Wrocław, 4 listopada 2024

Przemysł 4.0

Laboratorium 4

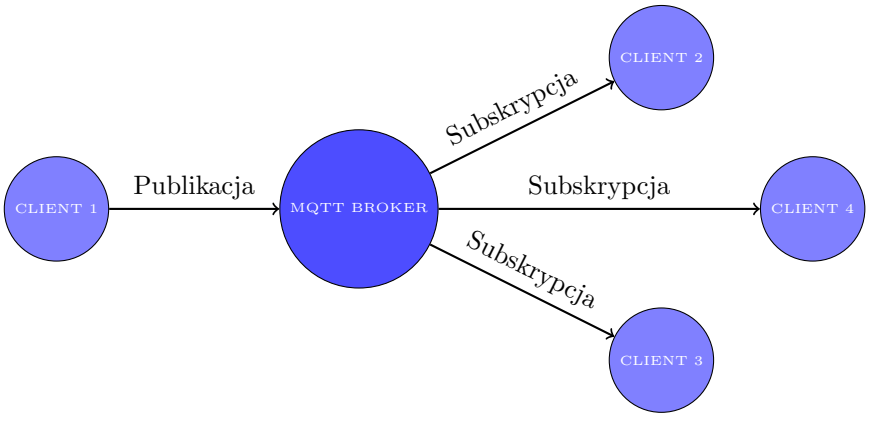
prowadzący: *Dr inż. Radosław Idzikowski*

1 Wprowadzenie

Celem laboratorium jest zapoznanie się z protokołem komunikacyjnym `MQTT`, zaprojektowanym z myślą o urządzeniach o ograniczonej mocy przetwarzania, typowych dla *Internet of Things* (`IoT`). Programy będą pisane w języku programowania `PYTHON` z użyciem połączenia zdalnego (*Remote Control*) z poziomu komputera klasy PC (preferowany system `Linux`) z wykorzystaniem `MS Visual Studio Code`. Czas przewidziany na wykonanie zadania to 1 termin wraz z ocenieniem pracy. Praca będzie oceniana na bieżąco w trakcie zajęć wraz z postępem programowania i testowania kolejnych przykładów. **Po ukończeniu każdego zadania należy zawołać prowadzącego w celu zaakceptowania etapu i odnotowania postępów.**

`MQTT` (*Message Queuing Telemetry Transport*) to lekki protokół komunikacyjny zaprojektowany do przesyłania danych w sposób niezawodny i efektywny. Jego główne cechy to:

- wydajność i lekkość — idealny do środowisk o ograniczonej przepustowości i zasobach (`IoT` i urządzenia wbudowane).
- model publikacja-subskrypcja — komunikacja odbywa się za pomocą brokerów, gdzie klienci mogą publikować dane z określonym tematem (np. "sensor/temperature") i subskrybować tematy, aby odbierać tylko interesujące ich dane.
- niezawodność – protokół oferuje różne poziomy jakości usług (`QoS`), zapewniając możliwość dostarczenia wiadomości nawet przy niestabilnym połączeniu.



Rysunek 1: Protokół MQTT

Dzięki tym cechom `MQTT` jest popularnym wyborem w systemach `IoT`, monitoringu oraz aplikacjach, gdzie liczy się niezawodna wymiana danych w czasie rzeczywistym. Protokół `MQTT` działa w oparciu o model publikacja-subskrypcja, który pokazano na rysunku 1. `Broker` pełni rolę centralnego węzła komunikacyjnego, który odbiera wiadomości od klientów publikujących oraz

rozsyła je do zainteresowanych subskrybentów. `Klient 1` publikuje dane na określony temat (np. "`sensor/data`") do *brokera*. `Klienci 2`, `3`, i `4` są subskrybentami tego tematu. Dzięki temu, gdy `Klient 1` wysyła dane do `brokera`, `broker` automatycznie przekazuje te dane do wszystkich subskrybentów. Ten model komunikacji pozwala na niezawodne i wydajne przesyłanie danych, nawet w środowiskach o ograniczonej przepustowości. `MQTT` jest często wykorzystywany w aplikacjach `IoT`, gdzie urządzenia (np. sensory) komunikują się z centralnym `brokerem`, a inne urządzenia lub aplikacje subskrybują ich dane.

2 Zadania

W ramach zajęć należy w zespołach wykonać następujące zadania:

1. Konfiguracja `brokera Mosquitto`.
2. Wizualizacja danych.
3. Informacja zwrotna.

Za wykonanie zadania nr 1 jest ocena dostateczna, za każde kolejne zadanie jest +1 do oceny. Na ocenę bardzo dobrą (5.0) należy wykonać wszystkie trzy zadania. Po zakończeniu zajęć należy kody źródłowe napisanych programów (w szczególności zadań nr 2 i 3) zgrać i przesłać do prowadzącego. Można dołączyć zdjęcia potwierdzające poprawne działanie. `Broker` **instalujemy koniecznie na naszym module** (*Raspberry Pi*).

3 Opis zadań

3.1 Zadanie 1

Zasadniczym celem zadania jest przesłanie danych z wykorzystaniem protokołu `MQTT`. W pierwszej kolejności należy zainstalować `broker Mosquitto` z użyciem podanych poniżej komend. Jest to implementacja serwera typu `open source` dla wersji 5.0, 3.1.1 i 3.1 protokołu `MQTT`.

```
sudo apt-get update
sudo apt-get install mosquitto
sudo apt-get install mosquitto-clients
```

Po zainstalowaniu `brokera` należy sprawdzić jego konfigurację. W tym celu konieczne jest otwarcie pliku konfiguracyjnego `/etc/mosquitto/mosquitto.conf` z uprawnieniami `sudo` przy użyciu np. edytora `nano`. Następnie trzeba upewnić się, czy znajdują się w nim następujące linie odpowiedzialne za nasłuchiwanie na porcie `1883` (domyślny dla protokołu `MQTT`).

```
listener 1883
allow_anonymous true
```

W przypadku kiedy należało dopisać brakujące linie, trzeba wymusić reset `brokera Mosquitto`.

```
sudo systemctl restart mosquitto
```

Przed przystąpieniem do działania należy zainstalować jeszcze odpowiednią paczkę dla języka `Python`. Na komputerze pod systemem *MS Windows* można skorzystać z `pip`, pod `Ubutnu` należy stworzyć wirtualne środowisko `python3 -m venv myenv` oraz następnie je aktywować `source myenv/bin/activate`, w przypadku `Raspberry` używamy `apt`.

```
sudo apt install python3-paho-mqtt
```

Dla każdego typu klienta należy podać parametry połączenia z `brokerem MQTT` (który został zainstalowany na `Raspberry`). Jako `broker_address` trzeba wpisać adres IP mikrokomputera `Raspberry`. Port należy wpisać zgodnie ze wcześniejszą konfiguracją.

```
broker_address = "192.168.1.19"
broker_port = 1883
topic = "sensor/data"
```

Na listingu 1 przedstawiono kompletny kod dla klienta, który jest nadawcą. Poza załączonymi niezbędnymi paczkami dopisano parametry połączenia. Funkcja `read_sensor` symuluje odczyt z czujnika wilgotności i temperatury. **W zadaniu należy użyć wartości z czujnika temperatury**. Funkcja `publish_data` pobiera dane z czujnika za pomocą `read_sensor`, formatuje odpowiednio tekst – ładunek (`payload`) oraz publikuje go na określonym temacie. Na koniec tworzymy instancję klienta `MQTT` oraz łączymy się z `brokerem MQTT` pod wskazanym adresem i portem. Dane są publikowane w pętli.

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

broker_address = "192.168.1.19"
broker_port = 1883
topic = "sensor/data"

def read_sensor():
    return random.randint(500, 999)/10, random.randint(150, 300)/10

def publish_data(client):
    humidity, temperature = read_sensor()
    if humidity is not None and temperature is not None:
        payload = f"{temperature:.2f} {humidity:.2f}"
        client.publish(topic, payload)
        print(f"Published: {payload}")
    else:
        print("Read error!")

client = mqtt.Client()
client.connect(broker_address, broker_port, 60)
while True:
    publish_data(client)
    time.sleep(5)
```

Listing 1: Klient nadawca (`publisher`)

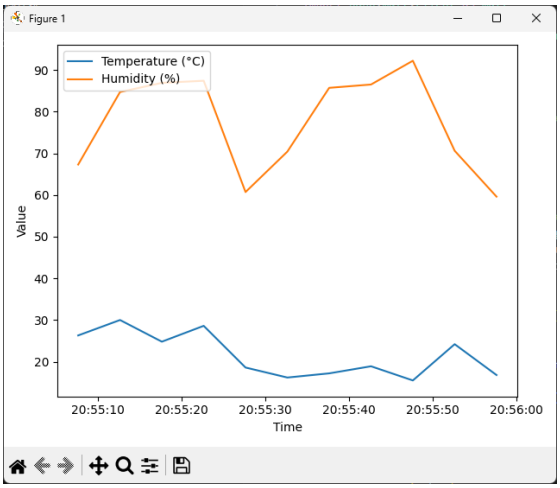
Na listingu 2 przedstawiono kompletną implementację dla klienta, który może subskrybować wybrany temat. W tym przypadku do działania wystarczy załączenie jedynie pakietu `paho.mqtt` oraz podanie parametrów połączenia. Funkcja `on_message` zostaje wywołana po odebraniu wiadomości na subskrybowanym temacie. Pokazany przykładowy program drukuje treść wiadomości w konsoli. Tutaj również tworzymy instancję klienta `MQTT` i nadpisujemy domyślną funkcję `on_message` naszą implementacją. Następnie łączymy się z naszym `brokerem` i subskrybujemy interesujący nas *topic*. Funkcja `client.loop_forever()` utrzymuje połączenie i oczekuje na przychodzące wiadomości.

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

broker_address = "192.168.1.19"
broker_port = 1883
topic = "sensor/data"

def on_message(client, userdata, message):
    payload = message.payload.decode("utf-8")
    temp, hum = map(float, payload.split(" "))
    print(f"temperature: {temp}  humidity: {hum}")

client = mqtt.Client()
client.on_message = on_message
client.connect(broker_address,broker_port)
client.subscribe(topic)
print(f"Topic subscription '{topic}' on broker {broker_address}:{broker_port}")
client.loop_forever()
```

Listing 2: Klientów subskrybujący (`subscriber`)

3.2 Zadanie 2

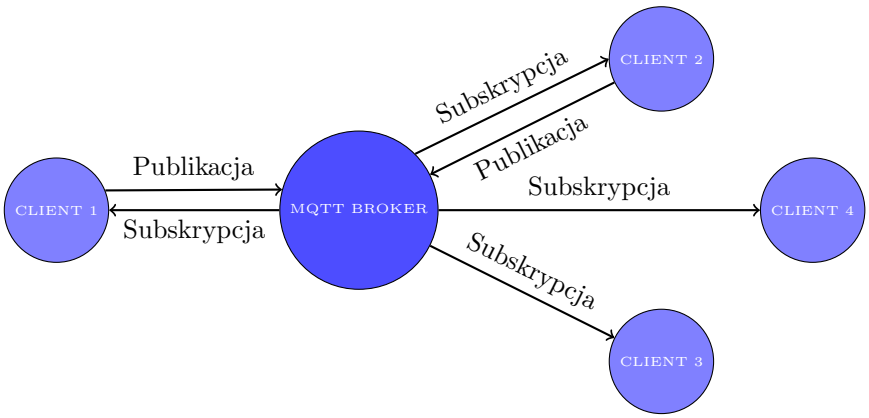
W tym zadaniu należy zmodyfikować klienta subskrybującego (tego na komputerze). Klient na wykresie ma przedstawiać trend zmian wartości odczytanych z czujnika w czasie rzeczywistym (wystarczy aktualizacja co $5s$). Do rysowania wykresu można użyć paczki `matplotlib.pyplot` oraz funkcji `FuncAnimation` z `matplotlib.animation`. Na rysunku 2 przedstawiono przykładową wizualizację do implementacji z dwoma wartościami. Ponadto koniecznie trzeba zasępić linię `client.loop_forever()` na `client.loop_start()`, a funkcje odpowiedzialną za aktualizowanie wykresu dać poniżej.



Rysunek 2: Klient subskrybujący z wizualizacją

3.3 Zadanie 3

Komunikacja z wykorzystaniem protokołu `MQTT` może się odbywać w dwie strony. Tak jak pokazano na rysunku 3, każdy klient może jednocześnie publikować jak i subskrybować, ale w ramach różnych `topików`. Celem zadania będzie obsłużenie komunikacji zwrotnej. W przypadku gdy klient na komputerze odbierze wartość temperatury powyżej założonego progu, ma przesłać na `topic "control/led"` komendę `"ON"` lub `"OFF"` gdy próg nie zostanie przekroczony.



Rysunek 3: Protokół MQTT

Na listingu 3 dla ułatwienia pokazano jak poprawnie przerobić klienta uruchamianego na module. Po pierwsze należy dołożyć bibliotekę `RPi.GPIO` niezbędną do obsługi diody `LED`. Poniższy listing zakłada konfigurację diody na `pinie 19`. Funkcje `read_sensors` oraz `publish_data`

pozostają bez zmian. Do obsługi diody należy zmodyfikować funkcję `on_message`, która w zależności od komendy `"ON"` lub `"OFF"` ustawi status diody. Do programu dołożono subskrypcję dla odpowiedniego tematu `topic_control`. Ponadto kod uzupełniono o obsługę przerwania programu, aby wyczyszczone zostały stany na pinach oraz poprawnie rozłączono klienta. W przypadku klienta na komputerze, ingerencja w kod ogranicza się głównie do poprawienia funkcji `on_message`.

```
import paho.mqtt.client as mqtt
import time
import random
import RPi.GPIO as GPIO

broker_address = "192.168.1.19"
broker_port = 1883
topic_sensor = "sensor/data"
topic_control = "control/led"

led_pin = 19
GPIO.setmode(GPIO.BCM)
GPIO.setup(led_pin, GPIO.OUT)

client = mqtt.Client()
client.on_message = on_message

client.connect(broker_address, broker_port, 60)
client.subscribe(topic_control)
client.loop_start()

try:
    while True:
        publish_data(client)
        time.sleep(5)
except KeyboardInterrupt:
    pass
finally:
    GPIO.cleanup()
    client.loop_stop()
    client.disconnect()
```

Listing 3: Zmodyfikowany klientów subskrybujący (`subscriber`)