

HTTP vs CoAP

The aim of this laboratory task is to compare HTTPS and CoAP. Wireshark will be used to analyze the way traffic is exchanged when making HTTPS and CoAP GET requests from websites.

The task

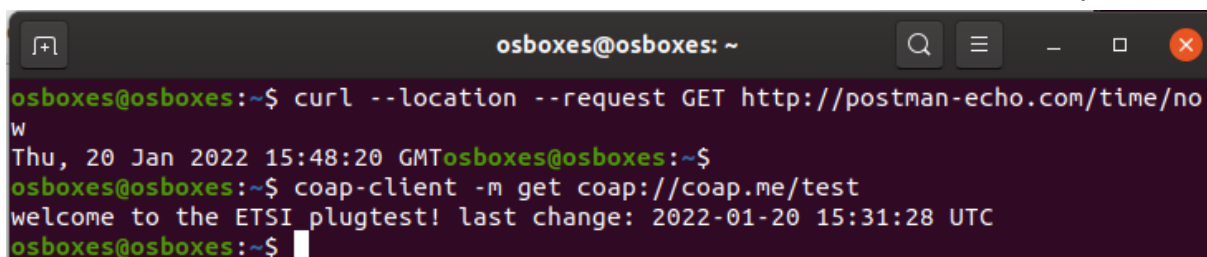
First we launch wireshark on the enp0s3 interface, then we execute the following commands to make GET requests:

```
curl --location --request GET http://postman-echo.com/time/now  
coap-client -m get coap://coap.me/test
```

After executing the commands, capturing is stopped and a wireshark filter is applied to only see HTTP and CoAP related packets.

Console output

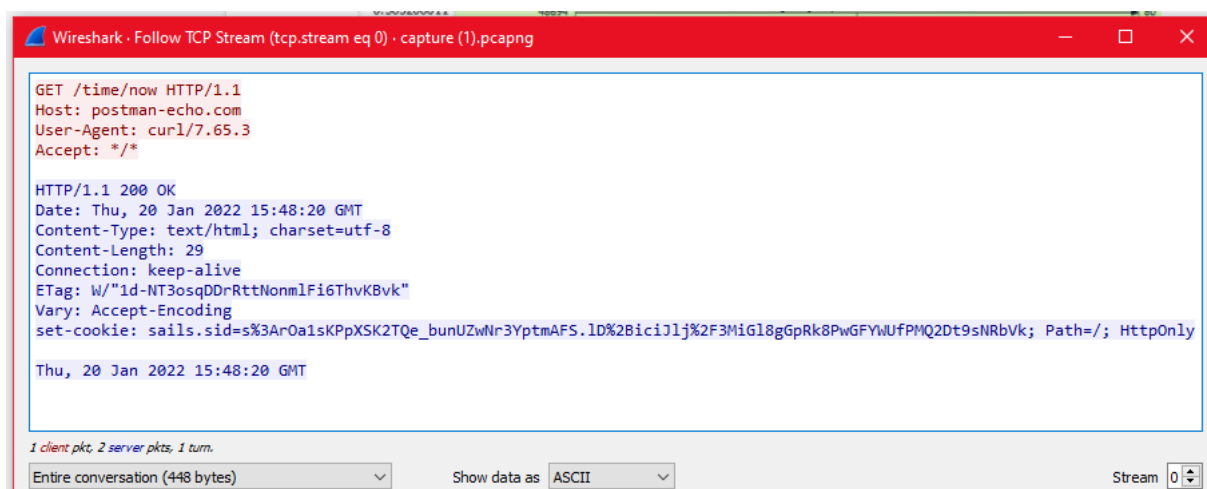
The commands in the console were executed, and this was the obtained output:

A terminal window titled 'osboxes@osboxes: ~' with a search icon, menu icon, and window controls. It shows the execution of two commands. The first command is 'curl --location --request GET http://postman-echo.com/time/now', which returns 'Thu, 20 Jan 2022 15:48:20 GMT'. The second command is 'coap-client -m get coap://coap.me/test', which returns 'welcome to the ETSI plugtest! last change: 2022-01-20 15:31:28 UTC'.

```
osboxes@osboxes:~$ curl --location --request GET http://postman-echo.com/time/now  
Thu, 20 Jan 2022 15:48:20 GMT  
osboxes@osboxes:~$ coap-client -m get coap://coap.me/test  
welcome to the ETSI plugtest! last change: 2022-01-20 15:31:28 UTC  
osboxes@osboxes:~$
```

Wireshark results - TCP (HTTP) stream

Below we show packets that correspond to the curl HTTP request:

A Wireshark window titled 'Wireshark · Follow TCP Stream (tcp.stream eq 0) · capture (1).pcapng'. It displays the details of a TCP stream. The client request is 'GET /time/now HTTP/1.1' with host 'postman-echo.com', user-agent 'curl/7.65.3', and accept '*//*'. The server response is 'HTTP/1.1 200 OK' with date 'Thu, 20 Jan 2022 15:48:20 GMT', content-type 'text/html; charset=utf-8', content-length '29', connection 'keep-alive', etag 'W/"1d-NT3osqDDrRttNonm1Fi6ThvKBvk"', vary 'Accept-Encoding', and set-cookie 'sails.sid=s%3ArOa1sKPpXSK2TQe_bunUZwNr3YptmAFS.1D%2Bici1lj%2F3MiG18gGpRk8PwGFYwUfPMQ2Dt9sNRbVvk; Path=/; HttpOnly'. The timestamp for both is 'Thu, 20 Jan 2022 15:48:20 GMT'. At the bottom, it shows '1 client pkt, 2 server pkts, 1 turn', 'Entire conversation (448 bytes)', 'Show data as ASCII', and 'Stream 0'.

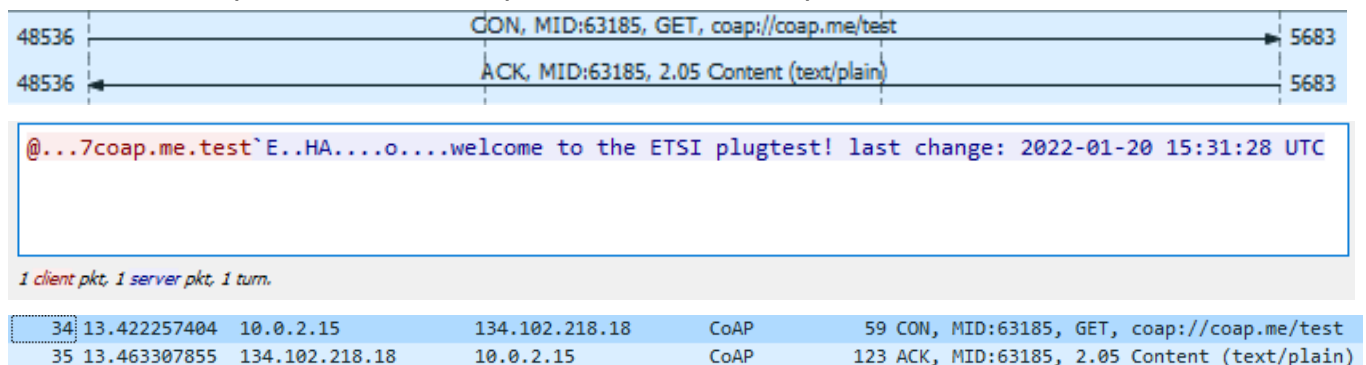
```
GET /time/now HTTP/1.1  
Host: postman-echo.com  
User-Agent: curl/7.65.3  
Accept: */*  
  
HTTP/1.1 200 OK  
Date: Thu, 20 Jan 2022 15:48:20 GMT  
Content-Type: text/html; charset=utf-8  
Content-Length: 29  
Connection: keep-alive  
ETag: W/"1d-NT3osqDDrRttNonm1Fi6ThvKBvk"  
Vary: Accept-Encoding  
set-cookie: sails.sid=s%3ArOa1sKPpXSK2TQe_bunUZwNr3YptmAFS.1D%2Bici1lj%2F3MiG18gGpRk8PwGFYwUfPMQ2Dt9sNRbVvk; Path=/; HttpOnly  
  
Thu, 20 Jan 2022 15:48:20 GMT
```



11	0.102430911	10.0.2.15	54.236.132.16	TCP	74	48694 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2924008444 TSecr=0 WS=128
12	0.231677241	54.236.132.16	10.0.2.15	TCP	60	80 → 48694 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
13	0.231712178	10.0.2.15	54.236.132.16	TCP	54	48694 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
14	0.232162187	10.0.2.15	54.236.132.16	HTTP	142	GET /time/now HTTP/1.1
15	0.232564079	54.236.132.16	10.0.2.15	TCP	60	80 → 48694 [ACK] Seq=1 Ack=89 Win=65535 Len=0
16	0.365189588	54.236.132.16	10.0.2.15	TCP	413	80 → 48694 [PSH, ACK] Seq=1 Ack=89 Win=65535 Len=359 [TCP segment of a reassembled PDU]
17	0.365206011	10.0.2.15	54.236.132.16	TCP	54	48694 → 80 [ACK] Seq=89 Ack=360 Win=63881 Len=0
18	0.365534993	54.236.132.16	10.0.2.15	HTTP	60	HTTP/1.1 200 OK (text/html)
19	0.365541212	10.0.2.15	54.236.132.16	TCP	54	48694 → 80 [ACK] Seq=89 Ack=361 Win=63881 Len=0
20	0.366000783	10.0.2.15	54.236.132.16	TCP	54	48694 → 80 [FIN, ACK] Seq=89 Ack=361 Win=63881 Len=0
21	0.366283712	54.236.132.16	10.0.2.15	TCP	60	80 → 48694 [ACK] Seq=361 Ack=90 Win=65535 Len=0
22	0.490552499	54.236.132.16	10.0.2.15	TCP	60	80 → 48694 [FIN, ACK] Seq=361 Ack=90 Win=65535 Len=0
23	0.490578595	10.0.2.15	54.236.132.16	TCP	54	48694 → 80 [ACK] Seq=90 Ack=362 Win=63881 Len=0

Wireshark results - UDP (CoAP) stream

Below we show packets that correspond to the CoAP request:



HTTP analysis

a. What is the response displayed in the Linux terminal? How many characters were shown (counting spaces and punctuation marks)?

The response shows UTC time at the moment of the request. The exact text is:

Wed, 19 Jan 2022 22:45:26 GMT

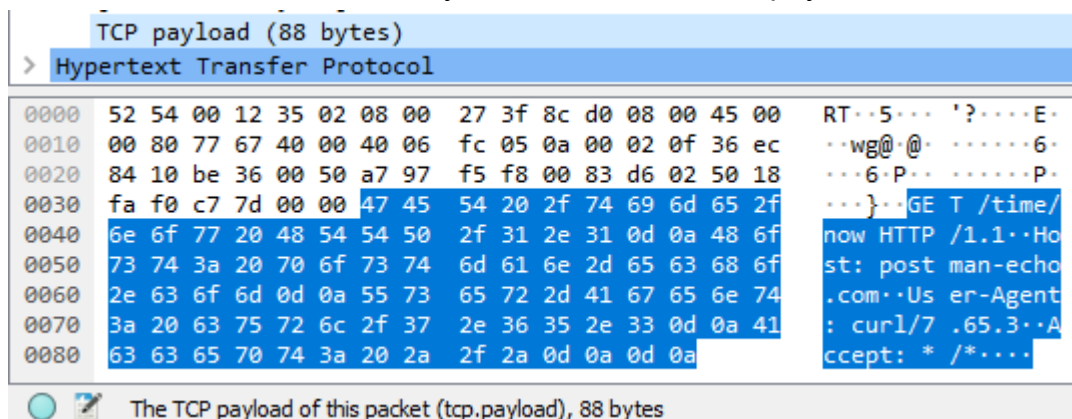
Which is in total 29 characters when including spaces.

b. How many HTTP messages were exchanged?

Two, as shown on the screenshots provided in **Wireshark results - TCP (HTTP) stream** paragraph (flow sequence and packet list).

c. How many bytes does the payload of the transport layer protocol need to encode the HTTP request? How many for the HTTP response?

As shown on screenshot - 88 bytes is the size of TCP payload:



TCP payload of the response is 359 bytes:

TCP payload (359 bytes)	
[Reassembled PDU in frame: 18]	
0020	02 0f 00 50 be 36 00 83 d6 02 a7 97 f6 50 50 18 ...P·6· ····PP
0030	ff ff 37 44 00 00 48 54 54 50 2f 31 2e 31 20 32 ..7D·HT TP/1.1
0040	30 30 20 4f 4b 0d 0a 44 61 74 65 3a 20 54 68 75 00 OK·D ate: Thu
0050	2c 20 32 30 20 4a 61 6e 20 32 30 32 32 20 31 35 , 20 Jan 2022 15
0060	3a 34 38 3a 32 30 20 47 4d 54 0d 0a 43 6f 6e 74 :48:20 G MT·Cont
0070	65 6e 74 2d 54 79 70 65 3a 20 74 65 78 74 2f 68 ent-Type : text/l
0080	74 6d 6c 3b 20 63 68 61 72 73 65 74 3d 75 74 66 tml; cha rset=ut
0090	2d 38 0d 0a 43 6f 6e 74 65 6e 74 2d 4c 65 6e 67 -8·Cont ent-Leng
00a0	74 68 3a 20 32 39 0d 0a 43 6f 6e 6e 65 63 74 69 th: 29· Connect
00b0	6f 6e 3a 20 6b 65 65 70 2d 61 6c 69 76 65 0d 0a on: keep -alive·
00c0	45 54 61 67 3a 20 57 2f 22 31 64 2d 4e 54 33 6f ETag: W/ "1d-NT3c
00d0	73 71 44 44 72 52 74 74 4e 6f 6e 6d 6c 46 69 36 sqDDrRtt NonmlFie
00e0	54 68 76 4b 42 76 6b 22 0d 0a 56 61 72 79 3a 20 ThvKBvk" ·Vary:
00f0	41 63 63 65 70 74 2d 45 6e 63 6f 64 69 6e 67 0d Accept-E ncoding
0100	0a 73 65 74 2d 63 6f 6f 6b 69 65 3a 20 73 61 69 ·set-coo kie: sa
0110	6c 73 2e 73 69 64 3d 73 25 33 41 72 4f 61 31 73 ls.sid=s %3ArOa1s
0120	4b 50 70 58 53 4b 32 54 51 65 5f 62 75 6e 55 5a KPpXSK2T Qe_bunU
0130	77 4e 72 33 59 70 74 6d 41 46 53 2e 6c 44 25 32 wNr3Yptm AFS.lD%
0140	42 69 63 69 4a 6c 6a 25 32 46 33 4d 69 47 6c 38 BiciJlj% 2F3MiGl
0150	67 47 70 52 6b 38 50 77 47 46 59 57 55 66 50 4d gGpRk8Pw GFYWUfP
0160	51 32 44 74 39 73 4e 52 62 56 6b 3b 20 50 61 74 Q2Dt9sNR bVvk; Pa
0170	68 3d 2f 3b 20 48 74 74 70 4f 6e 6c 79 0d 0a 0d h=/; Htt pOnly·
0180	0a 54 68 75 2c 20 32 30 20 4a 61 6e 20 32 30 32 ·Thu, 20 Jan 202
0190	32 20 31 35 3a 34 38 3a 32 30 20 47 4d 2 15:48: 20 GM

d. Divide the number of characters displayed in the screen (question 1.a) by the number of bytes included as the transport layer protocol payload of the HTTP response (question 1.d).

29/359 is equal to about 0.08, which means that the message text is only 8% of the used TCP payload byte size.

This is because TCP payload containing the response information also includes different information about the request response, for example: response code, HTTP version, date of response, charset, cookie info and other.

CoAP analysis

a. What is the response displayed in the Linux terminal? How many characters were shown (counting spaces and punctuation marks)?

The response is:

welcome to the ETSI plugtest! last change: 2022-01-20 15:31:28 UTC

Which contains in total 66 characters.

b. How many bytes does the payload of the transport layer protocol need to encode the CoAP request? How many for the CoAP response?

For the request, the UDP payload contains 17 bytes:

UDP payload (17 bytes)		
> Constrained Application Protocol, Confirmable, GET, MID:63185		
0000	52 54 00 12 35 02 08 00 27 3f 8c d0 08 00 45 00	RT..5... '?...E..
0010	00 2d 00 54 40 00 40 11 cd e4 0a 00 02 0f 86 66	...T@.@...f...
0020	da 12 bd 98 16 33 00 19 6c b2 40 01 f6 d1 37 633...l.@...7c
0030	6f 61 70 2e 6d 65 84 74 65 73 74	oap.me.t est

The response has 81 bytes in the UDP payload:

UDP payload (81 bytes)		
> Constrained Application Protocol, Acknowledgement, 2.05 Content, MID:63185		
0000	08 00 27 3f 8c d0 52 54 00 12 35 02 08 00 45 00	.. '?..RT..5...E..
0010	00 6d 04 7a 00 00 40 11 09 7f 86 66 da 12 0a 00	..m.z...@...f....
0020	02 0f 16 33 bd 98 00 59 a5 4e 60 45 f6 d1 48 41	...3...Y.N`E..HA
0030	fb ca 94 12 6f f0 90 80 ff 77 65 6c 63 6f 6d 65o... welcome
0040	20 74 6f 20 74 68 65 20 45 54 53 49 20 70 6c 75	to the ETSI plu
0050	67 74 65 73 74 21 20 6c 61 73 74 20 63 68 61 6e	gtest! l ast chan
0060	67 65 3a 20 32 30 32 32 2d 30 31 2d 32 30 20 31	ge: 2022 -01-20 1
0070	35 3a 33 31 3a 32 38 20 55 54 43	5:31:28 UTC

c. Divide the number of characters displayed in the screen (question 2.a) by the number of bytes included as the transport layer protocol payload of the HTTP response (question 2.b).

$66/81 = 0.815$. Which shows that the utilization of the UDP window is about 80%, i.e. 80% of the payload size is the data we are interested in.

d. Considering your answers from questions 1.c. and 2.b., which protocol is more efficient in terms of encoding? Why?

In terms of efficiency, CoAP is definitely better. 80% of the payload size is the data, whereas for the HTTP it is only roughly 8%. This is because CoAP features way less information in the response than HTTP. Furthermore, relevant information is stored as codes rather than full text like in HTTP.

Closing remarks - CoAP vs HTTP

CoAP in general is way more efficient than HTTP. It should be noted though, that CoAP is less reliable than HTTP, because it uses UDP. CoAP in general is very good for devices that have constraints such as limited power, memory and other resources. These kinds of devices are present in the IoT field, so using CoAP is very important in order to save mentioned resources.

One very important feature of CoAP is that it supports multicast, which is important in cases where we need to manage multiple IoT devices. HTTP does not support multicast.

In no way it is recommended though to use CoAP to communicate between e.g. laptops, PCs and servers, as the protocol is less reliable than HTTP and has less options in the overhead.