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 moduleEN3251 - Internet of Things, Department of Electronic and Telecommunication Engineering, University of Moratuwa.

1 Step 2 - QOS=0

1.1 Publisher

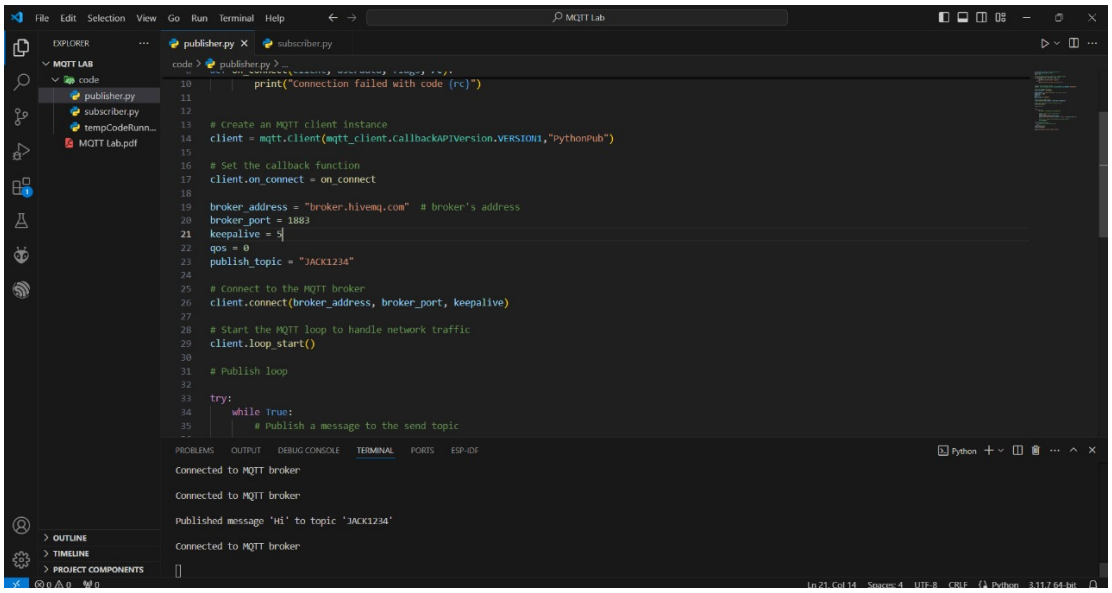


Figure 1: Python code for publisher (QOS - 0)

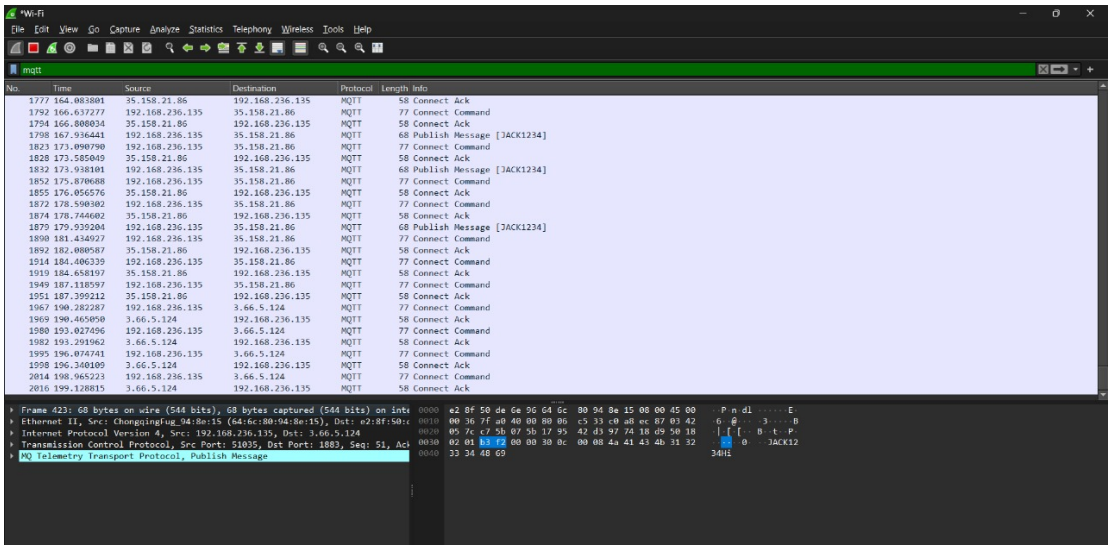


Figure 2: Wireshark capture for the publisher. (QOS - 0)

1.2 Subscriber

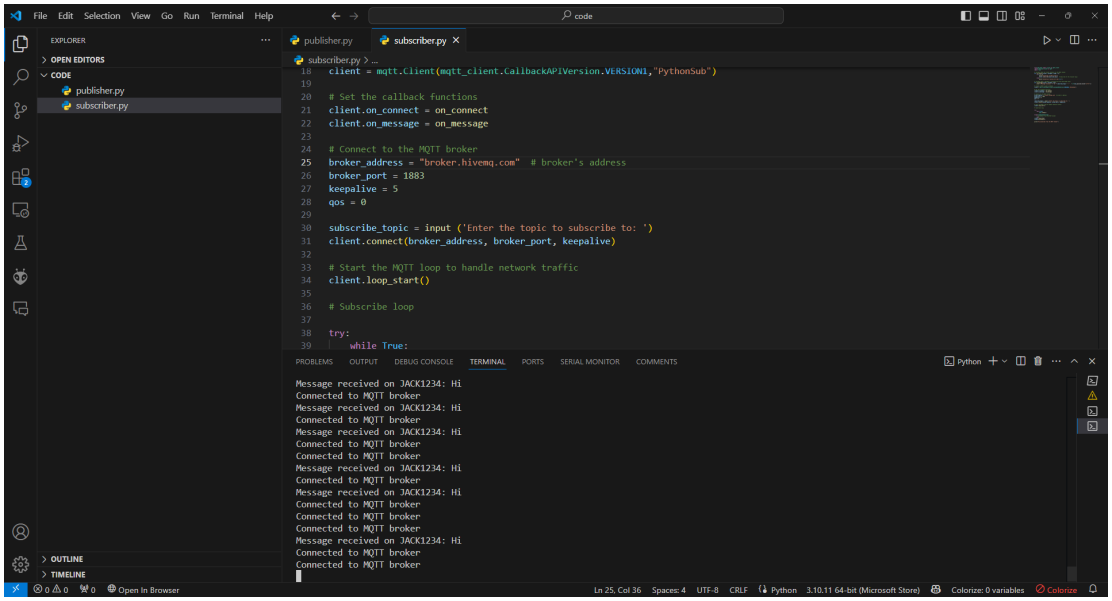


Figure 3: Python code for subscriber (QOS - 0)

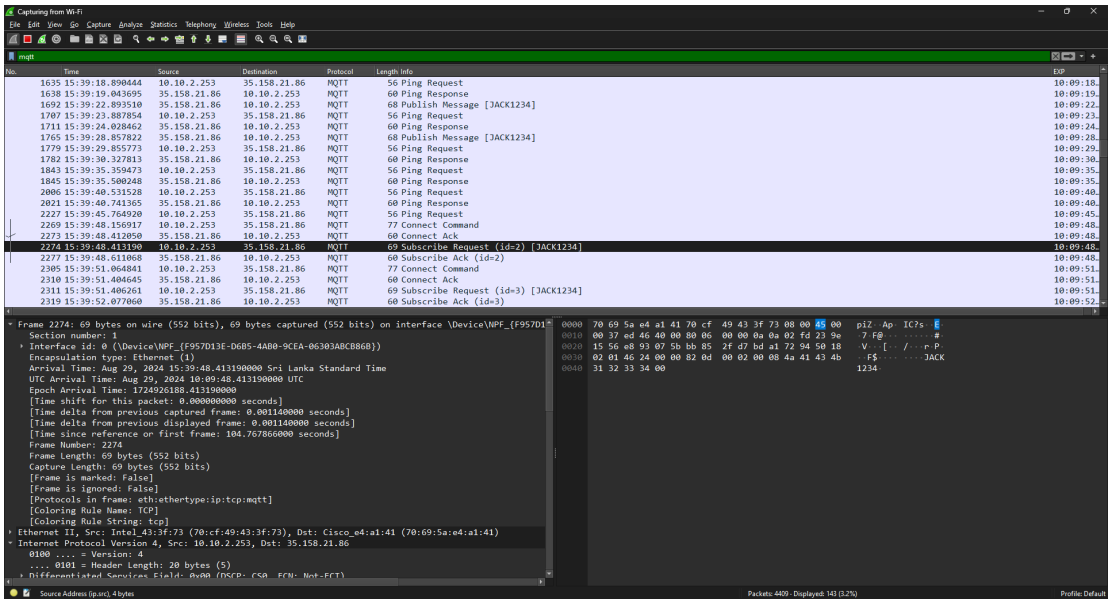


Figure 4: Wireshark capture for the subscriber. (QOS - 0)

1.3 MQTT.Cool Test Client

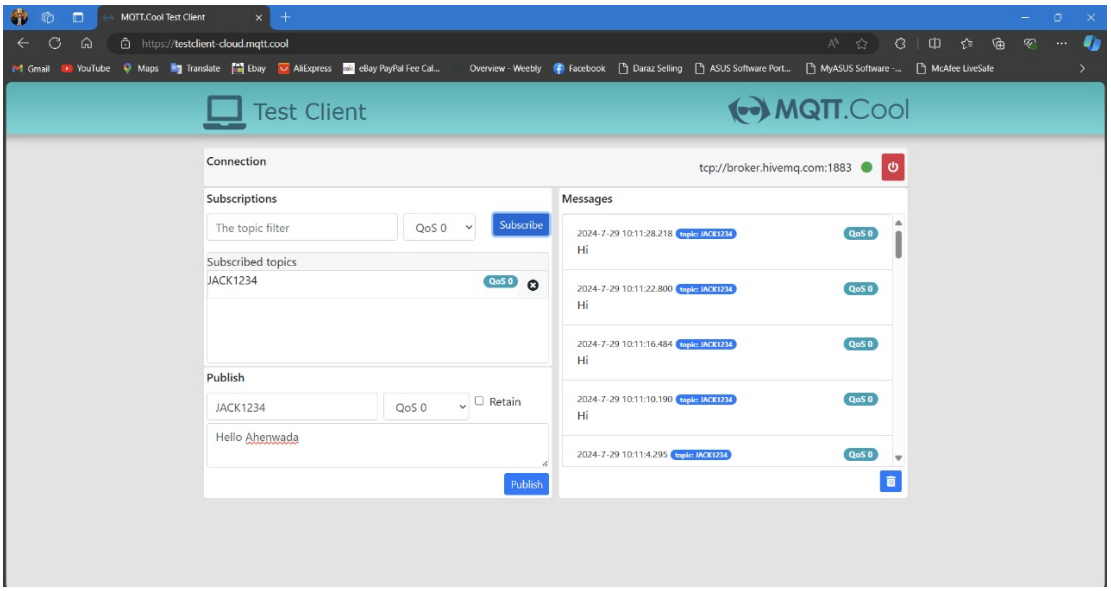


Figure 5: MQTT.cool results (QOS - 0)

2 Step 3.A - QOS=1

2.1 Publisher

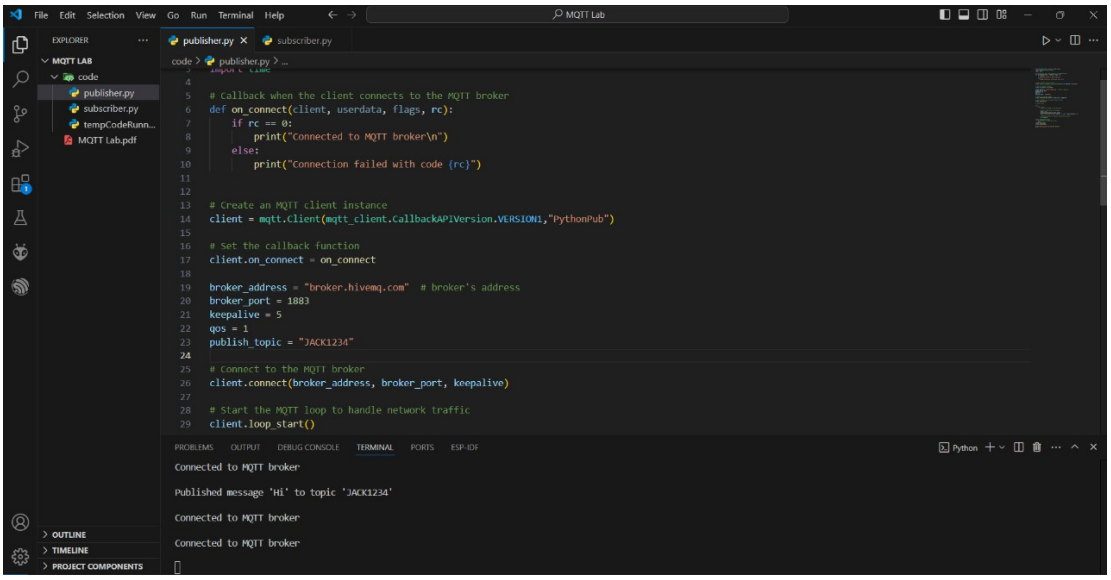


Figure 6: Python code for publisher (QOS - 1)

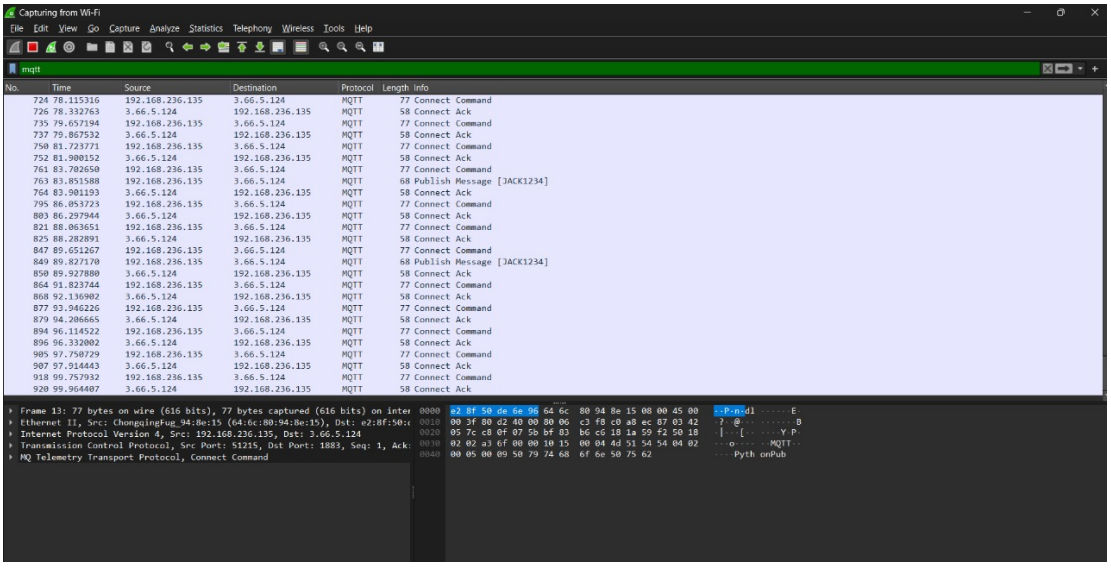


Figure 7: Wireshark capture for the publisher. (QOS - 1)

2.2 Subscriber

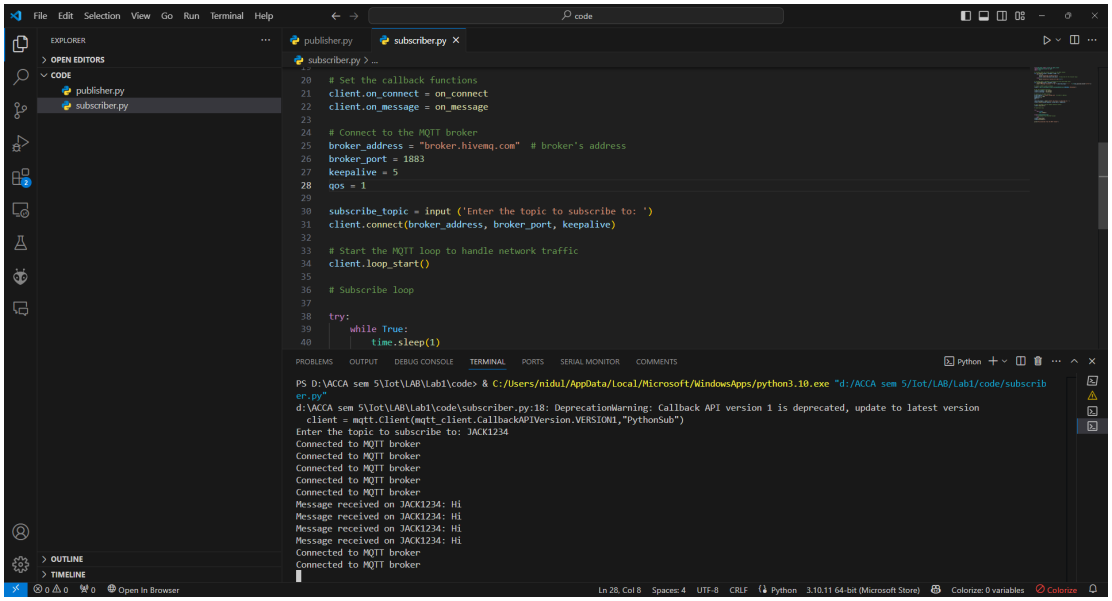


Figure 8: Python code for subscriber (QOS - 1)

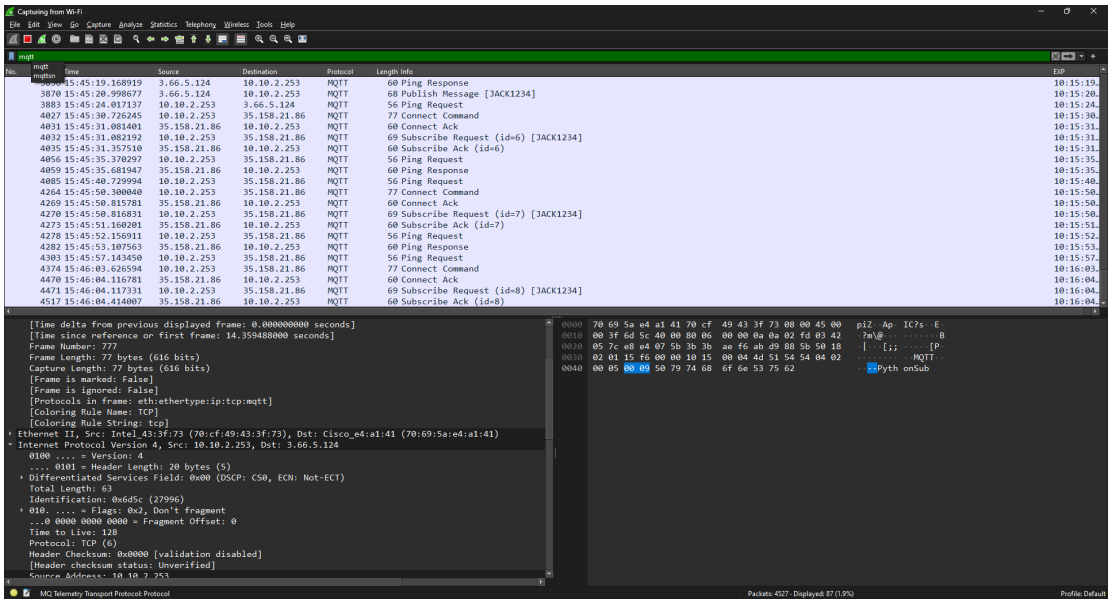


Figure 9: Wireshark capture for the subscriber. (QOS - 1)

2.3 MQTT.Cool Test Client

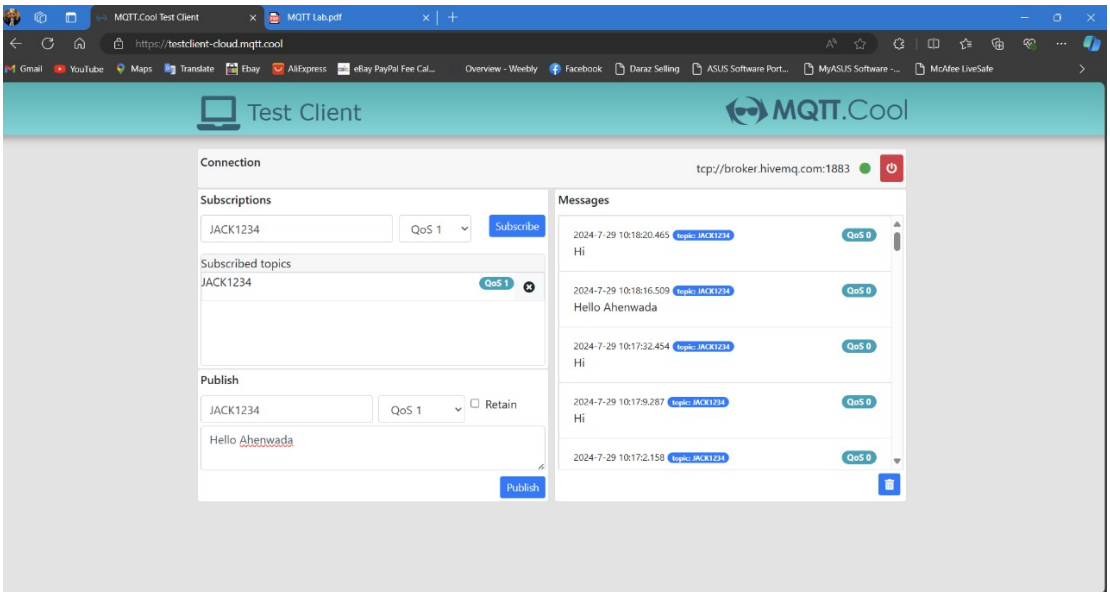


Figure 10: MQTT.cool results (QOS - 1)

3 Step 3.B - QOS=2

3.1 Publisher

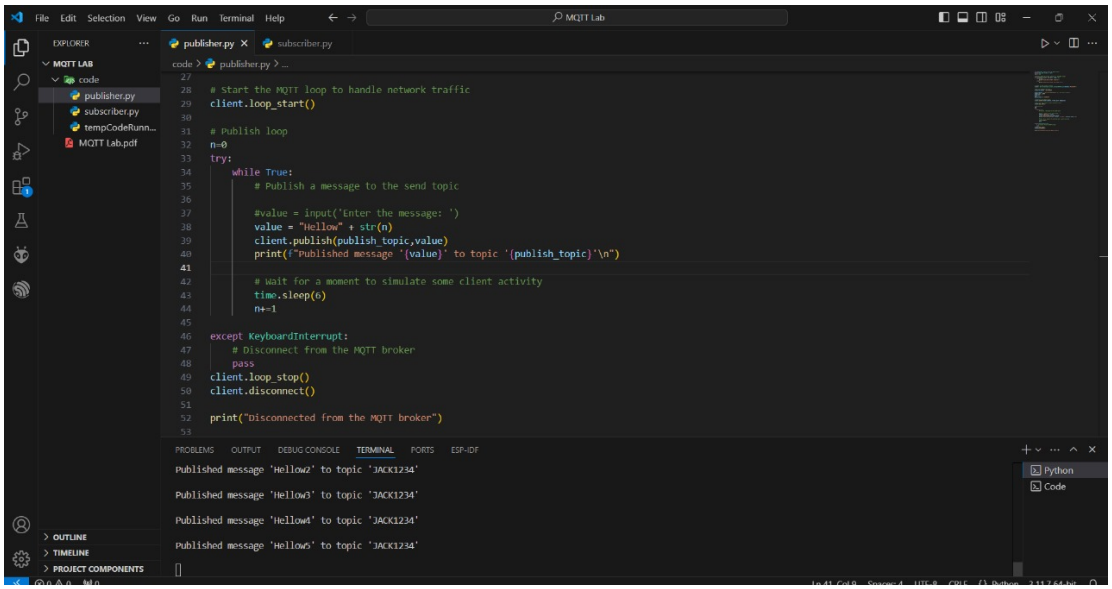


Figure 11: Python code for the publisher (QOS - 2)

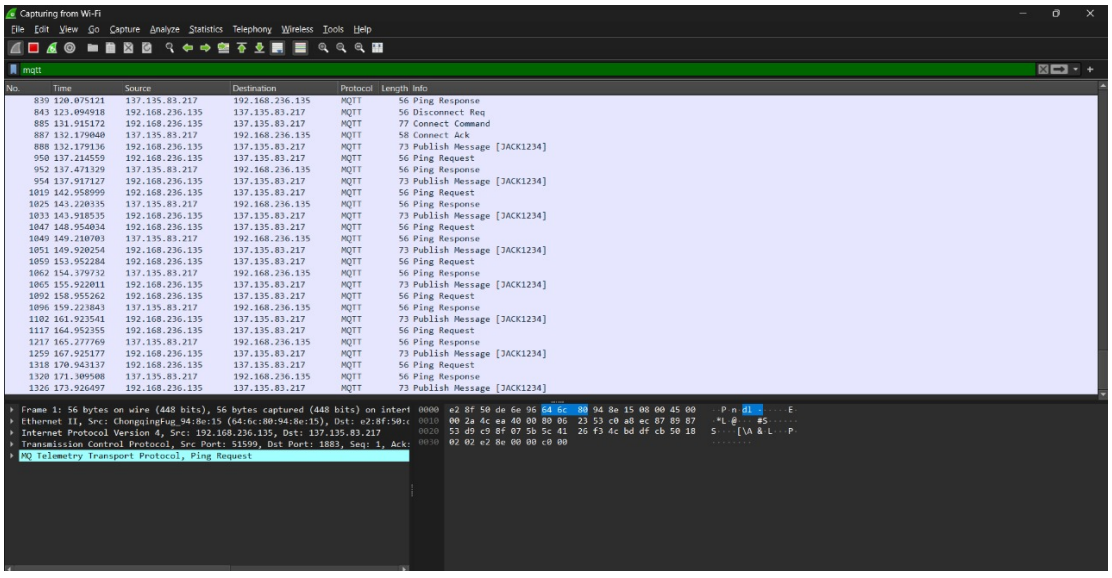
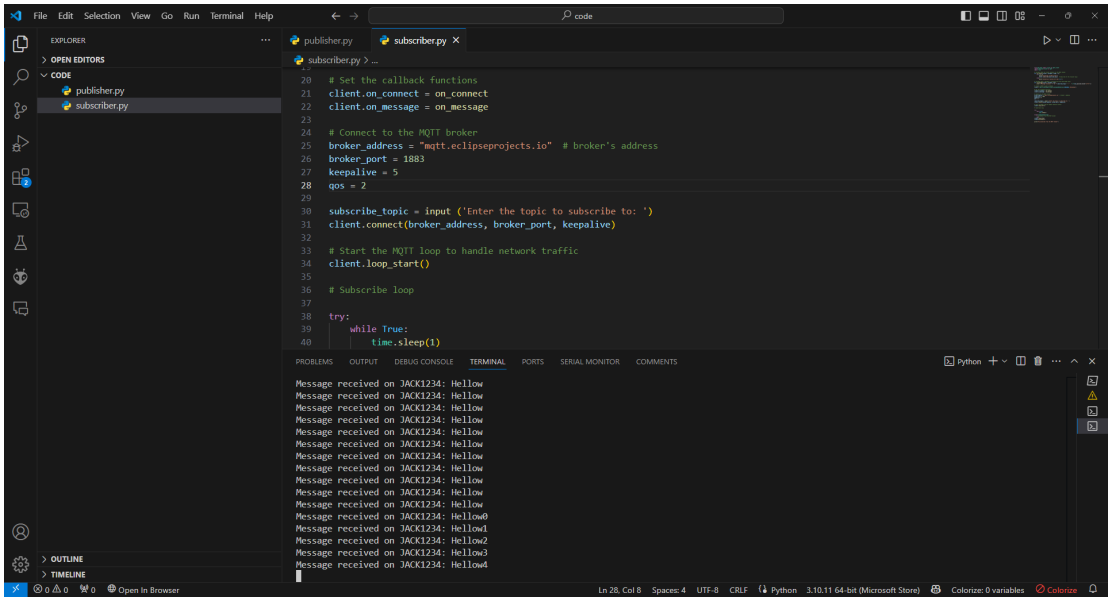


Figure 12: Wireshark capture for the publisher. (QOS - 2)

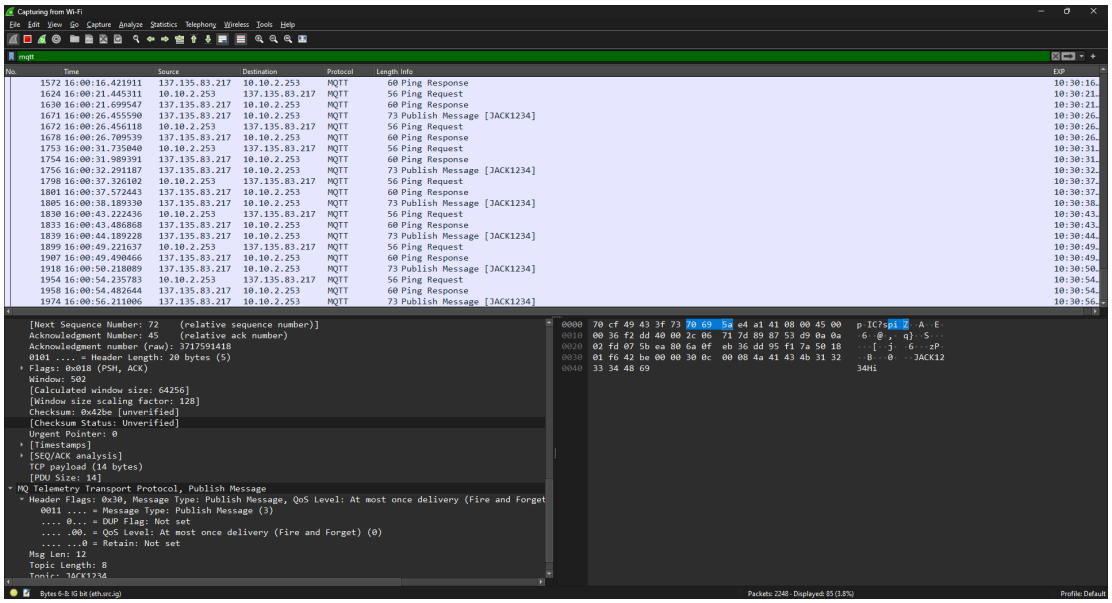
3.2 Subscriber



```python
# Set the callback functions
client.on_connect = on_connect
client.on_message = on_message

# Connect to the MQTT broker
broker_address = "mqtt.eclipseprojects.io"  # broker's address
broker_port = 1883
keepalive = 5
qos = 2

subscribe_topic = input ('Enter the topic to subscribe to: ')
client.connect(broker_address, broker_port, keepalive)

# Start the MQTT loop to handle network traffic
client.loop_start()

# Subscribe loop

try:
    while True:
        time.sleep(1)
```

```
Message received on JACK1234: Hellow
Message received on JACK1234: Hellow
Message received on JACK1234: Hellow
Message received on JACK1234: Hellow
Message received on JACK1234: Hellow
Message received on JACK1234: Hellow
Message received on JACK1234: Hellow
Message received on JACK1234: Hellow
Message received on JACK1234: Hellow
Message received on JACK1234: Hellow
Message received on JACK1234: Hellow
Message received on JACK1234: Hellow0
Message received on JACK1234: Hellow1
Message received on JACK1234: Hellow2
Message received on JACK1234: Hellow3
Message received on JACK1234: Hellow4
```

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



```
No.   Time                Source          Destination     Protocol  Length Info
1572  16:00:16.421911     137.135.83.217  10.10.2.253     MQTT      60 Ping Response
1624  16:00:21.445311     10.10.2.253     137.135.83.217  MQTT      56 Ping Request
1630  16:00:21.699547     137.135.83.217  10.10.2.253     MQTT      60 Ping Response
1671  16:00:26.455590     137.135.83.217  10.10.2.253     MQTT      73 Publish Message [JACK1234]
1672  16:00:26.456118     10.10.2.253     137.135.83.217  MQTT      56 Ping Request
1678  16:00:26.709539     137.135.83.217  10.10.2.253     MQTT      60 Ping Response
1753  16:00:31.735040     10.10.2.253     137.135.83.217  MQTT      56 Ping Request
1754  16:00:31.989391     137.135.83.217  10.10.2.253     MQTT      60 Ping Response
1756  16:00:32.291187     137.135.83.217  10.10.2.253     MQTT      73 Publish Message [JACK1234]
1798  16:00:37.326102     10.10.2.253     137.135.83.217  MQTT      56 Ping Request
1801  16:00:37.572443     137.135.83.217  10.10.2.253     MQTT      60 Ping Response
1805  16:00:38.189330     137.135.83.217  10.10.2.253     MQTT      73 Publish Message [JACK1234]
1830  16:00:43.222436     10.10.2.253     137.135.83.217  MQTT      56 Ping Request
1833  16:00:43.486868     137.135.83.217  10.10.2.253     MQTT      60 Ping Response
1839  16:00:44.189228     137.135.83.217  10.10.2.253     MQTT      73 Publish Message [JACK1234]
1899  16:00:49.221637     10.10.2.253     137.135.83.217  MQTT      56 Ping Request
1907  16:00:49.490466     137.135.83.217  10.10.2.253     MQTT      60 Ping Response
1918  16:00:50.218089     137.135.83.217  10.10.2.253     MQTT      73 Publish Message [JACK1234]
1954  16:00:54.235783     10.10.2.253     137.135.83.217  MQTT      56 Ping Request
1958  16:00:54.482644     137.135.83.217  10.10.2.253     MQTT      60 Ping Response
1974  16:00:56.211006     137.135.83.217  10.10.2.253     MQTT      73 Publish Message [JACK1234]

[Next Sequence Number: 72    (relative sequence number)]
Acknowledgment Number: 45    (relative ack number)
Acknowledgment number (raw): 3717591418
0101 .... = Header Length: 20 bytes (5)
Flags: 0x018 (PSH, ACK)
Window: 502
[Calculated window size: 64256]
[Window size scaling factor: 128]
Checksum: 0x42be [unverified]
[Checksum Status: Unverified]
Urgent Pointer: 0
[Timestamps]
[SEQ/ACK analysis]
TCP payload (14 bytes)
[PDU Size: 14]
MQ Telemetry Transport Protocol, Publish Message
Header Flags: 0x30, Message Type: Publish Message, QoS Level: At most once delivery (Fire and Forget
0011 .... = Message Type: Publish Message (3)
.... 0... = DUP Flag: Not set
.... .00. = QoS Level: At most once delivery (Fire and Forget) (0)
.... ...0 = Retain: Not set
Msg Len: 12
Topic Length: 8
Topic: JACK1234

0000  70 cf 49 43 3f 73 70 69  5a e4 a1 41 08 00 45 00   p·IC?spi Z··A··E·
0010  00 36 f2 dd 40 00 2c 06  71 7d 89 87 53 d9 0a 0a   ·6··@·,· q}··S···
0020  02 fd 07 5b ea 80 6a 0f  eb 36 dd 95 f1 7a 50 18   ···[··j· ·6···zP·
0030  01 f6 42 be 00 00 30 0c  00 08 4a 41 43 4b 31 32   ··B···0· ··JACK12
0040  33 34 48 69                                       34Hi
```

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

3.3 MQTT.Cool Test Client

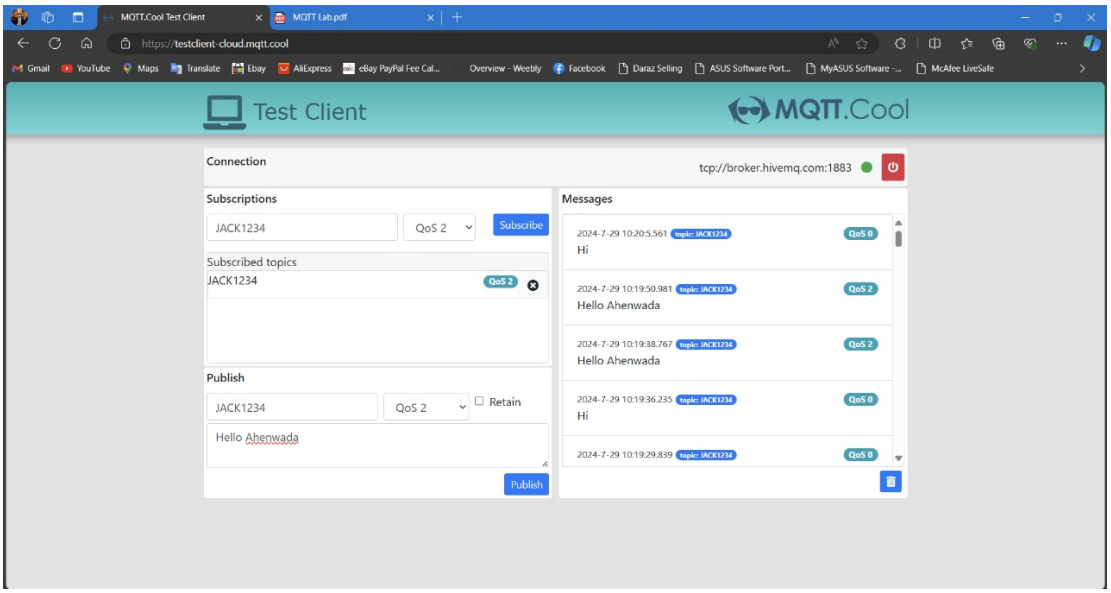


Figure 15: MQTT.cool results (QOS - 2)

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.

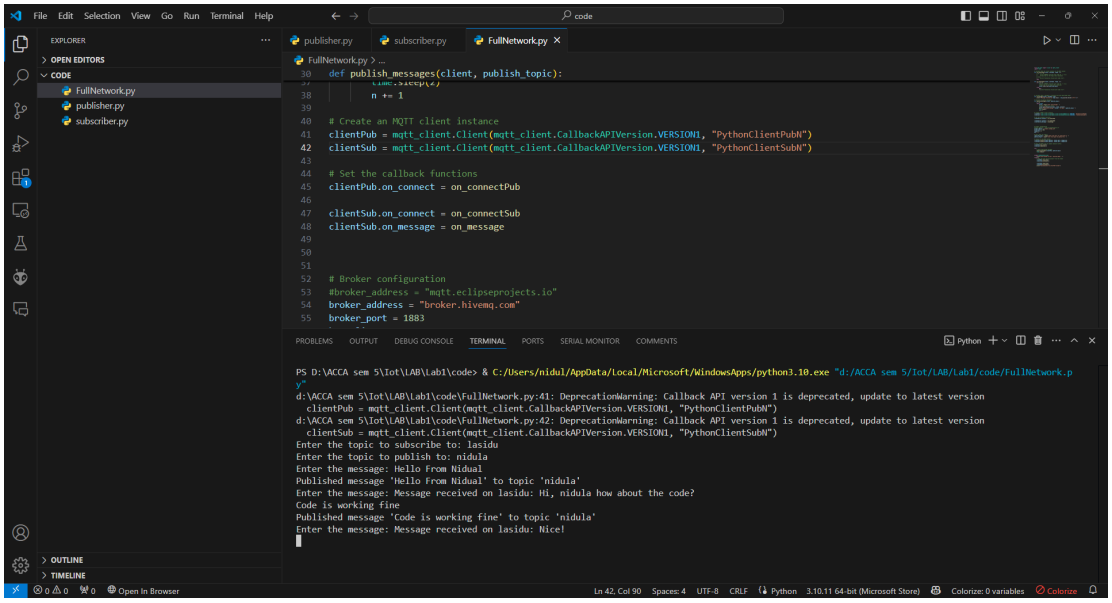


Figure 16: Python code and terminal for nidula side

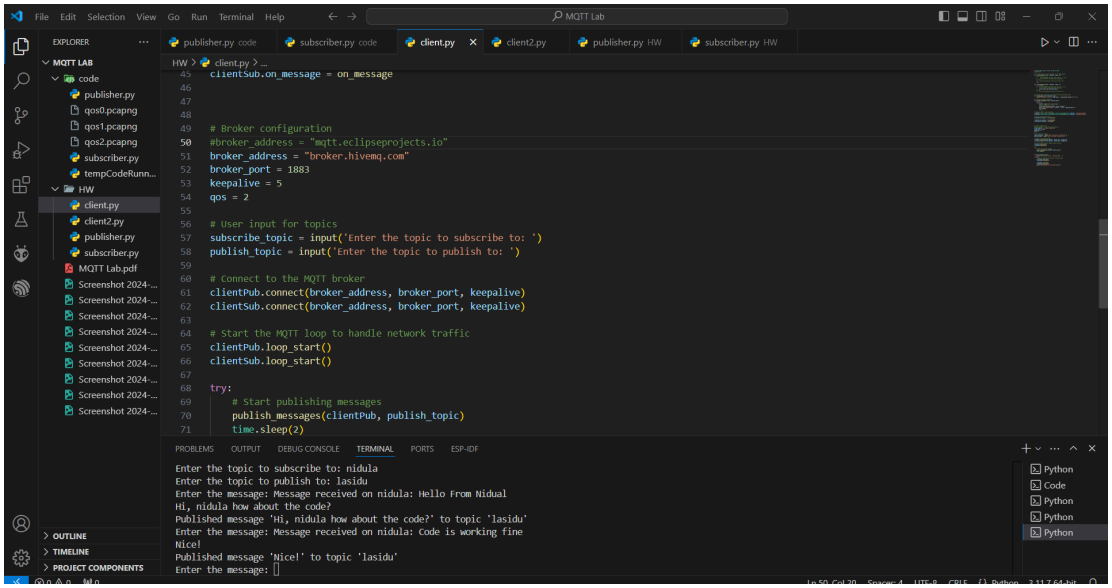


Figure 17: Python code and terminal for lasidu side