universität innsbruck

Increasing throughput of server applications by using asynchronous techniques: A case study on CoAP.NET

Philip Wille

Bachelor Thesis

Supervisor:
Dr. Michael Felderer
Department of Computer Science
Universität Innsbruck

Innsbruck, October 12, 2021

Contents

1 Einleitung

Heutzutage haben Webdienste einen hohen Stellenwert im Internet. Sie sind häufig ein essenzieller Bestandteil von Anwendungen und werden durch eine *Representation State Transfer* (REST)-Schnittstelle für Anwender und Entwickler angeboten. Auch das Internet of Things (IoT) bekommt mit fortschreitender Zeit eine wichtigere Rolle in der Entwicklung von Anwendungen, wie zum Beispiel Hausautomatisierung oder smarte Energieverwaltung.

Um die REST-Architektur auch für eingeschränkte Geräte anzubieten, wurde mit *Constrained RESTful Environments* (CoRE) eine geeignete Form gebildet. Somit können solche Geräte, wie zum Beispiel 8-Bit-Mikrocontroller mit begrenztem Arbeitsspeicher und Readonly-Memory, und Netzwerke, wie zum Beispiel *IPv6 over Low-Power Wireless Area Networks (6LoWPANs)*, solche Architekturen verwenden und realisieren. Damit eine Fragmentierung von Nachrichten in solchen Netzwerken so wenig wie möglich auftritt, wird im Constrained Application Protocol (CoAP) ein so niedriger Nachrichten-Overhead wie möglich angestrebt.

Mit CoAP wurde die Entwicklung eines generischen Webprotokolls für die speziellen Anforderungen dieser eingeschränkten Umgebungen, insbesondere mit Hauptaugenmerk auf Energie-, Gebäudeautomatisierungs- und andere Machine-to-Machine (M2M) Anwendungen. Dabei sollte CoAP keine komprimierte Abwandlung von HTTP sein, sondern vielmehr eine Submenge von HTTP mit Verwendung von REST, die für M2M-Szenarien optimiert ist. Somit könnte CoAP leicht dazu verwendet werden, einfache HTTP-Schnittstelle in ein kompaktes Protokoll überzuführen, jedoch bietet CoAP Funktionen die speziell für Machine-to-Machine Anwendungen eine Verwendung finden. Diese Funktionen sind:

- Eingebaute Entdeckung von, im Netzwerk, angebotenen Services und Ressourcen.
- Mutlicast-Unterstützung.
- Asynchronen Nachrichtenaustausch.

1.1 Constrained Application Protocol (CoAP)

Das Constrained Application Protocol, abgekürzt CoAP, ist ein Internetprotokoll, das speziell für M2M Anwendungen, wie zum Beispiel smarte Energieverwaltung oder Hausautomatisierung (Internet of Things (IoT)), entwickelt wurde. Dabei bietet das Protokoll ein REST-ähnliches Interface für Mikrocontroller mit angeschlossenen Aktoren oder Sensoren (*constrained nodes*) oder auch drahtlose Sensornetze (*constrained networks*) an. Die sogenannten *nodes* besitzen meist einen angeschlossenen 8-Bit-Mikrocontroller, der nur eine kleine begrenzte Menge an Arbeitsspeicher (RAM) und Readonly-Memory (ROM) beinhaltet. Bei *contstrained networks*, wie z.B. *IPv6 over Low-Power Wireless Area Networks (6LoWPANs)*, spielt die hohe Paketverlustrate und der für solche

Netzwerke typische Datendurchsatz von wenigen 10 kbit/s eine große Rolle. CoAP ist durch den RFC 7252 [2] spezifiziert.

Dabei bietet das *Constrained Application Protocol* ein Interaktionsmodell für Anfragen und Antworten zwischen den, in der Anwendung definierten, Endpunkten an. Auch ist eine eingebaute Entdeckung von im Netzwerk angebotenen Services und Ressourcen im Protokoll definiert. Zusätzlich werden Hauptbestandteile des Internets, wie *Unique Resource Identifiers* (URIs) und *Internet Media Types* (zum Beispiel *application/json*).

Eine Unterstützung von *Multicast* ist gegeben, wie auch ein sehr geringer Mehraufwand in der Datenübertragung. Dabei wurde Wert darauf gelegt, es einfach für eingebettete Systeme zu halten.

Zur Datenübertragung nutzt CoAP das User Datagram Protocol (UDP). Dieses unterscheidet sich zu Transmission Control Protocol (TCP) in den folgenden Punkten:

- kein Sitzungsaufbau von Sender zu Empfänger (Handshake).
- Stellt nicht sicher, ob alle Pakete beim Empfänger eintreffen.
- Geht ein Paket verloren, wird dieses nicht erneut versendet.

Die folgenden Funktionen sind ein essenzieller Bestandteil von CoAP:

- Erfüllung von M2M Anforderungen in eingeschränkten Umgebungen.
- UDP-Verbindung mit optionaler Zuverlässigkeit, die Unicast- und Multicast Anfragen unterstützt.
- Asynchroner Nachrichtenaustausch.
- Niedriger Mehraufwand durch veränderten Header und niedrige Komplexität des Parsings.
- URI- und Content-Typen-Support.
- Einfache Proxy- und Caching-Unterstützung.
- Zustandsloses HTTP-Mapping das die Entwicklung von Proxys erlaubt, die den Zugriff auf CoAP Ressourcen über HTTP auf einheitliche Weise ermöglichen, oder für einfache HTTP-Schnittstellen, die alternativ über CoAP realisiert werden können.
- Sicherheitsmechanismen durch das Anbinden von *Datagram Transport Layer Security* (DTLS).

Begriffe in CoAP

Um den Kontext innerhalb des *Constrained Application Protocols* zu verstehen, werden nachfolgend die wichtigsten Begriffe in CoAP kurz erklärt:

- Endpunkt (Endpoint):
  - Ein Endpunkt lebt auf einen "Knoten". Ein Knoten ist vergleichbar mit dem Begriff "Host", der vorwiegend in Internetstandards Erwähnung findet.
  - Ein Endpunkt wird durch Multiplexing-Informationen auf der Transportschicht identifiziert, die eine UDP-Portnummer und eine Sicherheitszuordnung enthalten könnte.
- Ursprungsserver (Origin Server):
  - Der Server, auf dem sich eine bestimmte Ressource befindet oder erstellt werden soll.
- Bestätigende Nachricht (Confirmable Message):
  - Einige Nachrichten benötigen eine Bestätigung des Empfängers. Diese Nachrichten werden als *bestätigt* behandelt.
  - Falls keine Pakete während der Übertragung verloren gingen, wird für jede Nachricht, die bestätigt werden muss, exakt eine Nachricht des Typs *Acknowledgement* oder *Reset*.
- Nicht bestätigende Nachricht (Non-confirmable Message):
  - Als Gegensatz zu bestätigenden Nachrichten gibt es auch Nachrichten die nicht bestätigt werden müssen.
  - Dies trifft auf Nachrichten zu, die für bestimmte Anwendungsanforderungen häufiger wiederholt werden müssen, wie zum Beispiel wiederholtes Lesen eines Sensors.
- Bestätigungsnachricht (Acknowledgement Message):
  - Eine solche Nachricht bestätigt den Empfang einer bestätigenden Nachricht. Eine Bestätigungsnachricht sagt nicht aus, ob die Anfrage, die mit einer bestätigenden Nachricht versendet wurde, erfolgreich war oder nicht.
  - Jedoch enthält die Bestätigungsnachricht auch eine sogenannte *Piggybacked Response*.
- Rücksetzende Nachricht (Reset message):
  - Diese Nachricht sagt aus, dass eine spezifische Nachricht (*Confirmable* oder *Non-confirmable*) empfangen wurde, jedoch einige Teile des Nachrichtenkontextes fehlt, um die richtig zu bearbeiten.

  - Dieses Verhalten tritt auf, wenn der zu empfangende Endpunkt neu gestartet hat und somit den Zustand vergessen hat, der zur vollständigen Interpretation der Nachricht nötig ist.
  - Ein absichtliches Provozieren einer rücksetzenden Nachricht *Reset Message*, zum Beispiel durch das Senden einer leeren bestätigenden Nachricht (*Empty Confirmable Message*), kann als eine kostengünstige Prüfung der Funktionsfähigkeit eines Endpunktes verwendet werden - vergleichbar mit einem *Ping*.

- Piggybacked Response:
  - Eine *Piggybacked Response* ist direkt in eine *Acknowledgement* (ACK) Nachricht inkludiert, die gesendet wird, um den Empfang der Anfrage für diese Antwort zu bestätigen.

- Separate Antwort (Separate Response):
  - Wenn eine *Confirmable* Nachricht mit einer Anfrage mit einer *Empty* Nachricht quittiert wird (z.B. weil der Server die Antwort nicht sofort hat), wird eine separate Antwort in einem separaten Nachrichtenaustausch gesendet.

- Leere Nachricht (Empty Message):
  - Eine Nachricht mit dem Code 0.00. Die Nachricht ist weder eine Anfrage noch eine Antwort. Es beinhaltet nur den 4-Byte-langen Kopf (*header*).

Nachrichtenübertragung

CoAP nutzt zur Nachrichtenübertragung UDP, um den Austausch von Nachrichten asynchron ausführen zu können. Dies wird dadurch erreicht, dass eine weitere Schicht auf UDP aufbauend eingefügt wird (siehe Bild 1). Diese Schicht kann optional auch einen Mechanismus zur Sicherstellung des Nachrichtenaustausches beinhalten. Dabei definiert CoAP vier verschiedene Arten von Nachrichten:

- bestätigende Nachricht (*Confirmable Message*)
- nicht bestätigende Nachricht (*Non-confirmable Message*)
- Bestätigungsnachricht (*Acknowledgement Message*)
- rücksetzende Nachricht (*Reset Message*)

Diese vier Arten stehen orthogonal zueinander, sprich:

- Bestätigende und nicht bestätigende Nachrichten sind für Anfragen (*requests*)
- und rücksetzende Nachrichten oder Nachrichten, die eine Anfrage bestätigen, sind für Antworten (*responses*) gedacht.

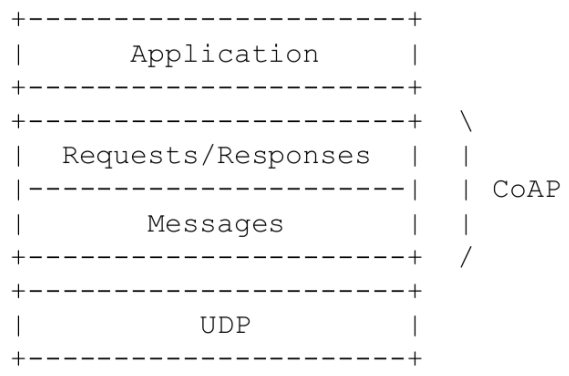

Figure 1: Abstrakte Darstellung der verschiedenen Schichten

Das Nachrichtenmodell des Constrained Application Protocols basiert auf den Austausch von Nachrichten über UDP. Dabei beginnt die Nachricht mit einem in der Länge fixierten Vier-Byte-Langen Kopfzeile (*header*), gefolgt von einer einem optionalen Token (null bis acht Bytes lang), null oder mehr sogenannten Optionen. Diese Optionen sind vergleich mit den *header fields* von HTTP. Nach den Optionen befindet sich ein sogenannter Anhang-Markierer (*Payload marker*) der einem Byte mit dem Wert 0xFF (255) entspricht. Anschließend an den *Payload marker* kommt der Anhang (*Payload*). Dieses Format ist sowohl für Anfragen als auch für Antworten dasselbe.

Damit Nachrichten als eindeutig identifiziert werden können, wird ein sogenannter Nachrichtenbezeichner (*Message ID*) verwendet. Dieser ist 16 Bit groß und erlaubt somit, bei Implementierungen mit Standardeinstellungen, bis zu 250 Nachrichten die Sekunde von einem Endpunkt zu einem anderen. Die *Message ID* wird auch für die Sicherstellung des Nachrichtenaustausches (*reliability*) benötigt. Dabei ist jedoch die *Message ID* nur zwischen zwei Endpunkten eindeutig. Kommuniziert ein Teilnehmer mit mehreren Teilnehmern gleichzeitig, dann können die *Message IDs* häufiger vorkommen. Für die eindeutige Identifizierung der Kommunikation zwischen zwei Teilnehmern wird der sogenannten Token verwendet. Dieser ist über mehrere Verbindungen hinweg eindeutig und kann als Identifikator für Verbindungen gesehen werden.

Die Sicherstellung des Nachrichtenaustausches erfolgt dadurch, dass man eine Nachricht als bestätigend (*Confirmable*) markiert. Eine als *Confirmable* gekennzeichnete Nachricht wird so lange an den jeweiligen Empfänger gesendet, bis dieser eine *Acknowledgement* Nachricht mit derselben *Message ID* zurücksendet (wie in Bild 2 dargestellt). Wenn der Empfänger die *Confirmable* Nachricht, aufgrund fehlender Daten oder fehlendem Kontext, nicht beantworten kann, sendet dieser eine *Reset* Nachricht zurück.

Jedoch wird für den Austausch von Nachrichten im CoAP Kontext kein Sicherheitsmechanismus für die Übertragung gefordert, sondern es können auch Nachrichten als *Non-confirmable* markiert werden (siehe Bild 3). Dies bietet sich zum Beispiel an, wenn man die Messdaten eines Sensors wiederholt ausliest. Dabei werden *Non-confirmable*

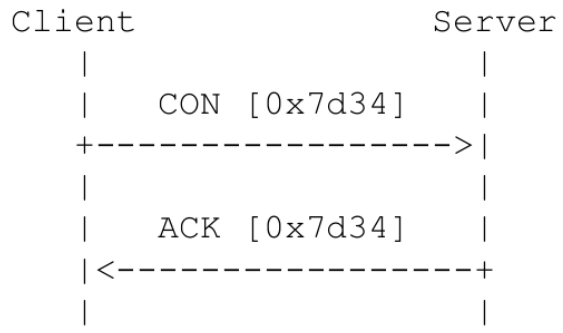

Figure 2: Nachrichtenaustausch mit Sicherstellung des Transfers (Quelle: [2]).

Nachrichten nicht bestätigt, jedoch wird eine *Message ID* benutzt, um Duplikate zu erkennen.

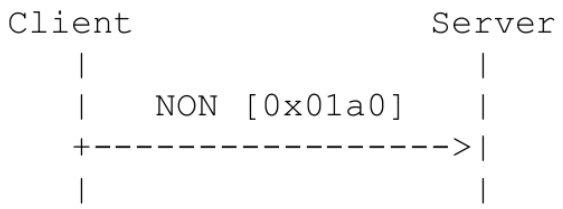

Figure 3: Nachrichtenaustausch ohne Sicherstellung des Transfers (Quelle: [2]).

Kann eine einkommende Anfrage (*Request*), die mithilfe einer *Confirmable* Nachricht versendet wurde, sofort beantwortet werden, wird die Antwort (*Response*) in der daraus resultierenden *Acknowledgment* Nachricht zurückgesendet. Dieses Prinzip nennt man auch *Piggybacked Response*. Das Bild 4 stellt diesen Mechanismus dar.

Ist der Server jedoch nicht sofort in der Lage die Anfrage zu beantworten, dann antwortet dieser mit einer leeren *Confirmable* Nachricht. Dies tut er, um den Client vom wiederholten Senden der Anfrage zu stoppen. Sind alle benötigten Daten zur Beantwortung der Anfrage vorhanden, sendet der Server die Antwort in einer neuen *Confirmable* Nachricht. Dieses Prinzip wird als separate Antwort (*separate response*) bezeichnet und kann mit dem Bild 5 nachvollzogen werden.

Dabei macht CoAP Gebrauch von den bekannten Internetmethoden *GET*, *PUT*, *POST* und *DELETE* in einer ähnlichen Art wie es HTTP tut.

Nachrichtenformat

Wie schon erwähnt, basiert der Nachrichtenaustausch von CoAP auf UDP. Dabei nimmt jede, über UDP versendete Nachricht ein ganzes UDP Datagramm. Dabei ist der Aufbau

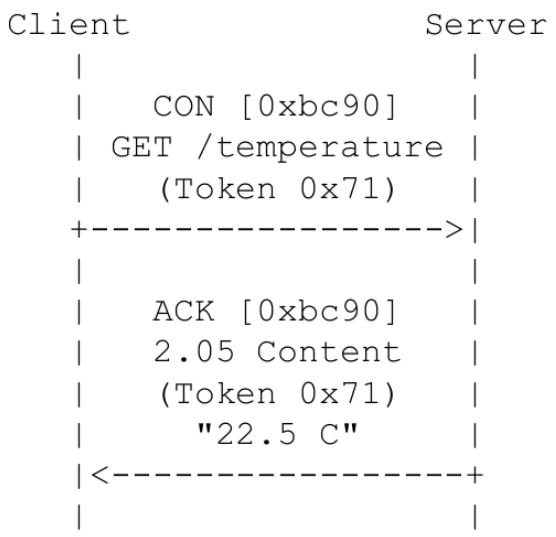

Figure 4: Beispiel eines erfolgreichen *Piggybacked Response* (Quelle: [2]).

einer CoAP Nachricht einfach gehalten und startet mit einer Vier-Byte-langen-Kopfzeile (*Header*). Diese beinhaltet folgende Daten (visualisiert im Bild 6):

- *Version*
- *Type*
- *Token Length*
- *Code*
- *Message ID*

Dabei repräsentieren die ersten zwei Bits des *Headers* die Versionsnummer. Die Versionsnummer gibt an, welcher CoAP Version die Nachricht erstellt wurde bzw. verarbeitet werden kann. In dieser Arbeit beschäftigen wir uns mit CoAP Nachrichten mit der Versionsnummer 1 (01 in Binär).

Die darauffolgenden zwei Bits entsprechen dem Typ der CoAP Nachricht. Der Typ gibt an, ob es sich um eine *Confirmable* (0 = 00 in Binär), *Non-Confirmable* (1 = 01), *Acknowledgement* (2 = 10) oder *Reset* (3 = 11) Nachricht handelt.

Als Nächstes kommt die vier Bit lange *Token Length*, die die Länge des *Tokens* angibt. Dabei kann der *Token* zwischen null (0000 in Binär) und acht (0111) Bytes betragen. *Token Lengths* zwischen neun und fünfzehn Bytes sind im RFC 7252 [2] für zukünftige Versionen reserviert.

Die nächsten 8 Bit geben den *Code* (vgl. mit dem Statuscode bei HTTP) der CoAP Nachricht an. Dabei unterteilt sich der *Code* in eine drei Bit lange *Code Class* (*most*

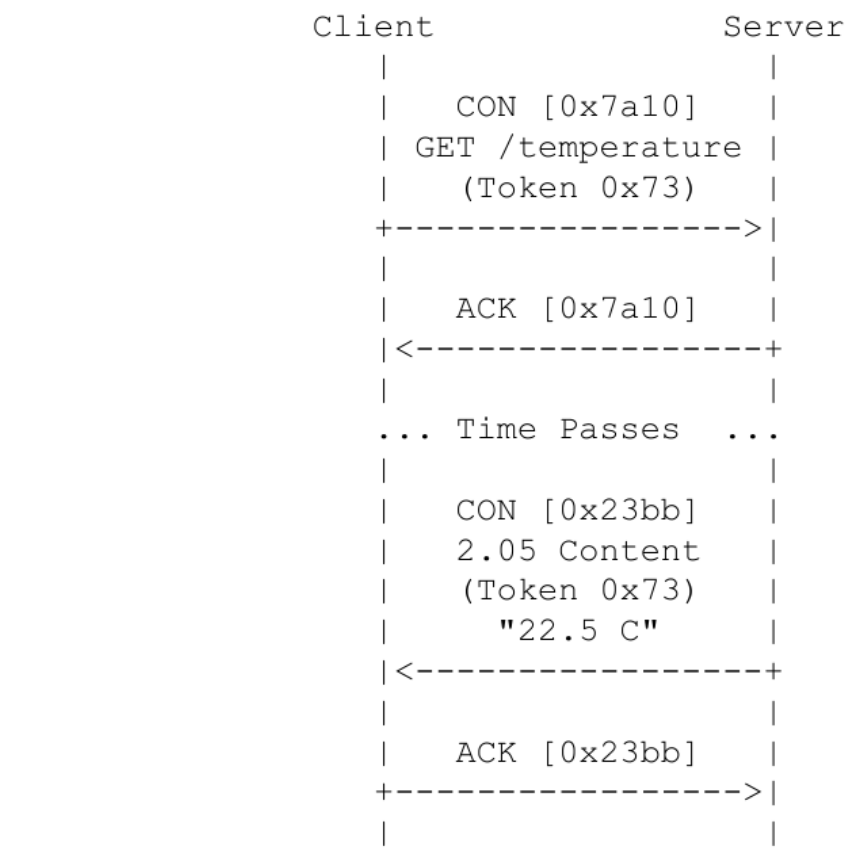

Figure 5: Beispiel einer *separate response* (Quelle: [2]).

significant bits) und einen fünf Bit lange *Code Detail* (*least significatn bits*). Dabei folgt der *Code* dem Schema "c.dd", wobei "c" Werte von 0 bis 7 annehmen kann und "dd" Werte von 00 bis 31. Die *Code Class* gibt dabei an, ob es sich um

- eine Anfrage (0),
- eine erfolgreiche Antwort (2),
- eine clientseitige, fehlerhafte Antwort (4),
- oder eine serverseitige, fehlerhafte Antwort (5).

Dabei nimmt der *Code* 0.00 eine besondere Stellung ein, da dieser eine leere Nachricht (*Empty Message*) markiert. Die *Codes* gleichen sich mit einigen Statuscodes, die man von HTTP kennt, jedoch ist nicht jeder Statuscode als CoAP *Code* abgebildet.

Der letzte Teil des *Headers* ist die sogenannte *Message Id*, die 16 Bit in Anspruch nimmt und in der *Networt Byte Order* (*Big Endian*) angegeben wird. Ihre Aufgabe ist es, Duplikate von Nachrichten zu erkennen. Auch wird die *Message Id* dazu benutzt, um

Nachrichten vom Typ *Acknowledgement* und *Reset* zu Nachrichten vom Typ *Confirmable* und *Non-Confirmable* zu verlinken. Somit besitzt der Server immer einen Überblick, zu welcher Anfrage schon eine Antwort geschickt wurde.

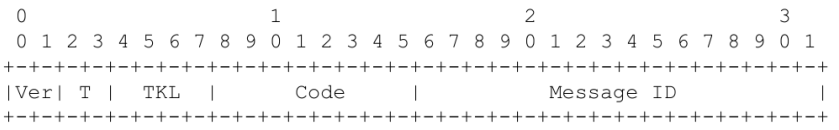

Figure 6: Binäre Struktur eines CoAP *Headers* (Quelle: [2]).

Anschließend an den *Header* kommt der *Token* der Nachricht. Dieser ist in seiner Länge variabel und hängt von der im *Header* angegebenen *Token Length* ab. Dieser ist zuständig für die Korrelation von Anfragen zu Antworten.

Nachfolgend können null oder mehr sogenannten *Options* folgen. Der Option können folgende Bestandteile einer CoAP Nachricht folgen:

- das Ende der CoAP Nachricht (EoF),
- eine weitere Option,
- oder der *Payload Marker* mit anschließender *Payload*.

Ist eine *Payload* gegeben, folgt nach der Gruppe von *Options* ein sogenannter *Payload Marker*. Dieser besteht aus einem Byte voller logischer Einsen (0xFF) und markiert somit das Ende der *Options*. Alle Daten, die nach dem *Payload Marker* befinden, werden als *Payload* behandelt. Dabei ist die Länge durch die *UDP Datagram* Paketgröße begrenzt. Wird für die Übertragung der Nachricht mehr Bytes benötigt, als ein *UDP Datagram* an Größe bereitstellen kann, werden die Bytes auf mehrere *UDP Datagrams* aufgespalten. Diesen Mechanismus nennt man auch *Blockwise transfer* und wird im RFC 7959 beschrieben [1].

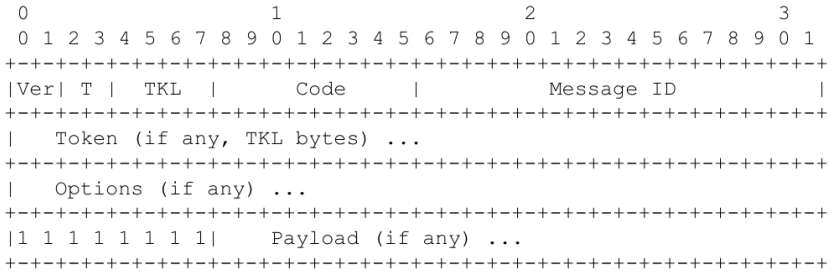

Figure 7: Binäre Struktur einer vollständigen CoAP Nachricht (Quelle: [2]).

Aufbau einer Option

Eine Option wird durch eine eindeutige Nummer identifiziert. Neben der Nummer besitzt eine Option auch einen Wert (*Value*), den diese Option hält, und einen Indikator für die Länge des Wertes. Dabei wird die Nummer nicht direkt in die Nachricht kodiert, sondern die Options werden zuerst aufsteigend nach ihrer Nummer sortiert und dann wird eine Deltakodierung (Differenzbildung) zwischen der aktuellen Option und deren Vorgängern gebildet. Dies geschieht dadurch, dass alle vorherigen Differenzen (*Deltas*) addiert werden und dann die Differenz zur aktuellen Option gebildet wird. Für die erste Option wird der Sonderfall behandelt, dass als vorheriges Delta ein Wert von null angenommen wird. Dies resultiert darin, dass für die erste Option die kodierte Differenz als Nummer der Option verwendet wird. Ein weiterer Sonderfall ist derjenige, wenn mehrere Instanzen der gleichen Option in der Kollektion von Options auftritt. Dabei ist die Differenz zwischen zwei gleichen Options immer null.

Eine Option fängt immer mit einem Byte an, das zwei Informationen enthält. Einmal die Differenz (*Option Delta*) und die Länge des Wertes (*Option Length*) der Option. Das *Option Delta* entspricht dabei den ersten vier Bits (*most significant bits*) und die *Option Length* die letzten vier Bits (*least significant bits*). Man kann dieses Byte auch als einen *Header* für Options bezeichnen, da dieser Informationen beinhaltet, die für das Kodierung bzw. Dekodieren von Options benötigt wird.

Um jedoch Differenzen und Längen, jenseits des Wertes fünfzehn, verwenden zu können, können dem *Option Header* das *Option Delta (extended)* und *Option Length (extended)* folgen. Diese beiden können jeweils zwischen null und zwei Bytes lang sein. Nach diesen beiden folgt der *Option Value*, der null oder mehr Bytes betragen kann.

1.2 Ausführungsparadigmen in der Informatik

In der Informatik spricht man von zwei großen Ausführungsparadigmen, mit denen Programme bzw. Codeteile ausgeführt werden können: **synchrone** und **asynchrone Ausführung**. Diese unterscheiden sich in wesentlichen Punkten deutlich voneinander und bieten in verschiedenen Einsatzszenarien Vor- und Nachteile. Dabei unterstützen die geläufigsten Programmiersprachen von Haus eine synchrone Ausführung von Programmen, jedoch bieten nicht alle eine asynchrone Ausführung.

	Synchron	Asynchron
Programmfluss	Stoppt den Programmfluss.	Kann im Programmfluss weiter gehen.
Beendigung	Überprüft periodisch, ob Funktion beendet ist.	Ein Event markiert die Beendigung der Funktion.
Main-Thread	Ist als "Blocked" oder "Waiting" markiert.	Ist frei für andere Aufgaben.

Table 1: Vergleich zwischen synchroner und asynchroner Ausführung

Um diese Unterschiede zwischen den beiden Paradigmen zu veranschaulichen, wird dies anhand einer Client-Server-Anwendung mit einer an den Server angeschlossenen Datenbank und zwei Clients verbildlicht (siehe Bild 8 und 9).

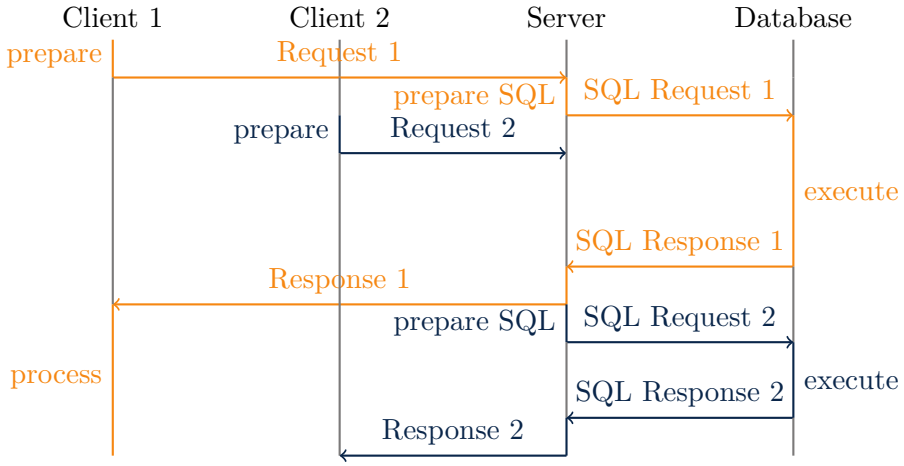

Figure 8: Sequenzdiagramm eines synchronen Servers

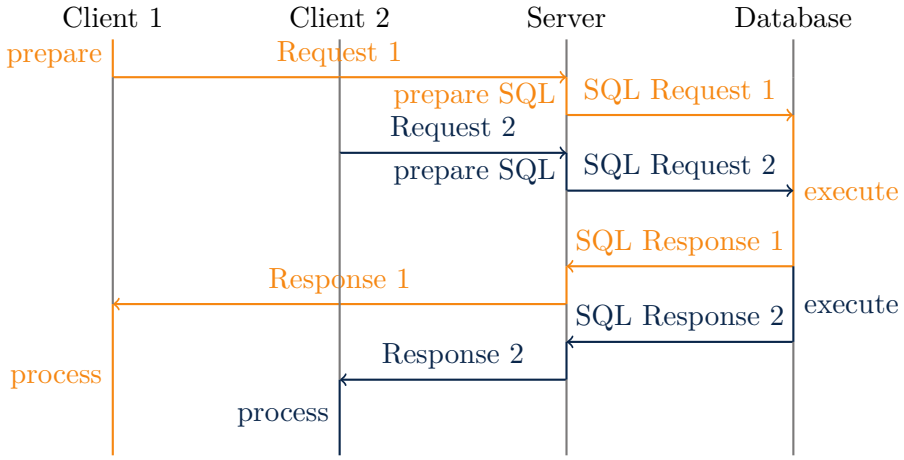

Figure 9: Sequenzdiagramm eines asynchronen Servers

Bibliography

[1] Z. Shelby and C. Bormann. *Block-Wise Transfers in the Constrained Application Protocol (CoAP)*. RFC 7959. RFC Editor, Aug. 2016. URL: http://www.rfc-editor.org/rfc/rfc7959.txt.

[2] Z. Shelby, K. Hartke, and C. Bormann. *The Constrained Application Protocol (CoAP)*. RFC 7252. RFC Editor, June 2014. URL: http://www.rfc-editor.org/rfc/rfc7252.txt.