Softwareentwurf und Anwendungen verteilter Systeme

Digital Product Design and Development B.A.

Semester 3

Hochschule für Gestaltung Schwäbisch Gmünd

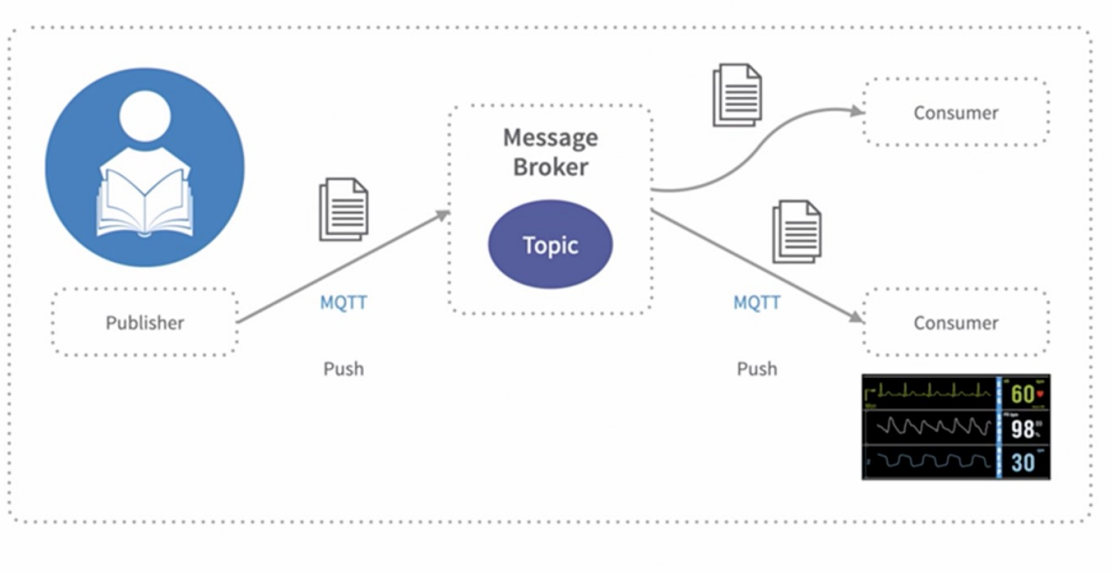
Dozent: Yannick Schiele

H f G

Hochschule für Gestaltung
Schwäbisch Gmünd

Netzwerkprotokolle

TCP, HTTP, MQTT, etc.



MQTT

- Message Queuing Telemetry Transport
- Von IBM und Eurotech entwickelt
- auf Umgebungen mit niedriger Bandbreite und hoher Latenz spezialisiert → ideales Protokoll für Machine-to-Machine-Kommunikation und IoT
- Funktioniert nach Publish/Subscribe-Modell
- OSI Schicht 7, meistens aufbauend auf TCP/IP

https://www.linkedin.com/learning/search?keywords=mqtt&u=82266650

```
const mqtt = require('mqtt‘)
const host = 'broker.emqx.io‘
const port = '1883‘
const clientId = `mqtt_{username}`
const connectUrl = `mqtt://${host}:${port}`
const client = mqtt.connect(connectUrl, {
            clientId,
            clean: true,
            connectTimeout: 4000,
            username: 'emqx‘,
            password: 'public‘,
            reconnectPeriod: 1000,
})
const topic = '/nodejs/mqtt‘
client.on('connect', () => {
            console.log('Connected‘)
            client.subscribe([topic], () => {
              console.log(`Subscribe to topic '${topic}‘`)
  })
  client.publish(topic, 'nodejs mqtt test', { qos: 0, retain: false },
(error) => {
            if (error) {
              console.error(error)
            }
  })
})
client.on('message', (topic, payload) => {
            console.log('Received Message:', topic, payload.toString())
})
```

MQTT Example

```
{
"Herausgeber": "Xema",
"Nummer": "1234-5678-9012-3456",
"Deckung": 2e+6,
"Waehrung": "EURO",
"Inhaber":
  {
        "Name": "Mustermann",
        "Vorname": "Max",
        "maennlich": true,
        "Hobbys": ["Reiten", "Golfen", "Lesen"],
        "Alter": 42,
        "Kinder": [],
        "Partner": null
  }
}
```

https://www.json.org/json-de.html

JSON

- JavaScript Object Notation
- besteht aus Schlüssel-Wert-Paaren, die auch weiter verschachtelt werden können
- Getrennt durch einen Doppelpunkt (Schlüssel : Wert)
- Datentypen:
  - Nullwerte - null
  - Bool – true/false
  - Zahl – 0-9
  - String – “ “
  - Array – [ ]
  - Objekt – { }

Übung

Sensorwerte in JSON packen und auf Konsole ausgeben

JSON Konsolen Input soll Aktor steuern

Implementierung

Eines Webservers auf dem ESP

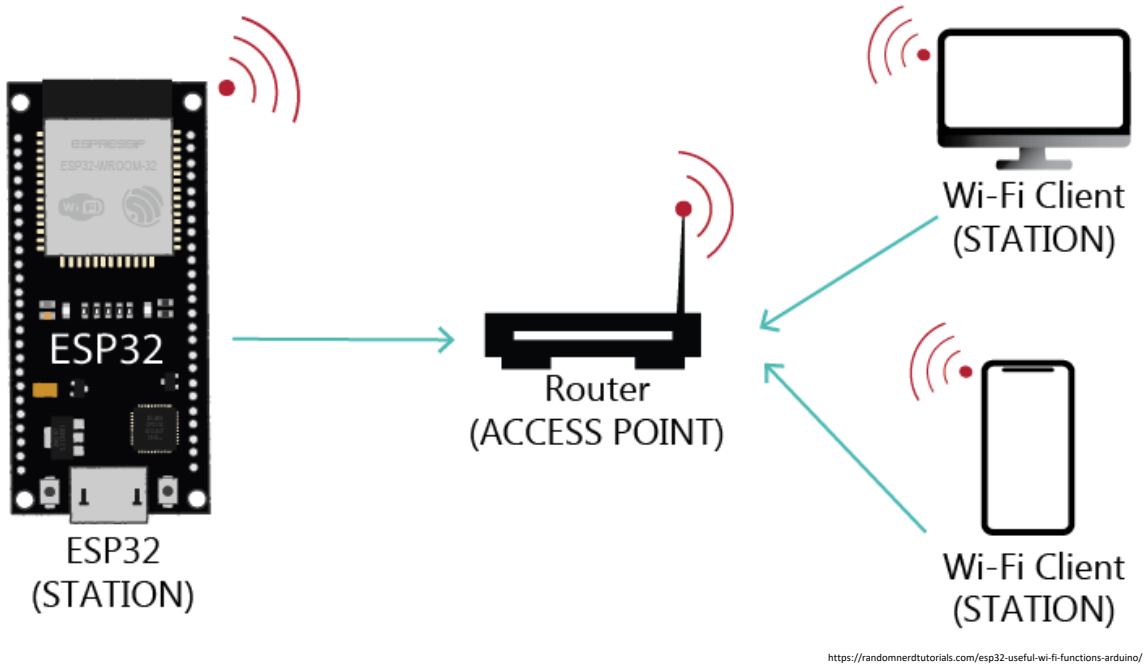
Wi-Fi Modes

Der ESP kann als Wi-Fi Station, Access Point oder beides fungieren.

Um den Wi-Fi-Modus einzustellen, wird `WiFi.mode()` verwendet und der gewünschte Modus als Argument übergeben

`WiFi.mode(WIFI_STA)`	station mode: der ESP32 verbindet sich mit einem Access Point
`WiFi.mode(WIFI_AP)`	access point mode: Geräte können sich mit dem ESP verbinden
`WiFi.mode(WIFI_STA_AP)`	Kombination aus beiden vorherigen Konfigurationen

https://randomnerdtutorials.com/esp32-useful-wi-fi-functions-arduino/



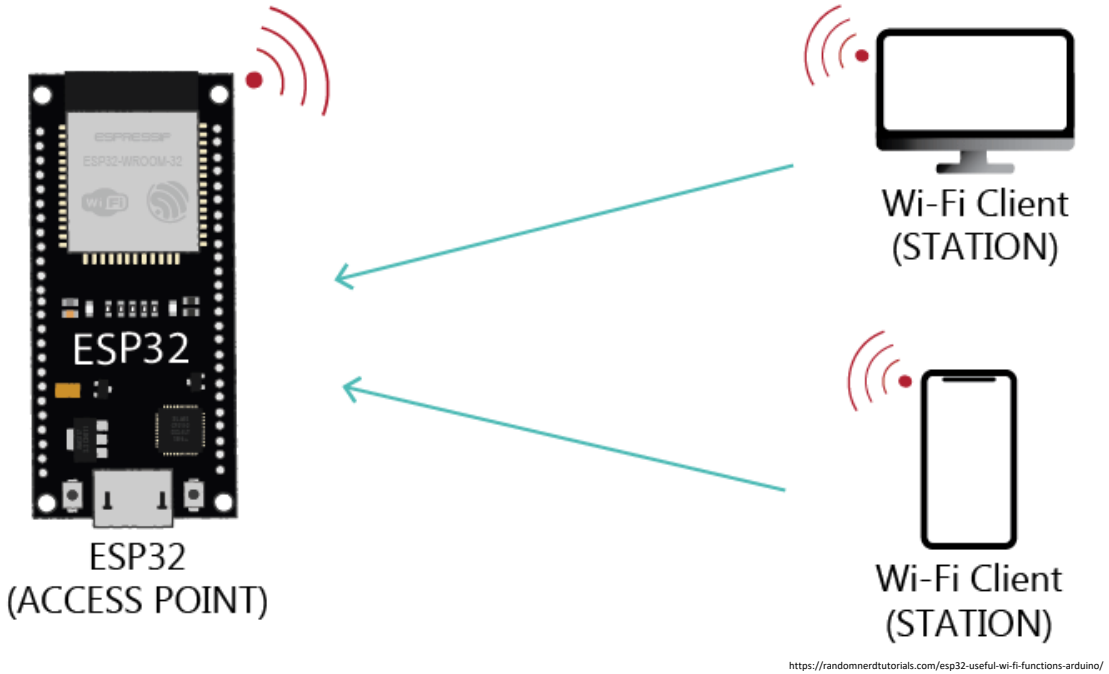
https://randomnerdtutorials.com/esp32-useful-wi-fi-functions-arduino/

Wi-Fi Station

Wenn der ESP als Wi-Fi-Station eingestellt ist, kann er sich mit anderen Netzwerken verbinden (wie ein Router)

In diesem Szenario weist der Router dem ESP eine eindeutige IP-Adresse zu, die für Kommunikation im netzt genutzt werden kann

Wenn der Router mit dem Internet verbunden ist, kann der ESP Informationen aus dem Internet abrufen, wie z. B. Daten aus APIs (z. B. Wetterdaten), etc.



https://randomnerdtutorials.com/esp32-useful-wi-fi-functions-arduino/

Access Point

- Wenn das ESP-Board als Access Point eingestellt ist, kann jedes Gerät mit Wi-Fi-Fähigkeiten eine Verbindung herstellen,
- Es wird ein eigenes Wi-Fi-Netzwerk erstellt, mit dem sich Wi-Fi-Geräte in der Nähe verbinden können
- Gute Möglichkeit wenn mehrere ESP32-Geräte alleine miteinander kommunizieren wollen
- Soft-AP (Soft Access Point): keine Verbindung mit dem Internet vorhanden

```
WiFi.mode(WIFI_AP_STA);
```

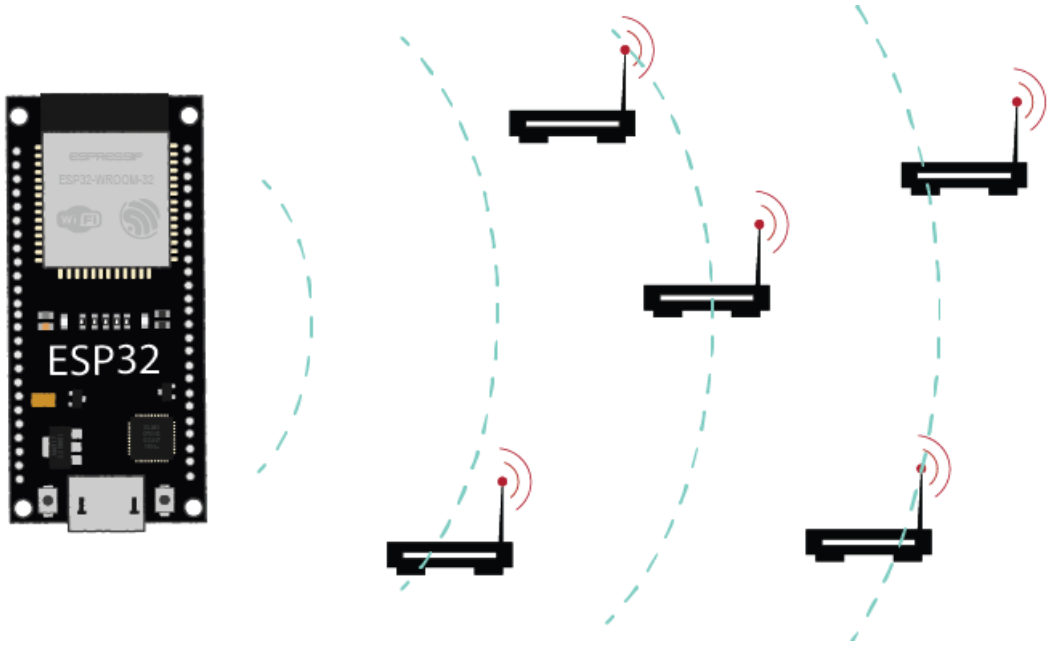
Wi-Fi Station + Access Point

Der ESP kann gleichzeitig als Wi-Fi-Station und Access Point eingestellt werden.

Dazu muss der Modus auf WIFI_AP_STA eingestellt werden

Wi-Fi Scan

Häufiges Beispiel in Bibliotheken



https://randomnerdtutorials.com/esp32-useful-wi-fi-functions-arduino/

Verbinden zu einem Netzwerk

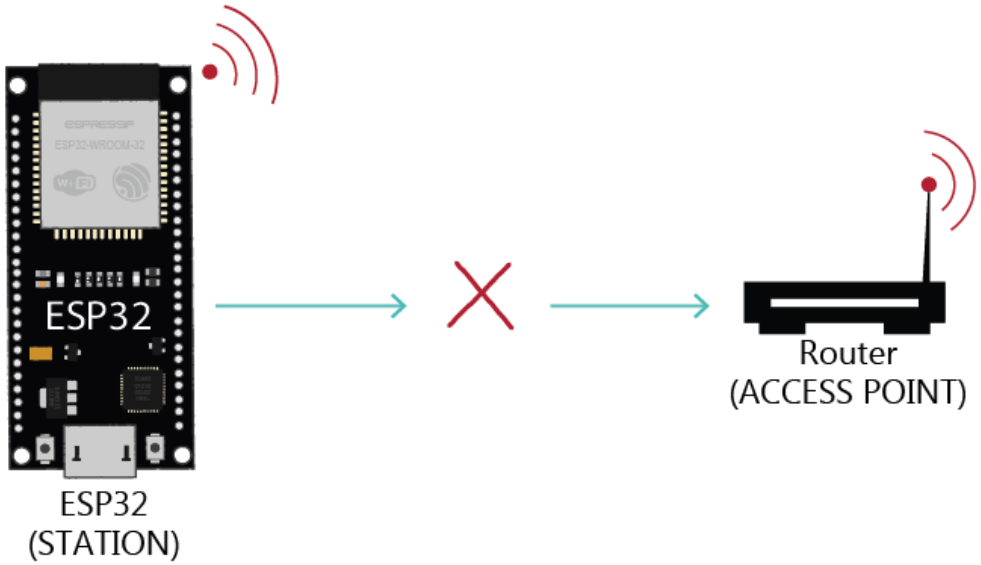
```
void initWiFi() {
 WiFi.mode(WIFI_STA);
 WiFi.begin(ssid, password);
 Serial.print("Connecting to WiFi ..");

 while (WiFi.status() != WL_CONNECTED) {
        Serial.print('.‘);
        delay(1000);
 }
 Serial.println(WiFi.localIP());
}
```

Wert	Bedeutung
`WL_IDLE_STATUS`	temporärer Status, der beim Aufruf von WiFi.begin() zugewiesen wird
`WL_NO_SSID_AVAIL`	wenn keine SSID verfügbar sind
`WL_SCAN_COMPLETED`	der Netzwerkscan ist abgeschlossen
`WL_CONNECTED`	wenn eine Verbindung zu einem WiFi-Netzwerk besteht
`WL_CONNECT_FAILED`	wenn die Verbindung bei allen Versuchen fehlschlägt
`WL_CONNECTION_LOST`	wenn die Verbindung unterbrochen wird
`WL_DISCONNECTED`	wenn die Verbindung zu einem Netzwerk getrennt wurde

Wi-Fi Connection Status

Um den Status der Wi-Fi-Verbindung abzufragen, kann `WiFi.status()` verwendet werden

Diese gibt einen der folgenden Werte zurück, die den Konstanten in der Tabelle entsprechen



Wi-Fi-Netzwerk Verbindung trennen

Um die Verbindung zu einem zuvor verbundenen Wi-Fi-Netzwerk zu trennen, verwendet man:

```
WiFi.disconnect()
```

Um sich nach einem Verbindungsabbruch wieder mit dem Wi-Fi zu verbinden, kann man mit `WiFi.reconnect()` versuchen, sich wieder mit dem zuvor verbundenen Access Point zu verbinden

```
0  ARDUINO_EVENT_WIFI_READY               < ESP32 WiFi ready
1  ARDUINO_EVENT_WIFI_SCAN_DONE                < ESP32 finish scanning AP
2  ARDUINO_EVENT_WIFI_STA_START                < ESP32 station start
3  ARDUINO_EVENT_WIFI_STA_STOP                 < ESP32 station stop
4  ARDUINO_EVENT_WIFI_STA_CONNECTED            < ESP32 station connected to AP
5  ARDUINO_EVENT_WIFI_STA_DISCONNECTED         < ESP32 station disconnected from AP
6  ARDUINO_EVENT_WIFI_STA_AUTHMODE_CHANGE      < the auth mode of AP connected by ESP32 station changed
7  ARDUINO_EVENT_WIFI_STA_GOT_IP               < ESP32 station got IP from connected AP
8  ARDUINO_EVENT_WIFI_STA_LOST_IP              < ESP32 station lost IP and the IP is reset to 0
9  ARDUINO_EVENT_WPS_ER_SUCCESS       < ESP32 station wps succeeds in enrollee mode
10 ARDUINO_EVENT_WPS_ER_FAILED        < ESP32 station wps fails in enrollee mode
11 ARDUINO_EVENT_WPS_ER_TIMEOUT       < ESP32 station wps timeout in enrollee mode
12 ARDUINO_EVENT_WPS_ER_PIN           < ESP32 station wps pin code in enrollee mode
13 ARDUINO_EVENT_WIFI_AP_START                < ESP32 soft-AP start
14 ARDUINO_EVENT_WIFI_AP_STOP                 < ESP32 soft-AP stop
15 ARDUINO_EVENT_WIFI_AP_STACONNECTED         < a station connected to ESP32 soft-AP
16 ARDUINO_EVENT_WIFI_AP_STADISCONNECTED      < a station disconnected from ESP32 soft-AP
17 ARDUINO_EVENT_WIFI_AP_STAIPASSIGNED        < ESP32 soft-AP assign an IP to a connected station
18 ARDUINO_EVENT_WIFI_AP_PROBEREQRECVED       < Receive probe request packet in soft-AP interface
19 ARDUINO_EVENT_WIFI_AP_GOT_IP6              < ESP32 ap interface v6IP addr is preferred
19 ARDUINO_EVENT_WIFI_STA_GOT_IP6             < ESP32 station interface v6IP addr is preferred
20 ARDUINO_EVENT_ETH_START               < ESP32 ethernet start
21 ARDUINO_EVENT_ETH_STOP                < ESP32 ethernet stop
22 ARDUINO_EVENT_ETH_CONNECTED           < ESP32 ethernet phy link up
23 ARDUINO_EVENT_ETH_DISCONNECTED        < ESP32 ethernet phy link down
24 ARDUINO_EVENT_ETH_GOT_IP              < ESP32 ethernet got IP from connected AP
19 ARDUINO_EVENT_ETH_GOT_IP6             < ESP32 ethernet interface v6IP addr is preferred
```

Wi-Fi Events

- Alternativ können WiFi Events verwendet werden, um zu erkennen, dass die Verbindung unterbrochen wurde
- Der ESP kann die folgenden Wi-Fi-Ereignisse verarbeiten
- Mit Wi-Fi Events ist es nicht notwendig, den Wi-Fi Status ständig zu überprüfen
- Wenn ein bestimmtes Ereignis eintritt, ruft es automatisch die entsprechende Funktion auf

```
onStationModeConnected (std::function< void(const
WiFiEventStationModeConnected &)>)

onStationModeDisconnected (std::function< void(const
WiFiEventStationModeDisconnected &)>)

onStationModeAuthModeChanged (std::function< void(const
WiFiEventStationModeAuthModeChanged &)>)

onStationModeGotIP (std::function< void(const
WiFiEventStationModeGotIP &)>)

onStationModeDHCPTimeout (std::function< void(void)>)

onSoftAPModeStationConnected (std::function< void(const
WiFiEventSoftAPModeStationConnected &)>)

onSoftAPModeStationDisconnected (std::function< void(const
WiFiEventSoftAPModeStationDisconnected &)>)
```

Wi-Fi Reconnect Event

Übung

Sensorwerte (als JSON) an Webserver übertragen und darstellen

Webserver soll nach dem Start die SSID und Passwort übergeben werden

Input auf dem Webserver soll den Aktor steuern