Department of Electronic and Telecommunication Engineering
University of Moratuwa
Sri Lanka

EN3251 - Internet of Things

Laboratory Exercise 3
The Constrained Application Protocol (CoAP)

Group 9

Name	Index Number
M. P. Wickramarathne	210703V
A.A.W.L.R.Amarasinghe	210031H
A.D.T. Dabare	210089P

This report is submitted as a partial fulfillment of module EN3251
2024.10.19

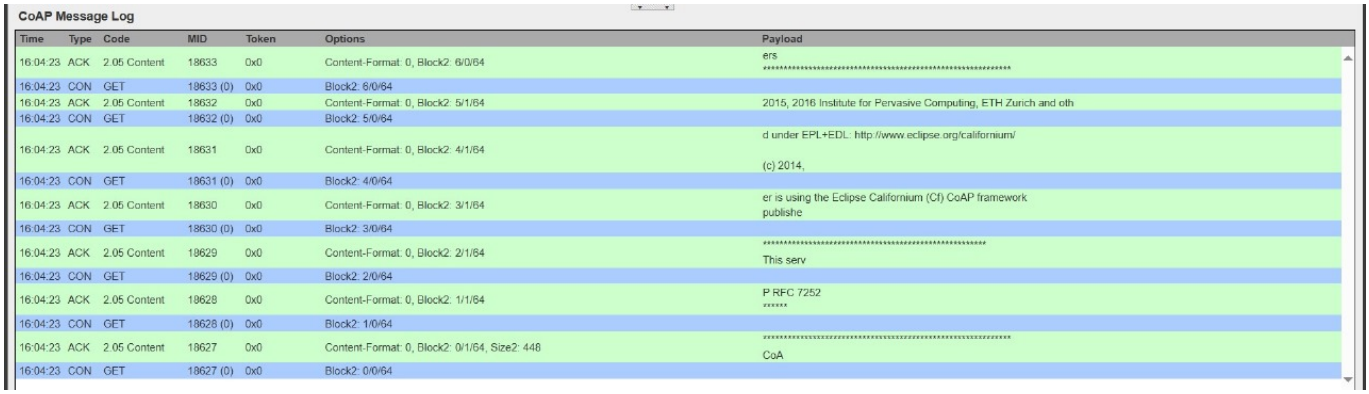
Contents

1 Task 1 - Configuring the CoAP Client

In **Task 1**, we configured the CoAP client by installing and setting up the Copper4Cr client for use with Microsoft Edge. After enabling Developer Mode and loading the Copper4Cr app and extension into the browser, we tested the client by sending requests to a public CoAP test server (`coap://coap.me:5683`). We used the Discover feature to list available resources on the server, confirming that the client was successfully communicating with the CoAP server.

2 Task 2 - Setting up and Configuring a CoAP Server

2.1 CoAP Message Log



CoAP Message Log

Time	Type	Code	MID	Token	Options	Payload
16:04:23	ACK	2.05 Content	18633	0x0	Content-Format: 0, Block2: 6/0/64	ers **********
16:04:23	CON	GET	18633 (0)	0x0	Block2: 6/0/64	
16:04:23	ACK	2.05 Content	18632	0x0	Content-Format: 0, Block2: 5/1/64	2015, 2016 Institute for Pervasive Computing, ETH Zurich and oth
16:04:23	CON	GET	18632 (0)	0x0	Block2: 5/0/64	
16:04:23	ACK	2.05 Content	18631	0x0	Content-Format: 0, Block2: 4/1/64	d under EPL+EDL: http://www.eclipse.org/californium/ (c) 2014,
16:04:23	CON	GET	18631 (0)	0x0	Block2: 4/0/64	
16:04:23	ACK	2.05 Content	18630	0x0	Content-Format: 0, Block2: 3/1/64	er is using the Eclipse Californium (Cf) CoAP framework publishe
16:04:23	CON	GET	18630 (0)	0x0	Block2: 3/0/64	
16:04:23	ACK	2.05 Content	18629	0x0	Content-Format: 0, Block2: 2/1/64	********** This serv
16:04:23	CON	GET	18629 (0)	0x0	Block2: 2/0/64	
16:04:23	ACK	2.05 Content	18628	0x0	Content-Format: 0, Block2: 1/1/64	P RFC 7252 ******
16:04:23	CON	GET	18628 (0)	0x0	Block2: 1/0/64	
16:04:23	ACK	2.05 Content	18627	0x0	Content-Format: 0, Block2: 0/1/64, Size2: 448	********** CoA
16:04:23	CON	GET	18627 (0)	0x0	Block2: 0/0/64	

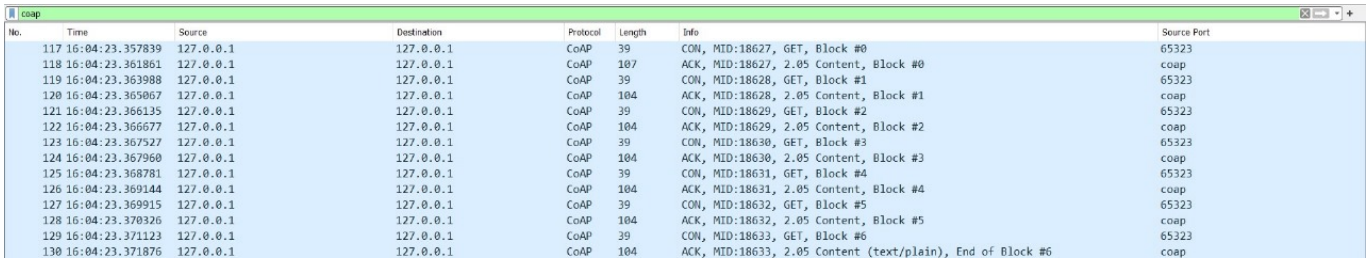
Figure 1: Task 2.4

This image displays the internal logs of the CoAP server during the communication.

- **CoAP GET Requests**:
  - The client sends multiple `GET` requests to the server. These requests are associated with different block numbers, such as `Block2: 0/0/64`, `Block2: 1/1/64`, and so on.
  - These GET requests are marked as `CON` (Confirmable Message) in the "Type" column, and the "Code" column shows `GET`.
  - Each request is identified by a unique `MID` (Message ID), for instance, `18633`, `18632`, etc.
  - The `Options` column shows the `Block2` transfer mechanism, which is used for transferring large payloads in CoAP by dividing the payload into smaller blocks.

- **Server Responses (ACK and Content)**:
  - After receiving each `GET` request, the server sends an acknowledgment (`ACK`) followed by a `2.05 Content` response. This is the server's way of confirming the request and returning part of the requested content.
  - The `Block2` field in the responses shows which block of the content is being returned. For example, `Block2: 0/0/64` is the first block, followed by `Block2: 1/1/64`, and so on.
  - The content is transmitted block by block until the full payload is delivered to the client.

2.2 Wireshark Capture



coap

No.	Time	Source	Destination	Protocol	Length	Info	Source Port
117	16:04:23.357839	127.0.0.1	127.0.0.1	CoAP	39	CON, MID:18627, GET, Block #0	65323
118	16:04:23.361861	127.0.0.1	127.0.0.1	CoAP	107	ACK, MID:18627, 2.05 Content, Block #0	coap
119	16:04:23.363988	127.0.0.1	127.0.0.1	CoAP	39	CON, MID:18628, GET, Block #1	65323
120	16:04:23.365067	127.0.0.1	127.0.0.1	CoAP	104	ACK, MID:18628, 2.05 Content, Block #1	coap
121	16:04:23.366135	127.0.0.1	127.0.0.1	CoAP	39	CON, MID:18629, GET, Block #2	65323
122	16:04:23.366677	127.0.0.1	127.0.0.1	CoAP	104	ACK, MID:18629, 2.05 Content, Block #2	coap
123	16:04:23.367527	127.0.0.1	127.0.0.1	CoAP	39	CON, MID:18630, GET, Block #3	65323
124	16:04:23.367960	127.0.0.1	127.0.0.1	CoAP	104	ACK, MID:18630, 2.05 Content, Block #3	coap
125	16:04:23.368781	127.0.0.1	127.0.0.1	CoAP	39	CON, MID:18631, GET, Block #4	65323
126	16:04:23.369144	127.0.0.1	127.0.0.1	CoAP	104	ACK, MID:18631, 2.05 Content, Block #4	coap
127	16:04:23.369915	127.0.0.1	127.0.0.1	CoAP	39	CON, MID:18632, GET, Block #5	65323
128	16:04:23.370326	127.0.0.1	127.0.0.1	CoAP	104	ACK, MID:18632, 2.05 Content, Block #5	coap
129	16:04:23.371123	127.0.0.1	127.0.0.1	CoAP	39	CON, MID:18633, GET, Block #6	65323
130	16:04:23.371876	127.0.0.1	127.0.0.1	CoAP	104	ACK, MID:18633, 2.05 Content (text/plain), End of Block #6	coap

Figure 2: Task 2.4 - Wireshark

This image shows the network traffic captured using Wireshark. Each row corresponds to a packet sent between the client and the server.

- **GET Requests**:
  - The rows labeled `CON, MID:..., GET, Block #x` represent the client's `GET` requests for each block. For example, at `16:04:23.357839`, the client sends a `GET` request (Message ID `18627`) for `Block #0`.
  - The block numbers progress incrementally, as shown in the message log (e.g., `Block #0`, `Block #1`, etc.), which matches the block numbers seen in the CoAP message log.
- **ACK and Content Responses**:
  - The server responds to each `GET` request with an `ACK` followed by a `2.05 Content` message, returning part of the payload. This is shown as `2.05 Content, Block #x` in the "Info" column.
  - The `MID` in each response matches the `MID` from the corresponding `GET` request. For instance, the `GET` request with `MID 18627` (Block #0) receives an `ACK` followed by a `2.05 Content` response for Block #0, just like in the CoAP message log.

3 Task 3 - Add Resources to CoAp Server

3.1 GET Hello World in Copper CoAP Client

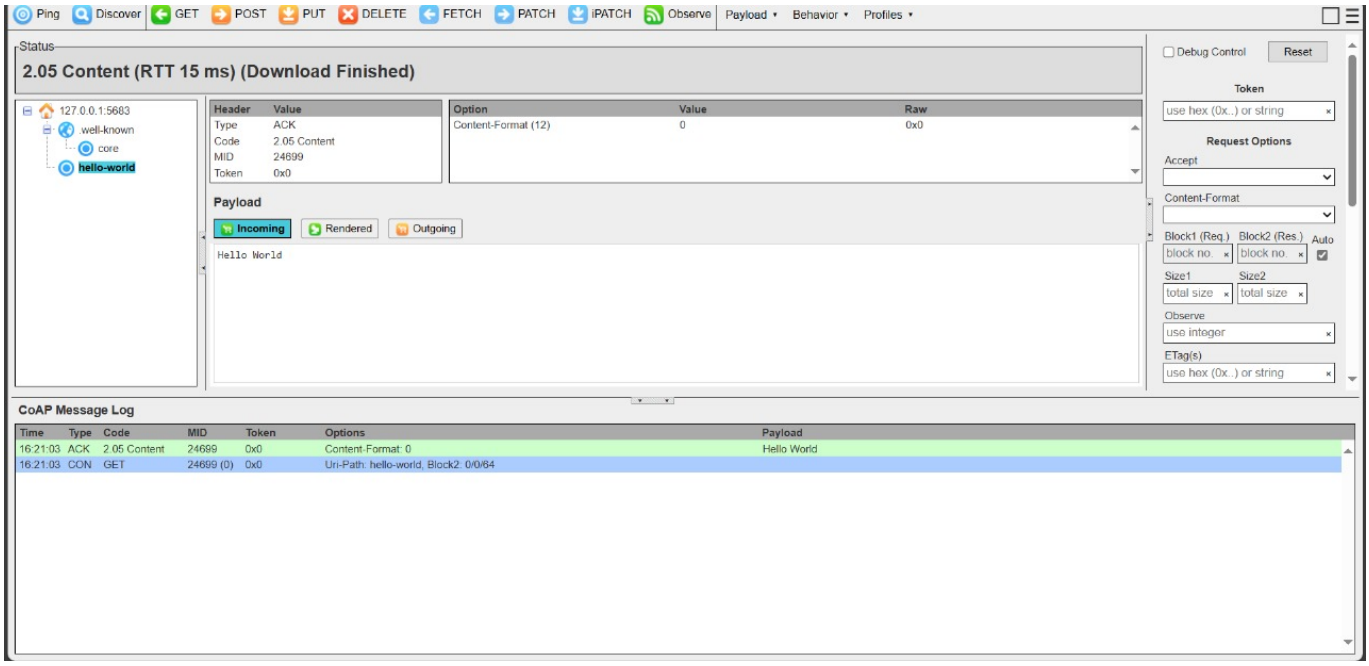


Figure 3: Task 3.4 - Get Hello World

This image shows the interaction using a CoAP client interface (Cu4Cr Extension).

- **GET Request**:
  - The client sends a `GET` request for the resource located at the URI path `/hello-world`.
  - The CoAP message log shows the `MID` (Message ID) of the request as `24699`.
  - The client requests the first block of content using the `Block2: 0/0/64` option, which indicates block-wise transfer is enabled and this is the first block of the content.
- **Server Response (ACK and Content)**:
  - The server responds with a `2.05 Content` message, acknowledging (`ACK`) the request.
  - The response includes the payload "Hello World", visible in both the payload section and the log.
  - The `Content-Format` option indicates that the payload is in plain text (`Content-Format: 0`).

3.2 Wireshark Capture of the Communication

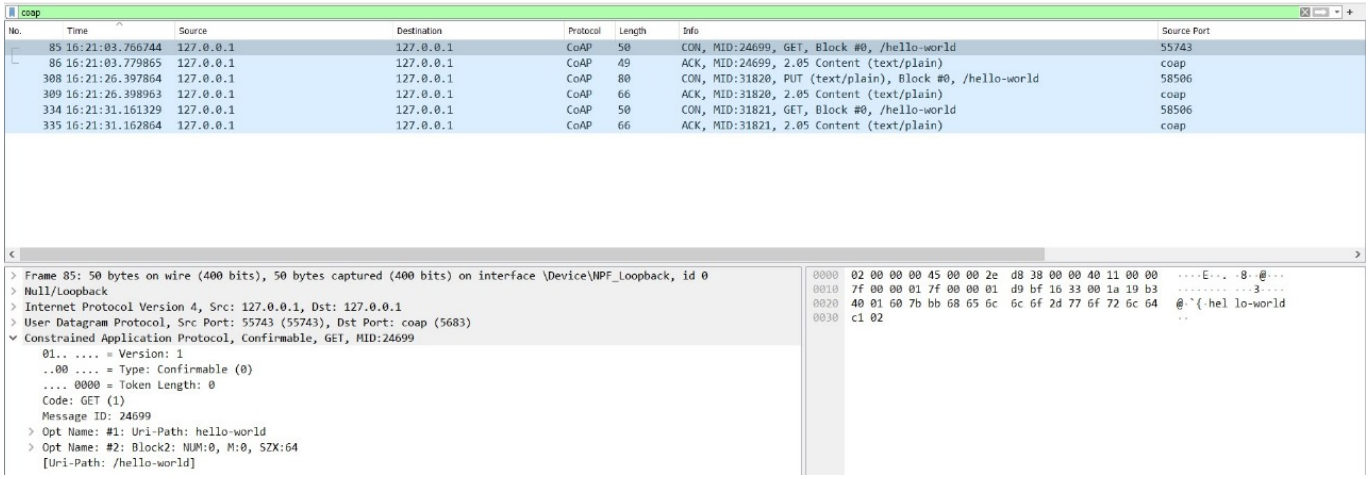


Figure 4: Task 3.4 - Get Hello World - Wireshark

This image shows the network traffic capture of the CoAP exchange using Wireshark.

- **GET Request**:
  - The first row shows the client sending a `GET` request to the server for the `/hello-world` URI.
  - The `MID` of the request is `24699`, matching the one in the CoAP client.
  - The `Block2: 0/0/64` option indicates this is the first block of a potentially large payload.
- **ACK and Content Response**:
  - The second row shows the server responding with an `ACK` to the `GET` request.
  - The response includes the `2.05 Content` message containing the "Hello World" payload.
  - The `MID` of the response matches the `MID` of the GET request (`24699`), indicating that the response is for the correct request.

3.3 PUT Request in Copper CoAP Client

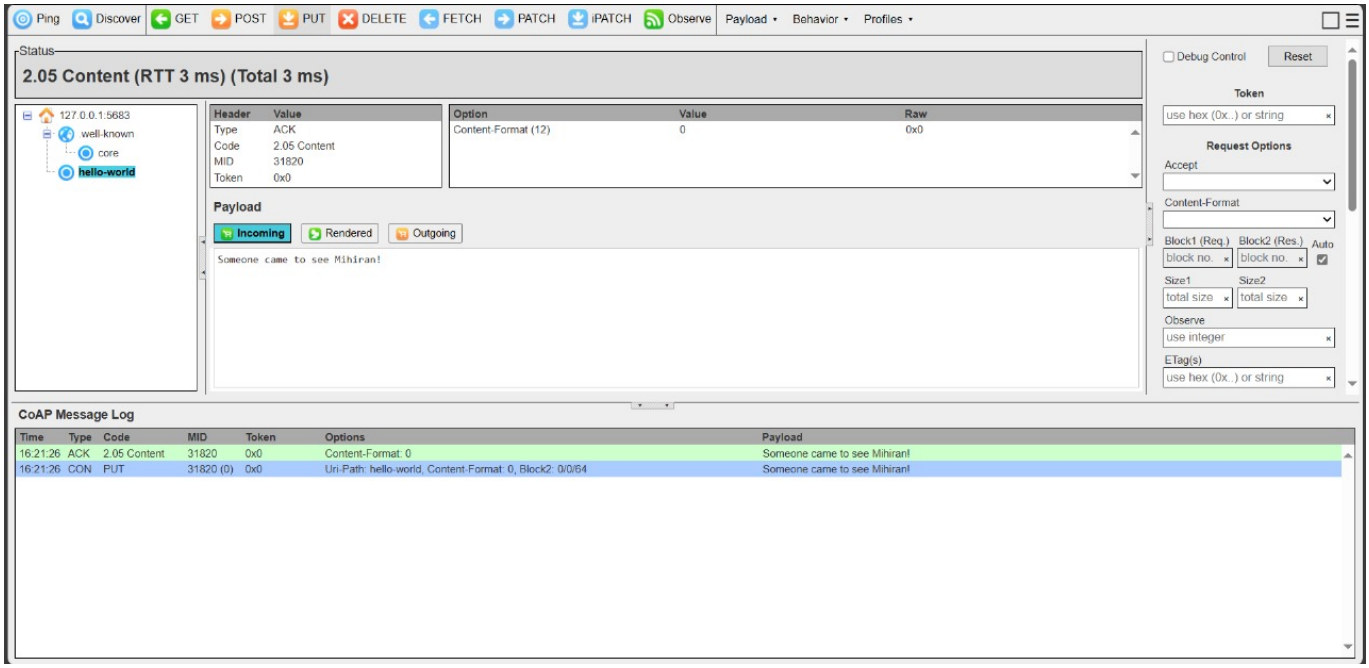


Figure 5: Task 3.4 - Put New Message

This image shows a `PUT` request being sent using the Copper CoAP client.

- **PUT Request**:
  - The client sends a `PUT` request to update the resource located at the URI path `/hello-world`.
  - The message log shows that the `MID` (Message ID) of the request is `31820`.
  - The content in the `Payload` section is "Someone came to see Mihiran!".
  - The `Block2: 0/0/64` option indicates that block-wise transfer is possible, but the message fits into a single block (28 bytes).
  - The `Content-Format` is plain text (`Content-Format: 0`).
- **Server Response (ACK)**:
  - The server responds to the `PUT` request with a `2.05 Content` message, acknowledging that the update has been applied.
  - The message log shows the same payload content in both the request and the response.

3.4 Wireshark Capture of PUT Request

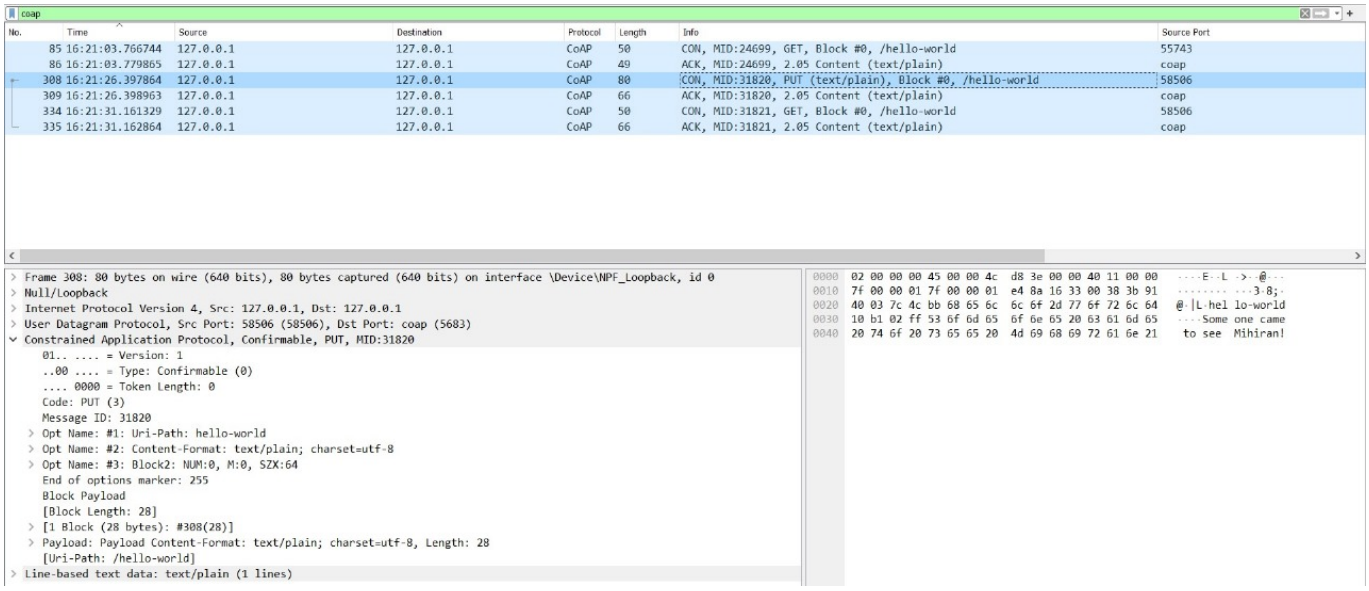


Figure 6: Task 3.4 - Put New Message Wireshark

This image shows the network traffic capture of the `PUT` request and response using Wireshark.

- **PUT Request**:
  - The row labeled `CON, MID:31820, PUT (text/plain), Block #0, /hello-world` shows the client sending a `PUT` request to update the resource at `/hello-world`.
  - The `MID` of the `PUT` request is `31820`, which matches the `MID` shown in the CoAP client.
  - The payload "Someone came to see Mihiran!" is visible in both the data portion and detailed breakdown of the frame.
  - The content format is identified as plain text (`text/plain; charset=utf-8`), and the payload length is 28 bytes.
- **Server Acknowledgment (ACK)**:
  - The subsequent row shows the server's acknowledgment (`ACK, MID:31820, 2.05 Content (text/plain)`), confirming the server received and processed the `PUT` request.
  - The `MID` of the acknowledgment matches the `MID` of the `PUT` request, ensuring that the response corresponds to the correct request.

3.5 CoAP Client Interface - GET Request

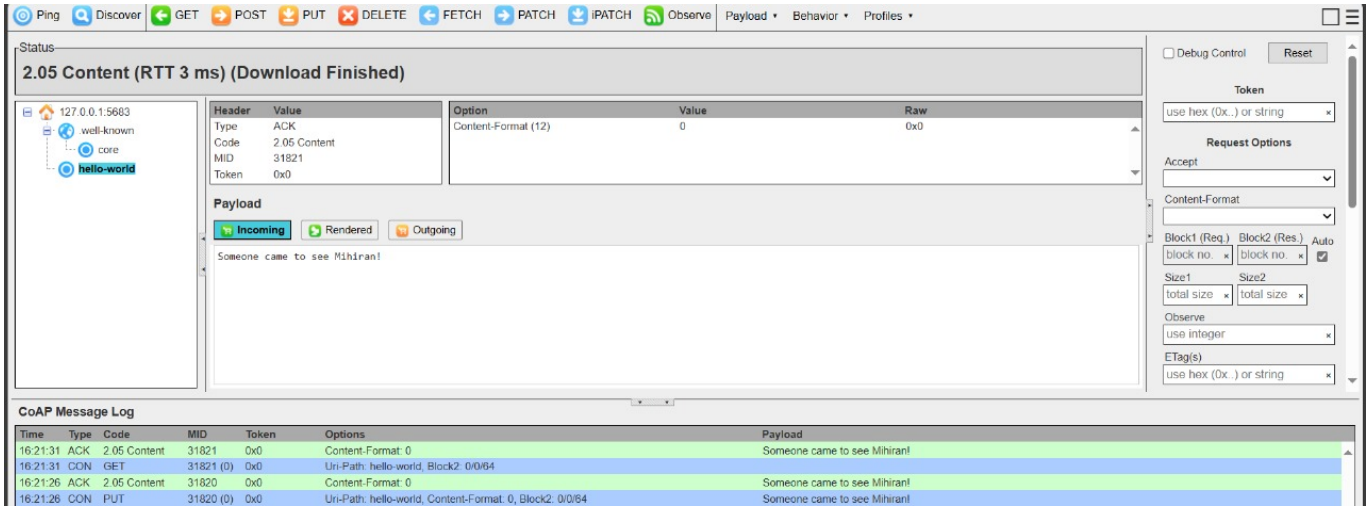
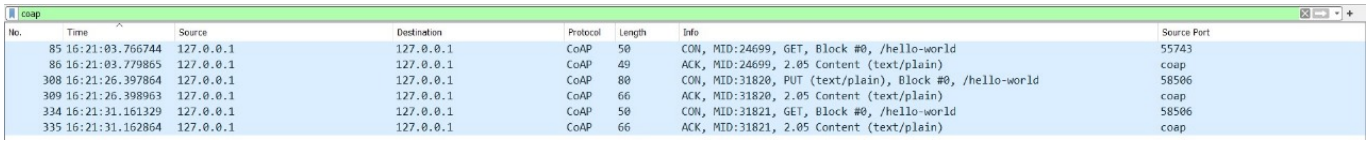


Figure 7: Task 3.4 - Get the Updated Value

This image shows the CoAP client (Cu4Cr Extension) interacting with the CoAP server.

- **GET Request**:
  - The client sends a `GET` request to retrieve the resource located at the URI path `/hello-world`.
  - The message log shows that the `GET` request has a `MID` (Message ID) of `31821`.
  - The request uses block-wise transfer (`Block2: 0/0/64`), but since the resource is small, it fits into a single block.
- **Server Response**:
  - The server responds with an `ACK` message and sends the content as a `2.05 Content` response.
  - The payload contains the updated message: *"Someone came to see Mihiran!"*, which was previously updated via a `PUT` request.
  - The content of the resource is visible in both the "Payload" section and the CoAP Message Log.

3.6 Wireshark Capture of the GET Request



coap

No.	Time	Source	Destination	Protocol	Length	Info	Source Port
85	16:21:03.766744	127.0.0.1	127.0.0.1	CoAP	50	CON, MID:24699, GET, Block #0, /hello-world	55743
86	16:21:03.779865	127.0.0.1	127.0.0.1	CoAP	49	ACK, MID:24699, 2.05 Content (text/plain)	coap
308	16:21:26.397864	127.0.0.1	127.0.0.1	CoAP	80	CON, MID:31820, PUT (text/plain), Block #0, /hello-world	58506
309	16:21:26.398963	127.0.0.1	127.0.0.1	CoAP	66	ACK, MID:31820, 2.05 Content (text/plain)	coap
334	16:21:31.161329	127.0.0.1	127.0.0.1	CoAP	50	CON, MID:31821, GET, Block #0, /hello-world	58506
335	16:21:31.162864	127.0.0.1	127.0.0.1	CoAP	66	ACK, MID:31821, 2.05 Content (text/plain)	coap

Figure 8: Task 3.4 - Get the Updated Value Wireshark

This image shows the Wireshark network capture of the CoAP communication during the retrieval of the updated message.

- **GET Request**:
  - The capture shows the client sending a `GET` request to the server for the `/hello-world` resource, with `MID: 31821`.
  - The request asks for block #0 of the content, and since the content is small, it is transmitted in a single block.
- **ACK and Content Response**:
  - The server sends back an `ACK` message with a `2.05 Content` response, similar to what is shown in the CoAP client interface.
  - The response contains the updated payload *"Someone came to see Mihiran!"*, which matches the content seen in the CoAP client interface.

4 Codes

4.1 Tutorial Server Code

```java
package coap_server;

import org.eclipse.californium.core.CoapServer;

public class TutorialServer extends CoapServer {
    public static void main(String[] args) {
        TutorialServer tutorialServer = new TutorialServer();
        HelloWorldResource hello = new HelloWorldResource("hello-world");
        tutorialServer.add(hello);
        tutorialServer.start();
    }
}
```

Listing 1: Tutorial Server Code

4.2 HelloWorldResource Code

```java
package coap_server;
import org.eclipse.californium.core.CoapResource;
import org.eclipse.californium.core.server.resources.CoapExchange;

public class HelloWorldResource extends CoapResource {
    public String content = "Hello World";

    public HelloWorldResource(String name) {
        super(name);
    }

    @Override
    public void handleGET(CoapExchange exchange) {
        exchange.respond(content);
    }

    @Override
    public void handlePUT(CoapExchange exchange){
        byte[] payload = exchange.getRequestPayload();

        try {
            content = new String(payload, "UTF-8");
            exchange.respond(content);
        } catch (Exception e){
            e.printStackTrace();
            exchange.respond("Invalid String");
        }
    }
}
```

Listing 2: HelloWorldResource Code