Department of Electronic and Telecommunication Engineering
University of Moratuwa
Sri Lanka

EN3251 - Internet of Things

Laboratory Exercise 2
Information transfer with MQTT and HTTP using JSON

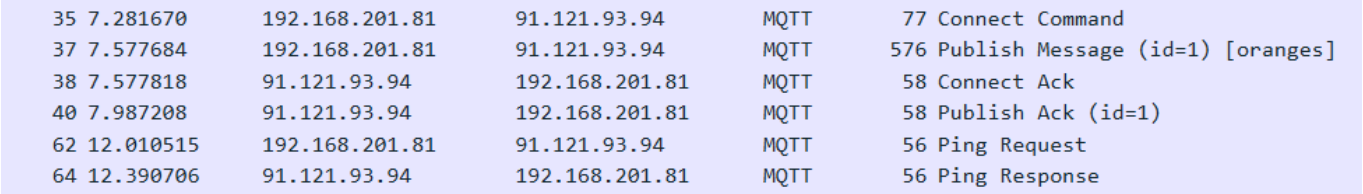
Group 9

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
2024.09.14

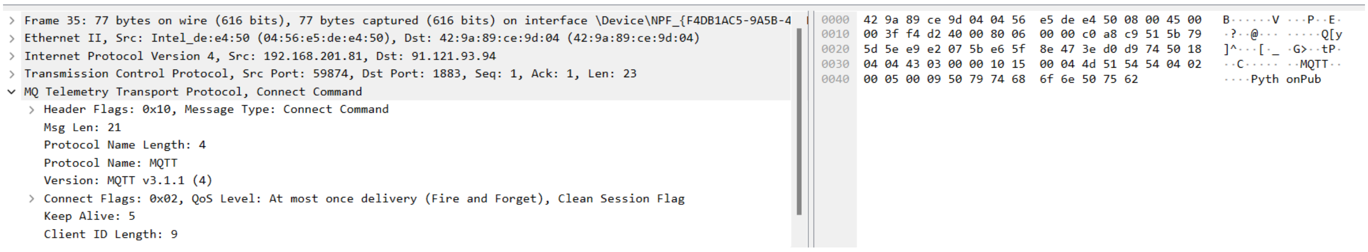
Contents

1 MQTT Publisher



```
35 7.281670    192.168.201.81    91.121.93.94      MQTT     77 Connect Command
37 7.577684    192.168.201.81    91.121.93.94      MQTT    576 Publish Message (id=1) [oranges]
38 7.577818    91.121.93.94      192.168.201.81    MQTT     58 Connect Ack
40 7.987208    91.121.93.94      192.168.201.81    MQTT     58 Publish Ack (id=1)
62 12.010515   192.168.201.81    91.121.93.94      MQTT     56 Ping Request
64 12.390706   91.121.93.94      192.168.201.81    MQTT     56 Ping Response
```

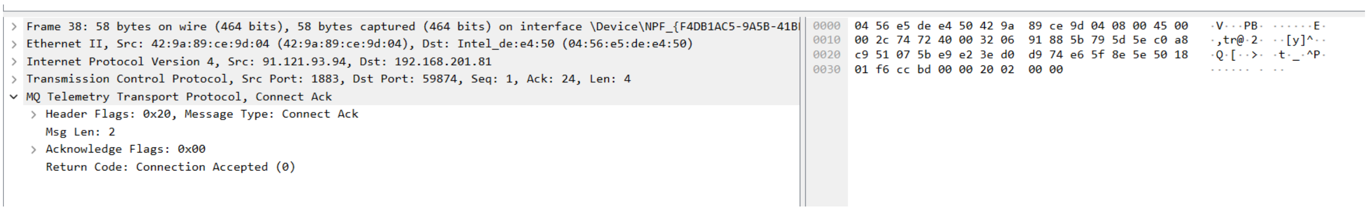
Figure 1: Publisher Wireshark



```
> Frame 35: 77 bytes on wire (616 bits), 77 bytes captured (616 bits) on interface \Device\NPF_{F4DB1AC5-9A5B-4
> Ethernet II, Src: Intel_de:e4:50 (04:56:e5:de:e4:50), Dst: 42:9a:89:ce:9d:04 (42:9a:89:ce:9d:04)
> Internet Protocol Version 4, Src: 192.168.201.81, Dst: 91.121.93.94
> Transmission Control Protocol, Src Port: 59874, Dst Port: 1883, Seq: 1, Ack: 1, Len: 23
v MQ Telemetry Transport Protocol, Connect Command
  > Header Flags: 0x10, Message Type: Connect Command
    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
```

```
0000  42 9a 89 ce 9d 04 04 56  e5 de e4 50 08 00 45 00   B······V ···P··E·
0010  00 3f f4 d2 40 00 80 06  00 00 c0 a8 c9 51 5b 79   ·?··@··· ·····Q[y
0020  5d 5e e9 e2 07 5b e6 5f  8e 47 3e d0 d9 74 50 18   ]^···[·_ ·G>··tP·
0030  04 04 43 03 00 00 10 15  00 04 4d 51 54 54 04 02   ··C····· ··MQTT··
0040  00 05 00 09 50 79 74 68  6f 6e 50 75 62            ····Pyth onPub
```

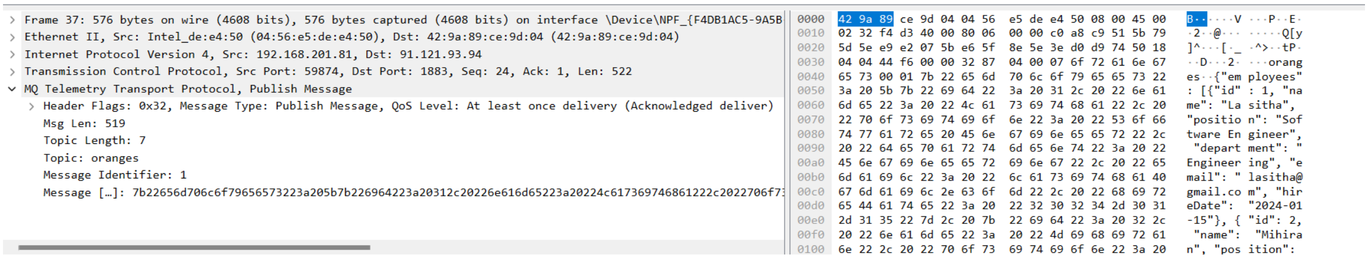
Figure 2: Publisher Connect Command



```
> Frame 38: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface \Device\NPF_{F4DB1AC5-9A5B-41B
> Ethernet II, Src: 42:9a:89:ce:9d:04 (42:9a:89:ce:9d:04), Dst: Intel_de:e4:50 (04:56:e5:de:e4:50)
> Internet Protocol Version 4, Src: 91.121.93.94, Dst: 192.168.201.81
> Transmission Control Protocol, Src Port: 1883, Dst Port: 59874, Seq: 1, Ack: 24, Len: 4
v MQ Telemetry Transport Protocol, Connect Ack
  > Header Flags: 0x20, Message Type: Connect Ack
    Msg Len: 2
  > Acknowledge Flags: 0x00
    Return Code: Connection Accepted (0)
```

```
0000  04 56 e5 de e4 50 42 9a  89 ce 9d 04 08 00 45 00   ·V···PB· ······E·
0010  00 2c 74 72 40 00 32 06  91 88 5b 79 5d 5e c0 a8   ·,tr@·2· ··[y]^··
0020  c9 51 07 5b e9 e2 3e d0  d9 74 e6 5f 8e 5e 50 18   ·Q·[··>· ·t·_·^P·
0030  01 f6 cc bd 00 00 20 02  00 00                     ······ · ··
```

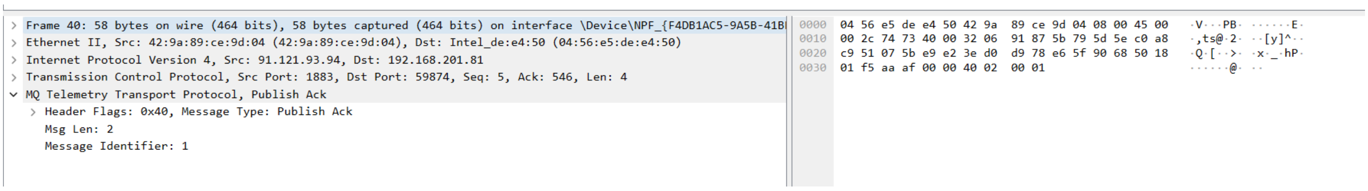
Figure 3: Publisher Connect ACK



```
> Frame 37: 576 bytes on wire (4608 bits), 576 bytes captured (4608 bits) on interface \Device\NPF_{F4DB1AC5-9A5B
> Ethernet II, Src: Intel_de:e4:50 (04:56:e5:de:e4:50), Dst: 42:9a:89:ce:9d:04 (42:9a:89:ce:9d:04)
> Internet Protocol Version 4, Src: 192.168.201.81, Dst: 91.121.93.94
> Transmission Control Protocol, Src Port: 59874, Dst Port: 1883, Seq: 24, Ack: 1, Len: 522
v MQ Telemetry Transport Protocol, Publish Message
  > Header Flags: 0x32, Message Type: Publish Message, QoS Level: At least once delivery (Acknowledged deliver)
    Msg Len: 519
    Topic Length: 7
    Topic: oranges
    Message Identifier: 1
    Message […]: 7b22656d706c6f79656573223a205b7b226964223a20312c20226e616d65223a20224c617369746861222c2022706f73
```

```
0000  42 9a 89 ce 9d 04 04 56  e5 de e4 50 08 00 45 00   B······V ···P··E·
0010  02 32 f4 d3 40 00 80 06  00 00 c0 a8 c9 51 5b 79   ·2··@··· ·····Q[y
0020  5d 5e e9 e2 07 5b e6 5f  8e 5e 3e d0 d9 74 50 18   ]^···[·_ ·^>··tP·
0030  04 04 44 f6 00 00 32 87  04 00 07 6f 72 61 6e 67   ··D···2· ···orang
0040  65 73 00 01 7b 22 65 6d  70 6c 6f 79 65 65 73 22   es··{"em ployees"
0050  3a 20 5b 7b 22 69 64 22  3a 20 31 2c 20 22 6e 61   : [{"id" : 1, "na
0060  6d 65 22 3a 20 22 4c 61  73 69 74 68 61 22 2c 20   me": "La sitha", 
0070  22 70 6f 73 69 74 69 6f  6e 22 3a 20 22 53 6f 66   "positio n": "Sof
0080  74 77 61 72 65 20 45 6e  67 69 6e 65 65 72 22 2c   tware En gineer",
0090  20 22 64 65 70 61 72 74  6d 65 6e 74 22 3a 20 22    "depart ment": "
00a0  45 6e 67 69 6e 65 65 72  69 6e 67 22 2c 20 22 65   Engineer ing", "e
00b0  6d 61 69 6c 22 3a 20 22  6c 61 73 69 74 68 61 40   mail": " lasitha@
00c0  67 6d 61 69 6c 2e 63 6f  6d 22 2c 20 22 68 69 72   gmail.co m", "hir
00d0  65 44 61 74 65 22 3a 20  22 32 30 32 34 2d 30 31   eDate":  "2024-01
00e0  2d 31 35 22 7d 2c 20 7b  22 69 64 22 3a 20 32 2c   -15"}, { "id": 2,
00f0  20 22 6e 61 6d 65 22 3a  20 22 4d 69 68 69 72 61    "name":  "Mihira
0100  6e 22 2c 20 22 70 6f 73  69 74 69 6f 6e 22 3a 20   n", "pos ition": 
```

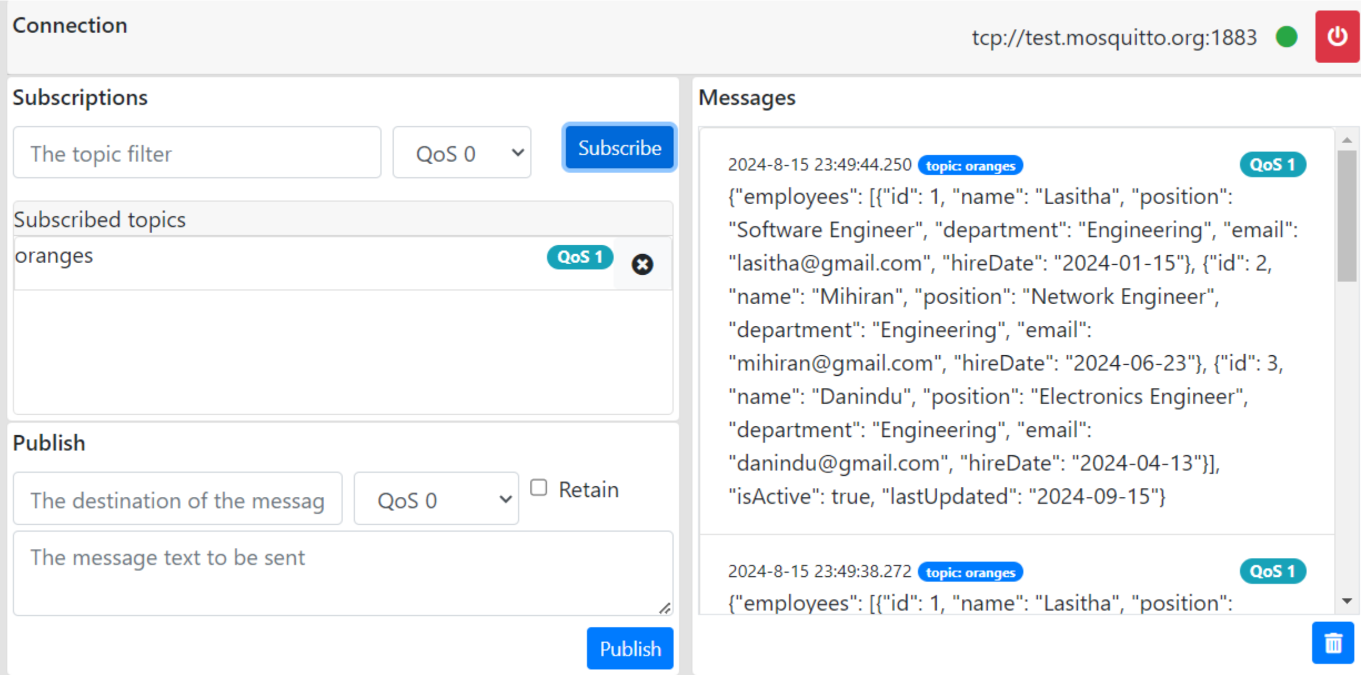
Figure 4: Publish Message



```
> Frame 40: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface \Device\NPF_{F4DB1AC5-9A5B-41B
> Ethernet II, Src: 42:9a:89:ce:9d:04 (42:9a:89:ce:9d:04), Dst: Intel_de:e4:50 (04:56:e5:de:e4:50)
> Internet Protocol Version 4, Src: 91.121.93.94, Dst: 192.168.201.81
> Transmission Control Protocol, Src Port: 1883, Dst Port: 59874, Seq: 5, Ack: 546, Len: 4
v MQ Telemetry Transport Protocol, Publish Ack
  > Header Flags: 0x40, Message Type: Publish Ack
    Msg Len: 2
    Message Identifier: 1
```

```
0000  04 56 e5 de e4 50 42 9a  89 ce 9d 04 08 00 45 00   ·V···PB· ······E·
0010  00 2c 74 73 40 00 32 06  91 87 5b 79 5d 5e c0 a8   ·,ts@·2· ··[y]^··
0020  c9 51 07 5b e9 e2 3e d0  d9 78 e6 5f 90 68 50 18   ·Q·[··>· ·x·_·hP·
0030  01 f5 aa af 00 00 40 02  00 01                     ······@· ··
```

Figure 5: Publish ACK



tcp://test.mosquitto.org:1883

Subscribed topics

oranges QoS 1

2024-8-15 23:49:44.250 topic: oranges QoS 1

{"employees": [{"id": 1, "name": "Lasitha", "position": "Software Engineer", "department": "Engineering", "email": "lasitha@gmail.com", "hireDate": "2024-01-15"}, {"id": 2, "name": "Mihiran", "position": "Network Engineer", "department": "Engineering", "email": "mihiran@gmail.com", "hireDate": "2024-06-23"}, {"id": 3, "name": "Danindu", "position": "Electronics Engineer", "department": "Engineering", "email": "danindu@gmail.com", "hireDate": "2024-04-13"}], "isActive": true, "lastUpdated": "2024-09-15"}

2024-8-15 23:49:38.272 topic: oranges QoS 1

{"employees": [{"id": 1, "name": "Lasitha", "position":

Figure 6: Test Client

MQTT Packet Structure

MQTT packets, including PUBLISH messages, are typically small. Each PUBLISH message contains:

- **Topic Name**: “oranges”
- **Message Payload**: The payload is formatted in JSON.
- **QoS Level**: The Quality of Service level is set to 1, which ensures that the message is delivered at least once.

Payload Inspection

The JSON payload appears as part of the MQTT packet’s payload. JSON data is observed as a string within the PUBLISH message.

Message Size

The size of MQTT messages is small compared to typical HTTP messages. This compact size is a significant advantage of MQTT, making it suitable for scenarios with limited bandwidth or where efficiency is critical.

Network Traffic

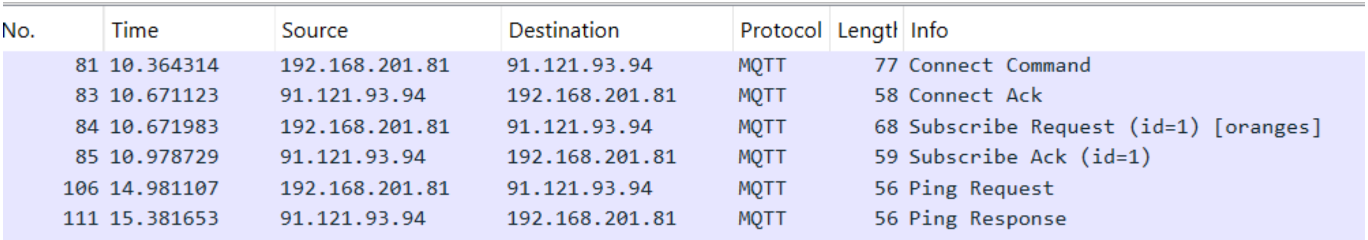
When the script runs, multiple PUBLISH messages are observed. Each message is timestamped, allowing tracking of the frequency of message publication and monitoring of network traffic over time.

Code

```
from paho.mqtt import client as mqtt_client
import paho.mqtt.client as mqtt
import json
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

read_file_name = "data.json"
with open(read_file_name) as json_file:
            sensor_out= json.load(json_file)

print ("sensor_out is a ", type(sensor_out))
data_out=json.dumps(sensor_out)

# Data to send (JSON format)
# data = {
#     "device_id": "sensor_001",
#     "timestamp": "2024-09-12T10:00:00Z",
#     "temperature": 22.5,
#     "humidity": 55
# }

# json_payload = json.dumps(data)

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

        #value = input(’Enter the message: ’)
        client.publish(publish_topic,data_out,qos)
        print(f"Published message ’{data_out}’ 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")
```

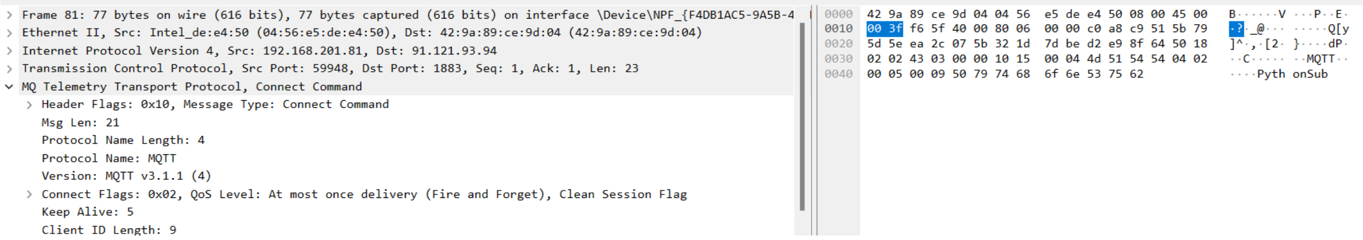
Listing 1: Publisher Code

2 MQTT Subscriber



No.	Time	Source	Destination	Protocol	Lengtl	Info
81	10.364314	192.168.201.81	91.121.93.94	MQTT	77	Connect Command
83	10.671123	91.121.93.94	192.168.201.81	MQTT	58	Connect Ack
84	10.671983	192.168.201.81	91.121.93.94	MQTT	68	Subscribe Request (id=1) [oranges]
85	10.978729	91.121.93.94	192.168.201.81	MQTT	59	Subscribe Ack (id=1)
106	14.981107	192.168.201.81	91.121.93.94	MQTT	56	Ping Request
111	15.381653	91.121.93.94	192.168.201.81	MQTT	56	Ping Response

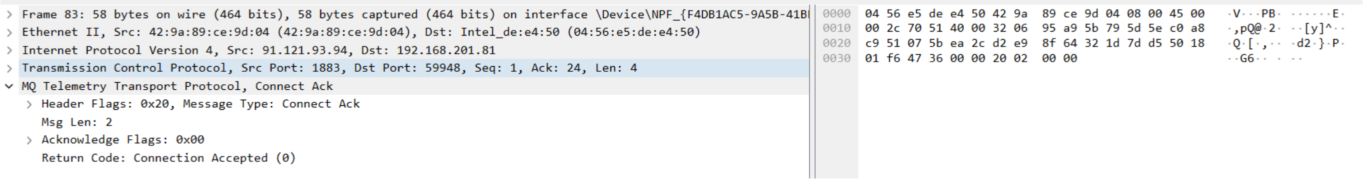
Figure 7: Subscriber Wireshark



```
> Frame 81: 77 bytes on wire (616 bits), 77 bytes captured (616 bits) on interface \Device\NPF_{F4DB1AC5-9A5B-4
> Ethernet II, Src: Intel_de:e4:50 (04:56:e5:de:e4:50), Dst: 42:9a:89:ce:9d:04 (42:9a:89:ce:9d:04)
> Internet Protocol Version 4, Src: 192.168.201.81, Dst: 91.121.93.94
> Transmission Control Protocol, Src Port: 59948, Dst Port: 1883, Seq: 1, Ack: 1, Len: 23
v MQ Telemetry Transport Protocol, Connect Command
  > Header Flags: 0x10, Message Type: Connect Command
    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
```

```
0000  42 9a 89 ce 9d 04 04 56  e5 de e4 50 08 00 45 00   B······V ···P··E·
0010  00 3f f6 5f 40 00 80 06  00 00 c0 a8 c9 51 5b 79   ·?·_@··· ·····Q[y
0020  5d 5e ea 2c 07 5b 32 1d  7d be d2 e9 8f 64 50 18   ]^·,·[2· }····dP·
0030  02 02 43 03 00 00 10 15  00 04 4d 51 54 54 04 02   ··C····· ··MQTT··
0040  00 05 00 09 50 79 74 68  6f 6e 53 75 62            ····Pyth onSub
```

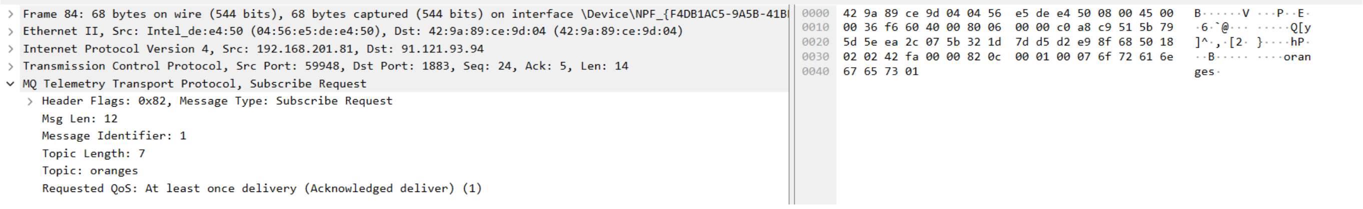
Figure 8: Subscriber Connect Command



```
> Frame 83: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface \Device\NPF_{F4DB1AC5-9A5B-41B
> Ethernet II, Src: 42:9a:89:ce:9d:04 (42:9a:89:ce:9d:04), Dst: Intel_de:e4:50 (04:56:e5:de:e4:50)
> Internet Protocol Version 4, Src: 91.121.93.94, Dst: 192.168.201.81
> Transmission Control Protocol, Src Port: 1883, Dst Port: 59948, Seq: 1, Ack: 24, Len: 4
v MQ Telemetry Transport Protocol, Connect Ack
  > Header Flags: 0x20, Message Type: Connect Ack
    Msg Len: 2
  > Acknowledge Flags: 0x00
    Return Code: Connection Accepted (0)
```

```
0000  04 56 e5 de e4 50 42 9a  89 ce 9d 04 08 00 45 00   ·V···PB· ······E·
0010  00 2c 70 51 40 00 32 06  95 a9 5b 79 5d 5e c0 a8   ·,pQ@·2· ··[y]^··
0020  c9 51 07 5b ea 2c d2 e9  8f 64 32 1d 7d d5 50 18   ·Q·[·,·· ·d2·}·P·
0030  01 f6 47 36 00 00 20 02  00 00                     ··G6·· · ··
```

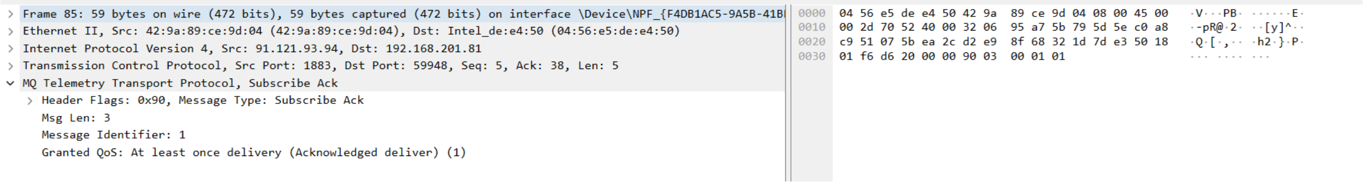
Figure 9: Subscriber Connect ACK



```
> Frame 84: 68 bytes on wire (544 bits), 68 bytes captured (544 bits) on interface \Device\NPF_{F4DB1AC5-9A5B-41B
> Ethernet II, Src: Intel_de:e4:50 (04:56:e5:de:e4:50), Dst: 42:9a:89:ce:9d:04 (42:9a:89:ce:9d:04)
> Internet Protocol Version 4, Src: 192.168.201.81, Dst: 91.121.93.94
> Transmission Control Protocol, Src Port: 59948, Dst Port: 1883, Seq: 24, Ack: 5, Len: 14
v MQ Telemetry Transport Protocol, Subscribe Request
  > Header Flags: 0x82, Message Type: Subscribe Request
    Msg Len: 12
    Message Identifier: 1
    Topic Length: 7
    Topic: oranges
    Requested QoS: At least once delivery (Acknowledged deliver) (1)
```

```
0000  42 9a 89 ce 9d 04 04 56  e5 de e4 50 08 00 45 00   B······V ···P··E·
0010  00 36 f6 60 40 00 80 06  00 00 c0 a8 c9 51 5b 79   ·6·`@··· ·····Q[y
0020  5d 5e ea 2c 07 5b 32 1d  7d d5 d2 e9 8f 68 50 18   ]^·,·[2· }····hP·
0030  02 02 42 fa 00 00 82 0c  00 01 00 07 6f 72 61 6e   ··B····· ····oran
0040  67 65 73 01                                        ges·
```

Figure 10: Subscribe Request



```
> Frame 85: 59 bytes on wire (472 bits), 59 bytes captured (472 bits) on interface \Device\NPF_{F4DB1AC5-9A5B-41B
> Ethernet II, Src: 42:9a:89:ce:9d:04 (42:9a:89:ce:9d:04), Dst: Intel_de:e4:50 (04:56:e5:de:e4:50)
> Internet Protocol Version 4, Src: 91.121.93.94, Dst: 192.168.201.81
> Transmission Control Protocol, Src Port: 1883, Dst Port: 59948, Seq: 5, Ack: 38, Len: 5
v MQ Telemetry Transport Protocol, Subscribe Ack
  > Header Flags: 0x90, Message Type: Subscribe Ack
    Msg Len: 3
    Message Identifier: 1
    Granted QoS: At least once delivery (Acknowledged deliver) (1)
```

```
0000  04 56 e5 de e4 50 42 9a  89 ce 9d 04 08 00 45 00   ·V···PB· ······E·
0010  00 2d 70 52 40 00 32 06  95 a7 5b 79 5d 5e c0 a8   ·-pR@·2· ··[y]^··
0020  c9 51 07 5b ea 2c d2 e9  8f 68 32 1d 7d e3 50 18   ·Q·[·,·· ·h2·}·P·
0030  01 f6 d6 20 00 00 90 03  00 01 01                  ··· ···· ···
```

Figure 11: Subscribe ACK

MQTT Packet Structure

MQTT packets, including SUBSCRIBE messages, are typically small. Each SUBSCRIBE message includes:

- **Topic Name**: “oranges”
- **QoS Level**: The Quality of Service level is set to 1, which ensures that the message is delivered at least once.

Payload Inspection

The JSON payload appears as part of the MQTT packet’s payload. The JSON data received in SUBSCRIBE messages is observed as a string within the PUBLISH message.

Message Size

The size of MQTT SUBSCRIBE messages, is small compared to typical HTTP messages. This compact size is beneficial, especially in scenarios with limited bandwidth or where efficiency is crucial.

Network Traffic

Multiple PUBLISH messages are observed, each related to the subscribed topic. Each message is timestamped, allowing for tracking of message reception frequency and monitoring of network traffic over time.

Code

```python
from paho.mqtt import client as mqtt_client
import paho.mqtt.client as mqtt
import json
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}")
write_file_name = "sensor_received.json"

# 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")))
    recieved = str(msg.payload.decode("utf-8"))
    sensor_in=json.loads(recieved) #convert incoming JSON to object
    print ("recieved is a ", type(recieved))
    print ("\nHumidity = ", sensor_in["humidity"])
    with open(write_file_name, 'w') as json_file:
        json.dump(sensor_in, json_file, indent=4)  # The 'indent' parameter adds pretty formatting
        print("Data has been written to", write_file_name)

# 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 = input ('Enter the topic to subscribe to: ')
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

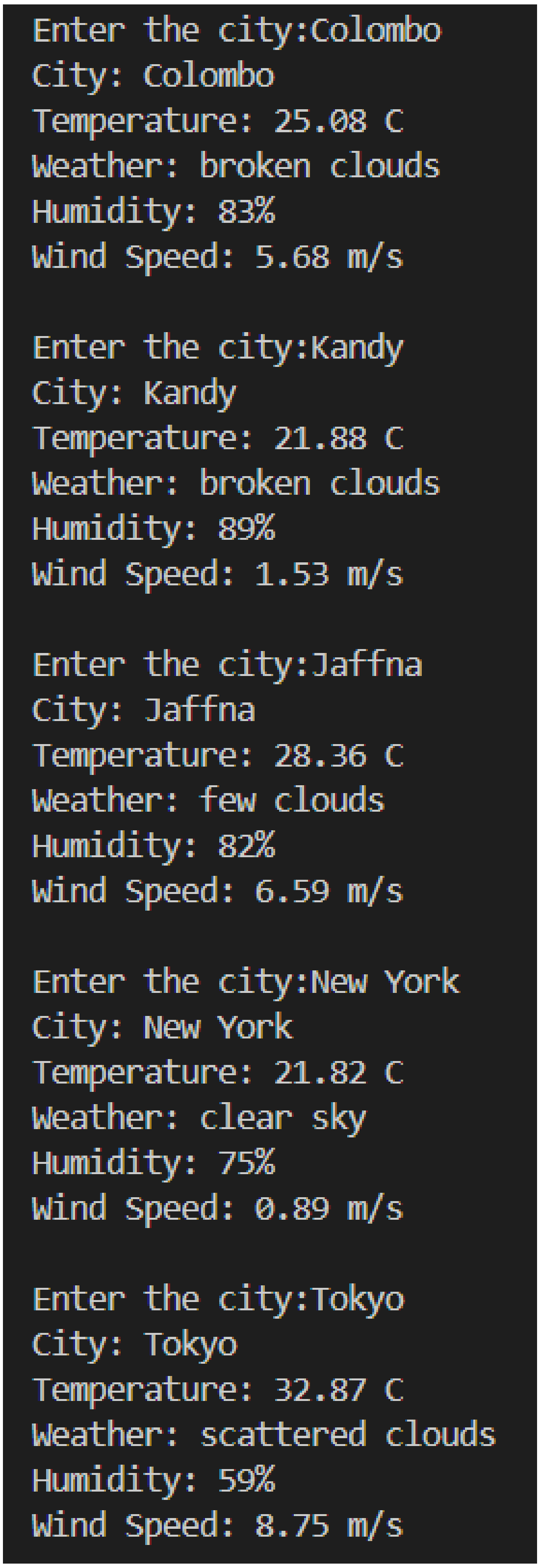
3 Weather Information from OpenWeather

Code

```
import requests

# Your OpenWeather API key (replace with your actual key)
api_key = "d77718b52619d73c26d0cb1f2e3ef25e"

for i in range(5):
    # City for which you want weather data
    city = input("Enter the city:")

    # API URL
    url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid=
    {api_key}&units=metric"

    # Send a GET request to the API
    response = requests.get(url)

    # Parse the JSON response
    weather_data = response.json()

    # Check if the response is valid
    if response.status_code == 200:
        # Print city and weather information
        print(f"City: {weather_data['name']}")
        print(f"Temperature: {weather_data['main']['temp']} C")
        print(f"Weather: {weather_data['weather'][0]['description']}")
        print(f"Humidity: {weather_data['main']['humidity']}%")
        print(f"Wind Speed: {weather_data['wind']['speed']} m/s")
        print( )

    else:
        print(f"Error fetching weather data: {weather_data.get('message',
            'Unknown error')}")
```

Listing 3: Openweather Code

When user enters a city, this constructs a URL to make a GET request to the OpenWeather API, retrieves the weather data in JSON format, and parses this data to extract key details like temperature, weather description, humidity, and wind speed and prints them.



```
Enter the city:Colombo
City: Colombo
Temperature: 25.08 C
Weather: broken clouds
Humidity: 83%
Wind Speed: 5.68 m/s

Enter the city:Kandy
City: Kandy
Temperature: 21.88 C
Weather: broken clouds
Humidity: 89%
Wind Speed: 1.53 m/s

Enter the city:Jaffna
City: Jaffna
Temperature: 28.36 C
Weather: few clouds
Humidity: 82%
Wind Speed: 6.59 m/s

Enter the city:New York
City: New York
Temperature: 21.82 C
Weather: clear sky
Humidity: 75%
Wind Speed: 0.89 m/s

Enter the city:Tokyo
City: Tokyo
Temperature: 32.87 C
Weather: scattered clouds
Humidity: 59%
Wind Speed: 8.75 m/s
```

Figure 12: Weather Information

4 Homework : Node-RED dashboard for displaying data from OpenWeatherMap

4.1 Node-RED Flow Explanation

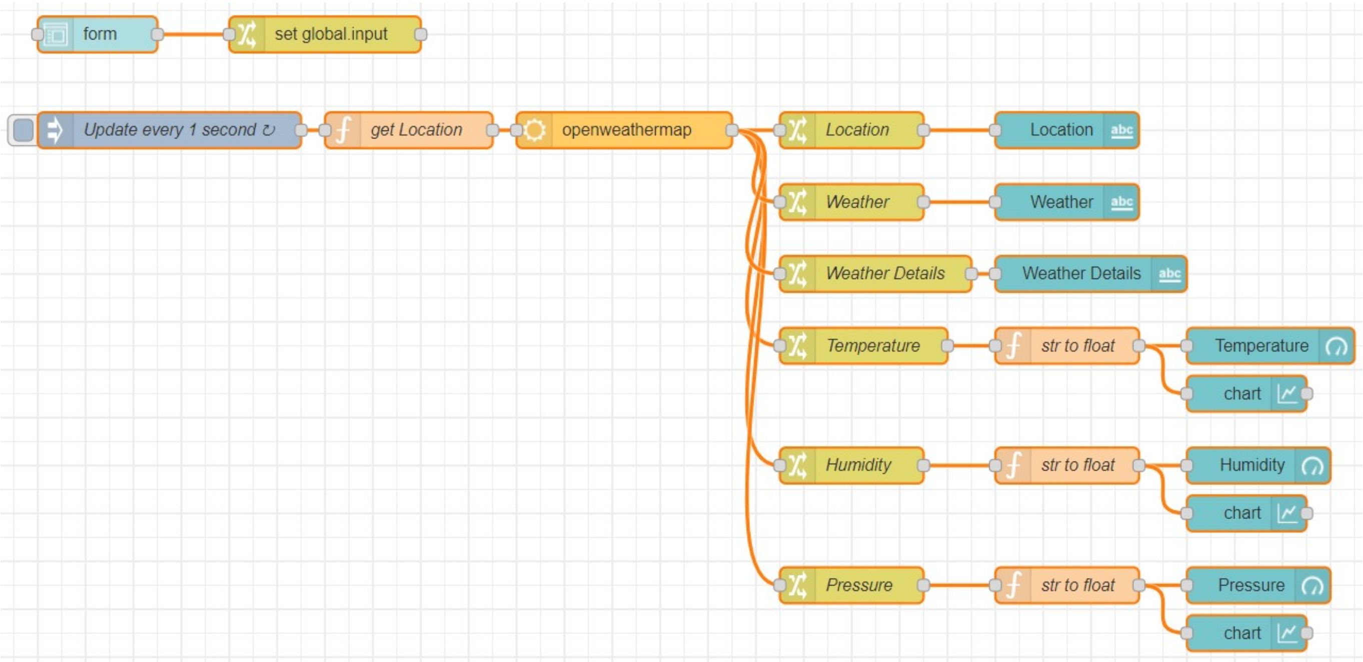
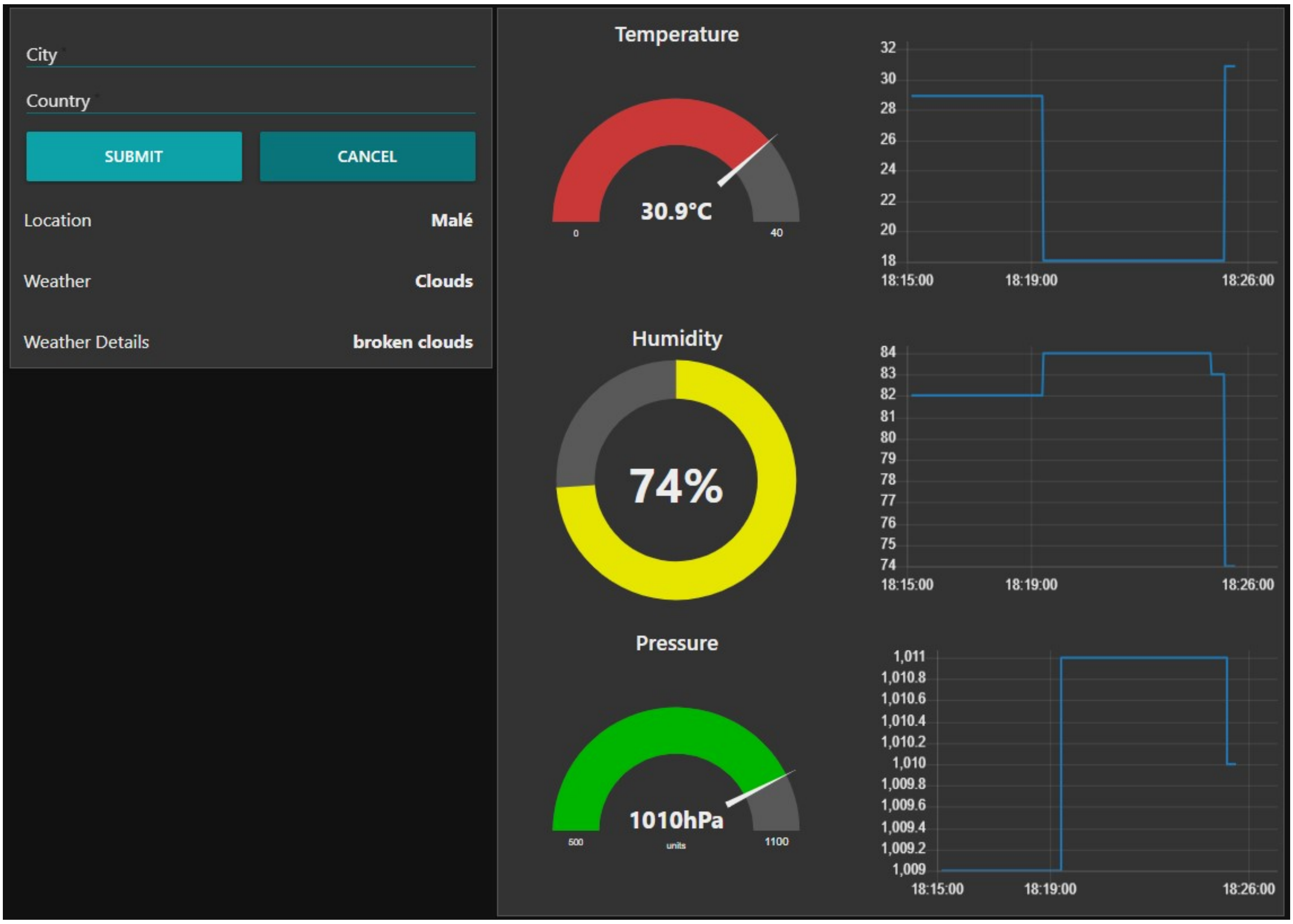


Figure 13: Subscribe Request

1. **User Form Input**:
   - An `ui_form` node captures city and country inputs from the user.
   - A `change` node then sets these inputs as global variables for access throughout the flow.
2. **Weather Data Fetching**:
   - An `inject` node triggers the flow every second.
   - A `function` node named ”get Location” fetches the location from the global variables and passes it to the `openweathermap` node.
   - The `openweathermap` node uses these location details to make API calls to fetch current weather data.
3. **Data Processing**:
   - Several `change` nodes extract specific pieces of data from the API response: temperature, location, weather conditions, humidity, and pressure.
   - These nodes set the extracted data to `msg.payload` for further processing.
4. **Data Conversion**:
   - Multiple `function` nodes labeled ”str to float” convert string data (like temperature and humidity) into float values suitable for graphical representation.
5. **Dashboard Output**:
   - `ui_gauge` nodes create gauge displays for temperature, humidity, and pressure.

- `ui_text` nodes display textual information about the location, weather, and weather details.
- `ui_chart` nodes provide line charts to display the changes in temperature, humidity, and pressure over time based on the repeated API calls.

4.2 Dashboard Explanation



Location Malé

Weather Clouds

Weather Details broken clouds

Figure 14: Subscribe Request

1. **Form for Input**:
   - Allows entry of city and country to fetch specific weather data.
2. **Dynamic Data Display**:
   - **Location**: Displays the current location as fetched by the API.
   - **Weather Overview**: Shows basic weather conditions and detailed descriptions.
   - **Temperature Gauge**: Visual representation of the current temperature.
   - **Humidity Gauge**: Shows current humidity levels.
   - **Pressure Gauge**: Indicates the atmospheric pressure.
   - **Graphs**: Line charts update every second to show trends in temperature, humidity, and pressure based on continuous data fetching.