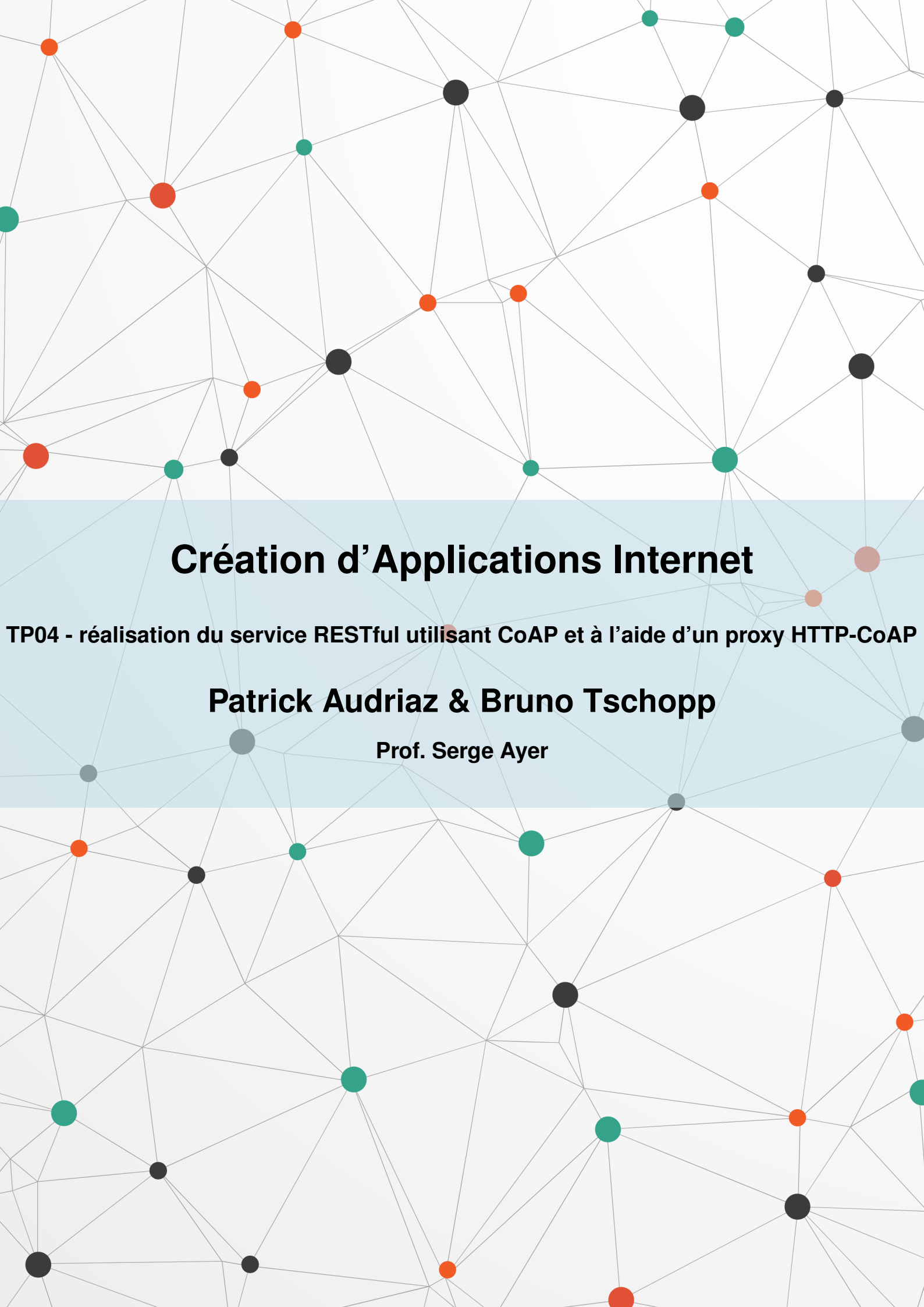

Création d'Applications Internet

TP04 - réalisation du service RESTful utilisant CoAP et à l'aide d'un proxy HTTP-CoAP

Patrick Audriaz & Bruno Tschopp

Prof. Serge Ayer

Table des matières

1. Introduction

Etendre les fonctionnalités du service RESTful développé dans le cadre du TP 3 afin de pouvoir se connecter à une station météo réalisant un serveur CoAP. Le client développé dans le TP 2 doit fonctionner correctement et sans modification avec le nouveau serveur CoAP. L'accès aux serveurs CoAP est réalisé à l'aide d'un proxy HTTP-CoAP qui doit être développé sur la base d'une réalisation de route Express existante

2. Questions

2.1 P1 - Analyse de trafic. A l'aide d'un outil d'analyse de trafic, affichez le trafic réseau (couche HTTP – entre le client et le serveur HTTP) requis pour la mise à jour des données d'une station météo (mise à jour des données d'un capteur choisi, par exemple la température, ou mise à jour de l'ensemble des capteurs).

2.1.1 GET

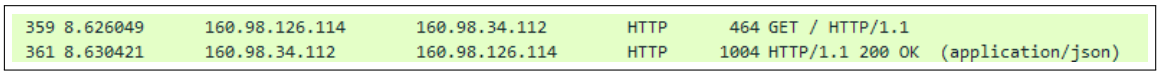

FIGURE 2.1 – Requête et réponse HTTP pour la méthode GET

Nous allons en premier analyser tout le traffic générer pour le GET.

TCP Request

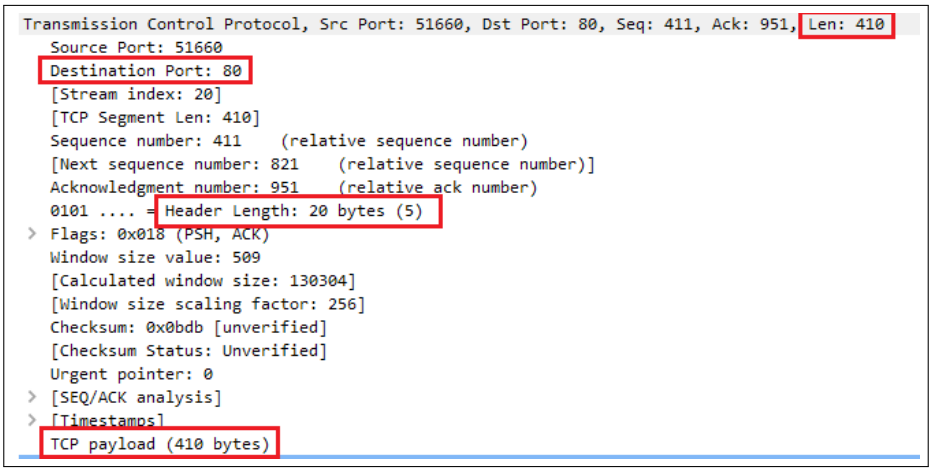

FIGURE 2.2 – Requête TCP

On utilise TCP. On y voit le port de destination (80) qui correspond à HTTP, La taille du payload (410 bytes) ainsi que la taille du header TCP (20 bytes).

HTTP Request

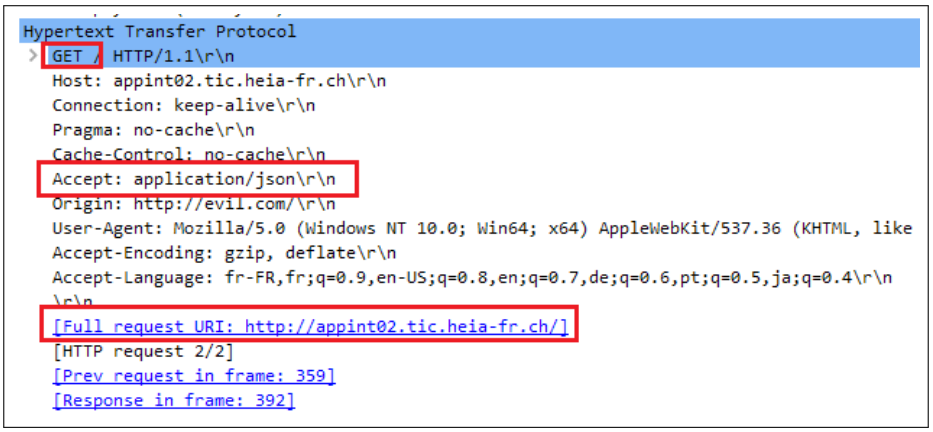

FIGURE 2.3 – Requête HTTP

On y voit la méthode HTTP faite (GET) ainsi que l'URI vers laquelle elle est faite.

TCP Response

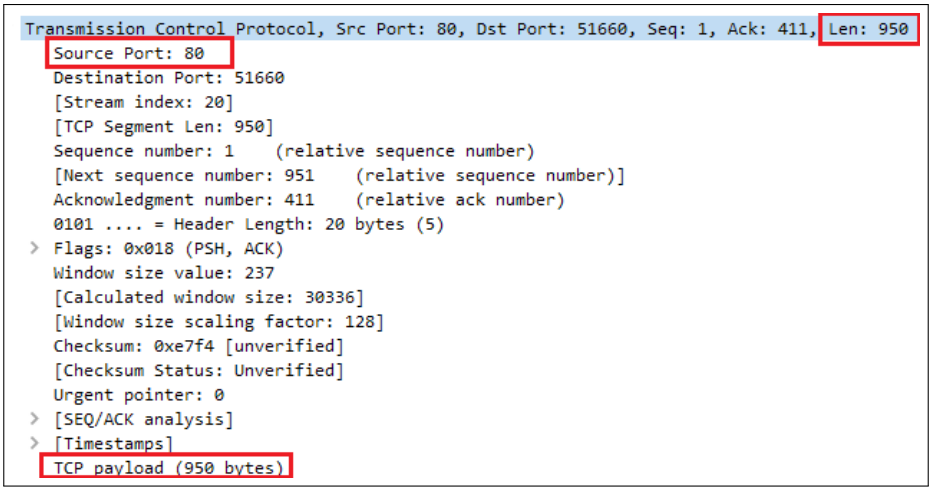

FIGURE 2.4 – Réponse TCP

On peut constater que la taille du payload est bien plus grande (950 bytes) car elle contient les données JSON demandés via l'URI en retour.

HTTP Response

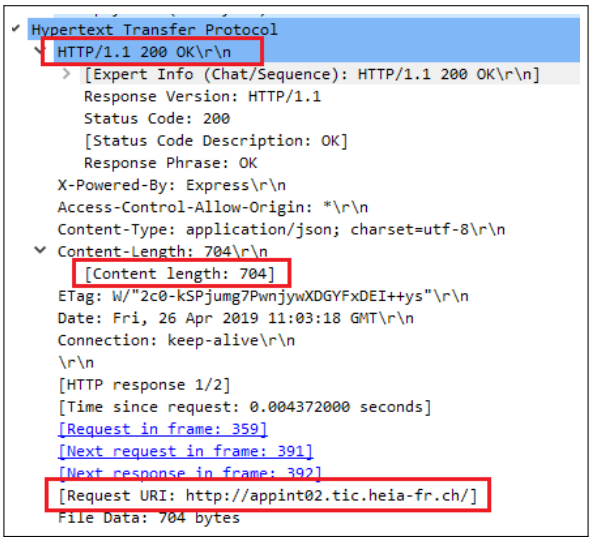

FIGURE 2.5 – Réponse HTTP

On a en retour un code HTTP (200) nous indiquant que tout s'est bien passé ainsi que la longueur du contenu du JSON (704 bytes).

JSON Response

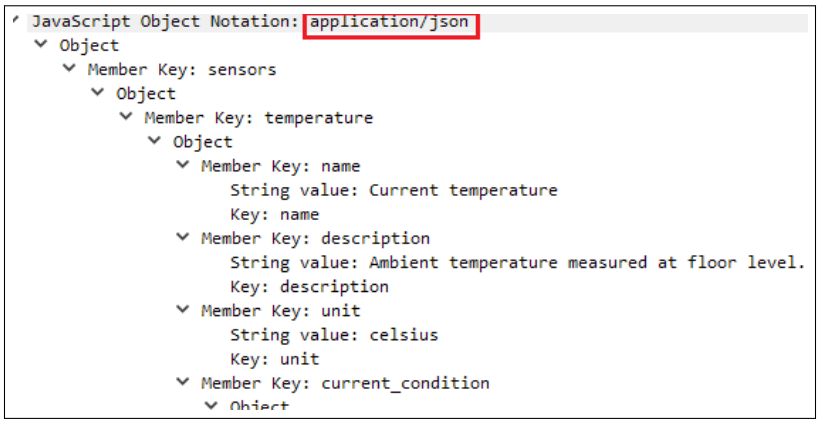

FIGURE 2.6 – Réponse JSON

Réponse contenant les données sous format JSON.

2.1.2 PUT

```
317 3.606761    160.98.126.114    160.98.34.112     HTTP    561 PUT /actuators/state HTTP/1.1  (application/json)
318 3.611534    160.98.34.112     160.98.126.114    HTTP    403 HTTP/1.1 200 OK  (application/json)
```

FIGURE 2.7 – Requête et réponse HTTP pour la méthode PUT

Nous allons montrer quels sont les différences par rapport au GET :

HTTP Request

```
Hypertext Transfer Protocol
  PUT /actuators/state HTTP/1.1\r\n
  Host: appint02.tic.heia-fr.ch\r\n
  Connection: keep-alive\r\n
  Content-Length: 37\r\n
  Pragma: no-cache\r\n
  Cache-Control: no-cache\r\n
  Accept: undefined\r\n
  Origin: http://evil.com/\r\n
  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, li
  Content-Type: application/json\r\n
  Accept-Encoding: gzip, deflate\r\n
  Accept-Language: fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7,de;q=0.6,pt;q=0.5,ja;q=0.4\r\n
  \r\n
  [Full request URI: http://appint02.tic.heia-fr.ch/actuators/state]
  [HTTP request 2/2]
  [Prev request in frame: 314]
  [Response in frame: 318]
  File Data: 37 bytes
```

FIGURE 2.8 – Requête HTTP

La requête est maintenant "PUT" et l'URI pointe vers /actuators/state.

JSON Request

```
JavaScript Object Notation: application/json
  Object
    Member Key: name
        String value: Station state
        Key: name
    Member Key: value
        True value
        Key: value
```

FIGURE 2.9 – Requête JSON

On envoie la valeur "true" à l'état.

2.2 P2 - Analyse de trafic. A l'aide d'un outil d'analyse de trafic, affichez le trafic réseau (couche COAP – entre le serveur HTTP/Proxy et le serveur COAP) requis pour la mise à jour des données d'une station météo – ceci doit être effectuée pour la même requête que la requête analysée à la question 2.

2.2.1 GET

```
16 1.297525    160.98.126.114    160.98.34.112     CoAP     53 CON, MID:57495, GET, TKN:ea 14 ce 99
17 1.301064    160.98.34.112     160.98.126.114    CoAP     770 ACK, MID:57495, 2.05 Content, TKN:ea 14 ce 99 (application/json)
```

FIGURE 2.10 – Requête et réponse CoAP pour la méthode PUT

Requête CoAP

```
✔ Internet Protocol Version 4, Src: 160.98.126.114, Dst: 160.98.34.112
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  › Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 39
    Identification: 0xd330 (54064)
  › Flags: 0x0000
    Time to live: 128
    Protocol: UDP (17)
    Header checksum: 0x85ee [validation disabled]
    [Header checksum status: Unverified]
    Source: 160.98.126.114
    Destination: 160.98.34.112
✔ User Datagram Protocol, Src Port: 52157, Dst Port: 5683
    Source Port: 52157
    Destination Port: 5683
    Length: 19
    Checksum: 0x5be3 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 3]
✔ Constrained Application Protocol, Confirmable, GET, MID:57495
    01.. .... = Version: 1
    ..00 .... = Type: Confirmable (0)
        0100 = Token Length: 4
    Code: GET (1)
    Message ID: 57495
    Token: ea14ce99
  ⌄ Opt Name: #1: Accept: application/json
      Opt Desc: Type 17, Critical, Safe
      1101 .... = Opt Delta: 13
      .... 0001 = Opt Length: 1
      Opt Delta extended: 4
      Accept: application/json
    [Response In: 17]
```

FIGURE 2.11 – Requête CoAP

On peut constater que la taille du payload a grandement diminué, qu'on utilise à présent UDP et que le port de destination change également (5683 pour CoAP). La méthode GET reste la même et est indiquée. Il n'y a par contre pas l'URI qui est indiqué dans la méthode GET et on utilise à la place les options du header CoAP. Le message est un message CON.

Réponse CoAP

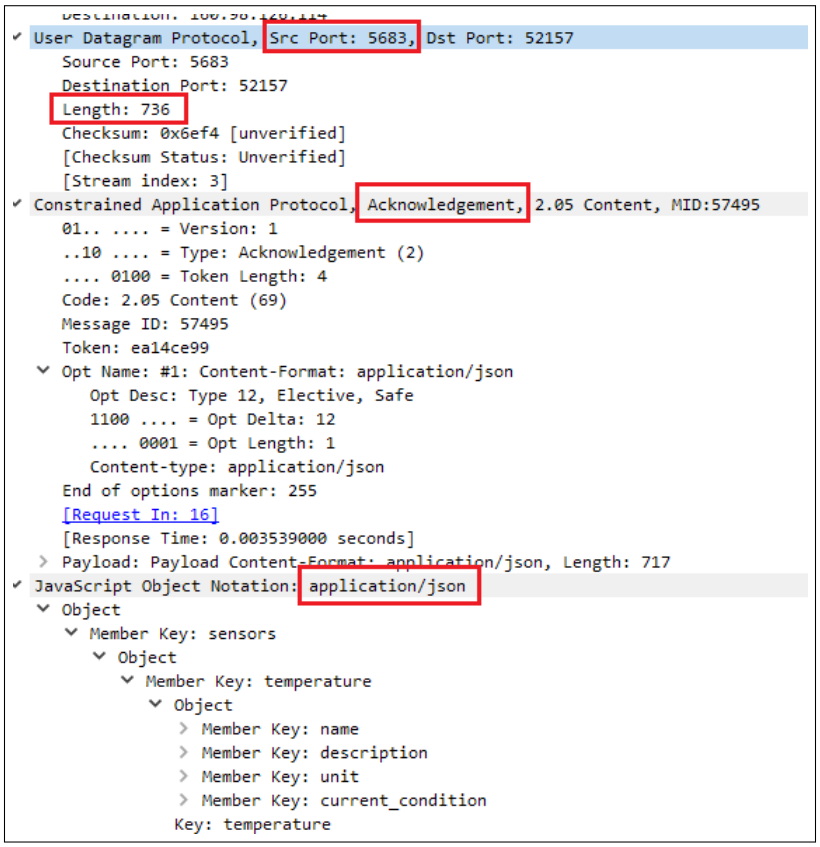

FIGURE 2.12 – Réponse CoAP

On répond au message CON par un message ACK. La taille est bien plus grande car le JSON contient les dnonnées demandées.

2.2.2 PUT

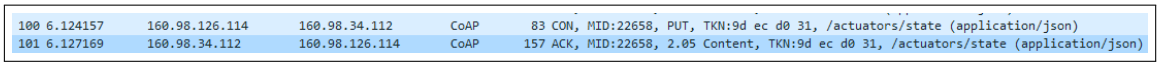

FIGURE 2.13 – Requête et réponse CoAP pour la méthode PUT

Requête CoAP

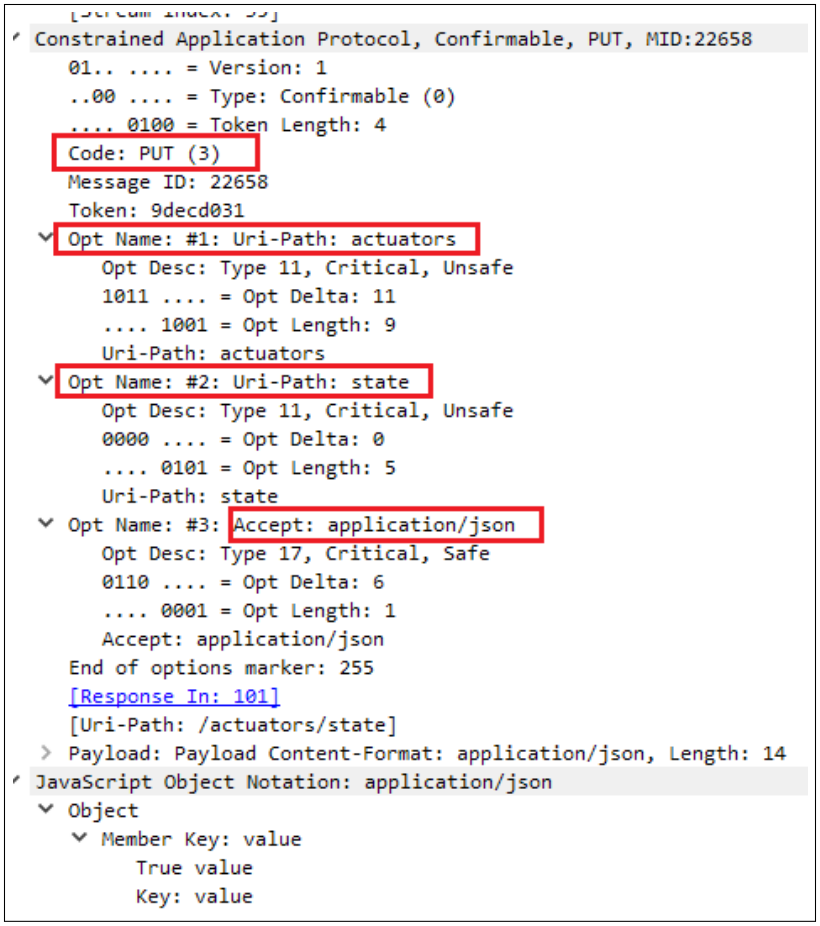

FIGURE 2.14 – Requête CoAP

On peut constater qu'on utilise les options pour accéder à l'URI souhaitée (/actuators/state) et que l'on passe la valeur via le JSON (true).

Réponse CoAP

La réponse se fait de la même manière que pour la réponse de la requête CoAP GET ci-dessus.

2.3 P3 - Comparer les résultats obtenus dans les questions 2 et 3, en commentant vos constatations.

Les différences notables que nous avons trouvé sont les suivantes :

2.3.1 TCP / UDP

CoAP : UDP
HTTP : TCP

2.3.2 Taille header

HTTP :
Total Payload IPv4 (avec en-tête 20 bytes) : 450 bytes
Header TCP : 20 bytes
Payload HTTP : 410 bytes

CoAP :
Total Payload IPv4 (avec en-tête 20 bytes) : 39 bytes
Header UDP : 8 bytes
Payload CoAP : 11 bytes

2.3.3 Constatations

CoAP utilise UDP plutôt que TCP (stateless, non-connecté et asynchrone) ce qui lui permet d'avoir de bien meilleures performances sur un réseaux sans fil à faible consommation comparé à HTTP. Il doit en contrepartie ajouter une couche de contrôle permettant la QoS.

Une taille de payload énormément réduite par rapport à HTTP : moins de mémoire et moins de bande passante

3. Conclusions

Un TP qui était assez vite fait pour tout ce qui est du serveur. Nous avons par contre peiné pour intégrer ceci dans notre application existante à cause d'un problème avec CORS. Des modifications dans le fichier htttp-coap-proxy.js ont permis de régler le problème. Tout fonctionne à présent en harmonie.
Nous avons compris l'utilité de CoAP, ce qui a motivé sa création et ses différences avec HTTP.

Fribourg, 26 avril 2019

Patrick Audriaz

Bruno Tschopp