Podstawy Internetu Rzeczy
Instrukcja do zajęć laboratoryjnych

Laboratorium 11

MQTT - przykład protokołu transmisji danych w sieciach Internetu Rzeczy

Opracował opiekun przedmiotu dr inż. Krzysztof Chudzik

Wydział Informatyki i Telekomunikacji
Politechnika Wrocławska

Wrocław, 2025.09.30 18:50:10

Spis treści

Lista zadań

Zanim przystąpimy do konfigurowania i programowania, pamiętajmy, że z tych samych zestawów korzysta wielu Studentów. Kolejne instrukcje laboratoryjne przygotowane są z założeniem, że zestawy posiadają oryginalną konfigurację laboratoryjną. **Zabrania się wprowadzania trwałych zmian konfiguracyjnych systemu operacyjnego oraz jakiegokolwiek oprogramowania w sposób zmieniający jego działanie, w tym działanie interfejsów graficznych. Zmiany takie mogą uniemożliwić prawidłowe przeprowadzenie kolejnych zajęć.** Proszę umożliwić innym Studentom odbycie zajęć, tak jak zostały one zaplanowane. Kto nie zastosuje się do tego i będzie dezorganizował zajęcia, będzie traktowany jako uszkadzający zestawy i zostanie odsunięty od zajęć.

1 Podstawowe informacje o protokole MQTT

Protokół MQTT jest jednym z bardziej popularnych protokołów aplikacyjnych wykorzystywanych w Internecie Rzeczy. Powstawał z przeznaczeniem dla urządzeń, gdzie istotny jest mały pobór mocy, i co jest tego konsekwencją, jak najmniejsze wymagania odnośnie pasma do przesyłu danych. Protokół oparty jest o wzorzec publikacja/subskrypcja.

Protokół ten jest dobrze udokumentowany w materiałach dostępnych w Internecie. Informacje na temat protokołu MQTT można znaleźć, na przykład, na następujących witrynach:

- Witryna mqtt.org (link),
- GitHub - MQTT (link),
- Wikipedia - MQTT (link).
- MQTT Essentials - The Ultimate Kickstart For MQTT Beginners (link),

Ostatnia pozycja na liście zawiera cykl przejrzystych, krótkich artykułów dotyczących najważniejszych aspektów protokołu MQTT. Nie ma konieczności pobierania książki (e-booka) w zamian za dane osobowe. Krótkie artykuły są zmieszczone poniżej informacji o książce (stan na dzień pisania/aktualizacji instrukcji).

2 Broker MQTT Mosquitto

Eclipse Mosquitto™ jest brokerem protokołu MQTT. Na witrynie projektu Eclipse Mosquitto (link) znajdziemy następujący opis, który przytoczymy w tłumaczeniu:

„Eclipse Mosquitto jest brokerem komunikatów o otwartym kodzie źródłowym (na licencji EPL/EDL), który implementuje protokół MQTT w wersjach 5.0, 3.1.1 i 3.1. Mosquitto jest lekki i nadaje się do użycia na wszystkich urządzeniach od jednopłytkowych komputerów o małej mocy po w pełni funkcjonalne serwery.

Protokół MQTT zapewnia lekką metodę realizacji przesyłania komunikatów z wykorzystaniem modelu publish/subscribe. To sprawia, że nadaje się do przesyłania wiadomości w Internecie Rzeczy, składających się z urządzeń takich jak czujniki o niskiej mocy lub urządzenia mobilne, telefony, systemy wbudowane lub mikrokontrolery.

Projekt Mosquitto dostarcza również bibliotekę C do implementacji klientów MQTT, oraz bardzo popularne programy (komendy) `mosquitto_pub` i `mosquitto_sub` działające jako oprogramowanie klienckie w linii poleceń."

Te dwa ostatnie można wykorzystać do weryfikacji poprawności funkcjonowania brokera, o czym powiemy w podrozdziale 3.

3 Uruchomienie Mosquitto w systemie Raspberry Pi OS

Na zestawach Rasperry Pi instalacja Mosquitto może zostać przeprowadzona tak, jak opisano to w artykule: Introduction to IoT: Build an MQTT Server Using Raspberry Pi (link). Proszę zapoznać się z tym artykułem.

Instalacja na Raspberry Pi OS sprowadza się do następujących komend:

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

W zestawach laboratoryjnych broker Mosquitto jest zainstalowany, ale skonfigurowany tak, aby nie uruchamiał się przy starcie systemu automatycznie. Ma to na celu polepszenie wydajności systemu, gdy Mosquitto nie jest wykorzystywany na zajęciach. Proszę nie zmieniać tego ustawienia. Jeśli zachodzi taka potrzeba, automatyczny start usługi brokera po starcie systemu konfigurujemy komendą (nie wykonywać na zestawach laboratoryjnych):

```
sudo systemctl enable mosquitto.service
```

Powyższe nie jest równoważne z uruchomieniem usługi natychmiast. Start usługi przeprowadzamy komendą:

```
sudo systemctl start mosquitto.service
```

Weryfikacji, czy usługa działa, dokonujemy komendą:

```
sudo systemctl status mosquitto.service
```

Teraz możemy sprawdzić, czy broker działa poprawnie. Wykonamy test będący odpowiednikiem programu "Hello World!". W tym celu wykorzystamy dwie konsole. W pierwszej konsoli uruchamiamy subskrypcję do tematu (*topic*) `"test/message"`:

```
mosquitto_sub -h localhost -t "test/message"
```

Program ten, dopóki jest uruchomiony, wypisuje na ekranie odebrane wiadomości:

W drugiej konsoli wydajemy komendę, która służy do opublikowania (wysłania) wiadomości:

```
mosquitto_pub -h localhost -t "test/message" -m "Hello, world"
```

Jeśli wszystko działa poprawnie, na pierwszej konsoli, uruchomiony program `mosquitto_sub` powinien wypisać na ekranie:

```
Hello, world
```

Na drugiej konsoli można wydawać kolejne polecenia opublikowania (wysłania) wiadomości, które będą wypisywane na pierwszej konsoli.

Powyższa procedura działa poprawnie jeśli polecenia `mosquitto_sub` i `mosquitto_sub` były wydawane bezpośrednio w terminalu uruchomionym lokalnie lub podłączonym przez protokół `ssh` do Raspberry, na którym uruchomiono broker Mosquitto.

W przypadku dostępu zdalnego za pomocą protokołu MQTT, wymagana jest dodatkowa konfiguracja brokera. Broker jest domyślnie zabezpieczony przed nieautoryzowanym dostępem zdalnym. Materiały dotyczące tego zagadnienia są wskazane w rozdziale 7.

Aby zezwolić na nieautoryzowany dostęp zdalny do brokera na Raspberry Pi należy w pliku `/etc/mosquitto/mosquitto.conf` dopisać następujące linie:

```
allow_anonymous true
listener 1883 0.0.0.0
```

Ponieważ wymagane są uprawnienia administracyjne, można użyć edytora, na przykład, `nano` wydając polecenie:

```
sudo nano /etc/mosquitto/mosquitto.conf
```

Aby konfiguracja została wdrożona, należy wydać jeszcze polecenie restartu brokera:

```
sudo systemctl restart mosquitto.service
```

Teraz można wydać na komputerze zdalnym polecenia `mosquitto_sub` i `mosquitto_sub` wskazując, zamiast `localhost`, numer IP skonfigurowany na Raspberry Pi.

Powyższa procedura jest procedurą, którą można wykorzystać jedynie do eksperymentów w sieciach izolowanych. Nie może być wdrażana w systemach produkcyjnych.

Konfiguracja ta, ze względów bezpieczeństwa informatycznego, nie została wprowadzona jako domyślna na zestawach laboratoryjnych z Raspberry Pi. Przypominam równocześnie, że po restarcie Raspberry Pi, konfiguracja ta zostanie utracona, ponieważ przywracana jest konfiguracja domyślna zestawu laboratoryjnego.

4 Instalacja Mosquitto w innych systemach

Generalnie, instalacja na systemie Linux jest zależna od dystrybucji. Na ogół oprogramowanie jest w repozytorium oprogramowania dystrybucji. Przykładowo, instalacja na systemach wywodzących od dystrybucji Debian może być podobna do procedury opisanej dla Raspberry Pi we wspomnianym już artykule Introduction to IoT: Build an MQTT Server Using Raspberry Pi (link). Alternatywą jest użycie oprogramowania `snap` służącego do zarządzania pakietami oprogramowani. Jeśli wykorzytany zostanie `snap` to instalacja odbywa się komendą:

```
snap install mosquitto
```

Aby zainstalować na systemie Windows należy pobrać odpowiedni pakiet instalacyjny z witryny: Mosquitto - Download (link). Zainstalować i zrestartować komputer. Odnaleźć katalog, gdzie zainstalowany został program (na przykład, `C:\Program Files\mosquitto`). Uruchomić konsolę `cmd`, przejść do tego katalogu. Powinny znajdować się tam dwa programy: `mosquitto_sub` i `mosquitto_pub`. Testy poprawności działania brokera można przeprowadzić tak, jak opisano w instrukcji instalacji dla Rasperry Pi.

Inne sposoby instalacji opisane są krótko na witrynie Mosquitto - Download (link).

5 Przykład wykorzystania brokera

Przykładowe skrypty, które korzystają z brokera `createdatabase.py` (kod 1), `sender.py` (kod 2) i `receiver.py` (kod 3) tworzą prosty system wysyłania i odbierania komunikatów z wykorzystaniem protokołu MQTT.

Do działania przykładów wymagane jest zainstalowanie biblioteki `paho-mqtt`. Można zrobić to komendą:

```
pip3 install paho-mqtt
```

W pierwszej kolejności upewnić się, że broker Mosquitto działa poprawnie. Uruchomić skrypt `createdatabase.py`. Wytworzy on pustą bazę danych. Następnie, uruchomić skrypty `receiver.py` i `sender.py`. Program `sender.py` posiada okienko z przyciskami. Po ich naciśnięciu wysyłany jest komunikat. Program receiver.py rejestruje te komunikaty. Proszę uruchamiać oba skrypty z osobnych konsolach (terminalach), ponieważ blokują konsolę lub piszą do konsoli. Proszę klikać przyciski w oknie programu `sender` i obserwować zachowanie okna konsoli `receiver`.

W razie potrzeby proszę dokonać edycji pierwszej linii wskazującej interpreter oraz w dalszej części kodu parametry opisujące broker (nazwa lub IP).

Kod 1: Plik `createdatabase.py`.

```
#!/usr/bin/env python3

import sqlite3
import time
import os


def create_database():
    if os.path.exists("workers.db"):
        os.remove("workers.db")
        print("An old database removed.")
    connection = sqlite3.connect("workers.db")
    cursor = connection.cursor()
    cursor.execute(""" CREATE TABLE workers_log (
        log_time text,
        worker text,
        terminal_id text
    )""")
    connection.commit()
    connection.close()
    print("The new database created.")


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

Kod 2: Plik sender.py.

```python
#!/usr/bin/env python3

import paho.mqtt.client as mqtt
import tkinter

# The terminal ID - can be any string.
terminal_id = "T0"
# The broker name or IP address.
broker = "localhost"
# broker = "127.0.0.1"
# broker = "10.0.0.1"

# The MQTT client.
client = mqtt.Client()

# Thw main window with buttons to simulate the RFID card usage.
window = tkinter.Tk()

def call_worker(worker_name):
    client.publish("worker/name", worker_name + "." + terminal_id,)


def create_main_window():
    window.geometry("300x200")
    window.title("SENDER")

    intro_label = tkinter.Label(window, text="Select employee:")
    intro_label.grid(row=0, columnspan=5)

    button_1 = tkinter.Button(window, text="Employee 1",
                              command=lambda: call_worker("Employee 1"))
    button_1.grid(row=1, column=0)
    button_2 = tkinter.Button(window, text="Employee 2",
                              command=lambda: call_worker("Employee 2"))
    button_2.grid(row=2, column=0)
    button_3 = tkinter.Button(window, text="Employee 3",
                              command=lambda: call_worker("Employee 3"))
    button_3.grid(row=3, column=0)
    button_4 = tkinter.Button(window, text="Employee 4",
                              command=lambda: call_worker("Employee 4"))
    button_4.grid(row=1, column=1)
    button_5 = tkinter.Button(window, text="Employee 5",
                              command=lambda: call_worker("Employee 5"))
    button_5.grid(row=2, column=1)
    button_6 = tkinter.Button(window, text="Employee 6",
                              command=lambda: call_worker("Employee 6"))
    button_6.grid(row=3, column=1)
    button_stop = tkinter.Button(window, text="Stop", command=window.quit)
    button_stop.grid(row=4, columnspan=2)


def connect_to_broker():
    # Connect to the broker.
    client.connect(broker)
    # Send message about conenction.
    call_worker("Client connected")


def disconnect_from_broker():
    # Send message about disconenction.
    call_worker("Client disconnected")
    # Disconnet the client.
    client.disconnect()


def run_sender():
    connect_to_broker()
    create_main_window()

    # Start to display window (It will stay here until window is displayed)
    window.mainloop()

    disconnect_from_broker()


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

Kod 3: Plik `receiver.py`.

```
#!/usr/bin/env python3

import paho.mqtt.client as mqtt
import tkinter
import sqlite3
import time

# The broker name or IP address.
broker = "localhost"
# broker = "127.0.0.1"
# broker = "10.0.0.1"

# The MQTT client.
client = mqtt.Client()

# Thw main window.
window = tkinter.Tk()

def process_message(client, userdata, message):
    # Decode message.
    message_decoded = (str(message.payload.decode("utf-8"))).split(".")

    # Print message to console.
    if message_decoded[0] != "Client connected" and message_decoded[0] != "Client disconnected":
        print(time.ctime() + ", " +
              message_decoded[0] + " used the RFID card.")

        # Save to sqlite database.
        connention = sqlite3.connect("workers.db")
        cursor = connention.cursor()
        cursor.execute("INSERT INTO workers_log VALUES (?,?,?)",
                       (time.ctime(), message_decoded[0], message_decoded[1]))
        connention.commit()
        connention.close()
    else:
        print(message_decoded[0] + " : " + message_decoded[1])


def print_log_to_window():
    connention = sqlite3.connect("workers.db")
    cursor = connention.cursor()
    cursor.execute("SELECT * FROM workers_log")
    log_entries = cursor.fetchall()
    labels_log_entry = []
    print_log_window = tkinter.Tk()

    for log_entry in log_entries:
        labels_log_entry.append(tkinter.Label(print_log_window, text=(
            "On %s, %s used the terminal %s" % (log_entry[0], log_entry[1], log_entry[2]))))

    for label in labels_log_entry:
        label.pack(side="top")

    connention.commit()
    connention.close()

    # Display this window.
    print_log_window.mainloop()


def create_main_window():
    window.geometry("250x100")
    window.title("RECEIVER")
    label = tkinter.Label(window, text="Listening to the MQTT")
    exit_button = tkinter.Button(window, text="Stop", command=window.quit)
    print_log_button = tkinter.Button(
        window, text="Print log", command=print_log_to_window)

    label.pack()
    exit_button.pack(side="right")
    print_log_button.pack(side="right")


def connect_to_broker():
    # Connect to the broker.
    client.connect(broker)
    # Send message about conenction.
    client.on_message = process_message
```

```
79      # Starts client and subscribe.
80      client.loop_start()
81      client.subscribe("worker/name")
82
83
84  def disconnect_from_broker():
85      # Disconnet the client.
86      client.loop_stop()
87      client.disconnect()
88
89
90  def run_receiver():
91      connect_to_broker()
92      create_main_window()
93      # Start to display window (It will stay here until window is displayed)
94      window.mainloop()
95      disconnect_from_broker()
96
97
98  if __name__ == "__main__":
99      run_receiver()
```

6 Dodatkowe materiały i rozwiązywanie problemów

Poniżej są dwa bardzo wyczerpujące artykuły, które mogą być pomocne przy rozwiązywaniu problemów z implementacją MQTT (drugi zawiera listę tutoriali):

- How to Use The Paho MQTT Python Client for Beginners (link),
- MQTT and Python For Beginners -Tutorials (link).

Gdyby Państwo potrzebowali, to proste tutoriale wideo odnośnie wykorzystanych w przykładach tkinter i sqlite, można znaleźć nawet na Youtube:

- Python GUI with Tkinter - 1 - Introduction (link),
- SQLite Database With Python (link).

Przedstawione materiały, należy traktować jako przykładowe. Podobnych, w różnej formie, jest wiele w Internecie i nie sposób wypisać wszystkich. Problemy związane z brokerem proszę rozwiązywać w miarę możliwości samodzielnie. Większość problemów będzie specyficzna dla Państwa programu lub konfiguracji zestawu Raspberry Pi lub komputera. Na pewno nie można narzekać na brak materiałów pomocnych w rozwiązywaniu problemów.

Zadanie 1: Implementacja protokołu MQTT do przekazywania informacji o zdarzeniach związanych z użyciem karty RFID

Proszę przygotować zestaw dwóch programów wykorzystujących protokół MQTT do komunikacji między sobą.

Pierwszy to program obsługujący czytnik kart RFID. Program ten, jako wydawca (publisher) protokołu MQTT, będzie wysyłał informację o identyfikatorze użytej karty i dokładnym czasie jej użycia. Proszę zadbać, aby karta RFID przyłożona do czytnika była odczytywana jeden raz, jeśli jest stale przyłożona do czytnika. Proszę sygnałem dźwiękowym i wizualnym poinformować użytkownika karty, że została ona odczytana.

Drugi program to klient, który jako subskrybent protokołu MQTT będzie odbierał informacje o użyciu kart RFID i zapisywał fakt użycia.

Jest to zadanie przykładowe i może zostać zmodyfikowane lub zmienione przez Prowadzącego zajęcia laboratoryjne.

7 UWAGA!: Bezpieczeństwo konfiguracji protokołu MQTT i brokera Mosquitto

Celem niniejszego laboratorium była jedynie prezentacja protokołu MQTT i brokera Mosquitto w działaniu, tak aby zrozumieć podstawowe zasady ich funkcjonowania. Przedstawiony w niniejszym materiale do laboratorium sposób konfiguracji i wykorzystania brokera Mosquitto nie zapewnia w żadnym stopniu ani bezpieczeństwa komunikacji, ani uwierzytelnienia komunikujących się stron. Prawidłowo wdrożony broker w środowisku produkcyjnym musi być skonfigurowany w sposób bezpieczny.

Jest wiele materiałów w Internecie na ten temat. Przykładowe to:

- MQTT Version 5.0, Par. 5 Security (non-normative) (link),
- Mosquitto - TLS support (link),
- Mosquitto - Username and password (link),
- `mosquitto_passwd` — manage password files for mosquitto (link),
- `mosquitto.conf` — the configuration file for mosquitto (link),
- Mosquitto SSL Configuration - MQTT TLS Security (link),
- Mosquitto Username and Password Authentication - Configuration and Testing (link),
- Mosquitto ACL - Configuring and Testing MQTT Topic Restrictions (link).

Zachęcam, do szukania kolejnych materiałów na ten temat.