

The background of the slide is a network diagram. It consists of a light gray background with a complex web of thin gray lines connecting various nodes. The nodes are represented by small circles in four colors: red, teal, black, and light gray. These nodes are distributed across the entire slide, with a higher density in the upper and lower portions. A horizontal light blue band runs across the middle of the slide, serving as a backdrop for the text.

# **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

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Questions</b>	<b>4</b>
<b>2.1</b>	<b>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).</b>	<b>4</b>
2.1.1	GET	4
2.1.2	PUT	7
<b>2.2</b>	<b>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.</b>	<b>8</b>
2.2.1	GET	8
2.2.2	PUT	10
<b>2.3</b>	<b>P3 - Comparer les résultats obtenus dans les questions 2 et 3, en commentant vos constatations.</b>	<b>11</b>
2.3.1	TCP / UDP	11
2.3.2	Taille header	11
2.3.3	Constatations	11
<b>3</b>	<b>Conclusions</b>	<b>12</b>

A background graphic featuring a network of interconnected nodes and lines. The nodes are represented by small circles in various colors: red, green, black, and orange. The lines are thin and grey, creating a complex web-like structure. A blue rounded rectangle is overlaid on the left side of the image, containing the section header.

# 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

359	8.626049	160.98.126.114	160.98.34.112	HTTP	464 GET / HTTP/1.1
361	8.630421	160.98.34.112	160.98.126.114	HTTP	1004 HTTP/1.1 200 OK (application/json)

FIGURE 2.1 – Requête et réponse HTTP pour la méthode GET

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

#### TCP Request

```
Transmission Control Protocol, Src Port: 51660, Dst Port: 80, Seq: 411, Ack: 951, Len: 410
  Source Port: 51660
  Destination Port: 80
  [Stream index: 20]
  [TCP Segment Len: 410]
  Sequence number: 411 (relative sequence number)
  [Next sequence number: 821 (relative sequence number)]
  Acknowledgment number: 951 (relative ack number)
  0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x018 (PSH, ACK)
  Window size value: 509
  [Calculated window size: 130304]
  [Window size scaling factor: 256]
  Checksum: 0x0bdb [unverified]
  [Checksum Status: Unverified]
  Urgent pointer: 0
  > [SEQ/ACK analysis]
  > [Timestamps]
  TCP payload (410 bytes)
```

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

```
Hypertext Transfer Protocol
> GET / HTTP/1.1\r\n
Host: appint02.tic.heia-fr.ch\r\n
Connection: keep-alive\r\n
Pragma: no-cache\r\n
Cache-Control: no-cache\r\n
Accept: application/json\r\n
Origin: http://evil.com/\r\n
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
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/]
[HTTP request 2/2]
[Prev request in frame: 359]
[Response in frame: 392]
```

FIGURE 2.3 – Requête HTTP

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

## TCP Response

```
Transmission Control Protocol, Src Port: 80, Dst Port: 51660, Seq: 1, Ack: 411, Len: 950
Source Port: 80
Destination Port: 51660
[Stream index: 20]
[TCP Segment Len: 950]
Sequence number: 1 (relative sequence number)
[Next sequence number: 951 (relative sequence number)]
Acknowledgment number: 411 (relative ack number)
0101 .... = Header Length: 20 bytes (5)
> Flags: 0x018 (PSH, ACK)
Window size value: 237
[Calculated window size: 30336]
[Window size scaling factor: 128]
Checksum: 0xe7f4 [unverified]
[Checksum Status: Unverified]
Urgent pointer: 0
> [SEQ/ACK analysis]
> [Timestamps]
TCP payload (950 bytes)
```

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

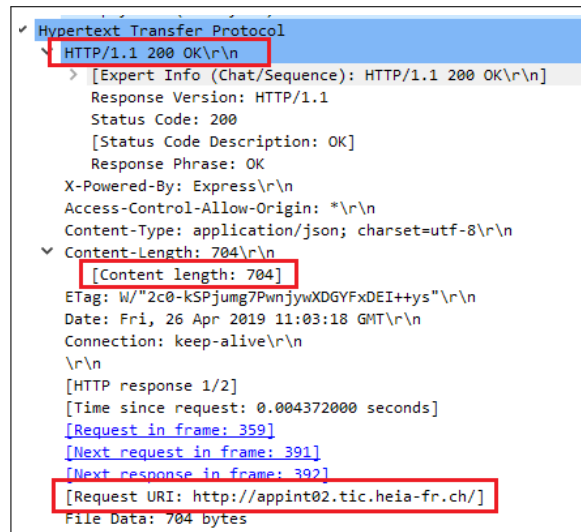


FIGURE 2.5 – Réponse HTTP

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

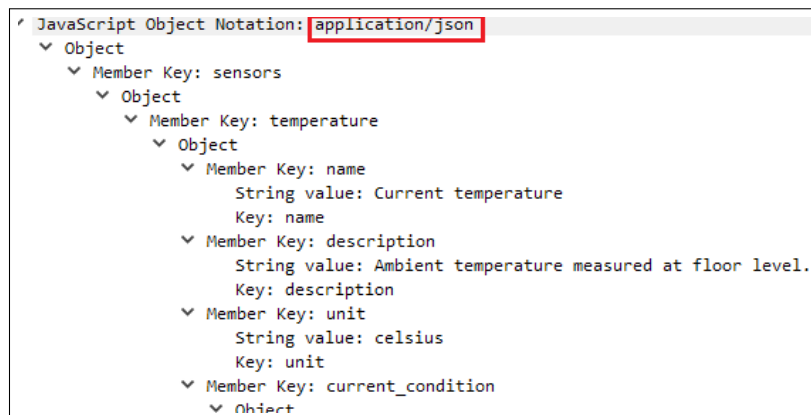


FIGURE 2.6 – Réponse JSON

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

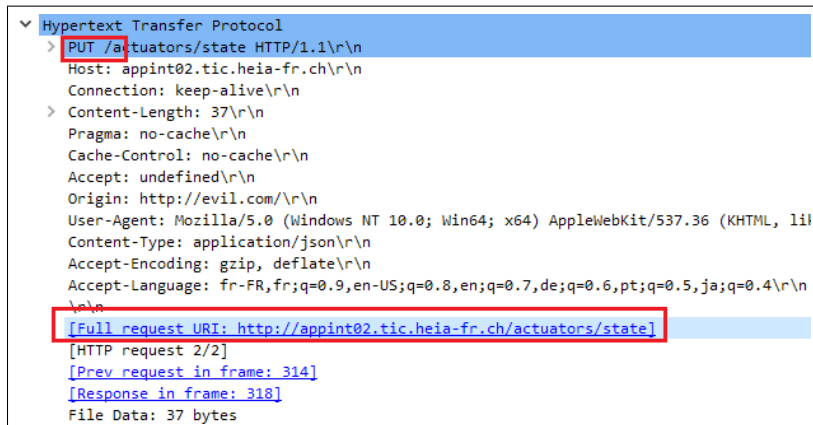


FIGURE 2.8 – Requête HTTP

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

#### JSON Request

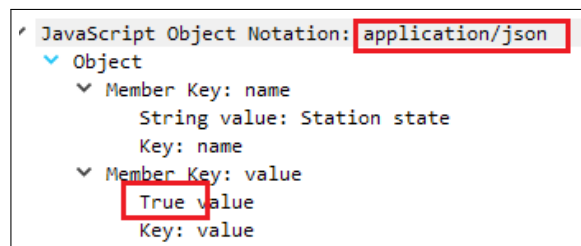


FIGURE 2.9 – Requête JSON

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 indiquée dans la méthode GET et on utilise à la place les options du header CoAP. Le message est un message CON.



## Réponse CoAP

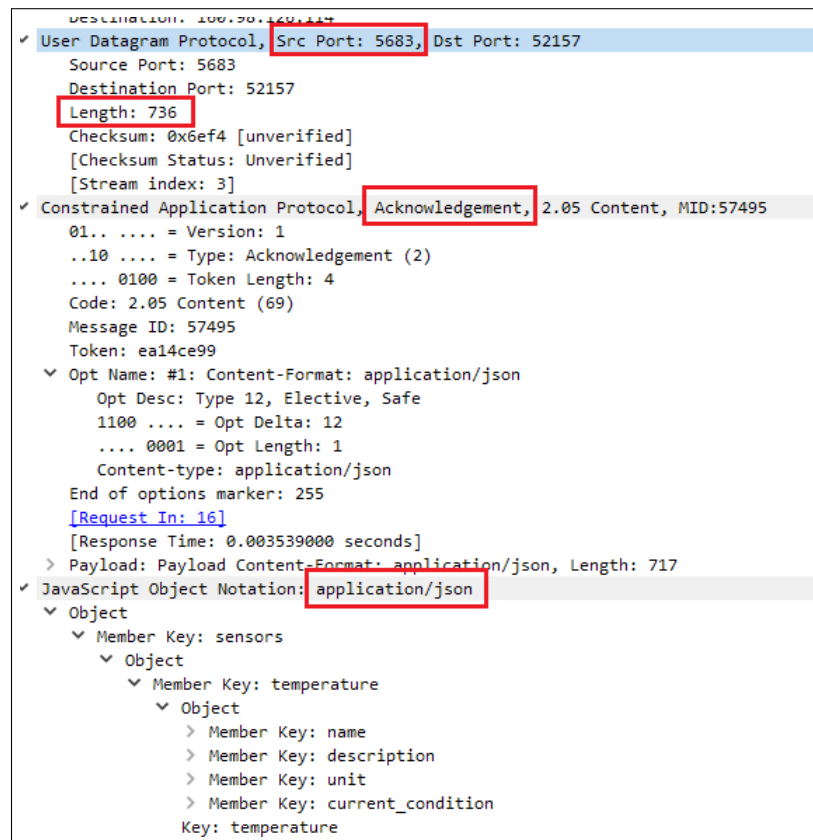


FIGURE 2.12 – Réponse CoAP

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

### 2.2.2 PUT

100 6.124157	160.98.126.114	160.98.34.112	CoAP	83 CON, MID:22658, PUT, TKN:9d ec d0 31, /actuators/state (application/json)
101 6.127169	160.98.34.112	160.98.126.114	CoAP	157 ACK, MID:22658, 2.05 Content, TKN:9d ec d0 31, /actuators/state (application/json)

FIGURE 2.13 – Requête et réponse CoAP pour la méthode PUT

#### Requête CoAP

```

[Source: Android: 55]
/ Constrained Application Protocol, Confirmable, PUT, MID:22658
  01.. .... = Version: 1
  ..00 .... = Type: Confirmable (0)
  .... 0100 = Token Length: 4
  Code: PUT (3)
  Message ID: 22658
  Token: 9decd031
  ✓ Opt Name: #1: Uri-Path: actuators
    Opt Desc: Type 11, Critical, Unsafe
    1011 .... = Opt Delta: 11
    .... 1001 = Opt Length: 9
    Uri-Path: actuators
  ✓ Opt Name: #2: Uri-Path: state
    Opt Desc: Type 11, Critical, Unsafe
    0000 .... = Opt Delta: 0
    .... 0101 = Opt Length: 5
    Uri-Path: state
  ✓ Opt Name: #3: Accept: application/json
    Opt Desc: Type 17, Critical, Safe
    0110 .... = Opt Delta: 6
    .... 0001 = Opt Length: 1
    Accept: application/json
  End of options marker: 255
  [Response In: 101]
  [Uri-Path: /actuators/state]
  > Payload: Payload Content-Format: application/json, Length: 14
/ JavaScript Object Notation: application/json
  ✓ Object
    ✓ Member Key: value
      True value
      Key: value

```

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 trouvées 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 réseau 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 `http-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