Department of Electronic & Telecommunication Engineering
University of Moratuwa, Sri Lanka.

EN3251 - Internet of Things
Laboratory Exercise 1: MQTT Implementation and Testing

Group 7

210387D	Mihiranga N.G.D.
210391J	Morawakgoda M.K.I.G.
210610H	Sirimanna N.T.W.

01st August 2024

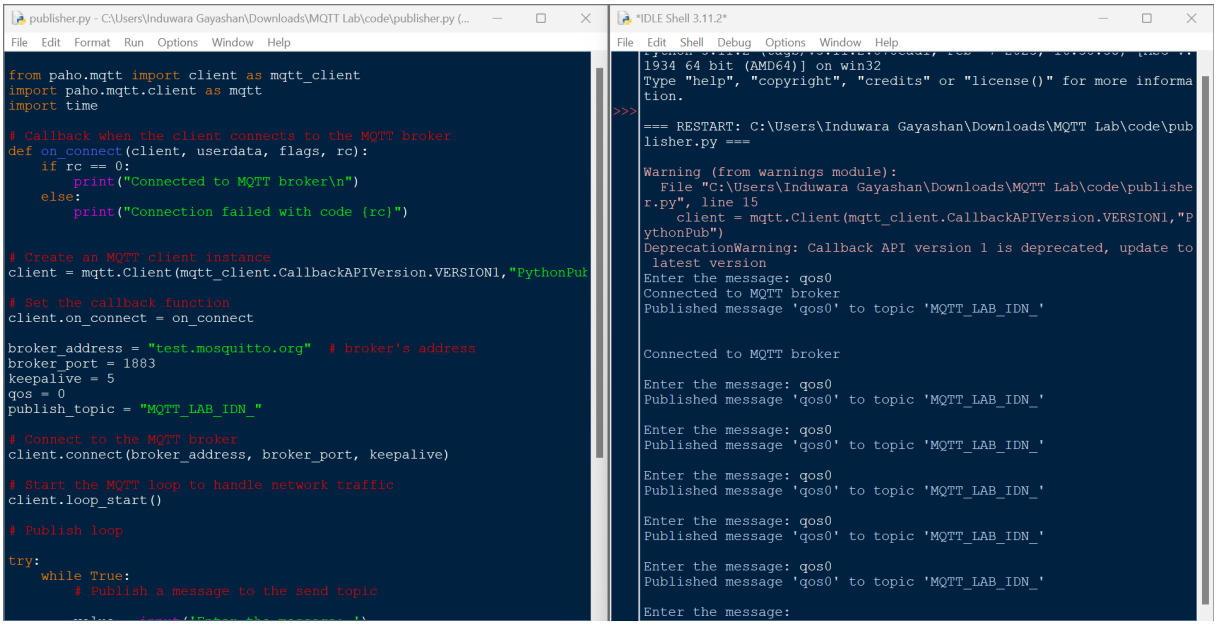
Contents

1 Step 2(QoS level 0)

1.1 Publisher

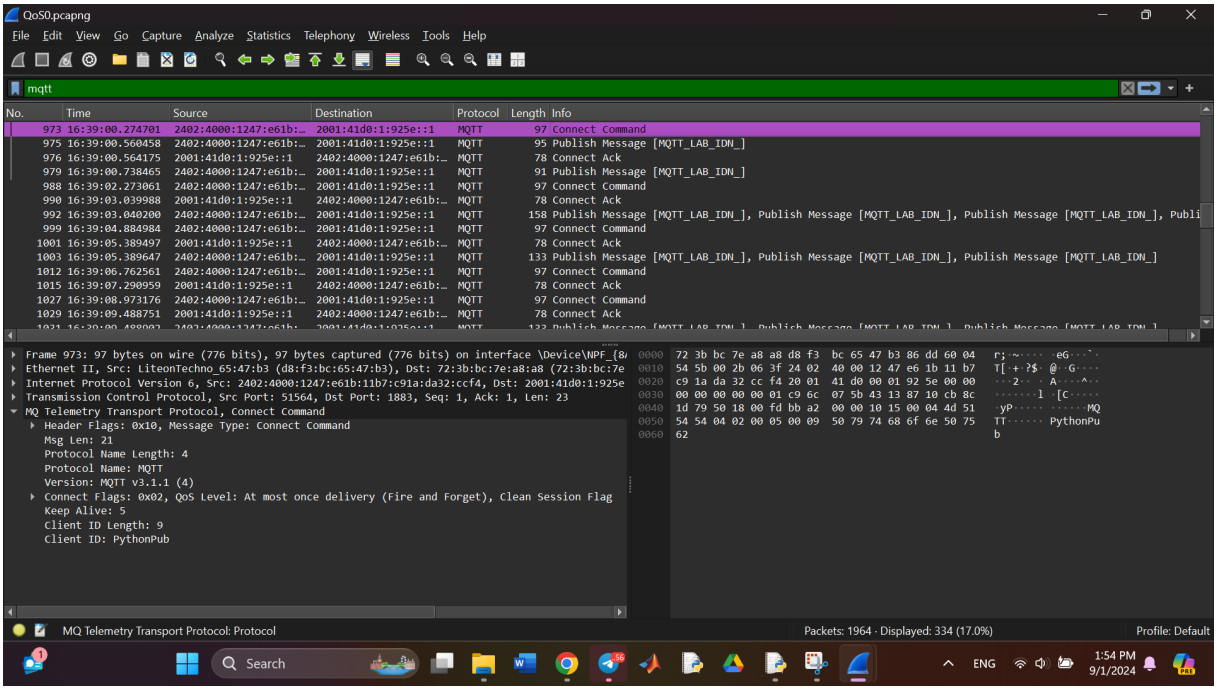
1.1.1 Connecting the Publisher

- **Topic :** MQTT_LAB_IDN_
- **Message :** qos0



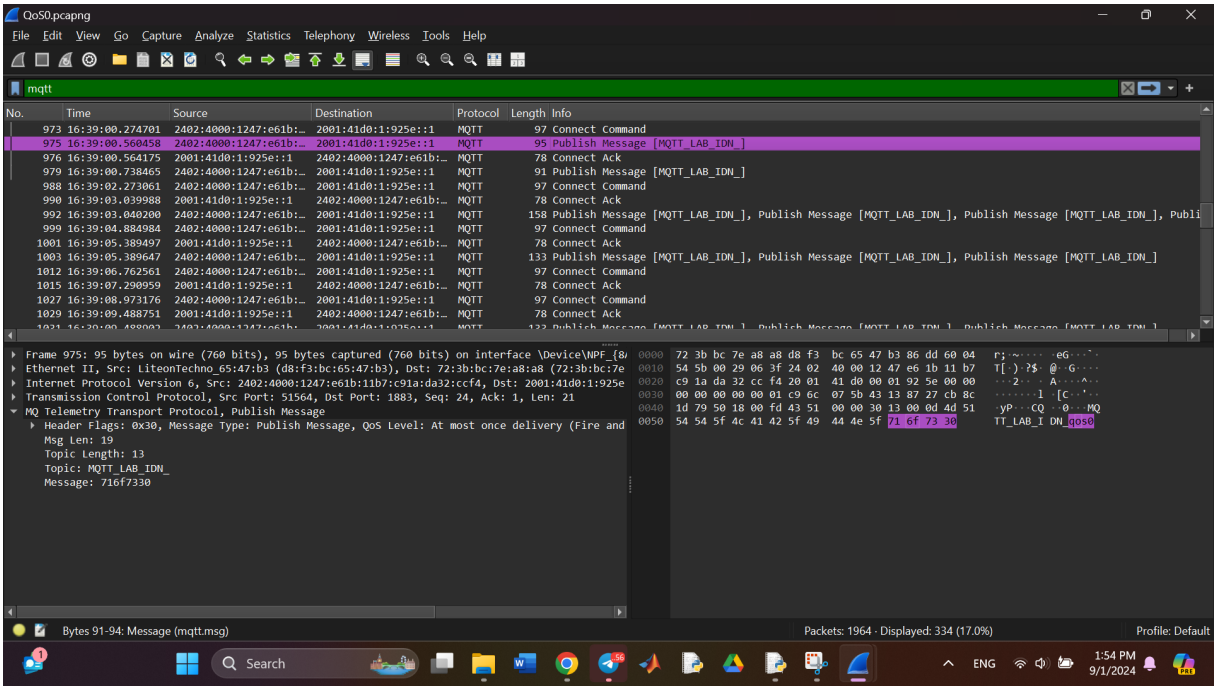
1.1.2 Wireshark Analysis

- **Connect Command:** This is used to establish a connection between the MQTT client and the MQTT broker. A Wireshark capture of the Connect Command at a QoS level of 0 is shown below.

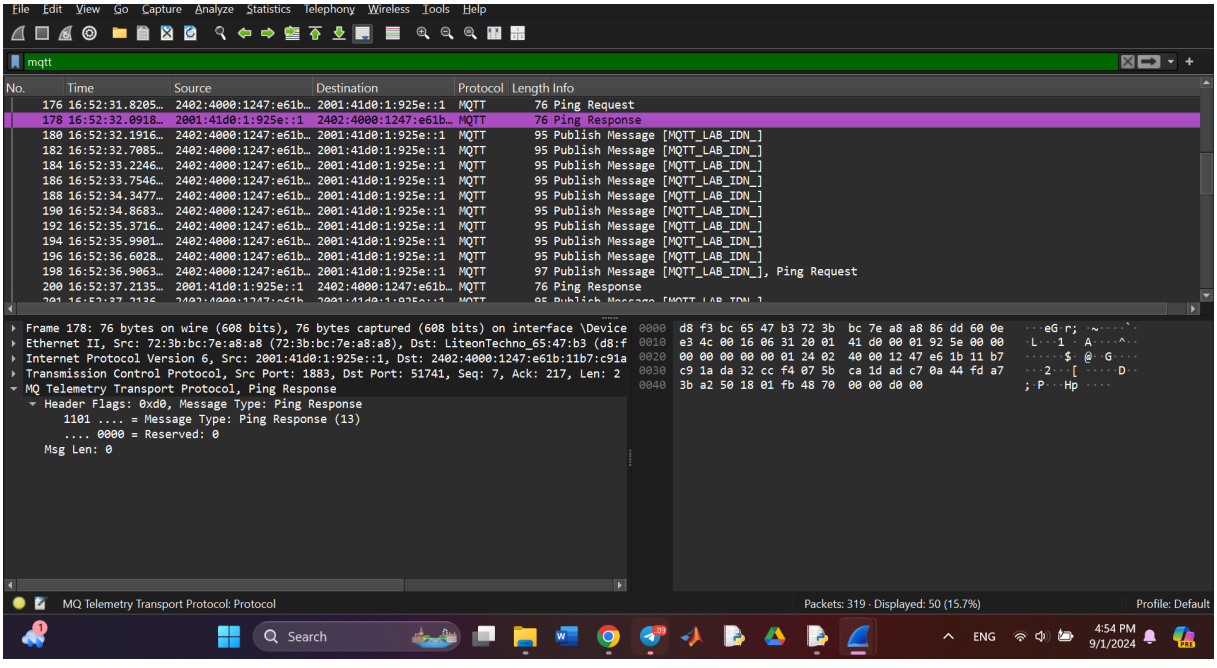


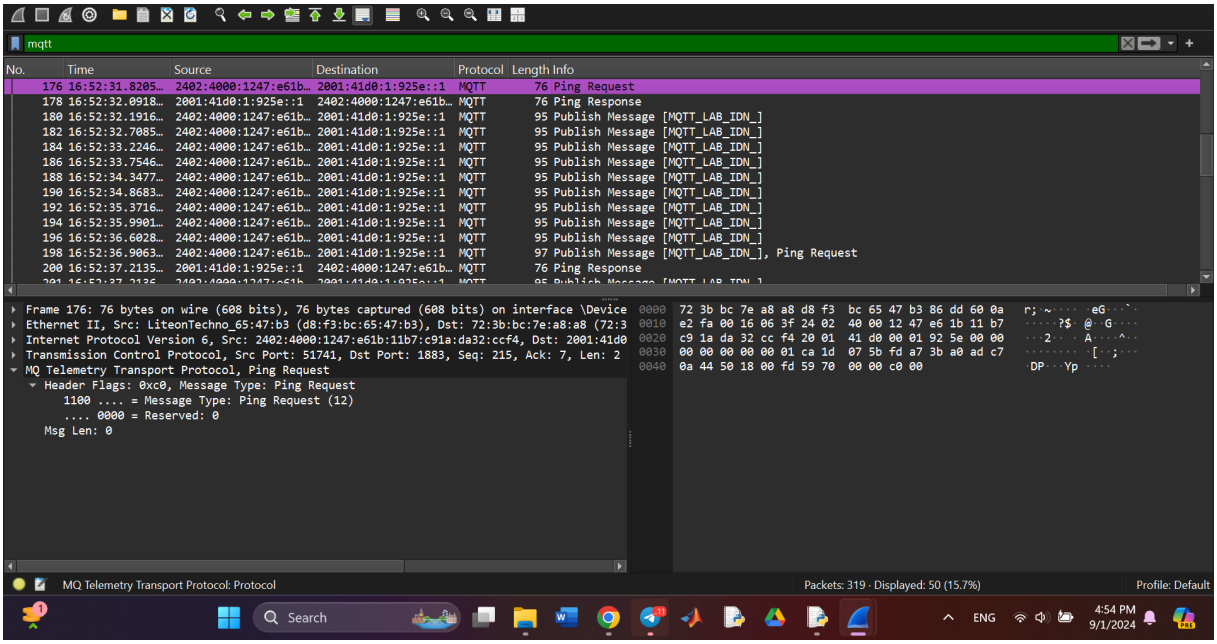
- **Publish Message :** The highlighted packet is an MQTT Publish message which is used to send data to the topic **MQTT_LAB_IDN**.The message has a QoS level of 0, which means it's sent

with at most once delivery, implying no acknowledgment is required, and there's a chance it might not be delivered.The actual message content **qos0** also visible as shown below.



- **Keep-alive message exchange :** The keep-alive interval of 5 seconds is a timer that dictates how often the MQTT client should send a heartbeat to the broker if there is no data communication. This is achieved using **Ping request** and **Ping response** messages as shown below.

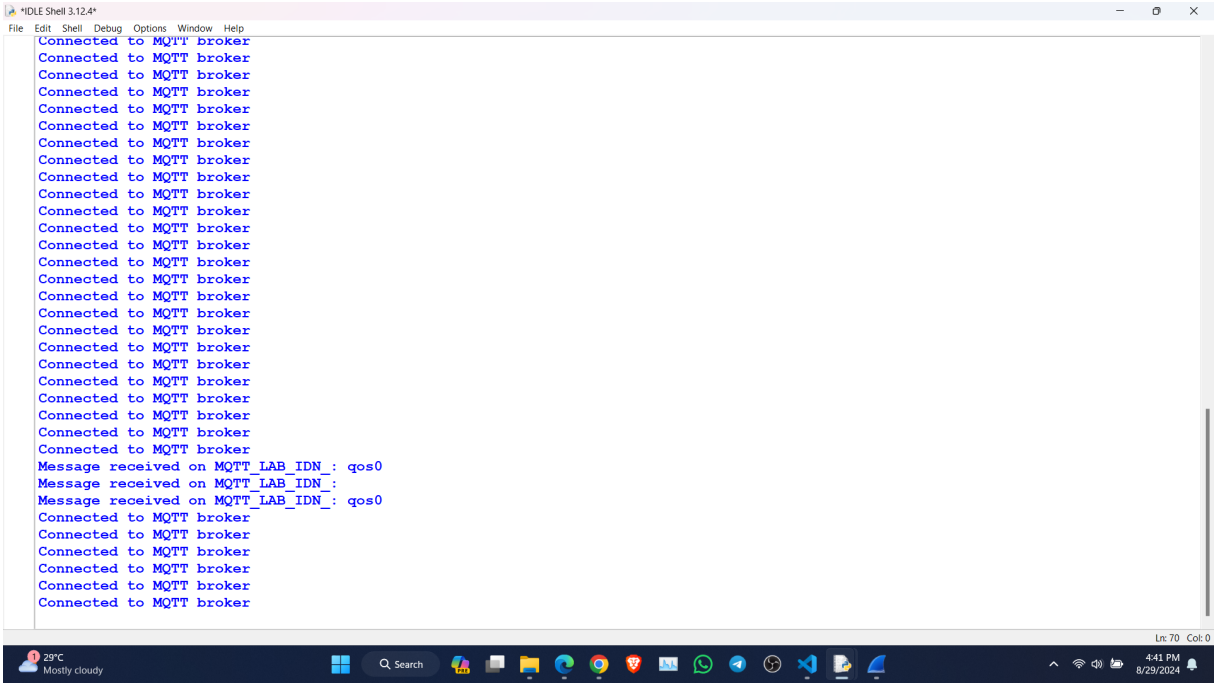




1.2 Subscriber

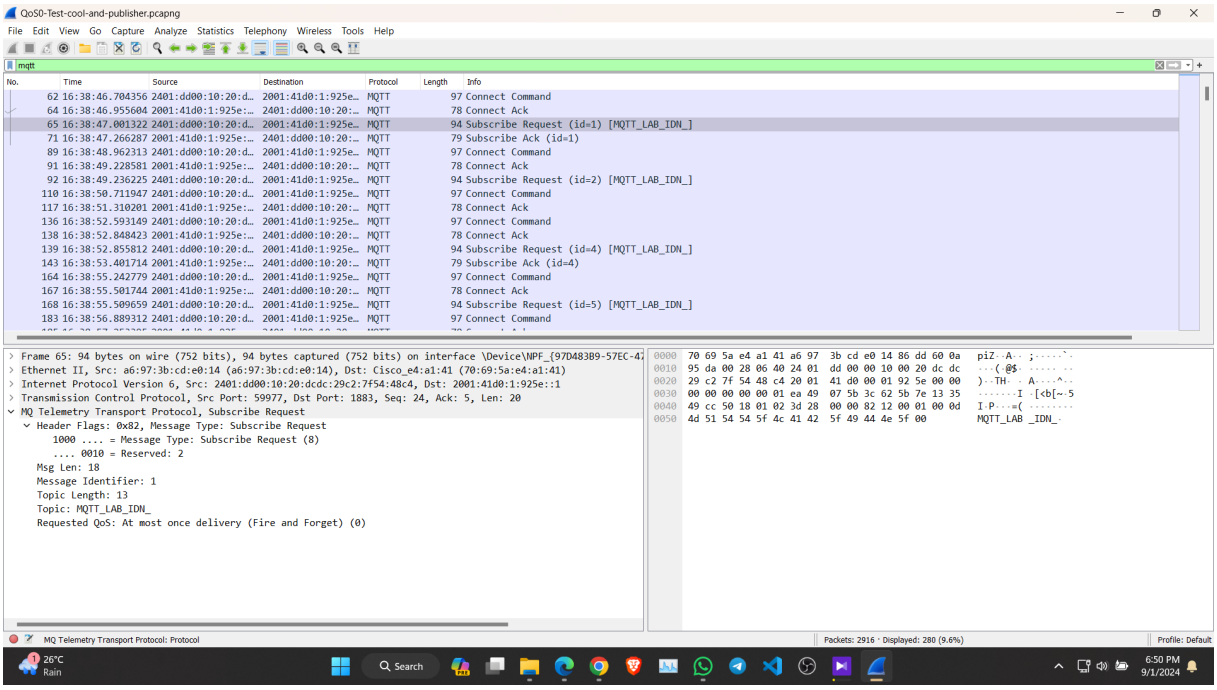
1.2.1 Message Received from Publisher

- **Topic :** MQTT_LAB_IDN_
- **Message :** qos0

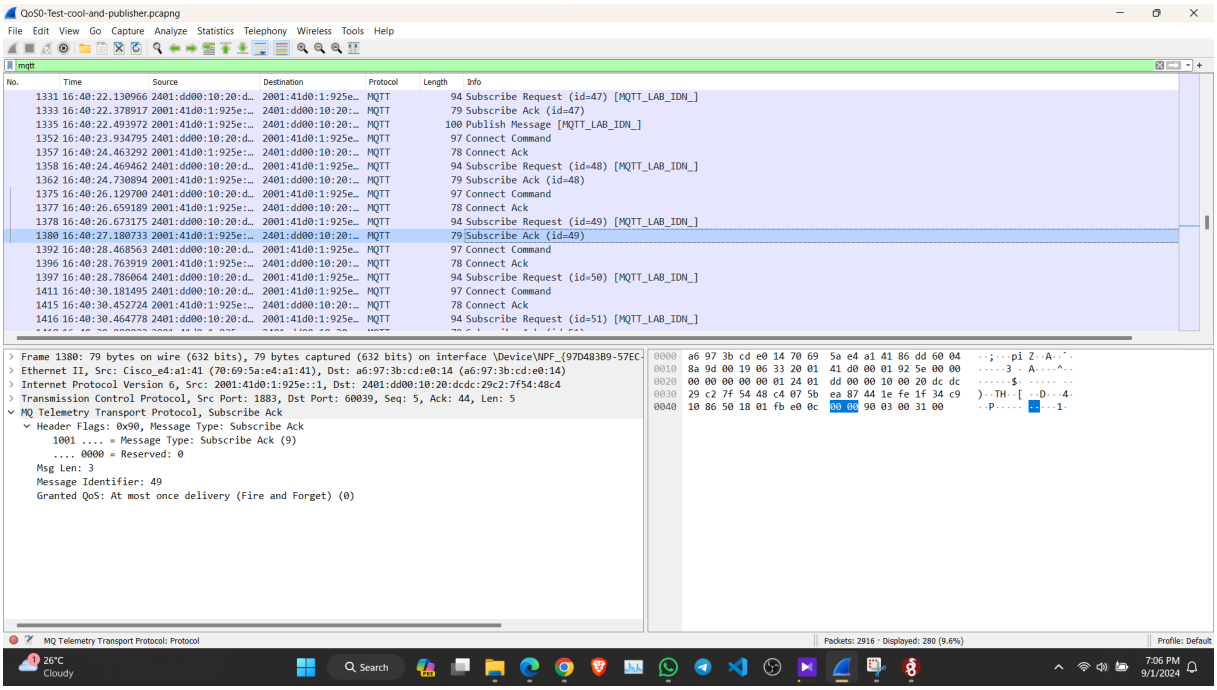


1.2.2 Wireshark Analysis

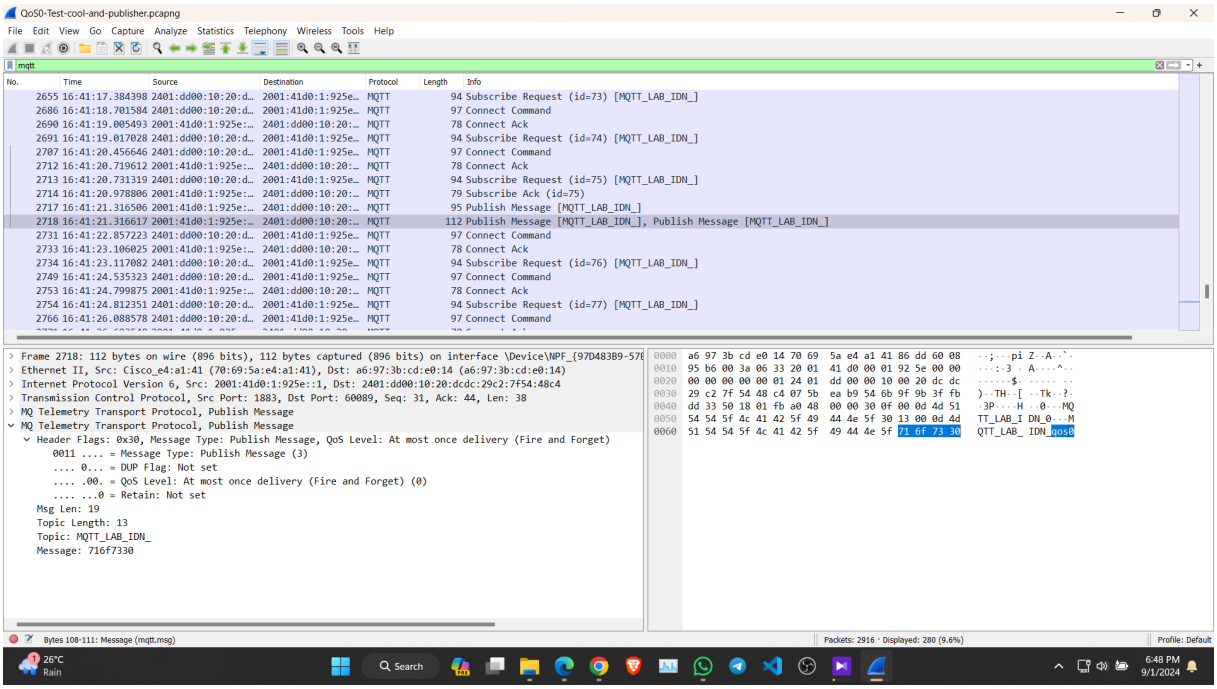
- **Subscribe Request [id=1] [MQTT_LAB_IDN_]:** After connecting, the client sends a `SUBSCRIBE` packet (with ID 1) to the broker. This packet tells the broker that the client wants to subscribe to a topic (in this case, `MQTT_LAB_IDN_`).



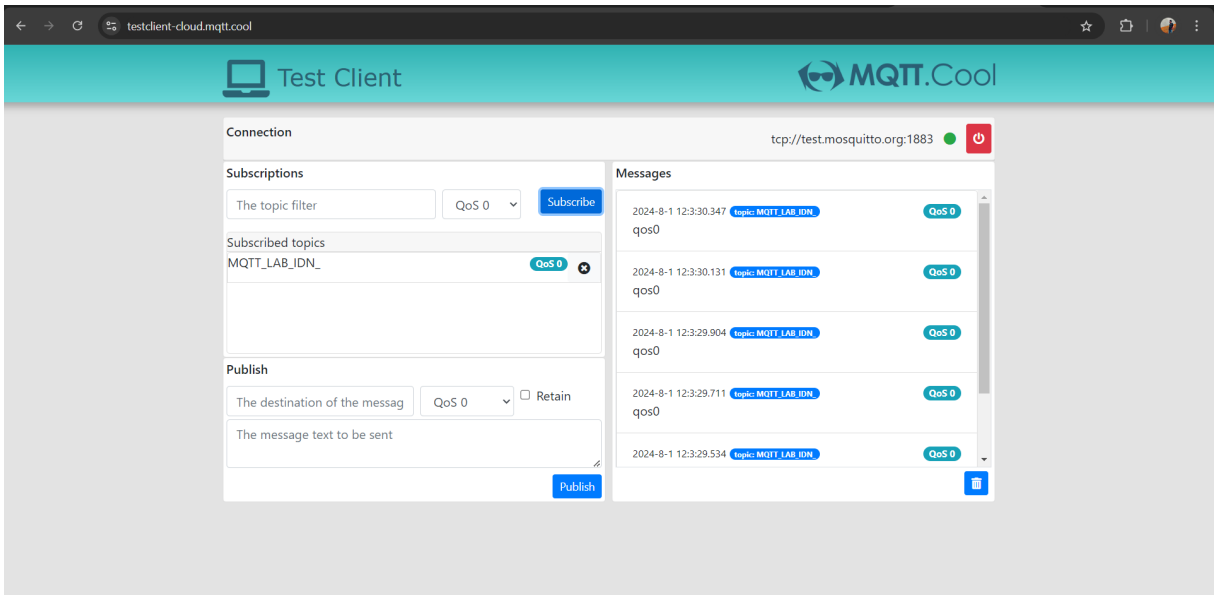
- **Subscribe Ack [id=49]:** The broker acknowledges the subscription request by sending a `SUBACK` packet, confirming that the subscription (with ID 49) was successful.



- **Received message packet:** The MQTT broker delivers this published message to all subscribed clients, including this computer, which has already subscribed to the "MQTT_LAB_IDN_" topic. The message content is 'qos0', as decoded from the hexadecimal payload 71 6f 73 30.



1.3 MQTT.Cool test client

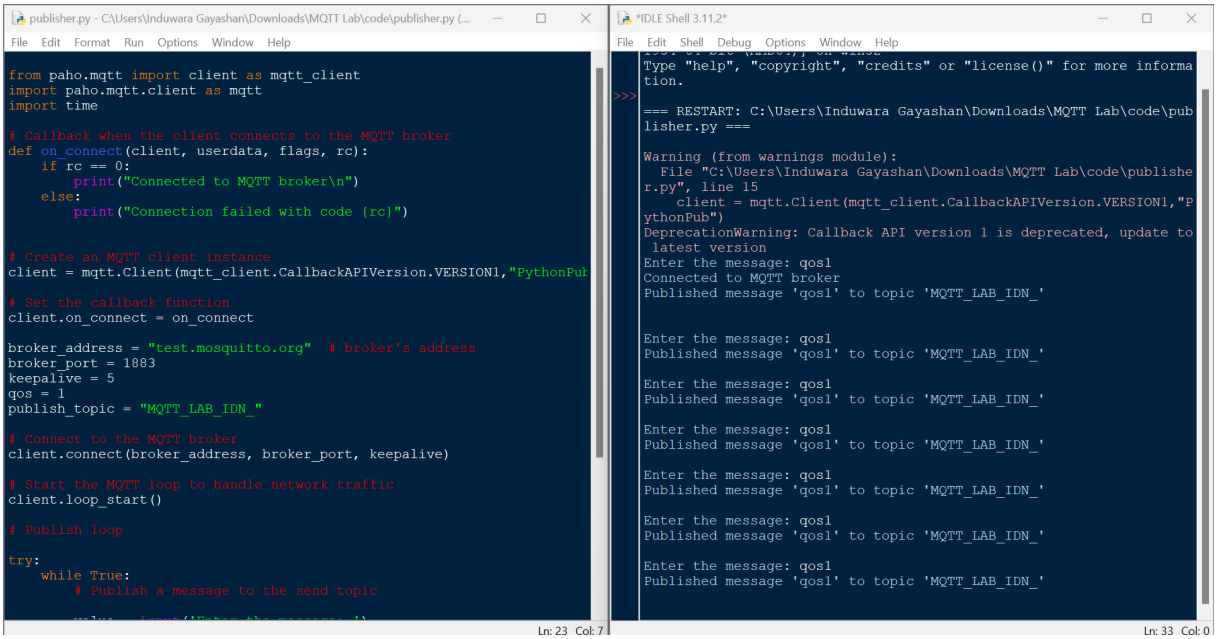


2 Step 3-A(QoS level 1)

2.1 Publisher

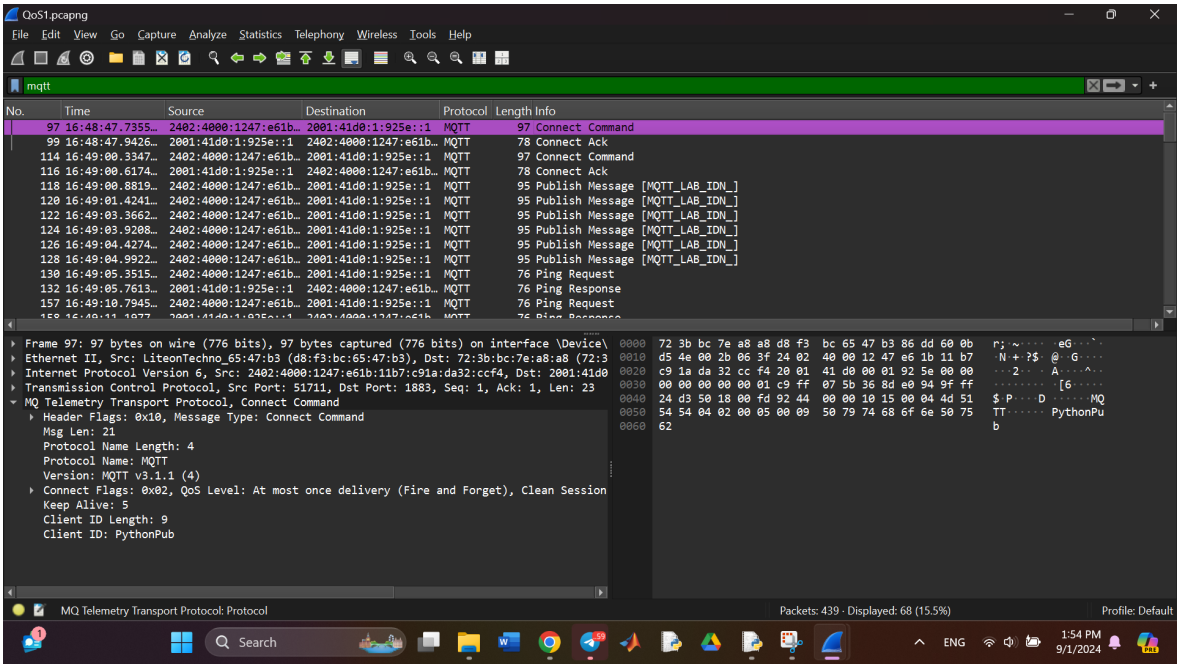
2.1.1 Connecting the Publisher

- **Topic :** MQTT_LAB_IDN_
- **Message :** qos1

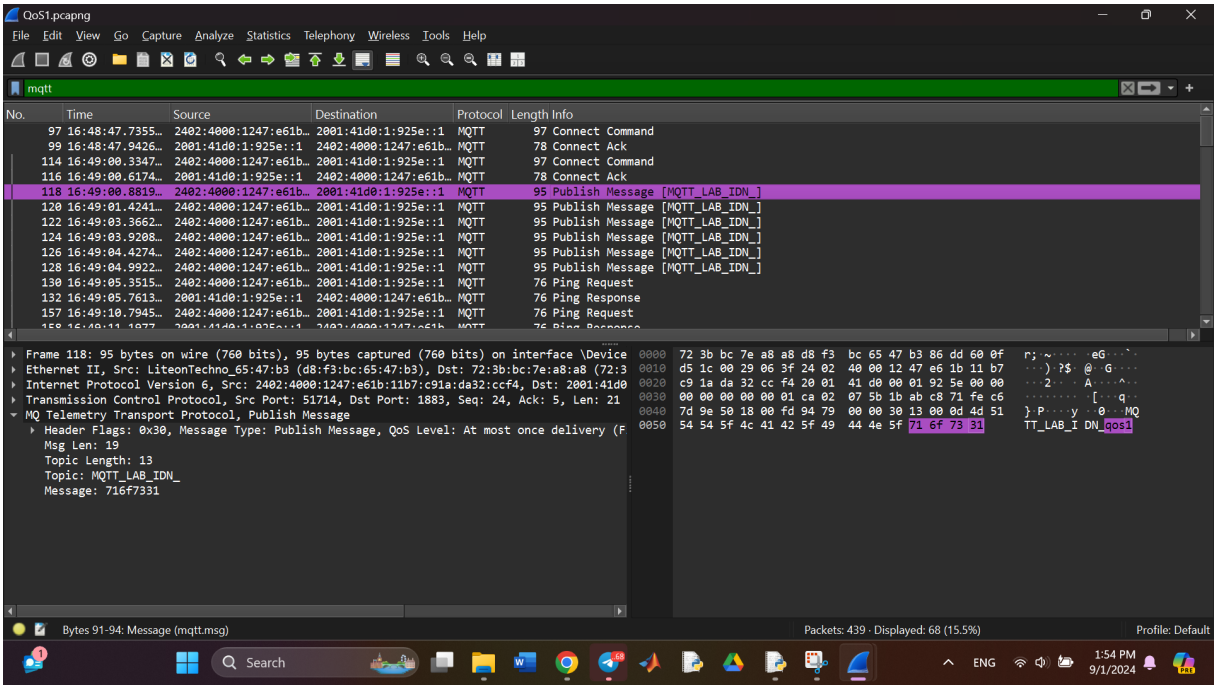


2.1.2 Wireshark Analysis

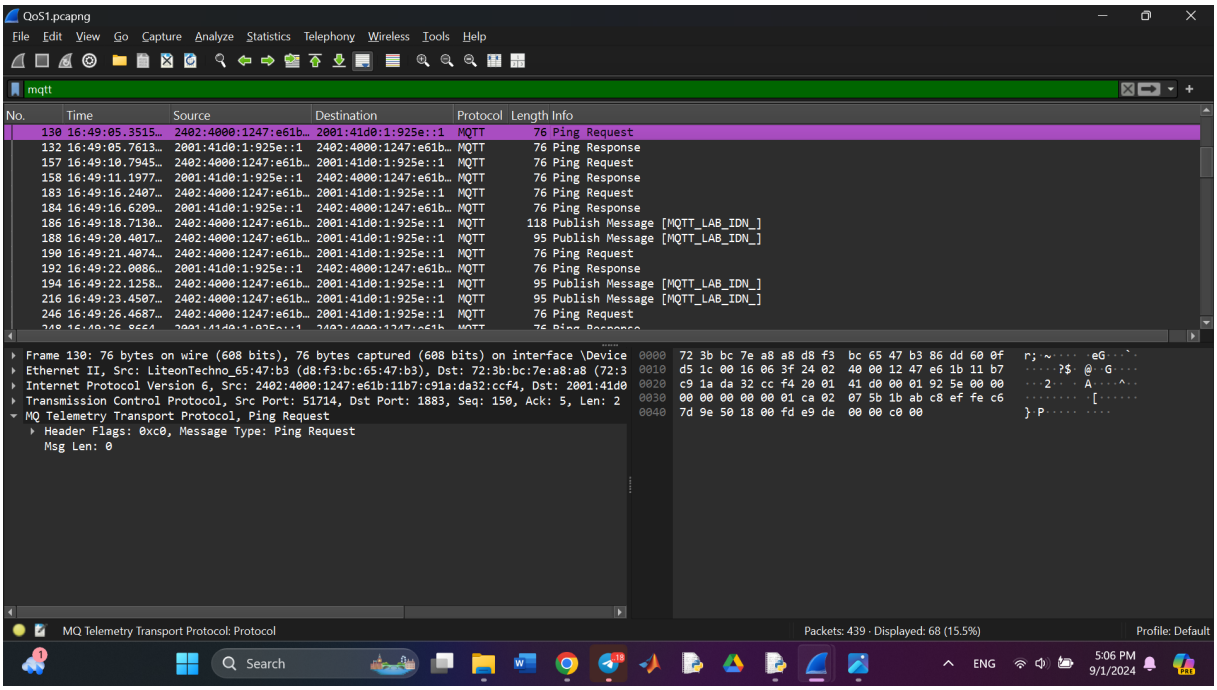
- **Connect Command :** A Wireshark capture of the Connect Command at a QoS level of 1 is shown below.

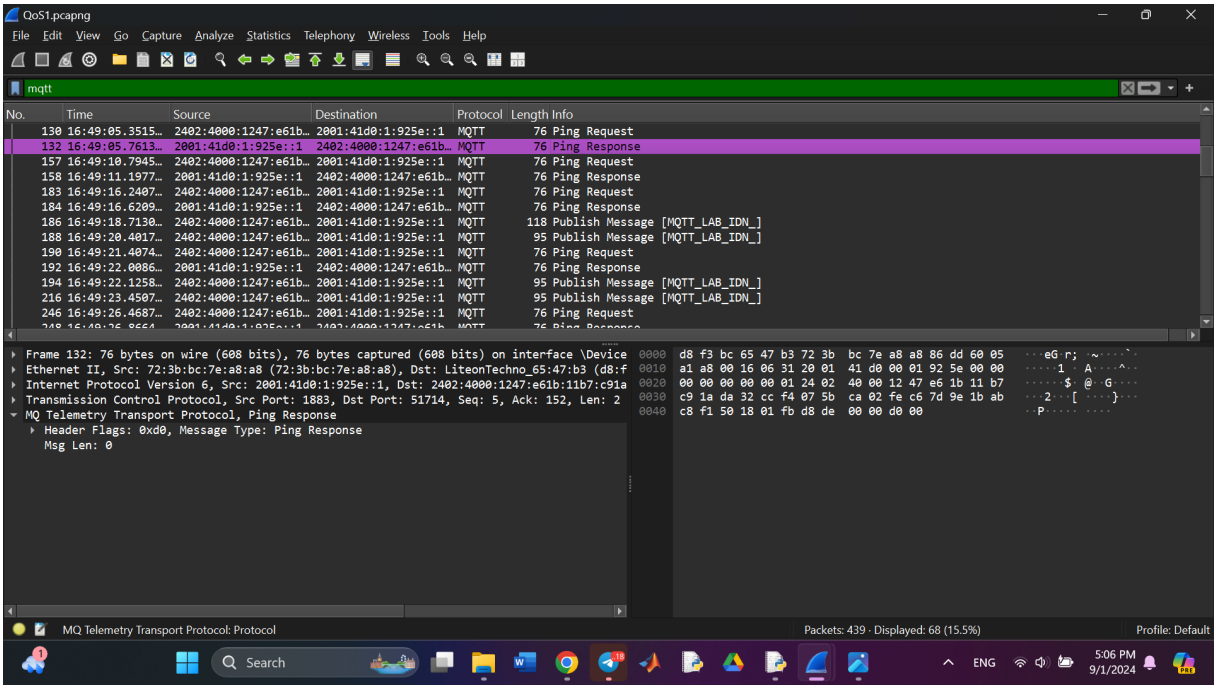


- **Publish Message :** The highlighted packet is an MQTT Publish message with a QoS level of 1. This means that the message is guaranteed to be delivered at least once to the topic **MQTT_LAB_IDN_**.The actual message content, **qos1** is also visible, as shown below.



- **Keep-alive message exchange :** Wireshark traces of **Ping request** and **Ping response** messages for the QoS level of 1 instance are shown below

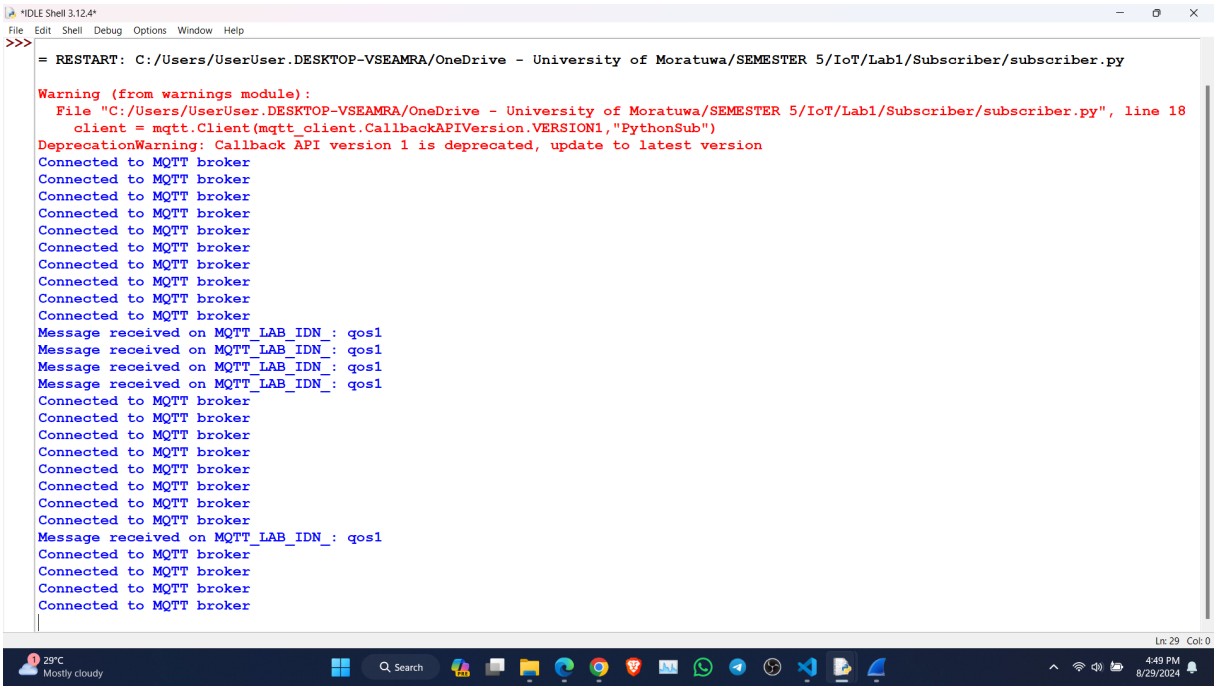




2.2 Subscriber

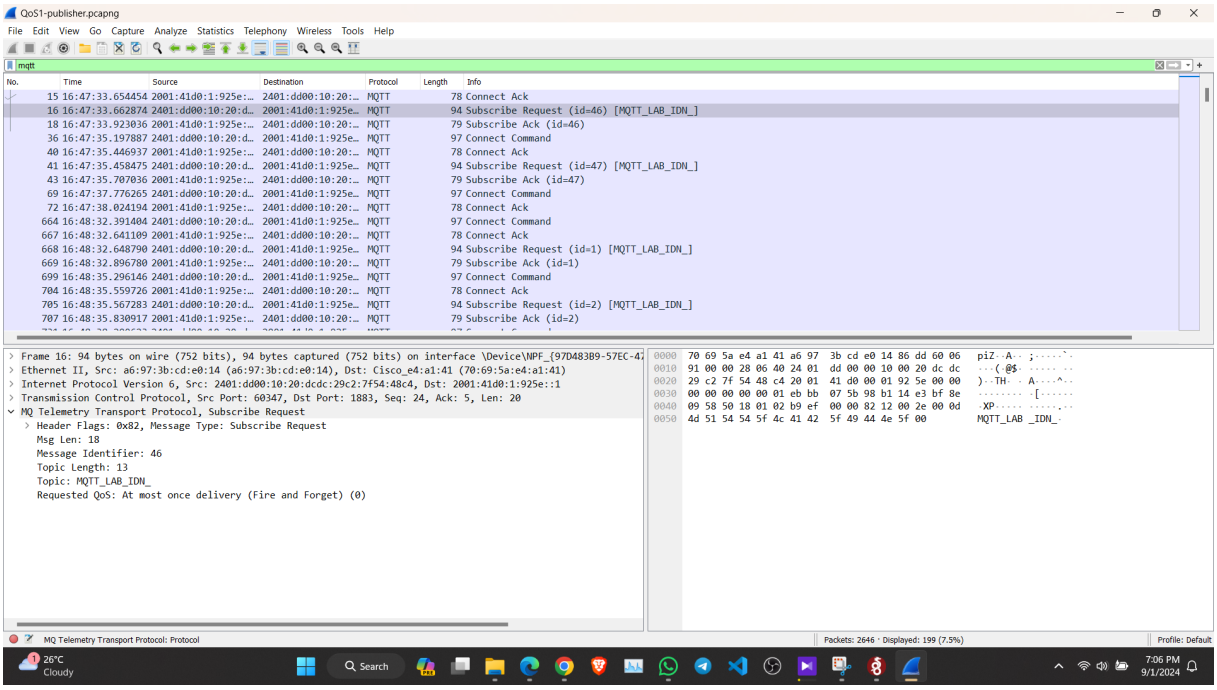
2.2.1 Message Received from Publisher

- **Topic :** MQTT_LAB_IDN_
- **Message :** qos1

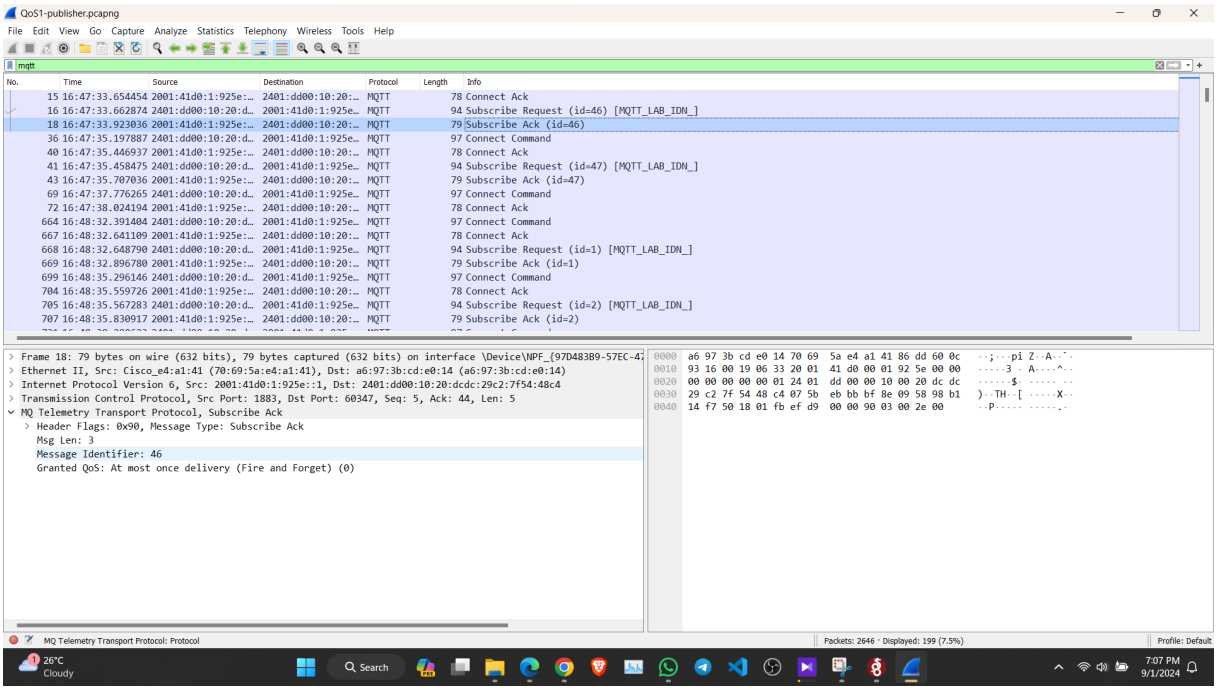


2.2.2 Wireshark Analysis

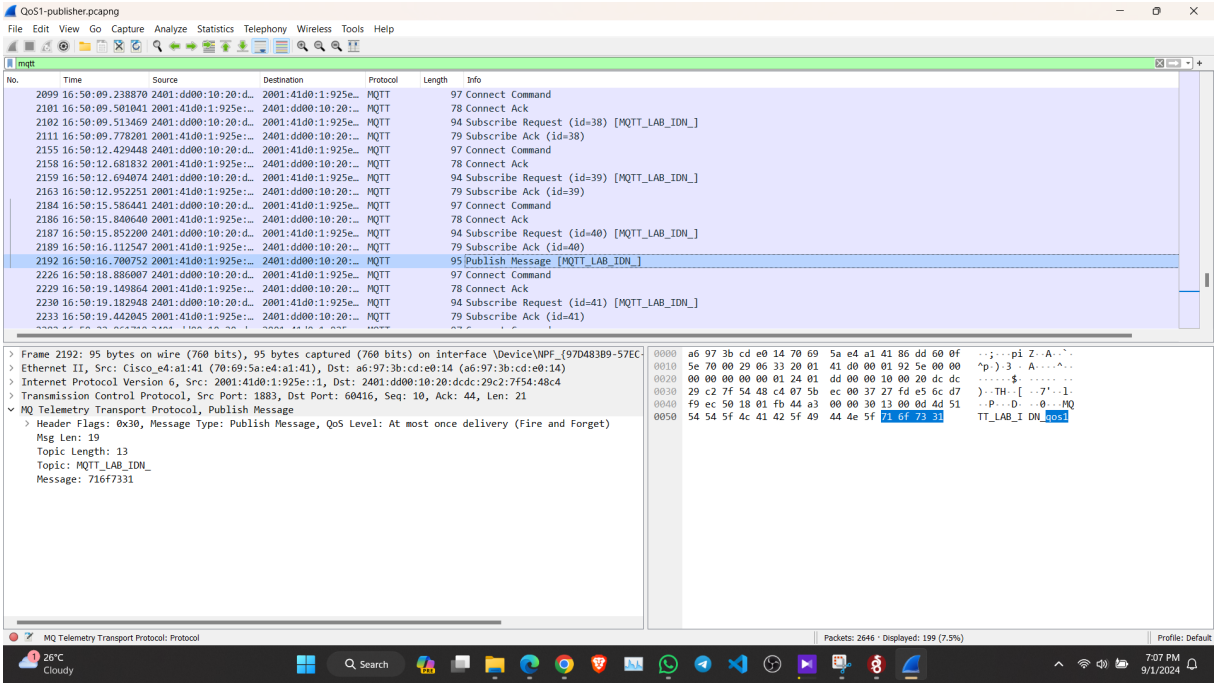
- **Subscribe Request [id=46] [MQTT_LAB_IDN_]:** After connecting, the client sends a `SUBSCRIBE` packet (with ID 46) to the broker. This packet tells the broker that the client wants to subscribe to a topic (in this case, `MQTT_LAB_IDN_`).



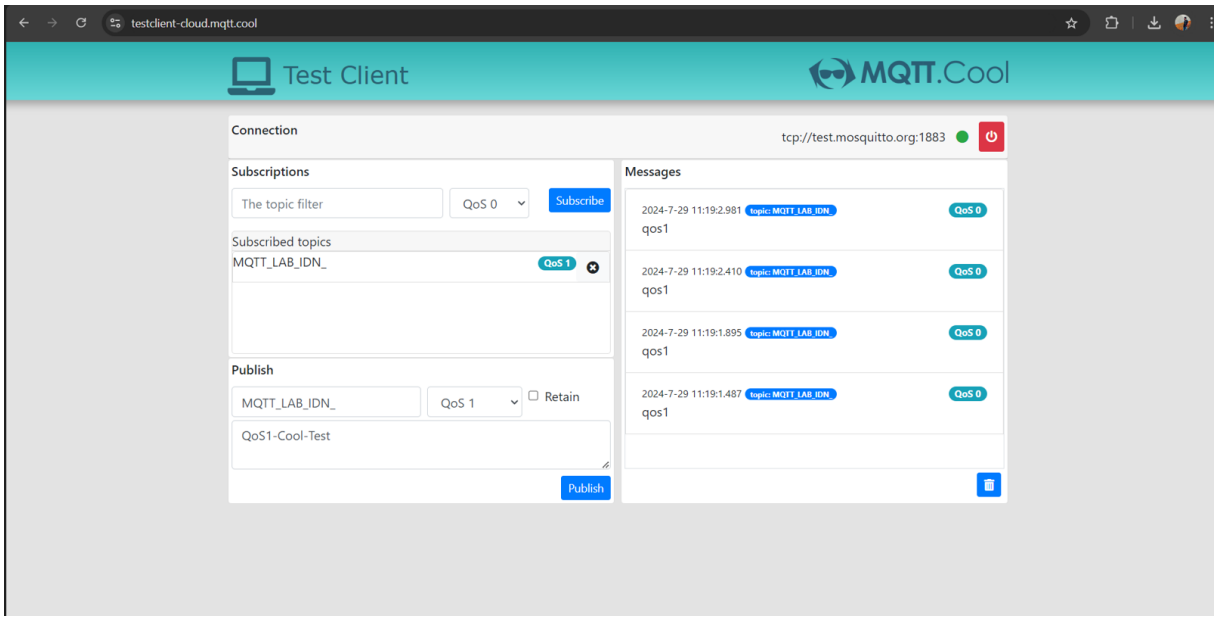
- **Subscribe Ack [id=46]:** The broker acknowledges the subscription request by sending a `SUBACK` packet, confirming that the subscription (with ID 46) was successful.



- **Received message packet:** The MQTT broker delivers this published message to all subscribed clients, including this computer, which has already subscribed to the "MQTT_LAB_IDN_" topic. The message content is 'qos1', as decoded from the hexadecimal payload 71 6f 73 31.



2.3 MQTT.Cool test client

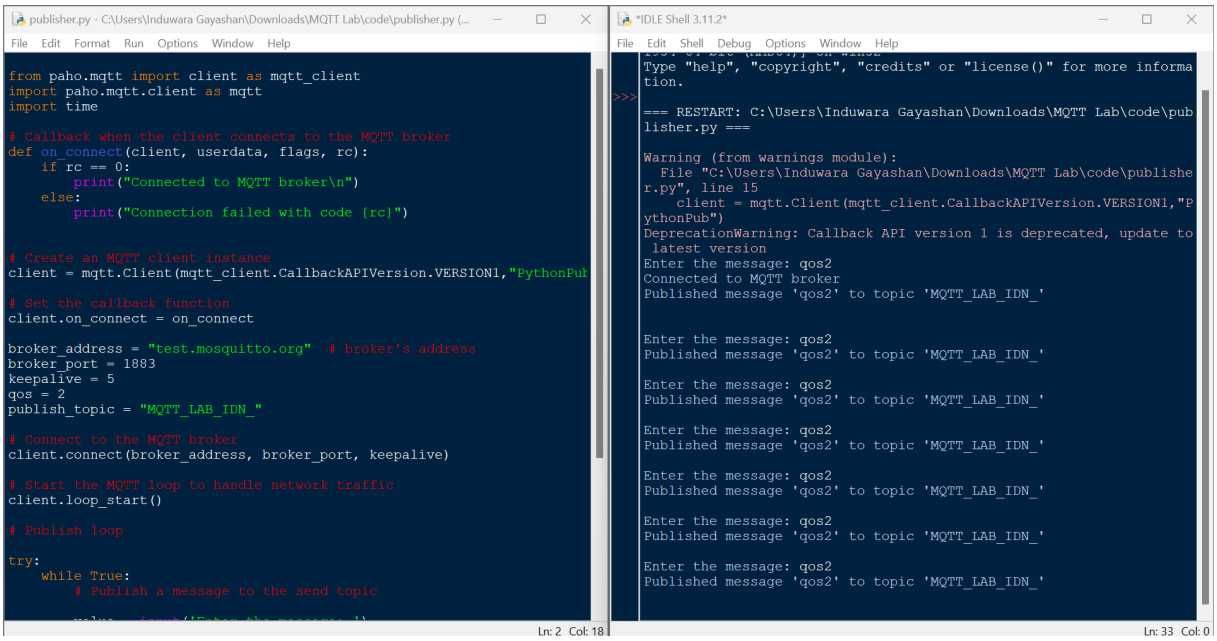


3 Step 3-B(QoS level 2)

3.1 Publisher

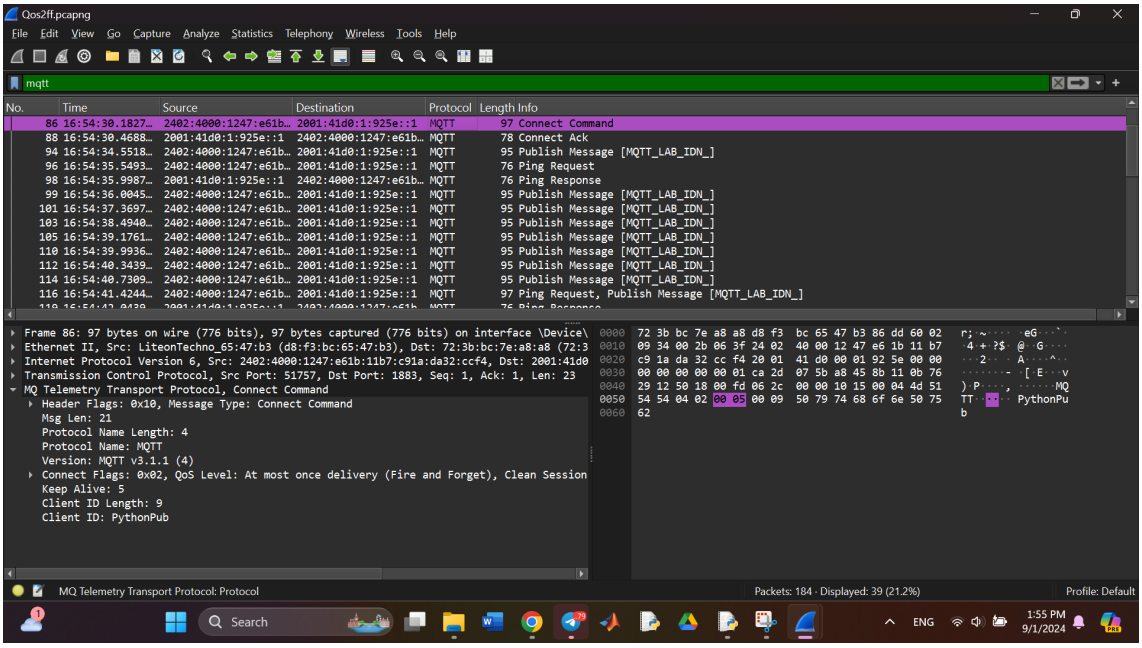
3.1.1 Connecting the Publisher

- **Topic :** MQTT_LAB_IDN_
- **Message :** qos2

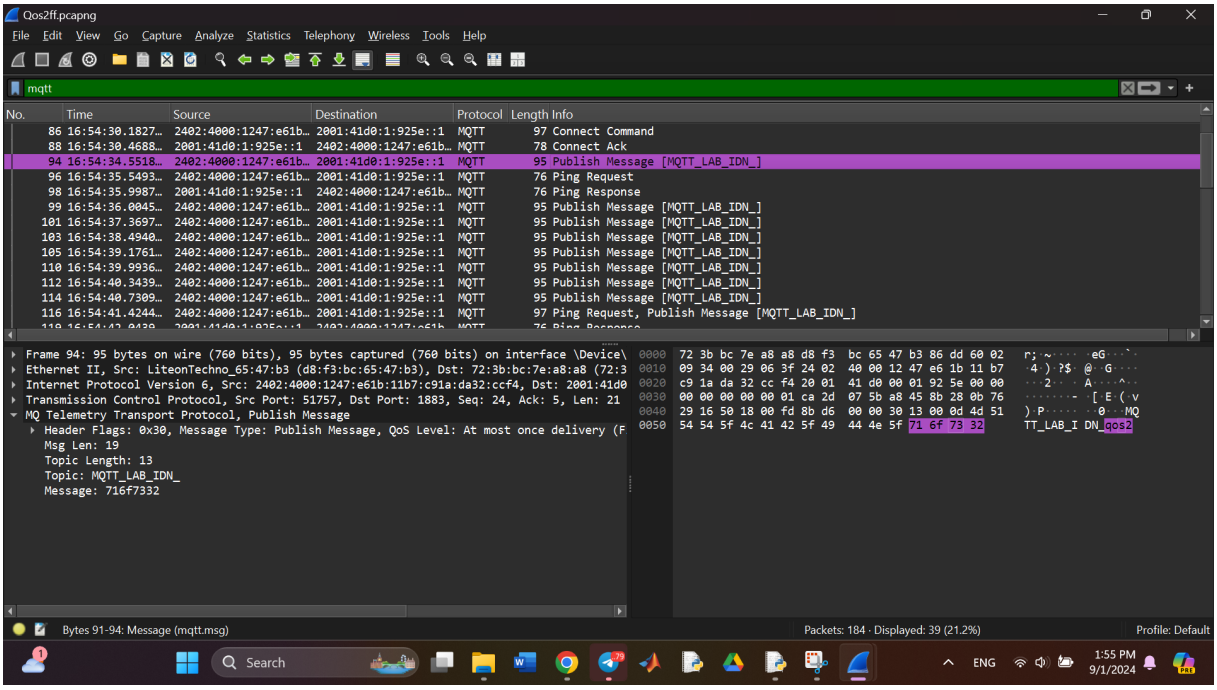


3.1.2 Wireshark Analysis

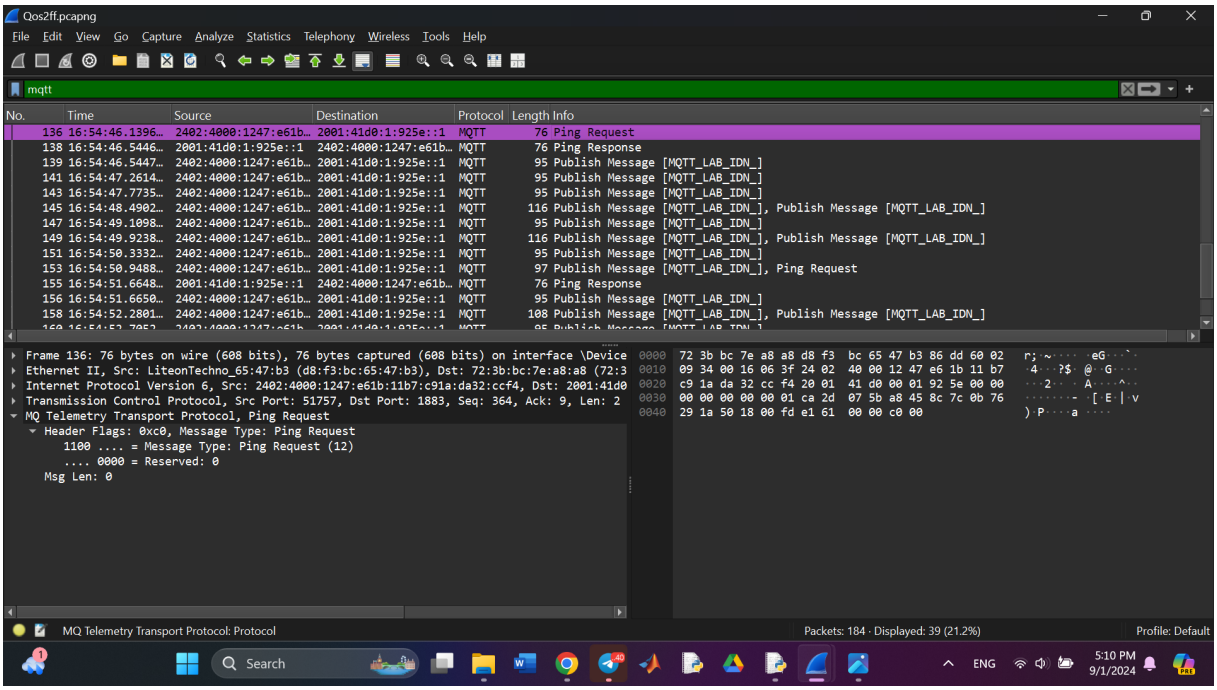
- **Connect Command :** A Wireshark capture of the Connect Command at a QoS level of 2 is shown below.

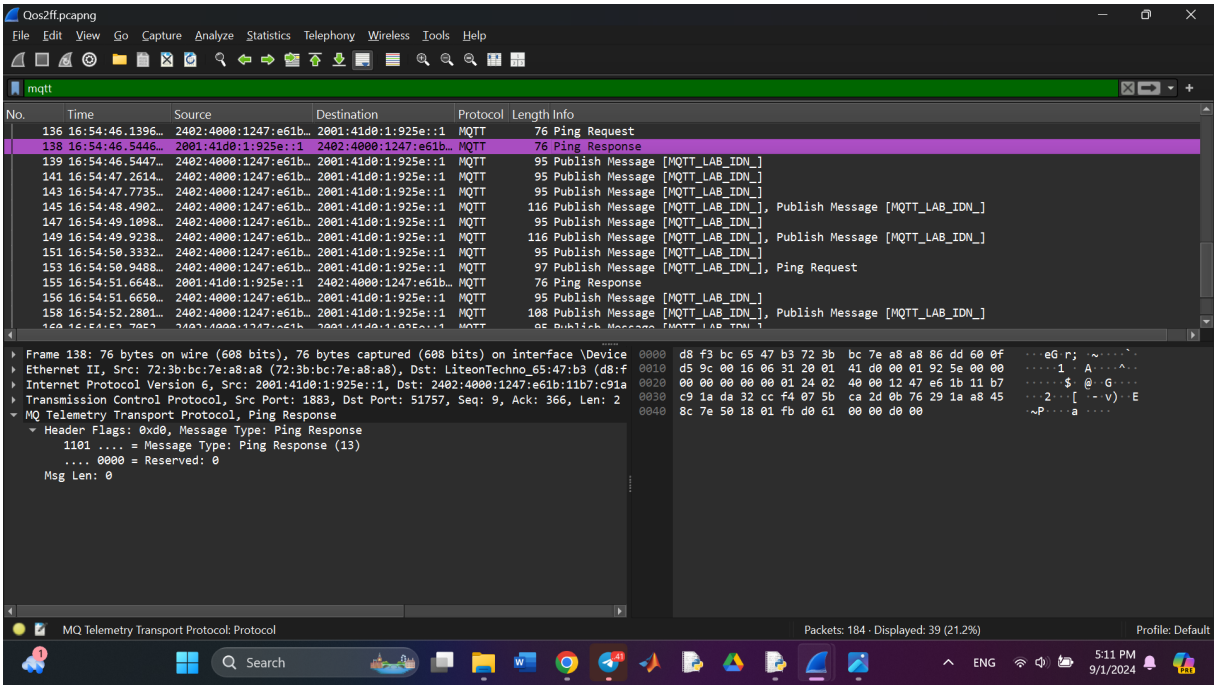


- **Publish Message :** The highlighted packet is an MQTT Publish message with a Quality of Service (QoS) level of 2. This level guarantees that the message is delivered exactly once to the topic **MQTT_LAB_IDN_**.The actual message content, **qos2** is also visible, as shown below.



- **Keep-alive message exchange :** Wireshark traces of **Ping request** and **Ping response** messages for the QoS level of 2 instance are shown below

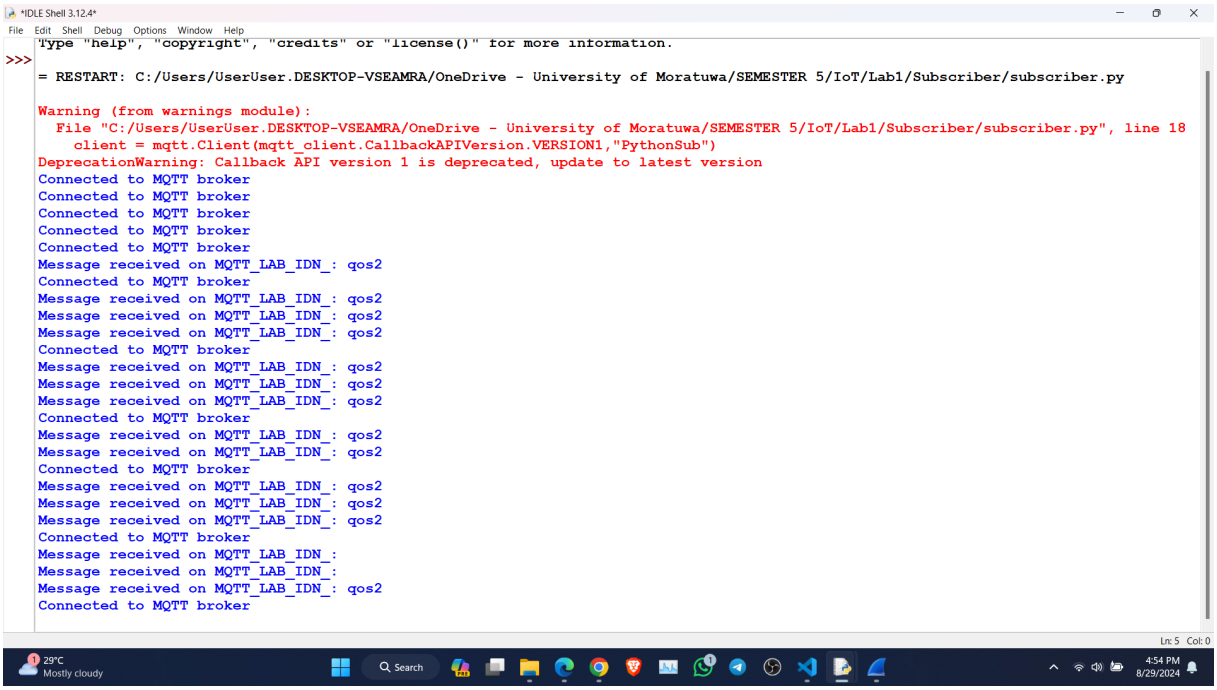




3.2 Subscriber

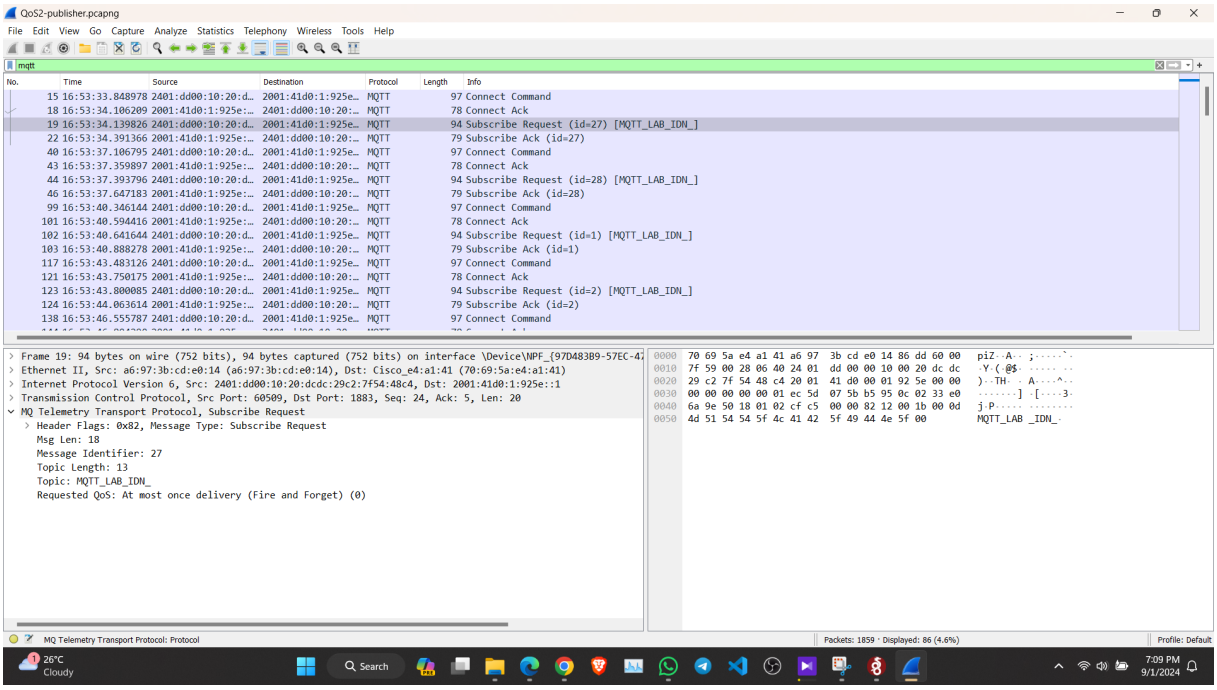
3.2.1 Message Received from Publisher

- **Topic :** MQTT_LAB_IDN_
- **Message :** qos2

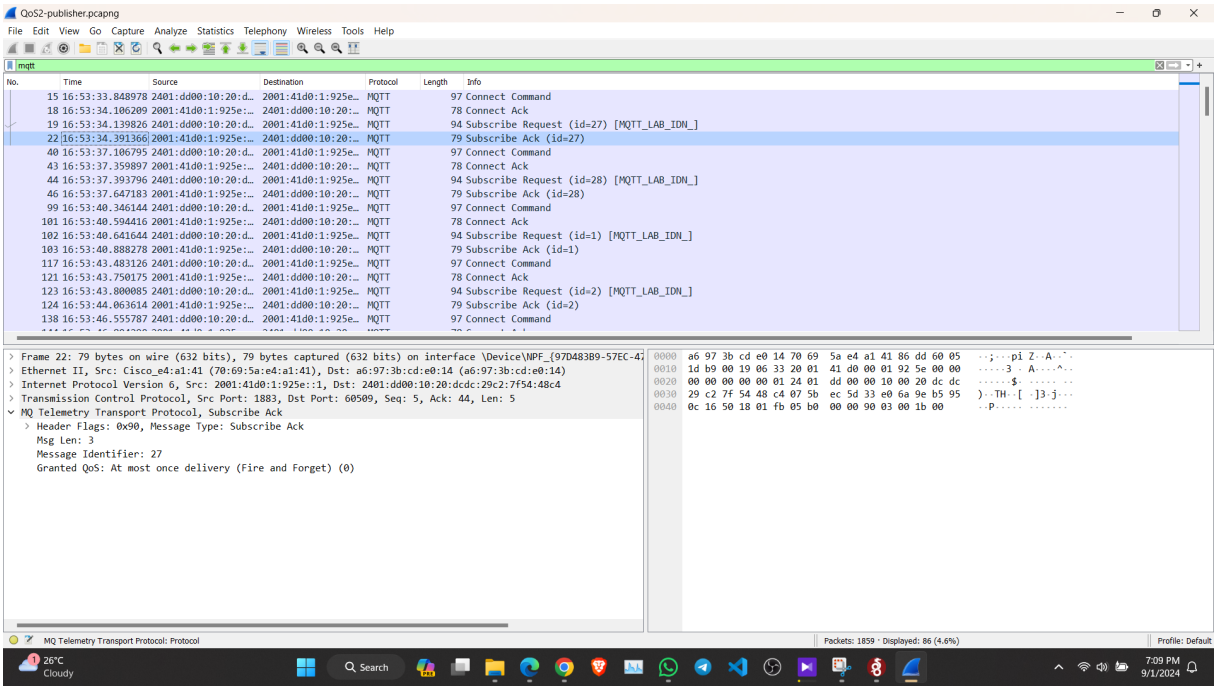


3.2.2 Wireshark Analysis

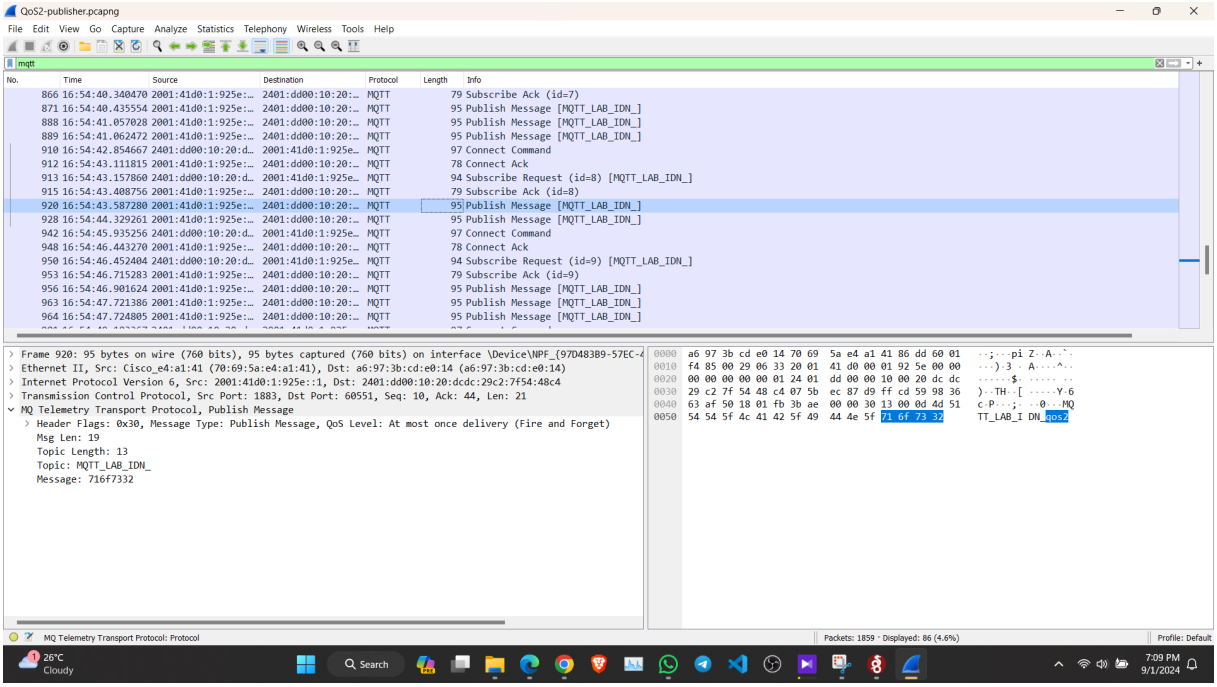
- **Subscribe Request [id=27] [MQTT_LAB_IDN_]:** After connecting, the client sends a `SUBSCRIBE` packet (with ID 27) to the broker. This packet tells the broker that the client wants to subscribe to a topic (in this case, `MQTT_LAB_IDN_`).



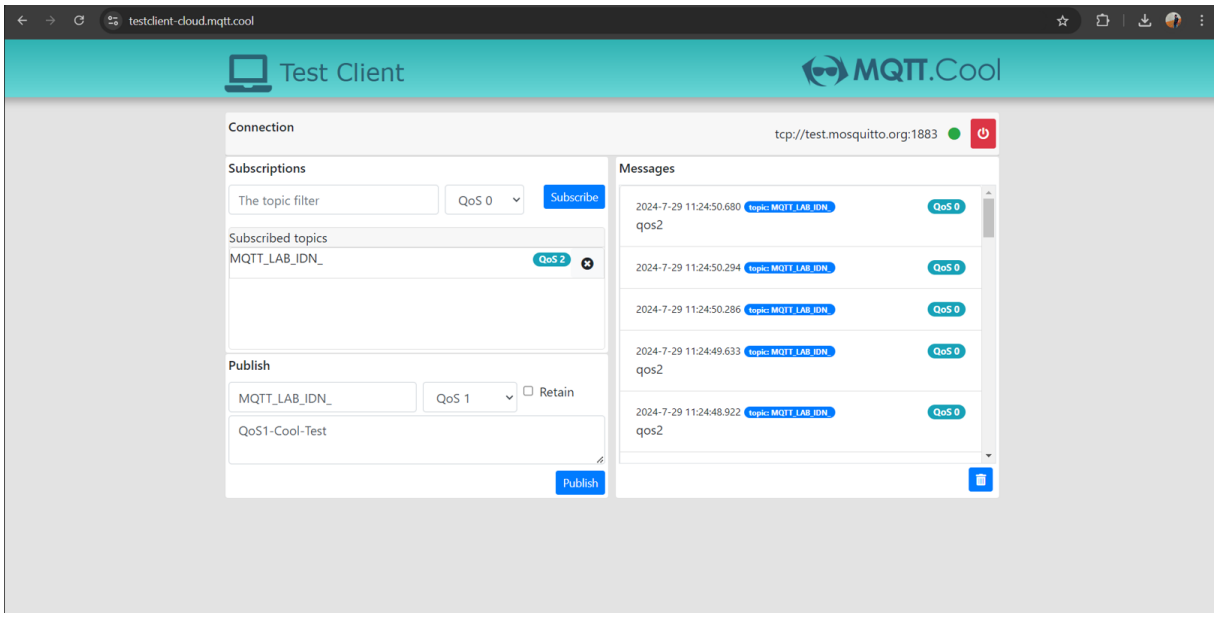
- **Subscribe Ack [id=27]:** The broker acknowledges the subscription request by sending a `SUBACK` packet, confirming that the subscription (with ID 27) was successful.



- **Received message packet:** The MQTT broker delivers this published message to all subscribed clients, including this computer, which has already subscribed to the "MQTT_LAB_IDN_" topic. The message content is 'qos2', as decoded from the hexadecimal payload 71 6f 73 32.



3.3 MQTT.Cool test client



4 Observations

In the lab, MQTT messages were sent using an MQTT broker across three scenarios, each with different QoS levels (QoS = 0, QoS = 1, QoS = 2). The key observations and behaviors for each QoS level are as follows:

- **QoS 0 (At most once):**
  - **Behavior:** Messages are delivered at most once, meaning they are sent to the broker without any guarantee of delivery. The sender does not wait for any acknowledgment (ACK) from the broker. If the message is lost during transmission, it will not be resent.
  - **Observation:** The message may or may not be delivered to the subscriber. If the network is reliable, messages will likely be delivered as expected. However, during network instability, messages may be lost without any indication of failure. No retransmission attempts are made.
  - **Example Scenario:** When network congestion or drop occurs, some messages may be missing in the subscriber's log.
- **QoS 1 (At least once):**
  - **Behavior:** Messages are delivered at least once. The sender requires an acknowledgment from the broker to confirm that the message was received. However, due to possible network issues or retransmissions, messages may be delivered more than once.
  - **Observation:** The message is typically received by the subscriber, but duplicates might appear. This is due to the sender retransmitting the message until it receives an acknowledgment from the broker. In a stable network, no duplicates may be seen, but under network issues, duplicate messages could be logged.
  - **Example Scenario:** During the test, you might observe some messages arriving twice at the subscriber's end, especially in cases of network interruptions.
- **QoS 2 (Exactly once):**
  - **Behavior:** Messages are delivered exactly once. This is the highest level of message delivery guarantee. It involves a four-step handshake between the sender and the broker to ensure that the message is received exactly once without any duplication.
  - **Observation:** The message is guaranteed to be delivered exactly once. The subscriber receives the message without any duplication, even under network issues. This QoS level introduces higher latency compared to QoS 0 and QoS 1 because of the additional acknowledgment steps required to ensure exact delivery.
  - **Example Scenario:** The subscriber will see every message exactly once, even if there were network issues, though with a slight delay compared to QoS 0 and QoS 1.

5 Full-blown publish/subscribe client - Temperature Monitoring System

This system uses the MQTT protocol to facilitate communication between two key components: the controller and the heater. The system is designed to monitor and control the temperature of an environment by enabling commands and receiving status updates through MQTT messaging.

5.1 Controller

The controller allows the user to send commands to set the temperature threshold, turn the heater on or off, and request the current status.

Listing 1: Controller Code: controller.py

```
import paho.mqtt.client as mqtt
import time

# MQTT server details
BROKER = "test.mosquitto.org"
PORT = 1883
PUBLISH_TOPIC = 'TemperatureCommands'
SUBSCRIBE_TOPIC = 'TemperatureStatus'
PUBLISH_QOS = 0
SUBSCRIBE_QOS = 0

# Callback function when a message is received
def on_message(client, userdata, message):
    received_msg = message.payload.decode()
    print(f"Status Update: {received_msg}")

# Initialize MQTT client
client = mqtt.Client()
client.on_message = on_message

# Connect to MQTT broker
client.connect(BROKER, PORT, 60)

# Subscribe to status updates
client.subscribe(SUBSCRIBE_TOPIC, SUBSCRIBE_QOS)
print(f"Subscribed to topic '{SUBSCRIBE_TOPIC}' for status updates")

def send_command(command):
    client.publish(PUBLISH_TOPIC, command, qos=PUBLISH_QOS)
    print(f"Sent command: {command}")

# Start the MQTT client loop
client.loop_start()

# Example interaction loop for sending commands
try:
    while True:
        print("\nCommand Center - Choose an action:")
        print("1. Set Temperature Threshold")
        print("2. Turn Heater ON")
        print("3. Turn Heater OFF")
        print("4. Request Status")
        choice = input("Enter choice: ")

        if choice == '1':
            threshold = input("Enter temperature threshold ( C ): ")
            send_command(f"SET_THRESHOLD {threshold}")
        elif choice == '2':
            send_command("HEATER_ON")
```

```
        elif choice == '3':
            send_command("HEATER_OFF")
        elif choice == '4':
            send_command("STATUS_REQUEST")
        else:
            print("Invalid choice. Please try again.")

        time.sleep(1)  # Short delay between commands

except KeyboardInterrupt:
    print("Command Center stopped.")

# Stop the MQTT client loop
client.loop_stop()
```

5.2 Heater

The heater module monitors the temperature and responds to commands to maintain the temperature threshold set by the user.

Listing 2: Heater Code: heater.py

```
import paho.mqtt.client as mqtt
import time

# MQTT server details
BROKER = "test.mosquitto.org"
PORT = 1883
PUBLISH_TOPIC = 'TemperatureStatus'
SUBSCRIBE_TOPIC = 'TemperatureCommands'
PUBLISH_QOS = 0
SUBSCRIBE_QOS = 0

# State variables
heater_status = "OFF"
temperature_threshold = None
temperature = 25  # Dummy current temperature

# Callback function when a message is received
def on_message(client, userdata, message):
    global heater_status, temperature_threshold

    received_msg = message.payload.decode()
    print(f"Received command: {received_msg}")

    # Parse and execute the received command
    if received_msg.startswith("SET_THRESHOLD"):
        _, threshold_str = received_msg.split()
        temperature_threshold = float(threshold_str)
        print(f"Temperature threshold set to {temperature_threshold} C ")
    elif received_msg == "HEATER_ON":
        heater_status = "ON"
        print("Heater turned ON")
        send_status(client)
    elif received_msg == "HEATER_OFF":
        heater_status = "OFF"
        print("Heater turned OFF")
        send_status(client)
    elif received_msg == "STATUS_REQUEST":
        send_status(client)

# Callback function when a message is published
def on_publish(client, userdata, mid):
```

```
    print(f"Status message {mid} published.")

# Function to send the current status
def send_status(client):
    status_msg = f"STATUS Heater: {heater_status}, Threshold: {
        temperature_threshold}, Current Temp: {temperature}"
    client.publish(PUBLISH_TOPIC, status_msg, qos=PUBLISH_QOS)
    print(f"Sent status: {status_msg}")

# Function to handle heating logic
def check_temperature(client):
    global heater_status
    if temperature_threshold is not None:
        if temperature < temperature_threshold and heater_status == "OFF":
            heater_status = "ON"
            client.publish(PUBLISH_TOPIC, "Heater ON", qos=PUBLISH_QOS)
            print(f"Temperature below threshold. Heater turned ON.")
            send_status(client)
        elif temperature >= temperature_threshold and heater_status == "ON":
            heater_status = "OFF"
            client.publish(PUBLISH_TOPIC, "Heater OFF", qos=PUBLISH_QOS)
            print(f"Temperature above threshold. Heater turned OFF.")
            send_status(client)

# Initialize MQTT client
client = mqtt.Client()
client.on_message = on_message
client.on_publish = on_publish

# Connect to MQTT broker
client.connect(BROKER, PORT, 60)

# Subscribe to command topic
client.subscribe(SUBSCRIBE_TOPIC, SUBSCRIBE_QOS)
print(f"Subscribed to topic '{SUBSCRIBE_TOPIC}' for commands")

# Start the MQTT client loop
client.loop_start()

# Main loop for checking temperature and updating status
try:
    while True:
        check_temperature(client)
        time.sleep(1)  # Check every second

except KeyboardInterrupt:
    print("Temperature Controller stopped.")

# Stop the MQTT client loop
client.loop_stop()
```

5.3 Testing Results

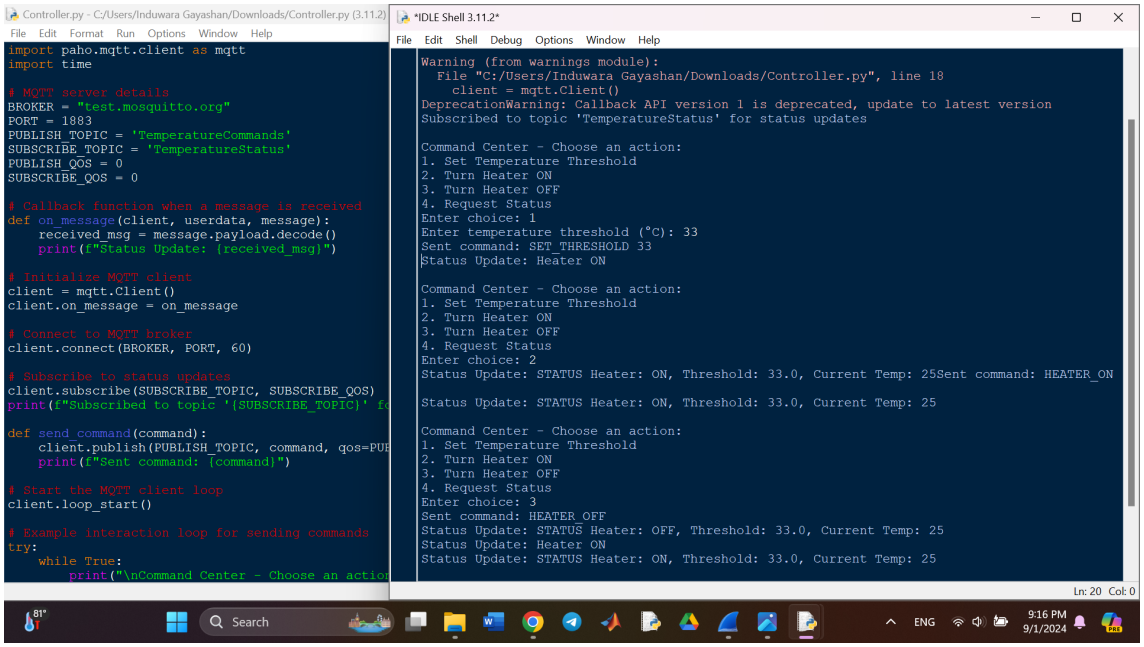


Figure 1: The controller interface for sending commands and receiving status updates.

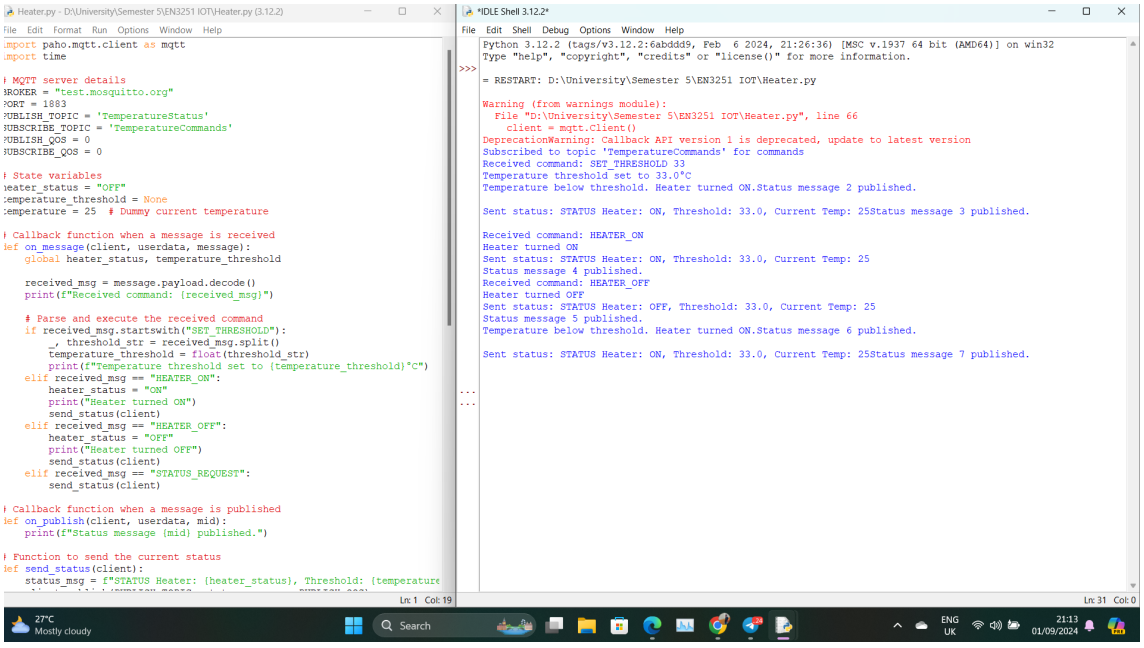


Figure 2: The heater module that responds to commands to maintain the temperature threshold.