# Constrained Application Protocol, ein Multicast-fähiger TCP Ersatz?

Bernhard Schneider
Betreuer: Christoph Söllner
Hauptseminar - Sensorknoten - Betrieb, Netze und Anwendungen SS 2013
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: schneidb@in.tum.de

## ABSTRACT

In diesem Paper wird ein Vergleich von den beiden Protokollen Transfer Control Protocol (TCP) und Constrained Application Protocol (CoAP) angestrebt. Zunächst zeigt sich in der Einleitung, dass ein Vergleich durch ähnliche Leistungen der beiden Protokolle motiviert ist, obwohl sie sich in unterschiedlichen Schichten befinden. Anschließend daran wird das Übertragungsverfahren für Daten der beiden Protokolle, sowie deren Eigenschaften zur Garantierung von Zuverlässigkeit näher betrachtet. Nach einer Erläuterung der Multicast-Möglichkeiten von CoAP folgt schließlich ein Vergleich von TCP- und CoAP Implementierungen mit Beschreibung der Vergleichsmethodik, sowie eine Auswertung der Resultate.

## Keywords

CoAP; TCP; IP Multicast;

## 1. EINLEITUNG

Das Transmission Control Protocol (TCP) ist heute weit verbreitet in Einsatz zur zuverlässigen übermittlung von Daten über das Internet. Es erlaubt den Austausch von Daten von einem Sender zu genau einem Empfänger (Unicast). [1]

In einigen Netzwerken, wie zum Beispiel Sensor Netzwerken werden jedoch auch zuverlässige Multicast-Nachrichten benötigt, um mehrere Empfänger gleichzeitig mit einer Nachricht zu erreichen. Ein Beispielszenario bilden zum Beispiel Beleuchtungseinrichtungen eines Hauses mit intelligentem Stromnetz (Smart Grid). So kann ein Hausbesitzer durch einen Knopfdruck mehrere Beleuchtungen gleichzeitig ein- oder ausschalten. Hierfür kommt das Constrained Application Protocol in Frage, welches durch die Nutzung von UDP als Transportprotokoll sich von der einfachen Bindung lösen kann und dennoch einige zusätzliche Eigenschaften von TCP ersetzen kann, die von UDP nicht gewährleistet werden. Ob CoAP wirklich als ein Ersatz für TCP genutzt werden kann, soll im folgenden durch einen Vergleich der beiden Protokolle untersucht werden.

## 2. MESSAGING VON COAP UND TCP

Deutliche Unterschiede finden sich bei CoAP und TCP bereits im Header der Nachrichten. Grund hierfür ist die unterschiedliche Ausrichtung der Protokolle.Es gilt zu beachten, dass es sich bei TCP um ein Transportprotokoll handelt, wohingegen CoAP der Applikationsebene im OSI-Schichtenmodell zuzuordnen ist, welche sich nicht mit der Übertragung einer Nachricht befasst.

### 2.1 Headervergleich

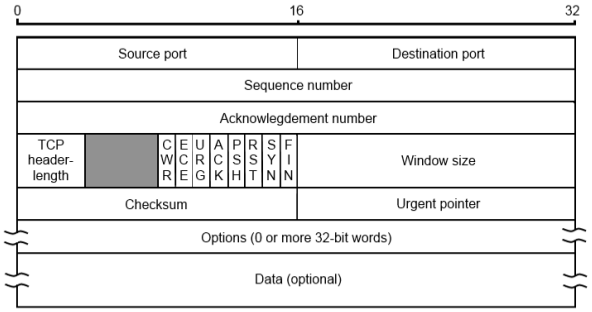


**Figure 1: TCP-Header [1]**

TCP beinhaltet im Header, wie in Abbildung 1 ersichtlich, bereits den Quell- und Ziel-Port der Nachricht. Diese sind bei CoAP selbst nicht, sondern im Header des übertragenden Protokolls, in der Regel UDP, vorhanden[1].

#### *2.1.1 Nachrichtenzuordnung*

Der TCP Header enthält ein 32-Bit „Sequence number“-Feld und ein ebenso großes Feld für die „Acknowledgement number“ . Hierbei handelt es sich um Felder die im ersten Feld die erste Bitnummer der aktuell übertragenen Sequenz enthält. Im zweiten Feld wird bei Bestätigungen ebenfalls eine Zahl übertragen, wobei es sich dann um die erste Zahl der zuletzt beim Empfänger korrekt eingegangen Nachricht handelt.

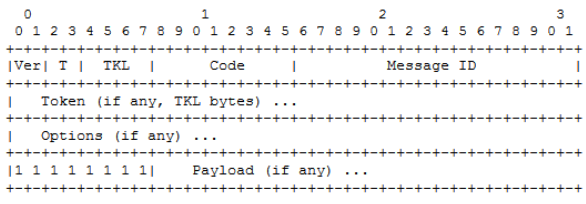


**Figure 2: CoAP-Header [2]**

Vergleichbar zu der„Sequence number“ und der „Acknowledgement number“ sind bei CoAP gemäß Abbildung 2 im Header die Felder *Token* und *Message-ID* vorhanden. Letzte-


doi: 10.2313/NET-2013-08-1_18

res wird dazu verwendet, um Nachrichtbestätigungen oder -ablehnungen den ursprünglich versandten Nachrichten beim Sender zuzuordnen. Die Aufgabe der „Sequence number“ ist mit der des Feldes *Token* vergleichbar. Der Feldinhalt beinhaltet einen Zahlenwert , der dazu genutzt wird, Antworten auf Anfragen des Senders zuordnen zu können [2].

### 2.1.2 Headerfeldunterschiede

Ferner besitzen beide Protokolle noch weitere Felder, für die im anderen Header keine vergleichbaren Felder zu finden sind. So existiert bei TCP zum einen ein Feld von 4-Bit für die Angabe der Headergröße. Zum anderen folgt auf dieses ein weiterer Bereich mit 4 Bits, welches bis heute ungenutzt bei jeder Nachricht mitversendet wird[1]. Zudem besitzt TCP im Header ein Feld zur Bestimmung der sogenannten Window-Size. Damit wird festgelegt, wie große Datenpakete vom Sender aktuell übertragen werden dürfen, ohne dass des Buffer des Empfängers überläuft. [2] Übermittelt wird diese beim Acknowlegdement einer bereits empfangenen Nachricht. Sofern genügend Platz im Buffer ist, kann der Sender mehrere Segmente als Burst an den Empfänger schicken. Es dient zur Staukontrolle, damit Nachrichten nicht trotz fehlenden Speicherplatzes übermittelt werden. Weitere Felder sind der „Checksum“ - und „Urgent pointer“ -Bereich. Die Checksum besitzt eine Länge von 16 Bit und beinhaltet eine Prüfsumme für den Header und den übertragenden Daten. Sofern diese beim Empfänger von der für das Paket berechnete Summe abweicht, weiß dieser, dass das Paket unvollständig beziehungsweise fehlerhaft ist.[1,Seite 231f.] CoAP selbst besitzt diese nicht, allerdings existiert UDP-seitig ein Prüfsummen-Header-Feld, welches dem von TCP gleichzusetzen ist.[1] Genauso groß ist das „Urgent Pointer“ -Feld, welches in einem TCP-Segment die letzte Bitnummer im Bit-Offset der Nachricht angibt, in welchem sich dringliche Daten befinden. Die wichtigen Daten sind daher vom ersten bit bis zum Urgent Pointer hinterlegt. Um auf dringeliche Daten hinzuweisen, muss das Urgent-Bit (URG) auf 1 gesetzt werden. Erst dann wird der Inhalt des Urgent-Pointer-Feldes beachtet .

Bei CoAP finden sich außerdem, wie in Abbildung 2 ersichtlich, die Felder *Version (Ver)*, *Type (T)*, *TokenLength (TKL)*, *Code*, sowie ein Feld *Token*, dessen Länge durch den Wert des Feldes TKL spezifiziert wird.

Das Feld *Version* gibt die genutzte Version von CoAP an. Es ist derzeit standardmäßig mit „01“ besetzt [2]. Im Codefeld wird der Inhalt der Nachricht näher spezifiziert. So kann es sich bei einer Nachricht um eine Anfrage oder eine Antwort handeln. Ein Beispielszenario für den Gebrauch dieses Feldes wäre die Abfrage der aktuellen Wertes eines Sensors mit Anbindung an ein Sensor-Netzwerk.

### 2.1.3 Nachrichtentypen

Im Header wird zudem sowohl bei CoAP als auch TCP der Nachrichtentyp der übermittelten Nachricht festgelegt. Mit Hinblick auf den Typ können Sender und Empfänger einander mitteilen, wie mit einer Nachricht genau umgegangen werden soll.

Bie CoAP gibt der 2-Bit große *Type*-Feldinhalt Aufschluss über den Typ der Nachricht. Es stehen genau vier Typen zur Verfügung. Dabei stehen die Zahlen 0-3 (binär) codiert für jeweils genau einen Nachrichtentyp [2]:

- Confirmable (0): Nachricht muss bei Erreichen des Empfängers von diesem bestätigt werden, kann entweder eine Anfrage (Request) oder Antwort (Response) sein.
- Non-Confirmable (1): Nachricht muss nicht bei Empfang durch Empfänger bestätigt werden, kann entweder eine Anfrage (Request) oder Antwort (Response) sein.
- Acknowledgement (2): Nachricht zum Bestätigen des Empfangs einer Confirmable-Nachricht, die Message ID der Confirmable-Nachricht beinhaltend. Entweder leer, also keine Daten oder Response-Daten enthalten können (piggy backed).
- Reset (3): Nachricht kann bei Empfang einer anderen Nachricht gesendet werden, beispielsweise um eine Confirmable Nachricht dem Sender als „abgewiesen“ mitzuteilen.

Der Typ der Nachricht bei TCP wird durch Flags im TCP-Header festgelegt. Die in Abbilung 1 in der vierten Reihe sichtbaren acht Flags belegen mit 8-Bit genau viermal so viel Platz wie die Typisierung in CoAP im Feld T. Im speziellen sind hier neben fünf anderen die Flags SYN und ACK sowie FIN zu nennen. Sie werden dazu benutzt, um mit Hilfe eines 3-Wege Handschläge eine Verbindung zu genau einem Empfänger aufzubauen und wieder zu trennen, welches in Kapitel 3 näher erläutert wird.

### 2.1.4 Payload

Unterschiede ergeben sich in der Größenwahl des Payloads der beiden Protokolle. Im Falle von TCP gibt es keine Größenbeschränkungen des Payloads. Es müssen aber bei der Bildung von TCP Segmenten die maximale Größe von IP-Paketen (65,535 Bytes), sowie beim Versenden die MTU (Maximum Transfer Unit) des Senders und Empfängers berücksichtigt werden, damit der Inhalt unfragmentiert übermittelt werden kann. In Ethernet Netzwerken ergibt sich damit ein maximaler Payload von 1460 Bytes nach Abzug der Headergrößen[1].

Bei CoAP hingegen hängt die Größe von dem gewählten Transportprotokoll ab. Im Falle des von der IETF empfohlenen UDP Protokolls errechnet sich die maximale Payload-Größe ebenfalls aus der Nachricht-Gesamtgröße und der zu subtrahierenden Größe des Headers. Bei einer unbekannten MTU wird allerdings eine maximale Größe der Nachricht von 1152 mit einem Payload von 1024 Bytes empfohlen. Hierbei zu beachten ist allerdings der Payload-Marker, welcher den Header vom Payload trennt. Abhängig von der Länge der Optionen im Header kann dessen Position und somit die Länge des Payloads variieren [2].

# 3. ZUVERLÄSSIGKEIT

## 3.1 Synchronisierte Verbindung

Das Transport Control Protocol legt, wie der Name vermuten lässt, vor allem Wert auf Kontrolle und Zuverlässigkeit. So stellt TCP beispielweise, wie im vorigen Kapitel erwähnt,


doi: 10.2313/NET-2013-08-1_18

Verbindungen her und sendet Daten an genau einen Empfänger [1]. Dazu wird im Gegensatz zu CoAP via UDP wird bei TCP mittels eines 3-Wege Handschläge eine direkte Verbindung zu einem Empfänger aufgebaut. Wie in Abbildung

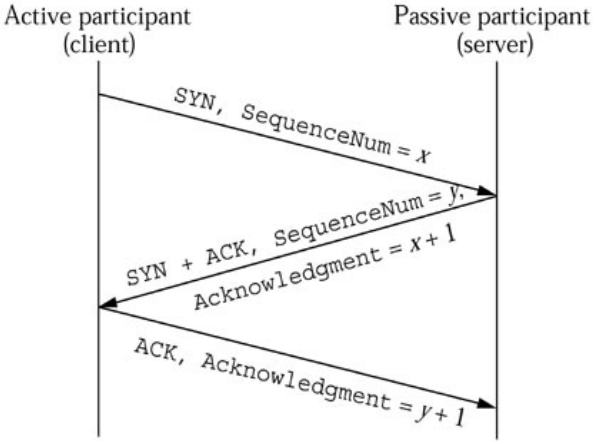


**Figure 3: 3-Wege-Handschlag**

3 ersichtlich wird, versendet dabei der Sender, hier der *client*, zunächst eine Nachricht mit gesetztem SYN-Bit an den Empfänger, in diesem Fall ein *server*, als Anfrage für eine neue Verbindung. Dieser kann mit einer weiteren SYN Nachricht mit ACK-Bit die Verbindunganfrage bestätigen. Der dritte Handschlag besteht wiederum aus einer Bestätigung dieser Nachricht seitens des Senders an den Empfänger. Nach Abschluss des 3-Wege-Handschläge besteht eine synchronisierte Verbindung, über die Daten in beiden Richtungen ausgetauscht werden können.

CoAP verzichtet beim Nachrichtenaustausch auf den Aufbau einer synchronisierten Verbindung. Das hat zur Folge, dass einerseits zwar eine geringere Menge an Daten benötigt wird um Nachrichten zu übermitteln. Andererseits besteht dadurch ein höheres Risiko von nicht erkannten Übertragungsverlusten beim Nachrichtenaustausch.

## 3.2 Zustandsbehaftete Verbindung

Während des Datentransfers durchlaufen bei TCP die Verbindungsteilnehmer einen Zustandsautomaten mit 11 Zuständen. In Abbildung 4 sieht man, dass beim 3-Wege-Handschlag sich der Zustand der Teilnehmer von CLOSED über LISTEN und SYN/RCVD bzw. SYN SEND auf ESTABLISHED ändert. Ein Status kann durch Nachrichten mit entsprechend gesetzten Flags geändert werden. Der Zustandsautomat zeigt, dass eine Verbindung bestimmten Regeln unterliegt. So wird festgelegt welche Aktionen ein Verbindungsteilnehmer in welchem Zustand durchführen kann oder muss. Sollten diese verletzt werden, wird eine Fehlermeldung generiert. Es weiß beispielsweise ein Empfänger damit zu jederzeit, ob er weiterhin auf Daten warten muss oder die Übertragung abgeschlossen ist. In letzterem Falle muss in der Regel eine Nachricht mit FIN-Bit an die jeweilige andere Partei gesendet worden sein, die von dieser bestätigt werden muss [2, p. 580f.].

## 3.3 Retransmission

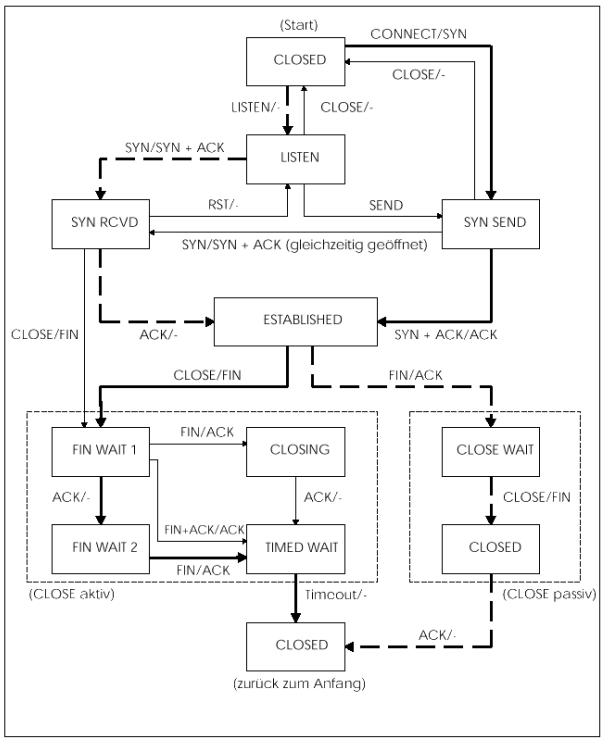


**Figure 4: Zustandsautomat TCP[3]**

Um den offensichtlichen Verlust einer Nachricht auszugleichen, wird diese sowohl mit CoAP als auch bei TCP erneut versendet. Dazu muss allerdings der Verlust einer Nachricht als sicher gelten. In der Regel lässt sich dies anhand einer Zeitspanne abschätzen, die eine Nachricht und deren Bestätigung zur Übertragung benötigen.

Verantwortlich für die Länge des Wartezeitraums ist im Falle von TCP ein Timeout Timer namens *Retransmission TimeOut* (RTO), der beim Nachrichtversand gestartet wird und bei dessen Ablauf eine erneute Sendung der aktuellen Sequenz erfolgt, sofern bis dahin keine Bestätigung vom Empfänger eingegangen ist. Berechnet wird die Länge des Timers über die RoundTripTime, also der streckenbedingten Übertragungszeit vom Versand einer Nachricht bis hin zum Eingang einer Bestätigung unter Berücksichtigung gewissen Varianz für den Fall dass z.B. der Empfänger nicht unmittelbar bei Empfang antworten kann, die *Smoothed RoundTriprime* (SRTT) genannt wird. Hierzu wird folgende Formel verwendet [1,s.588]:

$$SRTT = \alpha * SRTT + (1 - \alpha) * R$$

Dabei steht $R$ für eine gemessene RTT, die kontinuierlich aktualisiert wird. Die Konstante $\alpha$ stellt einen Abweichungsfaktor dar, der Unregelmäßigkeiten ausgleichen soll. Er ist im Normalfall mit 7/8 belegt.

Ein weiterer Timer ist der sogenannte „persistence timer“ . Er garantiert, dass im Falle eines Wartezustands des Sender, beispielsweise aufgrund eine Window-Size von 0 seitens des Empfängers, eine verlorene Bestätigung mit einer Vergrößerung der Window-Size nicht zum Deadlock führt. Stattdessen wird bei Ablauf des Timers eines ein-Byte TCP-Segment


doi: 10.2313/NET-2013-08-1_18

als Probe-Nachricht an den Empfänger gesendet, der daraufhin seine Window-Size-Änderung erneut bestätigen kann.

Der „keepalive timer“ gewährleistet als ein dritter Timer eine Erkennung einer vollständig abgebrochenen Verbindung, zum Beispiel durch Kabelriss. Wenn eine Verbindung längere Zeit inaktiv ist besitzt er eine Zeitspanne, die lang genug ist, dass davon ausgegangen werden kann dass alle Pakete verloren gegangen sind. Problematisch ist allerdings in diesem Zusammenhang die mögliche Beendigung einer funktionieren Verbindung aufgrund eines zu langen Verbindungswegs[1].

Im Gegensatz dazu sind bei CoAP weniger Zuverlässigkeitsfunktionen vorgesehen. Das Protokoll beschränkt auf folgendes, einfaches Retransmission System. Wie TCP bestzt auch CoAP einen Timer, der nach Ablauf eine erneute Sendung einer Nachricht auslöst und erneut gestartet wird. Um einem finalen Abbruch einer Verbindung entgegen zu wirken existiert ein Parameter *(MAX_RETRANSMIT)*, der durch Angabe der maximalen Anzahl an Neusendungen beim Verbindungsabbruch weitere Retransmissions verhindert. Er ist auch entscheidend für die Zeit bis zum Timeout, da er neben dem ACK_RANDOM_FACTOR in die Berechnung für den Maximalen Erwartungszeitraum einer Bestätigung sowie der Zeitspanne vom ersten bis zum letzten Retransmit miteingeht. Als Standard ist ein Wert von 4 Retransmissions (nach IETF) vorgesehen[2].

Im Gegensatz zu CoAP bietet TCP eine Option, um den Datenbestätigungsverkehr zu verringern. Es handelt sich hierbei um sogenannte SACK-Nachrichten (Selective Acknowledgements) [1]. Wird zum Beispiel bei der Datenübertragung ein Paket verloren, so erreicht das eigentlich folgende Paket zuerst den Empfänger. Es wird daher nur für das letzte in richtiger Reihenfolge empfangene Paket ein Acknowledegement mit der entsprechenden Bitnummer versendet. Im Option-Feld wird allerdings die Bitsequenz des empfangenen Folgepakets mit angegeben. Es werden so bis drei zu SACK-Bitsequenzen angegeben, so dass der Sender besser entscheiden kann, welche Pakete neu übertragen werden müssen und welche bereits übertragen worden sind [1, 598f.].

## 3.4 Zyklische Redundanzprüfung

Beide Protokolle besitzen außerdem eine sogenannte *Zyklische Redundanzprüfung* (CRC). Dahinter verbirgt sich ein Prüfcode, der Übertragungsfehler erkennbar macht. Hierzu wird die zu übertragende Nachricht binärcodiert als Polynom betrachtet. Zusätzlich verwenden Sender und Empfänger ein gemeinsames Generatorpolynom. Das Nachrichtenpolynom wird durch das Generatorpolynom geteilt. Der Rest der Division wird an die Nachricht angehängt. Beim Empfänger wird ebenfalls eine Polynomdivision mit der Nachricht samt Rest und dem Generatorpolynom durchgeführt. Sollte hierbei ein Rest bleiben, wird die übertragene Nachricht als fehlerhaft angesehen.

## 3.5 Staukontrolle

Damit keine Daten versendet werden, ohne dass der Empfänger sie aufnehmen kann, existiert bei TCP das Konzept namens „Sliding WindowâĂİ. Dabei wird bei Nachrichtensequenzen die Nummer des ersten Bits (Sequence number SEQ) im Kontext der übermittelten Daten angegeben, die in der Bestätigung (ACK) mit der Nummer des zuletzt empfangenen Bits vom Empfänger quittiert werden muss, Hierbei gibt er im Window-Size-Feld an, in welchem Umfang er derzeit Daten empfangen kann, ohne dass sein Nachrichtenbuffer überläuft. Wird dieses mit 0 belegt, so ist der Sender gezwungen zu warten, bis eine erneute Bestätigung gesendet wird, deren Window-Size einen Wert größer als 0 besitzt.

# 4. UNTERGEORDNETE (TRANSPORT-) SCHICHTEN

CoAP wird meistens mit Hilfe von UDP übertragen welches durch seine Verbindungslosigkeit somit Multicasts möglich macht. TCP hingegen ist nur in der Lage, Nachrichten an einen bestimmten Empfänger zu schicken.

## 4.1 Transport-Schichten Sicherheit

Bei TCP kann zur sicheren Übertragung von Daten die *Transport Layer Security* eingesetzt werden, welche dazu dient die Datenintegrität zu schützen. Im OSI-Schichtenmodell ist diese zwischen der Applikations-Schicht und der Transport-Schicht zu finden. Bei CoAP hingegen muss bei Verwendung von UDP als unzuverlässiges Transportprotokoll die modifizierte Form von TLS, die *Datagram Transport Layer Security* (DTLS) eingesetzt werden, da anders als bei TCP keine Verbindung aufgebaut wird und so Nachrichten beispielsweise nicht in richtiger Reihenfolge eintreffen müssen. Dieser Fall ist bei TSL nicht vorgesehen.

## 4.2 Alternativen zu TCP und UDP

Ein wesentlicher Vorteil gegenüber TCP ist bei CoAP die Tatsache, dass dieses nicht fest an das *Internet Protokoll*(IP) in der Netzwerkschicht gebunden ist. Laut IETF ist CoAP neben UDP auch über *Short Message Service*(SMS) und *Unstructured Supplementary Service Data* (USSD) nutzbar. Diese funktionieren auch, wenn keine Internetverbindung verfügbar oder abgeschaltet ist, sondern lediglich eine abgeschlossene Netzwerkumgebung vorhanden ist.

## 4.3 Multicast von CoAP

Im Gegensatz zu TCP, besitzt CoAP die Fähigkeit, Multicasts zu versenden [2]. Dieses dient vor allem dazu, eine Anfrage an mehrere Empfänger gleichzeitig zu senden. Möglich ist dies durch sogenannte IP-Multicast-Adressen.

### 4.3.1 IPv6 Multicast-Adresse

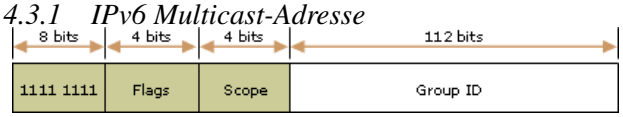


**Figure 5: IPv6 Multicastadresse[5]**

Das Grundprinzip hinter einem IP-Adressen-Multicast sieht vor, dass eine IP-Adresse eine ganze Gruppe von Empfängern anspricht, in welcher die Nachricht alle erhalten, unabhängig ob sie mit dieser etwas anfangen können oder nicht. Im Aufbau besteht eine IPv6-Mutlicast-Adresse aus einem festen Anteil der oberen acht Bits mit dem Wert 0xFF. Es Folgen vier Bits die mit Flags belegt sind, sowie vier weiteren Scope-Bits. Die unteren 112 Bits bilden die Gruppen-ID der jeweils adressierten Multicast-Gruppe. Dabei gilt dass die Gruppen-ID im Bereich (Scope) einzigartig sein muss


doi: 10.2313/NET-2013-08-1_18

Der Scope gibt an in welchem Raum der Mutlicast stattfindet.

- Wert 1: Node-local
- Wert 2: Link-local
- Wert 5: Site-local
- Wert 8: Organization-local
- Wert E: Global

### 4.3.2 CoAP Gruppenkommunikation

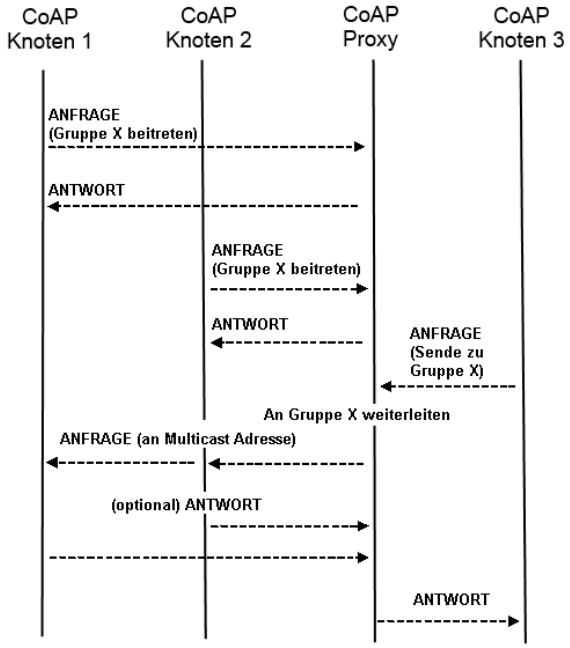


**Figure 6: CoAP Gruppenkommunikation [6]**

Als Anwendungsszenario dient die Gruppenkommunikation von CoAP. Diese sieht vor, dass in einer CoAP-Gruppe mehrere Knoten und ein Proxy-Knoten vorhanden sind. Um der Gruppe beizutreten, senden die Knoten eine Anfrage an den Proxy Knoten, welche von diesem beantwortet wird. Sofern nun eine Multicast-Anfrage an den CoAP-Proxy gelangt, wird der Request an alle Mutlicast-Gruppenteilnehmer weitergeleitet und deren Antworten gesammelt [6].

# 5. IMPLEMENTIERUNGSVERGLEICH VON TCP UND COAP

Im Folgenden sollen nun zwei konkrete Implementierungen von CoAP und TCP verglichen werden. Hierbei stehen vor allem die Code-Größe und die Performance im Vordergrund. Um den Schichtenunterschied der Protokolle ausgleichen zu können wird dazu das Applikationsprotokoll *HTTP* in den Vergleich mit TCP kombiniert mit einbezogen. Des Weiteren ist auch die Paketgröße bei gleichem Payload von Relevanz.

**Table 1: Größenvergleich von TCP/HTTP und UDP/CoAP[7]**

| | CoAP [Bytes] | HTTP [Bytes] |
|---|---|---|
| Header | 10 | 20 |
| *Options* | 51 | – |
| Payload | 163 | 163 |
| *GesamtesPaket* | 266 | 324 |

**Table 2: Zeitenvergleich von TCP/HTTP und UDP/CoAP[7]**

| | CoAP | HTTP |
|---|---|---|
| Durschschnittliche Übertragungszeit in Sek. | 3.8671 | 9.9248 |

## 5.1 Größenvergleich und Performance zweier Nachrichten

Die„Faculty of Organisational Science“ (Belgrad/Serbien) hat im Rahmen einer Evaluation von CoAP also Application Protocol in M2M Systemem (Machine-to-Machine) bereits einen Vergleich von CoAP Nachrichten mit UDP und HTTP (Hypertext Transfer Protocol) in der Applikationsebene mit TCP als Transportprotokoll vorgenommen und konnte bereits erste Aussagen über die Paketgröße treffen [6]. In der Evaluation wurden 2 Pakete mit jeweils einem Payload von 163 Bytes an einen Server versendet. Die Größe der Pakete konnte hierbei mit Hilfe des Netzwerkprotokollanalyseprogramms „Wireshark“ dokumentiert werden. Nach ihren Messungen ergeben sich für die in Tabelle 1 dargestellten Pakete Größen.

Es lässt sich leicht berechnen, dass das UDP-Paket nach Abzug der CoAP-spezifischen Daten von 224 Byte weitere 42 Byte an Headerdaten benötigt, während TCP nach Abzug von HTTP-Daten mit 141 Byte einen deutlich größeren Overhead besitzt.

Bei der Zeitmessung mussten bei CoAP und TCP grundlegende Unterschiede gemacht werden. Während bei CoAP eine einfache Übertragungszeit gemessen wurde, musste bei TCP berücksichtigt werden, dass erst eine Verbindung hergestellt werden musste, die Nachricht versendet und schließlich wieder getrennt werden musste. Hierzu wurden bezüglich TCP verschiedene Messungen bezüglich Verbindungsaufbau und Verbindungsdauer durchgeführt und davon die Mittelwerte gebildet und addiert. Für genauere Bestimmung eines Ergebnisses wurden die Messungen jeweils 50fach ausgeführt. Als Ergebnis werden folgende Zeiten genannt.

Wie deutlich aus Tabelle 2 hervorgeht, ist der benötigte Zeitaufwand zur Übertragung einer Nachricht mittels TCP mehr als 2,5 mal so groß wie CoAP. Schuld hieran ist vor allem das sehr zeit-und nachrichtenlastige Auf- und Abbauen einer Verbindung durch den 3-Wege Handschlag bzw. 2-Wege Handschlag beim Abbau einer TCP-Verbindung.

## 5.2 Codegrößenvergleich


doi: 10.2313/NET-2013-08-1_18

**Table 3: Lines of Code von Libcoap und uIP-Stack [Eigenanfertigung]**

| Library | libcoap 1.4.0 | uIP-Stack 1.0 |
|---|---|---|
| LoC-in C-Files | 3088 | 2101 |
| LoC in C-Header | 2147 | 695 |
| **Summe** | **5235** | **2796** |

Um ein besseres Bild werden zwei konkrete Implementierungen bezüglich ihre Codegröße verglichen. Um die Codegröße möglichst genau zu bestimmen wurde das Freeware-Tool „CLOC“ verwendet, welches es ermöglich die genaue Anzahl an „Lines of Code“ (LoC) zu ermitteln [8]. Zum Vergleich dient hier zum einen die Implementierung in C von CoAP namens „libcoap“ welche von der IETF working group „CoRE“ entwickelt und als Library im Internet frei verfügbar ist. Zum Anderen wurde auf seiten von TCP die Implementierung des uIP-Stack, ebenfalls in C programmiert, herangezogen, welche von Adam Dunkels ins Leben gerufen wurde und inzwischen als Teil von Contiki (Open Source Betriebssystem für das Internet Of Things) verwendet wird [10,11]. In beiden Fällen wird die reine Library und keine Beispiele von konkreten Client-Server Implementierungen betrachtet. Die Analyse mit CLOC ergab folgende Code-Größen

Wie an Tabelle 3 deutlich zu erkennen ist, ist die CoAP Implementierung fast doppelt so groß wie die von TCP.

# 6. AUSWERTUNG UND EINSCHÄTZUNG

Nach eingehender Analyse lässt sich diese nun wie folgt auswerten. CoAP ist vor allem auf Geschwindigkeit und minimalen Datentransfer ausgelegt und bietet zudem die Möglichkeit, mehrere Empfänger gleichzeitig via Multicast zu erreichen, was die Übertragung von Daten allgemein sehr effizient macht. Vor allem bei Netzwerken, die wie im Falle von 6LoWPAN-Netzwerken [12] sehr geringe Energie- und Speicherressourcen zur Verfügung haben und somit der Datenverkehr möglichst klein gehalten werden muss erscheint CoAP sehr gut geeignet, im speziellen Fall geeigneter als Verbindungen über TCP. Dieses hingegen zeigt seine Stärken deutlich im Bezug auf Zuverlässigkeit. So sind bei TCP durch die eben erläuterten Fähigkeiten zur Zuverlässigkeit und Staukontrolle deutlich ausgeprägter als bei CoAP vorhanden, was vor allem bei größeren Datenmengen von Relevanz ist. Somit bleibt festzuhalten, dass CoAP in energiesparsamen Netzwerken eher genutzt werden kann als TCP, weil dort das Energiermanagement wichtiger ist als die Zuverlässigkeit. In anderen Netzwerken hingegen sollte weiterhin auf TCP vertraut werden, dass von der Sparsamkeit abgesehen, deutlich zuverlässiger ist und mit weniger Aufwand zu implementieren ist als CoAP.

# 7. ZUSAMMENFASSUNG

In diesem Paper wurde zunächst das Nachrichtenwesen von CoAP und TCP verglichen. Bereits beim Headervergleich wurden deutliche Unterschiede im Bezug auf die Felderanzahl und Felderart deutlich. Während TCP viele Felder zur Staukontrolle und Sicherung der Zuverlässigkeit beinhaltet, wird bei CoAP sich auf ein Minimum an Overhead beschränkt. Im weiteren wurde erkennbar, dass TCP dank zuverlässiger Verbindungen weniger unerkannte Datenverlustmögleichkeiten besitzt als CoAP. Dennoch ist CoAP deutlich flexibler, was den Datentransfer angeht, weil es als Anwendungsschicht-Protokoll nicht auf eine einzige Verbindungsart wie TCP angewiesen ist, sondern auf verschiedenen anderen Systemen, wie SMS und USSD ebenfalls implementiert werden kann. Es ist des weiteren auch im Gegensatz zu TCP dank seiner Verbindungslosigkeit in der Lage, Multicast-Nachrichten zu versenden und somit mehrere Empfänger gleichzeitig zu erreichen. Ein abschließender Vergleich von Implementierungen und deren Performanz hat gezeigt, das Pakete von TCP größer ausfallen als die von CoAP mit UDP und auch bei der Geschwindigkeit CoAP dem TCP deutlich überlegen ist. Einzig allein die Codegröße fiel bei CoAP größer aus als die von TCP

# 8. REFERENCES


[1] Andrew S. Tanenbaum: *Computer Networks, Fifth Edition*, Pearson Education Inc., Boston, Massachusets (2011),[3, p. 229ff][6,p. 559ff]

[2] *Constrained Application Protocol (CoAP) draft-ietf-core-coap-16*, `http://tools.ietf.org/html/draft-ietf-core-coap-16`

[3] Thorsten Thormahlen, *Transmission Control Protocol*, `http://www.thormahlen.de/diplhtml/node24.html`

[4] The McGraw-Hill Companies, Inc., 2000, *Chapter 12, Transmission Control Protocol*, `http://medusa.sdsu.edu/network/CS576/Lectures/ch12_TCP.p`

[5] Microsoft,*Multicast IPv6 Addresses* `http://msdn.microsoft.com/en-us/library/aa924142.aspx`

[6] Akbar Rahman, Esko Dijk, IETF, *CoAP Group Communications Concept* `https://www.ietf.org/proceedings/81/slides/core-11.pdf`

[7] Tomislav Dimcic, Srdan Krco1, Nenad Gligoric, *CoAP (Constrained Application Protocol) implementation in M2M Environmental Monitoring System*

[8] Al Danial,*CLOC - Count Lines of Code*,`http://cloc.sourceforge.net/`

[9] Olaf Bergmann, *libcoap: C-Implementation of CoAP*, `http://sourceforge.net/projects/libcoap/`

[10] *Adam Dunkels*, `http://dunkels.com/adam/`

[11] Wikipedia, *uIP (micro IP)*, `http://en.wikipedia.org/wiki/UIP_%28micro_IP%29`

[12] Jozef Kozák, Martin Vaculík, *Application Protocol for constrained nodes in the Internet Of Things*, Journal of Information, Control and Management Systems, Vol. 10, (2012), No.2, YUinfo Conference