Internet Of Things mit dem mqtt-Protokoll

Handreichung zur Präsentation

Michael Dienert

8. Januar 2020

Inhaltsverzeichnis

1 Zur Entwicklung von MQTT

1.1 Für was steht die Abkürzung MQTT und von welcher Firma wurde MQTT entwickelt?

- In vielen Quellen wird *MQTT* als Abkürzung von ***Message Queuing Telemetry Transport*** beschrieben.
- E-Mail (smtp) basiert z.B. auf *Message Queuing*.
- Eine E-Mail wird zunächst der *Sendewarteschlange* hinzugefügt.
- Die Warteschlange (mailqueue) wird der Reihe nach abgearbeitet.
- Unterwegs wird die E-Mail auf jedem Zwischenknoten wieder in einer *Warteschlange* gespeichert, bis der Knoten bereit ist, sie weiterzuleiten.
- Hätte man kein *Message Queuing*, müsste man vor dem Senden sicherstellen, dass auf dem gesamten Übertragungspfad alle Knoten die Mail sofort weiterleiten können.
- *Message Queuing* kann man sich so vorstellen, als würden *Software-Anwendungen* mit einer Art E-Mail-Verfahren untereinander kommunizieren.

1.2 Verwendet MQTT überhaupt Message Queuing?

- Nein! Normalerweise verwendet MQTT **kein** Queuing!
- Warum heisst es dann so?
- MQTT wurde 1999 bei IBM von Andy Stanford-Clark (IBM) und Arlen Nipper (Fa. Arcom) entwickelt.
- Als Name des Protokolls wurde ***MQ Telemetry Transport*** verwendet, wobei das MQ sich auf das *Produkt* **IBM MQ** bezieht.

1.3 Standardisierung und Lizenzmodell

- Seit 2010 ist MQTT ein *offener Standard* (OASIS) und kann daher patent- und lizenzfrei verwendet werden.
- Mit der Standardisierung wird die Buchstabenfolge MQTT nicht mehr als Abkürzung betrachtet, sondern ist schlicht und einfach der *Name des Protokolls.*
- MQTT verwendet den TCP-Port **1883**. Möchte man verschlüsseln (SSL), wird Port 8883 verwendet.
- TLS/SSL ist Thema einer (evtl.) Anschlussfortbildung 2020.

2 Grundlagen von MQTT

2.1 Kommunikationsmodell

2.1.1 Klassisch: Server-Client

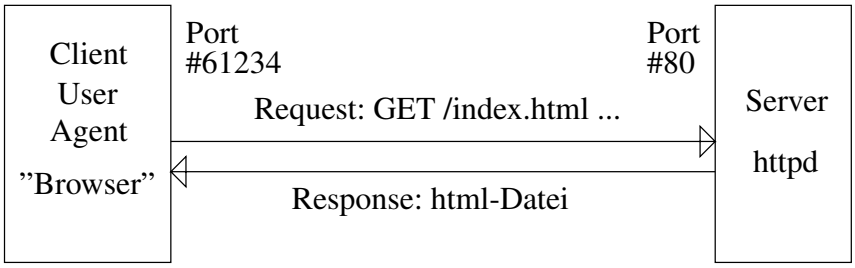


Abbildung 1: Server-Client bei http

2.1.2 Kommunikation über Broker (Makler)

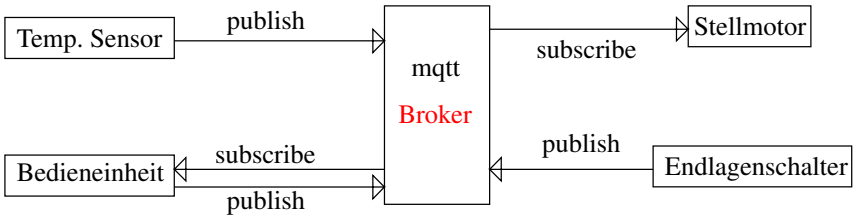


Abbildung 2: Architektur mit Broker (Makler)

- *keine* direkte Verbindung zwischen den Endpunkten (Z.B. Temperaturmessstelle und Temperaturanzeige) bei MQTT
- MQTT verwendet ein *Publish/Subscribe*-Muster: die Endpunkte werden dabei durch einen *Broker* (Makler) voneinander entkoppelt und kennen sich nicht.
- Der Broker filtert und speichert Meldungen der *Publisher* und verteilt sie korrekt an die *Subscriber*.
- Die Entkopplung ist *räumlich* aber auch *zeitlich*: Veröffentlichen und Abonnieren müssen nicht gleichzeitig stattfinden.
- Die Hauptlast der Meldungsverarbeitung liegt beim Broker. Die Endpunkte (Publisher/Subscriber) können daher auf Microcontrollern mit geringen Resourcen betrieben werden.

2.2 MQTT und TCP/IP

- MQTT ist eine Internet-Anwendung (Schicht 7).
- zwischen Client (Subscriber oder Publisher) und Broker wird eine TCP/IP - Sitzung aufgebaut.
- Beim Verbindungsaufbau meldet sich der Client mit einer Client-ID beim Broker an.
- Die TCP/IP-Sitzungen bleiben normalerweise ständig offen.

2.3 MQTT mit dem Softwarepaket `mosquitto`

- Wir brauchen: einen Broker, einen Subscriber und einen Publisher.
- Man kann nach Belieben diese 3 Instanzen auf einem einzelnen Host oder auf 2 oder 3 Rechnern laufen lassen.
- Alles was man benötigt, wird von den Debian-Paketen `mosquitto` und `mosquitto-clients` bereitgestellt. Zur Installation folgende Schritte in einem Linux-Terminal ausführen (zuerst Root-Rechte erlangen: Kommando `su` eingeben, Passwort: toor):

```
su
aptitude update
aptitude install mosquitto mosquitto-clients
exit #root-shell wieder verlassen
```

- Alternative: alle Übungen auf dem Raspberry-PI durchführen
- Durch die Installation des Pakets `mosquitto` wird auf dem jeweiligen Rechner ein Broker installiert und im Hintergrund gestartet.
- `mosquitto` ist quelloffen und wird von der *Eclipse Foundation* entwickelt.
- Umfangreiche Informationen zu den Kommandos erhält man durch Aufrufen der Handbuchseite. Beispiel:

```
man mosquitto_pub
```

- Das Kommando `man` gibt den Handbuchtext in einem sog. *Pager* aus. Dort kann man mit `/muster` nach Textmustern suchen und durch Drücken von n weitersuchen. Drücken von q beendet den Pager.

2.4 Einige Versuche mit `mosquitto`

- Veröffentlichen einer Nachricht. **-h** gibt den Hostnamen des Brokers an, **-t** ist das Thema (*topic*), **-m** die Meldung (*message*) und **-r** setzt das *retain-flag*:

```
mosquitto_pub -h localhost -t neubau/r229n -m mirIstSoHeiss -r
      #bzw.
mosquitto_pub -h localhost -t neubau/r229n -m mirIstSoHeiss
```

- Um ein Thema zu abonnieren, gibt man im Terminalfenster ein:

```
mosquitto_sub -h localhost -t neubau/r229n
```

  Statt den eigenen Broker zu nehmen, kann man auch einen Nachbarrechner mit dem entsprechenden Hostnamen (r229n-xxx) wählen.

- Um das Abonnement zu kündigen, stoppen Sie `mosquitto_sub` durch Drücken von Ctrl - c . (STRING- ©, 'ch, 'ch, 'ch...)

- Versuchen Sie durch Veröffentlichen und Abbonieren von Nachrichten herauszufinden, welchen Effekt das **Setzen** oder **Weglassen** des Retain-Flags (-r) hat.

- Neben Versuch und Irrtum lohnt auch ein Blick in die Handbuchseite vom Protokoll selbst:

```
man mqtt
```

3 Eigenschaften von MQTT

3.1 Wirkung des Retain-Flags

- Retain-Flag gesetzt: Broker speichert den publizierten Wert als *last known good value*.
- Subscriber meldet sich neu an: er erhält sofort den mit Retain-Flag gesetzten Wert.
- neue, aktuelle Messwerte ohne Retain treffen ein: Subscriber erhält aktuelle Daten.
- Löschen einer Retained-Meldung: Senden einer Nachricht mit

  1. gesetztem Retain-Flag
  2. entsprechendem Topic
  3. **leerer** Payload zu diesem Topic

```
mosquitto_pub -r -q 2 -h 10.10.0.254 -t qostest -m retainedMessage
mosquitto_pub -r -q 2 -h 10.10.0.254 -t qostest -m ""
```

3.2 MQTT Topics

- Um die Menge an Daten gut organisieren zu können, werden beim Veröffentlichen von Werten sog. *Topics* vergeben. Die Topics sind dabei ähnlich wie die Pfade in einem Dateisystem *hierarchisch* organisiert.
- Beispiele für Topics:

```
EG/bad/tempDS1820-005
EG/bad/humid4242
EG/wohnzimmer/temp0x721
garage/tor/sensorObenMS001
garage/tor/aufRel002
```

3.3 Topic-Wildcards

- Möchte man eine ganze Klasse von Topics abonnieren, kann man *Wildcards* einsetzen.
- *Single-Level-Wildcard*: **+** ⇒ eine beliebige, einzelne Hierarchieebene

```
+/flur/temperatur
```

- *Multi-Level-Wildcard*: **#** ⇒ ab einer bestimmten Hierarchie-Ebene alles darunter Liegende

```
EG/bad/#
```

  Das **#** - Zeichen **muss** dabei am Ende der Topic-Zeichenkette, nach einem Vorwärts-Schrägstich stehen!

3.3.1 Topic-Bezeichner

- keine unnötige, leere Hierarchieebene: **kein** Vorwärts-Schrägstrich am Anfang.
- **Keine Leerzeichen in Topics verwenden.** Das gilt allgemein auch für Dateinamen.
- **Nur ASCII-Zeichen verwenden!** Keine Umlaute, Sonderzeichen, nicht-druckbare UTF8-Zeichencodes! Ebenso bei Datei- und DNS-Namen.
- eindeutigen Client-Bezeichner (Client-ID) in den Topic einbetten. Hilft bei der Identifizierung des Subscribers.
- Kurze, prägnante Topic-Hierarchienamen um Header bei der Übertragung kompakt zu halten.
- Bei der Topic-Hierarchie auf spätere Erweiterbarkeit achten.

- Sensoren nicht zusammenfassen: pro Sensor ein eigenes Topic-Level am Pfadende.
- Testen Sie die Wirkungen der Wildcards und unterschiedlicher Topic-Hierarchien mit `mosquitto` (10min).
- mit welchem Topic kann man alle Meldungen auf dem Broker abonnieren?

```
mosquitto_sub -h -t +/#
```

- Wie kann man sich gleichzeitig die zugehörigen Topics anzeigen lassen?

```
mosquitto_sub -W 5 -v -h -t +/#
```

- was bewirkt der Schalter **-W 5** ?
- was wird hiermit ausgegeben:

```
mosquitto_sub -v -t '$SYS/#'
```

3.4 Quality of Service

- Es gibt 3 Qualitätsstufen bei MQTT:
- **At most once (Level 0)** Nachricht wird vom Sender genau einmal gesendet und dann gelöscht.
  - keine Übertragungskontrolle
  - keine Empfangsbestätigung
  - keine Übertragungswiederholung im Fehlerfall
- **At least once (Level 1)** Empfänger sendet Bestätigung
  - Sender speichert Nachricht bis Bestätigung eintrifft (PUBACK)
  - evtl. unerwünschte Mehrfachübertragung möglich
- **Exactly once (Level 2)** Es wird mit 4-Wege-Handshake sichergestellt, dass der Empfänger die Nachricht exakt einmal erhält.
  - nur einsetzen, wenn unerwünschte Mehrfachübertragung problematisch
- Broker trennt Kommunikationspartner
- Publisher und Subscriber können *verschiedene* QoS definieren
- D.h. Broker trennt auch evtl. unterschiedliche QoS beim Veröffentlichen und beim Abonnieren!

4 Sicherheit bei MQTT

4.1 Authentifizierung und Verschlüsselung

- Authentifizierung mit Benutzername und Passwort über Passwortdatei
- Schalter (`-u`) und Passwort (`-P`) bei `mosquitto_sub / mosquitto_pub` verwenden
- Aktivierung über Eintrag in der Konfigurationsdatei des Brokers (`mosquitto.conf`) notwendig
- Passwörter und Benutzernamen werden in einer Textdatei angelegt. Die Passwörter werden erst im nächsten Schritt verschlüsselt. Beispiel für Dateiaufbau:

```
joe:sample
lalah:hathaway
```

- Passwörter in der Textdatei verschlüsseln:

```
mosquitto_passwd -U pwdfile
```

- Achtung! Nicht doppelt verschlüsseln!
- Benutzer hinzufügen oder löschen:

```
mosquitto_passwd -b pwdfile alf mad #alf hinzufügen
mosquitto passwd -D pwdfile alf #alf löschen
```

- Konfigurationsdatei `/etc/mosquitto/mosquitto.conf` editieren (root-Rechte!)

  Hier ein Beispiel (Details siehe `man mosquitto.conf`):

```
#authentifizierung erzwingen
allow_anonymous false
password_file /etc/mosquitto/pwdfile
```

- Erstellen Sie eine Passwortdatei, verschlüsseln diese und aktivieren die Authentifizierung in der Konfigurationsdatei.
- Konfiguration neu einlesen (root-Rechte):

```
/etc/init.d/mosquitto reload
```

- Testen Sie nun die Authentifizierung beim Veröffentlichen und Abonnieren von Meldungen.

4.2 Zugangskontrolle mit ACLs

- Access Control Lists erlauben die Einstellung, welcher Benutzer welche Topics abonnieren oder veröffentlichen darf.
- Aktivierung mit folgender Zeile in `mosquitto.conf`:

```
acl_file /etc/mosquitto/aclfile
```

  Weitere Details wie immer: `man mosquitto.conf`

- Aufbau einer einfachen ACL-Datei:

```
# fuer alle subscriber ohne benutzernamen
# mosquitto.conf anpasssen: allow_anonymous true
topic read oeffentlich/#

# kontrolle auf basis des benutzernamens.
user alfred
topic +/#

user joe
topic music/#
```

4.3 Übungen zu ACLs

- Erstellen Sie eine ACL, passen die Konfiguration des Brokers an und lesen diese neu ein (reload).
- Veröffentlichen und abonnieren Sie Meldungen mit den entsprechenden Topics, um die Wirkung der ACL zu testen.

5 mqtt-Testament

5.1 Last Will and Testament

- Wird die Verbindung zu einem mqtt-Client unabsichtlich unterbrochen, kann man mit der Last Will and Testament (LWT)-Eigenschaft von mqtt die anderen Teilnehmer informieren.
- Jeder Client kann eine Last-Will-Meldung festlegen, wenn er sich mit dem Broker verbindet.
- Die Last-Will-Meldung ist eine normale mqtt-Nachricht mit Topic, Retain-Flag, QoS und dem Nachrichtentext (LWT) selbst.

- Erkennt der Broker, dass die Verbindung zum Client unabsichtlich unterbrochen wurde, sendet er dessen LWT an alle Clients, die das Last-Will-Topic abonniert haben.
- Meldet sich der Client ordnungsgemäss ab, löscht der Broker dessen LWT.

6 Webserver, Broker und Arduino-Clients

6.1 Anwendungsbeispiel

- Unterrichtseinheit zu mqtt in der Fachschule
- Reaktion der Schüler: schön und gut, aber wir wollen ein Smart-Home-Beispiel sehen!
- Idee: über eine Webanwendung werden Meldungen von und zu einem Broker gesandt
- Wegen der einfachen Hardware-Erweiterbarkeit werden Arduino-Uno mit Ethernet- und Relais-Shield als Clients verwendet.
- Problem mit doppelter Port-Belegung von Ethernet- und Relais-Shield: auf dem Relais-Shield müssen die Port-Pins D4 und D12 abgeschnitten und auf Ports D3 und D9 gebrückt werden.

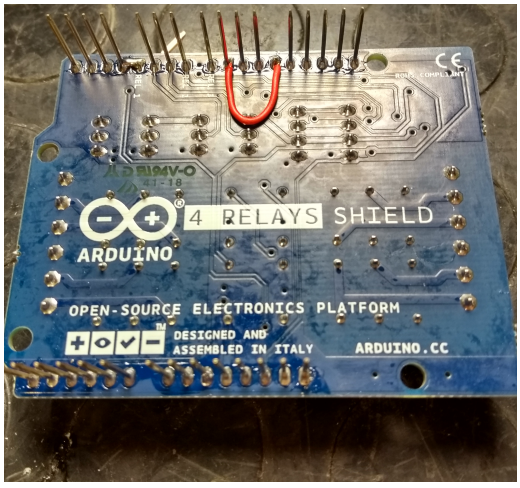


Abbildung 3: Ports umlegen auf Relais-Shield

- Als Temperatur-Sensor kommt ein 1-Wire-Sensor (DS1820) zum Einsatz.
- Verdrahtung des DS1820 evtl. mit Proto-Shield
- Eingänge A3 und A4 für Endlagenschalter
- Der Broker und der Webserver laufen auf einem Raspberry-PI mit neuestem Raspbian-Buster-Linux.

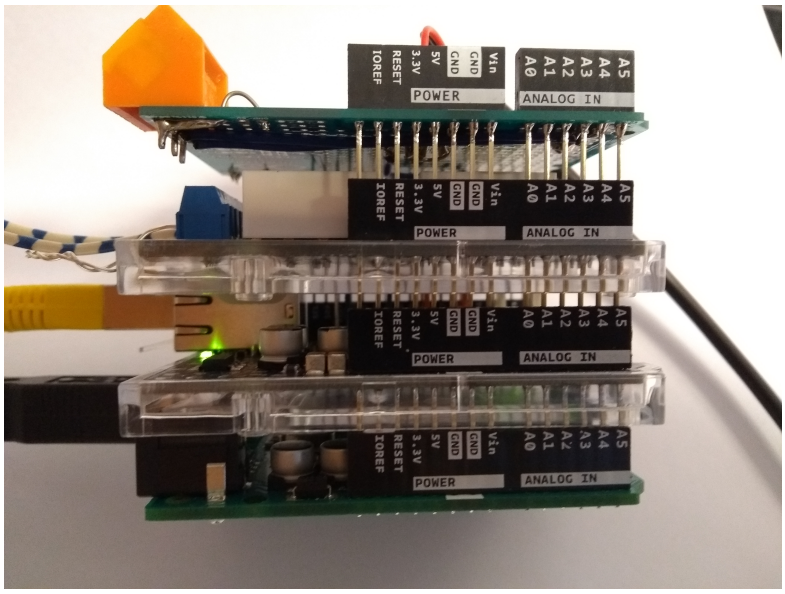
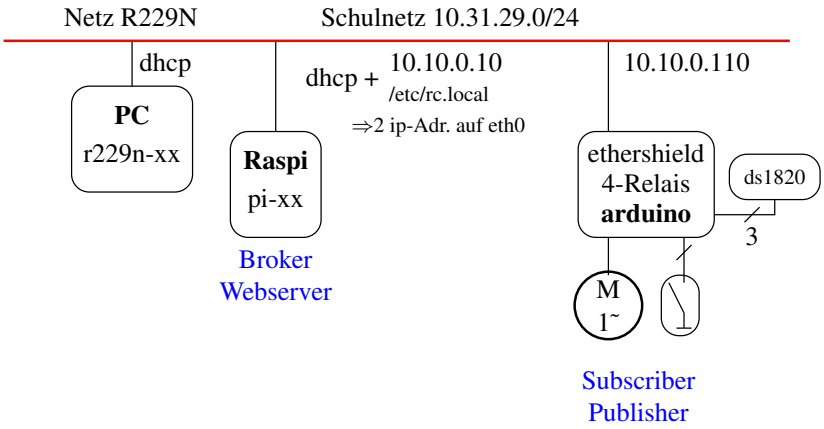


Abbildung 4: Arduino mit div. Shields

6.2 Aufbau im Schulnetz mit PC, Raspi und Arduino



6.3 MQTT auf Raspberry-PI

- Auf den Raspberry-PI ist das mosquitto-Paket aus den Quellen installiert worden:

```
https://mosquitto.org/files/source/mosquitto-1.6.3.tar.gz
```

-
  - Herunterladen
  - Entpacken
  - Übersetzen (make)
  - Installieren (make install)
  - Library-Pfade setzen

  - Start-/Stopp-Skript erstellen (/etc/init.d/mosquitto) und Links in den Runlevels setzen

- fertiges Raspberry-Pi-Image:

```
http://dt.wara.de/fobiMQTT/raspberryImage/
```

- mit dd auf eine SD-Karte schreiben. Unbedingt auf richtiges Ziellaufwerk achten! `/dev/sde` ist hier nur Beispiel! Das Kommando setzt root-Rechte veraus.

```
dd if=busterMqtt.img of=/dev/sde bs=4M
sync
```

- Eine ausführlichere Anleitung gibt es hier, Kapitel 2:

```
http://dt.wara.de/fobiMQTT/pdf/raspbianInstallieren.pdf
```

- SD Karte beschreiben oder fertige Karte verwenden
- Raspi mit dem Schulnetz verbinden und über Micro-USB-Kabel mit Spannung versorgen
- mit dem Kommando ssh von einem PC-Terminal aus eine Sitzung zum Raspi aufbauen:

```
ssh pi@mosquittoXX #XX=raspi-nr, password=raspberry
ip a # ip adressen auflisten lassen
mosquitto_sub ...
mosquitto_pub ...
```

- die Raspies haben 2 IP-Adressen
  1. IP-Adresse aus dem Schulnetz, wird per DHCP bezogen
  2. feste Adresse, die über Kommando in der Datei `/etc/rc.local` beim Starten gesetzt wird

6.4 Netzwerke zwischen Arduino, Raspberry und PC

- Die Raspberries beziehen automatisch eine IP Adresse vom dhcp-Schulserver und bekommen auch automatisch einen DNS-Eintrag (`mosquittoXX.wara.de`).
- D.h. die Raspberries und die PCs befinden sich alle im Netz 10.31.29.0/24.
- Über einen Eintrag in der Datei `/etc/rc.local` wird auf den Raspberries eine feste Adresse aus dem Netz 10.10.0.x/24 konfiguriert.

- Überprüfen Sie, ob nun Arduino und einer Ihrer Raspberries sich im gleichen Netz befinden. Evtl. müssen Sie dazu das Arduino-Programm und die Datei `/etc/rc.local` auf dem Raspberry anpassen (danach Neustart oder ip manuell vergeben).

6.5 Arduino und benötigte Libraries

- Folgende Libraries werden auf dem Arduino benötigt:
  1. PubSubClient: Installation über pubsubclient-master.zip
  2. OneWire: Installation über Bibliotheksverwalter
  3. DallasTemperature: Installation über Bibliotheksverwalter

6.6 Programmierübung auf dem Arduino

- Verbinden Sie den Arduino mit dem PC (USB) und dem Schulnetz (ethernet)und starten Sie die Arduino-IDE (Eingabe von `arduino` im Terminal)
- Öffnen Sie die Datei `mossieClient.ino`. Die gibt's hier:

```
http://dt.wara.de/fobiMQTT/ArduinoProjects/mossieClient.ino
```

- Fügen Sie die benötigten Libraries hinzu. PubSubClient gibt es hier:

```
http://dt.wara.de/fobiMQTT/ArduinoProjects/pubsubclient-master.zip
```

6.7 Programmierübung auf dem Arduino

- Legen Sie 2 Pullup-Widerstände (3k3) an die Eingänge A3 und A4.
- Passen Sie das Programm so an, dass der Portzustand von A3 und A4 mit passendem Topic und passender Meldung alle 2 s an den Broker auf Ihrem Raspberry veröffentlicht wird.
- Abonnieren Sie das Topic auf dem Raspberry und Testen die Übertragung durch Setzen der Pegel an A3 oder A4 auf 0V.

6.8 Webserver mit apache2 und CGI

- Neben dem mqtt-Broker ist auf den Raspberries der apache-Webserver konfiguriert.
- Es ist ein *Virtual-Host* `mosquitto.wara.de` konfiguriert:

```
/etc/apache2/sites-available/mosquitto.wara.de.conf
```

- Der `ServerName mosquitto.wara.de` muss angepasst werden, da die Teilnehmer-Raspies durchnummerierte Namen haben.
- Wie beim Broker auch, muss der Server nach Änderungen der Konfiguration neu gestartet werden.

```
<VirtualHost *:80>

    #muss auf den tatsaechlichen dns-namen des raspberries angepasst werden
    #ServerName mosquitto<nummer>.wara.de
    #z.B. ServerName mosquitto14.wara.de

    ServerName mosquitto.wara.de

    ServerAdmin webmaster@localhost

    #verzeichnis, in dem die dateien dieses virtualHosts liegen
    DocumentRoot /var/www/mosquitto

    #dateien mit der endung .bin werden vom webserver ausgefuehrt und ihre
    #standardausgabe wird als http-response zum web-client geschickt
    #das modul cgi muss mit
    #         a2enmod cgi
    #aktiviert worden sein (im raspi-image schon erledigt)

    <Directory /var/www/mosquitto>
       Options +ExecCGI
       AddHandler cgi-script .bin
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

</VirtualHost>
```

6.9 CGI-Programm auf Raspberry

- Im Pfad

  ```
  /home/pi/development/c/mosquittoCGI
  ```

  liegt das CGI-Programm im Quelltext.
- Um das URL-Decoding kümmert sich die Library `urlCoding`.
- Compilieren und Installieren des CGI-Programms

  ```
  gcc garLicht.c -L. -lurlCoding -L /usr/local/lib/ -lmosquitto -o garLicht.bin
  sudo cp garLicht.bin /var/www/mosquitto
  ```
- zu Testzwecken kann das CGI-Programm auch direkt gestartet werden. Vorher muss die Umgebungsvariable `QUERY_STRING` auf den gewünschten Wert gesetzt werden.