CoAP
Constrained Application Protocol

Lehrveranstaltung Sensornetze
der Fakultät Informatik/Mathematik

Eingereicht von: Martin Dönicke, Ulrich Meckel
Eingereicht am: 15. Februar 2013

Inhaltsverzeichnis

Abkürzungsverzeichnis

6LoWPAN IPv6 over Low power Wireless Personal Area Network

6LBR 6LoWPAN Border Router

ACK Acknowledgement

CoAP Constrained Application Protocol

CON Confirmable

CoRE Constrained RESTful Environments

DNS Domain Name System

DTLS Datagram Transport Layer Security

FQDN Fully Qualified Domain Name

HTTP Hypertext Transfer Protocol

GPRS General Packet Radio Service

IETF Internet Engineering Task Force

IP Internet Protocol

IPSec Internet Protocol (IP) Security

M2M machine-to-machine

MLD Multicast Listener Discovery

MPL Multicast Protocol for Low power and Lossy Networks

NON Non-confirmable

NoSec No Security

ODP Observe-Design-Pattern

PHY Physical Layer

RAM Random-Access-Memory

REST Representational State Transfer

RFC Request for Comments

ROM Read-Only-Memory

RPL Routing Protocol for Low power and Lossy Networks

RST Reset

SMS Short Messaging Service

TCP Transmission Control Protocol

TLV Type-Length-Value

UCS Universal Character Set

USSD Unstructured Supplementary Service Data

UDP User Datagram Protocol

UTF-8 8-Bit UCS Transformation Format

URI Uniform Resource Identifier

Glossar

Endpoint Einheit im CoAP Netzwerk, aber nicht zwingend ein Endknoten

Sender Der Ursprungs-Endpoint einer Nachricht

Recipient Der Ziel-Endpoint einer Nachricht

Client Der Ursprungs-Endpoint einer Request-Nachricht und Ziel-Endpoint einer Response-Nachricht

Server Der Ziel-Endpoint einer Request-Nachricht und Ursprungs-Endpoint einer Response-Nachricht

Origin Server Ein Server auf dem eine gewisse Ressource angelegt ist

Intermediary Ein Endpoint, der sowohl als Server, als auch als Client fungiert (bspw. ein Proxy)

Forward-Proxy Dient dem Weiterleiten von Nachrichten

Reverse-Proxy Bietet fremde Ressourcen als seine eigenen an

Cross-Proxy Ermöglicht Kommunikation mit CoAP-Knoten über ein anderes Protokoll

1. Einleitung

Das Constrained Application Protocol (CoAP) ist ein für den Einsatz in Sensornetzen konzipiertes Transferprotokoll. Spezielle Randbedingungen in solchen Netzen sind wenig Random-Access-Memory (RAM) und Read-Only-Memory (ROM) der einzelnen Kommunikationsteilnehmer, welche z.B. durch 8-Bit Mikrocontroller realisiert werden. Weiterhin bieten Sensornetze im Vergleich zu üblichen Computernetzen nur geringe Datenraten und haben möglicherweise hohe Paketfehlerraten.

Ein mögliches Netz in dem CoAP zum Einsatz kommen könnte, wäre beispielsweise ein IPv6 over Low power Wireless Personal Area Network (6LoWPAN)-Netz, in dem CoAP-Nachrichten via User Datagram Protocol (UDP) versendet werden. Außer UDP in der Transportschicht, werden jedoch keine weiteren unterliegenden Schichten für CoAP spezifiziert. Es wird lediglich versucht, den Overhead der einzelnen Nachrichten gering zu halten. Zudem hat CoAP eine Representational State Transfer (REST)-Architektur (siehe [Fie00]), weshalb die Last auf speicherarmen Knoten nicht so hoch ist, da keine Status gehalten werden müssen. Wie genau das versucht wird zu erreichen, wird später in diesem Dokument erläutert. Die Entwicklung von CoAP wird von der Constrained RESTful Environments (CoRE)-Gruppe vorangetrieben. Der erste Entwurf wurde im Jahr 2010 veröffentlicht. Zum Zeitpunkt der Erstellung dieses Dokuments war Version 13 (siehe [GSH+12b]) aktuell. Es wird versucht auf einige Änderung einzugehen, die während der Entwicklung entstanden sind.

1.1. Features

In [GSH+12b] wird CoAP mit folgenden Features beworben:

- Constrained web protocol fulfilling machine-to-machine (M2M) requirements.
- UDP binding with optional reliability supporting unicast and multicast requests.
- Asynchronous message exchanges.
- Low header overhead and parsing complexity.
- Uniform Resource Identifier (URI) and Content-type support.
- Simple proxy and caching capabilities.
- A stateless Hypertext Transfer Protocol (HTTP) mapping, allowing proxies to be built providing access to CoAP resources via HTTP in a uniform way or for HTTP simple interfaces to be realized alternatively over CoAP.
- Security binding to Datagram Transport Layer Security (DTLS).

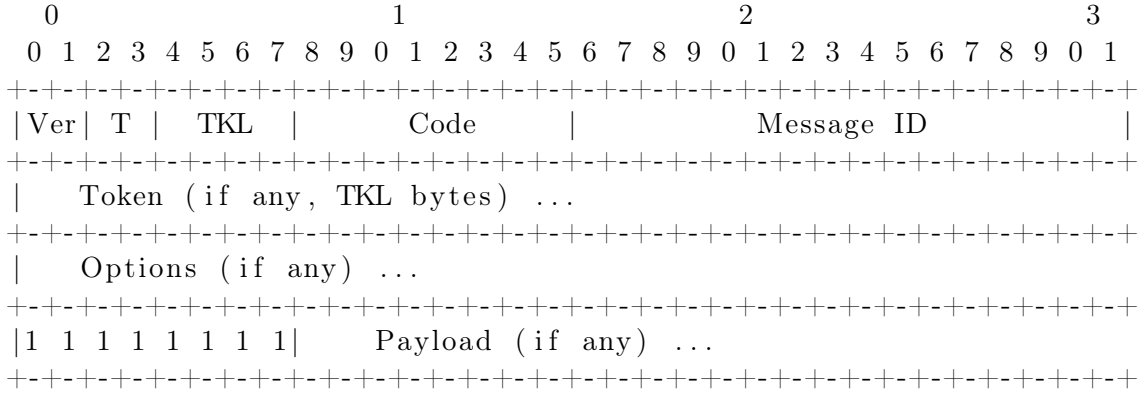
1.2. Abgrenzung und Ziel

Ziel von CoAP ist es nicht, ein komprimiertes HTTP umzusetzen, sondern viel mehr einen Teil von REST, wie es in HTTP zum Einsatz kommt, bereitzustellen. Insbesondere Optimierungen für eine M2M-Kommunikation wie beispielsweise eingebaute Discovery-Mechanismen, Unterstützung für Multicasts und asynchronen Nachrichtenaustausch sollen durch CoAP bereitgestellt werden.

2. Message Format

2.1. Header Format

Der CoAP-Header, wie in Abbildung 1 zu sehen, hat damit eine Mindestlänge von 4 Byte. Je nach Nachricht, folgt danach der Token und die Optionen, die für das Bearbeiten der Anfrage oder das Verstehen der Antwort wichtig sind. Vorallem bei Response-Nachrichten gibt es zusätzlich am Ende der Nachricht noch einen Payload, in dem beispielsweise die abgefragte Repräsentation einer Ressource enthalten ist.



Name	Länge	Wertebereich	Beschreibung
Version	2 - Bit	1	CoAP Version
Type	2 - Bit	0 - 3	Signalisiert den Nachrichtentyp
		0	Confirmable
		1	Non - Confirmable
		2	Acknowledgement
		3	Reset
Token Length	4 - Bit	0 - 8	Gibt die variable Länge des Tokens an
		0 - 8	Länge des Tokens
		0 -15	reserviert
Code	8 - Bit	0 - 31,64 - 191	Zeigt, ob es sich um eine
		0	Empty Message,
		1 - 31	Request Message (siehe Abb. 11) oder
		64 - 191	Response Message (siehe Abb. 12) handelt
Message ID	16 - Bit		unigned integer in Network Byte order
			dient dem Request/Response Matching

Abbildung 1: Message Format

2.1.1. Tokenlänge statt Option Count

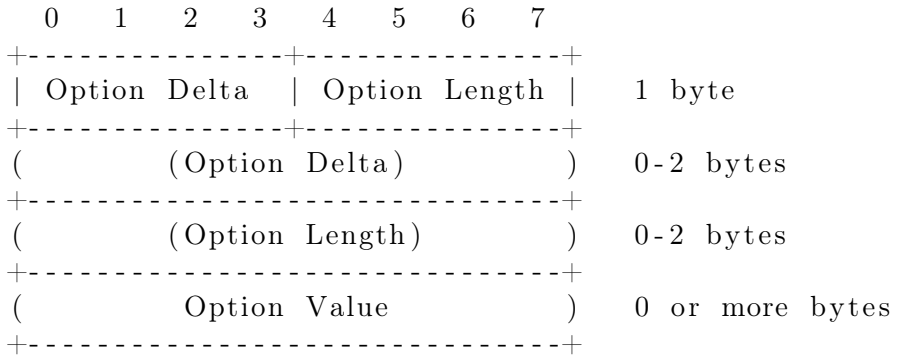
Mit dem draft-13[[GSH+12b]] von CoAP, welches am 6. Dezember 2012 erschien, hat der Token nun seinen Weg direkt in den Header einer CoAP-Nachricht gefunden. Vorher wurde der Token wie eine normale Option behandelt. Statt der jetzigen Token-Länge im Header wurde damals die Anzahl der Optionen übergeben. Doch da sich herausstellte, dass allein die Message-ID oft nicht ausreicht, um eine Response-Nachricht ihrer Request-Nachricht zuordnen zu können, oder man den Server damit nicht mehr die Wahl lassen konnte seine Antwort piggybacked (siehe Abschnitt 5.1 Piggy-backed Response) oder seperat (siehe Abschnitt 5.2 Separate Response) abzusenden. Bei der Verwendung von mehr als 14 Optionen musste damals ein zusätzlicher End-Of-Option-Marker benutzt werden, da im Option-Count-Feld mit der Länge 4 Bit nicht mehr angegeben werden konnte. Ein Option-Count von 15 bedeutete dementsprechend, dass der oben erwähnte Marker als Trenner zwischen Optionen und Payload diente. Statt dies nur in Ausnahmen so zu handhaben, ist es in der aktuellen Version Standard. Dafür müssen die Anzahl der Optionen nicht mehr angegeben werden und der, meist ohnehin verwendete, Token beschreibt in diesem Feld nun seine Länge. Bei einer Token Länge von 0 ist der Token mit dem Wert 0 anzusehen. Um nun allerdings Header, Optionen und Payload einer CoAP-Nachricht auseinander halten zu können, verwendet man den oben bereits erwähnten End-Of-Option-Marker. Allerdings nennt er sich nun Payloadmarker, welcher aus einem Byte mit Einsen besteht. Bei Optionen ist deshalb nun darauf zu achten, dass das erste Byte eben genau nicht aus einem Byte voller Einsen besteht.

2.1.2. Offengelassene Erweiterungsmöglichkeiten

Wie leicht zu erkennen ist, ist im Header noch viel Platz für spätere Ideen im CoAP-Protokoll. Das Token-Längenfeld darf gerade mal acht der 15 möglichen Werte annehmen. Dies reicht für die Tokenlänge auch aus, doch finden die restlichen Werte in dem Feld noch keine Verwendung. Ebenso gibt es noch viel Spielraum im Code-Feld. Dort werden nur 159 der 256 möglichen Werte ausgenutzt. Auch hier ist man schon sehr großzügig gewesen und hat sehr viele Fehlercodes aus dem HTTP-Protokoll (nach [FGM+99]) übernommen. Es ist also durchaus möglich, dass bis zum Übergang des drafts in einen Request for Comments (RFC), noch weitere Anpassungen, Alternativen oder Optimierungen am Header vorgenommen werden.

2.2. Option Format

In CoAP können keine bis zu theoretisch unendlich vielen Optionen zwischen Header und Payload eingehängt werden. Alle Optionen sind, wie in Abbildung 2 zu sehen, immer nach dem Schema Type-Length-Value (TLV) aufgebaut. Als Besonderheit ist hier alledings zu beachten, dass der Typ nicht explizit angegeben wird (siehe Abschnitt 2.2.1 Optionsnummern Differenz). Bei den Optionen wurde vorallem darauf Wert gelegt, sie sehr variabel gestalten zu können. Optionen, die beispielsweise keinen Wert benötigen, wie die „If-None-Match“ Option (Optionsnummer 5), wird so nur 1 Byte groß. Ebenso ist es aber auch möglich sehr große Optionswerte zu übetragen.



Name	Länge	Wert	Beschreibung
Option Delta	4 - Bit		Gibt die Optionsnummerndifferenz zur vorherigen Option oder von 0 bei der ersten Option an
		0 - 12	Optionsnummerndifferenz
		13	8 - Bit Options Delta folgt
		14	16 - Bit Options Delta folgt
		15	Reserviert für Payload Marker
Options Length	4 - Bit		Gibt die Länge des Optionswertes an
		0 - 12	Länge in Bytes
		13	8 - Bit Options Länge folgt
		14	16 - Bit Options Länge folgt
		15	Reserviert
optionales Delta	8 - Bit	Delta - 13	Für Werte zwischen 13 - 268
optionales Delta	16 - Bit	Delta - 269	Für Werte zwischen 269 - 65804
optionale Länge	8 - Bit	Länge - 13	Für Werte zwischen 13 - 268
optionale Länge	16 - Bit	Länge - 269	Für Werte zwischen 269 - 65804
Option Value	Option Length		Wert der Option

Abbildung 2: Option Format

2.2.1. Optionsnummern Differenz

Da sich einige Optionen wiederholen können, wäre es nicht sinnvoll immer die komplette Optionsnummer zu übertragen, wenn diese sehr groß ist. Aus diesem Grund wird bei CoAP immer die Differenz zur vorherigen Optionsnummer angegeben. Das bedeutet, dass eine Folge von gleichen Optionen mit einer großen Optionsnummer nur ein Delta von 0 angeben müssen, anstatt ihrer Optionsnummer. Dies hat als Nebeneffekt zur Folge, das die Optionen sortiert nach ihrer Optionsnummer aufgelistet werden, da Rückwärtssprünge nicht möglich sind.

Ein Problem hat das Delta-Prinzip allerdings, denn werden große Sprünge gemacht, kann das 4-Bit große Feld unter Umständen nicht ausreichen. In frühen CoAP-Versionen wurden deshalb sogenannte Fencepost-Optionen eingefügt. Diese hatten als Optionsnummer immer ein Vielfaches von 14. Wollte man beispielsweise als erstes eine Option mit der Nummer 30 an seine Nachricht hängen, so musste man erst 2 Optionen mit dem Delta 14 angeben und erst danach die Option mit einem Delta von 2. Es wurden also 2 Bytes für die Fencepost Optionen benötigt um diesen Sprung durchführen zu können. Ab CoAP-12[[GSH+12a]] gibt es nun keine Fencepost Optionen mehr. Dafür gibt es jetzt sogenannte „Extended“-Felder. Werden große Sprünge benötigt oder ist der Wert der Option sehr lang, so gibt es extra Felder um diesen großen Wert auszudrücken. Bei unserer Option mit der Nummer 30 beisielsweise, würden wir als Option-Delta die 13 eintragen und haben uns so ein 8-Bit großes Extended-Delta-Feld , wie oben beschrieben, reserviert. In diesem Feld brauchen wir nun nur noch eine 17 (= 30 - 13) eintragen. Anstatt wie bei der Fencepost-Methode 2 Byte für diesen großen Sprung zu bezahlen, zahlen wir mit der Extended-Feld-Methode nur noch 1 Byte. Bei größeren Sprüngen, ist die Ersparnis noch größer.

2.2.2. Optionswerte

In CoAP-Optionen gibt es nur vier verschiedene Typen, die auftreten können. Eine Option hat auch immer den gleichen Datentyp, wie in [GSH+12b] beschrieben(siehe Abbildung 3). Es muss also nicht jedes mal in der Option beschrieben werden, in welchem Format der Optionswert übertragen wurde.

empty Kein Optionswert (0-Byte Sequenz von Bytes)

opaque eine Sequenz von Bytes, die nicht interpretiert werden muss

uint eine nicht-negative Zahl in Network Byte Order

string ein Unicode String kodiert in 8-Bit UCS Transformation Format (UTF-8) (nach [Yer03]) als Net-Unicode form (nach [KP08])

3. Optionen

Im Abschnitt 2.2 Option Format ist bereits der Aufbau einer Option erläutert. In diesem Kapitel soll einmal genau der Aufbau der Optionsnummer betrachtet werden und ebenso welche verschiedenen Optionen CoAP zur Verfügung stellt und wozu diese notwendig sind.

No.	C	U	N	R	Name	Format	Length	Default
1	x			x	If-Match	opaque	0-8	(none)
3	x	x	-		Uri-Host	string	1-255	(see below)
4				x	ETag	opaque	1-8	(none)
5	x				If-None-Match	empty	0	(none)
7	x	x	-		Uri-Port	uint	0-2	(see below)
8				x	Location-Path	string	0-255	(none)
11	x	x	-	x	Uri-Path	string	0-255	(none)
12					Content-Format	uint	0-2	(none)
14		x	-		Max-Age	uint	0-4	60
15	x	x	-	x	Uri-Query	string	0-255	(none)
16				x	Accept	uint	0-2	(none)
20				x	Location-Query	string	0-255	(none)
35	x	x	-		Proxy-Uri	string	1-1034	(none)
39	x	x	-		Proxy-Scheme	string	1-255	(none)

C=Critical, U=Unsafe, N=No-Cache-Key, R=Repeatable

Uri-Host=destination IP literal, Uri-Port=destination UDP Port of request message

Abbildung 3: CoAP Optionen

3.1. Eigenschaften

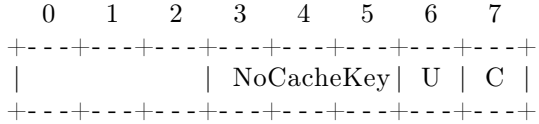


Abbildung 4: Optionsnummernmaske

3.1.1. Critical / Elective

Ist eine Option als kritisch markiert, muss diese verstanden werden, andernfalls führt sie zu folgenden drei Fehlerfällen:

Confirmable (CON)-Request Kann die Option bei einer CON-Request-Nachricht nicht verstanden werden, muss sie mit dem Fehlercode `4.02` (Bad Option) beantwortet werden. Zusätzlich sollte im Payload einer solchen Antwort eine kurze, menschlich lesbare, UTF-8 kodierte Nachricht stehen, die den Fehler beschreibt. Beispielsweise bei welcher Option der Fehler überhaupt auftrat und was nicht verstanden wurde oder man beispielsweise die Option überhaupt nicht kennt.

CON-Response Kann bei einer CON-Response oder einer durch Piggy-backed-Response (siehe Abschnitt 5.1 Piggy-backed Response) erhaltenen Nachricht eine kritische Option nicht verstanden werden, muss diese mit einer Reset (RST)-Nachricht beantwortet werden und die Response selbst aber ignoriert werden.

Non-confirmable (NON)-Nachricht Nachrichten mit einer oder mehreren nichtverstandenen kritischen Optionen müssen ignoriert werden. Der Empfänger einer solchen fehlerhaften oder nicht verstandenen Option kann diese Nachricht auch durch eine RST-Nachricht beantworten, muss sie aber dennoch ignorieren.

Kann die Option nicht verstanden werden und ist aber nicht kritisch, so wird die Option ignoriert, die Nachricht allerdings nicht.

Ob eine Option kritisch ist oder nicht, entscheidet sich durch das niedrigstwertige Bit der Optionsnummer, wie in Abbildung 4 zu sehen. Ist es gesetzt, so ist die Option kritisch, sonst nicht. Damit sind alle Optionen mit einer ungeraden Optionsnummer kritisch.

3.1.2. Proxy Unsafe/Safe

Proxys müssen nicht alle Optionen verstehen können. Aus diesem Grund, ist es für diese Gruppe von Endpoints nicht relevant, ob die Optionen kritisch sind oder nicht sind. An dieser Stelle wird zwischen Safe-to-Forward oder Unsafe-to-Forward unterschieden. Versteht ein Proxy also eine Option nicht, die mit Unsafe-to-Forward markiert ist, muss er diese mit einem `4.02` (Bad Option) Fehlercode beantworten. Safe-to-Forward Optionen werden von Proxys nochmals unterschieden. Nämlich ob sie zum Cache-Key gehören oder nicht. Wozu diese zusätzliche Unterscheidung dient, wird bei der Erklärung der ETag-Option (siehe Abschnitt 3.2.2 ETag) verdeutlicht. Das zweit-niedrigstwertige Bit der Optionsnummer entscheidet darüber, ob die Option Safe-to-Forward ist oder nicht, vergleiche Abbildung 4. Ist es gesetzt, so ist die Option Unsafe-to-Forward. Ist es nicht gesetzt, so unterscheiden die 3 nächsthöheren Bits darüber, ob die Option zum Cache-Key gehört oder nicht. Ist eines der 3 Bits nicht gesetzt, so gehört die Safe-to-Forward Option zum Cache-Key.

3.1.3. Repeatable

Manche Optionen können in CoAP mehrfach vorkommen. Dies ist zum Beispiel wichtig bei der URI-Path Option (siehe Abschnitt 3.2.1 Uri-Host, Uri-Port, Uri-Path, Uri-Query). In [GSH+12b] wird festgelegt, welche Optionen mehrfach in einer Nachricht

enthalten sein können und welche nicht. Kommt eine Option mehrfach vor, welche nicht mehrfach vorkommen darf, muss sie als nichtverstandene Option behandelt werden.

3.1.4. Längen und Standardwerte

Wie in Abbildung 3 zu sehen ist, haben fast alle Optionen eine Unter-/und Oberbeschränkung ihrer Länge. Hält eine erhaltene Option diesen Wertebereich nicht ein, so muss sie ebenso als nichtverstandene Option behandelt werden. Optionen können Standardwerte besitzen. Möchte man diese Werte verwenden, so muss die Option nicht extra übermittelt werden.

3.2. Die Optionen im Einzelnen

3.2.1. Uri-Host, Uri-Port, Uri-Path, Uri-Query

Die vier Uri-Optionen dienen dem genauen spezifizieren an welchen Endpoint, identifiziert mit einer eindeutigen URI, die CoAP-Nachricht gesendet werden soll. Im Draft sind genaue Regeln spezifiziert, wie aus einer URI die entsprechenden Optionen erzeugt werden und ebenso der Rückweg. Auf die Regeln möchten wir hier aber nicht weiter eingehen. Ein Beispiel soll an dieser Stelle genügen. Aus der URI
`coap://[fe80::d2df:9aff:fe72:75bb]:5683/wohnzimmer/temp?location=bottom`
würden

- eine Uri-Host Option mit dem Inhalt “fe80::d2df:9aff:fe72:75bb“ ,
- eine Uri-Port Option mit dem Inhalt “5683“,
- zwei Uri-Path Optionen, jeweils mit dem Inhalt “wohnzimmer“ und “temp“ und
- eine Uri-Query Option mit dem Inhalt “location=bottom“ entstehen.

Alle 4 Optionen sind kritisch und Unsafe-to-Forward. Wie auch schon im Beispiel zusehen ist, dürfen die Optionen Uri-Path und Uri-Query mehrfach vorkommen.

3.2.2. ETag

Der ETag ist eine Bytesequenz und identifiziert eine lokale Ressource auf einem Server. Dieser Server kann den ETag mit ihm möglichen Mitteln erzeugen, beispielsweise durch Hashing, Bildung eines Zeitstempels oder Nutzen einer Versionierung. Allerdings sollte er je nach Anwendungsumgebung nicht zu groß werden. Der Empfänger eines solchen ETags muss die Bytesequenz nicht selber auswerten können.

Bei dieser Option ist es wichtig zu unterscheiden, ob sie in einer Request- oder einer Response-Nachricht auftaucht. Als Option einer Response-Nachricht dient sie als lokaler Bezeichner für die auf dem Server angefragte Repräsentation einer Ressource. Dementsprechend darf die Option in einer Response Nachricht nur einmal vorkommen.

Ein Client kann diesen ETag, und möglicherweise zuvor ebenso erhaltene ETags dieser Ressource, nutzen, um möglicherweise Traffic zu sparen. Hängt der Client an seine

Request-Nachrichten seine ihm bekannten ETags an, so kann der Server unter Umständen mit einem `2.03` (Valid)-Code signalisieren, dass die Repräsentation eine der vom Client mitgelieferten Bezeichner der Ressource entspricht. Um bei mehreren mitgelieferten ETags zu unterscheiden, welcher nun auf dem Server vorliegt, sendet der Server bei seiner Antwort den aktuellen ETag wieder mit.

Der Vorteil der Nutzung von ETags wird einem schnell klar, wenn man sich eine Ressource vorstellt, deren Repräsentation mehrere Bytes umfasst. Anstatt immer die gesamte Repräsentation zu übertragen, kann es ab der 2. Abfrage beispielsweise reichen, den ETag zu bestätigen, wenn sich die Ressource nicht geändert hat.

Anhand des ETags soll beispielhaft erklärt werden, wozu die Unterscheidung der Safe-to-Forward Optionen in Cache-Key und No-Cache-Key dient. Die ETag Option hat die Nummer $4_{10} = 100_2$ und ist deshalb keine kritische Option, da die Anfrage auch ohne Verstehen der Option ausgeführt werden kann. Versteht ein Server die ETag Option nicht, da er möglicherweise über kein Verfahren verfügt um den ETag zu generieren, muss er diese Option auch nicht unterstützen können. Anstelle den ETag auszuwerten, der ihm scheinbar fälschlich zugesendet wurde, sendet er einfach eine Response Nachricht zurück und ignoriert diese Option. Ebenso ist die Option auch nicht Unsafe-to-Forward. Ein Proxy muss diese Option nicht verstehen können, da sie für die Ausführung seiner Aufgabe keine Rolle spielt. Die NoCacheKey-Bits `3-5` sehen also wie folgt aus: `001`. Dies bedeutet, dass der ETag zum Cache-Key gehört, da der NoCacheKey-Block nicht komplett mit Einsen gesetzt ist. Verwendung findet diese Unterscheidung des Cache-Keys vorallem bei Proxys, die die Option nicht verstehen können. Sind die Proxys auch dafür zuständig, Repräsentationen für Ressourcen zu cachen, so fließt diese ETag-Option trotzdem in den Caching-Algorithmus mit ein, obwohl sie die Option selbst nicht verstehen.

Beispiel Im Cache liegt eine Request-Nachricht mit der ETag Option sowie die dazugehörige Response-Nachricht. Im Falle einer Anfrage mit dem gleichen Cache-Key (im einfachsten Fall hier ist es nur der ETag), kann die Response aus dem Cache zurückgeliefert werden. Ist allerdings der ETag anders, kann die Response-Nachricht aus dem Cache nicht verwendet werden. Diese Unterscheidung kann der Proxy anhand des Cache-Keys treffen, ohne die ETag Option zu kennen.

3.2.3. Max-Age

Die Option Max-Age hat die Nummer $14_{10} = 1110_2$, und ist somit unkritisch, aber Unsafe-to-Forward. Die Option dient dem Caching-Verhalten von Proxys und muss deshalb auch zwingend von ihnen verstanden werden. Die, maximal einmalig vorkommende, Option in einer Response-Nachricht besagt, dass diese Response-Nachricht maximal dem Optionswert in Sekunden entsprechende Zeit im Cache lag. Der Wert dieser Option kann zwischen 0 bis zu $2^{32} - 1$ Sekunden (ca. 136,2 Jahren) variieren. Ist die Option nicht in einer Response enthalten, gilt der Standardwert von 60 Sekunden.

3.2.4. Location-Path, Location-Query

Die Location-Path-Option mit der Nummer $8_{10} = 1000_2$, und die Location-Query-Option mit der Nummer $20_{10} = 10100_2$, sind unkritische, Safe-to-Forward und wiederholbare Optionen. Sie werden benötigt, um relative Pfade anzugeben. Dies wird beispielsweise notwendig, wenn ein Client ein POST-Request ausführt und der Server eine neue Ressource anlegt. Dabei gibt er mit den Location-*-Optionen den relativen Pfad zu der neuen Ressource an.

3.2.5. Proxy-Uri, Proxy-Scheme

Um CoAP die Möglichkeit zu geben, auch in andere Protokolle, wie zum Beispiel HTTP, Anfragen zu senden, ist es vorgesehen, sogenannte Forward-Proxys einzusetzen. Diese senden dann die Anfrage weiter oder geben eine zur Anfrage passende, im Cache vorhandene, Antwort zurück. In der Proxy-Uri-Option steht ein absoluter Pfad zu der gewünschten Ressource. Anders als bei den Uri-*-Optionen wird hier die gesamte URI in eine Option geschrieben. Zudem ist es nicht erlaubt, Uri-*-Optionen und die Proxy-Uri-Option in einer Nachricht zu verwenden. Erhält ein Endpoint eine Nachricht mit einer Proxy-*-Option, ist aber nicht in der Lage oder nicht dafür konfiguriert, die Pakete weiterzuleiten, muss er den Fehlercode `5.05` (Proxying Not Supported) an den Client zurücksenden. Proxys, welche die Anfrage weiterleiten können, müssen über ihr Netzwerk so viele Kentnisse besitzen wie nötig sind, um zu entscheiden, ob sie die Anfrage nochmals an einen Proxy weiterleiten müssen oder direkt den gewünschten Server ansprechen.

Da die Proxy-Uri-Option als nicht wiederholbar markiert ist, kann es passieren, dass die Länge von 1034 nicht ausreicht, um den absoluten Pfad anzugeben. Ebenso ist es denkbar, dass mehrere Proxys existieren, welche in verschiedene Protokolle weiterleiten können, um eine Ressource anzusprechen. Ist dies der Fall, so kann mit Hilfe der Proxy-Scheme-Option gearbeitet werden. Statt nun den gesamten absoluten Pfad in eine Option schreiben zu müssen, steht nur der Scheme der absoluten URI in der Proxy-Scheme-Option und der Rest der URI wird, wie gewohnt, mit den Uri-*-Optionen ausgedrückt. Da die Uri-*-Optionen aber keine Fragmente unterstützen, können diese bei solchen Anfragen nicht mit angegeben werden.

Diese zwei Optionen sind als kritisch und Unsafe-to-Forward markiert, was uns ein Blick auf die Nummern $35_{10} = 100011_2$ und $39_{10} = 100111_2$ schnell verrät.

3.2.6. Content Format

Server können unter Umständen verschiedene Formate zur Darstellung einer Repräsentationen ihrer Ressource anbieten. Um dem Client mitzuteilen, welches Format nun im Payload einer Response-Nachricht enthalten ist, gibt er die Content-Format-Option mit der Nummer $12_{10} = 1100_2$ an. Im Optionswert steht einer der in Abbildung 5 aufgeführten IDs.

Media type	Encoding	Id.	Reference
text/plain; charset=utf-8	-	0	[RFC2046][RFC3676][RFC5147]
application/link-format	-	40	[RFC6690]
application/xml	-	41	[RFC3023]
application/octet-stream	-	42	[RFC2045][RFC2046]
application/exi	-	47	[EXIMIME]
application/json	-	50	[RFC4627]

Abbildung 5: CoAP Content Formats

3.2.7. Accept

Nun kann es aber vorkommen, dass ein Client manche Formate aus der Abbildung 5 nicht verarbeiten kann. Um dem Server mitzuteilen, welche Formate er untersützt, kann er die Accept-Option nutzen. Die Option darf wiederholt werden. Das gibt dem Client die Möglichkeit mehrere bevorzugte Formate anzugeben. Wie einem bei betrachten der Optionsnummer $16_{10} = 10000_2$ auffällt, ist die Option nicht kritisch. Es können also drei verschiedene Situationen auftreten:

Server versteht die Option nicht Da die Option nicht kritisch ist, führt der Server die Anfrage aus und ignoriert dabei die Accept-Option des Clients. Dies kann zur Folge haben, dass der Client ein Format zurückbekommt, dass er nicht verarbeiten kann.

4.06 Not Acceptable Der Server versteht die Option, kann aber keine der vom Client bevorzugten Formate anbieten. Er sendet als Antwort eine Fehlermeldung zurück, in der er angibt, dass er keine der gewünschten Formate zur Verfügung hat.

2.05 Content Versteht der Server die Option, so versucht er die Repräsentation der Ressource in den Formaten anzugeben, die der Client bevorzugt. Dabei ist die Reihenfolge der Optionen entscheidend. Das zuerst genannte ist ebenso auch das vom Client am meisten priorisierteste Format.

3.2.8. Bedingte Anfragen

In CoAP ist es auch möglich, bedingte Anfragen zu stellen. Sind gewünschte Bedingungen erfüllt, so führt der Server die Anfragen aus, als gäbe es diese Bedingungen

nicht. Sind diese nicht erfüllt, so muss er mit dem Fehlercode `4.12` (Precondition Failed) antworten. Zwei Möglichkeiten stehen zur Verfügung:

If-Match In dieser Option kann geprüft werden, ob die angefragte Ressource überhaupt existiert. Dazu wird ein leerer String als Optionswert mitgesendet oder ein Wert der Ressource verglichen. Möchte man beispielsweise einem Lost-Update bei einer `PUT`-Anfrage vorbeugen, so kann in der If-Match-Option ein ETag als Optionswert mitgegeben werden, was die zuletzt empfangene `GET`-Anfrage geliefert hat. If-Match ist eine wiederholbare Option. Bei dem Server werden sie als „Oder"-Bedingungen interpretiert, d.h. es muss nur eine der Bedingungen erfüllt werden, um die Anfrage auszuführen.

If-None-Match Um paralleles Erzeugen einer Ressource von mehreren Clienten zu verhindern gibt es die If-None-Match Option. Sie benötigt keinen Wert und ist auch nicht wiederholbar. Die Bedingung wird wahr, wenn die angefragte Ressource nicht existiert.

4. Nachrichtenmodell

Der Nachrichtenaustausch in CoAP basiert auf dem Versand von Nachrichten via UDP. Jede Nachricht besitzt eine Message-ID, welche zur Duplikaterkennung genutzt wird. Zuverlässigkeit kann durch das Versenden einer Confirmable (CON)-Nachricht erreicht werden. Solche Nachrichten werden mit einem Standard-Timeout erneut versendet, bis sie mit einer Acknowledgement (ACK)-Nachricht mit der selben Message-ID vom Empfänger bestätigt werden oder eine maximale Anzahl von Retransmits erreicht haben. Wenn der Empfänger die Nachricht nicht verarbeiten kann und diesen Fehler auch in keiner Antwort ausdrücken kann, dann antwortet der Sender statt mit einem ACK mit einer Reset (RST)-Nachricht.

Wenn eine Nachricht nicht zuverlässig übertragen werden muss, weil sie beispielsweise in einem regelmäßigen Intervall versendet wird und ein Verlust einer einzelnen Nachricht akzeptiert werden kann, dann kann sie auch als Non-confirmable (NON)-Nachricht verschickt werden. Diese werden nicht mit einem ACK quittiert, besitzen aber eine Message-ID zur Duplikaterkennung. Jedoch können sie genau wie CON-Nachrichten auch mit einem RST beantwortet werden.

5. Request-Response

CoAP bietet zwei verschiedene Möglichkeiten um auf Requests zu antworten. Im folgenden werden beide Varianten kurz erläutert und deren Daseinsberechtigung begründet.

5.1. Piggy-backed Response

Bei einer piggy-backed Response wird die eigentliche Antwort direkt mit dem ACK des Requests mitgesendet. Es wird somit unnötiger Netzwerktraffic eingespart. Im Gegenzug dazu ist es nötig, dass die Antwort quasi sofort verfügbar ist, da für das Senden eines ACKs ein gewisser zeitlicher Rahmen eingehalten werden muss. Ein Beispiel für diese Art der Kommunikation zeigt Abbildung 6.

5.2. Separate Response

Abbildung 7 zeigt einen beispielhaften Nachrichtenaustausch mit separaten Responses. Die Notwendigkeit der separaten Antworten besteht darin, dass eine Antwort eines Netzteilnehmers möglicherweise nicht sofort verfügbar ist. Um das ACK-Timeout einzuhalten ist es somit nötig, einen Request vorerst zu bestätigen und die eigentliche Antwort auf die Anfrage erst zu einem späteren Zeitpunkt abzuschicken. Weiterhin kann ein Request auch vom Typ NON sein, womit ein ACK keine Antwortmöglichkeit darstellt.

Probleme, auf welche man bei dieser Art des Nachrichtenaustauschs stößt, sind das Eintreffen der Response und des ACKs in falscher Reihenfolge, sowie das Request-Response-Matching für beide Nachrichtenteile der Response.

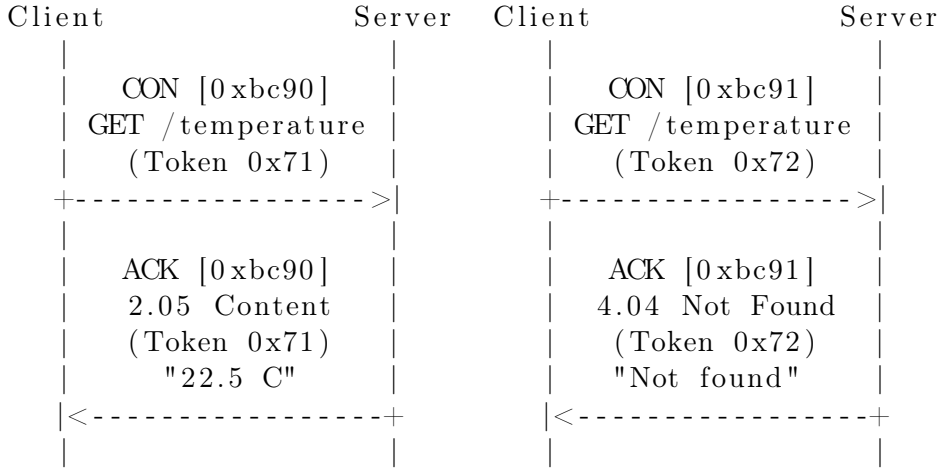


Abbildung 6: Get-Requests mit Piggy-backed Response

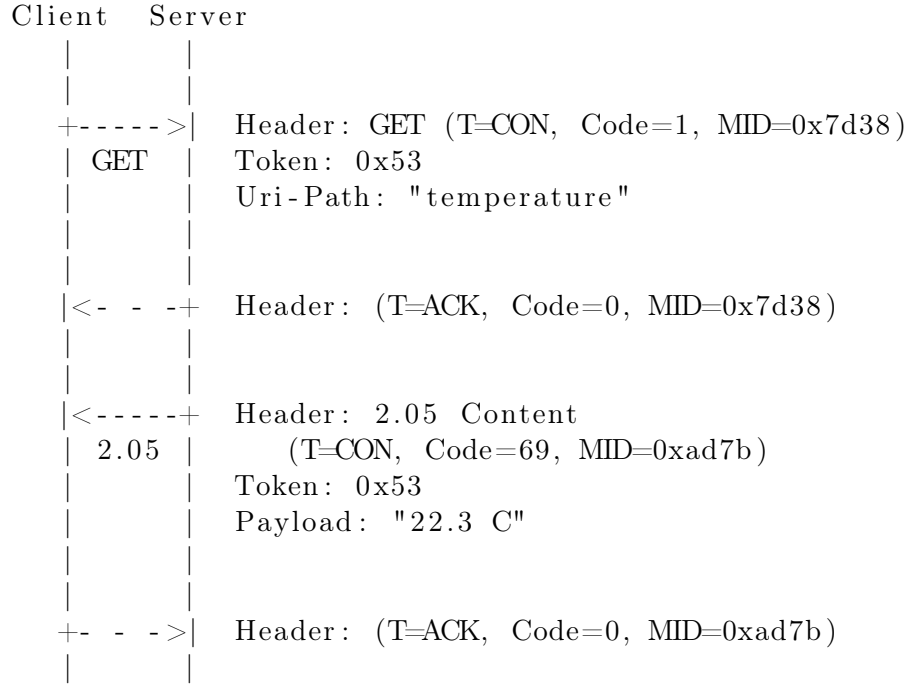


Abbildung 7: Get-Request mit separater Response

5.3. Matching

Das Matching lässt sich mit folgenden Regeln zusammenfassen

Allgemein Response muss vom Endpoint kommen, welcher auch Ziel des Requests war

Piggy-backed MessageID und Token müssen übereinstimmen

Separate nur Token muss übereinstimmen, ggf. MessageID für eine ACK-Message

Der Token ist elementarer Bestandteil jeder Nachricht, er kann 0 bis 8 Byte lang sein und wird vom Client generiert. Er dient der Zuordnung von Responses zu den entsprechenden Requests unabhängig von den eigentlichen Nachrichten. Jeder Request führt somit (ggf. implizit) einen Token mit. Responses auf einen Request können somit eindeutig über die Message-ID, den Token und das Tupel aus Quelle und Ziel der Nachricht identifiziert werden.

5.4. Discover

Um vorhandene Ressourcen eines Servers zu erfahren, bietet CoAP einen Discovery Mechanismus. Als Bedingung für das „Entdecken“ eines Servers muss dieser lediglich auf dem Standardport von CoAP(5683) lauschen. Je nachdem, welche Protokolle CoAP zugrunde liegen, kann dann mittels Unicast, Multicast oder Broadcast der oder die Server abefragt werden. Speziell für das Discover wurde der URI-Path `/.well-known/core` reserviert. Dieser kann mittels einer `GET`-Anfrage abgerufen wurden. Die Antwort enthält dann ein Linkformat (nach [She12]), in dem die angebotenen Ressourcen beschrieben sind.

6. Erweiterungen

CoAP wie es in [GSH+12b] beschrieben ist, bietet die Möglichkeit RESTful-Dienste in einem Low-Power-Netzwerk zu integrieren. Doch neben der Entwicklung des Protokolls selbst findet man auch viele Ideen auf dem Internet Engineering Task Force (IETF)-Datatracker [1]. Einige Überlegungen betreffen die Integration von CoAP in andere Netzwerke, wie zum Beispiel: „Transport of CoAP over Short Messaging Service (SMS), Unstructured Supplementary Service Data (USSD) and General Packet Radio Service (GPRS)“ (siehe [GMBL+13]). Andere Drafts befassen sich mit Sicherheitsaspekten (siehe [GB12]), oder noch genauer mit der Thematik wie man mit schlafenden Knoten in einem Sensornetz umgehen soll (siehe [GR12] und [GRF+12]) und viele weitere mehr. Als Standard-Erweiterungen haben sich die drei folgenden herausgestellt.

6.1. Block - nach [GBZS12]

Für kleine Payloadgrößen muss man sich in CoAP keine Gedanken über Fragmentierung unterliegender Schichten machen. Hat man jedoch größere Datenmengen zu übertragen, so wird einem durch Abbildung 8 schnell bewusst, dass auch beim Einsatz von CoAP mit eventuell vielen Optionen nicht viel Platz für den eigentlichen Payload bleibt. Sprengt man den 127-Byte-Rahmen, der vom Physical Layer (PHY) vorgegeben ist, so müssen die unteren Schichten eine Fragmentierung durchführen. Um dies zu vermeiden, gibt es die Block-Erweiterung in CoAP. Hier wird die Fragmentierung auf die Applikationsschicht angehoben. Dies bringt den Vorteil, dass die unteren Schichten, sprich die

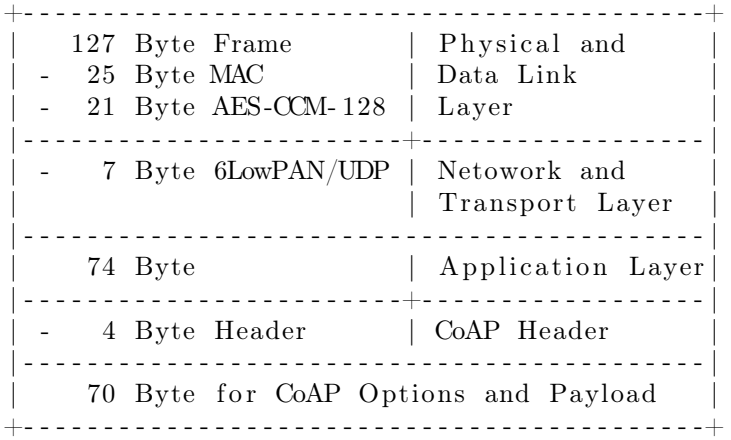


Abbildung 8: Aufteilung des 802.15.4 Frames

Adaptions- oder IP-schicht, nicht damit belastet werden. Ein weiterer Vorteil ist das Acknowledgment der einzelnen Pakete. Tritt ein Fehler auf, so muss nicht das gesamte Paket wiederholt werden, sondern nur ein Teil davon.

[1] `http://datatracker.ietf.org/wg/core/`

Um den Blocktransfer zu beschreiben, werden 3 Attribute in der Block-Option eingeführt.

SZX Die im jeden Request oder Response übermittelte Paketgröße. Sechs verschiedene Größen sind vorgesehen, von $2^4 = 16$ bis $2^{10} = 1024$ Bytes. Die Möglichkeit von sehr hohen Paketgrößen ist für den Einsatz von CoAP mit anderen Schichten gedacht, als beispielsweise mit dem 802.15.4/6LoWPAN/UDP Stack. Bis auf den letzten Block einer Blockübertragung muss der Payload genau diese Größe annehmen. Die Paketgröße wird in einem 3-Bit großen Feld angegeben, in der immer nur der Zweierexponent abzüglich 4 eingetragen ist.

M-Flag Dieses Attribut gibt an, ob es sich um das letzte Paket handelt, beziehungsweise ob noch mehr Pakete vorhanden sind. Die Information wird in einem Bit kodiert.

NUM Die Blocknummer gibt an, welches Paket übermittelt, beziehungsweise angefragt, wurde. Sie beginnt bei 0 und kann bis zu 1048576 (2^{20}) laufen. Die Paketnummer wird entweder in 4,12 oder 20 Bit geschrieben.

Type	C	U	N	R	Name	Format	Length	Default
23	C	U	-	-	Block2	uint	0-3 B	(none)
27	C	U	-	-	Block1	uint	0-3 B	(none)
28	-	-	N	-	Size	uint	0-4 B	(none)

C=Critical, U=Unsafe, N=No-Cache-Key, R=Repeatable

Abbildung 9: Optionen für den Blocktransfer

Wie in Abbildung 9 zu sehen ist, werden zwei Block-Optionen benötigt. Die `Block1`-Option wird verwendet, wenn der Client Daten im Block-Modus zu dem Server senden möchte oder muss, weil der Server es verlangt. Die `Block2`-Option wird verwendet, wenn bei einer `GET`-Anfrage an den Server die Antwort im Block-Modus zurückgeschickt wird. Die genaue Belegung der Attribute je nach Anfrage soll hier allerdings nicht genauer erläutert werden.

Die Block-Option selbst hat keine Beschreibung, wieviele Pakete noch kommen. Es gibt nur eine Paketnummer, die Blockgröße und ein Flag, was besagt, ob noch mehr Blöcke vorhanden sind. Da es in manchen Anwendungen notwendig ist, wurde ebenso auch die optionale Size-Option eingeführt. Mit Hilfe dieser Option kann man die Größe des eigentlichen Payloads angeben.

Die Einführung der Block-Option hebt keinesfalls das REST-Prinzip auf, denn weiterhin beschreiben alle Anfragen sich soweit, dass sie vom Server ausgeführt werden können, ohne sich einen Status zu merken.

6.2. Observe - nach [GH12]

In Anbetracht der Tatsache, dass CoAP dafür entwickelt ist, um mit möglichst wenig Overhead einen RESTful Dienst zu ermöglichen, treten Probleme bei der Anwendung von CoAP in Sensornetzen auf. Dort möchte man beispielsweise recht schnell mitbekommen, wann sich eine Tür geöffnet hat. Es bleibt also nur die Möglichkeit in engen Abständen den Zustand der Tür abzufragen. Gehen wir ein Stück weiter und stellen uns die Tür zu einem sehr selten frequentierten Raum vor, so wird sehr viel Aufwand und Netzlast erzeugt, um in einem aktzeptablen Abstand informiert zu werden, dass sich der Status der Tür geändert hat. Problem hier ist der Mechanismus der REST-Struktur, in der immer der Client eine Anfrage abschickt und eine Antwort erhält und die Kommunikation damit abgeschlossen ist.

6.2.1. Überwachen von Ressourcen

Mit der Idee der Überwachung wird ein neuer Mechanismus in CoAP eingeführt. Anstatt nur einmalig eine Antwort vom einen Server auf eine Anfrage zu bekommen, ist es möglich sich von dem Server über die Repräsentation einer Ressource informieren zu lassen. Das Observe-Design-Pattern (ODP)[GHJV95] wurde in CoAP integriert mit der Observe-Option, welche in Abbildung 10 dargestellt ist. Alle REST-Eigenschaften

No.	C	U	N	R	Name	Format	Length	Default
6		x	x		Observe	empty/uint	0 B/0-2 B	(none)

C=Critical, U=Unsafe, N=No-Cache-Key, R=Repeatable

Abbildung 10: Die Observe-Option

bleiben dabei erhalten, doch ist es nun einem Client möglich auf eine Anfrage mehrere Antworten zu bekommen. Um dies zu realisieren, gibt es die Möglichkeit sich bei dem Server als Observer in eine Liste eintragen zu lassen. Ist man in der Liste, so erhält man von nun an bei Statuswechseln, gewissen abgelaufenen Zeitabständen eine Notification über die neue/aktuelle Repräsentation der überwachten Ressource. Die genaue Implementation soll in diesem Dokument hier nicht dargestellt werden, doch sei darauf hingewiesen, das es ebenso möglich ist die Observe-Option mit der ETag-Option zu kombinieren.

6.2.2. Probleme mit Safe-to-Forward

Seit dem 6. Februar 2013 ist eine Diskussion in der Mailingliste zu beobachten, bei der es darum geht, dass es Probleme seit dem Draft-12 (siehe [GSH+12a]) von CoAP gibt. In diesem Draft wurde die Unterscheidung in Un-/Safe-to-Forward eingeführt. Diese Unterscheidung führt nun dazu, dass Proxys, welche zwar die Observe Funktion verstehen,

aber möglicherweise andere Optionen nicht, welche die Überwachung aber beeinflussen. Ebenso tritt es dadurch auf das Clienten sich gegenseitig aus der Liste der Observer löschen. Derzeit tendiert die Diskussion wohl dahin ein weitere Unterscheidung der Optionen einzubauen. Neben dem Cache-Key, gibt es womöglich demnächst ein Observe-Key, welcher genutzt werden soll um die oben angesprochenen Probleme zu beheben.

6.3. Groupcomm nach [GARED13]

6.3.1. Notwendigkeit

Für CoAP selbst wird nicht geklärt, wie eine Gruppenkommunikation auszusehen hat. Eine Gruppenkommunikation ist jedoch in der Praxis unerlässlich. Ein Beispiel hierfür wäre die Einteilung von Sensoren/Aktoren in einem Netz nach Typ oder Standort. Eine mögliche Gruppe von Aktoren wären alle Lichtschalter im Haus oder alle vorhandenen Thermostate. Es wird Wert darauf gelegt, dass sowohl in CoAP als auch in IP-Multicasts keine Änderungen hierfür vorgenommen werden müssen. Insbesondere wie einzelne Protokolle für verschiedene Szenarien miteinander eingesetzt werden müssen wird in dem Dokument erläutert.

6.3.2. Richtlinien

Da IP-Multicasts weit verbreitet sind, bildet dieser Mechanismus die Grundlage für die Gruppenkommunikation. Um Multicasts über mehrere Subnetze zu ermöglichen ist es nötig Routing-Protokolle oder Forwarding-Protokolle einzusetzen. Für den Einsatz denkbar wäre hier beispielsweise Routing Protocol for Low power and Lossy Networks (RPL) oder Multicast Protocol for Low power and Lossy Networks (MPL). Weiterhin wird noch ein sogenanntes Listener-Protokoll benötigt, um einzelne Geräte Gruppen zuzuordnen.

Die Gruppen-URI muss in der Nachricht die Request-URI sein. Der Authority-Teil muss eine Gruppen-IP-Multicast-Adresse oder ein Hostname sein, welcher zu einer solchen aufgelöst werden kann. Folgende Beispiele für mögliche Gruppen-URIs werden aufgelistet:

all.bldg6.example.com all nodes in building 6

all.west.bldg6.example.com all nodes in west wing, building 6

all.floor1.west.bldg6.examp... all nodes in floor 1, west wing, building 6

all.bu036.floor1.west.bldg6... all nodes in office bu036, floor1, west wing, building 6

Der Gruppen-Fully Qualified Domain Name (FQDN) muss via Domain Name System (DNS) aufgelöst werden können.

Weiterhin wird geklärt, wie ein Knoten die Mitgliedschaft einer Multicast-Gruppe ankündigen kann. Es wird die Vorgehensweise für die Protokolle Multicast Listener Discovery

(MLD), RPL, MPL kurz erläutert, was im Umfang dieses Dokuments hier aber zu weit führen würde.

Für Multi-LoWPAN-Szenarien wird empfohlen, Filtering-Regeln einzuführen, um unnötige Netzlast zu vermeiden.

Filtern durch 6LoWPAN Border Router (6LBR) mit Hilfe von Routinginformationen Dies ermöglicht es dem 6LBR, Multicast-Nachrichten nur in Netze weiterzuleiten, in denen auch ein Empfänger für die Nachricht existiert

Filtern durch 6LBR mit Hilfe von MLD-Reports Gleich dem vorherigen Verfahren, jedoch basierend auf MLD-Reports.

Filtern durch 6LBR mit Hilfe von Einstellungen Nutzen von Filter-Tabellen mit Blacklists bzw. Whitelists für alle 6LBR bzw. für spezifische 6LBR.

6.3.3. Probleme

Nach draft-13 (siehe [GSH+12b]) von CoAP muss die Gruppenkommunikation im No Security (NoSec)-Modus erfolgen. Weiterhin darf kein DTLS-gesichertes CoAP und auch kein IP Security (IPSec) verwendet werden. Aus diesen Gründen müssen Sicherheitsvorkehrungen auf anderen Schichten vorgenommen werden. Für die Zukunft wird darauf spekuliert, dass DTLS-basierte IP-Multicasts oder andere Herangehensweisen die Sicherheitsprobleme lösen könnten. Aktuell ist aber noch kein wirklicher Lösungsansatz für diese Probleme vorhanden.

7. Zusammenfassung

CoAP implementiert eine RESTful Architektur mit möglichst wenig Overhead, Was es vorallem im Einsatz mit Sensornetzen interessant macht: Zum einen ist es möglich, Sensorwerte abzufragen, zum anderen bietet es auch die Möglichkeit, Werte an den Endpoints zu ändern. Durch das Verwenden von UDP als Transportschicht, ist zwar die Zuverlässigkeit nicht mehr gesichert, doch spart man einiges an Overhead gegenüber Transmission Control Protocol (TCP). Ist man daran interessiert, dennoch Zuverlässigkeitsmechanismen zu benutzen, so bietet auch dort CoAP Möglichkeiten. Weiterhin bietet das Verwenden einer REST-Architektur leichte Anbindung an andere REST-Netzwerke wie z.B. HTTP. Ein Mapping von CoAP auf HTTP wurde in diesem Dokument nicht beschrieben, doch ist dies auch bereits in [GSH+12b] beschrieben. Durch Proxys und den Einsatz von den oben beschriebenen Proxy-*-Optionen, ist es möglich auch aus dem CoAP Netzwerk heraus oder hinein Anfragen zu stellen.

Vorallem wenn man die Mailingliste von CoAP verfolgt, stellt man fest, dass es zwar doch noch an einigen Stellen zu nicht-vorhergesehenen Fehlern kommt, doch durch die aktive Arbeit mehrerer Leute wird dies ausführlich diskutiert und Änderungen werden im Draft festgehalten. Trotz dieser auftretenen Probleme ist CoAP in einem Zustand, in dem es implementiert und bearbeitet werden kann. Bei der Implementierung allerdings scheitert es derzeit. Nur für wenige Sprachen sind derzeit Implementierungen einzelner Leute im Internet zu finden. Immerhin für die Programmiersprache C gibt es eine aktive Gruppe[2], welche versucht ihre Implementierung auf dem aktuellen Stand zu halten.

Um CoAP hinsichtlich Effizienz zu beurteilen fehlt es dadurch leider an Testberichten. Doch die Aufmerksamkeit steigt und nachdem nun die Entwicklung einen Stand erreicht hat, indem es sich lohnt, eine Bibliothek zu programmieren, wird es schon möglich sein, bessere Aussagen zu treffen. Aus theorethischer Sicht macht CoAP definitiv Sinn und hat gute Aussichten für die Zukunft, vorallem im Bereich Sensornetzen und anderen Low-Power M2M-Netzwerken.

[2] `http://libcoap.sourceforge.net/`

A. Tabellen

Code	Name	idempotent
1	GET	X
2	POST	
3	PUT	X
4	DELETE	X

Abbildung 11: Request Codes

Code	Description	Reference
65	2.01 Created	[RFCXXXX]
66	2.02 Deleted	[RFCXXXX]
67	2.03 Valid	[RFCXXXX]
68	2.04 Changed	[RFCXXXX]
69	2.05 Content	[RFCXXXX]
128	4.00 Bad Request	[RFCXXXX]
129	4.01 Unauthorized	[RFCXXXX]
130	4.02 Bad Option	[RFCXXXX]
131	4.03 Forbidden	[RFCXXXX]
132	4.04 Not Found	[RFCXXXX]
133	4.05 Method Not Allowed	[RFCXXXX]
134	4.06 Not Acceptable	[RFCXXXX]
140	4.12 Precondition Failed	[RFCXXXX]
141	4.13 Request Entity Too Large	[RFCXXXX]
143	4.15 Unsupported Content-Format	[RFCXXXX]
160	5.00 Internal Server Error	[RFCXXXX]
161	5.01 Not Implemented	[RFCXXXX]
162	5.02 Bad Gateway	[RFCXXXX]
163	5.03 Service Unavailable	[RFCXXXX]
164	5.04 Gateway Timeout	[RFCXXXX]
165	5.05 Proxying Not Supported	[RFCXXXX]

Abbildung 12: Response Codes

Literatur

[FGM+99] Fielding, R. ; Gettys, J. ; Mogul, J. ; Frystyk, H. ; Masinter, L. ; Leach, P. ; Berners-Lee, T.: *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616 (Draft Standard). `http://www.ietf.org/rfc/rfc2616.txt`. Version: Juni 1999 (Request for Comments). – Updated by RFCs 2817, 5785, 6266, 6585

[Fie00] Fielding, Roy T.: *Architectural styles and the design of network-based software architectures*, University of California, Irvine, Diss., 2000. – AAI9980887

[GARED13] Group, CoRE W. ; A. Rahman, Ed. ; E. Dijk, Ed.: *Group Communication for CoAP draft-ietf-core-groupcomm-05*. work in progress. `http://datatracker.ietf.org/doc/draft-ietf-core-groupcomm/`. Version: Februar 2013 (Internet-Draft)

[GB12] Group, CoRE W. ; Bormann, C.: *Using CoAP with IPsec draft-bormann-core-ipsec-for-coap-00*. work in progress. `http://datatracker.ietf.org/doc/draft-bormann-core-ipsec-for-coap/`. Version: Dezember 2012 (Internet-Draft)

[GBZS12] Group, CoRE W. ; Bormann, C. ; Z. Shelby, Ed.: *Block-wise transfers in CoAP draft-ietf-core-block-10*. work in progress. `http://datatracker.ietf.org/doc/draft-ietf-core-block/`. Version: Oktober 2012 (Internet-Draft)

[GH12] Group, CoRE W. ; Hartke, K.: *Observing Resources in CoAP draft-ietf-core-observe-07*. work in progress. `http://datatracker.ietf.org/doc/draft-ietf-core-observe/`. Version: Oktober 2012 (Internet-Draft)

[GHJV95] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design patterns: elements of reusable object-oriented software*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1995. – ISBN 0–201–63361–2

[GMBL+13] Group, CoRE W. ; M. Becker, Ed. ; Li, K. ; Kuladinithi, K. ; Poetsch, T.: *Transport of CoAP over SMS, USSD and GPRS draft-becker-core-coap-sms-gprs-03*. work in progress. `http://datatracker.ietf.org/doc/draft-becker-core-coap-sms-gprs/`. Version: Februar 2013 (Internet-Draft)

[GR12] Group, CoRE W. ; Rahman, A.: *Enhanced Sleepy Node Support for CoAP draft-rahman-core-sleepy-01*. work in progress. `http://datatracker.ietf.org/doc/draft-rahman-core-sleepy/`. Version: Oktober 2012 (Internet-Draft)

[GRF+12] Group, CoRE W. ; Rahman, A. ; Fossati, T. ; Loreto, S. ; Vial, M.: *Sleepy Devices in CoAP - Problem Statement draft-rahman-core-sleepy-problem-statement-01.* work in progress. `http://datatracker.ietf.org/doc/draft-rahman-core-sleepy-problem-statement/`. Version: Oktober 2012 (Internet-Draft)

[GSH+12a] Group, CoRE W. ; Shelby, Z. ; Hartke, K. ; Bormann, C. ; Frank, B.: *Constrained Application Protocol (CoAP) draft-ietf-core-coap-12.* work in progress. `http://datatracker.ietf.org/doc/draft-ietf-core-coap/`. Version: Oktober 2012 (Internet-Draft)

[GSH+12b] Group, CoRE W. ; Shelby, Z. ; Hartke, K. ; Bormann, C. ; Frank, B.: *Constrained Application Protocol (CoAP) draft-ietf-core-coap-13.* work in progress. `http://datatracker.ietf.org/doc/draft-ietf-core-coap/`. Version: Dezember 2012 (Internet-Draft)

[KP08] Klensin, J. ; Padlipsky, M.: *Unicode Format for Network Interchange.* RFC 5198 (Proposed Standard). `http://www.ietf.org/rfc/rfc5198.txt`. Version: März 2008 (Request for Comments)

[She12] Shelby, Z.: *Constrained RESTful Environments (CoRE) Link Format.* RFC 6690 (Proposed Standard). `http://www.ietf.org/rfc/rfc6690.txt`. Version: August 2012 (Request for Comments)

[Yer03] Yergeau, F.: *UTF-8, a transformation format of ISO 10646.* RFC 3629 (INTERNET STANDARD). `http://www.ietf.org/rfc/rfc3629.txt`. Version: November 2003 (Request for Comments)