Team members: Pica Eduard-Ionut, Bilciurescu Elena-Alina, Group 1231EA

Lightweight Protocols for Wearable Sensor Networks in Human Activity Recognition (HAR)

Introduction

This project implements a fitness tracker simulation that uses the MQTT protocol for lightweight communication in wearable sensor networks. It simulates sensor data such as temperature, acceleration, heart rate, steps, and calories burned, which are published and consumed via an MQTT broker. The project is designed for Human Activity Recognition (HAR), where wearable devices capture sensor data to identify user activities (e.g., walking, running, idle).

Objectives

- Implement lightweight protocols for wearable sensor networks using MQTT.
- Simulate fitness tracker data relevant for HAR.
- Provide a detailed explanation of network and computational efficiency.
- Document setup, code, and results.

System Setup

Environment

- Operating System: Debian Linux (on virtual machines).
- Programming Language: Python 3.
- MQTT Broker: Mosquitto.

Installed Packages

The following tools and libraries were installed:

- sudo apt install build-essential git python3 python3-pip -y
- sudo apt install mosquitto mosquitto-clients -y

Python Libraries:

- pip install paho-mqtt
- pip install tkinter

Implementation

1. MQTT Broker Configuration

Steps to Set Up Mosquitto Broker:

1. Install Mosquitto:
sudo apt install mosquitto mosquitto-clients -y

2. Start and Enable Mosquitto Service:

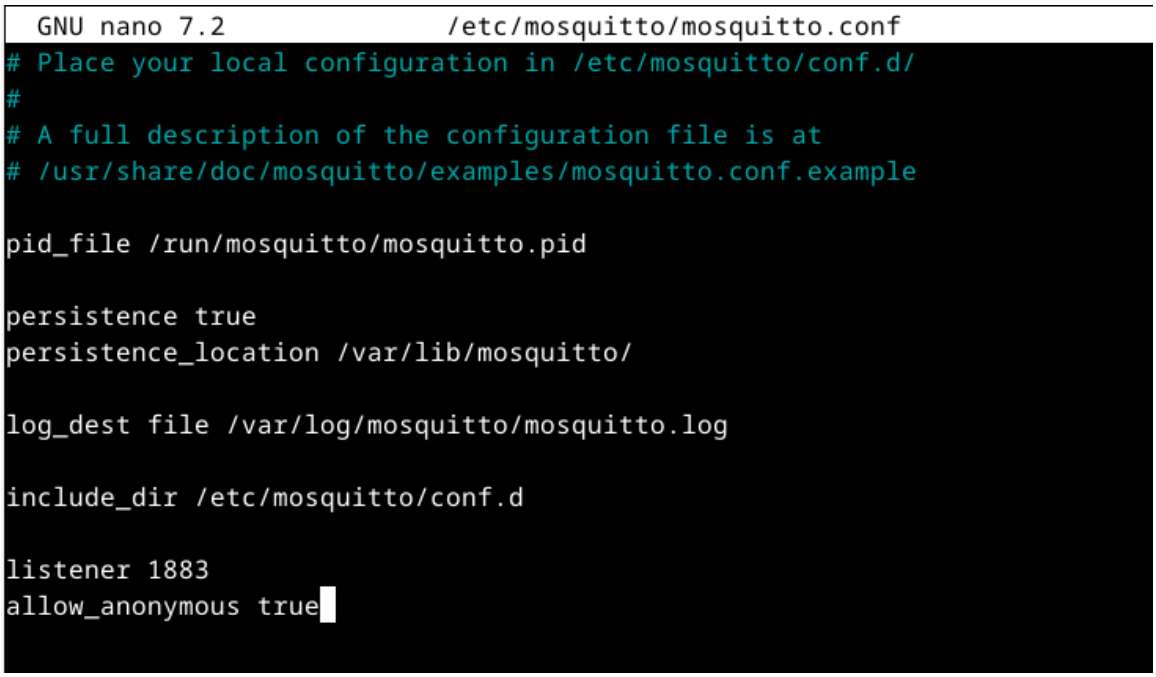
sudo systemctl start mosquitto
sudo systemctl enable mosquito

3. Configure Mosquitto:

Edit the Mosquitto configuration file to allow anonymous connections:
sudo nano /etc/mosquitto/mosquitto.conf

Add the following lines:

listener 1883
allow_anonymous true

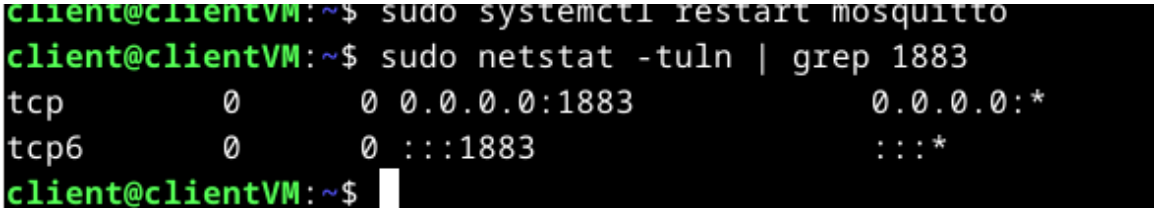


4. Restart Mosquitto:
sudo systemctl restart mosquito

5. Verify Mosquitto:

Check if Mosquitto is listening on port 1883:
sudo netstat -tuln | grep 1883

It should print this:



Testing Results

Verify if the publisher sends messages by subscribing to the topic `wearable/sensor` using:

```
client@serverVM:~$ mosquitto_sub -h localhost -t "test/topic"
```

Make the publisher send a message using

```
client@serverVM:~$ mosquitto_pub -h localhost -t "test/topic" -m "Hello World"
```

Received messages:

```
client@serverVM:~$ mosquitto_sub -h localhost -t "test/topic"
Hello World
```

2. Publisher: Simulating Fitness Tracker

The publisher script simulates fitness tracker data, including:
- Temperature (36.0–37.5 °C).
- Acceleration (0.0–10.0 m/s^2).
- Heart rate (60–180 BPM).
- Steps (100–500 per minute).
- Calories burned (0.5–10.0 kcal).

Code:

server.py

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

def simulate_fitness_tracker():
    # Initialize MQTT client
    client = mqtt.Client()
    client.connect("localhost", 1883, 60)

    print("Publishing fitness tracker data...")

    while True:
        sensor_data = {
            "temperature": round(random.uniform(36.0, 37.5), 2),  # Body temperature in Celsiu
            "acceleration": round(random.uniform(0.0, 10.0), 2),  # Acceleration in m/s^2
            "heart_rate": random.randint(60, 180),  # Heart rate in BPM
            "steps": random.randint(100, 500),  # Steps taken
            "calories_burned": round(random.uniform(0.5, 10.0), 2)  # Calories burned
        }

        client.publish("wearable/sensor", str(sensor_data))
        print(f"Published: {sensor_data}")

        time.sleep(3)

if __name__ == "__main__":
if __name__ == "__main__":
    simulate_fitness_tracker()
```

3. Subscriber: Receiving and Classifying Data

The subscriber script receives MQTT messages and performs activity classification based on sensor data.

Code:

tkinterClient.py

```python
import tkinter as tk
from tkinter import ttk
import paho.mqtt.client as mqtt

# Global variable to store fitness tracker data
sensor_data = {
    "temperature": "N/A",
    "acceleration": "N/A",
    "heart_rate": "N/A",
    "steps": "N/A",
    "calories_burned": "N/A",
    "current_activity": "Idle"
}

# Callback function for when a message is received
def on_message(client, userdata, msg):
    global sensor_data
    try:
        # Decode and convert the string to a dictionary
        sensor_data = eval(msg.payload.decode())
        update_gui()
    except Exception as e:
        print(f"Error parsing message: {e}")

def update_gui():
    temp_label_value.config(text=f"{sensor_data['temperature']} \u00b0C")
    accel_label_value.config(text=f"{sensor_data['acceleration']} m/s\u00b2")
    heart_label_value.config(text=f"{sensor_data['heart_rate']} BPM")
    steps_label_value.config(text=f"{sensor_data['steps']} steps")
    calories_label_value.config(text=f"{sensor_data['calories_burned']} kcal")

    if sensor_data['heart_rate'] > 120 or sensor_data['acceleration'] > 5.0:
        activity_label_value.config(text=f"Running")
    elif sensor_data['heart_rate'] > 80 and sensor_data['heart_rate'] <= 120 or sensor_data['a
```

tkinterClient.py

```python
def update_gui():
    elif sensor_data['heart_rate'] > 80 and sensor_data['heart_rate'] <= 120 or sensor_data['a
        activity_label_value.config(text=f"Walking")
    elif sensor_data['heart_rate'] <= 80 or sensor_data['acceleration'] < 1.0:
        activity_label_value.config(text=f"Idle")
    else:
        activity_label_value.config(text=f"Idle")

# MQTT Setup
def setup_mqtt(broker_ip, topic):
    client = mqtt.Client()
    client.on_message = on_message
    client.connect(broker_ip, 1883, 60)
    client.subscribe(topic)
    client.loop_start()

# GUI SETUP
def setup_gui():
    app = tk.Tk()
    app.title("Fitness Tracker Data Viewer")
    app.geometry("500x350")
    app.configure(bg="#2e3b4e")  # dark background

    # title
    title_label = ttk.Label(
        app,
        text="Fitness Tracker Data",
        font=("Helvetica", 20, "bold"),
        foreground="#FFFFFF",
        background="#2e3b4e"
    )
    title_label.pack(pady=15)
```

tkinterClient.py

```python
def setup_gui():

    # separator
    ttk.Separator(app, orient="horizontal").pack(fill="x", padx=20, pady=10)

    # centered frame for data
    data_frame = tk.Frame(app, bg="#2e3b4e")
    data_frame.pack(fill="both", expand=True, padx=20, pady=10)

    # temperature
    global temp_label_value
    temp_label_value = ttk.Label(data_frame, text="N/A", font=("Helvetica", 14, "bold"), foreg
    ttk.Label(data_frame, text="Temperature:", font=("Helvetica", 14), foreground="#f4b41a", b
    temp_label_value.grid(row=0, column=1, sticky="w", padx=10)

    # acceleration
    global accel_label_value
    accel_label_value = ttk.Label(data_frame, text="N/A", font=("Helvetica", 14, "bold"), fore
    ttk.Label(data_frame, text="Acceleration:", font=("Helvetica", 14), foreground="#f4b41a",
    accel_label_value.grid(row=1, column=1, sticky="w", padx=10)

    # heartrate
    global heart_label_value
    heart_label_value = ttk.Label(data_frame, text="N/A", font=("Helvetica", 14, "bold"), fore
    ttk.Label(data_frame, text="Heart Rate:", font=("Helvetica", 14), foreground="#f4b41a", ba
    heart_label_value.grid(row=2, column=1, sticky="w", padx=10)

    # steps
    global steps_label_value
    steps_label_value = ttk.Label(data_frame, text="N/A", font=("Helvetica", 14, "bold"), fore
    ttk.Label(data_frame, text="Steps:", font=("Helvetica", 14), foreground="#f4b41a", backgro
    steps_label_value.grid(row=3, column=1, sticky="w", padx=10)

    # calories
```

Go to Line/Column

tkinterClient.py

```python
    # steps
    global steps_label_value
    steps_label_value = ttk.Label(data_frame, text="N/A", font=("Helvetica", 14, "bold"), fore
    ttk.Label(data_frame, text="Steps:", font=("Helvetica", 14), foreground="#f4b41a", backgro
    steps_label_value.grid(row=3, column=1, sticky="w", padx=10)

    # calories
    global calories_label_value
    calories_label_value = ttk.Label(data_frame, text="N/A", font=("Helvetica", 14, "bold"), f
    ttk.Label(data_frame, text="Calories Burned:", font=("Helvetica", 14), foreground="#f4b41a
    calories_label_value.grid(row=4, column=1, sticky="w", padx=10)

    # activity
    global activity_label_value
    activity_label_value = ttk.Label(data_frame, text="N/A", font=("Helvetica", 14, "bold"), f
    ttk.Label(data_frame, text="Current activity:", font=("Helvetica", 14), foreground="#f4b41
    activity_label_value.grid(row=5, column=1, sticky="w", padx=10)


    return app

if __name__ == "__main__":
    MQTT_BROKER_IP = "192.168.1.8"
    MQTT_TOPIC = "wearable/sensor"

    # Start the MQTT client
    setup_mqtt(MQTT_BROKER_IP, MQTT_TOPIC)

    # Start the GUI
    app = setup_gui()
    app.mainloop()
```

Activity Classification

The subscriber performs simple rule-based classification:
- Idle: Low acceleration, low heart rate.
- Walking: Moderate acceleration, moderate heart rate.
- Running: High acceleration, high heart rate.

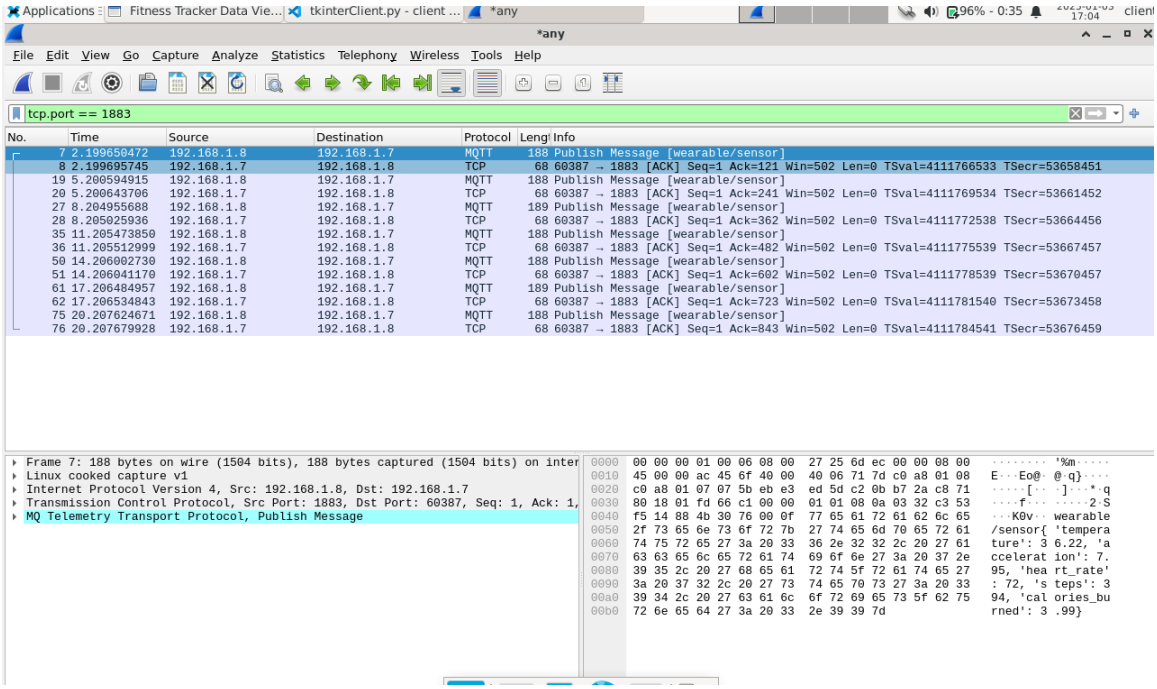
Classification Code:

```
if sensor_data['heart_rate'] > 120 or sensor_data['acceleration'] > 5.0:
    activity_label_value.config(text=f"Running")
elif sensor_data['heart_rate'] > 80 and sensor_data['heart_rate'] <= 120 or sensor_data['a
    activity_label_value.config(text=f"Walking")
elif sensor_data['heart_rate'] <= 80 or sensor_data['acceleration'] < 1.0:
    activity_label_value.config(text=f"Idle")
else:
    activity_label_value.config(text=f"Idle")
```

```
34    art_rate'] <= 120 or sensor_data['acceleration'] < 5.0:
```

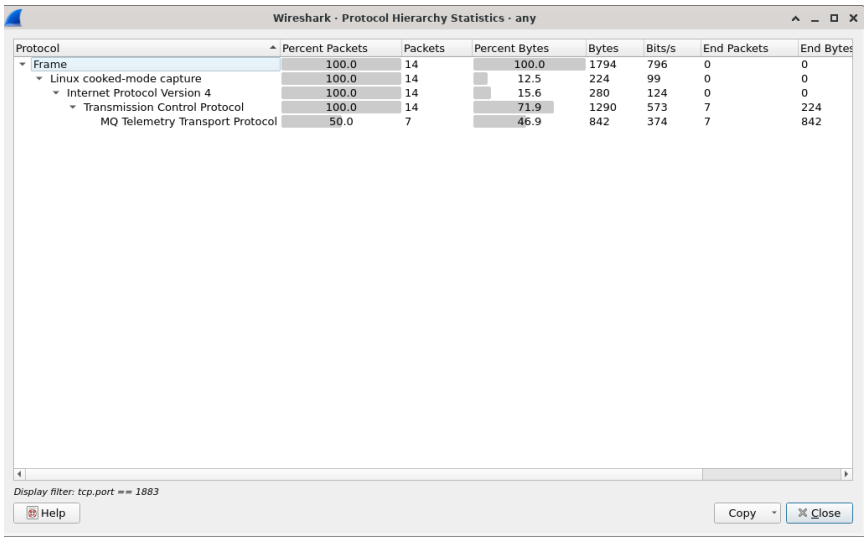
Network Usage

To test network usage, we used Wireshark, and we configured it to only look for data sent via port 1883 (the MQTT one):

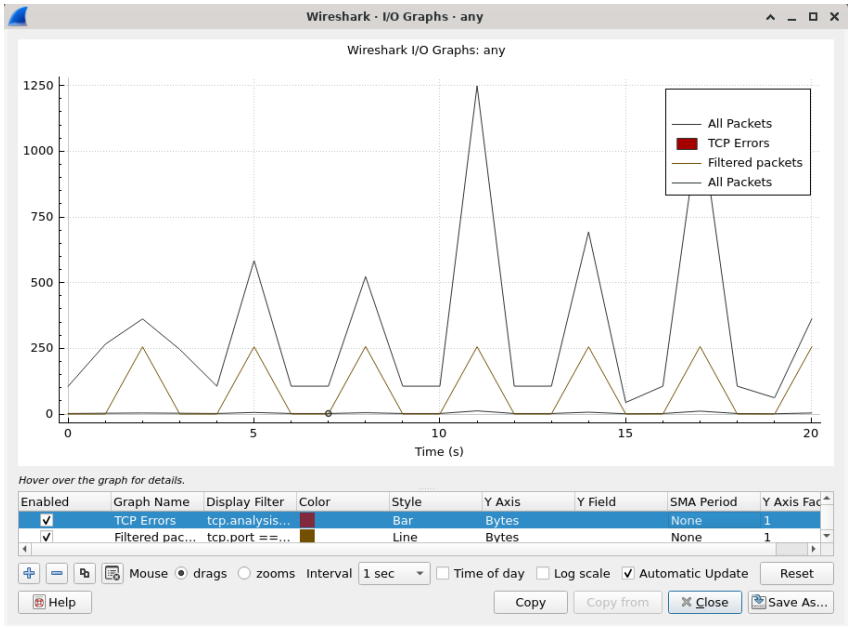


In the picture we can see that packets have the maximum length of 189 bytes and they are sent in intervals of 3 seconds.

Then we looked into the Hierarchy Statistics to see how much of the network the MQTT protocol uses.



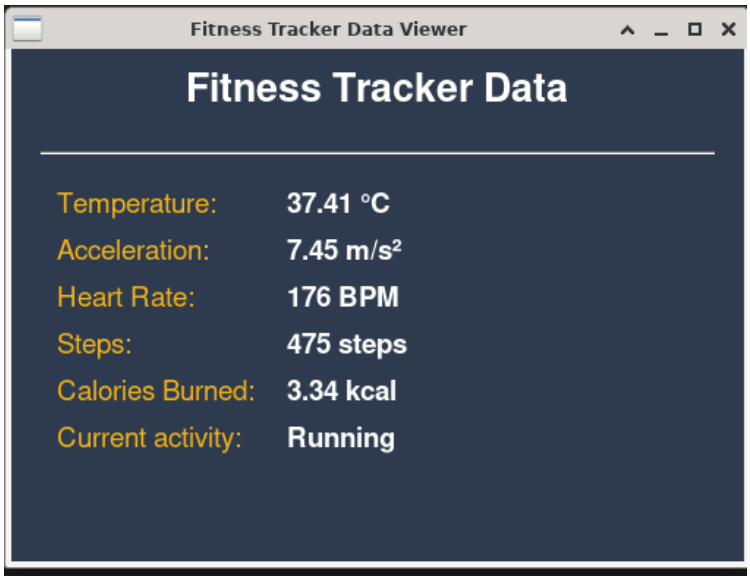
Then we checked the I/O graphs to see how many bytes are sent per tick.



1. Message Size:
   - Average message size: ~189 bytes (JSON-encoded sensor data).
2. Frequency:
   - Messages sent every 3 seconds.

3. Total Usage:
   - ~3.8 KB per minute.

Here is how the application looks like:



Conclusion

This project successfully demonstrates the use of MQTT as a lightweight protocol for wearable sensor networks. The system efficiently simulates and transmits fitness tracker data for Human Activity Recognition. Future improvements could include integrating machine learning models for activity classification and optimizing payload sizes for further network efficiency.