Constrained Application Protocol (CoAP): Einführung und Überblick

Roman Trapickin
Betreuer: Christoph Söllner
Autonomous Communication Networks (ACN) 2013
Lehrstuhl für Netzarchitekturen und Netzdienste
Lehrstuhl/Fachgebiet für Betriebssysteme
Fakultät für Informatik, Technische Universität München
Email: ga56yit@mytum.de

KURZFASSUNG

In der vorliegenden Arbeit wird das neue Anwendungsschichtprotokoll Constrained Application Protocol (CoAP) vorgestellt. Die Arbeit verschafft einen allgemeinen Überblick über Funktionalität, Aufbau und Struktur sowie Anwendungsbereiche und Integration des neuen Protokolls. Constrained Application Protocol wird im Rahmen des International Engineering Task Force (IETF) standardisiert und befindet sich zum Zeitpunkt der Erstellung dieser Arbeit in der Entwicklungsphase.
CoAP wird speziell für den Einsatz in ressourcenbeschränkten internetfähigen Geräten entwickelt, und soll mit seinem schlanken Aufbau und äquivalentem Leistungsumfang anstelle des Hypertext Transfer Protocol (HTTP) v. a. im Bereich von eingebetteten Systemen zum Nachrichten- bzw. Datenaustausch in Netzwerken mit besonders geringen Übertragungs- und hohen Verlustraten benutzt werden.

Schlüsselwörter

Constrained Application Protocol, CoAP, Einführung, Überblick, 802.15.4, 6LoWPAN, IPv6, UDP, DTLS, Multicasting.

1. EINLEITUNG

Das sogenannte Internet of Things (IoT) beschäftigt sich mit der Einbindung von unterschiedlichen Gegenständen in das Internet. Diese Gegenstände oder *Dinge* werden mit Kommunikationsbausteinen und Sensorik ausgestattet, damit sie Umweltdaten erfassen und weitergeben können. Dies impliziert, dass die Dinge *miteinander* kommunizieren müssen – die sog. Machine-to-Machine-Kommunikation (M2M). M2M steht für einen Nachrichtenaustausch zwischen mehreren Geräten, der meist ohne den menschlichen Eingriff geschieht.
Der IEEE-Standard **802.15.4** definiert Bitübertragungs- sowie Sicherungsschicht für kostengünstige Funkübertragungsnetze mit geringem Energieverbrauch [7, S. 1f.]. IEEE 802.15.4 ermöglicht die maximale Rahmengröße von 127 Bytes [7, S. 171] und eine maximale Datenrate von 250 kbit/s in einem 2,4 GHz-Band [7, S. 163]. Die Zielgeräte von IEEE 802.15.4 befinden sich während ihrer Betriebszeiten größtenteils im energiesparenden Schlafmodus und überprüfen periodisch ihr Übertragungskanal auf eine anstehende Nachricht [7, S. 14f.].
Um die Vorteile eines IPv6-Nachrichtenaustausches für solche Funkübertragungsnetze zu nutzen (z. B. Multicasting), definiert die IETF **6LoWPAN** Working Group die Übertragung von IPv6-Paketen über IEEE 802.15.4-Netzwerke [8] mittels einer *Anpassungsschicht* (Adaptation Layer). Die 6LoWPAN-Anpassungsschicht löst v. a. das Problem, dass die minimal unterstützte MTU (Maximum Transmission Unit) von IPv6 1280 Bytes beträgt und somit nicht in ein IEEE 802.15.4-Rahmen passt [8, S. 4], indem der IPv6-Nachricht von einem 6LoWPAN-Header gekapselt wird [8, S. 5ff.]. Ferner kann ein IPv6-Header komprimiert [8, S. 15ff.] und die Nachricht fragmentiert werden [8, S. 10ff.], sodass mehr Freiraum für Nutzdaten geschaffen wird. 6LoWPAN greift ebenso in die nächsthöhere Transportschicht und komprimiert Protokolle UDP, TCP und ICMP [8, S. 16f.].
Die IETF Constrained RESTful environments (CoRE) Working Group definiert ein neues Anwendungsprotokoll, das Constrained Application Protocol – **CoAP**, das speziell für ressourcenbeschränkte Geräte entwickelt wird [12]. Als Beispielzielgruppe von CoAP werden mit 8-Bit-Mikrocontrollern ausgestattete Geräte (Knoten) mit wenig ROM und RAM in verlustbehafteten (lossy) Netzwerken mit geringer Übertragungsrate angegeben [12]. Abbildung 1 verdeutlicht die Position von CoAP im OSI-Schichtenmodell für ressourcenbeschränkte Umgebungen.

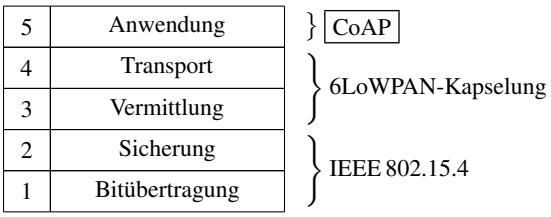

Abbildung 1: CoAP im OSI-Modell.

2. NACHRICHTENFORMAT

CoAP wird hauptsächlich für Anwendung mit UDP entwickelt und belegt somit das Datenfeld des UDP-Datagramms. Eine Übertragung über UDP ist jedoch nicht zwanghaft, so ist es bspw. möglich, CoAP zusammen mit TCP, SCTP oder SMS zu verwenden [12, S. 15]. Ist eine sichere Datenübertragung gewünscht, so wird zusätzlich das Verschlüsselungsprotokoll DTLS benutz [12, S. 6].
Eine CoAP-Nachricht besteht aus einem 4-Byte-Header sowie von Token, Optionen und schließlich Nutzdaten [12, S. 15].

2.1 Header

Abbildung 2 stellt den Aufbau des CoAP-Header grafisch dar.

Ver ist die Protokollversion. Diese wird mit einer 2-Bit-Zahl (vorzeichenlos) gefüllt. Das Feld muss für die aktuelle Implementierung auf binäre Eins (0b 01) gesetzt werden [12, S. 16].

doi: 10.2313/NET-2013-08-1_16

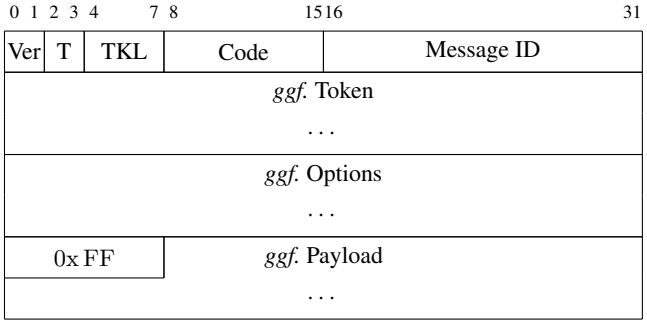

Abbildung 2: Aufbau einer CoAP-Nachricht [12, S. 15].

T ist der Nachrichtentyp. Das Feld ist 2 Bit breit und kann mit folgenden Werten belegt werden: Confirmable (0b 00), Non-confirmable (0b 01), Acknowledgement (0b 10) und Reset (0b 11) [12, S. 16]. Diese werden im Kap. 3.1 näher erläutert.

TKL ist die Länge des Token-Feldes in Bytes. Das 4-Bit-Feld kann entsprechend mit den Werten von 0 bis 15 belegt werden. In der aktuellen Implementierung wird jedoch die Länge auf max. 8 Bytes beschränkt, sodass die Werte 9–15 als Formatfehler betrachtet werden sollen [12, S. 16]. Tokens werden im Kap. 3.1 näher erläutert.

Code ist 8 Bit breit und gibt *numerisch* an, ob die Nachricht ein Request, eine Response oder gar leer ist. Die 3 höherwertigen Bits stellen die Code-Klasse dar, wobei die restlichen 5 Bits das Detail [12, S. 16]. Die Request/Response-Codes werden im Kap. 3.2 näher erläutert.

Message ID ist die Identifikationsnummer einer Nachricht innerhalb des Austauschprozesses. Die Message ID dient zur Zuweisung von Nachrichten vom Typ Confirmable zu Acknowledgement bzw. von Non-confirmable zu Reset. Des Weiteren wird es bei der Deduplizierung von Nachrichten benutzt. Das Feld ist 16 Bit breit, wobei die zwei Bytes im Network Byte Order angeordnet sind [12, S. 16]. Nachrichtentypen und Deduplizierung werden im Kap. 3.1 näher erläutert.

Die 32 Bits von Kopfdaten werden von einem **Token** (max. 8 Bytes; s. TKL) sowie von **Options** (dazu Kap. 2.2) gefolgt. Die Nutzdaten (Payload) werden von Options durch den **Payload Marker** mit dem festen 8-Bit-Wert 0x FF getrennt. Falls keine Nutzdaten übertragen werden, so wird auf den Marker verzichtet. Die Nutzdaten erstrecken sich bis zum Ende des UDP-Datagramms; deren Länge kann daher mit Hilfe des UDP-Längenfeldes errechnet werden [12, S. 16f.].

2.2 Options

Ähnlich einem HTTP-Header, kann CoAP zusätzliche Parameter (z. B. Host oder Content-Type) für Anfrage- und Antwortmethoden enthalten. Diese Argumente werden jedoch nicht explizit als ASCII-Strings gespeichert, sondern durch eigene Nummern eindeutig identifiziert. Ein Bündel aus einer solchen Nummer und einem zugehörigen Wert wird als *Option* bezeichnet [12, S. 17ff.]. Eine CoAP-Nachricht kann sowohl mehrere (nacheinander folgende) als auch gar keine Optionen enthalten [12, S. 15f.].
Eine Optionsnummer ist eine vorzeichenlose ganze Zahl aus dem 16-Bit-Bereich (vgl. Tab. 1). Diese Nummern sollen zukünftig in RFC-Memoranda referenziert werden. Gemäß Richtlinie RFC 5226 werden die meistbenutzten Nummern von 0 bis 255 durch IETF definiert (das Zuordnungsverfahren ist im [12, S. 90] definiert); die Restlichen werden durch die Internet Assigned Numbers Authority (IANA) vergeben [12, S. 88]. Die Oprtionsnummern werden

Nr.	Name	HTTP-Äq.	Länge [B]
1	`If-Match`	*gleichnamig*	0–8
3	`Uri-Host`	*vgl. Kap. 2.2.1*	1–255
4	`ETag`	*gleichnamig*	1–8
5	`If-None-Match`	*gleichnamig*	0
7	`Uri-Port`	*vgl. Kap. 2.2.1*	0–2
8	`Location-Path`	`Location`	0–255
11	`Uri-Path`	*vgl. Kap. 2.2.1*	0–255
12	`Content-Format`	`Content-Type`	0–2
14	`Max-Age`	*vgl. Kap. 3.2*	0–4
15	`Uri-Query`	*vgl. Kap. 2.2.1*	0–255
16	`Accept`	*gleichnamig*	0–2
20	`Location-Query`	`Location`	0–255
35	`Proxy-Uri`	*vgl. Kap. 4.1*	1–1034
39	`Proxy-Scheme`	*vgl. Kap. 4.1*	1–255

Tabelle 1: Optionsnummern [12, S. 88].

durch das sog. Options Delta (OΔ) repräsentiert [12, S. 17]. Das OΔ ist die Differenz zwischen den Nummern der aktuellen und der vorherigen Option und wird mit folgender Rekursionsgleichung 1 berechnet. In anderen Worten, ergeben alle OΔ_i aufsummiert die aktuelle Option-Nummer. Der Vorteil solcher Repräsentation wird erst dann offensichtlich, wenn man die Forderung nach Reihenfolge von Optionen betrachtet. Diese *müssen nämlich streng aufsteigend nach ihrer Nummer sortiert werden*. Man betrachte dazu die Abbildung 3.

$$\begin{aligned} \mathrm{O}\Delta_1 &= N_1 \\ \mathrm{O}\Delta_i &= N_i - N_{i-1} \end{aligned} \qquad N_i \overset{!}{\geq} N_{i-1} \tag{1}$$

Eq. 1: Berechnung von OΔ. N_i ist die i-te Optionsnummer (vgl. [12, S. 17f.]).

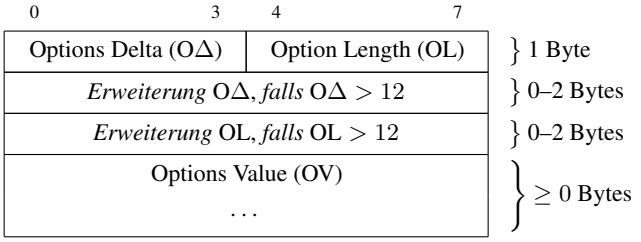

Abbildung 3: Aufbau von Optionsfeldern [12, S. 18].

Options Delta (OΔ) ist 4 Bit breit. Ist OΔ-Wert größer 12, so wird 13 geschrieben, ein zusätzliches Byte nach Option Length reserviert und mit O$\Delta-13$ belegt. Ist OΔ größer als $255+13 = 268$, so wird das Feld nochmals um ein Byte erweitert, das ursprüngliche Feld mit 14 belegt und in die beiden zusätzlichen Bytes O$\Delta - 269$ gemäß Network Byte Order geschrieben. Der Wert 15 ist für den Payload Marker reserviert. Ist O$\Delta = 15$ und OL $\neq 15$, so ist es als Formatfehler zu betrachten. Sind beide Felder 15 (0x FF), so folgen keine Optionen mehr, sondern nur noch die Nutzdaten [12, S. 20].

doi: 10.2313/NET-2013-08-1_16

Option Length (OL) ist 4 Bit breit. Die Länge von Options Value in Bytes wird analog zu OΔ erweitert, falls diese den Wert 12 überschreitet. Die Belegung O$\Delta \neq 15$ und OL $= 15$ ist zwar für zukünftige Benutzung reserviert, wird aktuell jedoch ebenfalls als Formatfehler betrachtet [12, S. 20].

Option Value (OV) ist eine Bytesequenz, deren Länge dem Wert von OL-Feld exakt entspricht. OV beschreibt den Wert bzw. das Argument einer Option. Der Datentyp (z. B. `string`, `uint` oder beliebig) ist von der Option abhängig [12, S. 19].

Schließt man bspw. nur die Optionsnummern 8 und 20 aus der Tabelle 1 in die Optionsfelder ein, so beträgt O$\Delta_1 = 8$ und O$\Delta_2 = 12$. Anstatt die zwei Zahlen in der 16-Bit-Darstellung (also jeweils 2 Bytes) zu speichern, werden nur noch 4 Bits pro OΔ-Feld benötigt. Die Ersparnis im Vergleich zu einem doppelten 2-Byte-Feld beträgt somit 3 Bytes. Durch aufsteigende Reihenfolge von Nummern wird sowohl OΔ minimal gehalten als auch ein negatives OΔ verhindert. Da sich die meistbenutzten Optionsnummern im Bereich 0–255 befinden, wird im schlimmsten Fall ein einziges OΔ-Zusatzfeld benötigt. Die Optionsnummern 256–65535 sind für weitere öffentliche, private, herstellerspezifische sowie experimentelle Zwecke reserviert. Der Extremfall mit zwei Zusatzfeldern, bei dem durch das initiale OΔ-Feld ein halbes Byte Overhead entsteht, sollte somit nur noch ausnahmsweise auftreten (vgl. [12, S. 88f.]).
Ähnlich zu Optionsnummern, werden auch einige Optionswerte kodiert: Tabelle 2 beinhaltet einige Identifikationsnummern für `Content-Format`, die durch 0 bis 2 Bytes dargestellt werden können. Alle vorgeführten Werte aus dieser Tabelle passen entsprechend in ein einziges Byte. Für die Optionswertelänge OL gilt das

Internet Media Type	Id.
`text/plain; charset=utf-8`	0
`application/link-format`	40
`application/xml`	41
`application/octet-stream`	42
`application/exi`	47
`application/json`	50

Tabelle 2: Einige mögliche OV von `Content-Format` [12, S. 90].

gleiche Prinzip wie für OΔ: Mit Ausnahme von `Proxy-Uri` haben alle Optionen aus der Tabelle 1 ihre Werte nicht länger als 255 Bytes, was ebenfalls in einem einzigen OL-Zusatzfeld resultiert.

2.2.1 URI

CoAP besitzt ein eigenes URI-Schema, das stark dem von HTTP ähnelt, unterscheidet sich aber in wenigen Einzelheiten. Das URI-Schema von CoAP wird in der Abbildung 4 dargestellt. Der rein

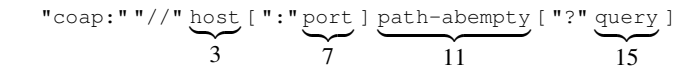

Abbildung 4: `coap`-URI-Schema mit Optionsnummern einzelner Komponenten [12, S. 58f.]

visuelle Unterschied besteht als Erstes in der Angabe von Pfaden nach der Host-Adresse. Die Host-Adresse (Option `Uri- Host`) darf nicht leer sein. Da die CoAP-Option `Uri-Path` die Länge von 0 Bytes haben darf, sind zwei aufeinander folgende Schrägstriche nach `host` möglich: `path-abempty` gibt entweder einen absoluten oder einen leeren Pfad an. In HTTP dagegen dürfen diese ausschließlich einmal nach Schemaangabe folgen. Als Zweites, sind für CoAP keine Fragmente (Komponenten, die mit „#“ beginnen) definiert [12, S. 59f.].
Der signifikante Unterschied zwischen zwei Protokollen ist jedoch die Dekomposition, also Zerlegung von URI in einzelne Komponente und deren Einbindung in den Header. HTTP speichert stets den absoluten Pfad und schließt bei einigen Anfragemethoden (z. B. `GET` oder `HEAD`) sogar die Abfragen (`query`) mit ein, wogegen CoAP alle Pfadstücke und Abfragen immer *getrennt in einzelne Optionen* speichert. Dasselbe gilt auch für Host-Adressen und Port-Nummern: Sie werden im Gegensatz zu HTTP ebenfalls voneinander getrennt gespeichert. Die jeweiligen URI-OΔ_i passen beim Zusammenbau des Headers stets in das initiale 4-Bit-Feld (vgl. Tab. 1).
Kann eine Ressource sicher angesprochen werden, so wird anstatt `coap` die Schemaangabe `coaps` (CoAP Secure) benutzt (dazu Kap. 4.4).

3. INTERAKTIONSMODELL

Das Interaktionsmodell von CoAP ist in zwei Schichten aufgeteilt [12, S. 9f.]:

- Nachrichtenschicht (Messaging Layer)
- Anfrage- /Antwort-Schicht (Request / Response Layer)

Auf der Nachrichtenschicht werden Nachrichten zwischen zwei oder mehr Endpunkten (Anwendungen) ausgetauscht. Auf dieser Schicht werden die Zuverlässigkeitsnachteile von UDP kompensiert. UDP ist verbindungslos, nicht zuverlässig und unsicher, dafür ist es mit seinem 8-Byte-Header wesentlich sparsamer als eine TCP-Nachricht. Für CoAP heißt dies, es muss eigene Zuverlässigkeits-, Koordinations-, Staukontroll- und Deduplizierungsmechanismen implementieren [12, S. 10ff.]. Dazu hat CoAP vier verschiedene Nachrichtentypen (dazu das nächste Kapitel).
Das Interaktionsmodell auf Anfrage-/Antwort-Schicht von CoAP ähnelt dem Client-Server-Modell von HTTP, in einer M2M-Kommunikation übernehmen die Knoten jedoch *beide Rollen zugleich* [12, S. 12ff.].
Der Client sendet eine Anfrage an den Server, spricht also mittels einer Operation (oder Methode) einen Dienst an, der vom Server angeboten wird. Der Server antwortet je nach Operation. Anfrage-/Antwort-Kommunikation von CoAP erfüllt alle Zwangsmäßigkeiten einer REST-Architektur: Ressourcen (Daten unterschiedlicher Formate – Repräsentation) werden durch URI identifiziert und mittels Methoden vom Server abgerufen und verarbeitet.
CoAP ist hauptsächlich ein Anwendungschichtprotokoll (OSI-Schicht 7 bzw. DoD-Schicht 4), greift jedoch in die Kompetenzen von Transportschichtprotokollen, indem es z. B. das Mechanismus zur Staukontrolle besitzt. Bei einer sicheren Verbindung bildet DTLS eine zusätzliche Schicht zwischen UDP und CoAP (vgl. Abb. 5).

3.1 Nachrichtenaustausch

CoAP-Nachrichten werden zwischen mehreren Endpunkten ausgetauscht. Es gibt vier Typen von Nachrichten [12, S. 8]:

Confirmable Message (`CON`) ist eine Nachricht, die Bestätigung erwartet. Auf diese Weise implementiert CoAP die bei UDP fehlende Zuverlässigkeit. Falls keine Nachrichten verloren gegangen sind, so wird jedem Confirmable Message eine Bestätigung vom Typ Acknowledgement oder vom Typ Reset

doi: 10.2313/NET-2013-08-1_16

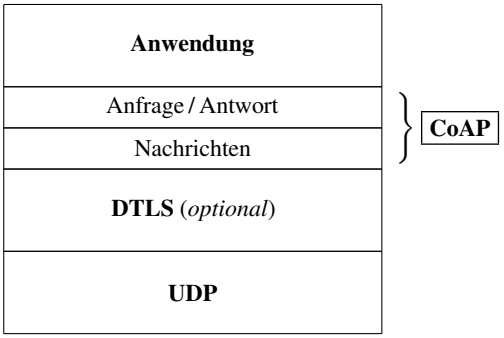

Abbildung 5: CoAP im Protokollstapel [12, S. 68].

zugeordnet. Wird nach einer festgelegten Zeitspanne keine Bestätigung erhalten, so kann die Nachricht ggf. noch einmal versandt werden (dazu Kap. 3.1.1).

Non-confirmable Message (`NON`) muss hingegen nicht bestätigt werden. Es heißt also, der Absender erhält keine Informationen darüber, ob die Nachricht jemals den Empfänger erreicht hat (dazu Kap. 3.1.2).

Acknowledgement (`ACK`) ist die Bestätigung einer Nachricht vom Typ Confirmable. Mit Acknowledgement wird der Absender vom Empfänger informiert, dass der letztere seine Nachricht empfangen hat.

Reset (`RST`) wird als Antwort auf eine Confirmable oder Non-confirmable Nachricht gesendet, falls aus der Sicht des Empfängers der Nachrichtenkontext verloren gegangen ist. In anderen Worten, der Empfänger war überhaupt nicht in der Lage, die Anfrage zu bearbeiten, sodass selbst kein Fehlercode zugeordnet und zurückgeschickt werden kann. Dies ist z. B. der Fall, wenn der Empfänger(-Server) neugestartet wurde. Zum anderen kann der Empfänger durch eine leere Confirmable Nachricht dazu veranlasst werden, eine Reset Nachricht zurückzuschicken und somit Informationen über seine Erreichbarkeit zu vermitteln (CoAP Ping).

Im Kap. 2.1. wurde das 2-Byte-Feld Message ID vorgestellt. Diese Identifikationsnummer dient dazu, einer Confirmable Nachricht ein Acknowledgement eindeutig zuzuordnen. Im Falle einer Nachricht vom Typ Non-confirmable wird Message ID empfängerseits zur Erkennung von Duplikaten einer bereits erhaltener Nachricht verwendet (Deduplizierung). Eine Reset-Nachricht muss ebenfalls einem Confirmable oder Non-confirmable Message zugeordnet werden können; Somit ist die Message ID ein Pflichtfeld jeder CoAP-Nachricht [12, S. 10ff.].

3.1.1 Confirmable Message

Eine Confirmable Nachricht erwartet stets das Acknowledgement oder Reset. Eine erneute Übertragung (Retransmission) geschieht im Falle einer fehlender Bestätigung oder Reset-Nachricht, wenn die maximale Wartedauer überschritten wurde und gleichzeitig die maximale Anzahl an Retransmissions noch nicht erreicht wurde. Die Wartedauer wird durch Parameter `ACK_TIMEOUT` angegeben. Anfangs wird die Wartedauer zufällig als ein Wert zwischen `ACK_TIMEOUT` und `ACK_TIMEOUT` multipliziert mit `ACK_RANDOM_FACTOR` ausgewählt (oft keine ganze Zahl). Der Retransmissions-Zähler wird dabei auf 0 gesetzt. Kommt innerhalb der Wartedauer kein Acknowledgement oder Reset zurück, so wird *der Zähler inkrementiert und die Wartedauer verdoppelt*. Dies wird so lange wiederholt, bis entweder Acknowledgement/Reset kommt oder `MAX_RETRANSMIT` (max. Anzahl an Retransmissions) erreicht wird. Hat der Zähler dieses Maximum erreicht, so wird die Übertragung abgebrochen und die Applikation über das Scheitern informiert (vgl. Abb. 6, [12, S. 21f.]). Dieses Verfahren gehört zu Exponential-

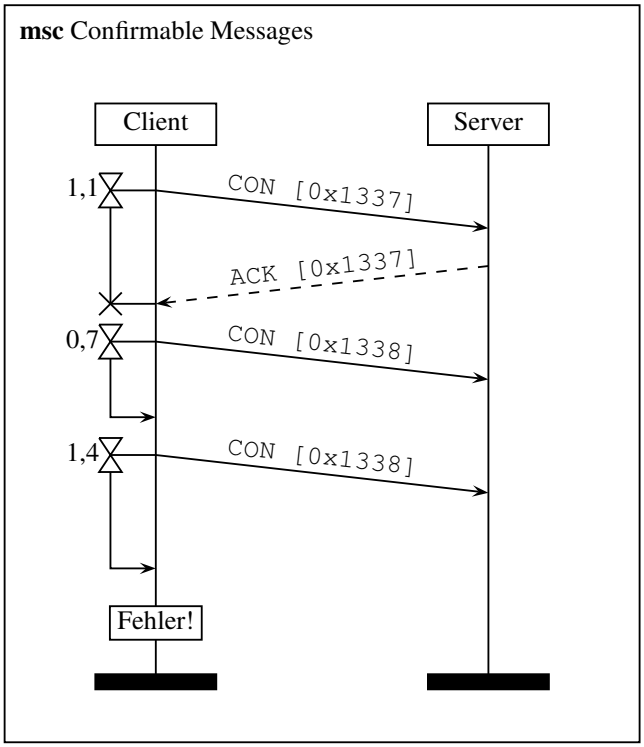

Abbildung 6: Zuverlässige Nachrichtenübertragung. Die Beispielswerte sind: `ACK_TIMEOUT` = 0,5; `ACK_RANDOM_FACTOR` = 4; und `MAX_RETRANSMIT` = 1. (vgl. [12, S. 19f.], [12, S. 102ff.])

Back-off-Algorithmen und wird v. a. auch zur Stauauflösung benutzt. Die Wartedauer wird mit $2^n \cdot t$ berechnet, wobei n die Anzahl an Retransmissions und t die initiale Wartedauer ist. Die Wartedauer wächst bzw. die Übertragungsrate schrumpft *exponentiell* [12, S. 12].

3.1.2 Non-confirmable Message

`NON`-Nachrichten sind z. B. dann sinnvoll, wenn im Verlustfall die Information wertlos wird und ein wiederholtes Versenden zwecklos ist. Abbildung 7 stellt die nicht-zuverlässige Nachrichtenübertragung mittels `NON` grafisch dar.

3.2 Anfrage / Antwort

Ebenso wie HTTP, macht sich CoAP ein Request-Response-Modell zunutze. Bei diesem Modell gibt es zwei Arten von Nachrichten: Ein Request (eine Anfrage an den Server) und eine Response (also die Antwort bzw. Reaktion des Servers auf diese Anfrage). In HTTP wird ein Methodenaufruf (z. B. `GET` oder `POST`) an den Server gesendet, wobei der letztere mit einem Statuscode antwortet (z. B. „200 OK“ oder „404 Not Found“) [12, S. 30ff.].
Doch wenn HTTP ihre Anfragen sowie Antworten menschenlesbar als ASCII-Strings speichert, wird bei CoAP pro Anfragemethode bzw. pro Antwortmeldung ein Code zugewiesen. Dieser Code belegt das 8-Bit-Feld des Headers (vgl. Kap. 2.1.) und wird mit der Maske `c.dd` gebildet: Der Teil `c` ist drei Bit breit und stellt die Klasse dar;

doi: 10.2313/NET-2013-08-1_16

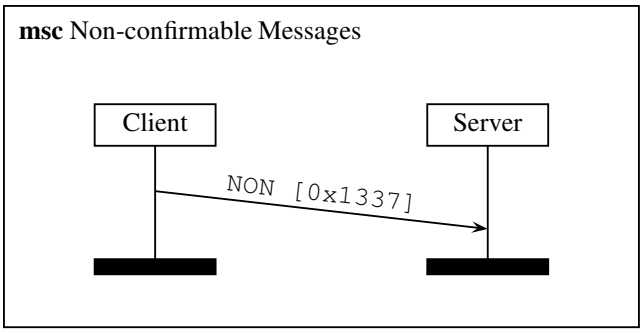

Abbildung 7: Nicht-zuverlässige Nachrichtenübertragung (vgl. [12, S. 11])

Der Teil `dd` ist fünf Bit Breit und steht für Detail. So steht der Code 0.00 für eine leere Nachricht, die Codes 0.`dd` für Anfragemethoden und 2.`dd`–5.`dd` für Antwortcodes [12, S. 86].
Um einer Anfrage (Request) eine Antwort (Response) eindeutig zuweisen zu können, sind in CoAP die sog. Tokens definiert. Ein Token ist max. 8 Byte breit und kann für ein Anfrage-Antwort-Bündel beliebig generiert werden. Ein Token ist unabhängig von der Message ID: Eine Antwort kann auch mit einer anderen Nachricht folgen; Ausschlaggebend für die Anfrage-Antwort-Zuweisung ist nur der Token [12, S. 34].
Möchte man die Antwortzeiten reduzieren, so kann CoAP auf Response-Caching zugreifen. Anstatt also, eine neue Antwort zu erzeugen, kann der Server eine bereits generierte Antwort zurückschicken, falls die Antwortnachricht gültig ist (die Option `Max-Age` der Nachricht überschreitet nicht einen bestimmten Zeitwert), oder mittels `ETag`-Option eine zwischengespeicherte Nachricht revalidieren [12, S. 41].
Ferner, kann man einen Proxyserver beauftragen, für seine Clients die Anfragen zu übermitteln. Dies kann nützlich sein, falls sonst keine Anfragegestaltung möglich ist, oder wenn man das Caching-Potenzial ausschöpfen möchte um Antwortzeiten und Energieverbrauch zu verringern [12, S. 43ff.].

3.2.1 Request

CoAP definiert insgesamt vier Anfragemethoden: `GET`, `POST`, `PUT` und `DELETE` (vgl. Tab. 3). Die restlichen 27 Codes aus dem Bereich 0.05–0.31 sind für zukünftige Benutzung reserviert [12, S. 84].
Mit der Methode `GET` wird mittels URI angegebene Ressource angefordert, mit `POST` werden die Daten an den Server zur Verarbeitung geschickt, mit `PUT` wird eine Ressource aktualisiert bzw. angelegt, und mit `DELETE` wird Ressource vom Server gelöscht. Diese Methoden stimmen mit den HTTP-Methoden überein, sodass ein HTTP-Mapping von `GET`, `POST`, `PUT` und `DELETE` problemlos umgesetzt werden kann (vgl. Kap. 4.1).

Code:	0.01	0.02	0.03	0.04
Name:	`GET`	`POST`	`PUT`	`DELETE`

Tabelle 3: CoAP-Codes von Anfragemethoden [12, S. 85].

3.2.2 Response

Es gibt drei Möglichkeiten, Responses zu versenden: **Piggy-backed**, **Separate** und **Non-confirmable**. Eine Piggy-backed Response ist eine Antwort, die mit einer Acknowledgement-Nachricht zugestellt wird. Manchmal ist es nicht möglich (z. B. wegen langer Bearbeitungszeit), die Antwort gleich zu verschicken, so wird zuerst eine leere Acknowledgement und erst später die Antwort in einer *separaten* Confirmable-Nachricht geschickt. Letztendlich, falls die Anfrage mit einer Non-confirmable-Nachricht empfangen wurde, so muss die Antwort ebenfalls als Non-confirmable versandt werden [12, S. 34ff.].
Die Antwortcodes sind in drei Klassen geteilt [12, S. 32]:

Success (2.`dd`) – Anfrage wurde erfolgreich bearbeitet.

Client-Error (4.`dd`) – Anfrage enthält Syntax- bzw. Formatfehler und kann nicht bearbeitet werden

Server-Error (5.`dd`) – anscheinend gültige Anfrage konnte vom Server nicht bearbeitet werden

Einige Antwortcodes entsprechen zwar den HTTP-Statuscodes (z. B. 4.04 und 404 „Not Found“), andere dagegen haben unterschiedliche Codes (2.05 „Content“ ist äquivalent zu 200 „OK“; 2.05 wird jedoch nur als Antwort auf `GET` benutzt) oder sind gar nicht vertreten (vgl. Tab. 4).

Code	Beschreibung	Code	Beschreibung
2.01	Created	4.05	Method Not Allowed
2.02	Deleted	4.06	Not Acceptable
2.03	Valid	4.12	Precondition Failed
2.04	Changed	4.13	Request Entity Too Large
2.05	Content	4.15	Unsupported Content-Format
4.00	Bad Request	5.00	Internal Server Error
4.01	Unauthorized	5.01	Not Implemented
4.02	Bad Option	5.02	Bad Gateway
4.03	Forbidden	5.03	Service Unavailable
4.04	Not Found	5.04	Gateway Timeout
		5.05	Proxying Not Supported

Tabelle 4: CoAP-Antwortcodes [12, S. 86].

4. WEITERE FUNKTIONEN

Neben dem vorgestellten zweischichtigen Interaktionsmodell, besitzt CoAP weitere Funktionen wie HTTP-Proxying, Multicasting sowie Service & Resource Discovery. Außerdem kann CoAP-Nachrichtenaustausch mittels Sicherheitsmechanismen abhörsicher gemacht werden.

4.1 HTTP-Proxying

Es ist möglich, dass verschiedene Geräte nur eines der Protokolle CoAP oder HTTP implementieren. Daher ist es sinnvoll, dass Nachrichten durch eine Vermittlungsinstanz bzw. Proxy übersetzt werden können (vgl. Abb. 8). Man unterscheidet zwei Arten des *Cross-Protocol Proxying*:

CoAP-HTTP Proxying erlaubt den Zugang zu den Ressourcen eines HTTP-Servers für CoAP-Clients. Dafür müssen die Optionen `Proxy-Uri` und `Proxy-Scheme` entsprechend mit einer HTTP-URL bzw. `http` (oder `https`) belegt sein (vgl. Tab. 1). Da CoAP-Methoden äquivalent zu HTTP-Methoden sind, kann die Konstruktion von HTTP, TCP sowie ggf. TLS problemlos erfolgen. Je nachdem, ob der Proxy die Anfragen nicht weiterleiten kann, die Antwortwartezeit überschritten wurde oder die Antwort vom Proxy nicht bearbeitet werden konnte; liefert Proxy entsprechend die Codes 5.05, 5.04 bzw.

doi: 10.2313/NET-2013-08-1_16

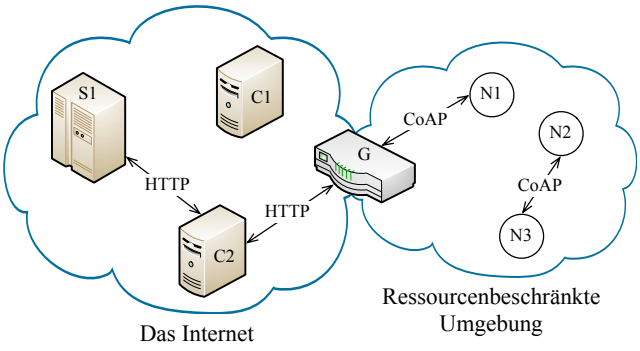

Abbildung 8: HTTP ↔ CoAP. C2 verbindet sich mit N1 über Gateway G.

5.02 zurück (vgl. Tab. 4). Eine genauere Gegenüberstellung von HTTP- und CoAP-Anfragemethoden sowie Status- bzw. Antwortcodes ist im [12, S. 73ff.] zu finden.

HTTP-CoAP Proxying erlaubt den Zugang zu den Ressourcen eines CoAP-Servers für HTTP-Clients. Um eine HTTP-Anfrage an den Proxy zu senden, muss der Client den absoluten Pfad zur Ressource inklusive Schemaangabe (`coap`/`coaps`) im Methodenaufruf angeben. Sobald der Proxy die Nachricht empfangen hat, wird er die angegebene CoAP-Resource anfordern. Alle HTTP-Methoden aus Spezifikation RFC 2616, mit Ausnahme von `OPTIONS`, `TRACE` und `CONNECT` lassen sich ebenfalls in CoAP umwandeln. Die Methode `HEAD` lässt sich zwar nicht direkt übersetzen, der Proxy kann jedoch `HEAD` durch ein CoAP-`GET` ersetzen und die Antwort ohne den Nachrichtenkörper an den HTTP-Knoten zurückschicken. Ähnlich zu CoAP-HTTP-Proxying werden nun HTTP-Statuscodes „501 Not Implemented“ (Methode vom Proxy nicht unterstützt), „502 Bad Gateway“ (Proxy hat eine ungültige Antwort vom Empfänger bekommen) oder „504 Gateway Timeout“ (Proxy hat keine rechtzeitige Antwort vom Empfänger bekommen) im Fehlerfall geliefert [12, S. 76ff.]. Optimale Vorgehensweisen für das HTTP-CoAP-Mapping werden in [3] diskutiert.

4.2 Service & Resource Discovery

In einer M2M-Kommunikation ist es erforderlich, dass das Auffinden (Discovery) von Ressourcen und Servern ohne den menschlichen Eingriff geschehen kann.

In CoAP kann ein Server durch seine URI entdeckt werden, d. h. der Client weiß bescheid über den Server, oder zu diesem Server weitergeleitet wird. Ist der Server nirgendwo durch eine Ressource referenziert, so kann man eine CoAP-Multicast-Anfrage an alle Knoten mittels „All CoAP Nodes“-Adresse verschicken (dazu das nächste Kapitel).

RFC 6690 [11] definiert das sog. Well-Known Interface für CoRE Resource Discovery. Das Interface liefert eine Liste von verfügbaren Ressourcen im CoRE Link Format. Das CoRE Link Format wird ebenfalls in RFC 6690 definiert und ist ein neues Internet Media Type (`Content-Format` – vgl. `application/link-format` aus der Tabelle 2). Um auf das Well-Known Interface zuzugreifen, muss ein `GET` auf die Server-Ressource `/.well-known/core` ausgeübt werden. Als Antwort bekommt man die Liste von Links zu den Ressourcen sowie ihre Attribute [11, S. 4f.]. Es gibt 4 Arten von Attributen: `ct` (Inhaltstyp), `if` (Interface), `rt` (Beschreibung) und `sz` (erwartete Maximalgröße) [11, S. 9f.]. RFC 6690 stellt das Beispiel Listing 1 vor.

```
REQ: GET /.well-known/core
RES: 2.05 Content
</sensors>;ct=40
```

```
REQ: GET /sensors
RES: 2.05 Content
</sensors/temp>;rt="temperature-c";if="sensor",
</sensors/light>;rt="light-lux";if="sensor"
```

Listing 1: Resource Discovery mittels Well-Known-Interface. Mit dem ersten `GET` wird `/.well-known/core` angesprochen, und als Antwort wird die Ressource `/sensors` als CoRE-Link (`ct=40` – vgl. Tab. 2) zurückgeliefert. Der zweite `GET` wird nun auf die entdeckte Ressource ausgeübt: Die Antwort beinhaltet CoRE-Links zu einem Temperatur- und zu einem Lichtsensor [11, S. 13].

Falls nach einer bestimmten Ressource gesucht wird, so kann man die Anfrage filtern (vgl. Lst. 2).

```
REQ: GET /.well-known/core?rt=light-lux
RES: 2.05 Content
</sensors/light>;rt="light-lux";if="sensor"
```

Listing 2: Die Anfrage wird durch URI-Query `rt=light-lux` parametrisiert, sodass ausschließlich CoRE-Links für Ressourcen mit der Beschreibung `light-lux` zurückgegeben werden [11, S. 13].

Ferner, kann eine Ressource durch eine andere (sogar externe) beschrieben werden. Dazu werden Attribute `anchor` (URI zur Ressource) und `rel` (Beschreibung der Relation zur Ressource) benutzt. Im Beispiel Lst. 3 wird die Ressource `/sensors/temp` zum einen durch eine externe Ressource (mit vollständigem URI) und zum anderen durch einen alternativen URI definiert.

```
REQ: GET /.well-known/core
RES: 2.05 Content
</sensors>;ct=40;title="Sensor Index",
</sensors/temp>;rt="temperature-c";if="sensor",
</sensors/light>;rt="light-lux";if="sensor",
<http://www.example.com/sensors/t123>
      ;anchor="/sensors/temp";rel="describedby",
</t>;anchor="/sensors/temp";rel="alternate"
```

Listing 3: Ressourcen `http://www.example.com/sensors/t123` und `/t` stehen in Beziehung zum Temperatursensor durch den Parameter `anchor="/sensors/temp"`. Die Arten von Beziehungen werden mit dem Parameter `rel` angegeben [11, S. 13].

4.3 Multicasting

Die Benutzung von UDP-over-IP macht es für CoAP möglich, die Nachrichten an eine bestimmte Gruppe von Knoten zu übertragen. Die IP-Multicast-Adressen sind im Bereich 224.0.0.0/4 (IPv4) und ff00::/8 (IPv6). Eine Gruppe von Knoten ist durch eine IP-Adresse aus diesen Bereichen definiert. Sendet man eine *non-confirmable* Nachricht an eine IPv4- bzw. IPv6-Multicast-Adresse, so wird diese von allen Teilnehmern dieser Gruppe empfangen.

Um eine Anfrage an Teilnehmer einer Gruppe zu verschicken, verwendet man eine Gruppen-URI. Diese ist entweder die IP-Adresse der Gruppe (ggf. mit Portnummer) oder der Hostname der Gruppe (ebenfalls optional mit Portnummer), der entweder sitelokal oder sogar global mittels DNS aufgelöst werden kann. Im Folgenden: Eine

doi: 10.2313/NET-2013-08-1_16

linklokale Multicast-Adresse ist nur innerhalb eines abgeschlossenen Netzwerksegments gültig (keine Weiterleitung durch Router); Sitelokale Adresse nur innerhalb des Organisationsnetzwerks (keine Weiterleitung durch Border-Router); Globale Adresse ist im gesamten Internet gültig (vgl. Abb. 9). Linklokale sowie sitelokale Geltungsbereiche sind Features von IPv6, in IPv4 sind nur Gruppen-URIs möglich. Eine nähere Beschreibung von Gruppenkommunikation mit CoAP und IP-Multicasting wird in [9] vorgestellt.

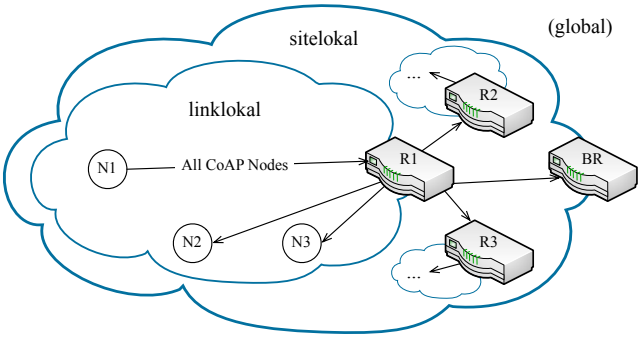

Abbildung 9: Über eine *linklokale* Multicast-Adresse erreicht N1 nur die Knoten N2 und N3. Über eine *sitelokale* Adresse wird die Nachricht von Router R1 zusätzlich an R2, R3 sowie an Border-Router BR weitergeleitet. Im Gegensatz zu R2 und R3 leitet BR jedoch *nicht* weiter.

Ferner, für CoAP werden zukünftig IPv4- sowie IPv6-„All CoAP Nodes"-Multicast-Adressen festgelegt (vgl. Abb. 9). Eine IPv4 wird aus dem Internetwork Control Block (224.0.1.x) kommen. Für IPv6-„All CoAP Nodes"-Multicast wird eine linklokale (ff02::nn) sowie eine sitelokale (ff05::nn) Adresse vergeben [12, S. 93]. Wird bspw. ein Knoten an ein Netzwerk angeschlossen, so kann dieser durch eine Anfrage an „All CoAP Nodes"-Adresse und Well-Known Interface über andere Knoten (link- oder sitelokal) erfahren [12, S. 61ff.]. Voraussetzung für Ressource Discovery ist die serverseitige Unterstützung von CoAP-Port 5683 [12, S. 62].
Um zu verhindern, dass eine große Anzahl an Antworten auf eine Multicast-Anfrage zur Überlastung des Netzes führt (z. B. Überfüllung von Receive Buffer eines Routers und somit Paketverlust), werden die Antwortnachrichten *verzögert* verschickt. Die Verzögerungszeit wird als „Leisure" bezeichnet. Untere Schranke von Leisure in Sekunden ($\inf L$) kann mit Formel 2 berechnet werden.

$$\inf L = \frac{S \cdot G}{R} \tag{2}$$

Eq. 2: Berechnung der unteren Schranke von Leisure ($\inf L$). S ist die geschätzte Antwortgröße [B], G die geschätzte Anzahl an Gruppenmitgliedern sowie R die Datentransferrate [B/s] (vgl. [12, S. 64]).

Im [12, S. 64] wird das „konservative" Vorgehen bei der Schätzung von relevanten Größen vorgeschlagen. Falls die Größen nicht geschätzt werden können, so wird der Standardwert von 5 Sekunden benutzt.

4.4 Sicherheitsmechanismen

Um eine abhörsichere Datenübertragung zu gewährleisten, kann für HTTP-over-TCP ein zusätzliches Verschlüsselungs-protokoll TLS (Transport Layer Security) benutzt werden. Eine verschlüsselte Übertragung ist dann an der Schemaangabe `https` erkennbar.
Da TLS auf einer zuverlässigen Übertragung auf der Transportschicht beruht, kann TLS ohne Weiteres nicht zusammen mit CoAP benutzt werden. Zum einen erfolgt die Integritätsüberprüfung von Daten sequenziell, d. h. wird eine bestimmte Nachricht nicht empfangen, so basiert die Integritätsüberprüfung der nächsten Nachricht auf der falschen Sequenznummer und wird demnach fehlschlagen. Zum anderen kann der TLS-Handschlag (Authentifizierung sowie Schlüsselaustausch) gar nicht stattfinden, falls die Nachrichten nicht-zuverlässig, also ungeordnet oder verlustbehaftet, ausgetauscht werden.
Um diese Probleme für nicht-zuverlässige Datenübertragung zu lösen, wurde das Protokoll Datagram Transport Layer Security (**DTLS**, aktuell 1.2) entwickelt. Das Ziel von DTLS ist es, die Funktionalität von TLS für Transportprotokolle UDP, DCCP, SCTP und SRTP anzubieten [10].
Im Gegensatz zu TLS, werden die DTLS-Nachrichten *explizit* nummeriert [10, S. 4f.]. TLS ist auf die richtige Reihenfolge von Nachrichten angewiesen, die nur durch TCP gesichert ist. Die explizite Nummerierung von Nachrichten erlaubt nun auch die Sicherung des Handschlags: DTLS überträgt Pakete *erneut* (Retransmission), falls nach einer bestimmten Zeit keine Antwort empfangen wurde (Timeout). Ferner, DTLS-Records können sowohl fragmentiert als auch dedupliziert werden [10, S. 14ff.].
CoAP-Internet-Draft [12, S. 65] definiert vier Sicherheitsmodi, in denen ein CoAP-Gerät betrieben wird, wobei NoSec und RawPublicKey obligatorisch zu implementieren sind:

NoSec: DTLS wird nicht benutzt. Alternativ kann CoAP mit IPsec benutzt werden [2].

PreSharedKey (PSK): Symmetrische Schlüssel werden im Vorhinein an die Knoten verteilt. Ein Schlüssel enthält u. a. eine Liste von Knoten, mit denen kommuniziert werden kann [13, S. 4]. Die nutzbaren Cipher-Suites werden in [6] aufgelistet, `TLS_PSK_WITH_AES_128_CCM_8` ist obligatorisch. [5] liefert eine detaillierte Beschreibung von PSK.

RawPublicKey (RPK): Es wird ein asymmetrisches Schlüsselpaar benutzt, jedoch ohne Zertifikat. Die öffentlichen Schlüssel eines Geräts werden bspw. in der Firmware „roh" (ohne X.509v3-Struktur) abgespeichert und können aktualisiert werden. `TLS_ ECDHE_ECDSA_WITH_AES_128_CCM_8` ist für RawPublicKey *obligatorisch*. Eigene Cipher-Suites für RawPublicKey werden jedoch nicht benötigt, da RPK mit allen Schlüsselaustausch-Suites kompatibel ist.

Certificate: Benutzung eines X.509v3-Zertifikats, das von einer Zertifizierungsstelle (CA) herausgegeben worden ist. `TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8` ist obligatorisch.

Ähnlich zu `https` wird in CoAP das `coaps`-URI-Schema definiert (vgl. Abb. 10).

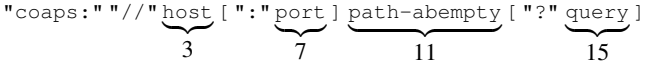

Abbildung 10: `coaps`-URI-Schema mit Optionsnummern einzelner Komponenten [12, S. 57f.].

Diese unterscheidet sich von `coap` nur in der Schemaangabe. `coap` und `coaps` sind als eigenständige Server zu betrachten selbst wenn die `host`-Angabe von URIs übereinstimmt [12, S. 58]. Der Standardport für `coaps` ist noch nicht definiert [12, S. 92f.].

doi: 10.2313/NET-2013-08-1_16

5. EINSATZGEBIET

Internet of Things beschäftigt sich mit der Integration von sog. Dingen in das gewöhnliche Internet. Die Dinge beschränken sich jedoch nicht auf Smartphones oder internetfähige Fernseher, ganz im Gegenteil, diese Dinge stehen für alle mögliche Gegenstände v. a. des alltäglichen Lebens, wie Waschmaschine oder Wasserkocher.
Um diese Dinge internetfähig zu machen, müssen diese zusätzlich mit Kommunikationsbausteinen ausgestattet werden. Ferner, möchte man die Zimmertemperatur in seiner Wohnung wissen, so muss diese mit einem *Sensor*, in diesem Fall mit einem digitalen Thermometer, ausgestattet werden. Das Ziel ist es, die Informationen, die über Sensoren erfasst werden, über ein Netzwerk sowohl für Menschen als auch für andere Geräte bereitzustellen. Das IoT ist somit die Weiterentwicklung von Sensornetzen (Wireless Sensor Network, WSN) übertragen auf unterschiedlichsten Gegenstände.
Das Open Source-Betriebssystem Contiki, das speziell für 8-Bit-Mikrocontroller entwickelt wird, vermarktet sich selbst als Betriebssystem für Internet of Things. Contiki hat volle Unterstützung von 6LoWPAN sowie CoAP [4]. Auf der Webseite werden Contiki-betriebene Vorzeigeprojekte vorgestellt, u. a. intelligentes Heizungssystem oder eine ferngesteuerte Glühbirne. Auf der Hardwareseite von Contiki ist das Modul AVR Raven des US-amerikanischen Herstellers Atmel aufgelistet, das mit Contiki ausgestattet ist (vgl. Abb. 11).

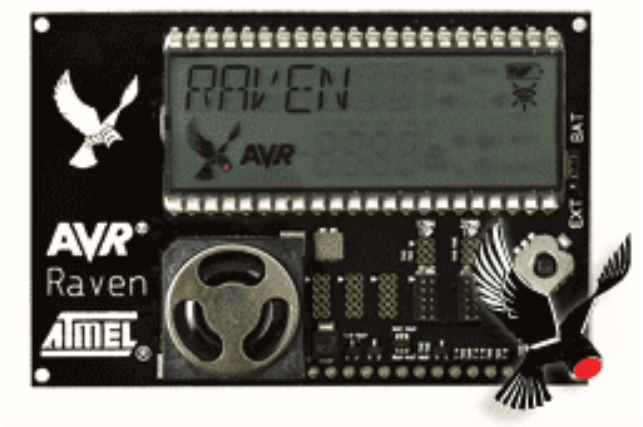

Abbildung 11: AVR Raven [1].

Das Modul hat einen 20 MHz 8-Bit-Mikrocontroller ATmega1284P mit (je nach Ausstattung) max. 16 kB RAM und 128 kB Flash Memory sowie einen 2,4 GHz-Transceiver mit IEEE 802.15.4-Unterstützung und einen Temperatursensor. Das Modul kann entweder batteriebetrieben oder an eine externe 5V–12V-Stromquelle angeschlossen werden. Der Betrieb kann auch mit 1,8V erfolgen [1].
Auf das Modul ist die Contiki-Firmware installiert, die die Unterstützung von CoAP implementiert. Alternativ kann das Modul mit HTTP und TCP betrieben werden.

6. ZUSAMMENFASSUNG

Diese Arbeit gab eine Übersicht über das neue Anwendungsprotokoll CoAP. CoAP bietet Vorteile einer REST-Architektur speziell für verlustbehaftete Funkübertragungsnetze mit geringem Stromverbrauch, die 6LoWPANs. Die IETF CoRE Working Group entwickelt das Protokoll um den Ressourcenzugriff ähnlich zu HTTP für kleine Geräte in M2M-Kommunikation zu ermöglichen. Solche Geräte werden ein Bestandteil von Internet of Things.
Zusammen mit Headerkomprimierung von 6LoWPAN, resultiert der Einsatz von CoAP in einem geringeren Overhead. In der Arbeit wurde beschrieben, wie sich der CoAP-Header inkl. Optionen zusammensetzt: Anstatt die Felder als Strings zu kodieren, werden diese binär, also möglichst sparsam, gespeichert.
Es wurde das zweischichtige Interaktionsmodell von CoAP vorgestellt: Auf der Nachrichtenschicht wird Übertragungs-zuverlässigkeit gesichert, auf der Anfrage-Antwort-Schicht die Funktionalität der REST-Architektur implementiert.
Die Arbeit stellte weitere Funktionen von CoAP vor, die für M2M wichtig sind: Ressource Discovery und IP-Multicasting. Falls man mittels HTTP auf CoAP-Ressourcen zugreifen möchte, ist HTTP-Proxying möglich. CoAP-Nachrichten können mit Verschlüsselungsprotokoll DTLS abhörsicher gemacht werden.
Der schlanke Aufbau im Bezug auf Headerfelder einer CoAP-Nachricht wirkt überzeugend. Aus der Funktionalität von CoAP ist M2M als Hauptrichtlinie klar ersichtlich. Die wirklichen Vor- und Nachteile im Bezug auf andere Anwendungsprotokolle wie HTTP werden erst in der Zukunft mit einer weiteren Verbreitung und Implementierung von CoAP deutlich.

7. LITERATURVERZEICHNIS

[1] ATMEL: *AVR Raven*. http://www.atmel.com/tools/AVRRAVEN.aspx, Abruf: 23.06.2013

[2] BORMANN, C. : *Using CoAP with IPsec*. Internet-Draft (Informational). http://tools.ietf.org/id/draft-bormann-core-ipsec-for-coap-00.txt. Version: Dez. 2013 (9). – Ungültig ab 9. Juni 2013

[3] CASTELLANI, A. ; LORETO, S. ; RAHMAN, A. ; FOSSATI, T. ; DIJK, E. : *Best Practices for HTTP-CoAP Mapping Implementation*. Internet-Draft (Informational). http://tools.ietf.org/id/draft-castellani-core-http-mapping-07.txt. Version: Febr. 2013 (7). – Ungültig ab 29. August 2013

[4] CONTIKI PROJECT: *Contiki: The Open Source Operating System for the Internet of Things*. http://www.contiki-os.org/, Abruf: 23.06.2013

[5] ERONEN, P. ; TSCHOFENIG, H. : *Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)*. RFC 4279 (Proposed Standard). http://www.ietf.org/rfc/rfc4279.txt. Version: Dez. 2005 (Request for Comments)

[6] IANA: *TLS Cipher Suite Registry*. http://www.iana.org/assignments/tls-parameters/tls-parameters.xml#tls-parameters-4. Version: Mai 2013, Abruf: 23.06.2013

[7] IEEE: IEEE Standard for Local and metropolitan area networks–Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs). In: *IEEE Std 802.15.4-2011 (Revision of IEEE Std 802.15.4-2006)* (2011), S. 1–314. http://dx.doi.org/10.1109/IEEESTD.2011.6012487. – DOI 10.1109/IEEESTD.2011.6012487

[8] MONTENEGRO, G. ; KUSHALNAGAR, N. ; HUI, J. ; CULLER, D. : *Transmission of IPv6 Packets over IEEE 802.15.4 Networks*. RFC 4944 (Proposed Standard). http://www.ietf.org/rfc/rfc4944.txt. Version: Sept. 2007 (Request for Comments). – Aktualisiert von RFCs 6282, 6775

[9] RAHMAN, A. ; DIJK, E. O.: *Group Communication for CoAP*. Internet-Draft (Informational). http://www.ietf.org/id/draft-ietf-core-groupcomm-09.txt. Version: Mai 2013 (9). – Ungültig ab 30. November 2013

[10] RESCORLA, E. ; MODADUGU, N. : *Datagram Transport Layer Security Version 1.2*. RFC 6347 (Proposed Standard). http://www.ietf.org/rfc/rfc6347.txt. Version: Jan. 2012 (Request for Comments)

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

[12] SHELBY, Z. ; HARTKE, K. ; BORMANN, C. : *Constrained Application Protocol (CoAP)*. Internet-Draft (Standards Track). http://tools.ietf.org/id/draft-ietf-core-coap-17.txt. Version: Mai 2013 (17). – Ungültig ab 27. November 2013

[13] WOUTERS, P. ; GILMORE, J. ; KIVINEN, T. ; TSCHOFENIG, H. ; WEILER, S. : *Out-of-Band Public Key Validation for Transport Layer Security (TLS)*. Internet Draft (Standards Track). http://www.ietf.org/id/draft-ietf-tls-oob-pubkey-07.txt. Version: Febr. 2011 (7). – Ungültig ab 19. August 2013