Department of Electronic and Telecommunication Engineering
University of Moratuwa
Sri Lanka

EN3251 - Internet of Things

Laboratory Exercise 1
MQTT Implementation and Testing

Group 9

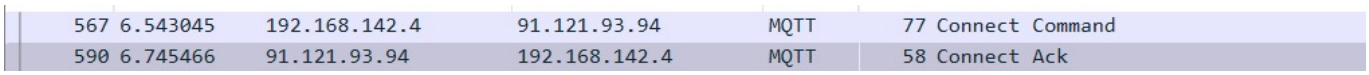
Name	Index Number
M. P. Wickramarathne	210703V
A.A.W.L.R.Amarasinghe	210031H
A.D.T. Dabare	210089P

This report is submitted as a partial fulfillment of module EN3251
2024.09.01

Contents

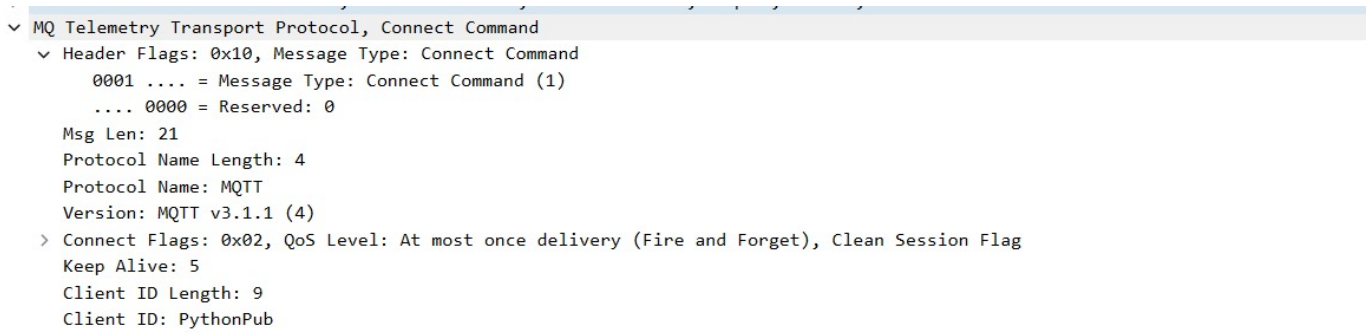
1 Connecting to the Broker

1.1 Publisher



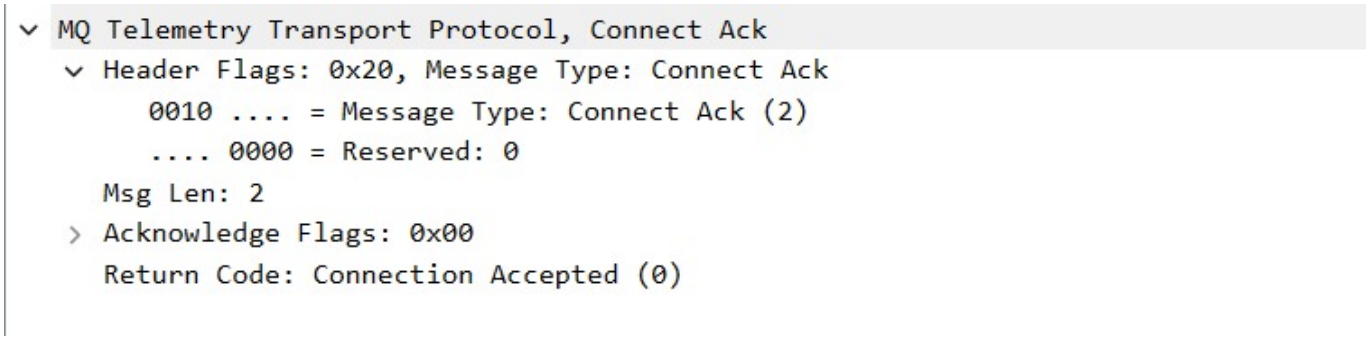
567	6.543045	192.168.142.4	91.121.93.94	MQTT	77 Connect Command
590	6.745466	91.121.93.94	192.168.142.4	MQTT	58 Connect Ack

Figure 1: Publisher Connect Wireshark



```
MQ Telemetry Transport Protocol, Connect Command
  Header Flags: 0x10, Message Type: Connect Command
      0001 .... = Message Type: Connect Command (1)
      .... 0000 = Reserved: 0
    Msg Len: 21
    Protocol Name Length: 4
    Protocol Name: MQTT
    Version: MQTT v3.1.1 (4)
  > Connect Flags: 0x02, QoS Level: At most once delivery (Fire and Forget), Clean Session Flag
    Keep Alive: 5
    Client ID Length: 9
    Client ID: PythonPub
```

Figure 2: Publisher Connect Command



```
MQ Telemetry Transport Protocol, Connect Ack
  Header Flags: 0x20, Message Type: Connect Ack
      0010 .... = Message Type: Connect Ack (2)
      .... 0000 = Reserved: 0
    Msg Len: 2
  > Acknowledge Flags: 0x00
    Return Code: Connection Accepted (0)
```

Figure 3: Publisher Connect ACK

- **Connect Command**: The first row shows the client (192.168.142.4) sending a *Connect Command* to the MQTT broker (91.121.93.94). This command initiates the connection request, where the client identifies itself and specifies any connection parameters (like keep-alive time, client ID, etc.).
- **Connect Ack**: The second row indicates that the broker (91.121.93.94) responds with a *Connect Ack (Acknowledgment)* to the client (192.168.142.4). This acknowledgment confirms that the connection has been successfully established.

1.2 Subscriber

Figure 4: Subscriber - Connect Wireshark

Figure 5: Subscriber - Connect Command

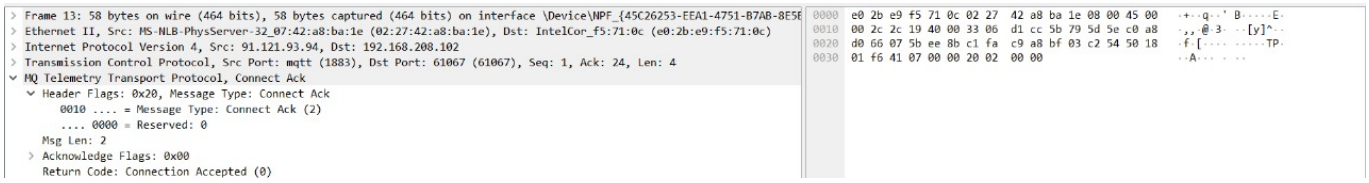


Figure 6: Subscriber - Connect ACK

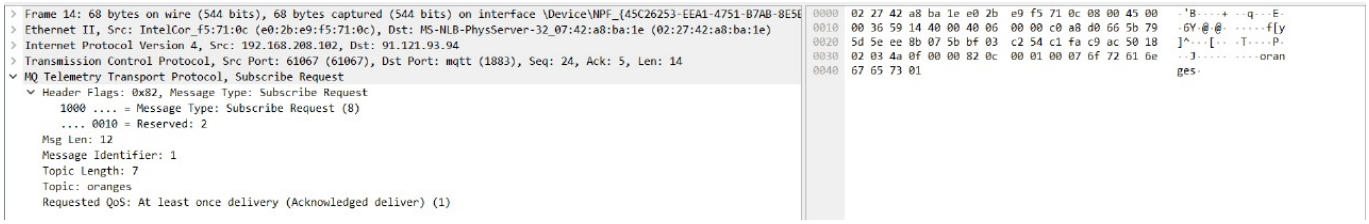


Figure 7: Subscriber - Subscribe Request

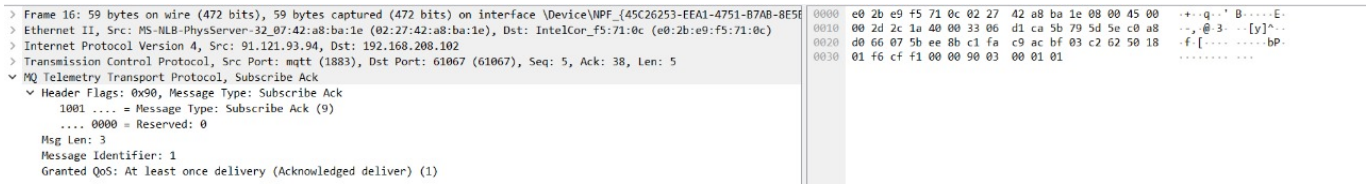


Figure 8: Subscriber - Subscribe ACK

The first packet (Packet 10) shows the MQTT client with the IP address `192.168.208.102` sending a *Connect Command* to the MQTT broker at `91.121.93.94`. This command is the initial step in establishing a connection between the client and the broker. The *Connect Command* typically includes information such as the client's identifier, protocol level, and any optional flags for features like authentication or last will messages.

In the second packet (Packet 13), the MQTT broker responds to the client's connection request with a *Connect Ack* message. This acknowledgment signifies that the broker has accepted the client's

connection request, and the communication channel is now established. The *Connect Ack* may also include information such as the session present flag, which indicates whether the broker has a persistent session for the client.

The third packet (Packet 14) involves the client sending a *Subscribe Request* to the broker. In this case, the client is subscribing to the topic "oranges," which is identified by an ID of 1. This subscription request indicates that the client wishes to receive any messages that are published to the "oranges" topic. The broker will process this request and, if successful, start sending messages on this topic to the client.

Finally, the fourth packet (Packet 16) shows the MQTT broker sending a *Subscribe Ack* message back to the client. This acknowledgment confirms that the subscription request was successful and that the client is now subscribed to the "oranges" topic. From this point onward, the client will receive messages that are published to the "oranges" topic, completing the typical MQTT subscribe workflow.

2 QoS=0 instance

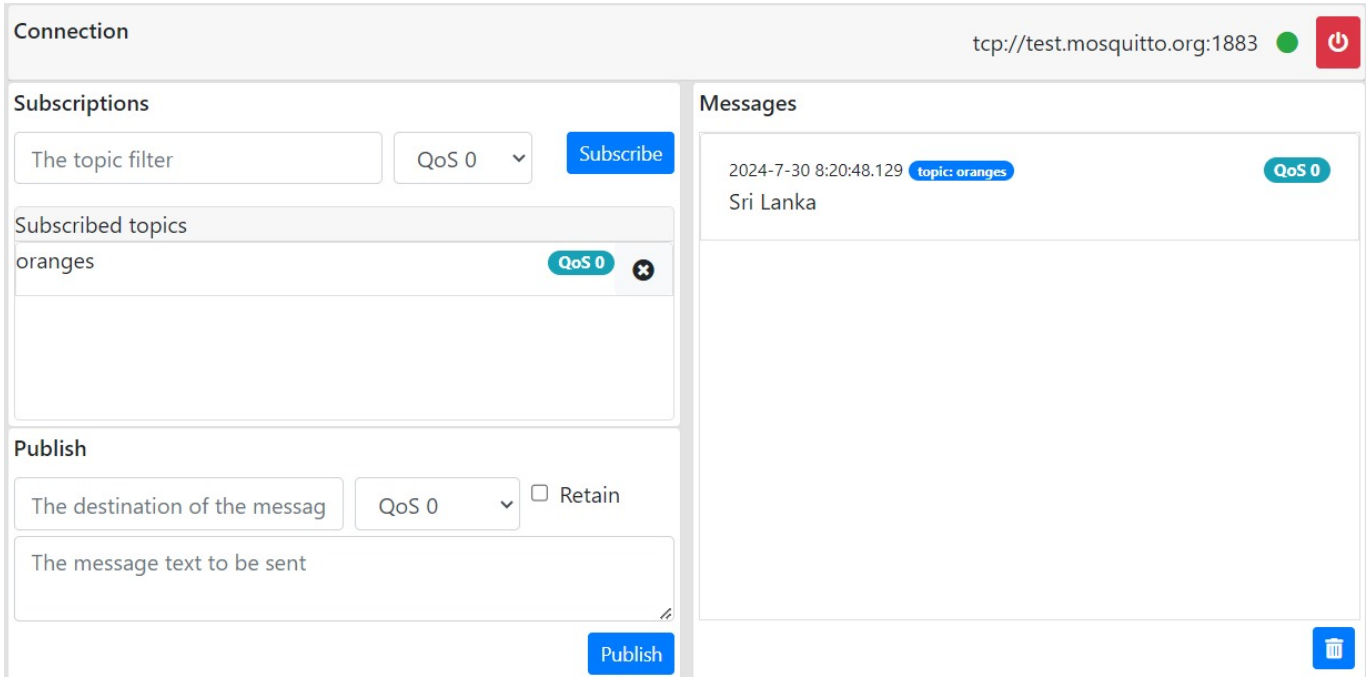


Figure 9: Test QoS 0

The client has subscribed to the topic "oranges" with a Quality of Service (QoS) level of 0, which is visible under the "Subscribed topics" section. In the "Messages" panel, the client has received a message with the content "Sri Lanka," which was published to the "oranges" topic. The QoS level for this message is also 0, indicating that the message was delivered without requiring acknowledgment. This setup demonstrates a basic MQTT interaction where the client successfully subscribes to a topic and receives a message published to that topic.

7	0.929201	192.168.142.4	91.121.93.94	MQTT	56 Ping Request
8	1.224446	91.121.93.94	192.168.142.4	MQTT	56 Ping Response
20	5.387192	192.168.142.4	91.121.93.94	MQTT	74 Publish Message [oranges]
24	6.392468	192.168.142.4	91.121.93.94	MQTT	56 Ping Request
28	6.665245	91.121.93.94	192.168.142.4	MQTT	56 Ping Response

Figure 10: Publisher QoS 0 - Wireshark

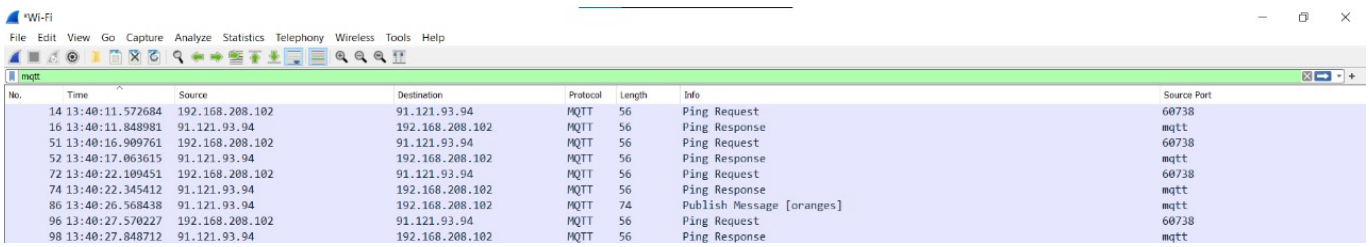


Figure 11: Subscriber QoS 0 - Wireshark

1. **Consistent Testing Environment:** Both captures display MQTT traffic, indicating that they are part of the same set of tests. The presence of "Ping Request" and "Ping Response" packets in both captures shows that the devices are maintaining a connection via the MQTT protocol, which is standard behavior in MQTT to keep the connection alive.

2. **Publish Message [oranges]:** In both captures, there is a "Publish Message [oranges]," suggesting that you were testing message publishing as part of the MQTT protocol. The repeated appearance of this specific message in both captures indicates that this was part of the testing script or sequence you used to verify the correct operation of MQTT message publishing.

3. **Different IP Pairs:** The first capture involves communication between `192.168.142.4` and `91.121.93.94`, while the second involves `192.168.208.102` and `91.121.93.94`. This suggests that the same MQTT broker or service (likely the one at `91.121.93.94`) was tested with different clients (the different local IPs). This helps validate that the broker can handle multiple clients and that the MQTT protocol works consistently across different devices.

4. **Same MQTT Process:** The patterns in both captures are identical in terms of the MQTT protocol usage (pings and publishing messages), which demonstrates that both sets of packets are part of the same overall testing process. The tests were aimed at verifying whether the MQTT protocol handles connections and message publishing reliably across different devices or scenarios.

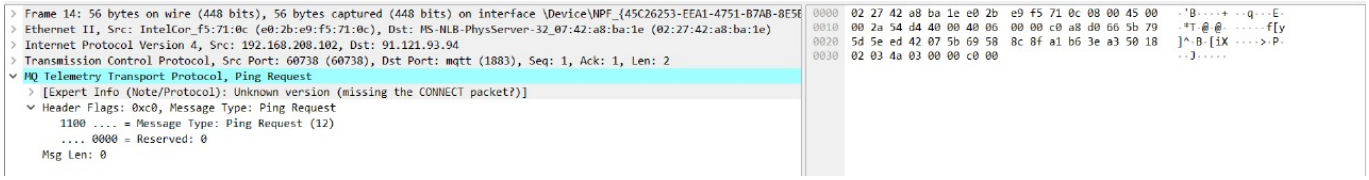


Figure 12: Subscriber QoS 0 - Ping Request

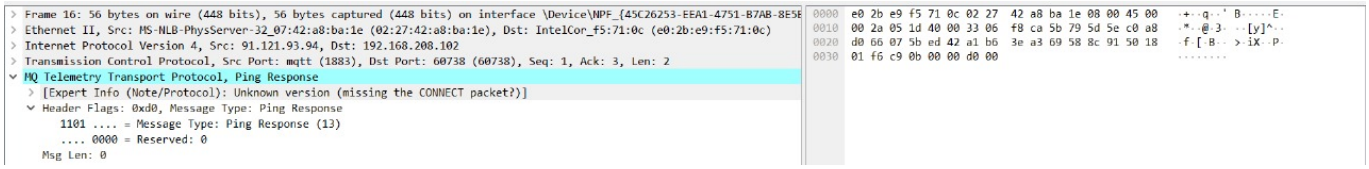


Figure 13: Subscriber QoS 0 - Ping Response

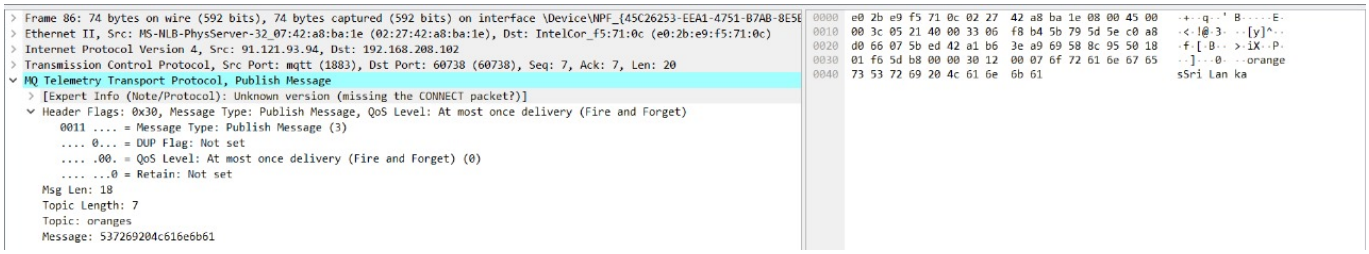


Figure 14: Subscriber QoS 0 - Publish Message

3 QoS=1 instance

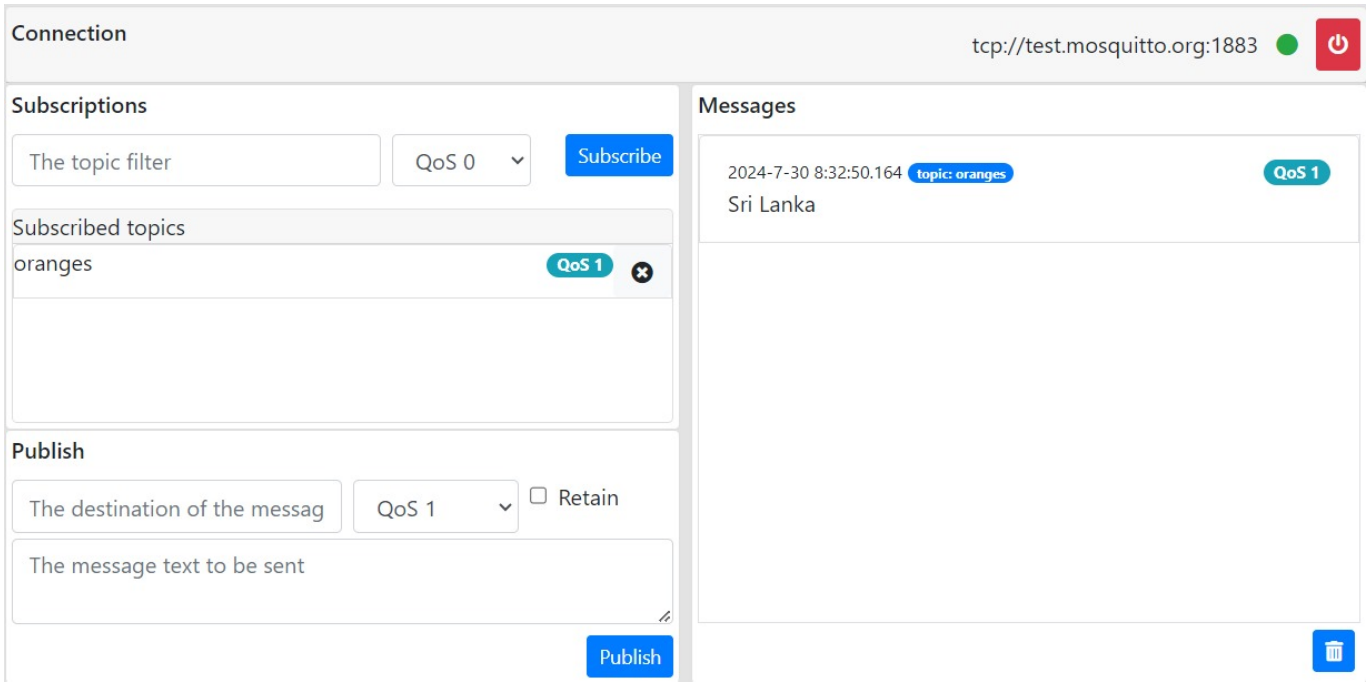


Figure 15: Test QoS 1

This demonstrates the test setup for MQTT Quality of Service (QoS) level 1 using the Mosquitto broker. The interface shows a subscription to the topic "oranges" with QoS 1, ensuring that messages are delivered at least once. A message titled "Sri Lanka" is published to the topic "oranges" and successfully received with QoS 1, as indicated in the Messages panel. The test confirms that the MQTT broker correctly handles message delivery and acknowledgment under QoS 1, where the message is delivered reliably with confirmation of receipt.

```
 6 3.043824    91.121.93.94      192.168.142.4     MQTT      56 Ping Response
30 7.334033    192.168.142.4     91.121.93.94      MQTT      73 Publish Message (id=4) [oranges]
54 7.650797    91.121.93.94      192.168.142.4     MQTT      58 Publish Ack (id=4)
```

Figure 16: Publisher QoS 1 - Wireshark

- **Ping Response:**
  The broker (91.121.93.94) sends a Ping Response to the client (192.168.142.4), confirming the MQTT connection is still active.
- **Publish Message (id=4):**
  The client (192.168.142.4) sends a Publish message with QoS 1, containing the payload "oranges" and message ID 4, ensuring that the broker receives the message at least once.
- **Publish Acknowledgment (id=4):**
  The broker (91.121.93.94) acknowledges the receipt of the Publish message by sending a Publish Ack with the same message ID 4, confirming successful delivery.

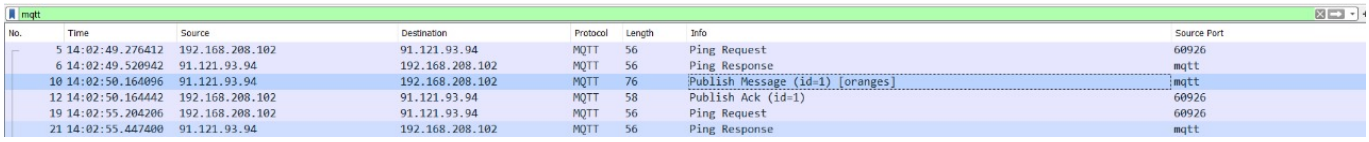


Figure 17: Subscriber QoS 1 - Wireshark

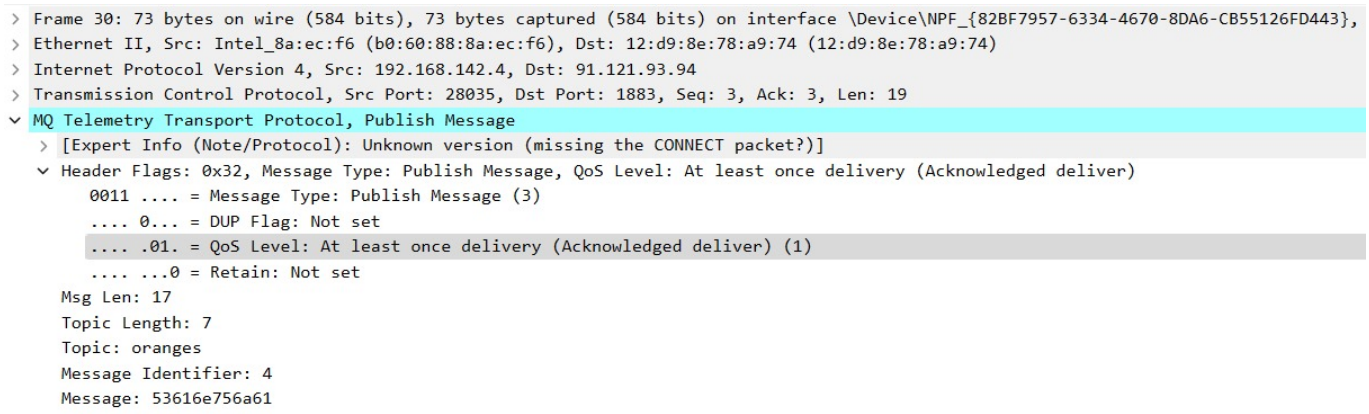


Figure 18: Subscriber QoS 1 - Publish Message

The MQTT Publish message has the topic "oranges" with QoS Level 1, meaning the message will be delivered at least once with acknowledgment. The message identifier is 4, and the payload of the message is "53616e756a61" (hexadecimal representation). The message does not have the DUP or Retain flags set, indicating it is not a duplicate and should not be stored by the broker after delivery.

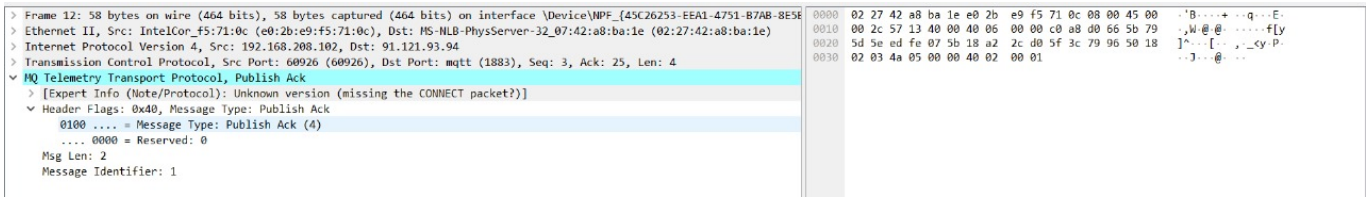


Figure 19: Subscriber QoS 1 - Publish ACK

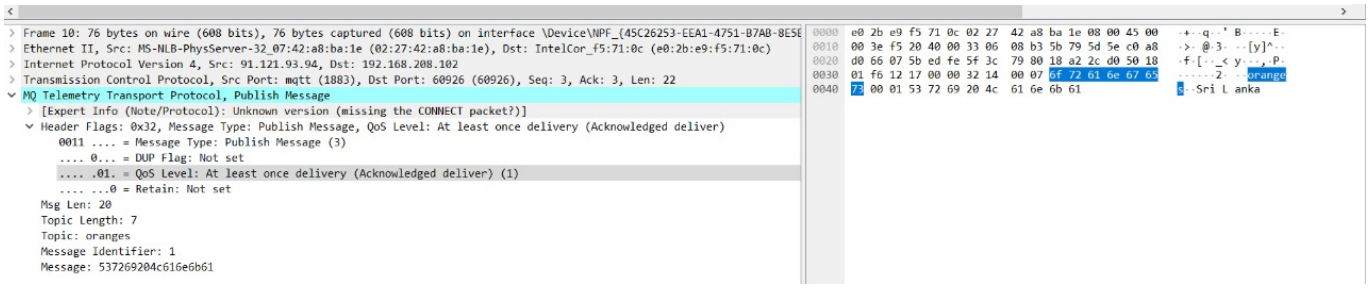


Figure 20: Subscriber QoS 1 - Publish Message

- **Ping Request:**
  The client (192.168.208.102) sends a Ping Request to the broker (91.121.93.94) to verify the MQTT connection is still active.

- **Ping Response:**
  The broker (91.121.93.94) replies to the client's Ping Request, confirming connection is alive.

- **Publish Message (id=1):**
  The broker (91.121.93.94) sends a Publish message containing "oranges" with message ID 1 to the client (192.168.208.102) under QoS 1.

- **Publish Acknowledgment (id=1):**
  The client (192.168.208.102) sends a Publish Ack with message ID 1 back to the broker (91.121.93.94), confirming successful receipt of the message.

4 QoS=2 instance

99	3.347005	192.168.142.4	91.121.93.94	MQTT	70 Publish Message (id=2) [oranges]
100	3.541833	91.121.93.94	192.168.142.4	MQTT	58 Publish Received (id=2)
101	3.542431	192.168.142.4	91.121.93.94	MQTT	58 Publish Release (id=2)
102	3.726005	91.121.93.94	192.168.142.4	MQTT	58 Publish Complete (id=2)

Figure 21: Publisher QoS 2 - Wireshark

```
MQ Telemetry Transport Protocol, Publish Received
  [Expert Info (Note/Protocol): Unknown version (missing the CONNECT packet?)]
  Header Flags: 0x50, Message Type: Publish Received
      0101 .... = Message Type: Publish Received (5)
      .... 0000 = Reserved: 0
    Msg Len: 2
    Message Identifier: 2
```

Figure 22: QoS 2 - Publish Received

```
Frame 101: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface \Device\NPF_{82BF7957-6334-4670-8DA6-CB55126FD443}
Ethernet II, Src: Intel_8a:ec:f6 (b0:60:88:8a:ec:f6), Dst: 12:d9:8e:78:a9:74 (12:d9:8e:78:a9:74)
Internet Protocol Version 4, Src: 192.168.142.4, Dst: 91.121.93.94
Transmission Control Protocol, Src Port: 28308, Dst Port: 1883, Seq: 19, Ack: 7, Len: 4
MQ Telemetry Transport Protocol, Publish Release
  [Expert Info (Note/Protocol): Unknown version (missing the CONNECT packet?)]
  Header Flags: 0x62, Message Type: Publish Release
      0110 .... = Message Type: Publish Release (6)
      .... 0010 = Reserved: 2
    Msg Len: 2
    Message Identifier: 2
```

Figure 23: QoS 2 - Publish Release

- **Publish Message (id=2):**
  The client (192.168.142.4) sends a Publish message containing "oranges" with message ID 2 to the broker (91.121.93.94) under QoS 2, ensuring the message is delivered exactly once.
- **Publish Received (id=2):**
  The broker (91.121.93.94) acknowledges the receipt of the Publish message by sending a Publish Received message with message ID 2 back to the client, indicating that the message has been received but not yet completely processed.
- **Publish Release (id=2):**
  The client (192.168.142.4) responds to the broker's acknowledgment by sending a Publish Release message with message ID 2, indicating that the message can now be processed and delivered.
- **Publish Complete (id=2):**
  The broker (91.121.93.94) confirms that the message processing is complete by sending a Publish Complete message with message ID 2 back to the client, finalizing the QoS 2 message flow.

mqtt

No.	Time	Source	Destination	Protocol	Length	Info	Source Port
3	14:10:03.670927	192.168.208.102	91.121.93.94	MQTT	56	Ping Request	60962
4	14:10:03.847416	91.121.93.94	192.168.208.102	MQTT	56	Ping Response	mqtt
8	14:10:08.909333	192.168.208.102	91.121.93.94	MQTT	56	Ping Request	60962
9	14:10:08.927878	91.121.93.94	192.168.208.102	MQTT	70	Publish Message (id=4) [oranges]	mqtt
13	14:10:09.087534	91.121.93.94	192.168.208.102	MQTT	56	Ping Response	mqtt
14	14:10:09.087632	192.168.208.102	91.121.93.94	MQTT	58	Publish Received (id=4)	60962
15	14:10:09.247864	91.121.93.94	192.168.208.102	MQTT	58	Publish Release (id=4)	mqtt
16	14:10:09.248158	192.168.208.102	91.121.93.94	MQTT	58	Publish Complete (id=4)	60962
23	14:10:14.289209	192.168.208.102	91.121.93.94	MQTT	56	Ping Request	60962
25	14:10:14.487597	91.121.93.94	192.168.208.102	MQTT	56	Ping Response	mqtt

Figure 24: Subscriber QoS 2 - Wireshark

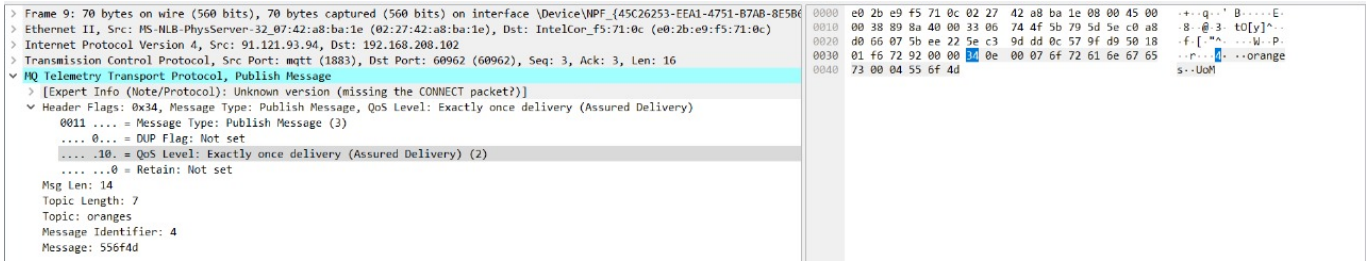


Figure 25: Subscriber QoS 2 - Publish Message

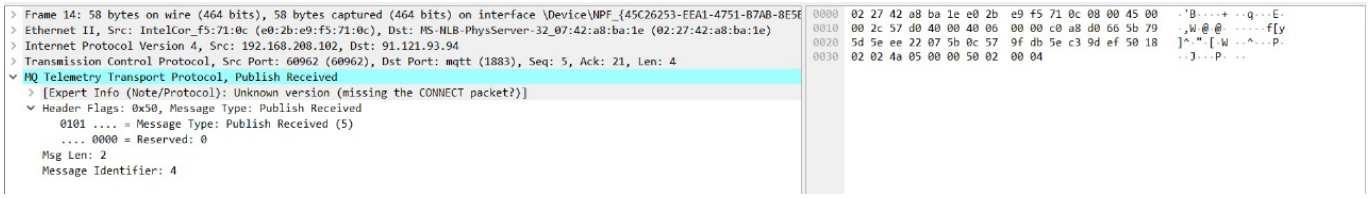


Figure 26: Subscriber QoS 2 - Publish Received

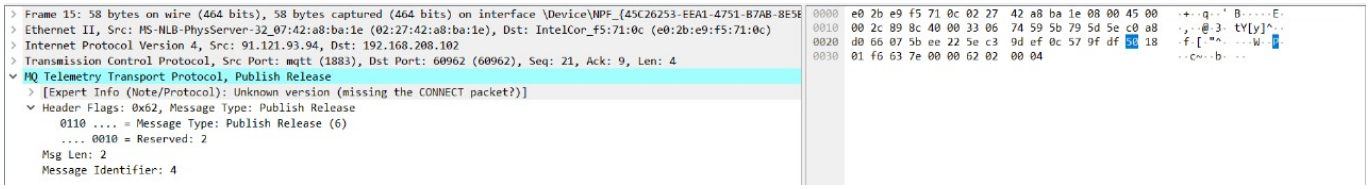


Figure 27: Subscriber QoS 2 - Publish Release

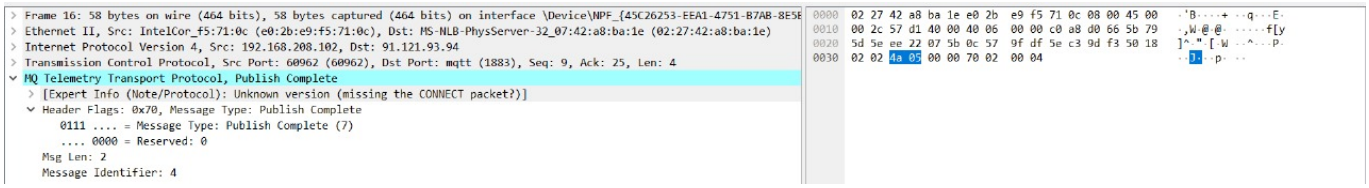


Figure 28: Subscriber QoS 2 - Publish Complete

- **Ping Response:**
  The broker (91.121.93.94) sends a Ping Response to the client (192.168.208.102), confirming the MQTT connection is still active.
- **Publish Message (id=4):**
  The client (192.168.208.102) sends a Publish message containing "oranges" with message ID 4 to the broker (91.121.93.94) under QoS 2.
- **Publish Received (id=4):**
  The broker (91.121.93.94) acknowledges the receipt of the Publish message by sending a Publish Received message with message ID 4 back to the client.
- **Publish Release (id=4):**
  The client (192.168.208.102) sends a Publish Release message with message ID 4 to the broker, indicating the message can be processed.
- **Publish Complete (id=4):**
  The broker (91.121.93.94) confirms the message processing is complete by sending a Publish Complete message with message ID 4 back to the client.

5 Codes

5.1 Publisher

```python
from paho.mqtt import client as mqtt_client
import paho.mqtt.client as mqtt
import time

# Callback when the client connects to the MQTT broker
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to MQTT broker\n")
    else:
        print("Connection failed with code {rc}")


# Create an MQTT client instance
client = mqtt.Client(mqtt_client.CallbackAPIVersion.VERSION1,"PythonPub")

# Set the callback function
client.on_connect = on_connect

broker_address = "test.mosquitto.org"  # broker's address
broker_port = 1883
keepalive = 5
qos = 1
publish_topic = "oranges"

# Connect to the MQTT broker
client.connect(broker_address, broker_port, keepalive)

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

# Publish loop

try:
    while True:
        # Publish a message to the send topic

        value = input('Enter the message: ')
        client.publish(publish_topic,value,qos)
        print(f"Published message '{value}' to topic '{publish_topic}'\n")

        # Wait for a moment to simulate some client activity
        time.sleep(6)

except KeyboardInterrupt:
    # Disconnect from the MQTT broker
    pass
client.loop_stop()
client.disconnect()

print("Disconnected from the MQTT broker")
```

Listing 1: Publisher Code

5.2 Subscriber

```python
from paho.mqtt import client as mqtt_client
import paho.mqtt.client as mqtt
import time

# Callback when the client connects to the MQTT broker
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to MQTT broker")
        client.subscribe(subscribe_topic, qos)  # Subscribe to the receive topic
    else:
        print("Connection failed with code {rc}")

# Callback when a message is received from the subscribed topic
def on_message(client, userdata, msg):
    print("Message received " + "on " + subscribe_topic + ": " +
       str(msg.payload.decode("utf-8")))

# Create an MQTT client instance
client = mqtt.Client(mqtt_client.CallbackAPIVersion.VERSION1, "PythonSub")

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

# Connect to the MQTT broker
broker_address = "test.mosquitto.org"  # broker's address
broker_port = 1883
keepalive = 5
qos = 1

subscribe_topic = "oranges"
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(6)

except KeyboardInterrupt:
    # Disconnect from the MQTT broker
    pass
client.loop_stop()
client.disconnect()

print("Disconnected from the MQTT broker")
```

Listing 2: Subscriber Code

6 Homework - Smart Plant Watering System

This project implements a Smart Plant Watering System using MQTT, which automates the process of watering a plant based on soil moisture and water level readings. The system consists of a controller, moisture sensor, water level sensor, and a water pump, all communicating via MQTT. The controller makes decisions to start or stop the water pump based on real-time data from the sensors, ensuring the plant is adequately watered while conserving resources. This project demonstrates the effective use of MQTT in IoT-based automation.

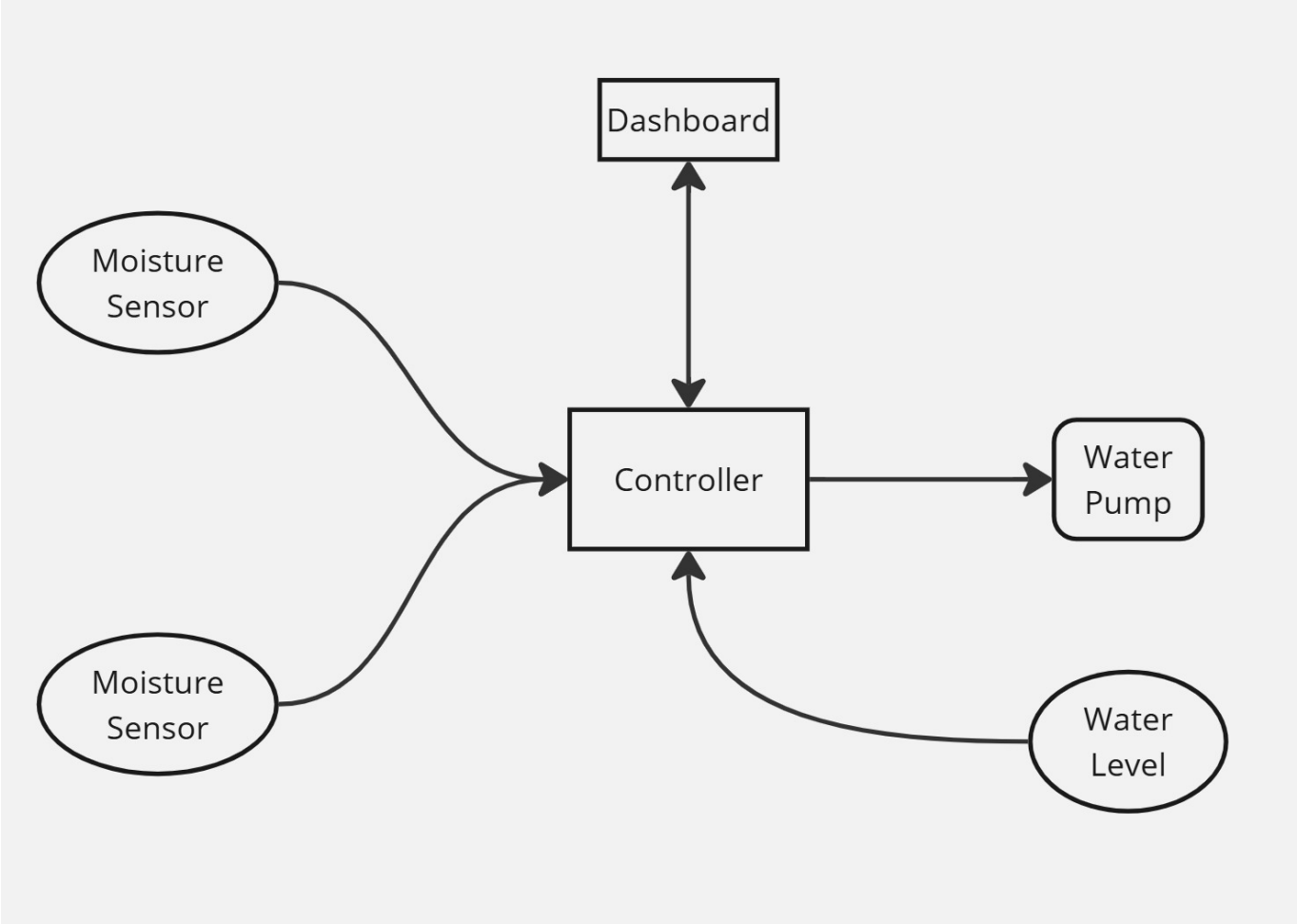


Figure 29: System Architecture

6.1 Controller Code

The controller code serves as the central unit of the Smart Plant Watering System. It is responsible for subscribing to MQTT topics to receive data from the soil moisture sensor and water level sensor. The controller then processes this data to decide whether to activate or deactivate the water pump. Additionally, the controller publishes status updates and commands to relevant MQTT topics, allowing other components to respond accordingly. The primary goal of this code is to ensure that the plant receives adequate water without depleting the water supply.

```
from paho.mqtt import client as mqtt_client
import paho.mqtt.client as mqtt
import time
import os
from dotenv import load_dotenv

# Callback when the client connects to the MQTT broker
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to MQTT broker\n")
        client.subscribe("Sensor_1", qos)
        client.subscribe("Water_level", qos)
    else:
        print("Connection failed with code {rc}")

def on_message(client, userdata, msg):
    global sensor_data
    global water_level
    topic = msg.topic
    if topic == "Sensor_1":
        sensor_data = str(msg.payload.decode("utf-8"))
    elif topic == "Water_level":
        water_level = str(msg.payload.decode("utf-8"))
    else:
        print("Unknown topic")

# Create an MQTT client instance
client = mqtt.Client(client_id="controller", clean_session=True)

# Set the callback function
client.on_connect = on_connect
client.on_message = on_message

# Using a Mosquitto broker hosted on a server
broker_address = "137.184.9.146"  # broker's address
broker_port = 1883
keepalive = 5
qos = 1

publish_topic = "RPS_client_to_server"
subscribe_topic = "RPS_server_to_client"

load_dotenv()  # Load the environment variables
username = os.environ['MQTT_USERNAME']
password = os.environ['MQTT_PASSWORD']

client.username_pw_set(username, password)

# Connect to the MQTT broker
client.connect(broker_address, broker_port, keepalive)

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

```
54
55 # Publish loop
56 time.sleep(2)  # Wait for the connection to establish
57 sensor_data = "High"
58 water_level = "High"
59
60 try:
61     while True:
62         if sensor_data == "Low":
63             if water_level == "Low":
64                 print("Water level is low, please refill the tank")
65                 client.publish("Dashboard", "Water level is low, please refill
                       the tank", qos)
66             else:
67                  print("Water level is normal. Water Pump Started")
68                  client.publish(publish_topic, "Start", qos)
69         else:
70             print("Soil moisture is sufficient")
71             client.publish(publish_topic, "Stop", qos)
72         time.sleep(3)  # Wait for 3 seconds
73
74 except KeyboardInterrupt:
75     # Disconnect from the MQTT broker
76     pass
77 client.loop_stop()
78 client.disconnect()
79
80 print("Disconnected from the MQTT broker")
```

Listing 3: Controller Code

6.2 Moisture Sensor Code

The moisture sensor code reads the soil moisture levels and communicates this data to the MQTT broker. This code is designed to operate continuously, monitoring the soil's moisture content and reporting it in real time. Based on the moisture level, the controller decides whether the plant requires watering. The simplicity of this code lies in its focused task—accurately reporting soil moisture to enable efficient water management.

```
from paho.mqtt import client as mqtt_client
import paho.mqtt.client as mqtt
import time
import os
from dotenv import load_dotenv

# Callback when the client connects to the MQTT broker
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to MQTT broker\n")
        client.subscribe("Pump", qos)
    else:
        print("Connection failed with code {rc}")

def on_message(client, userdata, msg):
    message = str(msg.payload.decode("utf-8"))
    global pump
    if message == "Start":
        pump = "start"
    elif message == "Stop":
        pump = "stop"

# Create an MQTT client instance
client = mqtt.Client(client_id="pump", clean_session=True)

# Set the callback function
client.on_connect = on_connect
client.on_message = on_message

# Using a Mosquitto broker hosted on a server
broker_address = "137.184.9.146"  # broker's address
broker_port = 1883
keepalive = 5
qos = 1

load_dotenv()  # Load the environment variables
username = os.environ['MQTT_USERNAME']
password = os.environ['MQTT_PASSWORD']
client.username_pw_set(username, password)

# Connect to the MQTT broker
client.connect(broker_address, broker_port, keepalive)

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

# Publish loop
time.sleep(2)  # Wait for the connection to establish

def pump_start():
    print("Water Pump Started")

def pump_stop():
    print("Water Pump Stopped")

```

```
pump = "stop"
try:
    while True:
        time.sleep(3)
        if pump == "start":
            pump_start()
            pump = "stop"  # Reset the pump status in case of controller failure
        elif pump == "stop":
            pump_stop()

except KeyboardInterrupt:
    # Disconnect from the MQTT broker
    pass
client.loop_stop()
client.disconnect()

print("Disconnected from the MQTT broker")
```

Listing 4: Moisture Sensor Code

6.3 Water Level Sensor Code

The water level sensor code is responsible for monitoring the water level in the tank. It continuously checks the tank's water level and publishes this information to the MQTT broker. The controller uses this data to determine if there is sufficient water to activate the water pump. This code is critical in ensuring that the pump does not run dry, thereby preventing potential damage and ensuring the system's reliability.

```
from paho.mqtt import client as mqtt_client
import paho.mqtt.client as mqtt
import time
import os
from dotenv import load_dotenv

# Callback when the client connects to the MQTT broker
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to MQTT broker\n")
        client.subscribe("Water_level", qos)
    else:
        print("Connection failed with code {rc}")

def on_message(client, userdata, msg):
    water_level = str(msg.payload.decode("utf-8"))
    global level
    level = water_level

# Create an MQTT client instance
client = mqtt.Client(client_id="water_level_sensor", clean_session=True)

# Set the callback function
client.on_connect = on_connect
client.on_message = on_message

# Using a Mosquitto broker hosted on a server
broker_address = "137.184.9.146"  # broker's address
broker_port = 1883
keepalive = 5
qos = 1

load_dotenv()  # Load the environment variables
username = os.environ['MQTT_USERNAME']
password = os.environ['MQTT_PASSWORD']
client.username_pw_set(username, password)

# Connect to the MQTT broker
client.connect(broker_address, broker_port, keepalive)

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

# Publish loop
time.sleep(2)  # Wait for the connection to establish
level = "High"

try:
    while True:
        client.publish("Water_level", level, qos)
        time.sleep(3)  # Publish water level every 3 seconds

except KeyboardInterrupt:
    # Disconnect from the MQTT broker
    pass
```

```
client.loop_stop()
client.disconnect()

print("Disconnected from the MQTT broker")
```

Listing 5: Water Level Sensor Code

6.4 Water Pump Code

The water pump code controls the activation and deactivation of the water pump based on commands received from the controller via MQTT. When the controller sends a "Start" command, the pump is activated to water the plant. Conversely, a "Stop" command deactivates the pump. This code ensures that the pump operates only when necessary, conserving water and power while maintaining the health of the plant.

```
from paho.mqtt import client as mqtt_client
import paho.mqtt.client as mqtt
import time
import os
from dotenv import load_dotenv

# Callback when the client connects to the MQTT broker
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to MQTT broker\n")
        client.subscribe("Pump", qos)
    else:
        print("Connection failed with code {rc}")

def on_message(client, userdata, msg):
    pump_command = str(msg.payload.decode("utf-8"))
    global pump
    pump = pump_command

# Create an MQTT client instance
client = mqtt.Client(client_id="water_pump", clean_session=True)

# Set the callback function
client.on_connect = on_connect
client.on_message = on_message

# Using a Mosquitto broker hosted on a server
broker_address = "137.184.9.146"  # broker's address
broker_port = 1883
keepalive = 5
qos = 1

load_dotenv()  # Load the environment variables
username = os.environ['MQTT_USERNAME']
password = os.environ['MQTT_PASSWORD']
client.username_pw_set(username, password)

# Connect to the MQTT broker
client.connect(broker_address, broker_port, keepalive)

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

# Publish loop
time.sleep(2)  # Wait for the connection to establish
pump = "stop"

try:
    while True:
       if pump == "start":
            print("Water Pump Started")
            # Code to activate the water pump hardware
            pump = "stop"  # Reset pump status
       elif pump == "stop":
            print("Water Pump Stopped")
```

```
            # Code to deactivate the water pump hardware
        time.sleep(3)  # Loop every 3 seconds

except KeyboardInterrupt:
    # Disconnect from the MQTT broker
    pass
client.loop_stop()
client.disconnect()

print("Disconnected from the MQTT broker")
```

Listing 6: Water Pump Code

6.5 Outputs

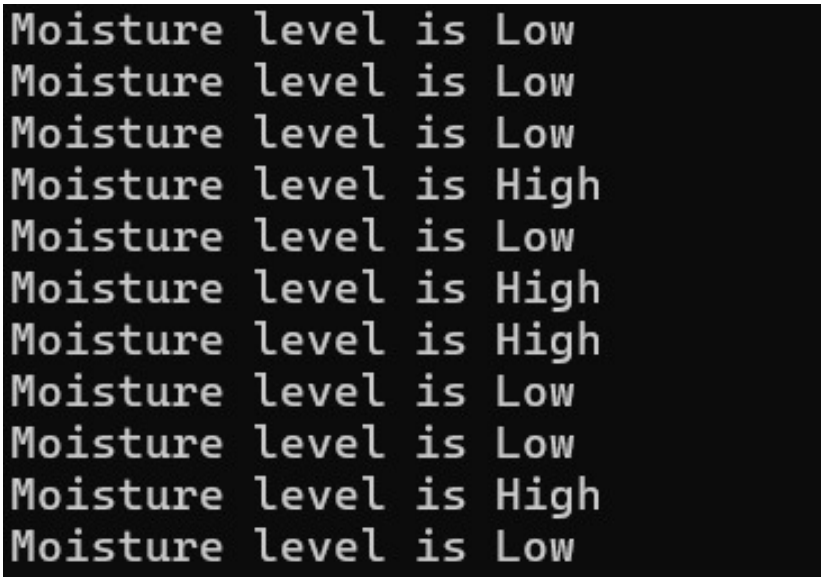


Figure 30: Moisture Sensor

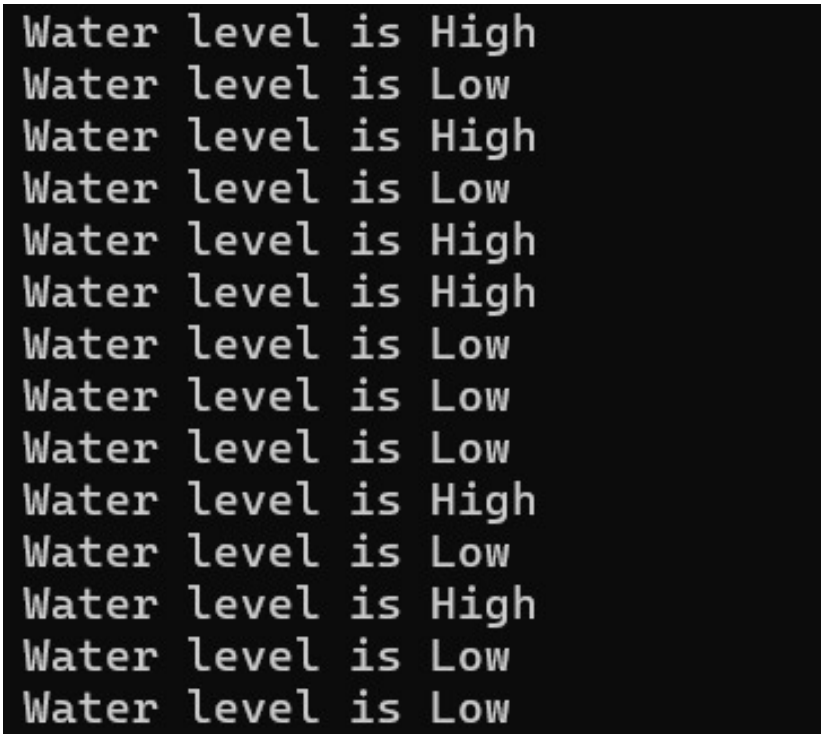


Figure 31: Water Level

Figure 32: Water Pump

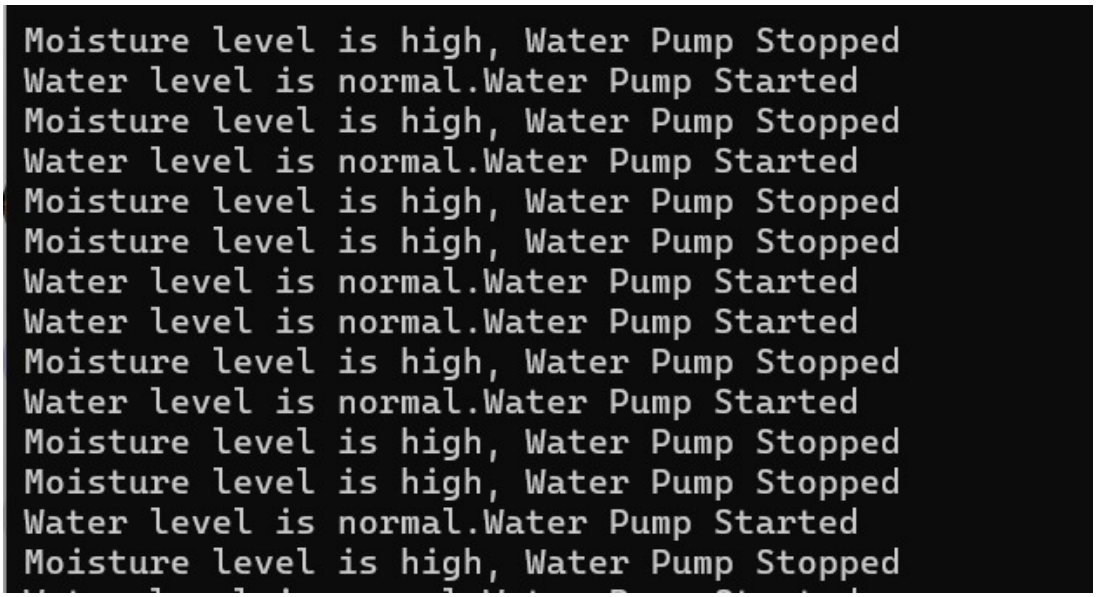


Figure 33: Controller