CoAP vs HTTP

Projektdokumentation zum Vergleich der Eigenschaften von CoAP und HTTP

Speicher- und Datennetze im IoT
SoSe17
Prof. Dr. Jens-Peter Akelbein

Autoren:
Ibrahim Cinar (754513),
Ken Hasenbank (754067)

Aufgabenstellung

Use-Case-Szenario

Es wird eine Applikation erstellt, die verschiedene Datenszenarien über CoAP und HTTP gegen verschiedene Endgeräte überprüft. Ziel ist es die Performance bzw. das Brutto-Nettoverhalten der beiden Protokolle zu vergleichen.

Hardware- und System-Setup

Der Client kann mit einem Browser verschiedene Datenszenarien über das CoAP bzw. HTTP Protokoll von einem Server abfragen. Der Server läuft auf einem Laptop, Raspberry PI oder optional auf einem ESP826. Dieser kann mit Hilfe einer URI (z.B. http://irgend.was oder coap://irgend.was) aufgerufen werden.

Zu erstellende Demo-Software

Die Applikationen können verschiedene Datenszenarien darstellen, diese können mit wenigen Bytes, wie zum Beispiel einer API Abfrage oder Sensorwerten, aufgerufen werden. Es können auch Kilobytes (z.B. das Aufrufen einer Webseite) als Message dargestellt werden. Für größere Datenszenarien können Dateien übertragen werden. Diese werden sowohl für den HTTP als auch für CoAP implementiert.

Ermittlung von Eigenschaften

Wie findet der Nachrichtenaustausch der beiden Protokolle statt und wie wirkt sich die Datengröße auf die Performance aus? Welche Vor- und Nachteile gibt es zwischen den Protokollen? Wie wirkt sich die Performance der Geräte auf die Antwortzeiten aus?

Evaluation

Kommunikationsverhalten der beiden Protokolle analysieren:

1. Brutto- Nettoverhalten
2. Response-Times (über verschiedene Plattformen)

Präsentationsziele

- Aufgabestellung darstellen
- Eigenschaften bzw. Vergleich der Technologie (HTTP vs. CoAP)
- Abgrenzung von nicht betrachteten Einflussfaktoren
- Darstellung der Evaluationsergebnisse
- Bewertung/Schlussfolgerung der Ergebnisse
- Darstellung des Arbeitsumfangs der Erstellung, Eindruck zur Technologiereife
- Live Demo

Umsetzung

Informationsfindung

Da wir uns bisher noch nicht mit dem Protokoll CoAP beschäftigt hatten, recherchierten wir die hauptsächlichen Unterschiede zwischen den beiden Protokollen.
Wir haben uns zuerst einige Videos angesehen, bis wir schließlich ein gewisses Grundverständnis erworben hatten. Genauere Informationen konnten wir aus dem RFC 7252 „*The Constrained Application Protocol*“ und dem RFC 2616 „*Hypertext Transfer Protocol*“ entnehmen.
Die Hauptsächlichen Unterschiede, die für unsere Messungen relevant waren, sind in der folgenden Tabelle aufgeführt:

	HTTP	CoAP
Verbindungsprotokoll	TCP	UDP
Header	> 150 Byte	4 Byte (+ Optionen)
Max Payload	unbegrenzt	1024

Bei dieser Aufstellung wird schon ersichtlich, dass das CoAP Protokoll auf kleine Geräte mit wenig Rechenleistung sowie möglichst kleinem Payloads ausgelegt ist.
Ein wichtiger Punkt ist, dass CoAP auf das einfachere Verbindungsprotokoll UDP setzt. Allerdings gewährleistet UDP nicht das Ankommen versendeter Pakete, weshalb CoAP zusätzlich den Payload auf maximal 1024 Byte beschränkt. Dadurch ist gewährleistet, dass Teile eines Datagrammes nicht auf dem Weg verloren gehen, da diese so komplett in ein Ethernet Frame passen und nicht fragmentiert werden müssen.
Des Weiteren ist der CoAP Header in seiner minimalen Ausführung nur 4 Byte groß. Es wird nur durch wenige Optionen erweitert, um den nötigen Overhead möglichst gering zu halten.

Auswahl der Testdaten

Zur Auswahl der Testdaten haben wir uns an die in der Aufgabenstellung angegebenen Szenarien gehalten und diese mit unseren Erkenntnissen über den generellen Aufbau der Protokolle kombiniert. Da der Aufbau des Payloads in beiden Protokollen unbedeutend ist, haben wir uns entschieden einfache Textdateien in verschiedenen Größen zu senden. Die Dateien bestehen alle aus der Zeichenfolge „*0123456789ABCDEF*“. Diese werden so lange wiederholt bis die gewünschte Dateigröße erreicht ist.

Folgende Dateigrößen haben wir zum Testen herangezogen:

32 Byte	Als Szenario eine Übertragung von Sensorwerten
2 kByte	Aufruf kleiner Websites !!! Fragmentierungsoverhead gering)
8 kByte	Aufruf kleiner Websites !!! Fragmentierungsoverhead hoch
1 mByte	Download einer Datei (z.b. Bild) !!! Fragmentierungsoverhead sehr hoch

Implementierung eines CoAP Servers

Zur Implementierung des CoAP Servers haben wir uns für die Java Bibliothek *Californium* entschieden.
Diese Bibliothek bot uns den Vorteil, dass sie schon eine große Auswahl an Beispielcode zur Verfügung stellt, sodass es uns sehr leicht fiel einen einfachen Server zu implementieren.

Da wir in unseren Tests die CoAP Implementierung einem HTTP Server gegenüberstellen wollen, haben wir uns dazu entschieden den Server so zu implementieren, dass er beliebige Dateien bereitstellen kann.

```
class SimpleFileRessource extends CoapResource{
    byte[] payload;
    int mediaType;
    public SimpleFileRessource() {
        super("emptyFile");
        getAttributes().setTitle(this.getName());
    }
    public SimpleFileRessource(Path path) {
        super(path.getFileName().toString());
        try {
            this.payload = Files.readAllBytes(path);
        } catch (IOException e) {
            e.printStackTrace();
        }
        String[] tmp = path.toString().split("\\.");
        switch (tmp[tmp.length -1]){
            case "jpg":
            case "jpeg":
                mediaType = MediaTypeRegistry.IMAGE_JPEG;
                break;
            case "png":
                mediaType = MediaTypeRegistry.IMAGE_PNG;
                break;
            default:
                mediaType = MediaTypeRegistry.TEXT_PLAIN;
        }
        getAttributes().setTitle(this.getName());
    }
    public void handleGET(CoapExchange exchange) {
        if (payload == null) {
            exchange.respond("no content");
        }
        else {
            Response resp = new Response(CoAP.ResponseCode.CONTENT);
            resp.getOptions().setContentFormat(this.mediaType);
            resp.setPayload(this.payload);
            exchange.respond(resp);
        }
    }
}
```

Anschließend haben wir den Server so angepasst, dass er alle Dateien aus einem bestimmten Verzeichnis als CoAP Ressource bereitstellt.

```
Path dir = Paths.get("content");
if (Files.exists(dir)) {
    try {
        Files.list(dir).forEach(file ->
                add(new SimpleFileRessource(file.toAbsolutePath())));
    } catch (IOException e) {
        e.printStackTrace();
    }
} else {
    try {
        Files.createDirectory(dir);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Dadurch konnten wir beim Aufsetzen der Testinfrastruktur eine gemeinsame Filebasis für den CoAP und http-Server verwenden.

Implementierung eines Test Clients

Zur Implementierung des Test Clients haben wir uns genau wie beim CoAP Server für die Java Bibliothek *Californium* entschieden. Zunächst implementierten wir ein Interface, dass es uns ermöglichen sollte später die Antwortzeiten der CoAP und http-Anfragen auf eine möglichst gleiche Art und Weise zu messen.

```
private static interface TimeMeasure {
   public String getName();
   public long measure();
}
```

Als nächstes implementierten wir die Klassen *CoapMeasure* und *HttpMeasure*, die das oben definierte Interface verwenden.

```
private static class CoapMeasure implements TimeMeasure {
   private URI uri;
   private CoapClient client;
   private int loop;
   private String name;
   public CoapMeasure(String name, URI uri, int loop) {
      this.uri = uri;
      this.loop = loop;
      this.name = name;

      client = new CoapClient(this.uri);
      client.useEarlyNegotiation(1024);
      client.get();
   }
   public long measure() {
      long start = System.currentTimeMillis();
      boolean isSuccess = true;
      CoapResponse response = null;

      for (int i = 0; i < loop; i++) {
         String content = "";
         response = client.get();
         if (response != null) {
            content += response.getResponseText();
         }
         isSuccess &= response != null && response.isSuccess();
      }
      long duration = System.currentTimeMillis() - start;
      if (isSuccess)
         return duration;
      else
         return -1;
   }
   public String getName() {
      return this.name;
   }
}
```

Da wir Daten transferieren wollten, die das Maximum des CoAP Payloads überschreiten, mussten wir eine Fragmentierung einbauen und haben aus diesem Grund für z.B. den 2KB Testfall die 1KB Datei (passt in den Payload) mehrfach geladen.

Da das http Protokoll keine solchen Limitierungen mitbringt, musste hier keine zusätzliche Logik implementiert werden.

```
private static class HttpMeasure implements TimeMeasure {
   private URL url;
   private int loop;
   private String name;
   public HttpMeasure(String name, URL url, int loop) {
      this.url = url;
      this.loop = loop;
      this.name = name;
   }
   public long measure() {
      try {
         long start = System.currentTimeMillis();
         for (int i = 0; i < loop; i++) {
            HttpURLConnection con =
               (HttpURLConnection) url.openConnection();
            InputStream is = con.getInputStream();
            String content = "";
            try (BufferedReader buffer =
               new BufferedReader(new InputStreamReader(is))) {
               content = buffer.lines().collect(Collectors.joining("\n"));
            }
         }
         long duration = System.currentTimeMillis() - start;

         return duration;
      }
      catch (Exception e) {
         return -1;
      }
   }
   public String getName() {
      return this.name;
   }
}
```

Anschließend werden die Aufrufe nacheinander ausgeführt.

```
List<TimeMeasure> lst = new ArrayList<>();
lst.add(new HttpMeasure("http:32b",
        new URL("http://localhost/32b.txt"), 1));
lst.add(new CoapMeasure("coap:32b",
        new URI("coap://localhost:5683/32b.txt"), 1));

lst.add(new HttpMeasure("http:2kb",
        new URL("http://localhost/2048b.txt"), 1));
lst.add(new CoapMeasure("coap:2kb",
        new URI("coap://localhost:5683/1024b.txt"), 2));

lst.add(new HttpMeasure("http:8kb",
        new URL("http://localhost/8192b.txt"), 1));
lst.add(new CoapMeasure("coap:8kb",
        new URI("coap://localhost:5683/1024b.txt"), 8));

lst.add(new HttpMeasure("http:1mb",
        new URL("http://localhost/1024kb.txt"), 1));
lst.add(new CoapMeasure("coap:1mb",
        new URI("coap://localhost:5683/1024b.txt"), 1024));

for (TimeMeasure m: lst) {
   System.out.print(m.getName() + "(ms)");
   System.out.print("\t");
}
System.out.println();
for (TimeMeasure m: lst) {
   System.out.print(m.measure());
   System.out.print("\t");
}
```

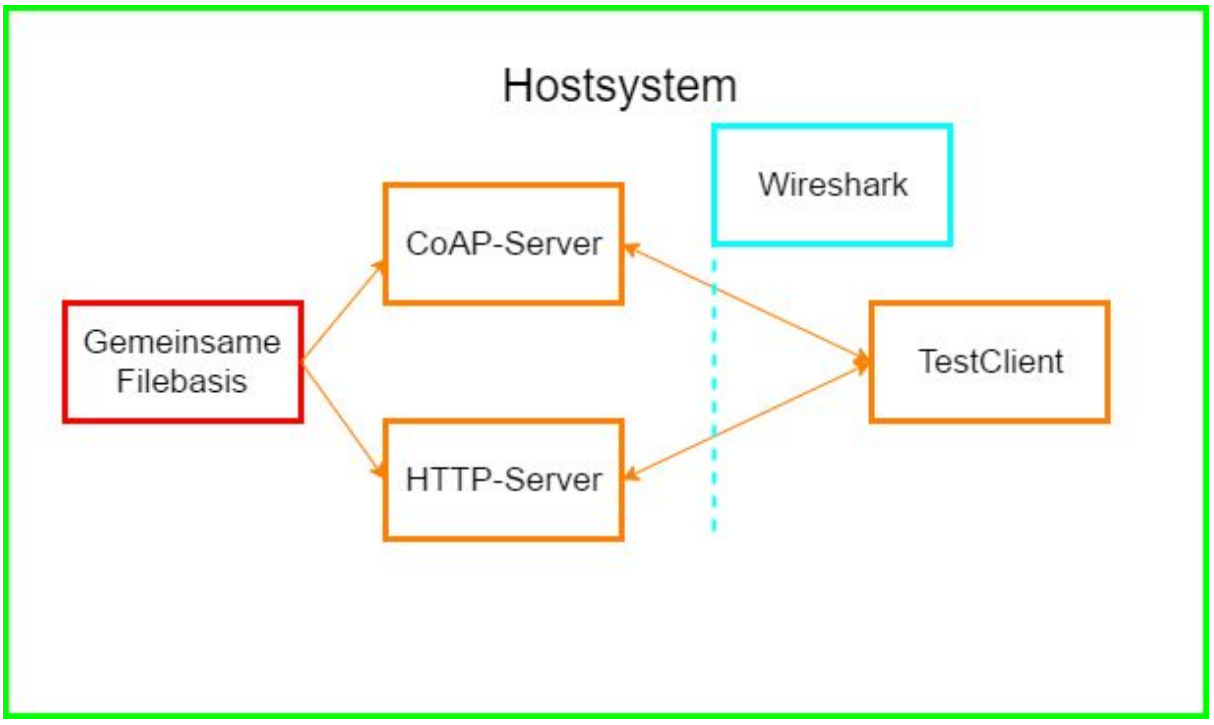
Aufbau des Testumfeldes

Als Testumgebung haben wir uns dazu entschieden alle Komponenten immer auf einem System laufen zu lassen, um so mögliche Verfälschungen über WLAN oder hoher Auslastung des Netzwerkes auszuschließen.
Als Hostsysteme hatten wir 3 verschiedene Konfigurationen:

	Laptop	Raspberry Pi 3	Raspberry Pi 1 (Modell B)
CPU	Intel Core i5 6300U 64bit 2 cores (Hyper Threading) 2.4 GHz	Broadcom BCM2837 ARM Cortex-A53 64bit 4 cores 1.2 GHz	Broadcom BCM2835 ARM Cortex-A53 32bit 1 core 700 MHz
RAM	8 GB	1 GB	256 MB

Zur Auswertung der Header Größen haben wir den Traffic beim Testlauf mit Wireshark aufgezeichnet und ausgewertet.

Als http-Server haben wir auf allen Systemen *NGINX* verwendet.



Konfiguration

Zum Nachstellen der Infrastruktur (z.B. auf einem Debian System) müssen die Binaries des CoAP-Servers und des Test Clients zusammen mit der Filebasis auf dem System (z.B. im ordner /opt/) in folgender Struktur existieren:

```
/ content
        /32b.txt
        /1024b.txt
        /1024kb.txt
        /2048b.txt
        /8192b.txt
/cf-helloworld-client-1.1.0-SNAPSHOT.jar
/cf-helloworld-server-1.1.0-SNAPSHOT.jar
```

Jetzt kann der NGINX Server Installiert werden (apt-get update && apt-get install nginx). Ist die Installation abgeschlossen, muss noch die Konfiguration angepasst werden. Dazu muss man die Datei „*/etc/nginx/sites-available/default*“ mit einem geeigneten Editor öffnen und mit folgendem Inhalt ersetzen:

```
server {
        listen 80;

        location / {
                root /opt/content
        }
}
```

Anschließend muss der Server noch neu gestartet werden, damit die neue Konfiguration geladen wird (service nginx restart).

Jetzt können wir im Ordner /opt/ folgenden Befehl ausführen, um den CoAP Server zu starten:
java -jar cf-helloworld-server-1.1.0-SNAPSHOT.jar

Schließlich kann der Test Client ausgeführt werden:
java -jar cf-helloworld-client-1.1.0-SNAPSHOT.jar

Zur Ausführung des CoAP-Servers sowie des Test-Clients muss auf dem System das Java Runtime Environment Version 8 Installiert sein.

Ergebnisse

Brutto- Nettoverhalten

Zur Auswertung des Brutto- Nettoverhaltens haben wir mithilfe von Wireshark unsere Testverläufe aufgezeichnet.

Der folgende Screenshot zeigt das Datagramm der Response mit den 32 byte Payload der http Anfrage:

```
> Frame 2: 346 bytes on wire (2768 bits), 346 bytes captured (2768 bits) on interface 0
> Linux cooked capture
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 80, Dst Port: 34914, Seq: 1, Ack: 157, Len: 278
> Hypertext Transfer Protocol
> Line-based text data: text/plain
```

```
0000  00 00 03 04 00 06 00 00  00 00 00 00 00 00 08 00   ........ ........
0010  45 00 01 4a 23 b7 40 00  40 06 17 f5 7f 00 00 01   E..J#.@. @.......
0020  7f 00 00 01 00 50 88 62  5c d8 a3 6b 4e 9f f7 ef   .....P.b \..kN...
0030  80 18 01 5e ff 3e 00 00  01 01 08 0a 00 8a 80 dd   ...^.>.. ........
0040  00 8a 80 dd 48 54 54 50  2f 31 2e 31 20 32 30 30   ....HTTP /1.1 200
0050  20 4f 4b 0d 0a 53 65 72  76 65 72 3a 20 6e 67 69    OK..Ser ver: ngi
0060  6e 78 2f 31 2e 31 30 2e  30 20 28 55 62 75 6e 74   nx/1.10. 0 (Ubunt
0070  75 29 0d 0a 44 61 74 65  3a 20 54 68 75 2c 20 30   u)..Date : Thu, 0
0080  38 20 4a 75 6e 20 32 30  31 37 20 31 39 3a 32 31   8 Jun 20 17 19:21
0090  3a 30 34 20 47 4d 54 0d  0a 43 6f 6e 74 65 6e 74   :04 GMT. .Content
00a0  2d 54 79 70 65 3a 20 74  65 78 74 2f 70 6c 61 69   -Type: t ext/plai
00b0  6e 0d 0a 43 6f 6e 74 65  6e 74 2d 4c 65 6e 67 74   n..Conte nt-Lengt
00c0  68 3a 20 33 32 0d 0a 4c  61 73 74 2d 4d 6f 64 69   h: 32..L ast-Modi
00d0  66 69 65 64 3a 20 57 65  64 2c 20 30 37 20 4a 75   fied: We d, 07 Ju
00e0  6e 20 32 30 31 37 20 31  32 3a 34 31 3a 31 39 20   n 2017 1 2:41:19
00f0  47 4d 54 0d 0a 43 6f 6e  6e 65 63 74 69 6f 6e 3a   GMT..Con nection:
0100  20 6b 65 65 70 2d 61 6c  69 76 65 0d 0a 45 54 61    keep-al ive..ETa
0110  67 3a 20 22 35 39 33 37  66 34 36 66 2d 32 30 22   g: "5937 f46f-20"
0120  0d 0a 41 63 63 65 70 74  2d 52 61 6e 67 65 73 3a   ..Accept -Ranges:
0130  20 62 79 74 65 73 0d 0a  0d 0a 30 31 32 33 34 35    bytes.. ..012345
0140  36 37 38 39 41 42 43 44  45 46 30 31 32 33 34 35   6789ABCD EF012345
0150  36 37 38 39 41 42 43 44  45 46                     6789ABCD EF
```

Das folgende Bild zeigt das gleiche für die CoAP Anfrage:

```
> Frame 2: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface 0
> Linux cooked capture
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> User Datagram Protocol, Src Port: 5683, Dst Port: 56107
> Constrained Application Protocol, Acknowledgement, 2.05 Content, MID:39184
```

```
0000  00 00 03 04 00 06 00 00  00 00 00 00 00 00 08 00   ........ ........
0010  45 00 00 4a 49 fd 40 00  40 11 f2 a3 7f 00 00 01   E..JI.@. @.......
0020  7f 00 00 01 16 33 db 2b  00 36 fe 49 68 45 99 10   .....3.+ .6.IhE..
0030  c8 cf 19 24 c3 be 3c 14  c0 ff 30 31 32 33 34 35   ...$..<. ..012345
0040  36 37 38 39 41 42 43 44  45 46 30 31 32 33 34 35   6789ABCD EF012345
0050  36 37 38 39 41 42 43 44  45 46                     6789ABCD EF
```

In den folgenden Tabellen werden die Ergebnisse der Auswertung zusammengefasst.

Requested Payload	CoAP				HTTP			
	Payload	Proto	UDP	Frame	Payload	Proto	TCP	Frame
32b	32	14	8	90	32	246	32	346
2kb	2*1024	2*19	2*8	2*1078	2048	249	32	2365
8kb	8*1024	8*19	8*8	8*1078	8192	250	32	8510
1mb	1024 *1024	1024 *19	1024 *8	1024 *1078	Streamed via TCP			

Verhältnis:

Requested Payload	CoAP	HTTP
32b	36%	9%
2kb	95%	86%
8kb	95%	96%
1mb	95%	>96%

Antwortzeiten

Zur Auswertung der Antwortzeiten haben wir den Versuchsaufbau auf allen drei Systemen so aufgebaut, dass die Tests 20 mal durchlaufen werden. Daraus haben wir die Mittelwerte gebildet und gegenübergestellt (Zeitangaben in ms):

run	coap:32b	coap:2kb	coap:8kb	coap:1mb	http:32b	http:2kb	http:8kb	http:1mb
1	1	1	2	288	2	2	2	44
2	0	0	2	203	2	3	3	44
3	0	1	3	535	2	4	3	42
4	1	1	3	188	3	2	3	47
5	0	1	1	220	2	2	2	45
6	2	1	2	413	15	6	9	46
7	1	3	4	830	2	22	13	55
8	1	1	8	526	4	9	12	41
9	2	2	4	1006	3	3	6	62
10	1	1	5	481	3	3	4	60

Übersicht:

Hostsystem	CoAP				HTTP			
	32b	2kb	8kb	1mb	32b	2kb	8kb	1mb
PC	1.06	1.42	4.32	541.58	5.2	4.08	3.84	52.52
Pi 3	1.24	1.76	5.96	691.38	7.1	2.1	2.2	96.4
Pi B	5.1	18.6	88.3	8828.6	14.35	9.75	11.2	726.9

Fazit

Anhand der oben gezeigten Werte lässt sich schließen, dass das CoAP Protokoll mit weniger Rechenleistung und kleineren Payloads weitaus effizienter (bzgl. Antwortzeiten) sowie auch effektiver (bzgl. Brutto- Nettoverhalten) ist.
Unsere Tests zeigen, dass der extrem große Overhead (> 240 Bytes)des http-Protokolls selbst bei genügend Rechenleistung einen nicht Vernachlässigbaren unterschied macht.
Allerdings setzen die Einschränkungen des CoAP Protokolls eine obere Schranke für diese positiven Effekte. Sodass CoAP ein maximales Brutto- Nettoverhältnis von 95% erreicht, sobald das Paket vollständig gefüllt ist. Größere Daten müssen fragmentiert und in mehrere CoAP Pakete unterteilt werden.
Bei http hingegen steigt dieses Verhältnis mit größer werdenden Payloads immer weiter an, sodass http CoAP schnell überholt, sobald die CoAP-Fragmentierung eingesetzt wird.

Quellen

RFC 7252 „The Constrained Application Protocol“ URL:
https://tools.ietf.org/html/rfc7252
(zuletzt abgerufen: 16.06.2017)

RFC 2616 „Hypertext Transfer Protocol“ URL:
https://www.ietf.org/rfc/rfc2616.txt
(zuletzt abgerufen: 16.06.2017)

Californium URL:
https://github.com/eclipse/californium
(zuletzt abgerufen: 16.06.2017)