Fachhochschule
Dortmund

PA2

Oleksii Baida
Matrikelnummer 7210384

Projektarbeit 2

Bericht

11. Oktober 2024

Inhaltsverzeichnis

1 Einleitung

1.1 Gesamtüberblick über das System

Im Rahmen der Projektarbeit 2 entwickle ich mein System aus der Projektarbeit 1 weiter. Ziel des Projektes ist es, das System aus PA 1 dem Endbenutzer möglichst nahtlos zur Verfügung zu stellen. Die Kernidee des Projekts basiert auf der Entwicklung eines Prototyps, um einen realen Bedarf zu decken.

In der PA 1 habe ich ein Sicherheitssystem für das Haus entwickelt. Das System reagiert auf gefährliche Ereignisse wie Feuer, Gas oder Fremdbewegungen und bietet einen sicheren Zugang zum Haus. Das System ist nur offline verfügbar und hat keine Möglichkeit, den Endbenutzer über die Entfernung zu informieren. In diesem Projekt wird das System durch den Einsatz verschiedener Technologien und Hardwarekomponenten für den Endnutzer über das Internet verfügbar gemacht.

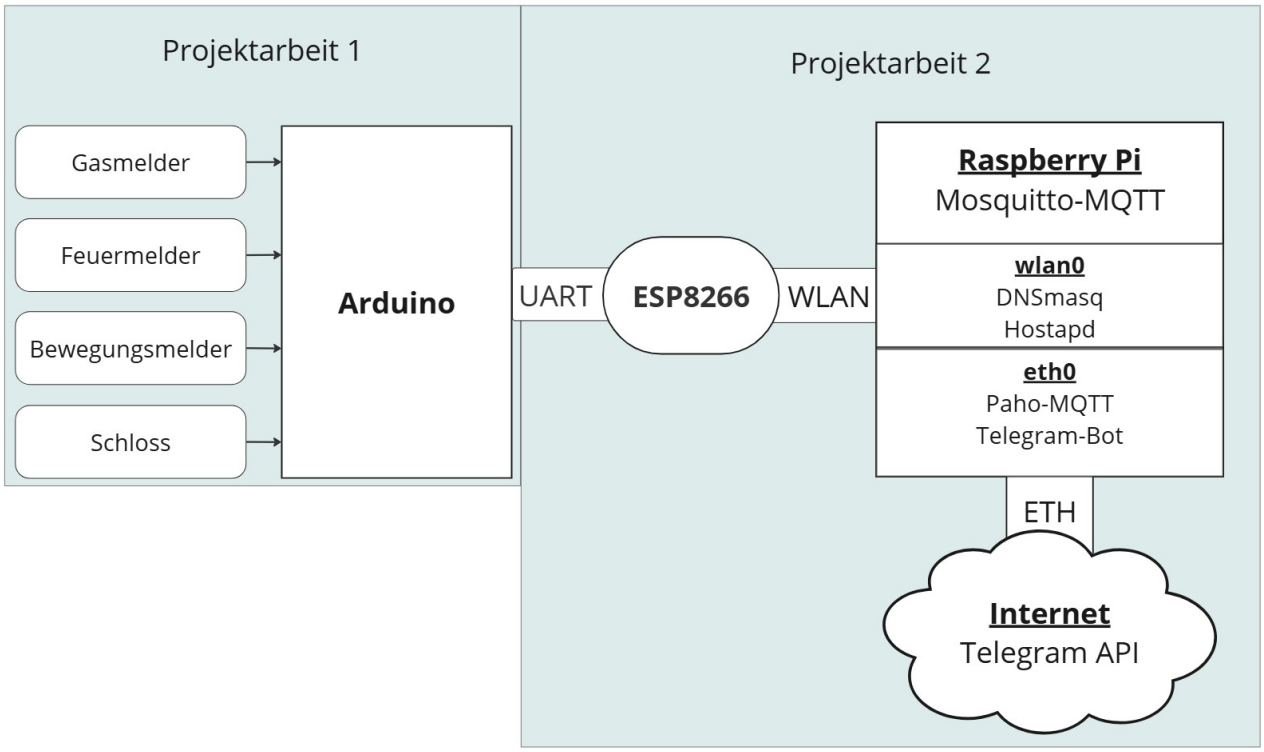

Abbildung 1: Gesamtplan

1.2 Verwendete Hardware

1.2.1 Raspberry Pi

Im Kern des Systems liegt der Raspberry Pi. Der Raspberry Pi dient als zentraler Server des Systems. Er hat verschiedene Funktionen im System. Der Raspberry Pi dient als MQTT-Broker, um die MQTT-Nachrichten zu verarbeiten. Er stellt einen WLAN-Zugriffspunkt zur Verfügung. Der Raspberry Pi dient auch als Host für den Telegram-Bot.

Für das Projekt verwende ich den Raspberry Pi 1.0 Modell B+. Raspberry Pi ist ein kompakter und kostengünstiger Einplatinencomputer. Die technischen Daten sind in der Tabelle 1 aufgeführt.

Modell	Raspberry Pi 1.0 Modell B+
Prozessor	ARM-Prozessor
Arbeitsspeicher (RAM)	512 MB
USB-Ports	4 Vorhanden
MicroSD-Kartensteckplatz	1 Vorhanden
HDMI-Anschluss	1 Vorhanden
Ethernet-Anschluss	1 Vorhanden
GPIO-Pins	40 Vorhanden
Wi-Fi	Vorhanden
Bluetooth	Vorhanden
Stromversorgung	5V Micro-USB-Anschluss
Betriebssystem	Raspberry Pi OS Debian 11 "Bullseye"

Tabelle 1: Technische Daten des Raspberry Pi

1.2.2 ESP8266

Das System, das ich in PA 1 entwickelt habe, basiert auf Arduino und verfügt über keine Schnittstelle zum Internet. In diesem Projekt verwende ich dafür ein ESP8266 Modul. Der Arduino ist über eine UART-Schnittstelle mit dem ESP8266 verbunden. Das ESP8266 verbindet sich mit dem vom Raspberry Pi bereitgestellten WLAN sowie mit dem MQTT-Broker, der auf dem Raspberry Pi läuft. Dadurch wird eine Verbindung vom Arduino zum Internet über den Raspberry Pi hergestellt.

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.

Das 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. Über den UART-Port wird der Programmcode in das ESP8266 geladen. Das Modul muss mit einer Spannung von 5 V versorgt werden.

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

1.3 Verbindungstheorie

1.3.1 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.

1.3.2 UART

UART[2] ist eine Hardwarekomponente, die die serielle Datenübertragung bei der Kommunikation zwischen Mikrocontrollern und anderen Geräten ermöglicht. Eine UART-Schnittstelle ist der Standard für serielle Schnittstellen an PCs und Mikrocontrollern und wird zum Senden und Empfangen von Daten über eine Datenleitung verwendet.

UART arbeitet asynchron, d.h. es gibt keine gemeinsame Taktverbindung zwischen den kommunizierenden Geräten. Stattdessen wird die Datenübertragung durch Start- und Stoppbits synchronisiert. Diese ermöglichen es dem Empfänger, den Beginn und das Ende eines Datenpakets zu erkennen. Ein typisches Datenpaket besteht aus einem Startbit, gefolgt von einer festgelegten Anzahl von Datenbits (normalerweise 8), einem optionalen Paritätsbit zur

[1] Message Queuing Telemetry Transport

[2] Universal Asynchronous Receiver Transmitter

Fehlerkontrolle und einem oder mehreren Stoppbits. Der Vorteil besteht darin, dass keine permanente Synchronisation zwischen Empfänger und Sender erforderlich ist. Sie müssen nur für die Dauer der Übertragung synchronisiert sein. Wenn keine Daten zu übertragen sind, setzt der Sender die Leitung auf die Polarität des Stopbits.

Ein entscheidender Vorteil der seriellen Kommunikation ist ihre Einfachheit, sowohl in der Implementierung als auch in der Verkabelung. Um die Kommunikation zwischen den Geräten herzustellen, muss die Rx-Leitung (eng. Receiver, Empfänger) eines Gerätes mit der Tx-Leitung (eng. Transceiver, Sender) eines anderen Gerätes verbunden werden. Dadurch entsteht eine Master-Slave-Beziehung zwischen den beiden Geräten, auch wenn sie gleichberechtigt kommunizieren können. Zudem ist UART in den meisten Mikrocontrollern integriert, was die Anwendung vereinfacht und die Kosten minimiert.

In meinem Projekt verwende ich eine UART-Schnittstelle für die Verbindung von Arduino und ESP8266.

1.4 Verwendete Software

1.4.1 Pyhton

Python ist eine weit verbreitete höhere Programmiersprache. Sie wurde 1991 von Guido van Rossum veröffentlicht.

Python hat eine klare und leicht lesbare Syntax. Die Strukturierung von Blöcken erfolgt nicht durch geschweifte Klammern, sondern durch Einrückungen. Python unterstützt verschiedene Programmierparadigmen wie objektorientierte, aspektorientierte und funktionale Programmierung. Zudem bietet Python eine dynamische Typisierung.

Python ist plattformunabhängig. Der Code von Python-Skripten wird nicht direkt in Maschinencode kompiliert, sondern zur Laufzeit in den Bytecode übersetzt. Der Bytecode ist nicht systemspezifisch, sondern stellt eine Zwischenschicht dar. Der Bytecode wird dann von der Python Virtual Machine (PVM) interpretiert und ausgeführt. Die PVM übernimmt die betriebssystemspezifischen Aufgaben wie Speicherverwaltung, Zugriff auf Prozessoren oder die Nutzung von Systembibliotheken. Dadurch können Python-Programme auf verschiedenen Plattformen ausgeführt werden, sofern eine geeignete PVM für die Plattform vorhanden ist.

Die Standardbibliothek von Python umfasst eine umfangreiche Sammlung von Modulen und Funktionen, die auf allen unterstützten Plattformen lauffähig sind. Darüber hinaus steht eine Vielzahl von Drittanbieter-Bibliotheken zur Verfügung, die ebenfalls plattformunabhängig sind.

Im Rahmen meines Projektes werden Python-Skripte für die Behandlung der MQTT-Nachrichten sowie für die Steuerung des Telegram-Bots verwendet. Die Ausführung des Codes erfolgt auf dem Raspberry Pi.

1.4.2 paho-mqtt

Für die Behandlung der MQTT-Nachrichten, die an den Broker gesendet wurden, wird eine Paho-MQTT-Bibliothek für Python verwendet. Die Bibliothek wurde von der Eclipse Foundation entwickelt und ermöglicht eine einfache und schnelle Kommunikation im MQTT-Protokoll. Sie ist sehr gut für IoT-Anwendungen geeignet.

Durch die Verwendung von Paho-Mqtt können die Nachrichten an bestimmte Topic gesendet werden und der Client kann ein oder mehrere Topics abonnieren. Die Bibliothek bietet eine ereignisorientierte Programmierung, die auf spezifische Ereignisse wie Verbindungsaufbau oder Nachrichteneingang reagiert. Dank der Thread-Sicherheit eignet sich Paho-MQTT gut für Anwendungen, die in einer multithreaded Umgebung arbeiten.

Beim Start des Programms muss in der run()-Methode eine loop()-Funktion gestart werden. Sie sorgt dafür, dass Nachrichten empfangen, gesendet und Callback-Funktionen ausgeführt werden, wenn Nachrichten eintreffen oder Verbindungsereignisse auftreten.

1.4.3 Telegram

Telegram ist ein Cloud-basierter Instant-Messaging-Dienst. Er wurde 2013 von den Brüdern Durov entwickelt. Der Messenger zeichnet sich durch hohe Sicherheit, Flexibilität und Plattformunabhängigkeit aus. Telegram kann auf unterschiedlichen Geräten genutzt werden. Die Chats sowie die Nutzerdaten werden in einer Cloud gespeichert und in Echtzeit synchronisiert. Telegram unterstützt Einzel- und Gruppenchats, Sprach- und Videoanrufe, die Übertragung von Bildern, Videos oder anderen Dateien. Durch den Einsatz einer Ende-zu-Ende-Verschlüsselung in sogenannten SSecret Chats"gewährleistet Telegram eine hohe Datensicherheit. Die regulären Chats werden in der Cloud gespeichert und verschlüsselt übertragen.

Was Telegram besonders auszeichnet und bei Entwicklern beliebt macht, ist die Unterstützung von Bots. Bots sind automatisierte Programme, die über Telegram mit Nutzern interagieren können. Bots können zur Automatisierung von Aufgaben, zur Bereitstellung von Informationen oder zur Unterhaltung eingesetzt werden. Die offene API von Telegram ermöglicht es Entwicklern, ihre eigenen Bots zu erstellen und deren Funktionalität an spezifische Anforderungen anzupassen. Die neueste Version unterstützt auch die Nutzung von Webanwendungen direkt über den Bot.

Jeder Telegram-Nutzer kann einen Telegram-Bot erstellen. Die Bots werden vom Bot @BotFather verwaltet. Der Benutzer muss @BotFather aufrufen und dann einen neuen Bot erstellen. Dabei wird nach dem Namen und dem Benutzernamen gefragt, der mit "bot" enden muss. Anschließend wird ein Bot erstellt und diesem ein API-Token zugewiesen. Dieses Token wird im Programmcode verwendet, um den Zugriff auf den Bot zu ermöglichen.

1.4.4 Pyhton-Telegram-Bot

Für die Programmierung des Telegram-Bots wird eine `python-telegram-bot`-Bibliothek verwendet. Die Bibliothek bietet eine umfangreiche API zur Verwaltung der Bot-Funktionalität. Um den Bot zu starten, muss eine Applikation mit dem Token erstellt und auf Polling gesetzt werden.

Die Bibliothek unterstützt den Empfang und Versand von Nachrichten in verschiedenen Formaten. Entwickler können den Bot so konfigurieren, dass er auf bestimmte Nachrichten oder Befehle reagiert, indem sie "Handler"definieren, die das Verhalten des Bots steuern. Der Bot kann auch Nachrichten senden, ohne dass der Benutzer im Chat aktiv ist.

Ein zentrales Konzept der Bibliothek ist der CommandHandler. Mit Hilfe des CommandHandlers können Bots auf bestimmte Eingaben wie `/start`, `/help` oder benutzerdefinierte Befehle reagieren. Dies bietet eine einfache Möglichkeit, die Interaktionen des Bots zu steuern und verschiedene Funktionen zu implementieren. Der Befehl `/start` wird immer beim ersten Start des Bots vom neuen Benutzer aufgerufen.

Die Nachrichten, die der Bot empfängt oder versendet, werden asynchron bearbeitet. Das heißt, jede Funktion, die Nachrichten absendet oder auf die Nachrichten reagirt muss mit `async`-Befehl deklariert sein. Der Versand der Nachricht muss dann mit `await`-Befehl erwartet sein. An sonsten bekommt man ein Fehler T̈he Funktion was never awaited:

Die Bibliothek unterstützt Webhooks, die eine Kommunikation in Echtzeit ermöglichen. Webhooks ermöglichen es dem Bot, Nachrichten von Telegram direkt über HTTP-Requests zu empfangen, ohne den Server regelmäßig nach neuen Nachrichten abfragen zu müssen.

Außerdem bietet die Bibliothek eine Vielzahl von Interaktionen mit dem Benutzer. Es ist möglich eine Inline- oder Antworttastatur zu erstellen, sodass Nutzer vorgefertigte Antworten

auswählen können. Die Benutzerverwaltung ist auch in der Bibliothek vorhanden. Entwickler können Benutzerinformationen abrufen, speichern und auf diese Weise personalisierte Erfahrungen bieten.

Nach dem Aufbau der Applikation, muss die Polling-Funktion des Bots aufgerufen werden. Polling ist eine Methode, bei der der Bot kontinuierlich die Telegram-Server nach neuen Nachrichten abfragt. Der Bot sendet in regelmäßigen Abständen Anfragen an den Server, um zu prüfen, ob neue Nachrichten eingegangen sind. Dies ist ein einfacher Mechanismus, um den Bot zu betreiben, ohne auf Webhooks angewiesen zu sein.

2 Raspberry Pi

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. Das 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.

2.1 Einrichtung als Access Point

Für die Einrichtung des Raspberry Pi als WLAN Access Point wird das Softwarepaket `hostapd`[3] 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
```

[3]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

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 Änderungen 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.

2.2 Einrichtung des DNS- und DHCP-Servers

Der DNS[4]-Server übersetzt Domains in die IP-Adressen. Der DHCP[5]-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

interface=wlan0
bind-dynamic
domain-needed
bogus-priv
dhcp-range=192.168.1.100,192.168.1.110,255.255.255.0,12h

#End
```

- **interface** — definiert Interface, an welchem der DNS-Server fungiert.
- **bind-dynamic** — erlaubt die Verbindung nur mir exsistierenden Inrefaces.
- **domain-needed** — ignoriert DNS-Anfragen ohne Domänennamen.
- **bogus-priv** — die DNS-Anfragen aus lokalem Netzwerk werden nicht an Haupt-DNS-Server weitegeleitet.

[4]Domain Name System
[5]Dynamic Host Configuration Protocol

- **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

#End
```

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

2.3 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`[6] 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
   ```

   ```
   #Konfigurationsdatei für mosquitto

   pid_file /run/mosquitto/mosquitto.pid

   persistence true
   persistence_location /var/lib/mosquitto/

   log_dest file /var/log/mosquitto/mosquitto.log
   ```

[6] https://mosquitto.org/

```
include_dir /etc/mosquitto/conf.d

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
```

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.

2.4 Telegram Bot und Paho-Client

Bisher kann der Raspberry Pi einen WLAN-Access-Point zur Verfügung stellen und MQTT-Nachrichten empfangen. Es fehlt noch eine Schnittstelle zum Endbenutzer, die ebenfalls auf dem Raspberry Pi implementiert wird. Ich habe mich für ein Telegram-Bot entschieden. Dazu muss ein Verwaltungsmechanismus für die MQTT-Nachrichten entwickelt werden. Für diese Zwecke verwende iche ein Python-Bibliothek `"paho-mqtt"`.

Die Struktur ist wie folgendes aufgebaut(SPÄTER WEG, GESAMTSTRUKTUR WIRD AM ANFANG BESCHRIEBEN):

```
|-python-scripts
|
|--app.py
|--mqtt.py
|--tg-bot.py
```

2.4.1 Paho-MQTT

Die Implementierung des Paho-Clients erfolgt in der Datei `mqtt.py`. Folgende Bibliotheke(????) müssen importiert werden:

```
#mqtt.py

from paho.mqtt import client as mqtt_client
import threading
from queue import Queue
```

Um die Erstellung eines Objekts zu erlauben muss ein Klass MqttBroker deklariert werden. Der Konstruktor ist in der Funktion `__init__` definiert:

```
def __init__(self, queue, broker, port, topic, client_id):
  threading.Thread.__init__(self)
  self.queue = queue
  self.broker = broker
  self.port = port
  self.topic = topic

  self.client = mqtt_client.Client(client_id)
  self.client.on_connect = self.on_connect
  self.cleint.on_message = self.on_message
  self.client.on_disconnect = self.on_disconnect
  self.client.connect(self.broker, self.port)
```

Konstruktor-Parameter:

- **queue** – ein Queue-Objekt. Queue wird in der Startklasse app.py initialisiert. Erlaubt Threadsafe Synchronisation des MQTT-Broker und Telegram-Bots.

- **broker** – String. IP-Adresse des MQTT-Brokers. In meinem Fall `localhost` auf dem Raspberry Pi.
- **port** – Integer. Port, auf den der MQTT-Broker hört.
- **topic** – String-Array. Topics, zu dem sich der MQTT-Broker anmeldet. Kann ein oder mehrere Topic sein.
- **client_id** – String. Name des Clients.

Sowohl MqttBroker als auch Telegram-Bot verwenden eine Endlosschleife, um kontinuierlich Anfragen an den Server bzw. an den Broker zu senden. Damit die beiden Programme parallel laufen können, müssen sie in getrennten Threads gestartet werden. Die Threads werden direkt bei der Initialisierung des Klassenobjekts gestartet. Danach wird ein `mqtt_client`-Objekt mit übergebener Id initialisiert. Anschließend werden die Callback-Funktionen für ausgewählte Ereignisse deklariert und dem Client zugewiesen. Als letztes verbindet sich der Client mit dem Broker.

Nach der Initialisierung muss das Programm mit der Funktion `run()` gestartet werden.

```
def run(self):
  # mehrere Topic abonnieren
  for t in self.topic:
    self.client.subscribe(t)

  self.client.loop_start()
```

Die `loop_start()`-Funkltion startet eine Endlosschleife und fragt den MQTT-Broker stätig an den Nachrichten. Beim Nachrichteneingang wird die Funktion `on_message()` aufgerufen.

```
def on_message(self, client, userdata, msg):
  mes = str(msg.payload.decode('utf-8'))
  message = {'topic': msg.topic, 'text': mes}
  self.queue.put(message)
```

Als Parameter werden die Client- und Benutzerdaten sowie die Nachricht selbst übergeben. Die empfangene Nachricht wird zunächst dekodiert, um den Text in einem lesbaren Format zu erzeugen. Text und Topic der Nachricht werden in einem Dictionary gespeichert und an die Queue übergeben. Der Telegram-Bot kann nun dieses Dictionary problemlos und nahezu in Echtzeit empfangen.

2.4.2 Telegram-Bot

Für die Implementierung des Telegram-Bots wird eine Python-Bibliothek verwendet. Die Implementierung erfolgt in der Datei `tg_bot.py`. In dieser Datei wird die Klasse `Bot` deklariert, die von der `threading.Thread` erbt. Zunächst sind folgende Imports erforderlich:

```
from telegram import Update
from telegram.ext import filters, ApplicationBuilder, CommandHandler,
      MessageHandler, CallbackContext
import threading
from queue import Queue, Empty

class Bot(threading.Thread):
...
```

Dem Konstruktor muss ein Queue-Objekt und ein Telegram-Token übergeben werden. Das Token ermöglicht dem Programm den Zugriff auf den Telegram-Bot. In der Klasse `Bot` werden die Callback-Funktionen für die Reaktion auf empfangene Benutzerbefehle und MQTT-Nachrichten deklariert.

Der Bot wird mit dem Aufruf der `run()`-Funktion gestartet:

```
def run(self):
  self.application = ApplicationBuilder().token(self.token).build()
  # Händler für die Benutzerbefehle
  self.application.add_handler(CommandHandler('start', self.command_start))

  job_queue = self.application.job_queue
  if job_queue:
    job_queue.run_repeating(self.wait_mqtt_message, interval = 1)

  self.application.run_polling()
```

Zunächst muss die Applikation erstellt werden. Danach können die in den entsprechenden Funktionen deklarierten Handler den ausgewählten Ereignissen zugewiesen werden. Anschließend wird eine `job_queue` gesucht. Das `job_queue` ist ein Queue-Objekt und erlaubt den asynchronen Zugriff auf die Shared Queue. Wenn ein `job_queue` gefunden wurde, wird die Queue mit einem Intervall von 1 Sekunde nach neuen Nachrichten abgefragt. Zuletzt muss die Funktion `run_polling()` aufgerufen werden.

```
async def wait_mqtt_message(self, context: CallbackContext):
  try:
    mqtt_message = self.queue.get_nowait()
    await self.handle_mqtt_message(mqtt_message)
    self.queue.task_done()
  except Empty:
    # kein neuer Eingang in Queue
    pass
```

Die Funktion `wait_mqtt_message` sucht nach neuen Einträgen in der Queue. Die Funktion enthält einen Try-Except-Block. Wenn keine neuen Einträge gefunden werden, läuft das Programm ohne Unterbrechung weiter. Wird ein neuer Eintrag gefunden, so wird dieser in `mqtt_message` gespeichert und der Funktion `handle_mqtt_message` übergeben. In dieser Funktion wird die eingehende Nachricht verarbeitet und eine Meldung an den Bot ausgegeben.

3 ESP8266

3.1 Verbindung von ESP8266 und Arduino

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. Weiter unten ist der Programmcode für die Verarbeitung der Daten von Arduino zu sehen.

```
void readSerialData()
{
  if (Serial.available() > 0)
  {
    String readString = "";
    // Lese Daten aus Serial als String ab
    readString = Serial.readStringUntil('\n');

    if (sizeof(readString) > buss_serial)
    {
      return;
    }
    // String in char - Feld konvertieren
    char readSerialChar[readString.length() + 1];
```

```
    readString.toCharArray(readSerialChar, readString.length() + 1);
    // 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;
      // MQTT-Nachricht senden
      mqttClient.publish(topic, message);
    }
    else // kein : gefunden
    {
      Serial.print("FALSCHES FORMAT");
      return;
    }
  }
}
```

3.2 ESP8266 als Schnittstelle für die MQTT-Verbindung

In meinem Projekt verwende ich das 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. Das ESP8266 muss sich mit dem Zuganspunkt von Raspberry Pi verbinden und eine MQTT-Verbindung zu dem Broker erstellen. Außerdem muss das ESP8266 die Daten von Arduino erhalten.

3.2.1 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 **??**) zu sehen. Diese Funktion wird beim Start des Moduls in der `setup()`-Funktion aufgerufen.

```
void connect_wifi()
{
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".")
  }

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

```
  Serial.print(WiFi.localIP());
}
```

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

3.2.2 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 das 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()` eingepackt. Diese Funktion wird beim Start vom ESP8266 in der `setup()`-Funktion aufgerufen.

```
void connect_mqtt()
{
  Serial.print("\nConnecting ESP to MQTT Broker with IP: ");
  Serial.print(MQTT_BROKER_ADRRESS);
  while (!mqttClient.connected() & WiFi.status() == WL_CONNECTED)
  {
    if (mqttClient.connect(CLIENT_ID)) mqttClient.subscribe(SUBSCRIBE_TOPIC);
    else delay(1000);
  }
}
```

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.

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.

Die Verarbeitung der empfangenen Nachricht erfolgt in der `callback`-Funktion. 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());
}
```

Abbildungsverzeichnis