Fachhochschule
Dortmund

PA2

Oleksii Baida
Matrikelnummer 7210384

Projektarbeit 2

Bericht

22. September 2024

Inhaltsverzeichnis

1 Einleitung

2 Verbindungstheorie

2.1 MQTT

2.1.1 MQTT

Das MQTT-Protokoll[1] ist ein einfaches, effizientes und leichtgewichtiges Nachrichtenprotokoll, das speziell für den Einsatz in IoT-Systemen entwickelt wurde. Es ermöglicht die Kommunikation von Geräten mit geringer Bandbreite und begrenzten Ressourcen. Das Protokoll basiert auf dem Publish-Subscribe-Modell und erfordert daher eine zentrale Instanz, den Broker. Dies ermöglicht es den Geräten, Nachrichten an einen zentralen MQTT-Broker zu senden und von diesem zu empfangen. Die Nachrichten werden unter Topics veröffentlicht, welche Kanäle darstellen, zu denen sich Subscriber registrieren können. Das Backend für das MQTT-Protokoll kann mit NodeRed realisiert werden. NodeRed bietet ein sehr benutzerfreundliches Interface, um die Nachrichten vom Publisher zu empfangen, zu verarbeiten und an ein Endgerät zu senden, wie zum Beispiel einen Telegram-Bot oder ein Cloud-System.

MQTT wird aufgrund seiner Unkompliziertheit oft in Smart-Home-Anwendungen verwendet. Es zeichnet sich durch eine gute Zuverlässigkeit aus, da die Nachrichten immer in einer bestimmten Reihenfolge vermittelt werden. Jede Nachricht wird genau einmal gesendet. Obwohl es keine Garantie für die Zustellung der Nachricht gibt, werden Duplikate vermieden. MQTT ermöglicht das Speichern der letzten Nachricht im Topic, wodurch neue Abonnenten diese Nachricht sofort nach der Registrierung zum Topic erhalten. Aus eigener Erfahrung kann ich bestätigen, dass es bei häufigem Senden der Nachrichten keine Probleme mit der Zustellung gibt. Wenn Nachrichten für eine bestimmte Zeit ausbleiben, sollte überprüft werden, ob die Verbindung unterbrochen ist. MQTT bietet Last-Will-und-Testament-Funktion, was ermöglicht dem Broker eine bestimmte Nachricht bei dem Ausfall der Verbindung zu senden. Diese Nachricht kann bei der Registrierung des Subscribers zum Topic definiert werden und wird im Brocker gespeichert, bis er einen Verbindungsausfall erkennt.

Bei der Wahl des Protokolls sollten die Entwickler die folgenden Schlüsselpunkte berücksichtigen:

- **Data latency** – Wie schnell sollen die Daten übergeben werden? Wie kann man ein Packet vom Startpunkt zum Endpunkt vernünftig übergeben?
- **Reliability** – Welche Folgen hat Datenverlust im IoT-System? Wie kann das System zuverlässiger werden?
- **Bandwidth** – Wie groß sind Datenmengen, die transportiert werden sollen?
- **Transport** – Welches Protokoll ist für den Transport am besten geeignet? Am meisten wird zwischen TCP, UDP und HTTP entschieden.

2.2 UART

Hier wird die UART-Verbindung ESP-Arduino beschrieben

2.3 WLAN

Hier wird die WLAN-Verbindung ESP-Raspberry beschrieben

[1] Message Queuing Telemetry Transport

3 Raspberry Pi

3.1 Was ist Raspberry Pi?

Für das Projekt verwende ich einen Raspberry Pi 1.0 Modell B+. Raspberry Pi ist ein kompakter und kostengünstiger Einplatinencomputer. Er ist mit einem ARM-Prozessor, 512 MB RAM, USB-Ports, MicroSD-Kartensteckplatz, HDMI- und Ethernet-Anschluss sowie GPIO-Pins ausgestattet. Zusätzlich verfügt der Raspberry Pi über Wi-Fi- und Bluetooth-Funktionalität, die eine kabellose Vernetzung und Datenübertragung ermöglicht. Die Stromversorgung erfolgt über einen Micro-USB-Anschluss. Auf dem Raspberry Pi läuft das Betriebssystem Raspberry Pi OS, das auf Debian 12 "Bookworm"basiert.

In meinem Projekt verwende ich den Raspberry Pi als zentralen Server des Systems sowie den MQTT-Broker. Die Verbindung zwischen dem ESP8266 und dem Raspberry Pi erfolgt über WLAN. Dazu müssen beide Geräte an ein Netzwerk angeschlossen werden. Hierfür gibt es zwei Möglichkeiten: entweder werden die beiden Geräte über ein Heimnetzwerk mit dem Router verbunden, oder das Netzwerk wird vom Raspberry Pi zur Verfügung gestellt. Für mein Projekt habe ich mich für die zweite Variante entschieden. Der Raspberry Pi wird als ein Access Point konfiguriert und über Ethernet mit dem Internet verbunden. Der ESP8266 verbindet sich dann mit dem MQTT-Broker, der auf dem Raspberry Pi gehostet wird. Der Raspberry übernimmt somit die Rolle eines Routers. Dies hat folgende Vorteile:

- Das lokale WLAN-Netzwerk kann direkt am Raspberry Pi konfiguriert werden. Das vereinfacht die Verbindung der angeschlossenen Geräten sowie die Überwachung der Datenübertragung.
- Die Sicherheit der Verbindung wird dadurch erhöht, dass das lokale WLAN von außen (aus dem Internet) nicht sichtbar ist. Zusätzlich kann auf dem Raspberry Pi eine Firewall für die Internetverbindung eingerichtet werden.
- Das gesamte System ist portabler. Wenn das System an einem anderen Ort installiert wird, muss nur die Internetverbindung mit dem Raspberry Pi eingerichtet werden. Das lokale WLAN muss nicht angepasst werden.

Weiter unten wird die Einstellung des Raspberry Pi beschrieben.

3.2 Einrichtung als Access Point

Für die Einrichtung des Raspberry Pi als WLAN Access Point wird das Softwarepaket `hostapd`[1] verwendet. Mit Hilfe von hostapd können normale Netzwerkkarten in Access Points umgewandelt werden. Für meine Zwecke muss das Interface `wlan0` als Access Point umgewandelt werden.

Zunächst muss das Paket auf dem Raspberry Pi installiert werden.

```
sudo apt-get install hostapd
```

Für die Konfiguration des Access Points muss die Konfigurationsdatei erstellt werden.

```
sudo nano /etc/hostapd/hostapd.conf

#Konfigurationsdatei für hostapd
```

[1] Host Access Point Daemon

```
interface=wlan0
driver=nl80211
country_code=DE
ssid=RaspEsp
hw_mode=g
channel=6
wmm_enabled=0
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=mqtt1234
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP

#End
```

In Bezug auf die Vielzahl an verfügbaren Optionen, die sich für die Konfiguration des hostapd anbieten, habe ich für mein Projekt folgende Einstellungen gewählt:

- **interface** — bezeichnet das Interface, das zu Access Point konfiguriert wird.
- **driver** — nl80211 ist der standarte, meistverwendete Driver für `hostapd`.
- **country_code** — bezeichnet das Land, wo das Netzwerk läuft.
- **ssid** — der Name des Netzwerks.
- **hw_mode** — Operation Mode. 'g' steht für Standard IEEE 802.11g.
- **channel** — der verwendete Kanal für die WLAN-Verbindung
- **wmm_enabled** — Wireless Multimedia Extension muss für die WLAN-Verbindung aktiviert sein.
- **macaddr_acl** — Authentifizierung auf Basis des MAC-Protokolls. 0 akzeptiert alle MAC-Adressen, die nicht in Ablehnungsliste sind.
- **auth_algs** — Shared-Key-Authentifizierung
- **ignore_broadcast_ssid** — Standardeinstellung. Ignoriert Anfragen, die kein vollständiges SSID erhalten.
- **wpa** — Wi-Fi Protected Access. Art der Sicherheit des Access Points. Für das Projekt wird WPA 2 verwendet.
- **wpa_passphrase** — das Passwort für den Access Points
- **wpa_key_mgmt, wpa_pairwise, rsn_pairwise** — sind für WPA 2 benötigt.

Anschließend soll das hostapd automatisch beim Hochfahren starten. Dazu ist es erforderlich, die Einstellungen für den Systemstart anzupassen

```
sudo nano /etc/default/hostapd

#Auskommentieren oder neu schreiben
DAEMON\_CONF="/etc/hostapd/hostapd.conf"
```

Nun ist der `hostapd` konfiguriert, aber noch nicht gestartet. Die beiden Befehle starten die `hostapd`:

```
sudo systemctl unmask hostapd
sudo systemctl enable hostapd

sudo reboot #Neustart des Raspberry Pi
```

Danach wird ein Neustart des Raspberry Pi empfohlen, um die Änderung anzuwenden. Der Access Point sollte nun mit entsprechendem SSID auf anderen Geräten sichtbar sein, allerdings ist eine Verbindung zu ihm nicht möglich. Um die Verbindung zu ermöglichen, müssen der DNS- sowie der DHCP-Server auf dem Raspberry Pi eingerichtet werden.

3.3 Einrichtung des DNS- und DHCP-Servers

Der DNS[2]-Server übersetzt Domains in die IP-Adressen. Der DHCP[3]-Server weist die IP-Adressen den verbundenen Geräten zu. Für die reibungslose Verbindung und Kommunikation müssen beide Server auf dem Raspberry PI eingerichtet werden.

Für die Konfiguration des DNS-Servers sowie des DHCP-Servers wird das Softwarepaket `dnsmasq` verwendet. Das Paket ermöglicht eine einfache und schnelle Konfiguration der beiden Server auf Linux-basierten Systemen.

In der Konfigurationsdatei sind eine Vielzahl an die Optionen für die Einstellung des DNS-Servers vorgegeben, wobei lediglich ein Teil davon für die Bearbeitung relevant ist.

```
sudo nano /etc/dnsmasq.conf

#Konfigurationsdatei für DNS-Server
#Auskommentieren oder neu schreiben
interface=wlan0
bind-dynamic
domain-needed
bogus-priv
dhcp-range=192.168.1.100,192.68.1.110,255.255.255.0,12h

#End
```

- **interface** — definiert Interface, an welchem der DNS-Server fungiert.
- **bin-dynamic** — erlaubt die Verbindung nur mir exsistierenden Inrefaces.
- **domain-needed** — ignoriert DNS-Anfragen ohne Domänennamen.

[2]Domain Name System
[3]Dynamic Host Configuration Protocol

- **bogus-priv** — die DNS-Anfragen aus lokalem Netzwerk werden nicht an Haupt-DNS-Server weitegeleitet.
- **dhcp-range** — gibt den Bereich der IP-Adressen für DHCP-Server an.

Anschließend muss der DHCP-Server eingerichtet werden.

```
sudo nano /etc/dhcpcd.conf

#Konfigurationsdatei für DHCP-Server
nohook wpa_supplicant
interface=wlan0
static ip_address=192.168.1.10/24
static routers=192.168.1.1
```

Entsprechend den oben beschriebenen Einstellungen werden die IP-Adressen von 192.168.1.100 bis 192.168.1.110 durch den DHCP-Server verteilt und laufen nach 12 Stunden automatisch ab. Für den Host (Raspberry Pi) ist die IP-Adresse 192.168.1.10 reserviert. Default-Gateway hat die IP-Adresse 192.168.1.1

3.4 MQTT-Broker

Der MQTT Broker ermöglicht die Kommunikation zwischen den MQTT-Geräten. Der Broker empfängt Nachrichten von sogenannten "Publishernünd leitet sie an SSubscriber"weiter. Die Subscriber müssen sich für bestimmte Themen (Topics) registrieren.

In meinem System läuft der MQTT-Broker auf dem Raspberry Pi. Für die Einrichtung des Brokers wird das Softwarepaket `mosquitto`[99] verwendet. Mosquitto erlaubt schnelle und einfache Einstellung und Verwaltung von MQTT-Broker.

1. Installation:

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

2. Autostart einschalten:

   ```
   sudo systemctl enable mosquitto
   sudo systemctl start mosquitto
   ```

3. Konfiguration:

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

   listener 1883
   allow_anonymous true
   ```

   allow_anonymous für die Ersteinrichtung und zu Testzwecken einschalten. Die Konfiguration der Benutzer erfolgt weiterhin im Text LINK.

4. Konfigurationsdatei speichern `Strg+O` und schließen `Strg+X` und Mosquitto neu starten

   ```
   sudo systemctl restart mosquitto
   ```

[99] https://mosquitto.org/

5. Testen. Ein Subscriber für Topic registrieren:

```
mosquitto_sub -h localhost -t <DEIN_TOPIC>
```

Neues Fenster öffnen `Strg+T` und über einen Publisher eine Nachricht in das Topic senden.

```
mosquitto_pub -h localhost -t <DEIN_TOPIC> -m "MEINE 1. MQTT-NACHRICHT"
```

Nach dem Parameter -t (Topic) keine Klammern setzen. Nach dem Parameter -m (Message) den Text der Nachricht in Klammern setzen.

4 ESP8266

4.1 Was ist ESP8266?

Das ESP8266 ist ein kleines, günstiges WLAN-Modul. Es wurde von Espressif Systems entwickelt. Das Modul ermöglicht die kabellose Verbindung von Mikrocontrollern mit dem Internet. In meinem Projekt verwende ich das ESP8266, um den Arduino drahtlos mit dem Raspberry PI zu verbinden.

Der ESP8266 basiert auf einem 32-Bit-Prozessor und hat einen Systemtakt von 80 MHz bis 160 MHz. Das Modul verfügt über 64 kB RAM als Befehlsspeicher und 96 kB RAM als Datenspeicher. Das ESP8266 besitzt keinen internen Flash-Speicher für die Firware. Ansonsten wird die Firmware in einem externen Flash-Speicher abgelegt und wird blockweise in den RAM-Speicher geladen. Je nach Modell verfügt das ESP8266 über verschiedene Schnittstellen wie I/O-Ports und I2C. Ich verwende das Modell mit WLAN und UART [1]. Über den UART-Port wird der Programmcode in das ESP8266 geladen. Das Modul muss mit einer Spannung von 5 V versorgt werden.

Der ESP8266 ist mit vielen Programmiersprachen kompatibel. Für die Programmierung des Moduls verwende ich Visual Studio Code mit der PlatformIO Erweiterung.

4.2 Verbindung von Arduino und ESP8266

Die Verbindung zwischen Arduino und ESP8266 kann auf unterschiedliche Weise hergestellt werden. Im Rahmen meines Projekts erfolgt die Verbindung über eine serielle UART-Schnittstelle.

Der Arduino verfügt über RX- und TX-Pins (Pin 0 und 1) für die serielle Kommunikation. Der RX-Pin des Arduino muss mit dem TX-Pin auf dem ESP8266 verbunden werden und der TX-Pin umgekehrt.In der Folge können die Daten des Arduino über die Funktion Serial.print() an das Modul ESP8266 über die serielle Schnittstelle übermittelt werden. Dabei ist zu berücksichtigen, dass bei einer Verbindung des Arduino mit einem Rechner über den USB-Port der Arduino den USB-Port als serielle Schnittstelle definiert. Infolgedessen werden keine Daten über die TX- bzw. RX-Ports übergeben. Daher ist eine Versorgung des Arduino mittels eines Netzteiles mit 5 V erforderlich.

Aufgrund der Weiterleitung der Daten vom Arduino über die MQTT-Verbindung ist eine entsprechende Berücksichtigung der jeweiligen Formatierung erforderlich. Der Arduino übermittelt die Befehle mit dem Format ̈topic:message: Dies erleichtert dem ESP8266 die Verarbeitung der empfangenen Nachricht sowie deren Weiterleitung mit dem entsprechenden Topic. Auf der Abbildung 1 ist der Programmcode für die Verarbeitung der Daten von Arduino zu sehen.

[1] Universal Asynchronous Receiver-Transmitter

```ino
// Format topic:message
void readSerialData()
{
  if (Serial.available() > 0)
  {
    String readString = "";
    // Lese Daten aus Serial als String ab
    readString = Serial.readStringUntil('\n');
    Serial.print("\nReceived SERIAL: ");
    Serial.print(readString);
    if (sizeof(readString) > buss_serial)
    {
      Serial.print("Message too long!");
      return;
    }
    Serial.print("\nConvert String to char: ");
    // String in char - Feld konvertieren
    char readSerialChar[readString.length() + 1];
    readString.toCharArray(readSerialChar, readString.length() + 1);
    for (unsigned int i = 0; i < sizeof readSerialChar; i++)
    {
      Serial.print(readSerialChar[i]);
    }
```

```ino
    // Suche Position von ':'
    char *delim_pos = strchr(readSerialChar, ':');
    if (delim_pos != NULL)
    {
      size_t topic_length = delim_pos - readSerialChar;
      char topic[topic_length + 1];
      strncpy(topic, readSerialChar, topic_length);
      topic[topic_length] = '\0';
      char *message = delim_pos + 1;
      Serial.printf("\nMessage %s", message);
      // MQTT-Nachricht senden
      mqttClient.publish(topic, message);
    }
    else // kein : gefunden
    {
      Serial.print("FALSCHES FORMAT");
      return;
    }
  }
}
```

Abbildung 1: Verarbeitung der Daten vom Arduino

4.3 ESP8266 als Schnittstelle für die MQTT-Verbindung

In meinem Projekt verwende ich den ESP8266 für die Erstellung der MQTT-Verbindung zwischen Arduino und Raspberry Pi. Wie oben beschrieben, wird der Raspberry Pi als MQTT-Broker und WLAN-Access-Point eingerichtet. Der ESP8266 muss sich mit dem Zuganspunkt von Raspberry Pi verbinden und eine MQTT-Verbindung zu dem Broker erstellen. Außerdem muss der ESP8266 die Daten von Arduino erhalten.

4.3.1 Verbindung mit WLAN

Zunächst ist eine Verbindung des ESP8266 mit dem vom Raspberry Pi bereitgestellten WLAN erforderlich. Für diesen Zweck wird die Bibliothek `ESP8266WIFI.h` verwendet. Es muss ein Objekt `WifiClient` und zwei konstanten Variablen `WIFI_SSID` und `WIFI_PASSWORD` erstellt werden. In der Konstante `WIFI_SSID` wird der Name des WLAN-Netzwerks gespeichert und in der Konstante `WIFI_PASSWORD` wird das Passwort gespeichert.

Die Verbindungsroutine ist in der Funktion `connect_wifi()` (Abbildung 2) zu sehen. Diese Funktion wird beim Start des Moduls in der `setup()`-Funktion aufgerufen.

Das ESP8266 versucht jede 200 Milisekunden sich mit dem WLAN zu verbinden. Die Serial-Ausgaben in der Konsole sind nur für die Testzwecken benötigt.

4.3.2 Verbindung mit dem MQTT-Broker

Für die Verbindung des ESP8266 mit dem MQTT-Broker wird eine Bibliothek `PubSubClient.h` verwendet. Es ist erforderlich, dass der MQTT-Broker mit dem gleichen Netzwerk wie der ESP8266 verbunden ist. In meinem System läuft der Broker auf dem Raspberry Pi und das WLAN wurde auch vom Raspberry Pi zur Verfügung gestellt. So befinden sich die beiden Module in einem Netzwerk.

Im Programmcode am ESP8266 muss ein Objekt `PubSubClient` erstellt werden. Als Parameter wurde den `WifiClient` übergeben. In der konstanten Variable `MQTT_BROKER_ADRRESS` ist die IP-Adresse des Brokers und in der Variable `MQTT_PORT` ist die Portnummer gespeichert. Die Verbindungsroutine ist in die Funktion `connect_mqtt()` (Abbildung 3) eingepackt. Diese Funktion wird beim Start von ESP8266 in der `setup()`-Funktion aufgerufen.

Nach erfolgreicher Erstellung der Verbindung mit dem MQTT-Broker muss das ESP8266 das Topic `UBSCRIBE_TOPIC` abonnieren, um die Befehle von dem Broker zu erhalten. Das ermöglicht beideseitige Kommunikation zwischen ESP8266 und MQTT-Broker.

```
// Verbindung mit WLAN
void connect_wifi()
{
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("\nConnecting ESP to WIFI ");
  Serial.print(WIFI_SSID);

  while (WiFi.status() != WL_CONNECTED)
  {
    delay(200);
    Serial.print(".");
  }

  Serial.print("\nESP connected to WIFI with IP Address: ");
  Serial.print(WiFi.localIP());
}
```

Abbildung 2: ESP8266 connect_wifi()

```
void connect_mqtt()
{

  while (!mqttClient.connected())
  {
    if (mqttClient.connect(CLIENT_ID))
    {
      mqttClient.subscribe(SUBSCRIBE_TOPIC);
    }
    else
    {
      delay(1000);
    }
  }
}
```

Abbildung 3: ESP8266 connect_mqtt()

Sobald das ESP8266 die Daten von dem Arduino empfängt, wird die MQTT-Nachricht an den MQTT-Broker mit dem Befehl `mqttClient.publish(topic, message)` versandt (Abbildung 1).

Die Verarbeitung der empfangenen Nachricht erfolgt in der `callback`-Funktion (Abbildung 4). Für die Testzwecken wird die Nachricht sofort in das Topic `PUBLISH_TOPIC` gesendet.

```
void callback(char *topic, byte *payload, unsigned int length)
{
  Serial.print("\nReceived message on topic: ");
  Serial.print(topic);
  Serial.print(". Payload: ");
  String callbackMessage = "Received message:  ";
  for (unsigned int i = 0; i < length; i++)
  {
    Serial.print((char)payload[i]);
    callbackMessage += (char)payload[i];
  }
  mqttClient.publish(PUBLISH_TOPIC, callbackMessage.c_str());
}
```

Abbildung 4: ESP8266 callback()

Abbildungsverzeichnis