Übung 06
Verwendung von MQTT
4abHWII

January 14, 2025

Übungsleiter: Albert Greinöcker

Ziel der Übung:

- Kennenlernen von MQTT:
  - Installation des Broker
  - Grundsätzliche Implementierung von MQTT Clients in Python, Plain Java und Android
  - Erhalten von Statusmeldungen

1 Installation und Einrichten von Mosquitto

1.1 Installation und erster Test

Der Einfachheit halber verwenden wir für diese Übung noch eine lokale (eigene) Installation eines brokers, nämlich Mosquitto : https://mosquitto.org/ - hier unter Downloads bitte die entsprechende Software abholen und am Laptop installieren (kommt später auf einen Server bzw. wird ein Broker in der Cloud wie HiveMQ (https://www.hivemq.com) verwendet).

a. Für's Erste kann man die Anwendung einfach starten (.exe ausführen) und sie läuft dann im Hintergrund: Unter Linux reicht der Befehl: mosquitto, unter Windows sollte schon ein Dienst im Hintergrund laufen (macht manchmal Probleme deshalb wie vorher angedeutet manuell starten).

b. Testen kann man das korrekte Funktionieren des Broker indem man die beiden ex_03-Skripte aus den raspberry_examples (https://github.com/albertgreinoecker/raspberry_examples) laufen lässt. Zuerst die Datei zu subscribe[1], dann die zu publish[2].

Abgabe dazu: Eventuelle Probleme, die bei der Installation aufgetaucht sind.

1.2 Konfiguration des Servers

Die Konfiguration geschieht über die Datei **mosquitto.conf**. Unter Linux befinden sich die meisten Konfigurationsdateien unter /etc, also /etc/mosquitto, unter Windows im Installationverzeichnis (c:/Programme/mosquitto
Nach den Änderungen muss man sicherheitshalber den mosquitto-Dienst neu starten:

- Linux: sudo service mosquitto restart

[1] ex_03_mqtt_mosquitto_subscribe
[2] ex_03_mqtt_mosquitto_publish.py

- Windows: Über die Verwaltung der Services (einfach links unten nach "Service" suchen) oder `net start` bzw. `net stop mosquitto` (oder die .exe neu starten)

Wenn die Einstellungen nicht gleich klappen, dann erst mal ohne Passwortzugang und Portumstellung weiterarbeiten, das können wir danach noch umstellen. wenn es benötigt wird!!!

Hier die Aktionen und die Einträge in `mosquitto.conf`[3]:

a. **Festlegen des Ports, z.B. 2222**:
`port 2222`
Möchte man den Port nach außen öffnen muss man anstatt `port` den Befehl `listener` verwenden:
`listener 2222`

Man muss dann auch in den Python-Beispielen den Port bei den `publish/subscribe`-Scripts eintragen:
`client.connect('127.0.0.1', port=2222)`

b. **Zugangsbeschränkung**: Um Beschränkung mittels user/password herzustellen braucht man eine Datei, die diese Informationen enthält. Das Erstellen dieser Datei wird mit einem speziell mitgeliefertem Programm erledigt: `mosquitto_passwd`
`mosquitto_passwd -c passwordfile user`, z.B.:
`mosquitto_passwd -c pw.txt albert` (konkretes Passwort wird danach erfragt und verschlüsselt in der Datei abgelegt)

In mosquitto.conf:
`allow_anonymous false`
`password_file /etc/mosquitto/pw.txt` # Wo die Datei liegt ist egal, nur muss der vollständige Pfad angegeben werden!!!!

in den Python-Beispielen:
Vor der Erstellung der Connection: `client.username_pw_set('albert', 'XXX')`
Um Herauszufinden, ob die Authentifizierung funktioniert hat, muss man eine entsprechende Callback-Methode festlegen und implementieren:
`client.on_connect = on_connect` #muss vor dem Verbindungsaufbau stehen

Die registrierte Methode könnte dann so aussehen:

```
def on_connect(client, userdata, flags, rc, properties=None):
  if rc == 0: #Wenn der return code 0 ist hat die Verbindung gepasst
    print("CONNECTION established")
  else:
    print("authentication error.")

```

Abgabe dazu: Die Datei `mosquitto.conf` und die verwendeten Befehle.

2 Konkrete Aufgabe

Unter der Verwendung des gerade erstellten Brokers (Mosquitto) soll eine kleine Anwendung erstellt werden, die Clients in Python, Plain Java und Android beinhaltet. Welche Daten inhaltlich geschickt werden bzw.

[3]liegt direkt in dem Verzeichnis in das `Mosquitto` installiert wurde, z.B: c://mosquitto.conf

welche kleine Anwendung dann tatsächlich daraus wird bleibt Euch überlassen, aber hier ein Beispiel was an Kommunikation denkbar wäre:

a. Der Android-Client (Beispiel unter `https://github.com/albertgreinoecker/Android_MQTTExamples`) könnte z.B. Sensordaten auslesen und publishen. Von den anderen Clients könnte auf diese mit einfachen Nachrichten reagiert werden. Wie Sensordaten ausgelesen werden befindet sich in unseren gemeinsamen Beispielen zu Android `https://github.com/albertgreinoecker/Android_Examples`, dort unter:
`app/src/main/java/at/ac/htlinn/androidexamples/sensor/SensorActivity.java`. Wir besprechen das noch im Unterricht.

b. Ein Python-Client der die Sensordaten entgegennimmt und Nachrichten schickt. Beispiele dazu sind unter: `https://github.com/albertgreinoecker/raspberry_examples/blob/master/ex_03_mqtt_mosquitto_publish.py` bzw.
`https://github.com/albertgreinoecker/raspberry_examples/blob/master/ex_03_mqtt_mosquitto_subscribe.py`

c. Ein Java-Client, der die Sensordaten sammelt und entsprechend darstellt (Ein Beispiel für die Verwendung liegt bei der Aufgabenstellung in Moodle).

Sensordaten können übrigens auch im Emulator simuliert werden. Wenn der Emulator läuft das Symbol ganz rechts (...) wählen und dort unter `Virtual Sensors`.

3 Zu den einzelnen Clients

3.1 Python

Grundsätzliches zur Programmierung: In unseren Beispielen gibt es am Ende immer den Aufruf: `client.loop_forever()`. Das passt natürlich wenn z.B. keine Benutzerinteraktion oder ähnliches stattfinden soll. Wenn man aber möchte dass der Prozess des "Horchens" auf neue Eingaben **nicht** blockiert, dann bietet sich die Verwendung von `client.loop_start()` an. Um das saubere Trennen der Verbindung muss man sich dann selber kümmern. Das ganze könnte dann so aussehen (Code nicht von hier sondern von den Beispielen kopieren):

```
import paho.mqtt.client as mqtt

# Callback-Funktionen werden hier definiert (so wie in unserem gemeinsamen Beispiel \texttt{ex\_03\_mqtt\_mosquitto\_subscribe.py})

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

client.connect("mqtt.broker.address", 1883, 60)

# Start the loop in the background
client.loop_start()

# Main loop for user input
try:
  while True:
  user_input = input("Enter your message (or type 'exit' to quit): ")
  if user_input.lower() == 'exit':
    break
  # Handle the user input
  client.publish("your/topic", user_input)
finally:
  client.loop_stop()  # Stop the loop
```

```
client.disconnect()
```

3.2 Java

Hier sind bei der Aufgabenstellung die wesentlichen Dateien dabei. So kann man diese in ein Projekt integrieren:

a. Erstellen eines `Maven`-Projektes z.B. in Eclipse

b. im `pom.xml` muss die Abhängigkeit eingetragen werden:

```
<dependencies>
  <dependency>
    <groupId>org.eclipse.paho</groupId>
    <artifactId>org.eclipse.paho.client.mqttv3</artifactId>
    <version>1.2.5</version>
  </dependency>
</dependencies>
```

c. die beiden Java-Dateien sollten dann schon funktionieren und sind ähnlich aufgebaut wie die Python-Lösungen

3.3 Android (Java)

Auch hier gibt es ein Projekt dass schon grundsätzlich mit dem (lokalen) Mosquitto-Server kommunizieren kann `https://github.com/albertgreinoecker/Android_MQTTExamples`. Folgende Einstellungen sind hier zu beachten (bitte nicht vom PDF herauskopieren sondern vom Projekt oder einfach das bestehende Projekt weiter bearbeiten):

- `AndroidManifest.xml`: Festlegen der Permissions und registrieren des `MQTT`-Service:

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<!-- damit auch wenn der Bildschirm gesperrt ist die Kommunikation weitergeht -->
<uses-permission android:name="android.permission.WAKE_LOCK" />
```

```
<service android:name="org.eclipse.paho.android.service.MqttService" />
```

- `build.gradle.kts` (Module app): Dependencies zu den MQTT-Bibliotheken und Legacy-Kompatibilität:

```
implementation ("androidx.legacy:legacy-support-v4:1.0.0")
implementation ("com.android.support:support-v4:28.0.0")
implementation( "org.eclipse.paho:org.eclipse.paho.client.mqttv3:1.1.0")
implementation( "org.eclipse.paho:org.eclipse.paho.android.service:1.1.1")
```

- Der Quellcode in ActitityMain wird im Unterricht besprochen. Wichtig ist die IP-Adresse `tcp://10.0.2.2:1883` - So kann man sich auf den Localhost des Laptops auf den der Emulator läuft hinverbinden (und somit auf den Mosquitto-Server). Möchte man die App direkt am Smartphone laufen lassen dann muss man die interne IP des Laptops eingeben.

Bitte einen Emulator mit der API Version 27 erzeugen, dann sollte es klappen!

4 Umstellen auf externe Broker

Leider ist es nicht möglich einen lokal installierten Broker für andere zugänglich zu machen (wegen Einstellungen am Netz), deshalb greifen wir auf diese beiden Lösungen zurück:

4.1 HiveMQ

Möchte man einen eigenen Broker in der Cloud erstellen bietet sich `HiveMQ` (`https://www.hivemq.com`) an. Dort kann man nach erfolgter Registrierung gratis einen Broker erstellen. Der Verwendete Port 883 ist in der HTL freigeschalten und sollte somit erreichbar sein. Um diesen zu verwenden muss man nicht viel am Code ändern, ein Beispiel dazu findet sich unter:
`https://github.com/albertgreinoecker/raspberry_examples/blob/master/ex_04_hivemq_client.py`. Hier müssen die Variablen `ids.hivemv_user` und `ids.hivemq_password` durch die konkreten Zugangsdaten ersetzt werden.

4.2 Mosquitto-Installation auf einem externen Server

Möchte man HiveMQ nicht verwenden kann man diesen Zugang probieren:

```
client.username_pw_set("htl4x", "rrG3q8y8Vg")
client.connect('23.88.107.144', 22)
```

Sonst bleiben die Einstellungen gleich wie bei beim Zugang zur lokalen Installation.

5 Grundsätzliche Aufgabenstellung

Ein Vorschlag wäre mittels gesendeten Sensordaten des Smartphones (z.B. Bewegen des Geräts) etwas an den Clients steuert. Hier kann man beim Ausbau zu einer beliebigen verteilten Anwendung kreativ sein.

Abgabe dazu: nur den Quellcode er einzelnen Clients