

CoAP Client

Gruppe 6: Oliver Hahn, Sebastian Broede
und Pascal Seiz

Kurs: Verteilte Systeme
Semester: SoSe 2022

Inhalt

1. Theoretische Grundlagen CoAP

- 1.1 Basisdaten
- 1.2 Interaktionsmodell
- 1.3 Nachrichtenmodell
- 1.4 URI
- 1.5 Message Format

2. Externe Softwarekomponenten und Lizenzen

- 2.1 TCP/IP Stack: lightweight IP (lwIP)
- 2.2 Netzwerkbibliothek: Mongoose
- 2.3 Real-Time Operating System: FreeRTOS

3. Projekt „CoAP Client: Sensor to Actuator“

- 3.1 Projektziele
- 3.2 Projektumsetzung
- 3.3 Praktische Demonstration

1. Theoretische Grundlagen CoAP

1. Theoretische Grundlagen

1.1 Basisdaten

- CoAP kurz für Constrained Application Protocol
- Von der Internet Engineering Task Force (IETF) und der Constrained RESTful Environments Working Group (CoRE) entwickeltes Web-Transfer-Protokoll
- Für Geräte mit beschränkten Ressourcen entworfen. Bsp.:
 - Kabellose Sensoren, Industrie 4.0 und IoT-Geräte
 - Geräte mit wenig Rechenleistung, Speicher oder geringem Energieverbrauch
- Basiert auf den Funktionsprinzipien von REST mit Request-/Response-Format
- Typischer Einsatz in Machine-To-Machine (M2M) Kommunikation

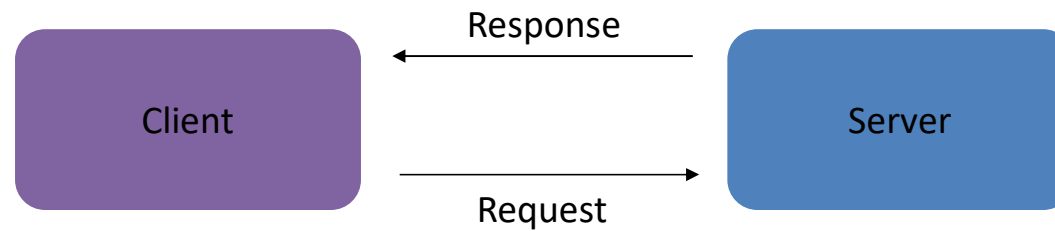
1. Theoretische Grundlagen

1.1 Basisdaten

- Verwendet UDP als Transportprotokoll
 - Header beinhaltet weniger zusätzliche Informationen zur Übertragung
 - Besitzt kleineren Header (4 Byte) um Bandbreite zu sparen
 - 1024 Byte große Payload
- Durch die Ähnlichkeit mit HTTP und REST ist eine einfache Interoperabilität über einen Proxy mit HTTP möglich
- Nutzlasten (payload) lassen sich sowohl im JSON als auch im XML-Format transportieren
- Unterstützt Transportverschlüsselung, Uniform Resource Identifier (URI) und asynchronen Nachrichtenaustausch

1. Theoretische Grundlagen

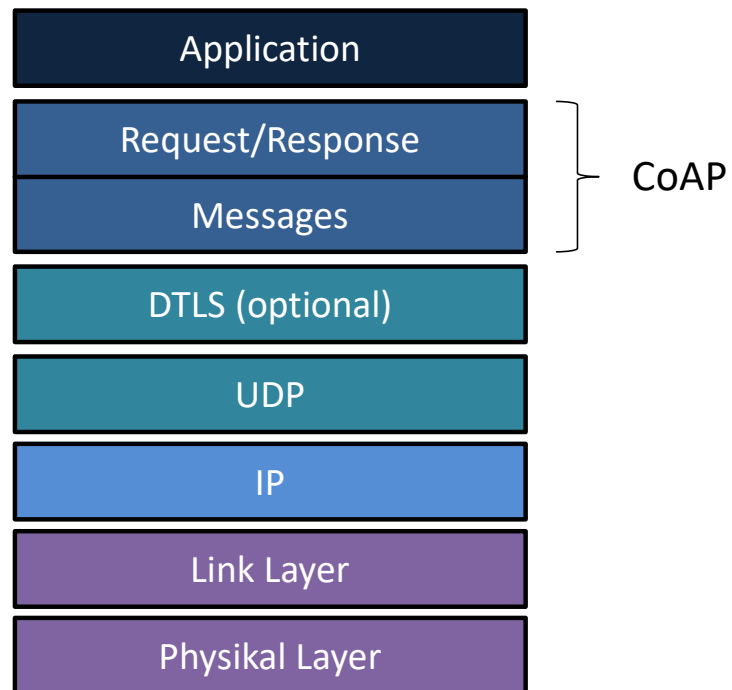
1.2 Interaktionsmodell



- CoAP Interaktionsmodell ist identisch zu HTTP
- Client sendet Request mit Method Code an Server
- Server antwortet mit Response Code an Client

1. Theoretische Grundlagen

1.2 Interaktionsmodell – Osi-Modell



1. Theoretische Grundlagen

1.3 Nachrichtenmodell – Method Codes

Client teilt Server mithilfe von Method Codes seine Absicht mit:

GET

- Client möchte eine Ressource abfragen

POST

- Client möchte eine Ressource anlegen

PUT

- Client möchte eine Ressource aktualisieren

DELETE

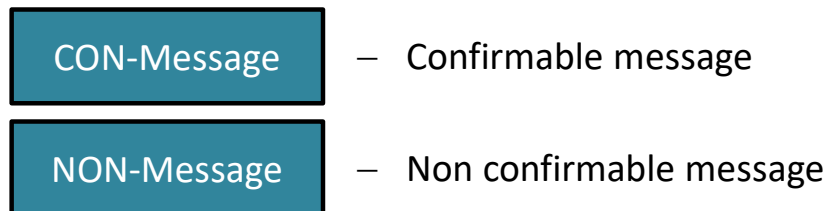
- Client möchte eine Ressource löschen

1. Theoretische Grundlagen

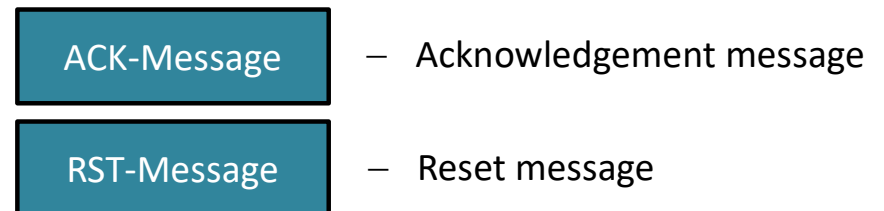
1.3 Nachrichtenmodell - Nachrichtentypen

Bei jedem CoAP Nachrichtenaustausch unterscheidet man zwischen 4 Nachrichtentypen:

Request-Nachrichten



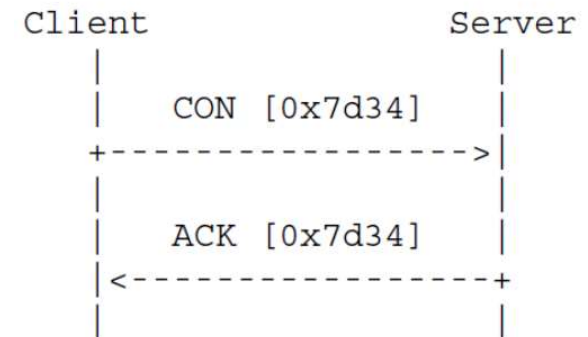
Response-Nachrichten



1. Theoretische Grundlagen

1.3 Nachrichtenmodell - CON-Message

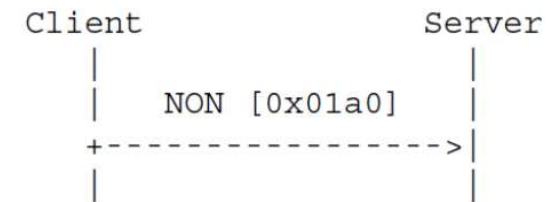
- kurz für „Confirmable message“
- Der Client sendet eine CON-Message mit einer Message-ID an den Server
- Der Client sendet wiederholt seine Nachricht, bis sich der Server mit einer entsprechenden ACK-Message und Message-ID zurückmeldet
- Kann der Server die Nachricht nicht verarbeiten antwortet er mit einer RST-Message



1. Theoretische Grundlagen

1.3 Nachrichtenmodell - NON-Message

- kurz für „Non confirmable message“
- Der Client sendet eine NON-Message mit einer Message-ID an den Server
- Der Client verlangt keine Bestätigung durch den Server mit einer entsprechenden ACK-Message und Message-ID zurückmeldet
- Kann der Server die Nachricht nicht verarbeiten antwortet er mit einer RST-Message



1. Theoretische Grundlagen

1.4 Uniform Resource Identifier (URI)

CoAP URI = “coap” “//” host [“:” port] path-abempty [“?” query]

Default port: 5683

CoAPs URI = “coaps” “//” host [“:” port] path-abempty [“?” query]

Default port: 5684

1. Theoretische Grundlagen

1.5 Message Format – CoAP Message

Header (4 Bytes)							
Token (if any, TKL Bytes 0 - 8)							
Options (if any)							
1	1	1	1	1	1	1	1
Payload (if any)							

Bits:	0	1	2	3	4	5	6	7	8 -> 15	16 -> 23	24 -> 31
Byte 0 -> 3	Ver		T		TKL			Code		Message ID	

Ver = CoAP version number

T = Message Type (siehe Kapitel 1.3)

Code = Request/Response Code (siehe Kapitel 1.3)

TKL = Tokenlength

Message ID = Message ID

1. Theoretische Grundlagen

1.5 Message Format – Message Codes

Method codes		Response codes Success		Response codes Client error	
0.00	Empty	2.01	Created	4.00	Bad request
0.01	GET	2.02	Deleted	4.01	Unauthorized
0.02	POST	2.03	Valid	4.02	Bad option
0.03	PUT	2.04	Changed	4.03	Forbidden
0.04	DELETE	2.05	Content	4.04	Not found

2. Externe Softwarekomponenten und Lizenzen

2. Externe Softwarekomponenten und Lizenzen

2.1 TCP/IP Stack: lightweight IP (lwIP) – Basics

- Entwickelt für eingebettete Systeme
- Ressourcenschonende Implementation eines TCP Stacks
- Unterstützt **UDP** als Transport Protokoll
- Umfasst Funktionen für IP, IPv6, TCP, DHCP, DNS, etc.
- Verwendung mit und ohne Betriebssystem möglich

2. Externe Softwarekomponenten und Lizenzen

2.1 TCP/IP Stack: lightweight IP (lwIP) – Lizenz

Version 1.4.1

Modifizierte BSD Lizenz

- Berkeley Software Distribution-Lizenz von der „Universität of California, Berkeley“ verfasst
- Weiterverbreitete Software-Quelltexte müssen den Copyright-Vermerk und Haftungsklausel enthalten
- Weiterverbreitete kompilierte Exemplare müssen den Copyright-Vermerk und Haftungsklausel in der Dokumentation und/oder anderen Materialien (Programm-Code) enthalten
- Werbematerialien die Eigenschaften oder die Benutzung erwähnen, müssen auf die Entwickler der Software hinweisen (siehe erster Punkt).

2. Externe Softwarekomponenten und Lizenzen

2.1 TCP/IP Stack: lightweight IP (lwIP) – Implementierung

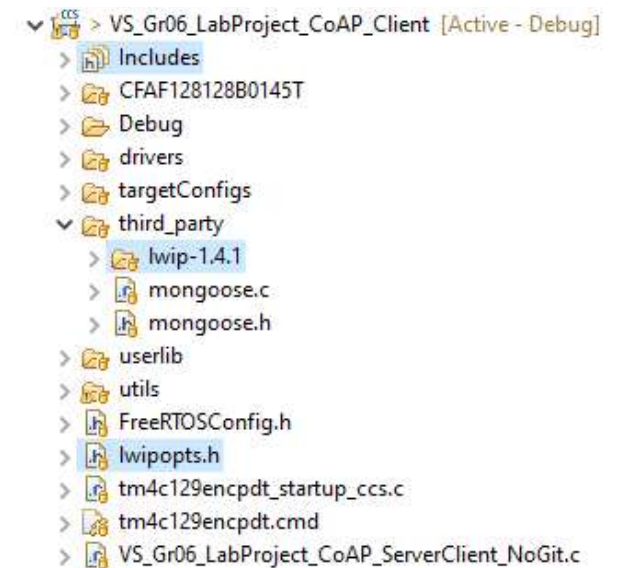
- lwIP in der Main initialisieren

```
// Initialize the lwIP library, using DHCP.  
lwIPInit(g_ui32SysClock, pui8MACArray, 0, 0, 0, IPADDR_USE_DHCP);
```

- Aktuelle IP-Adresse im Interrupthandler abrufen

```
void lwIPHostTimerHandler(void)  
{  
    uint32_t ui32NewIPAddress;  
  
    // Get the current IP address.  
    ui32NewIPAddress = lwIPLocalIPAddrGet();  
    ...  
}
```

- lwIP Bibliothek in das Projekt einbinden



2. Externe Softwarekomponenten und Lizenzen

2.2 Netzwerkbibliothek: Mongoose – Basisdaten

- Eventgesteuerte Networking Library für C/C++
- Cross-platform Support: **FreeRTOS**, Linux/Unix, Windows, Android und MacOS
- Unterstützt HTTP, MQTT, CoAP, TCP/UDP und Websockets
- Detaillierte Dokumentation und Beispiele
- Einfache Integration in Projekte

2. Externe Softwarekomponenten und Lizenzen

2.2 Netzebibliothek: Mongoose – Lizenz

Version 6.11

GPLv2 Lizenz

- Von der „Free Software Foundation“ (FSF) verfasst
- Sublizenzen müssen dieselben Lizenzbedingungen wie die GPLv2 Lizenz verwenden
- Bei rein privatem Vertrieb ist keine Offenlegung des Quellcodes vorgeschrieben
- Bei kommerziellem Vertrieb ist die Offenlegung des Quellcodes vorgeschrieben

2. Externe Softwarekomponenten und Lizenzen

2.2 Netzbibliothek: Mongoose – Implementierung

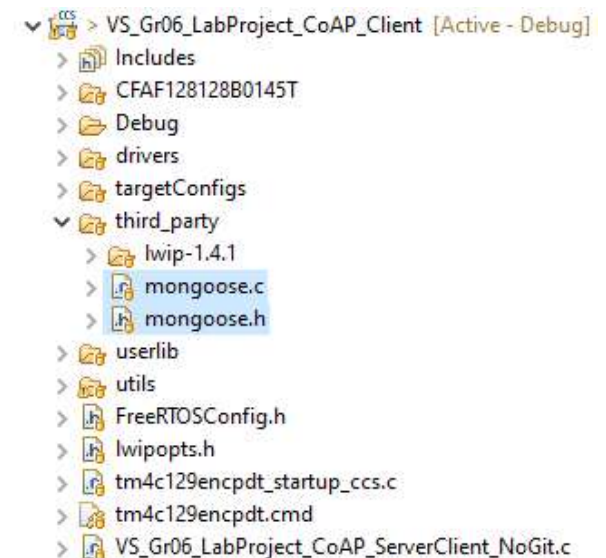
- Event Manager initialisieren

```
// Mongoose Eventmanager      // Initialize Mongoose
struct mg_mgr mgr;           mg_mgr_init(&mgr, 0);
```

- Verbindung aufbauen

```
// Mongoose Connection
struct mg_connection *nc;
// Connect to server
nc = mg_connect(&mgr, s_default_address, coap_handler);
```

- Mongoose Bibliothek in das Projekt einbinden



2. Externe Softwarekomponenten und Lizenzen

2.2 Netzbibliothek: Mongoose – Implementierung

- CoAP Event-Handler mit der Verbindung verknüpfen

```
// set COAP Protocol  
mg_set_protocol_coap(nc);
```

- Poll-Funktion in einer Schleife mit Verzögerung aufrufen. Diese ruft bei neuen Verbindungen oder bei Datenaustausch den Event-Handler auf.

```
// set COAP Protocol  
mg_set_protocol_coap(nc);  
  
while(1)  
{  
    mg_mgr_poll(&mgr, 0);  
    vTaskDelay(pdMS_TO_TICKS(100));  
}
```

2. Externe Softwarekomponenten und Lizenzen

2.2 Netzbibliothek: Mongoose – Implementierung

- CoAP Event-Handler des Clients: Anfrage an den Server

```
void coap_send_get(struct mg_connection *nc, char *uri_path, uint16_t msg_id)
{
    struct mg_coap_message cm;
    uint32_t res;

    memset(&cm, 0, sizeof(cm));

    cm.msg_id = msg_id;
    cm.msg_type = MG_COAP_MSG_CON;
    cm.code_class = MG_COAP_CODECLASS_REQUEST;
    cm.code_detail = METHOD_CODE_GET;

    mg_coap_add_option(&cm, COAP_OPTION_URI_PATH, uri_path, strlen(uri_path));

    UARTprintf("\nSending CON with GET...\n");
    res = mg_coap_send_message(nc, &cm);
    if (res == 0)
    {
        UARTprintf("Sent GET with msg_id = %d\n", cm.msg_id);
    }
    else
    {
        UARTprintf("Error: %d\nmsg_id = %d\n", res, cm.msg_id);
    }
    mg_coap_free_options(&cm);
}
```

2. Externe Softwarekomponenten und Lizenzen

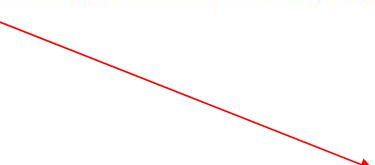
2.2 Netzbibliothek: Mongoose – Implementierung

- CoAP Event-Handler des Clients: Antwort des Servers

```
void coap_handler(struct mg_connection *nc, int ev, void *ev_data)
{
    struct mg_coap_message *incoming = (struct mg_coap_message *) ev_data;
    switch (ev)
    {
        case MG_EV_COAP_ACK:
        {
            UARTprintf("Server send ACK with msg_id = %d\n", incoming->msg_id);
            coap_parse_ack(incoming, nc);
            break;
        }
        case MG_EV_COAP_RST:
        {
            UARTprintf("Server RST\n");
            break;
        }
        case MG_EV_CLOSE:
        {
            UARTprintf("Server closed connection\n");
            break;
        }
    }
}

void coap_parse_ack(struct mg_coap_message *cm, struct mg_connection *nc)
{
    UARTprintf("\nPayload: %s\n", cm->payload.p);
    uint32_t brightness = atoi(cm->payload.p);

    uint16_t msg_id = cm->msg_id + 1; //increase msg_id to identify the new message;
    ...
}
```



2. Externe Softwarekomponenten und Lizenzen

2.3 Real-Time Operating System: FreeRTOS – Basisdaten

- Echt-Zeit Betriebssystem für eingebettete Systeme
- Unterstützt großes Spektrum an Prozessorarchitekturen
- Umfangreiche Dokumentation
- Umfangreiche Summe an Bibliotheken für unterschiedlichste Zwecke
- **Beispiel-Implementation** von Herrn Böck bereitgestellt

2. Externe Softwarekomponenten und Lizenzen

2.3 Real-Time Operating System: FreeRTOS – Lizenz

Version 10.4.6

MIT Lizenz

- Von dem „Massachusetts Institute of Technology“ (MIT) verfasst
- Keinerlei Beschränkung bezüglich des Quellcodes (kopieren, modifizieren, veröffentlichen, sublizensieren, etc.)
- Der „MIT open-source license text“ sollte sich in jeder Kopie oder in jedem größeren Ausschnitt der Software befinden

2. Externe Softwarekomponenten und Lizenzen

2.3 Real-Time Operating System: FreeRTOS – Implementierung

- RTOS Tasks erstellen

```
// Create new task
xTaskCreate(vTaskDisplay, (const portCHAR *)"displaytask", configMINIMAL_STACK_SIZE, NULL, 1, NULL);
xTaskCreate(mongooseClientTask, (const portCHAR *)"mongooseClientTask", configMINIMAL_STACK_SIZE, NULL, 1, NULL);
xTaskCreate(mongooseSendingTask, (const portCHAR *)"mongooseSendingTask", configMINIMAL_STACK_SIZE, NULL, 1, NULL);
```

- Beispiel-Implementation eines Tasks: Sendet alle 2 Sekunden eine GET-Anfrage an den Server

```
void mongooseSendingTask(void *parameters)
{
    vTaskDelay(pdMS_TO_TICKS(7000));
    uint16_t msg_id = 0;

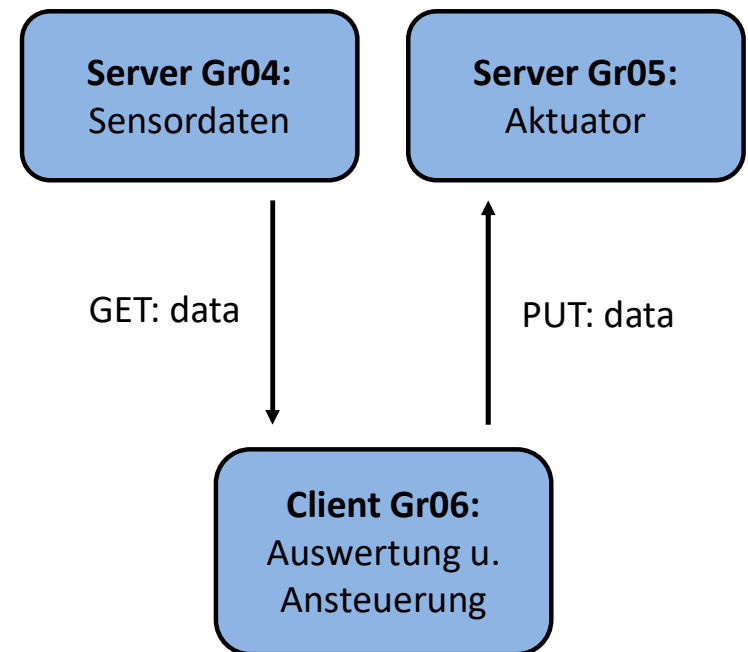
    while (1)
    {
        coap_send_get(nc, uri_path1, msg_id);
        msg_id++;
        vTaskDelay(pdMS_TO_TICKS(2000));
    }
}
```

3. Projekt „CoAP Client: Server to Actuator“

3. Projekt „CoAP Client: Server to Actuator“

3.1 Projektziele

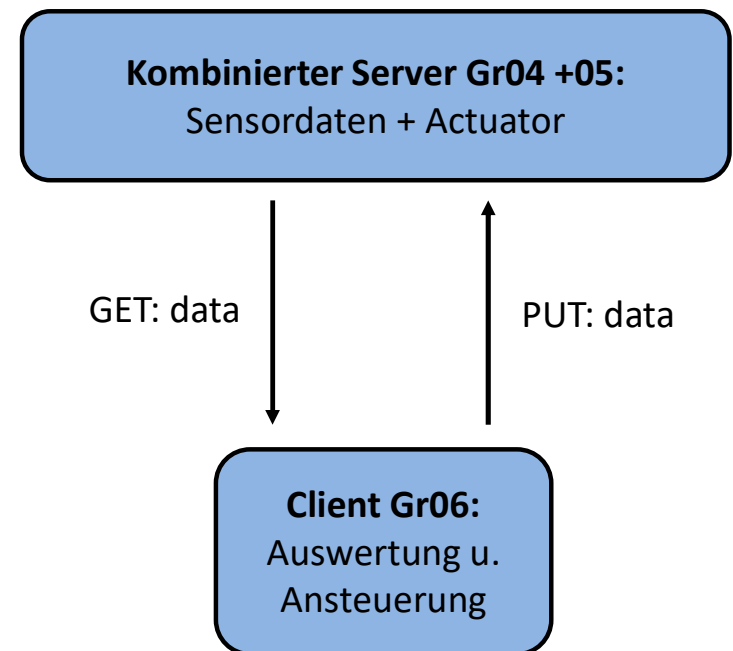
- Erstellung eines CoAP Clients
- Geeignete Bibliotheken auswählen und implementieren
- Verschlüsselte Datenübertragung einrichten
- Mithilfe der Server von Gruppe 4 und 5 Kommunikation aufbauen
- Sensordaten von Gruppe 4 auswerten und damit Actuator von Gruppe 5 ansteuern



3. Projekt „CoAP Client: Server to Actuator“

3.2 Projektumsetzung

- Erstellung eines CoAP Clients ☒
- Geeignete Bibliotheken auswählen und implementieren ☒
- Verschlüsselte Datenübertragung einrichten
 - Implementierung von DTLS nicht geschafft ☒
- Mit Servern von Gruppe 4 und 5 Kommunikation aufbauen ☒
 - Es standen leider nur 2 µC mit fester IP-Adresse zur Verfügung, Weshalb die beiden Server kombiniert wurden
- Sensordaten auswerten und Actuator ansteuern ☒
 - Als Actuator wurde das LCD-Display des Booster Packs verwendet
 - Server von Gruppe 4 musste erweitert werden



3. Projekt „CoAP Client: Server to Actuator“

3.1 Praktische Demonstration

- Client und Server kommunizieren nach dem Starten automatisch
- Kein Postman oder Copper notwendig
- Server kann jedoch ebenfalls PUT- und GET-Nachrichten von Postman und Copper verarbeiten

Weiter geht's mit der Laborübung