 contain two (e.g., "hospital/area3/temperature"), etc. No single-segment or 5-segment topics were found in either capture, hence depth=1 and depth=5 are zero.

Answer CQ4

Zero messages received by clients from the broker

Question #5 – Last Will Delivery via Wildcard Subscription

Network Infrastructure

Broker identities were resolved by combining DNS A-record responses and link-layer analysis within the capture. One broker runs on the local IPv6 loopback interface and is not resolvable via DNS.

IP Address / Address	Hostname	Role
`::1 (IPv6 loopback)`	—	Local Mosquitto broker (same host)
`18.192.151.104`	broker.hivemq.com	Public HiveMQ broker
`35.158.43.69`	broker.hivemq.com	Public HiveMQ broker (DNS round-robin)
`5.196.78.28`	test.mosquitto.org	Public Mosquitto broker
`10.0.2.15`	—	Client machine (VirtualBox NAT)

MQTT Clients Identified

Forty-six CONNECT packets were found across all streams. Twenty-one of these originate from automated Mosquitto subscriber instances (client IDs prefixed mosq-, length 23) connecting to the local broker; they are excluded from the table below as they carry no Will and issue no SUBSCRIBE relevant to this analysis. The remaining 24 named clients are listed below.

Client Identifier	Len	Broker	MQTT v	Will ?	Will Topic
pcbtjyj	7	HiveMQ 35.158.43.69	3.1	No	—
cpoepjzkhibxgjiu	16	HiveMQ 18.192.151.104	3.1	No	—
tukvxesuhe	10	HiveMQ 18.192.151.104	3.1	No	—
fcthvjikxjul	12	HiveMQ 18.192.151.104	3.1	No	—
giuxfijwus	10	HiveMQ 18.192.151.104	3.1.1	No	—

atqeblw	7	HiveMQ 18.192.151.104	3.1.1	No	—
pohujuv	7	HiveMQ 18.192.151.104	5.0	No	—
dzcxnwdqef	10	HiveMQ 18.192.151.104	5.0	No	—
rolsmcwhshlbc	13	HiveMQ 18.192.151.104	5.0	No	—
wikdxsrbyewyocm	15	**Mosquitto 5.196.78.28**	**3.1.1**	**Yes**	**metaverse/room2/floor4**
lstvfjwz	8	**Mosquitto 5.196.78.28**	**5.0** ⚠	**Yes**	**hospital/facility3/area3**
kiurrznrgj	10	**Mosquitto 5.196.78.28**	**3.1.1**	**Yes**	**metaverse/room2/room2**
(empty)	0	Local ::1	3.1.1	**Yes**	**university/department12/room1/temperature**
auyvhrhdudnm	12	Local ::1	3.1.1	No	—
pcdwkgeslfh	11	Local ::1	3.1	No	—
yfyyxj	6	Local ::1	3.1.1	No	—
ggmqfceaplfl	12	Local ::1	3.1	No	—
sjinzrhqg	9	Local ::1	3.1	No	—
fixstgpixtt	11	Local ::1	3.1	No	—
zmjnxudohrkaegmh	16	**Local ::1**	**5.0** ⚠	No	—
frfltwzmz	9	Local ::1	3.1.1	No	—
mjdocmjxt	9	Local ::1	3.1.1	No	—
lgmfxxhwoa	10	Local ::1	5.0	No	—
ukikif	6	Local ::1	3.1	No	—

(⚠) Client `lstvfjwz` and client `zmjnxudohrkaegmh` use MQTT 5.0. Their frames include an additional **Properties Length** field not present in v3.1.1, which causes offset errors in v3.1.1-only parsers. This is discussed in detail under **Parsing Considerations**.

For a subscriber to receive a Last Will and Testament (LWT) message through a wildcard subscription, three conditions must hold simultaneously:

- A client must have registered a Will Topic in its CONNECT packet (`will_flag = 1`).
- Another client must have issued a SUBSCRIBE whose topic filter contains at least one wildcard (+ or #) on the same broker.
- The wildcard filter must match the Will Topic according to MQTT rules: + matches exactly one topic level; # matches one or more remaining levels and must appear last in the filter.

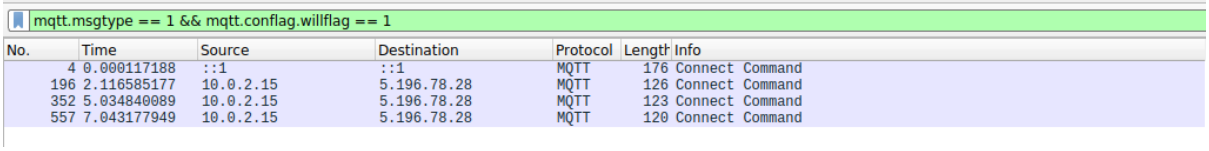
All three conditions must be satisfied on the same broker. A will registered on broker A is never delivered to subscribers of broker B: MQTT brokers are independent and do not bridge messages between each other.

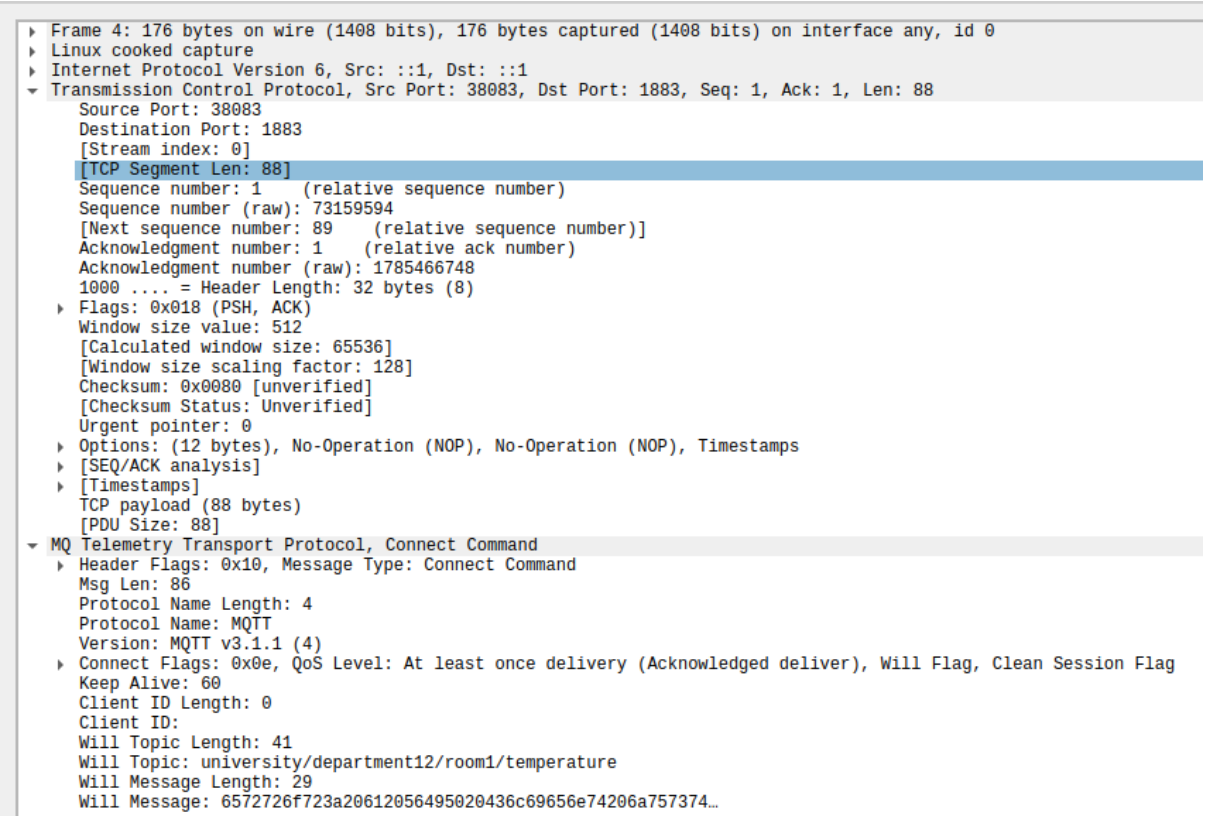
Wireshark Filters

Step 1 — Collect Will Topics

```
mqtt.msgtype == 1 && mqtt.conflag.willflag == 1
```

This filter is intentionally left without an IP-version restriction. The local broker runs on IPv6 loopback (::1), so constraining to ip.version == 4 would exclude the CONNECT packet carrying the will topic university/department12/room1/temperature, which travels over IPv6.

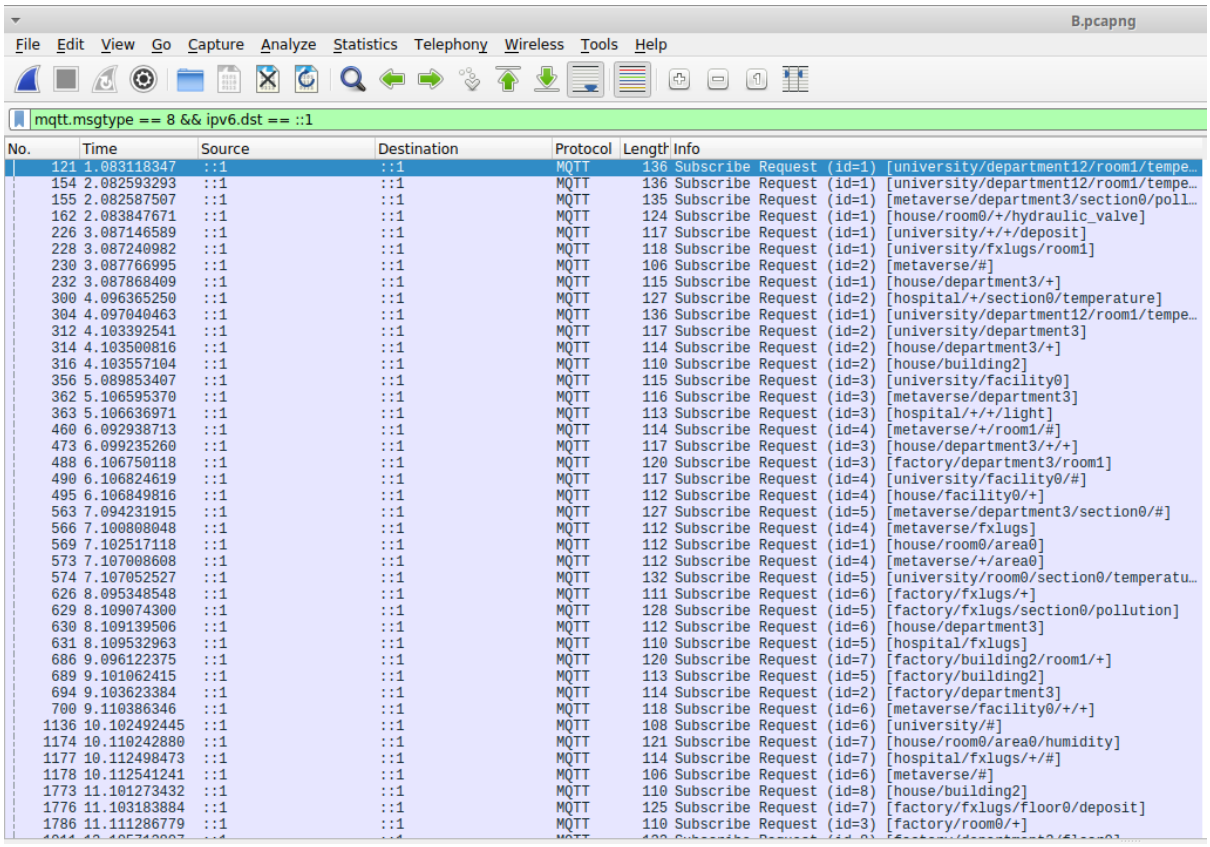




Step 2 — Collect Wildcard Subscriptions on the local broker

```
mqtt.msgtype == 8 && ipv6.dst == ::1
```

From the results, only topic filters containing + or # characters are retained. Wireshark does not expose a wildcard count field; the count is evaluated programmatically on the parsed filter strings.



Step 3 — Cross-check broker and topic matching

Each wildcard filter is evaluated against each Will Topic using the MQTT topic matching algorithm, considering only filters registered on the same broker as the corresponding will.

Parsing Considerations — MQTT v5 Compatibility

Two clients in the capture use MQTT 5.0: lstvfjwz (CONNECT to 5.196.78.28) and zmjnxudohrkaegmh (CONNECT and SUBSCRIBE to ::1). In MQTT 5.0, the CONNECT and SUBSCRIBE packets include a Properties section that is absent in v3.1.1. Even when no properties are set, a one-byte Properties Length field (0x00) is always present, inserted immediately after the Keepalive in CONNECT and after the Message ID in SUBSCRIBE.

A parser written exclusively for MQTT v3.1.1 reads this extra 0x00 byte as the high byte of the next length field (Client ID length in CONNECT, Topic Filter length in SUBSCRIBE). This shifts all subsequent field offsets by one byte, producing:

- In CONNECT: Client ID length reads as 0 (empty), then the actual Client ID bytes are misinterpreted as the Will Topic length, producing a value far beyond the remaining payload. The will topic is never read.
- In SUBSCRIBE: Topic Filter length reads as 0 (empty topic), and the following byte is misread as QoS. Both outcomes (empty topic, out-of-range QoS such as 15 or 21) are syntactically legal in MQTT, so no error is raised — the parser silently discards the data and moves on.

The fix requires detecting the Protocol Level byte in each client's CONNECT (0x04 for v3.1.1, 0x05 for v5.0) and applying the appropriate parsing logic for all subsequent packets on that TCP flow. Wireshark handles this automatically; a custom parser must do it explicitly. Without this correction, the will topic of lstvfjwz and the topic filters of zmjnxudohrkaegmh — including the decisive university/# — would not be recovered.

Results

Four Will Topics were registered across the capture:

Client	Broker	Will Topic
wikdxsrbyewyocm	test.mosquitto.org (5.196.78.28)	metaverse/room2/floor4
lstvfjwz (MQTT 5.0)	test.mosquitto.org (5.196.78.28)	hospital/facility3/area3
kiurrznrgj	test.mosquitto.org (5.196.78.28)	metaverse/room2/room2
(empty) ← key	**Local broker ::1**	**university/department12/room1/temperature**

For the three Will Topics on test.mosquitto.org (5.196.78.28), no SUBSCRIBE message directed to that broker appears in the capture: the three connected clients exclusively sent CONNECT and PUBLISH packets. Because brokers are independent, any subscriber on a different broker cannot receive those wills regardless of filter content.

For the Will Topic on the local broker ::1, the wildcard subscriptions registered on the same broker were evaluated. The relevant match is:

Subscriber	Topic Filter	Will Topic	Match?
zmjnxudohrkaegmh (MQTT 5.0)	**`university/#`**	**`university/department12/room1/temperature`**	**YES**
(other local subscribers)	university/+/+/deposit, university/department3/#, etc.	university/department12/room1/temperature	No — level mismatch or final level mismatch

The filter `university/#` matches because the `#` wildcard covers any number of remaining topic levels. The topic `university/department12/room1/temperature` starts with `university/` and has further levels — the condition is satisfied. All other wildcard filters on the local broker fail due to a fixed-level mismatch in the second or last level (e.g., `university/department3/#` does not match `department12`; `university/+/+/deposit` does not match the last level `temperature`).

The subscription `university/#` was only recovered after applying the correct MQTT v5 parser to the frames from client `zmjnxudohrkaegmh`. With a v3.1.1 parser, all topic filters from this client appear as empty strings and are silently dropped (see Parsing Considerations above).

Answer to CQ5: ***1***

Question #6 – Clients Erasing Retained Values on HiveMQ

In MQTT, a retained message is stored by the broker and delivered immediately to any future subscriber on that topic. The only way to erase a retained value is to publish to that exact topic with the RETAIN flag set to 1 and a zero-byte payload. The broker then deletes the stored message.

The analysis identifies PUBLISH packets satisfying all the following:

- MQTT message type = 3 (PUBLISH)
- RETAIN flag = 1
- Payload length = 0 bytes (mqtt.len == 0)
- Destination IP is a HiveMQ broker (18.192.151.104 or 35.158.43.69)
- Direction is Client to Broker (ip.dst is the broker address)

Following the instruction to ignore retransmissions, each packet is deduplicated based on its (TCP stream, sequence number) pair. In this capture, no TCP retransmissions were detected among the erase packets: the count before and after deduplication is identical (50 packets in both cases, 0 removed). Each stream is associated with a client identifier by matching the source port and broker IP to the CONNECT packet seen on the same TCP connection.

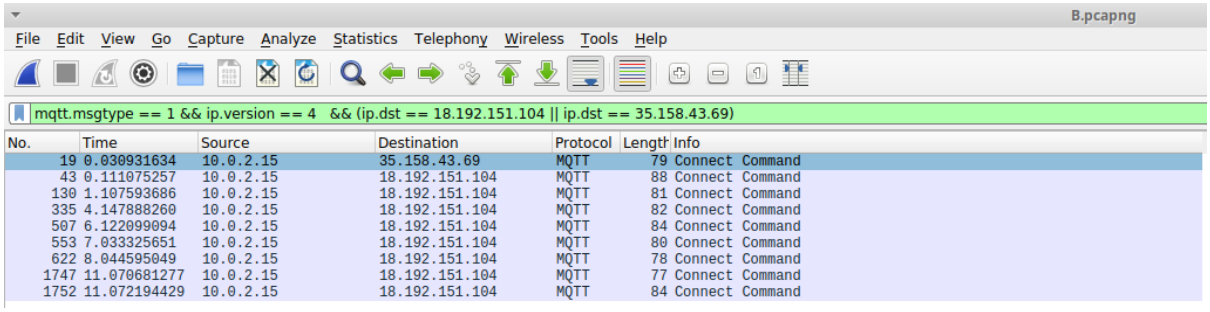
Wireshark filters

Erase PUBLISH packets — unique only

```
mqtt.msgtype == 3 && mqtt.retain == 1 && mqtt.len == 0
&& (ip.dst == 18.192.151.104 || ip.dst == 35.158.43.69)
&& ip.version == 4 && !tcp.analysis.retransmission
```

CONNECT packets to HiveMQ — to retrieve client identifiers

```
mqtt.msgtype == 1 && ip.version == 4
&& (ip.dst == 18.192.151.104 || ip.dst == 35.158.43.69)
```



CQ6b — client identifier length filter

mqtt.msgtype == 1 && mqtt.clientid_len > 7 && ip.version == 4

&& (ip.dst == 18.192.151.104 || ip.dst == 35.158.43.69)

Results — CQ6a

Fifty unique erase-publish packets were found, sent by three distinct client identifiers:

Client Identifier	Broker	Erase Messages Sent
pcbtjyj	HiveMQ 35.158.43.69	27
atqeblw	HiveMQ 18.192.151.104	12
giuxfijwus	HiveMQ 18.192.151.104	11

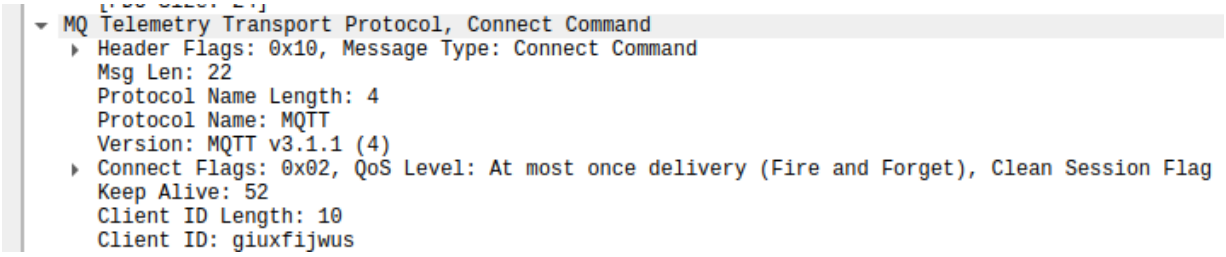
***Answer to CQ6a:* 3**

Results — CQ6b

The three clients from CQ6a are evaluated against the strict length threshold (> 7, i.e., at least 8 bytes):

Client Identifier	Length (bytes)	Strictly > 7?
pcbtjyj	7	No — equal to 7, threshold requires strictly greater
atqeblw	7	No — equal to 7, threshold requires strictly greater
giuxfijwus	10	Yes

Only giuxfijwus, with a 10-byte identifier, satisfies the strict inequality. Both pcbtjyj and atqeblw are exactly 7 bytes and therefore do not qualify.



Answer to CQ6b: *1*

Question #7 – SUBSCRIBE Requests to the Local Broker with At Least Two Wildcards

The local broker in this capture is the Mosquitto instance running on the IPv6 loopback address ::1 (TCP port 1883). A topic filter with at least two wildcards is one where the combined count of + and # characters is greater than or equal to 2 (e.g., hospital/+/+/light or metaverse/+/room1/#). The # wildcard, if present, must appear as the last level.

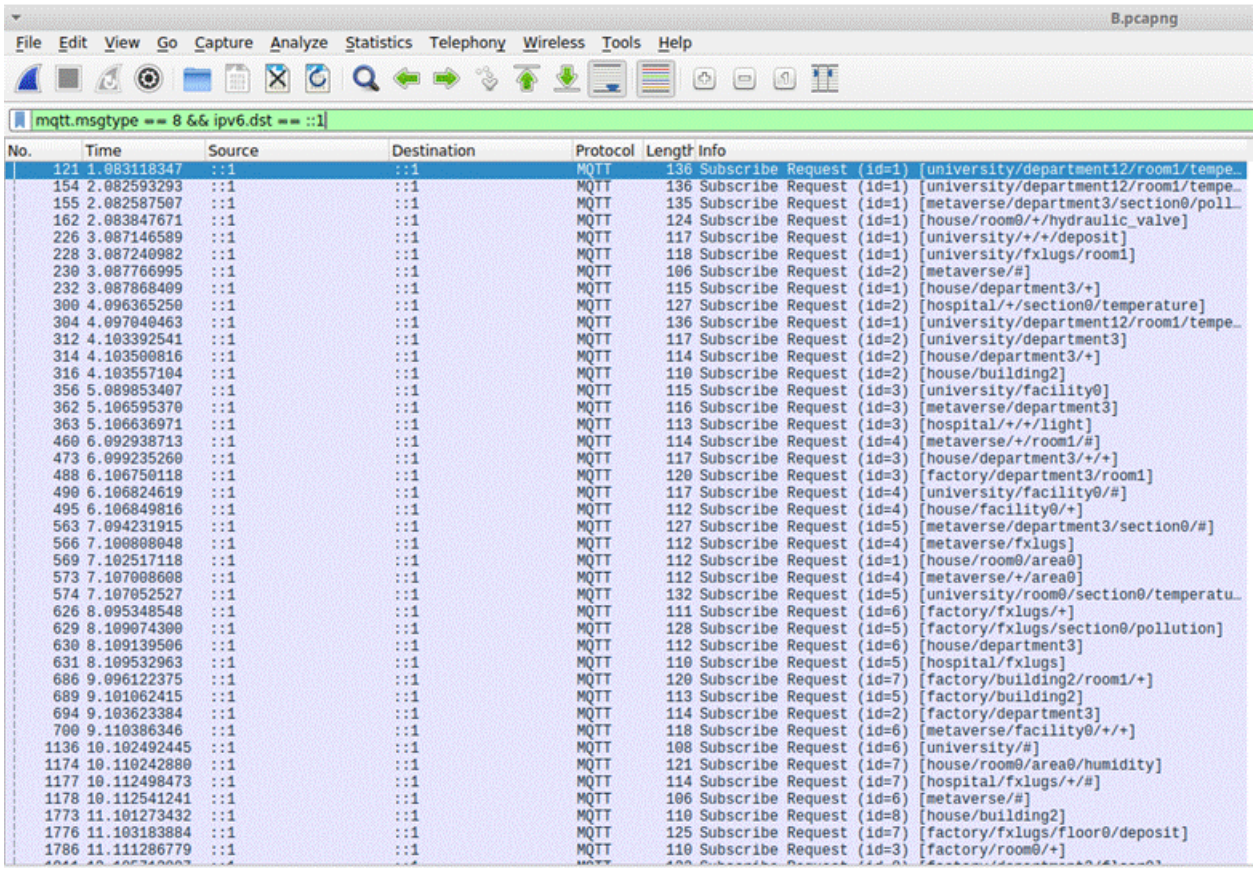
Two conditions must hold: the SUBSCRIBE must be addressed to the local broker (::1), and at least one topic filter within that SUBSCRIBE must contain two or more wildcard characters.

Wireshark Filters

SUBSCRIBE packets to the local broker with at least two wildcards per filter string

```
mqtt.msgtype == 8 && ipv6.dst == ::1
```

The filter uses `ipv6.dst == ::1` rather than `ip.dst` because the local broker is reachable only over IPv6 loopback. A filter restricted to IPv4 (`ip.version == 4`) would return zero results and incorrectly suggest no SUBSCRIBE packets were sent to the local broker. Wireshark does not expose a wildcard count field; an additional check must be performed programmatically on the parsed filter strings: only filters where count('+') + count('#') ≥ 2 are retained.



Note on MQTT v5: the 21 `mosq-` automated subscriber instances and client `zmjnxudohrkaegmh` all connect to the local broker. The mosq- clients issue no SUBSCRIBE and are irrelevant to this question. Client `zmjnxudohrkaegmh` uses MQTT 5.0 and its SUBSCRIBE frames include the extra Properties Length byte described in the Parsing Considerations section of CQ5. Without the corrected parser, the two multi-wildcard filters from this client are invisible.

Results

Eighty-three SUBSCRIBE messages were found directed to the local broker ::1, each carrying exactly one topic filter. Nine topic filters contain two or more wildcard characters:

#	Topic Filter	Wildcards	Subscriber Client	MQTT
1	`university/+/+/deposit`	+, +	(port :44613)	3.1.1
2	`hospital/+/+/light`	+, +	(port :43797)	3.1.1
3	`metaverse/+/room1/#`	+, #	(port :42777)	3.1.1

4	`metaverse/facility0/+/+`	`+, +`	(port :53121)	3.1.1
5	`hospital/fxlugs/+/#`	`+, #`	(port :53121)	3.1.1
6	`metaverse/+/+/hydraulic_valve`	`+, +`	(port :53121)	3.1.1
7	`factory/+/#`	`+, #`	(port :43797)	3.1.1
8	**`house/department3/+/+`**	**`+, +`**	**zmjnxudohrkaegmh (port :51743)**	**5.0** ⚠
9	**`house/+/room1/#`**	**`+, #`**	**zmjnxudohrkaegmh (port :51743)**	**5.0** ⚠

Entries 8 and 9 were recovered only after applying the correct MQTT v5 parser to client `zmjnxudohrkaegmh`. With a v3.1.1 parser, this client's Properties Length byte (0x00) is absorbed into the Topic Filter length field, yielding length = 0 and an empty topic string, followed by the next byte being misread as QoS (obtaining out-of-range values such as 15 or 21). Both outcomes are syntactically valid in MQTT, so the parser discards the entries silently. The actual filters `house/department3/+/+` and `house/+/room1/#` — each containing two wildcard characters — are therefore invisible without version-aware parsing.

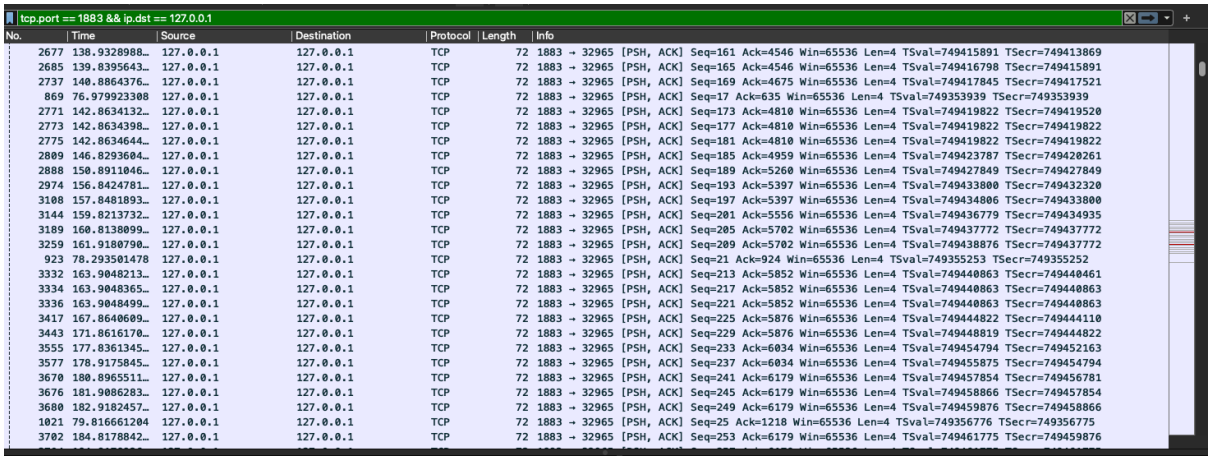
Answer to CQ7: 9

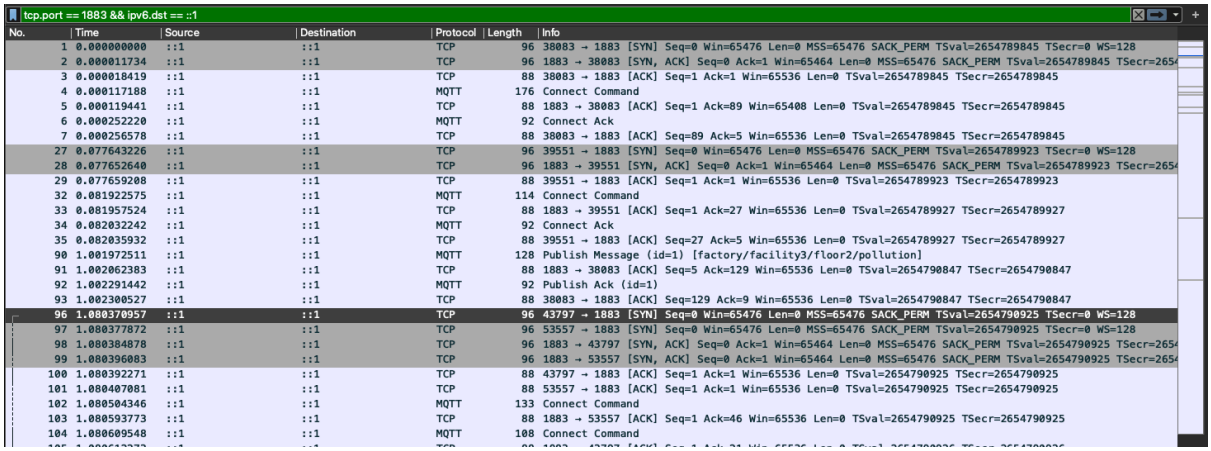
Question #8 – MQTT PUBLISH

First, we apply the following filter to search for base packets:

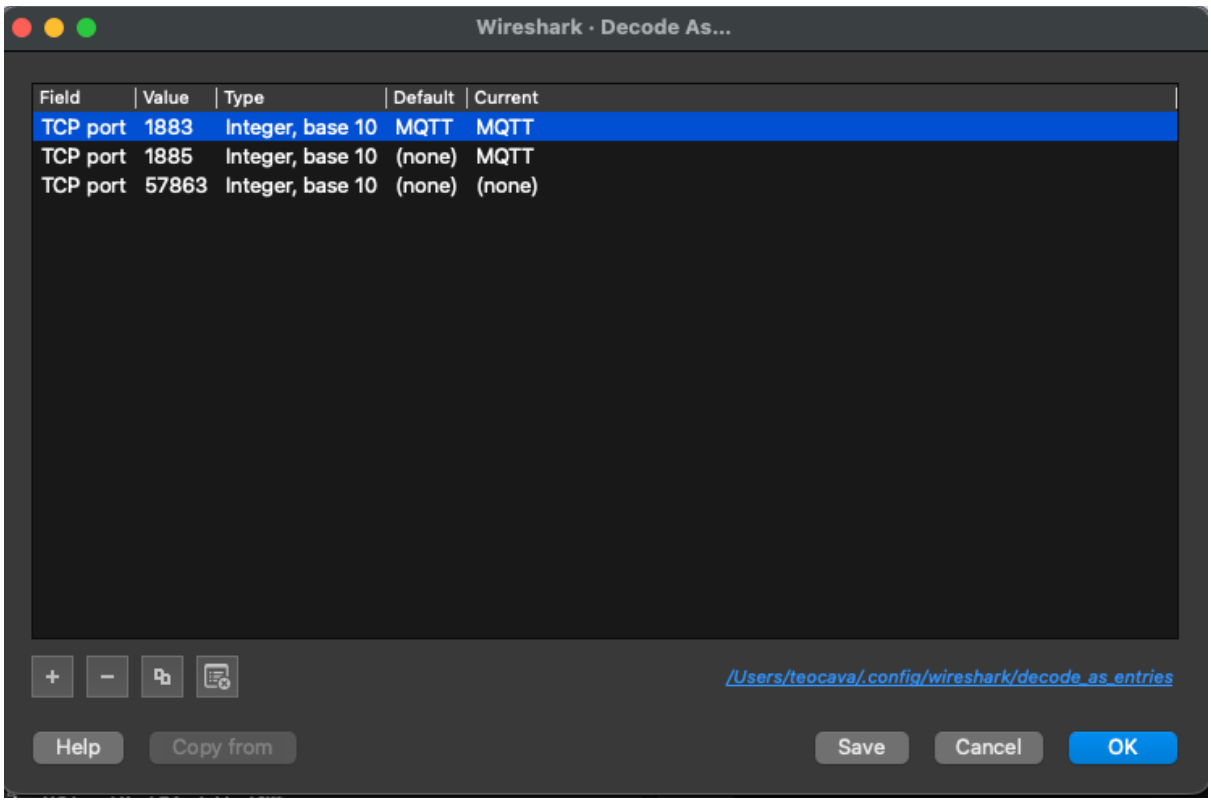
Filter: tcp.port == 1883 && ip.dst == 127.0.0.1 (A.pcapng)

Filter: tcp.port == 1883 && ipv6.dst == ::1 (B.pcapng)



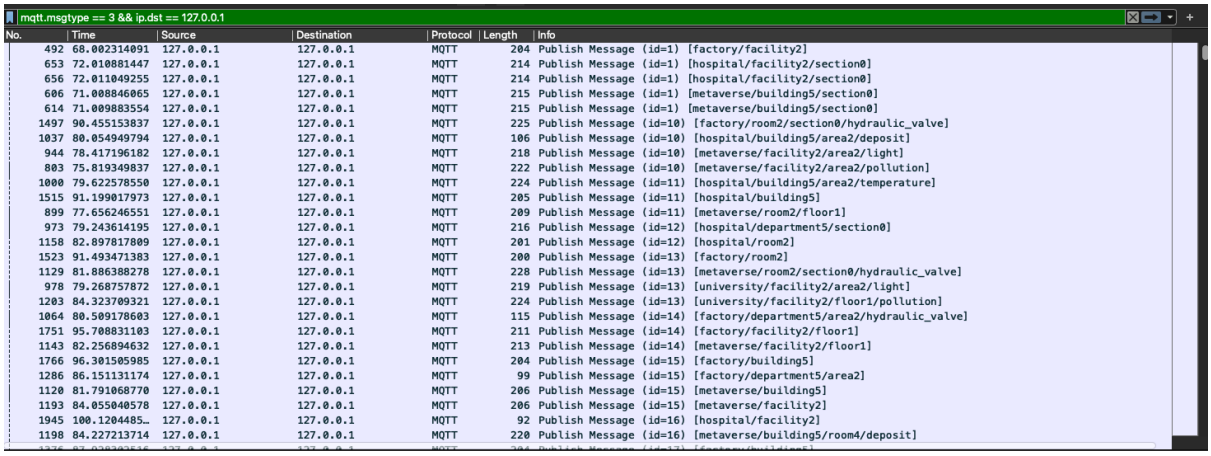


Then is necessary to decode all the packets from TCP to MQTT and, by applying the following filters, is possible to identify the local broker messages distribution:



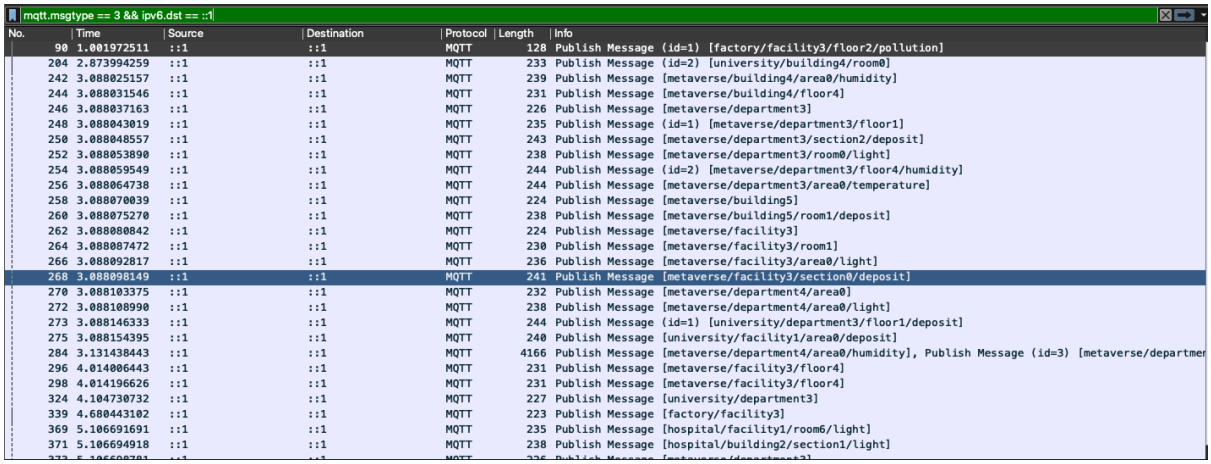
File: A.pcapng

Filter: mqtt.msgtype == 3 && ip.dst == 127.0.0.1



File: B.pcapng

Filter: mqtt.msgtype == 3 && ipv6.dst == ::1



mqtt.msgtype == 3 && ipv6.dst == ::1

No.	Time	Source	Destination	Protocol	Length	Info
90	1.001972511	::1	::1	MQTT	128	Publish Message (id=1) [factory/facility3/floor2/pollution]
204	2.873994259	::1	::1	MQTT	233	Publish Message (id=2) [university/building4/room0]
242	3.088025157	::1	::1	MQTT	239	Publish Message [metaverse/building4/area0/humidity]
244	3.088031546	::1	::1	MQTT	231	Publish Message [metaverse/building4/floor4]
246	3.088037163	::1	::1	MQTT	226	Publish Message [metaverse/department3]
248	3.088043019	::1	::1	MQTT	235	Publish Message (id=1) [metaverse/department3/floor1]
250	3.088048557	::1	::1	MQTT	243	Publish Message [metaverse/department3/section2/deposit]
252	3.088053890	::1	::1	MQTT	238	Publish Message [metaverse/department3/room0/light]
254	3.088059549	::1	::1	MQTT	244	Publish Message (id=2) [metaverse/department3/floor4/humidity]
256	3.088064738	::1	::1	MQTT	244	Publish Message [metaverse/department3/area0/temperature]
258	3.088070039	::1	::1	MQTT	224	Publish Message [metaverse/building5]
260	3.088075270	::1	::1	MQTT	238	Publish Message [metaverse/building5/room1/deposit]
262	3.088080842	::1	::1	MQTT	224	Publish Message [metaverse/facility3]
264	3.088087472	::1	::1	MQTT	230	Publish Message [metaverse/facility3/room1]
266	3.088092817	::1	::1	MQTT	236	Publish Message [metaverse/facility3/area0/light]
268	3.088098149	::1	::1	MQTT	241	Publish Message [metaverse/facility3/section0/deposit]
270	3.088103375	::1	::1	MQTT	232	Publish Message [metaverse/department4/area0]
272	3.088108990	::1	::1	MQTT	238	Publish Message [metaverse/department4/area0/light]
273	3.088146333	::1	::1	MQTT	244	Publish Message (id=1) [university/department3/floor1/deposit]
275	3.088154395	::1	::1	MQTT	240	Publish Message [university/facility1/area0/deposit]
284	3.131438443	::1	::1	MQTT	4166	Publish Message [metaverse/department4/area0/humidity], Publish Message (id=3) [metaverse/departmer
296	4.014006443	::1	::1	MQTT	231	Publish Message [metaverse/facility3/floor4]
298	4.014196626	::1	::1	MQTT	231	Publish Message [metaverse/facility3/floor4]
324	4.104730732	::1	::1	MQTT	227	Publish Message [university/department3]
339	4.680443102	::1	::1	MQTT	223	Publish Message [factory/facility3]
369	5.106691691	::1	::1	MQTT	235	Publish Message [hospital/facility1/room6/light]
371	5.106694918	::1	::1	MQTT	238	Publish Message [hospital/building2/section1/light]

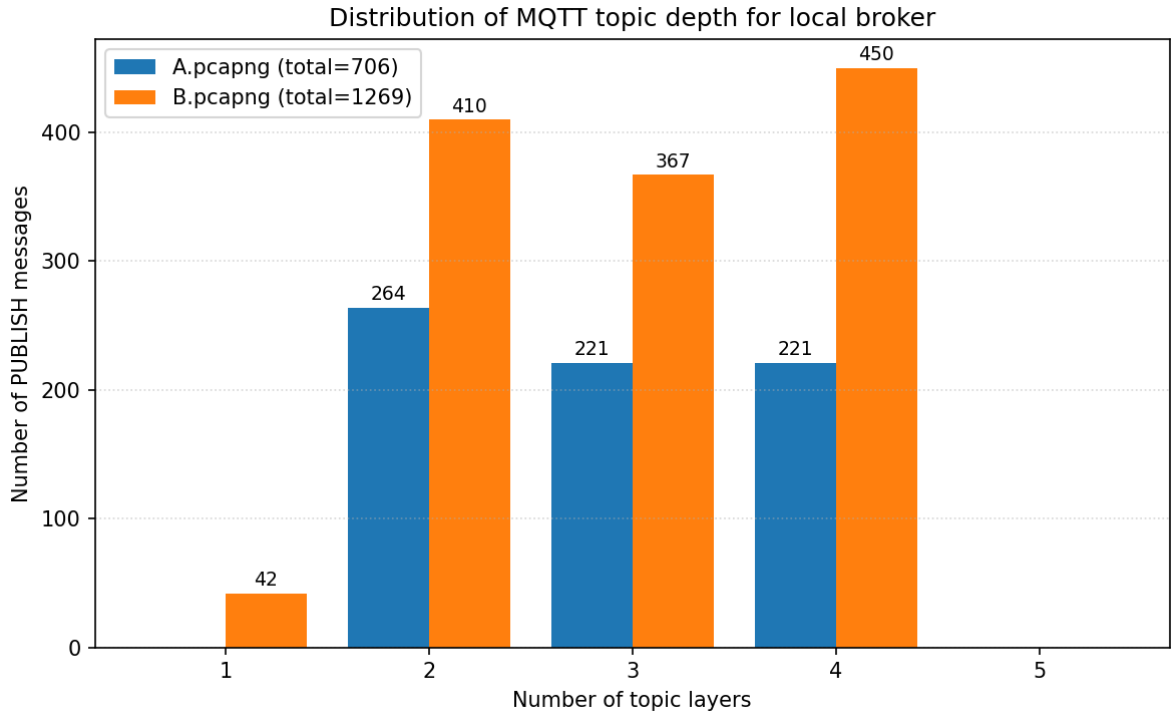
The depth is defined by the numbers of "/" separated layers. Extracting the packets in a .CSV file, filtering the Topic Name with:

"=LUNGHEZZA(G632)- LUNGHEZZA (SOSTITUISCI(G632;"/";"")) + 1"

is possible to identify the value of the depth.
In the table are resumed all the values:

Depth	***A.pcapng***	***B.pcapng***
1	0	42
2	264	410
3	221	367
4	221	450
5	0	0

In the figure is shown the distribution of MQTT topic depth for local broker:



Answer CQ8

Total PUBLISH to local broker in ***A.pcapng = 706***

Total PUBLISH to local broker in ***B.pcapng = 1269***