



University of Moratuwa, Sri Lanka

Faculty of Engineering

Department of Electronics and Telecommunication Engineering
Semester 5 (Intake 2021)

EN3251 - Internet of Things

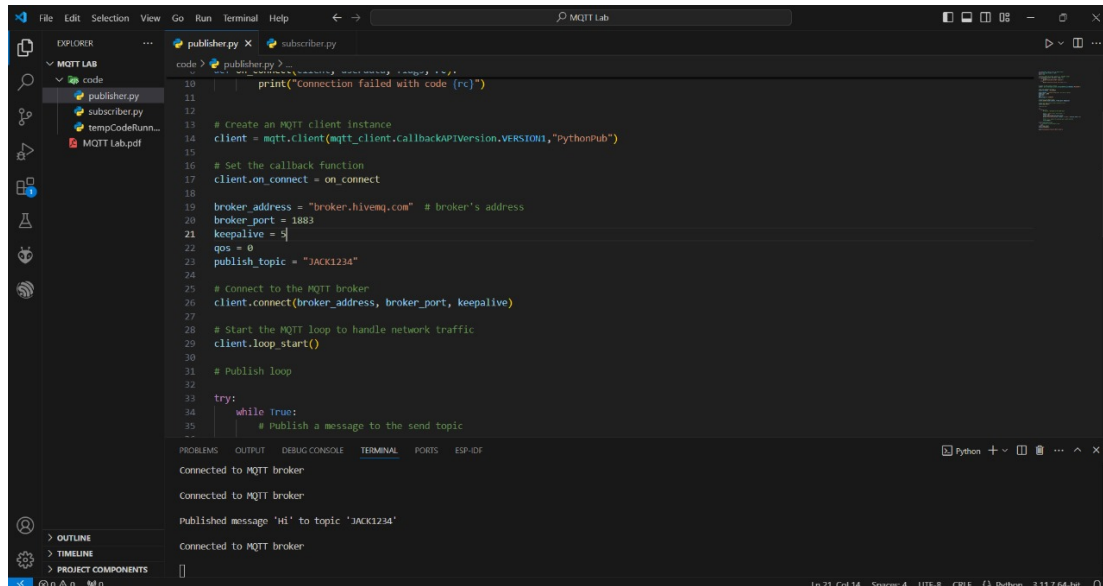
MQTT Implementation and Testing

Gunawardana W.N.M.	210199D
Dilshan N.L.	210129P
Sehara G.M.M.	210583B

This report is submitted as a partial fulfillment for the module EN3251 - Internet of Things, Department of Electronic and Telecommunication Engineering, University of Moratuwa.

1 Step 2 - QOS=0

1.1 Publisher



```
code > publisher.py
10 print("Connection failed with code (rc)")
11
12
13 # Create an MQTT client instance
14 client = mqtt.Client(mqtt_client.callbackAPIVersion.VERSION1,"pythonPub")
15
16 # Set the callback function
17 client.on_connect = on_connect
18
19 broker_address = "broker.hivemq.com" # broker's address
20 broker_port = 1883
21 keepalive = 5
22 qos = 0
23 publish_topic = "JACK1234"
24
25 # Connect to the MQTT broker
26 client.connect(broker_address, broker_port, keepalive)
27
28 # Start the MQTT loop to handle network traffic
29 client.loop_start()
30
31 # Publish loop
32 try:
33     while True:
34         # Publish a message to the send topic
35         client.publish(publish_topic, "Hi")
36         time.sleep(1)
37 except KeyboardInterrupt:
38     client.loop_stop()
39     sys.exit(0)
40
```

Connected to MQTT broker
Connected to MQTT broker
Published message 'Hi' to topic 'JACK1234'
Connected to MQTT broker

Figure 1: Python code for publisher (QOS - 0)

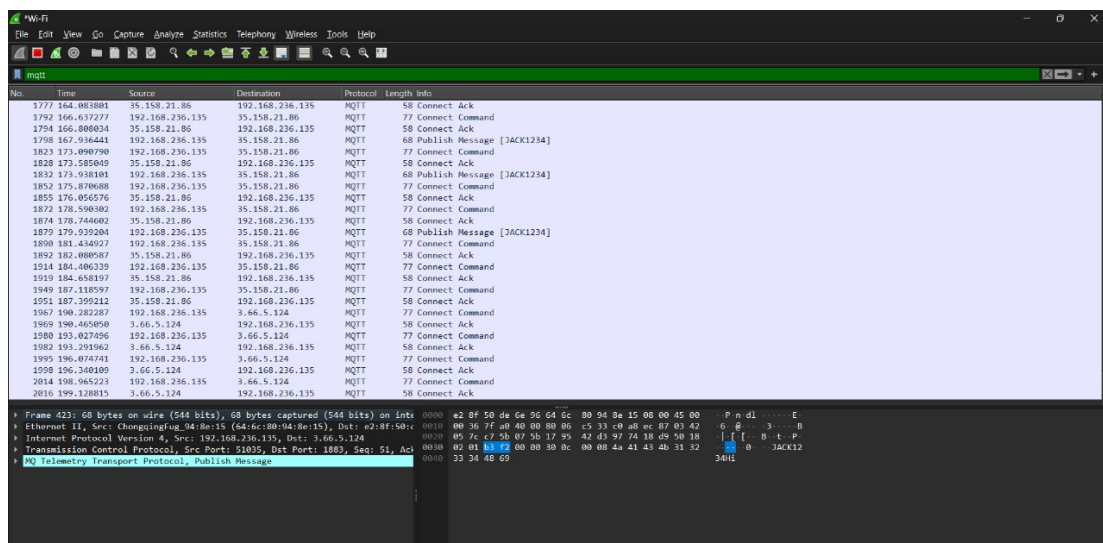


Figure 2: Wireshark capture for the publisher. (QOS - 0)

1.2 Subscriber

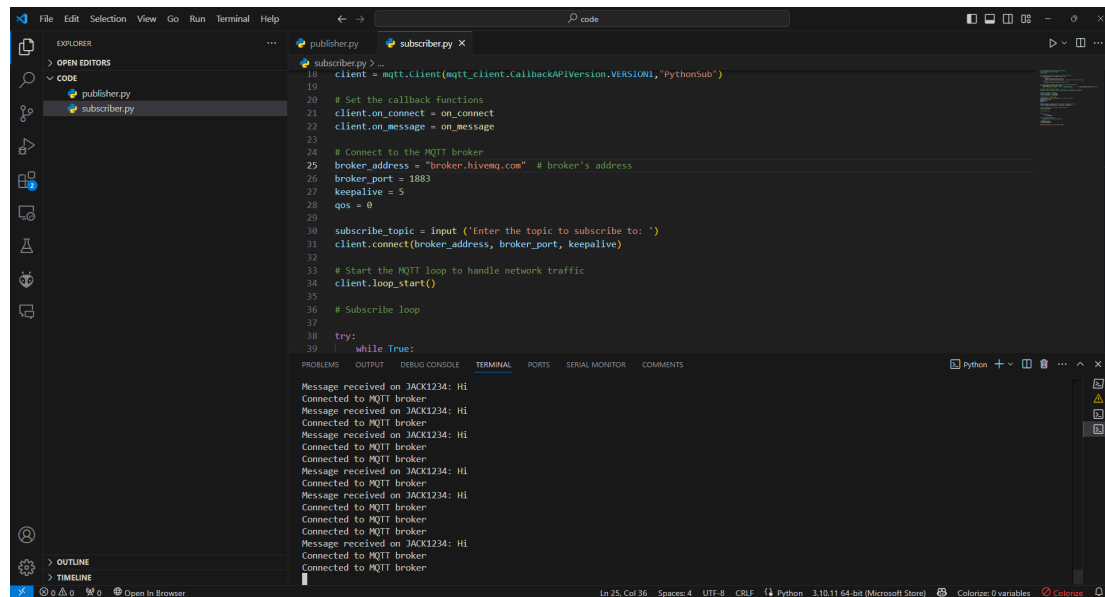


Figure 3: Python code for subscriber (QOS - 0)

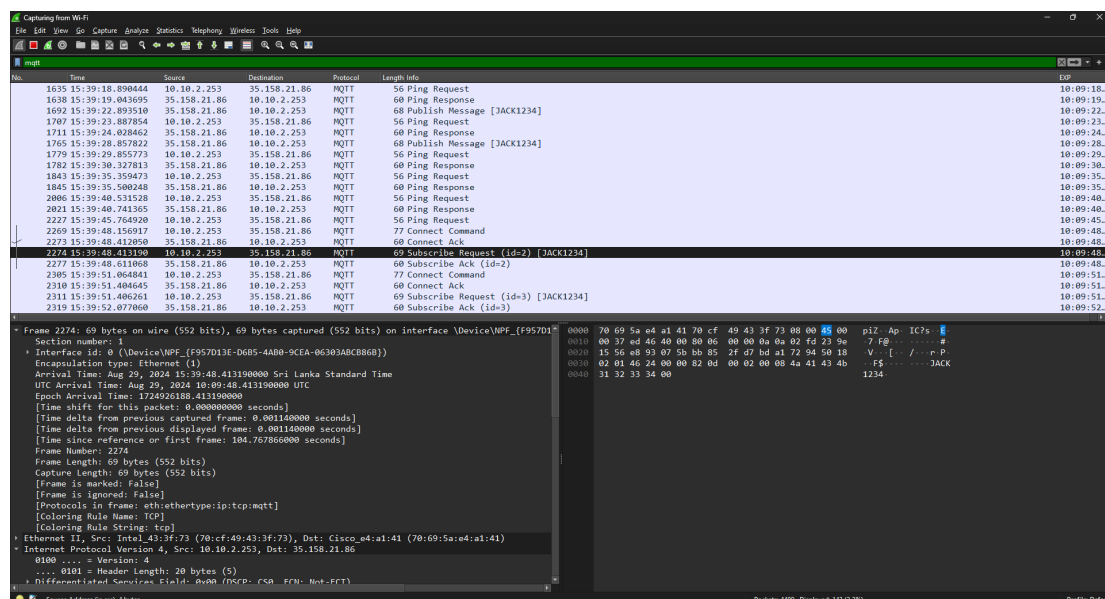


Figure 4: Wireshark capture for the subscriber. (QOS - 0)

1.3 MQTT.Cool Test Client

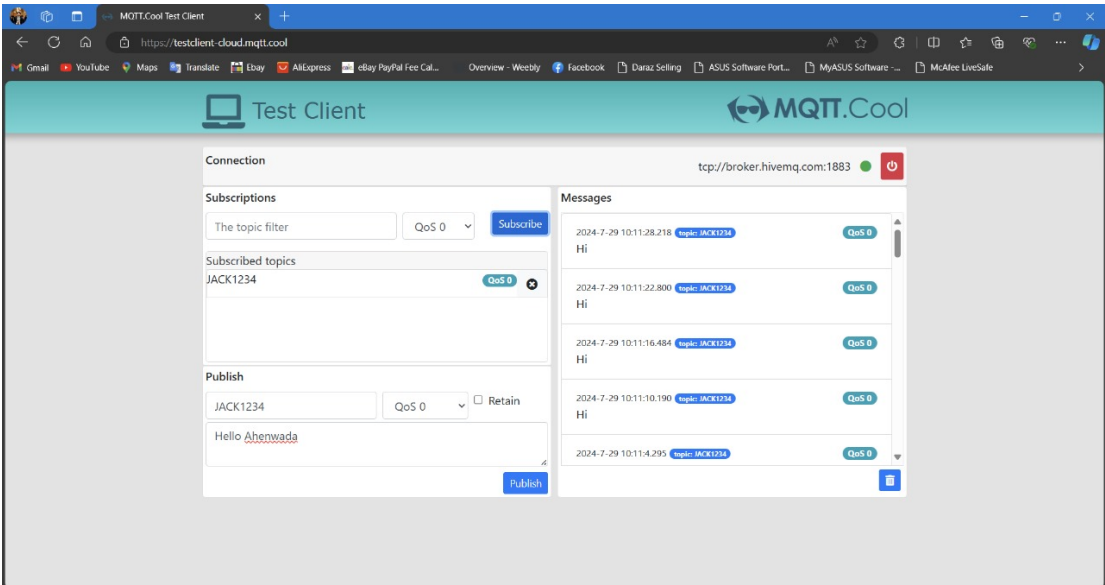
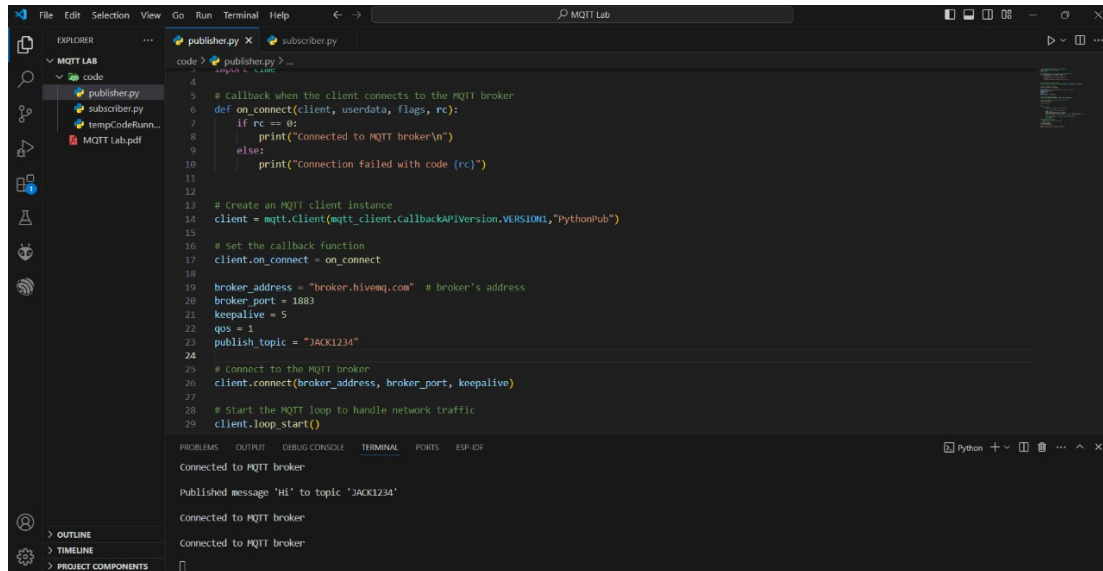


Figure 5: MQTT.cool results (QOS - 0)

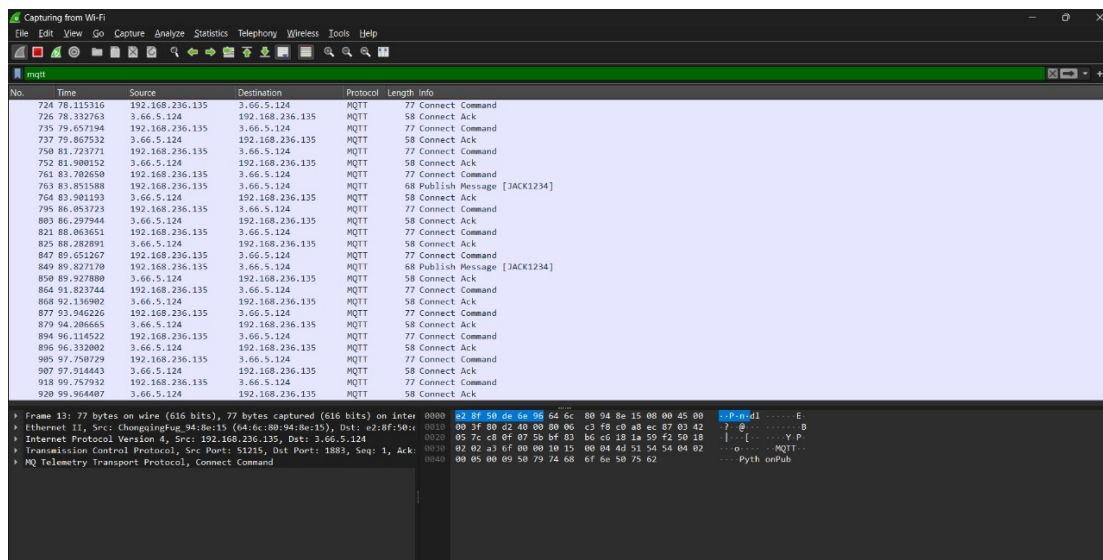
2 Step 3.A - QOS=1

2.1 Publisher



```
File Edit Selection View Go Run Terminal Help
MQTT Lab
EXPLORER
  code
    publisher.py
    subscriber.py
    tempCodefun...
    MQTT Lab.pdf
  ...
code > publisher.py > ...
4
5 # callback when the client connects to the MQTT broker
6 def on_connect(client, userdata, flags, rc):
7     if rc == 0:
8         print("Connected to MQTT broker\n")
9     else:
10        print("Connection failed with code {rc}")
11
12
13 # Create an MQTT client instance
14 client = mqtt.Client(mqtt_client.CallbackAPIVersion.VERSION1, "PythonPub")
15
16 # Set the callback function
17 client.on_connect = on_connect
18
19 broker_address = "broker.hivemq.com" # broker's address
20 broker_port = 1883
21 keepalive = 5
22 qos = 1
23 publish_topic = "JACK1234"
24
25 # Connect to the MQTT broker
26 client.connect(broker_address, broker_port, keepalive)
27
28 # Start the MQTT loop to handle network traffic
29 client.loop_start()
30
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS ESP-IDF
Python + ~
Connected to MQTT broker
Published message 'Hi' to topic 'JACK1234'
Connected to MQTT broker
Connected to MQTT broker
```

Figure 6: Python code for publisher (QOS - 1)

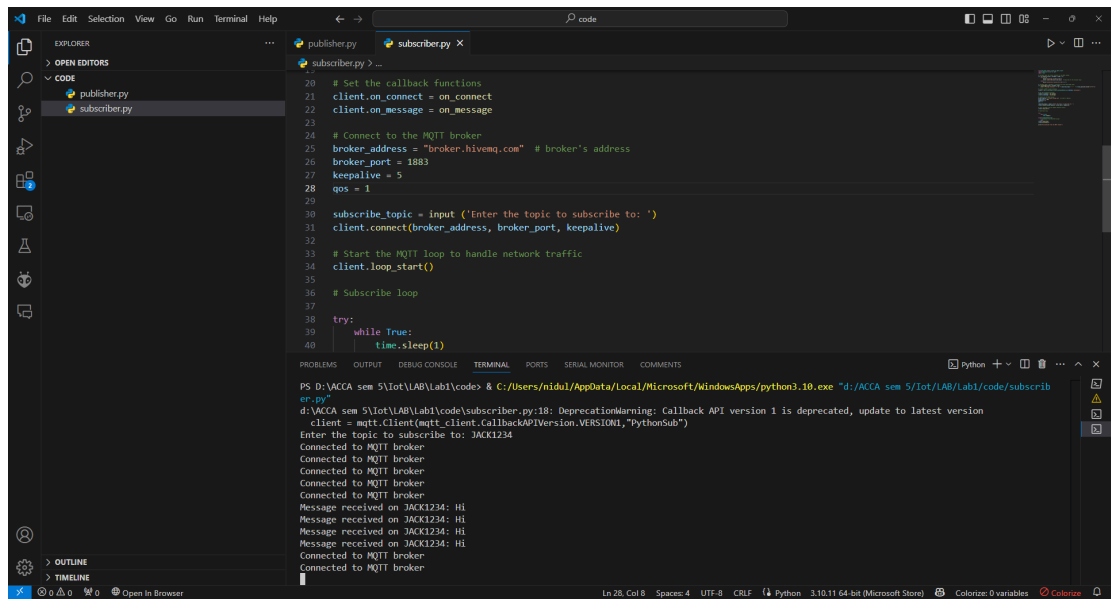


No.	Time	Source	Destination	Protocol	Length	Info
724	78.115316	192.168.236.135	3.66.5.124	MQTT	77	Connect Command
726	78.232763	3.66.5.124	192.168.236.135	MQTT	58	Connect Ack
735	79.657194	192.168.236.135	3.66.5.124	MQTT	77	Connect Command
737	79.867532	3.66.5.124	192.168.236.135	MQTT	58	Connect Ack
740	81.723771	192.168.236.135	3.66.5.124	MQTT	77	Connect Command
752	81.980152	3.66.5.124	192.168.236.135	MQTT	58	Connect Ack
761	83.702650	192.168.236.135	3.66.5.124	MQTT	77	Connect Command
763	83.851588	192.168.236.135	3.66.5.124	MQTT	68	Publish Message [JACK1234]
764	83.901193	3.66.5.124	192.168.236.135	MQTT	58	Connect Ack
795	86.053723	192.168.236.135	3.66.5.124	MQTT	77	Connect Command
803	86.297944	3.66.5.124	192.168.236.135	MQTT	58	Connect Ack
821	88.063651	192.168.236.135	3.66.5.124	MQTT	77	Connect Command
825	88.202891	3.66.5.124	192.168.236.135	MQTT	58	Connect Ack
847	89.651267	192.168.236.135	3.66.5.124	MQTT	77	Connect Command
849	89.827170	192.168.236.135	3.66.5.124	MQTT	68	Publish Message [JACK1234]
850	89.927880	3.66.5.124	192.168.236.135	MQTT	58	Connect Ack
864	91.823744	192.168.236.135	3.66.5.124	MQTT	77	Connect Command
868	92.136902	3.66.5.124	192.168.236.135	MQTT	58	Connect Ack
877	93.946226	192.168.236.135	3.66.5.124	MQTT	77	Connect Command
879	94.206665	3.66.5.124	192.168.236.135	MQTT	58	Connect Ack
894	96.114522	192.168.236.135	3.66.5.124	MQTT	77	Connect Command
896	96.332002	3.66.5.124	192.168.236.135	MQTT	58	Connect Ack
895	97.750729	192.168.236.135	3.66.5.124	MQTT	77	Connect Command
907	97.914443	3.66.5.124	192.168.236.135	MQTT	58	Connect Ack
918	99.757932	192.168.236.135	3.66.5.124	MQTT	77	Connect Command
920	99.964407	3.66.5.124	192.168.236.135	MQTT	58	Connect Ack

Frame 13: 77 bytes on wire (616 bits), 77 bytes captured (616 bits) on interface 0
Ethernet II, Src: ChanginGigap, 94:8e:15:64:8c:15 (64:8c:80:94:8e:15), Dst: e2:8f:50:00:00:00 (02:8f:50:00:00:00)
Internet Protocol Version 4, Src: 192.168.236.135, Dst: 3.66.5.124
Transmission Control Protocol, Src Port: 51215, Dst Port: 1883, Seq: 1, Ack: 1
MQ Telemetry Transport Protocol, Connect Command

Figure 7: Wireshark capture for the publisher. (QOS - 1)

2.2 Subscriber



```
20 # Set the callback functions
21 client.on_connect = on_connect
22 client.on_message = on_message
23
24 # Connect to the MQTT broker
25 broker_address = "broker.hivemq.com" # broker's address
26 broker_port = 1883
27 keepalive = 5
28 qos = 1
29
30 subscribe_topic = input('Enter the topic to subscribe to: ')
31 client.connect(broker_address, broker_port, keepalive)
32
33 # Start the MQTT loop to handle network traffic
34 client.loop_start()
35
36 # Subscribe loop
37
38 try:
39     while True:
40         time.sleep(1)
```

PS D:\ACCA sem 5\IoT\LAB\code> & C:/Users/nidul/AppData/Local/Microsoft/WindowsApps/python3.10.exe "d:\ACCA sem 5\IoT\LAB\code\subscriber.py"

d:\ACCA sem 5\IoT\LAB\code\subscriber.py:18: DeprecationWarning: Callback API version 1 is deprecated, update to latest version

client = mqtt.Client(mqtt_client.CallbackAPIVersion.VERSION1, "PythonSub")

Enter the topic to subscribe to: JACK1234

Connected to MQTT broker

Connected to MQTT broker

Connected to MQTT broker

Connected to MQTT broker

Message received on JACK1234: HI

Message received on JACK1234: HI

Message received on JACK1234: HI

Message received on JACK1234: HI

Connected to MQTT broker

Connected to MQTT broker

Figure 8: Python code for subscriber (QOS - 1)

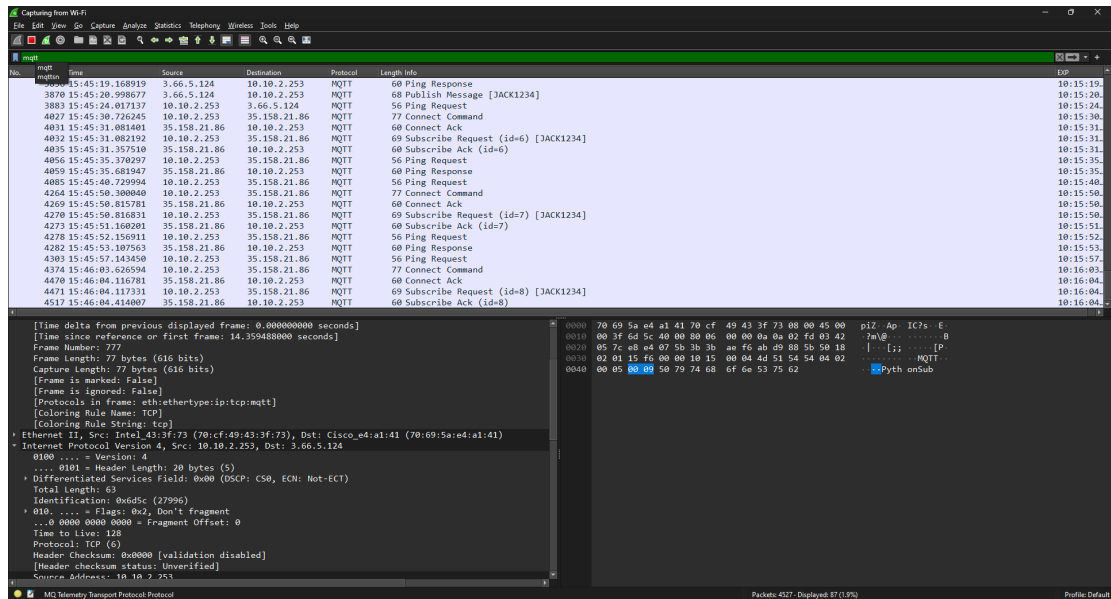


Figure 9: Wireshark capture for the subscriber. (QOS - 1)

2.3 MQTT.Cool Test Client

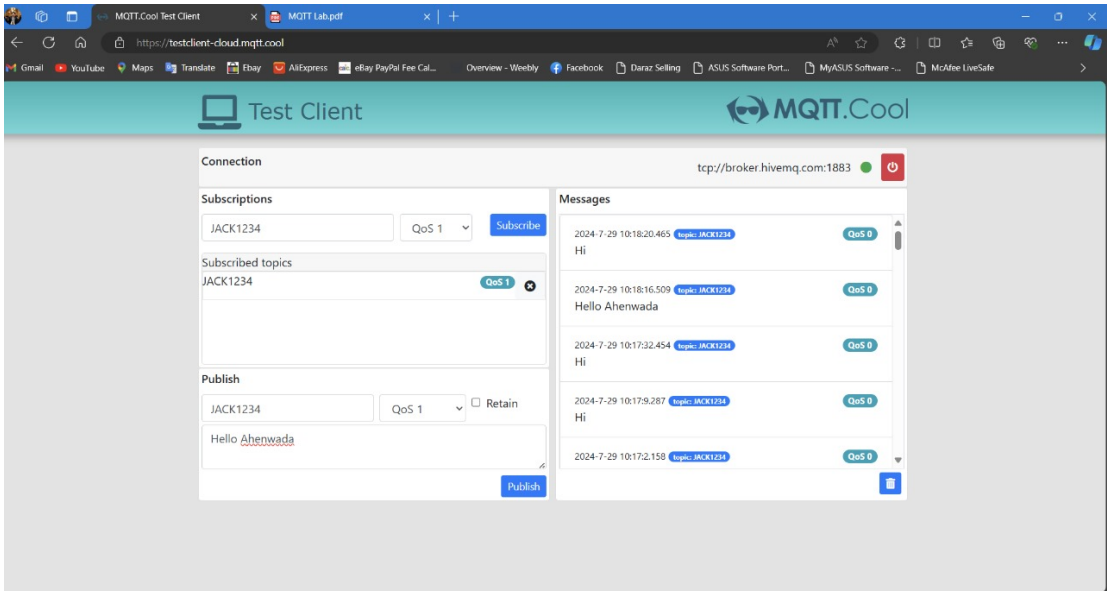
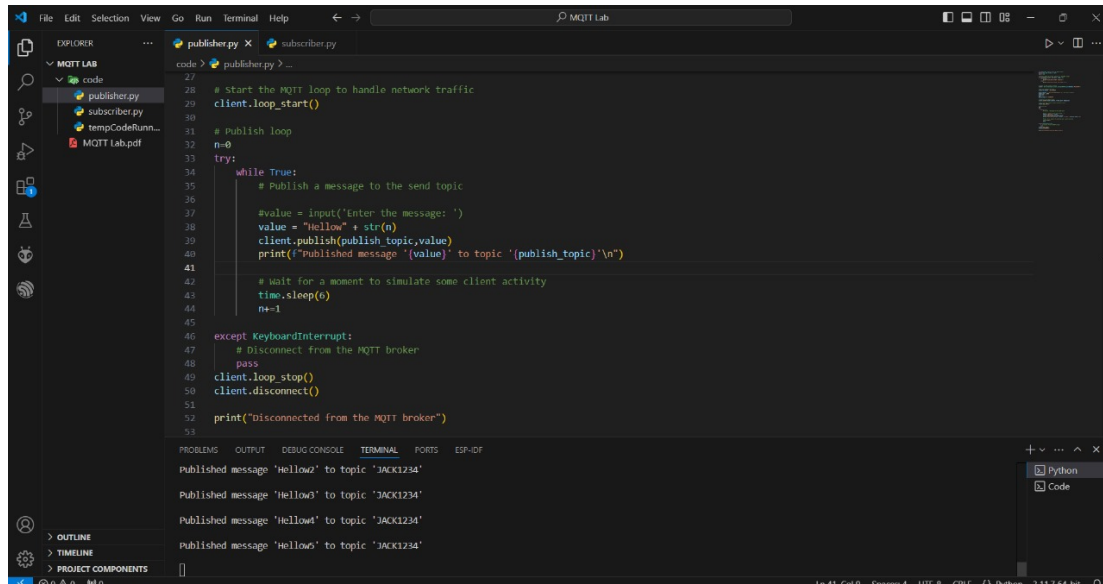


Figure 10: MQTT.cool results (QOS - 1)

3 Step 3.B - QOS=2

3.1 Publisher



```
code > publisher.py > ...
27
28 # start the MQTT loop to handle network traffic
29 client.loop_start()
30
31 # publish loop
32 n=0
33 try:
34     while True:
35         # Publish a message to the send topic
36
37         #value = input('Enter the message: ')
38         value = "Hello" + str(n)
39         client.publish(publish_topic,value)
40         print("Published message '"value"' to topic '"publish_topic'"")
41
42         # wait for a moment to simulate some client activity
43         time.sleep(5)
44         n+=1
45
46 except KeyboardInterrupt:
47     # Disconnect from the MQTT broker
48     pass
49     client.loop_stop()
50     client.disconnect()
51
52 print("Disconnected from the MQTT broker")
53
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS ESP-IDF

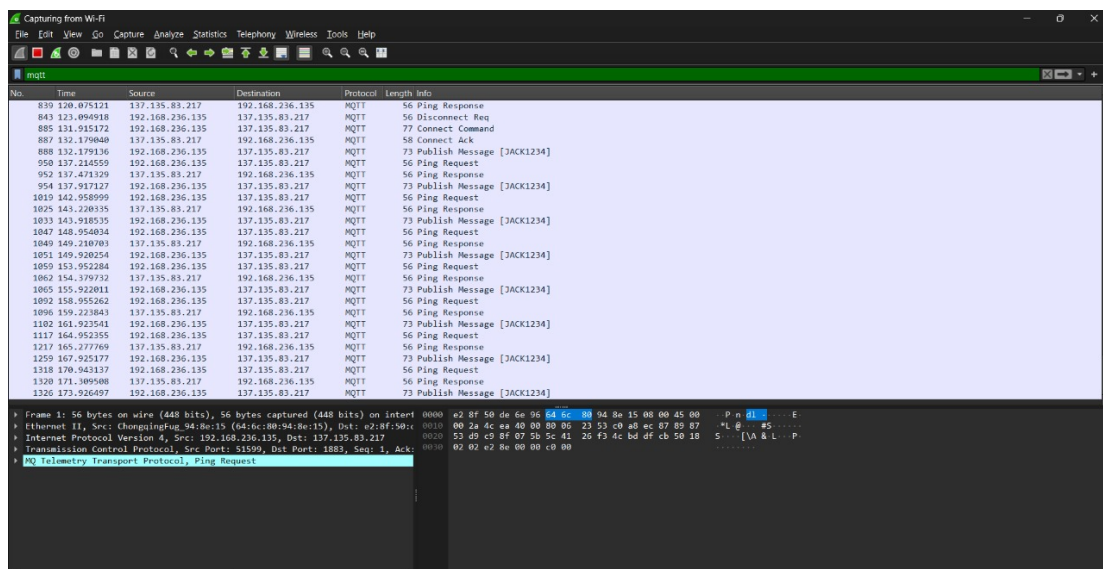
Published message 'Hello' to topic 'JACK1234'

Published message 'Hello' to topic 'JACK1234'

Published message 'Hello' to topic 'JACK1234'

Published message 'Hello' to topic 'JACK1234'

Figure 11: Python code for the publisher (QOS - 2)



No.	Time	Source	Destination	Protocol	Length	Info
839	120.875121	137.135.83.217	192.168.236.135	MQTT	56	Ping Response
843	123.094918	192.168.236.135	137.135.83.217	MQTT	56	Disconnect Req
885	131.915172	192.168.236.135	137.135.83.217	MQTT	77	Connect Command
887	132.179840	137.135.83.217	192.168.236.135	MQTT	58	Connect Ack
888	132.179136	192.168.236.135	137.135.83.217	MQTT	73	Publish Message [JACK1234]
950	137.214559	192.168.236.135	137.135.83.217	MQTT	56	Ping Request
952	137.471129	137.135.83.217	192.168.236.135	MQTT	56	Ping Response
954	137.917127	192.168.236.135	137.135.83.217	MQTT	73	Publish Message [JACK1234]
1010	142.958999	192.168.236.135	137.135.83.217	MQTT	56	Ping Request
1025	143.220355	137.135.83.217	192.168.236.135	MQTT	56	Ping Response
1033	143.948535	192.168.236.135	137.135.83.217	MQTT	73	Publish Message [JACK1234]
1047	148.954034	192.168.236.135	137.135.83.217	MQTT	56	Ping Request
1049	149.210703	137.135.83.217	192.168.236.135	MQTT	56	Ping Response
1051	149.920254	192.168.236.135	137.135.83.217	MQTT	73	Publish Message [JACK1234]
1050	153.952284	192.168.236.135	137.135.83.217	MQTT	56	Ping Request
1062	154.379732	137.135.83.217	192.168.236.135	MQTT	56	Ping Response
1065	155.922011	192.168.236.135	137.135.83.217	MQTT	73	Publish Message [JACK1234]
1092	158.955262	192.168.236.135	137.135.83.217	MQTT	56	Ping Request
1096	159.223843	137.135.83.217	192.168.236.135	MQTT	56	Ping Response
1102	161.923541	192.168.236.135	137.135.83.217	MQTT	73	Publish Message [JACK1234]
1117	164.952355	192.168.236.135	137.135.83.217	MQTT	56	Ping Request
1237	165.277769	137.135.83.217	192.168.236.135	MQTT	56	Ping Response
1259	167.925177	192.168.236.135	137.135.83.217	MQTT	73	Publish Message [JACK1234]
1318	170.943137	192.168.236.135	137.135.83.217	MQTT	56	Ping Request
1320	171.909588	137.135.83.217	192.168.236.135	MQTT	56	Ping Response
1326	173.926497	192.168.236.135	137.135.83.217	MQTT	73	Publish Message [JACK1234]

Frame 1: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface 0: e2 8f 58 d9 6a 96 84 8e 15 08 08 45 08 ... P n 8 15 ... E

Ethernet II, Src: CharmingHub 94:8c:15 (94:8c:15), Dst: e2:8f:58:00:2a:4c:ea:40:00:00:23:53:c9:a8:ec:07:09:87 ... L 8 #5 ...

Internet Protocol Version 4, Src: 192.168.236.135, Dst: 137.135.83.217 ... 53 d9 c9 0f 07 5b 5c 41 26 f3 4c bd df cb 50 18 ... [VA & L P

Transmission Control Protocol, Src Port: 51599, Dst Port: 1883, Seq: 1, Ack: 6202 ... 02 02 e2 8e 00 00 c0 00

MQ Telemetry Transport Protocol, Ping Request

Figure 12: Wireshark capture for the publisher. (QOS - 2)

3.2 Subscriber

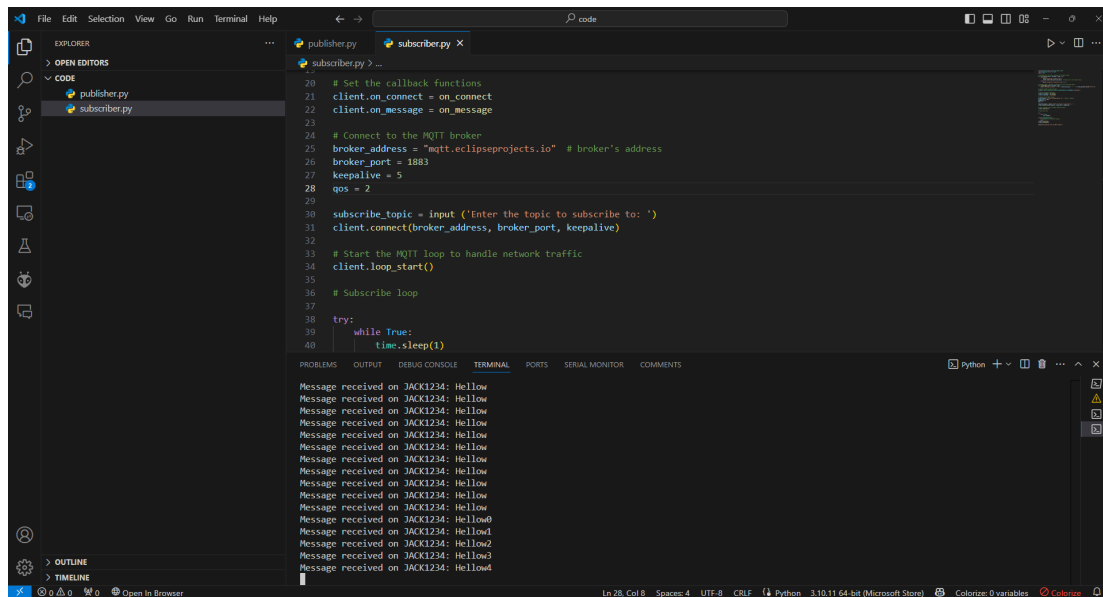


Figure 13: Python code for subscriber (QOS - 2)

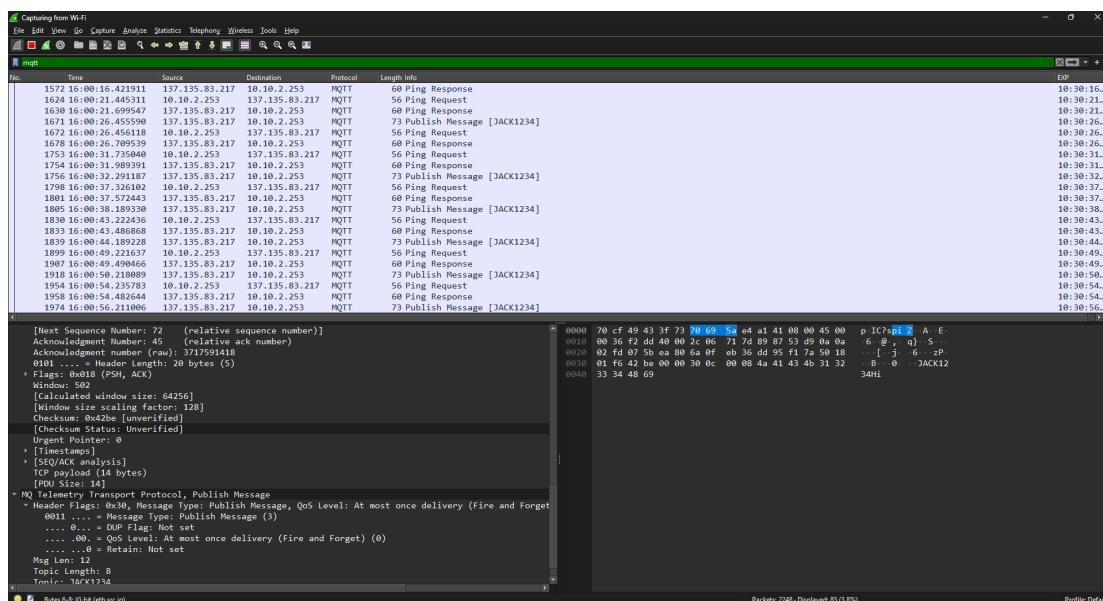


Figure 14: Wireshark capture for the subscriber. (QOS - 2)

3.3 MQTT.Cool Test Client

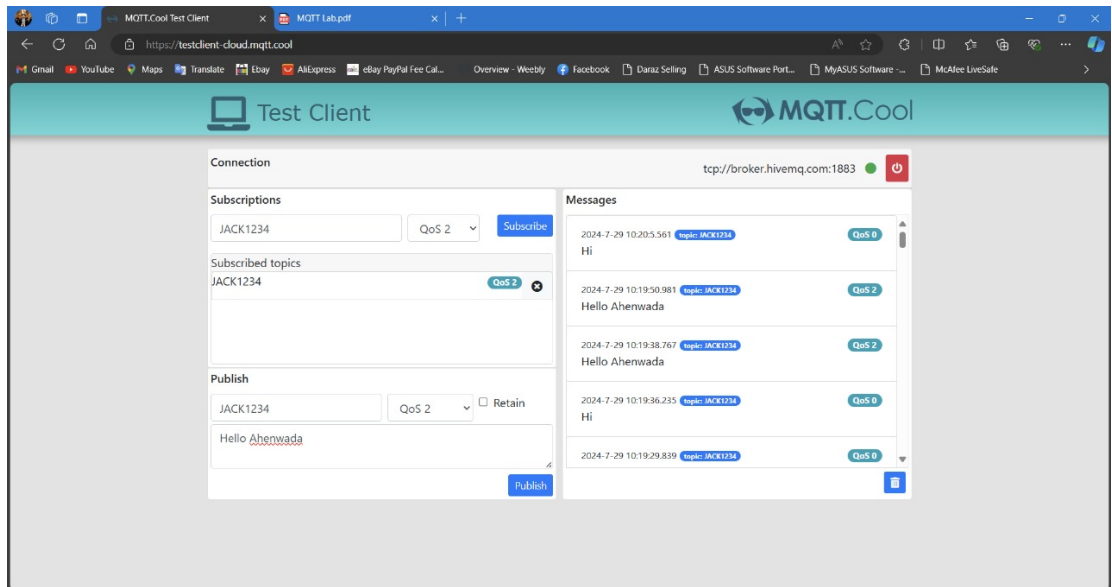


Figure 15: MQTT.cool results (QOS - 2)

4 Observations

4.1 Publisher

During our experiment using the `'publisher.py'` script, Wireshark captured the MQTT traffic and provided several key insights. The capture revealed a sequence of MQTT packets between the client (`'192.168.236.135'`) and the broker (`'35.158.81.26'`), including `'Connect'` and `'Connect Ack'` messages, indicating that the client successfully established connections with the broker. The captured `'Publish Message'` packets, with payloads like `"Hellow1"` and `"JACK1234,"` confirmed that the client successfully published messages.

The sequence of `'Connect Command'` followed by `'Connect Ack'` in the Wireshark capture showed that the client consistently established connections with the broker. Initially, each message was sent without requiring acknowledgment from the broker, corresponding to a QoS level of 0. However, when the QoS was increased to 1 and later to 2, we observed significant performance improvements. Wireshark confirmed these enhancements through additional acknowledgment packets for each published message, ensuring higher reliability and confirming message delivery.

During the lab, we encountered performance issues with several MQTT brokers due to high traffic. However, the `'mqtt.eclipseprojects.io'` broker consistently provided stable connections and better performance, making it the most reliable option for our tests. These observations demonstrate the effectiveness of the MQTT protocol in handling message publishing and confirm that increasing the QoS level improves reliability, with `'mqtt.eclipseprojects.io'` being the best-performing broker under the lab conditions.

4.2 Subscriber

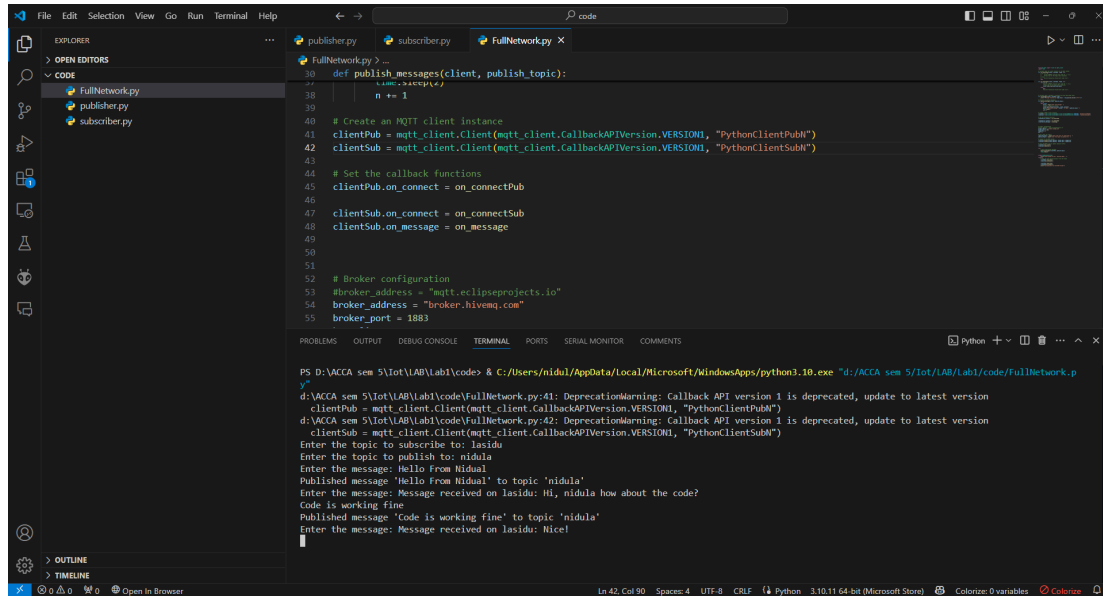
The `subscriber.py` script was used to observe the MQTT traffic for different QoS levels. Wireshark captured the incoming MQTT messages from the broker to the subscriber, providing critical insights into how the QoS levels affect message delivery and reliability.

- **QoS 0:** At QoS level 0, the Wireshark capture revealed that messages were received without any acknowledgment from the subscriber. The lack of acknowledgment confirms that the messages are delivered on a "best effort" basis, where the broker sends the message only once, and there is no guarantee of delivery or duplicate detection. This was evident in the capture, where each "Publish Message" was followed by no additional packets, indicating that the subscriber received the messages without any further interaction.
- **QoS 1:** When the QoS level was increased to 1, Wireshark captured additional packets indicating a more reliable delivery process. Specifically, for every "Publish Message" received, the subscriber sent a "PubAck" acknowledgment back to the broker. This acknowledgment ensures that the message was received and allows the broker to remove the message from its queue. The captured packets show a clear sequence of publish-acknowledge exchanges, confirming the enhanced reliability at QoS 1.
- **QoS 2:** At the highest QoS level, QoS 2, Wireshark captured an even more robust exchange between the broker and subscriber. Each message involved a four-step handshake process, including "Publish," "PubRec" (Received), "PubRel" (Release), and "PubComp" (Complete) packets. This process guarantees that each message is delivered exactly once, with no duplicates. The Wireshark captures illustrate this process with a clear sequence of packets for each message, confirming the most reliable message delivery method in MQTT.

Overall, the Wireshark captures showed how the MQTT protocol's reliability increases with the QoS level, with QoS 2 providing the highest assurance of message delivery.

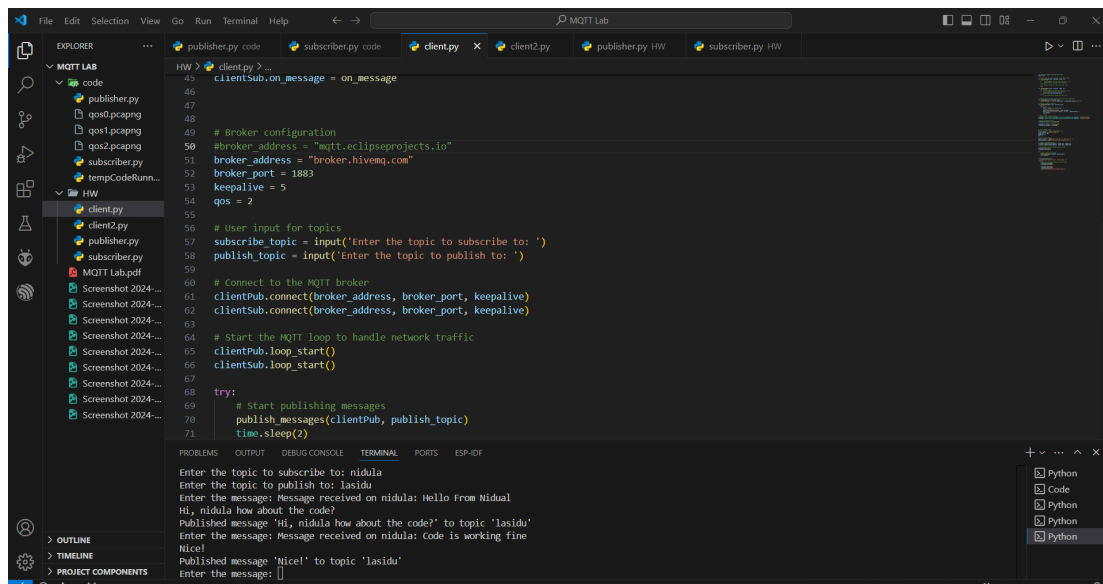
5 Full-blown publish/subscribe client

In this implementation, both the publisher and subscriber are incorporated into a single Python script, effectively creating a client that facilitates communication between two computers. This example demonstrates how the 'nidula' side and the 'lasidu' side use this client to engage in a chat.



```
FullNetwork.py
30 def publish_messages(client, publish_topic):
31     n = 1
32     while n <= 10:
33         # Create an MQTT client instance
34         clientPub = mqtt_client.Client(mqtt_client.CallbackAPIVersion.VERSION1, "PythonClientPub")
35         clientSub = mqtt_client.Client(mqtt_client.CallbackAPIVersion.VERSION1, "PythonClientSub")
36
37         # Set the callback functions
38         clientPub.on_connect = on_connectPub
39
40         clientSub.on_connect = on_connectSub
41         clientSub.on_message = on_message
42
43         # Broker configuration
44         broker_address = "mqtt.eclipseprojects.io"
45         broker_address = "broker.hivemq.com"
46         broker_port = 1883
47
48     PS D:\VACCA sem 5\IoT\LAB\code & C:\Users\nidul\AppData\Local\Microsoft\WindowsApps\python3.10.exe "d:\VACCA sem 5\IoT\LAB\code\FullNetwork.py"
49 d:\VACCA sem 5\IoT\LAB\code\FullNetwork.py:41: DeprecationWarning: Callback API version 1 is deprecated, update to latest version
50 clientPub = mqtt_client.Client(mqtt_client.CallbackAPIVersion.VERSION1, "PythonClientPub")
51 d:\VACCA sem 5\IoT\LAB\code\FullNetwork.py:42: DeprecationWarning: Callback API version 1 is deprecated, update to latest version
52 clientSub = mqtt_client.Client(mqtt_client.CallbackAPIVersion.VERSION1, "PythonClientSub")
53 Enter the topic to subscribe to: lasidu
54 Enter the topic to publish to: nidula
55 Enter the message: Hello from Nidul!
56 Published message 'Hello from Nidul!' to topic 'nidula'
57 Enter the message: Message received on lasidu: Hi, nidula how about the code?
58 Code is working fine
59 Published message 'Code is working fine' to topic 'nidula'
60 Enter the message: Message received on lasidu: Nice!
```

Figure 16: Python code and terminal for nidula side



```
client.py
45 clientSub.on_message = on_message
46
47 # Broker configuration
48 #broker_address = "mqtt.eclipseprojects.io"
49 broker_address = "broker.hivemq.com"
50 broker_port = 1883
51 keepalive = 5
52 qos = 2
53
54 # User input for topics
55 subscribe_topic = input('Enter the topic to subscribe to: ')
56 publish_topic = input('Enter the topic to publish to: ')
57
58 # Connect to the MQTT broker
59 clientPub.connect(broker_address, broker_port, keepalive)
60 clientSub.connect(broker_address, broker_port, keepalive)
61
62 # Start the MQTT loop to handle network traffic
63 clientPub.loop_start()
64 clientSub.loop_start()
65
66 try:
67     # Start publishing messages
68     publish_messages(clientPub, publish_topic)
69     time.sleep(2)
70
71 Enter the topic to subscribe to: nidula
72 Enter the topic to publish to: lasidu
73 Enter the message: Message received on nidula: Hello from Nidul!
74 Hi, nidula how about the code?
75 Published message 'Hi, nidula how about the code?' to topic 'lasidu'
76 Enter the message: Message received on nidula: code is working fine
77 Nice!
78 Published message 'Nice!' to topic 'lasidu'
79 Enter the message: []
```

Figure 17: Python code and terminal for lasidu side