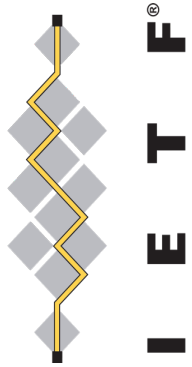
CoAP Tutorial

as defined in **RFC 7252**!

Georgios Z. PAPADOPOULOS, PhD, HDR
Professor at IMT Atlantique, campus of Rennes, France

THE 6TISCH PROTOCOL STACK



Applications	CoJP		
CoAP / OSCORE	6LoWPAN ND	RPL	
UDP	ICMPv6		
IPv6			
6LoWPAN HC / 6LoRH HC	Scheduling Function (MSF)		
6top inc. 6top Protocol (6p)			
IEEE Std 802.15.4 TSCH			
IEEE Std 802.15.4 PHY			

CONTEXT

- **Constrained Application Protocol (CoAP)**
- **RFC 7252**
- **A Web Transfer Protocol for Internet of Things**

In This Tutorial

1. CoAP Features
2. CoAP Terminology
3. CoAP Architecture
4. CoAP Message Format
5. CoAP Type of Messages
6. CoAP Request/Response Model

1. CoAP Features

Click me!

as defined in **RFC 7252**!

What CoAP is (and What is not)

- CoAP **is**:
  - An efficient RESTful protocol.
  - Suitable for constrained devices and networks.
  - Designed for M2M and IoT applications.
  - Proxy to/from HTTP.

What CoAP is (and What is not)

- CoAP **is**:
  - An efficient RESTful protocol.
  - Suitable for constrained devices and networks.
  - Designed for M2M and IoT applications.
  - Proxy to/from HTTP.

- CoAP **is not**:
  - A replacement for HTTP.
  - HTTP compression mechanism.
  - Restricted to “automation” networks.

CoAP Features

CoAP Features

1. Open IETF Standard.

CoAP Features

1. Open IETF Standard.
2. Embedded web transfer protocol, i.e., **coap://** or **coaps://**
   - Ports: 5683 and 5684.

CoAP has a *scheme*, i.e., **coap://**, or with Security which is **coaps://**.
These schemes come with default ports, the 5683, and 5684, respectively.

CoAP Features

1. Open IETF Standard.
2. Embedded web transfer protocol, i.e., **coap://** or **coaps://**
   - Ports: 5683 and 5684.
3. Strong DTLS security:
   - Modes: NoSec, PSK, RPK, and Certificate.

CoAP includes strong security built into the protocol using
DTLS based on the following modes Pre-Shared Keys (PSK), Raw Public Key (RPK) and Certicate.

CoAP Features

1. Open IETF Standard.
2. Embedded web transfer protocol, i.e., **coap://** or **coaps://**
   - Ports: 5683 and 5684.
3. Strong DTLS security:
   - Modes: NoSec, PSK, RPK, and Certificate.
4. UDP and TCP support:
   - UDP binding with *optional* reliability and multicast support.

CoAP Features

1. Open IETF Standard.
2. Embedded web transfer protocol, i.e., **coap://** or **coaps://**
   - Ports: 5683 and 5684.
3. Strong DTLS security:
   - Modes: NoSec, PSK, RPK, and Certificate.
4. UDP and TCP support:
   - UDP binding with *optional* reliability and multicast support.
5. Asynchronous message exchanges.

CoAP Features

1. Open IETF Standard.
2. Embedded web transfer protocol, i.e., **coap://** or **coaps://**
   - Ports: 5683 and 5684.
3. Strong DTLS security:
   - Modes: NoSec, PSK, RPK, and Certificate.
4. UDP and TCP support:
   - UDP binding with *optional* reliability and multicast support.
5. Asynchronous message exchanges.
6. Built-in discovery mechanism.
7. GET, POST, PUT, DELETE methods.
8. URI and content-type support.
9. Compact 4-byte Header.

CoAP Features

1. Open IETF Standard.
2. Embedded web transfer protocol, i.e., **coap://** or **coaps://**
   - Ports: 5683 and 5684.
3. Strong DTLS security:
   - Modes: NoSec, PSK, RPK, and Certificate.
4. UDP and TCP support:
   - UDP binding with *optional* reliability and multicast support.
5. Asynchronous message exchanges.
6. Built-in discovery mechanism.
7. GET, POST, PUT, DELETE methods.
8. URI and content-type support.
9. Compact 4-byte Header.
10. Observation (i.e., subscription) and Block-wise transfer.

2. CoAP Terminology

Click me!

as defined in **RFC 7252**!

TERMINOLOGY

Endpoint	• An Entity participating in the CoAP Protocol.

TERMINOLOGY

Endpoint	• An Entity participating in the CoAP Protocol.
Sender	• The Originating Endpoint of a Message.

TERMINOLOGY

Endpoint	• An Entity participating in the CoAP Protocol.
Sender	• The Originating Endpoint of a Message.
Recipient	• The Destination Endpoint of a Message.

TERMINOLOGY

Endpoint	• An Entity participating in the CoAP Protocol.
Sender	• The Originating Endpoint of a Message.
Recipient	• The Destination Endpoint of a Message.
Client	• The Originating Endpoint of a Request.

TERMINOLOGY

Endpoint	• An Entity participating in the CoAP Protocol.
Sender	• The Originating Endpoint of a Message.
Recipient	• The Destination Endpoint of a Message.
Client	• The Originating Endpoint of a Request.
Server	• The Destination Endpoint of a Request.

TERMINOLOGY

Endpoint	• An Entity participating in the CoAP Protocol.
Sender	• The Originating Endpoint of a Message.
Recipient	• The Destination Endpoint of a Message.
Client	• The Originating Endpoint of a Request.
Server	• The Destination Endpoint of a Request.
Origin Server	• The Server on which a given Resource resides.

TERMINOLOGY

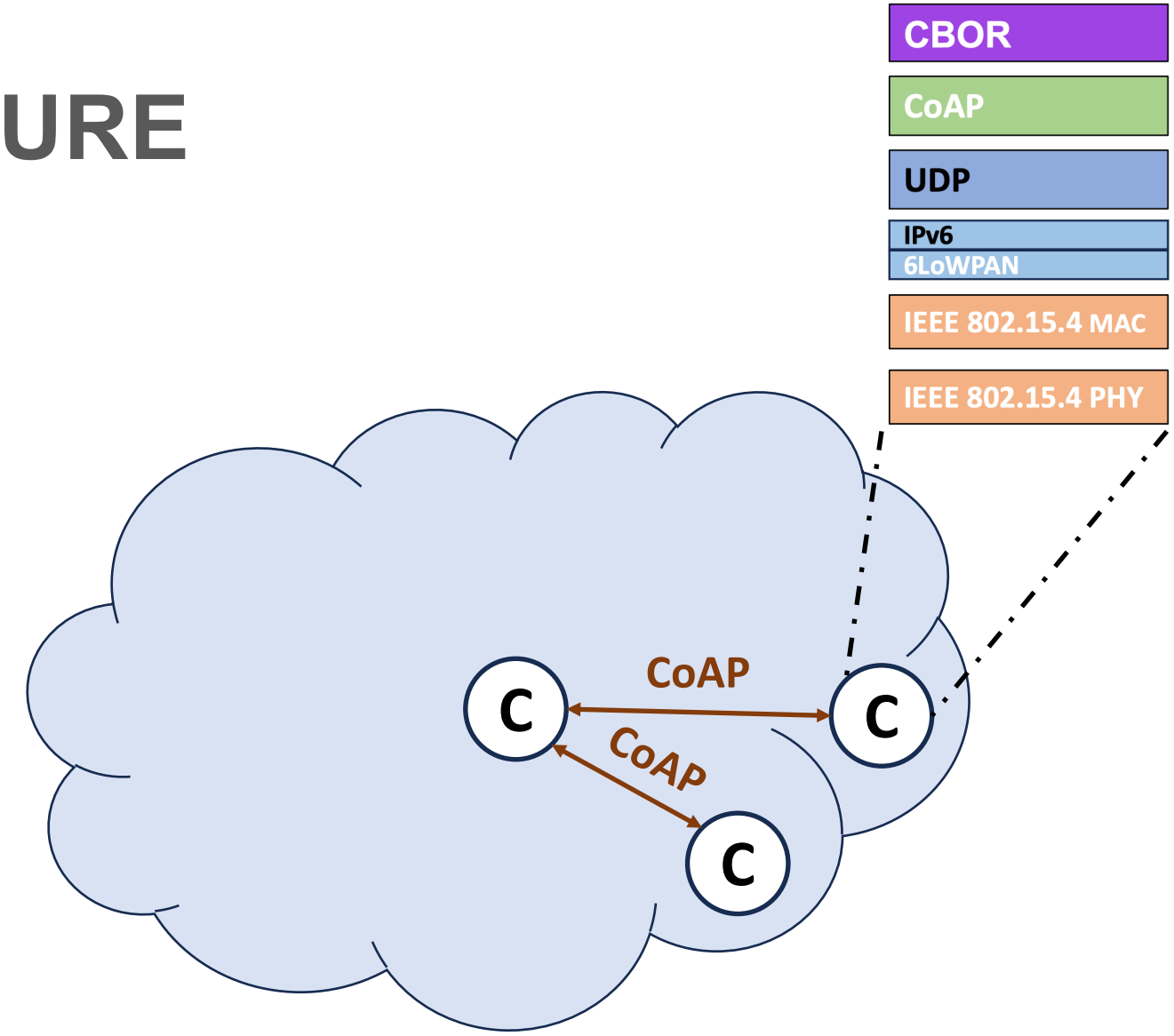
Endpoint	• An Entity participating in the CoAP Protocol.
Sender	• The Originating Endpoint of a Message.
Recipient	• The Destination Endpoint of a Message.
Client	• The Originating Endpoint of a Request.
Server	• The Destination Endpoint of a Request.
Origin Server	• The Server on which a given Resource resides.
Intermediary	• A CoAP Endpoint that acts both as a Server and as a Client.

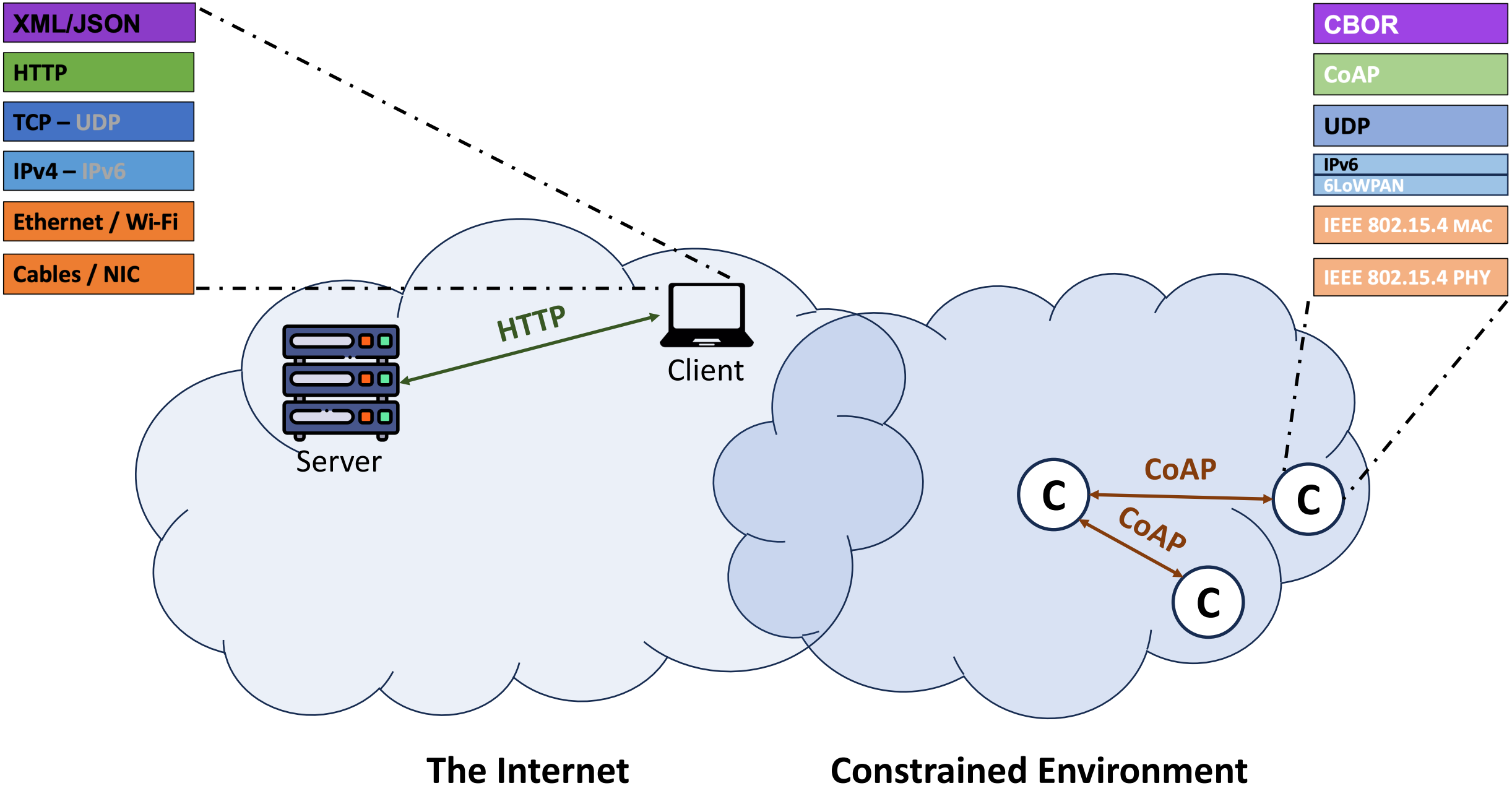
3. CoAP Architecture

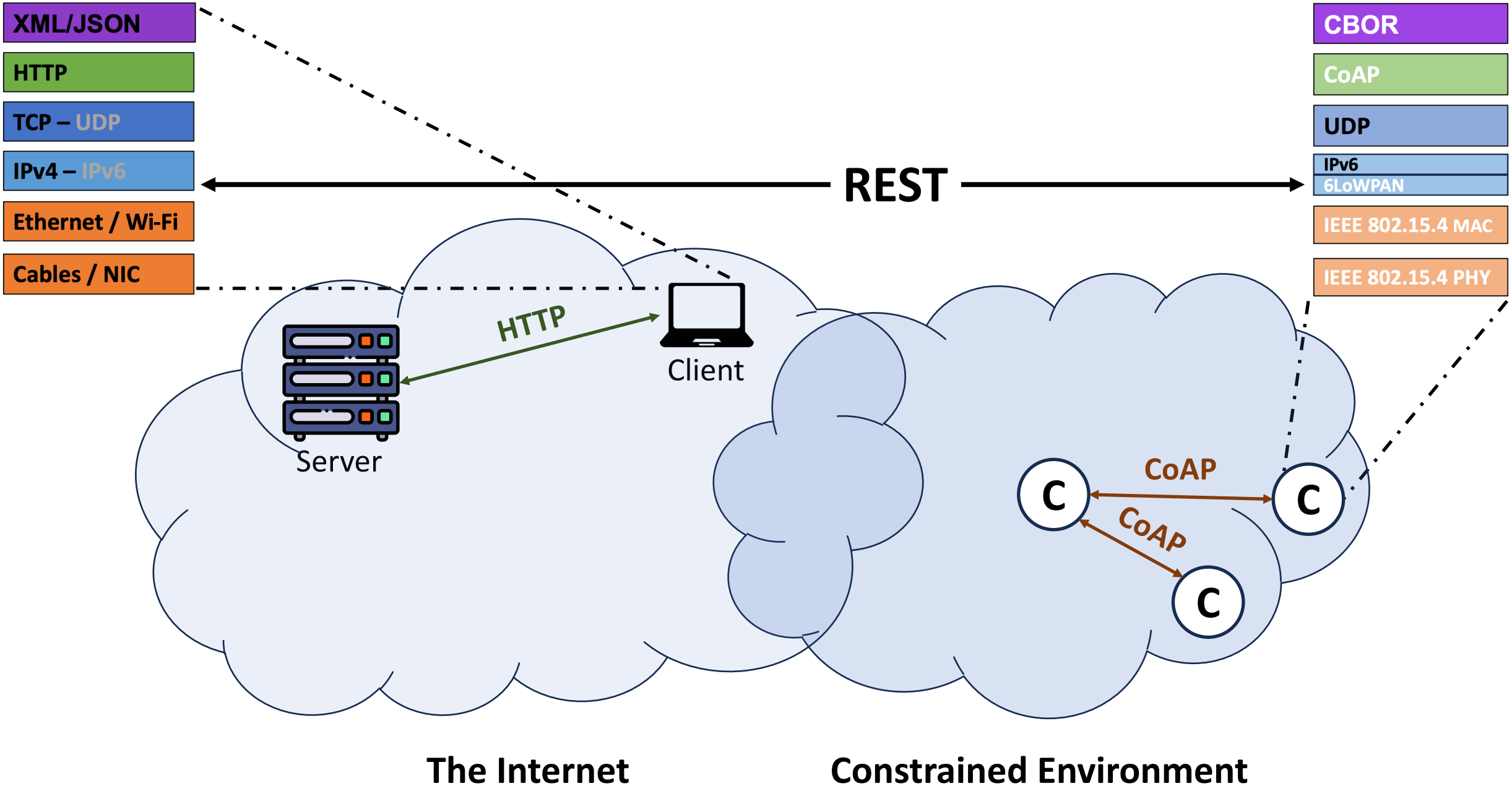
Click me!

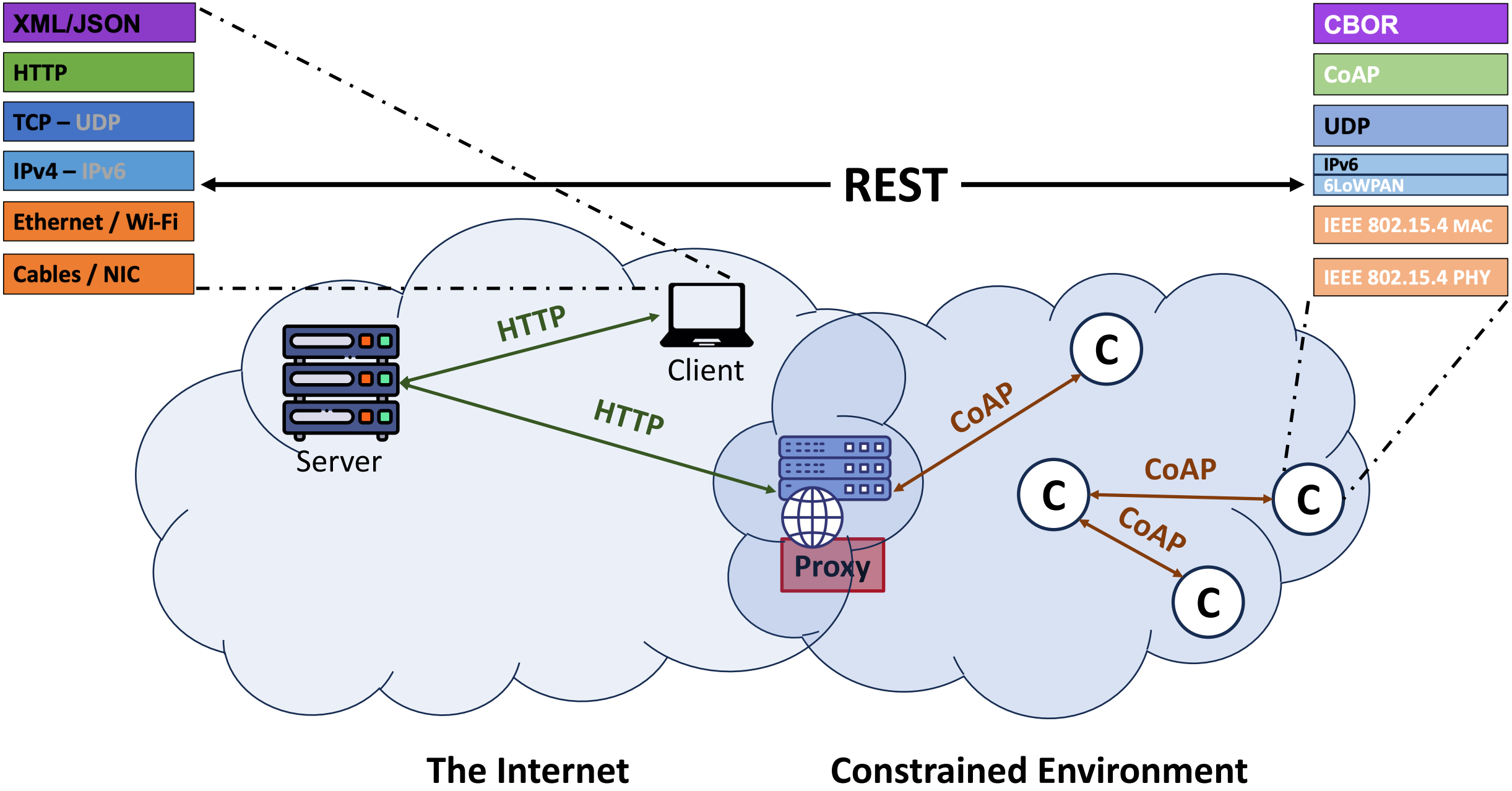
as defined in **RFC 7252**!

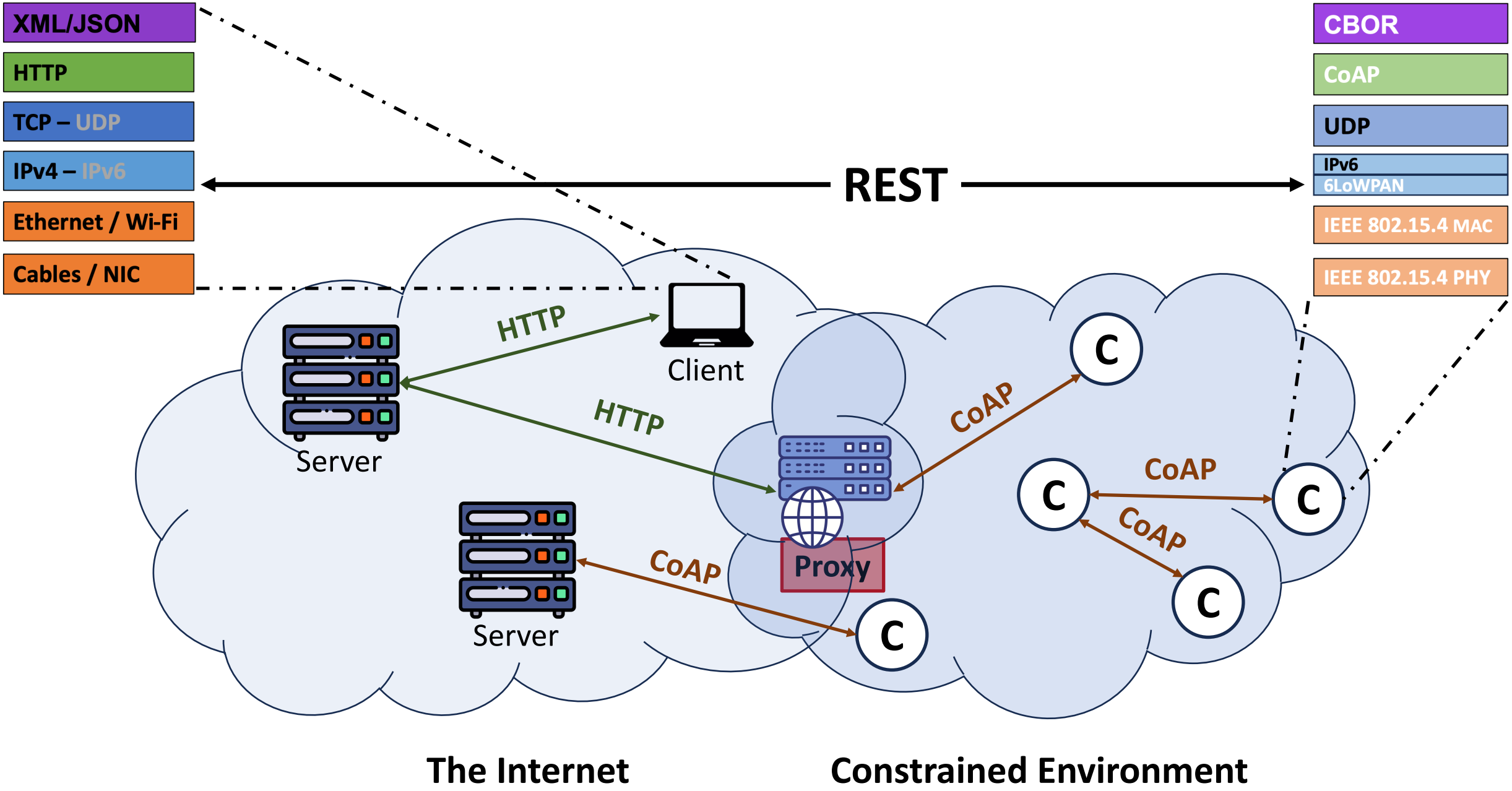
CoAP Architecture

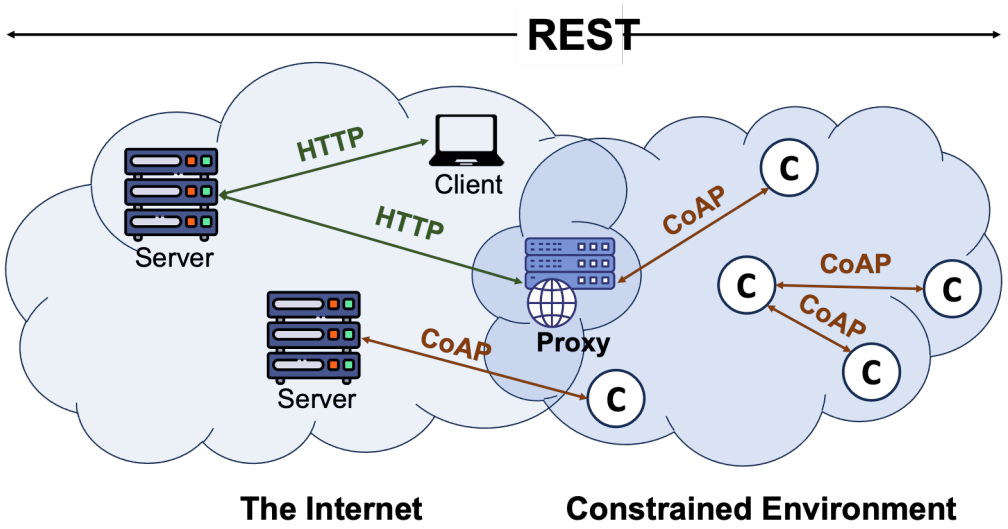


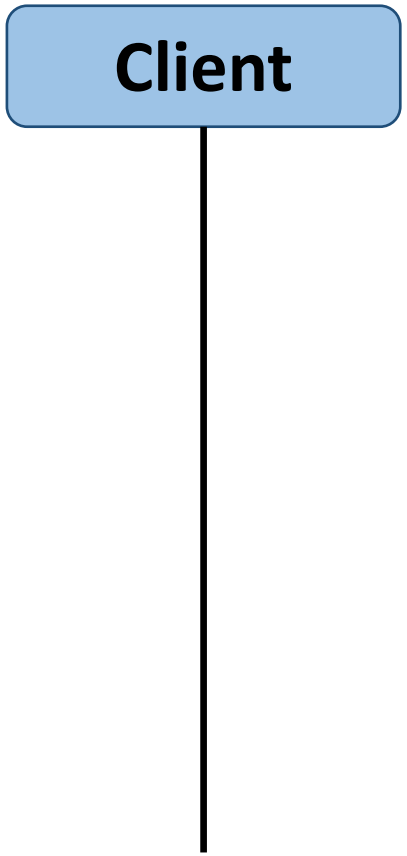

XML/JSON
HTTP
TCP – UDP
IPv4 – IPv6
Ethernet / Wi-Fi
Cables / NIC
CBOR
CoAP
UDP
IPv6
6LoWPAN
IEEE 802.15.4 MAC
IEEE 802.15.4 PHY
HTTP
Client
Server
CoAP
CoAP
C
C
C
The Internet
Constrained Environment

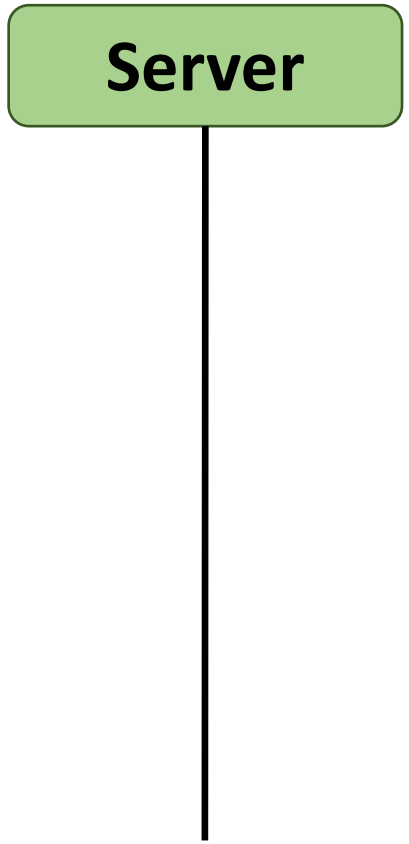

XML/JSON
HTTP
TCP – UDP
IPv4 – IPv6
Ethernet / Wi-Fi
Cables / NIC
REST
CBOR
CoAP
UDP
IPv6
6LoWPAN
IEEE 802.15.4 MAC
IEEE 802.15.4 PHY
HTTP
Client
Server
CoAP
CoAP
C
C
C
The Internet
Constrained Environment


XML/JSON
HTTP
TCP – UDP
IPv4 – IPv6
Ethernet / Wi-Fi
Cables / NIC
REST
CBOR
CoAP
UDP
IPv6
6LoWPAN
IEEE 802.15.4 MAC
IEEE 802.15.4 PHY
HTTP
HTTP
Client
Server
Proxy
CoAP
CoAP
CoAP
C
C
C
C
The Internet
Constrained Environment

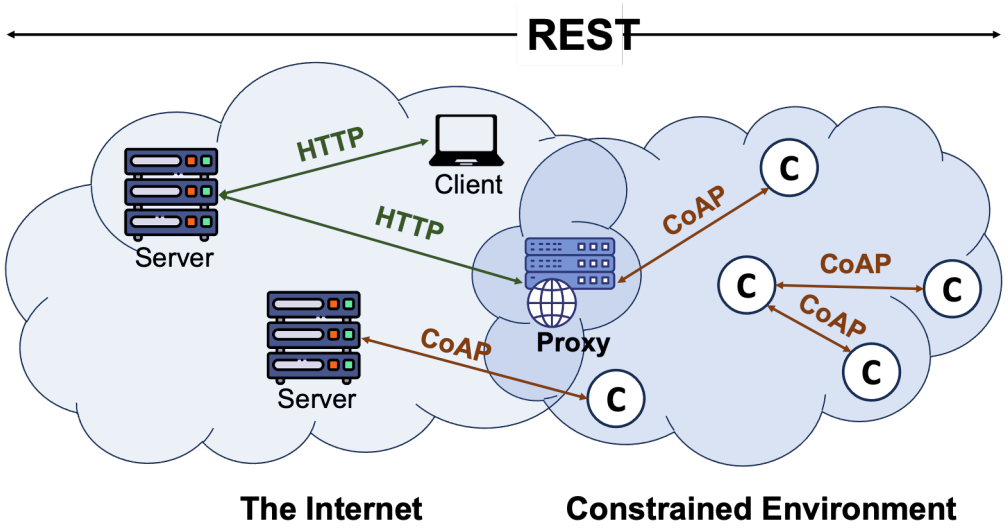

XML/JSON
HTTP
TCP – UDP
IPv4 – IPv6
Ethernet / Wi-Fi
Cables / NIC
REST
CBOR
CoAP
UDP
IPv6
6LoWPAN
IEEE 802.15.4 MAC
IEEE 802.15.4 PHY
HTTP
HTTP
Client
Server
Server
Proxy
CoAP
CoAP
CoAP
CoAP
C
C
C
C
C
The Internet
Constrained Environment

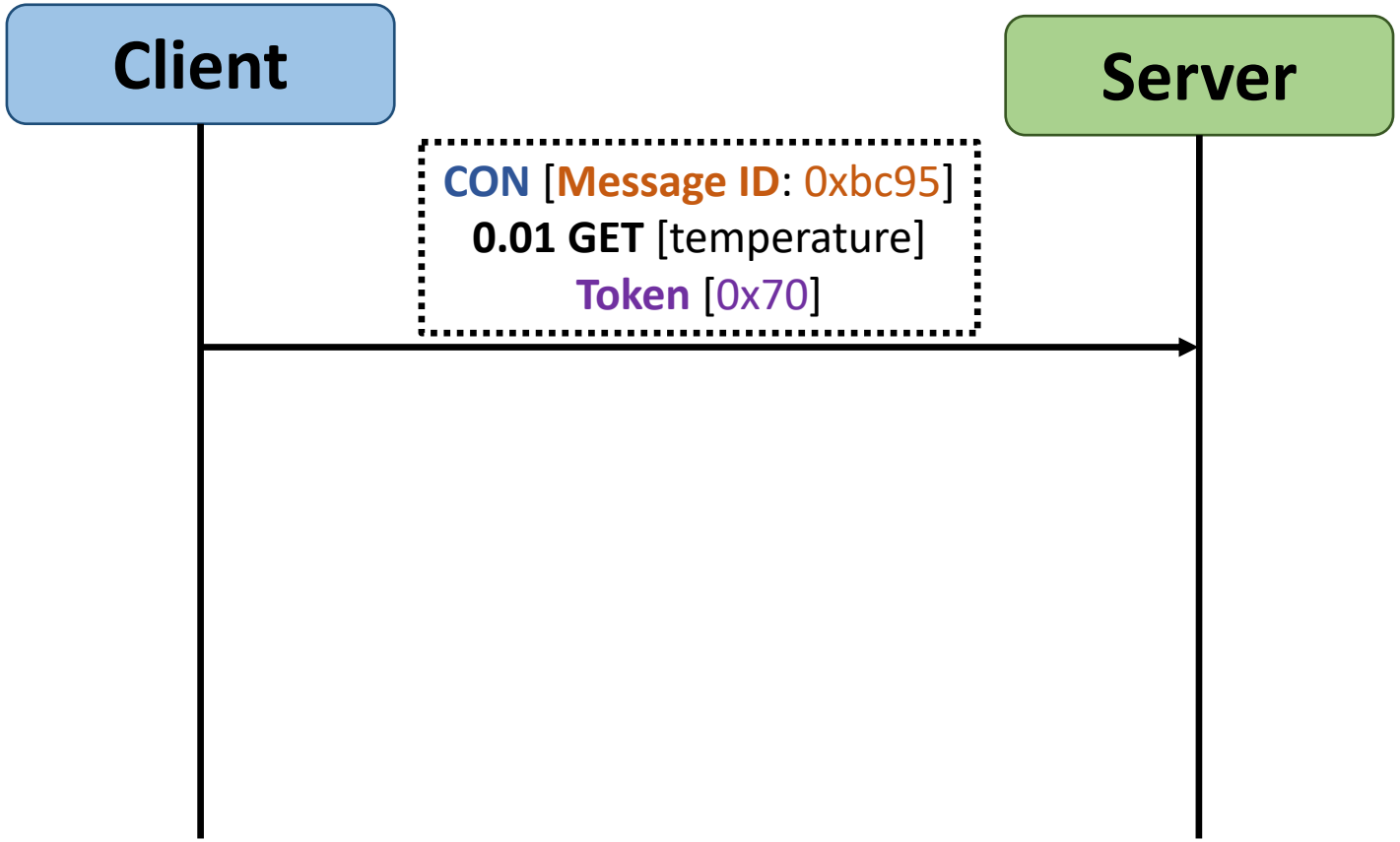




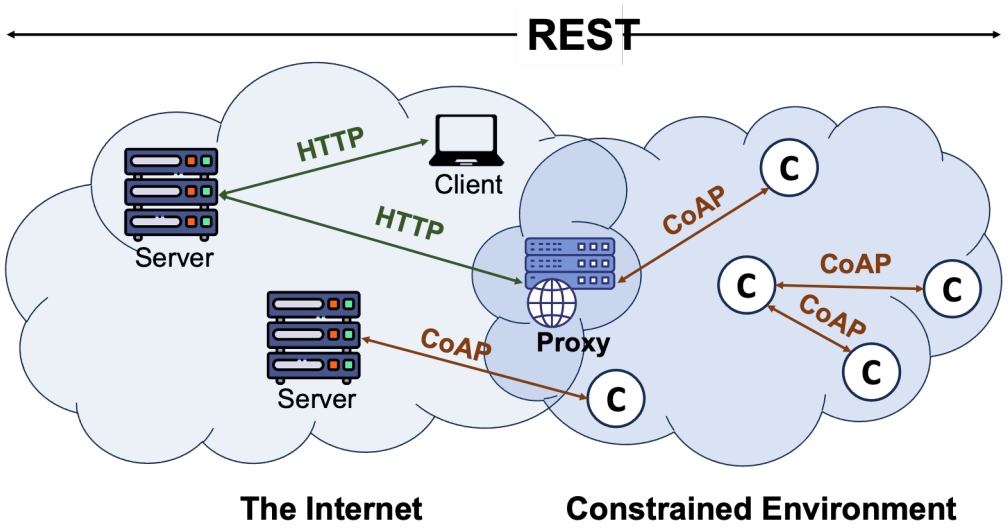


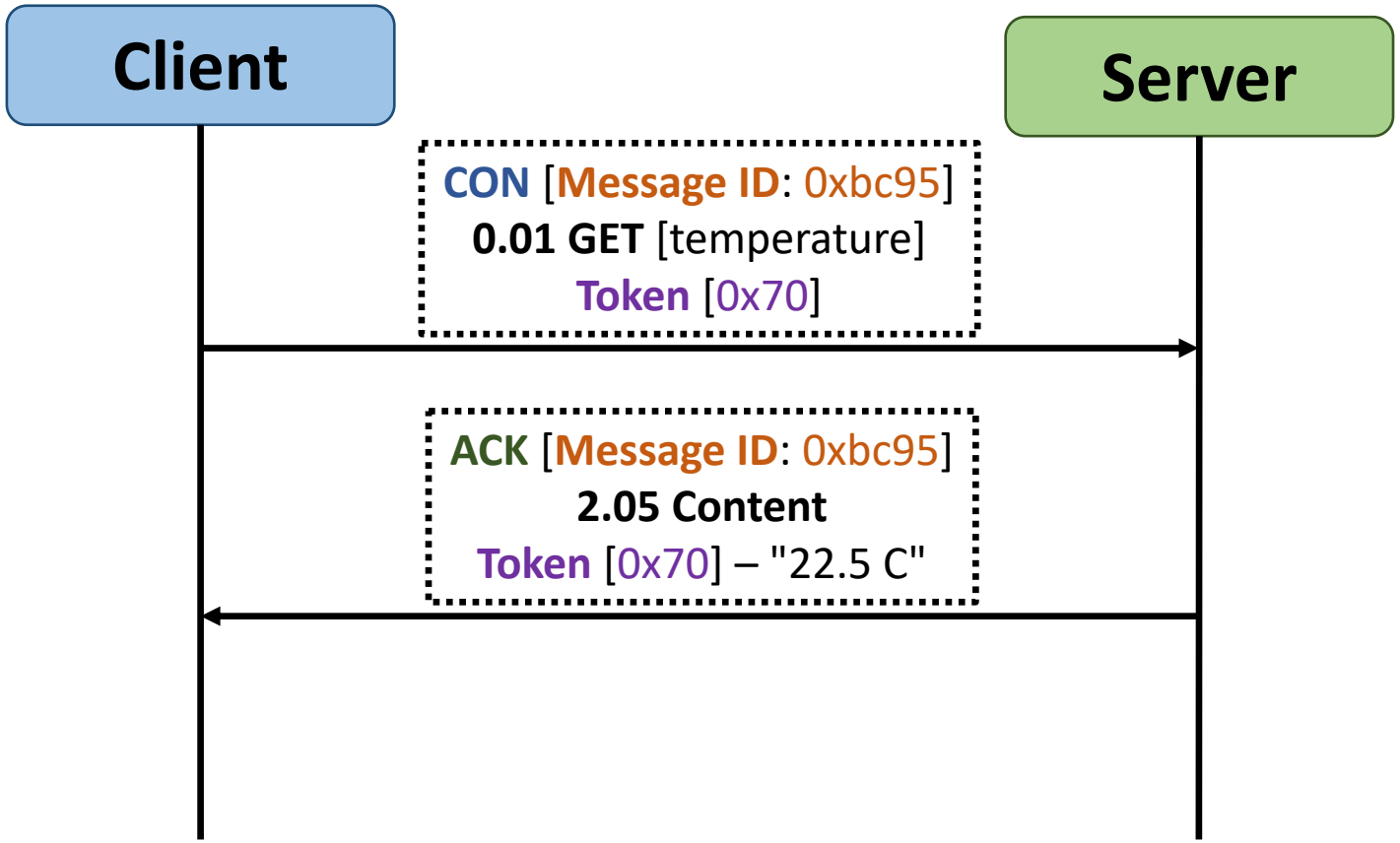
The interaction model of CoAP is similar to the client/server model of HTTP.



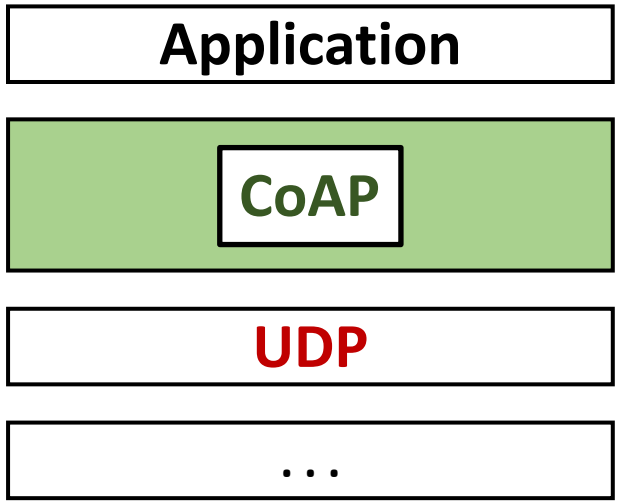


A CoAP request is equivalent to that of HTTP and is sent by a client to request an action on a resource on a server.

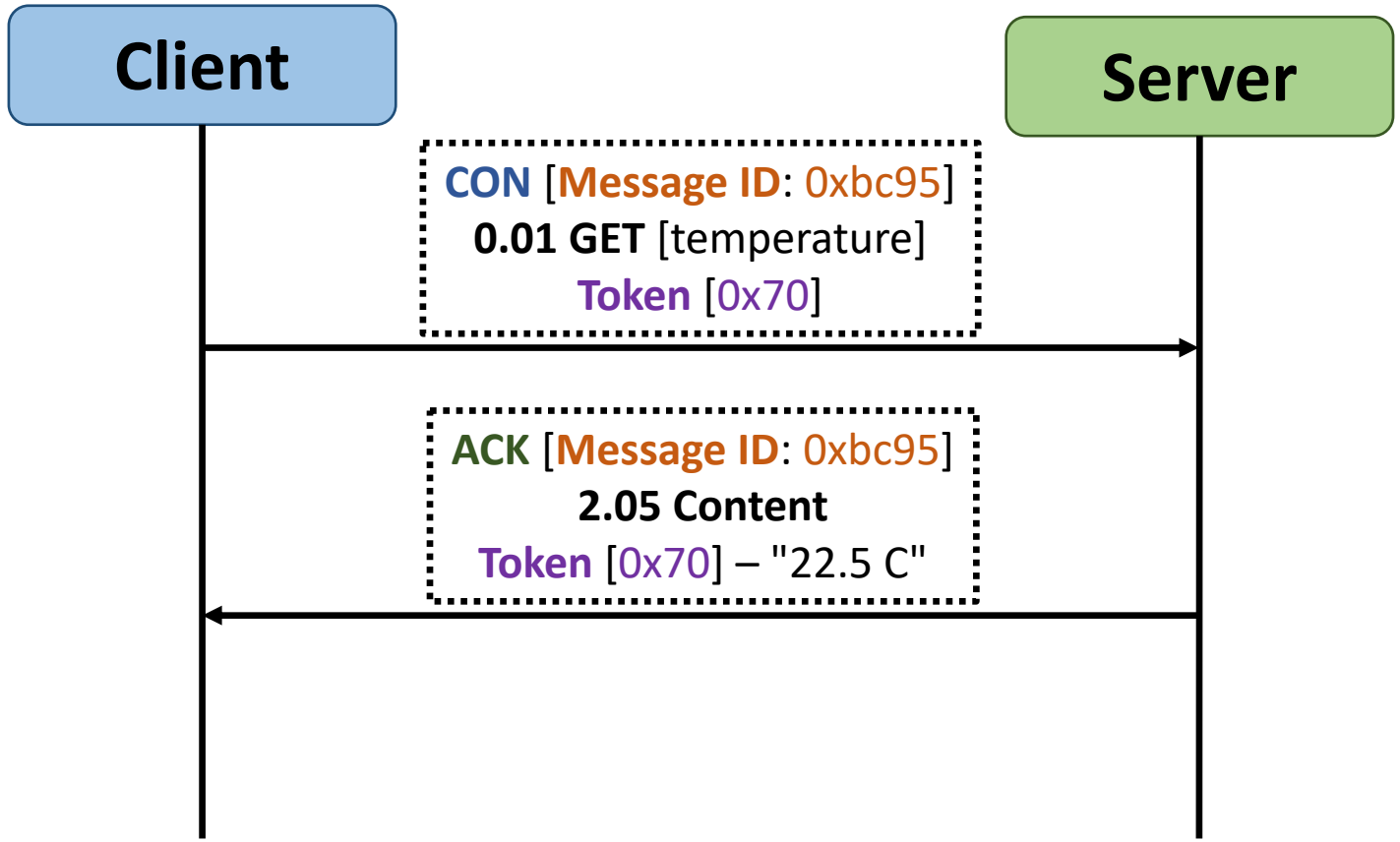




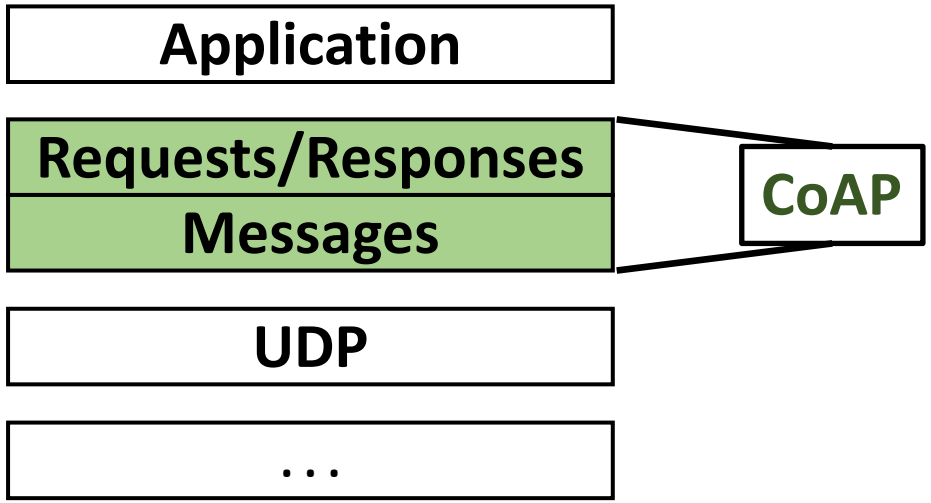
If the Response is immediately available, then it will be carried in the resulting Acknowledgement message.



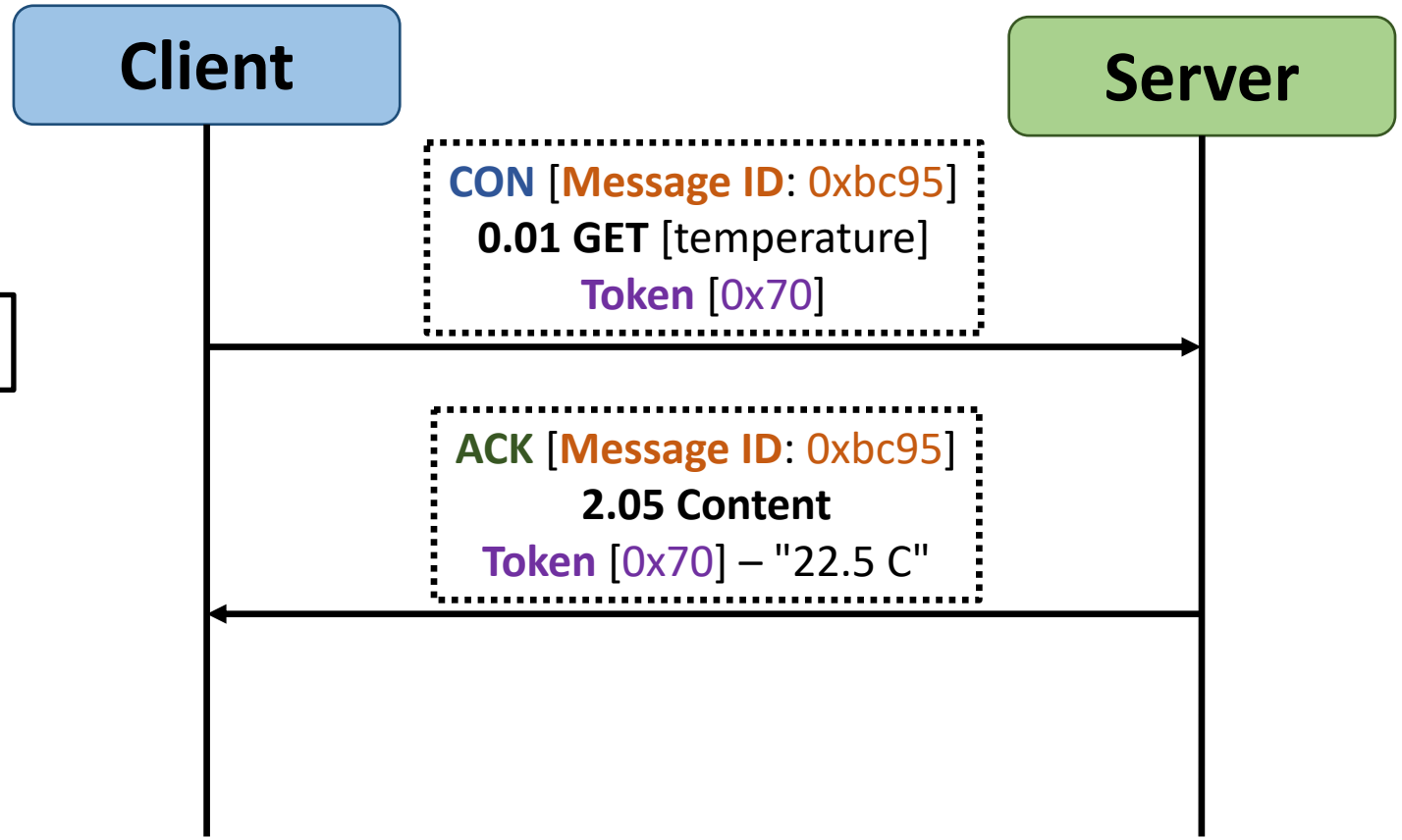
Abstract Layering of CoAP.



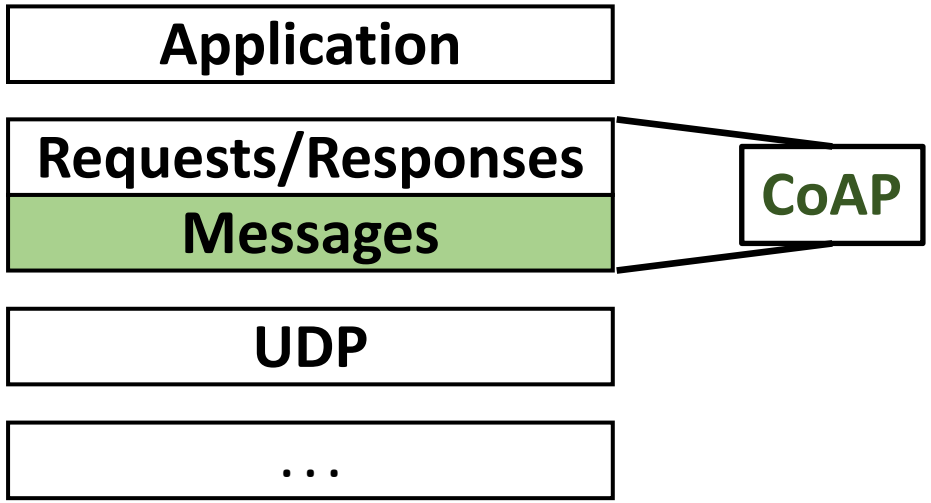
CoAP deals with these Request/Response exchanges asynchronously over the UDP transport protocol.



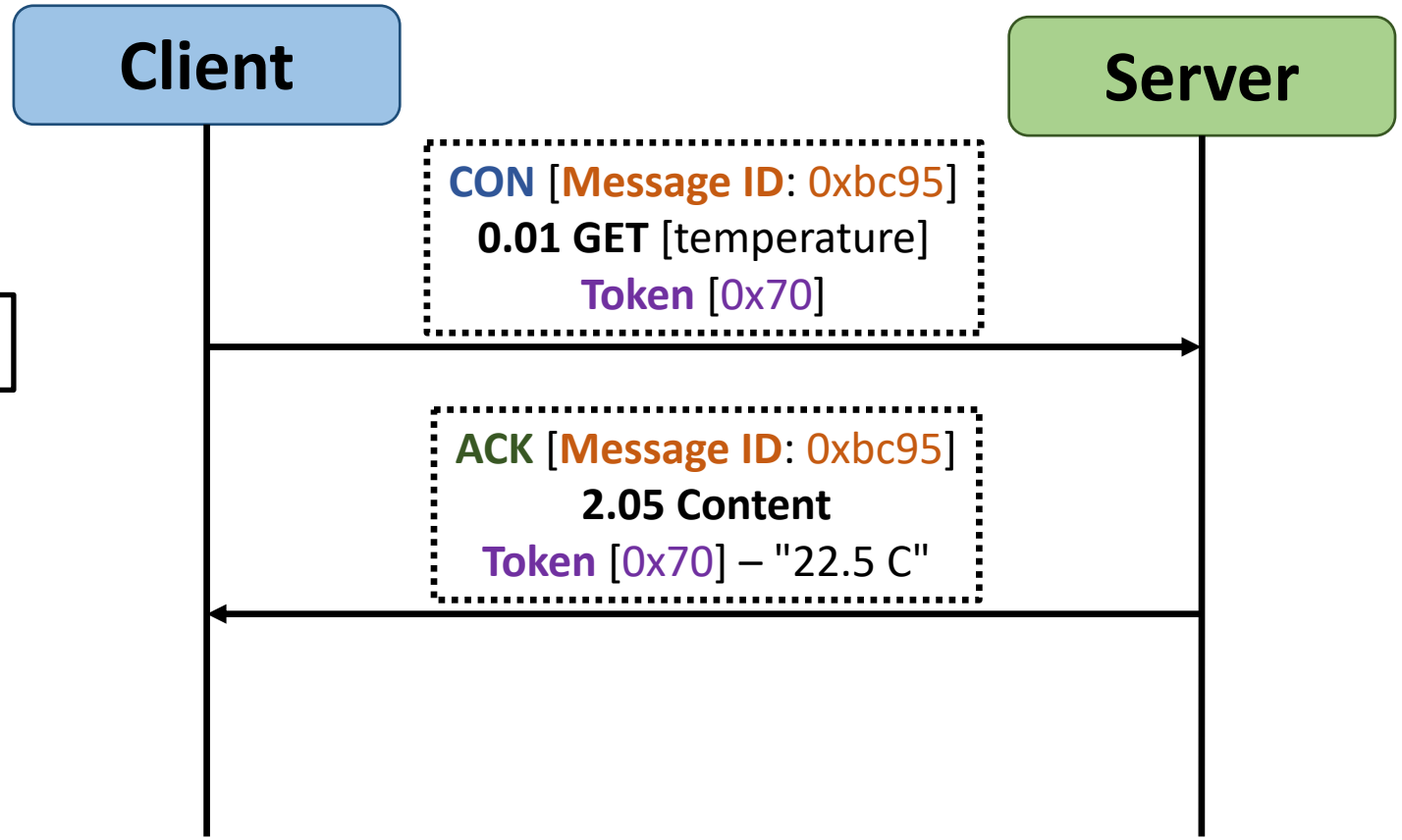
Abstract Layering of CoAP.



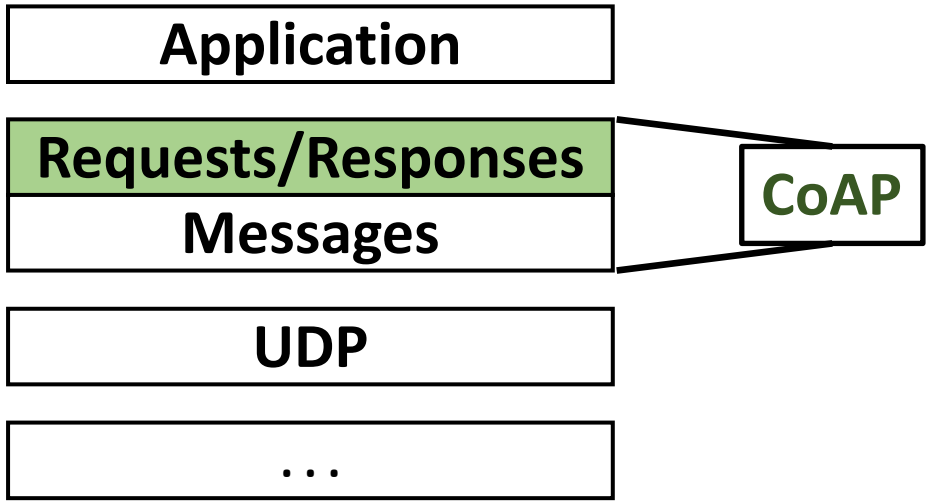
Logically, CoAP could be considered as a two-layer approach:



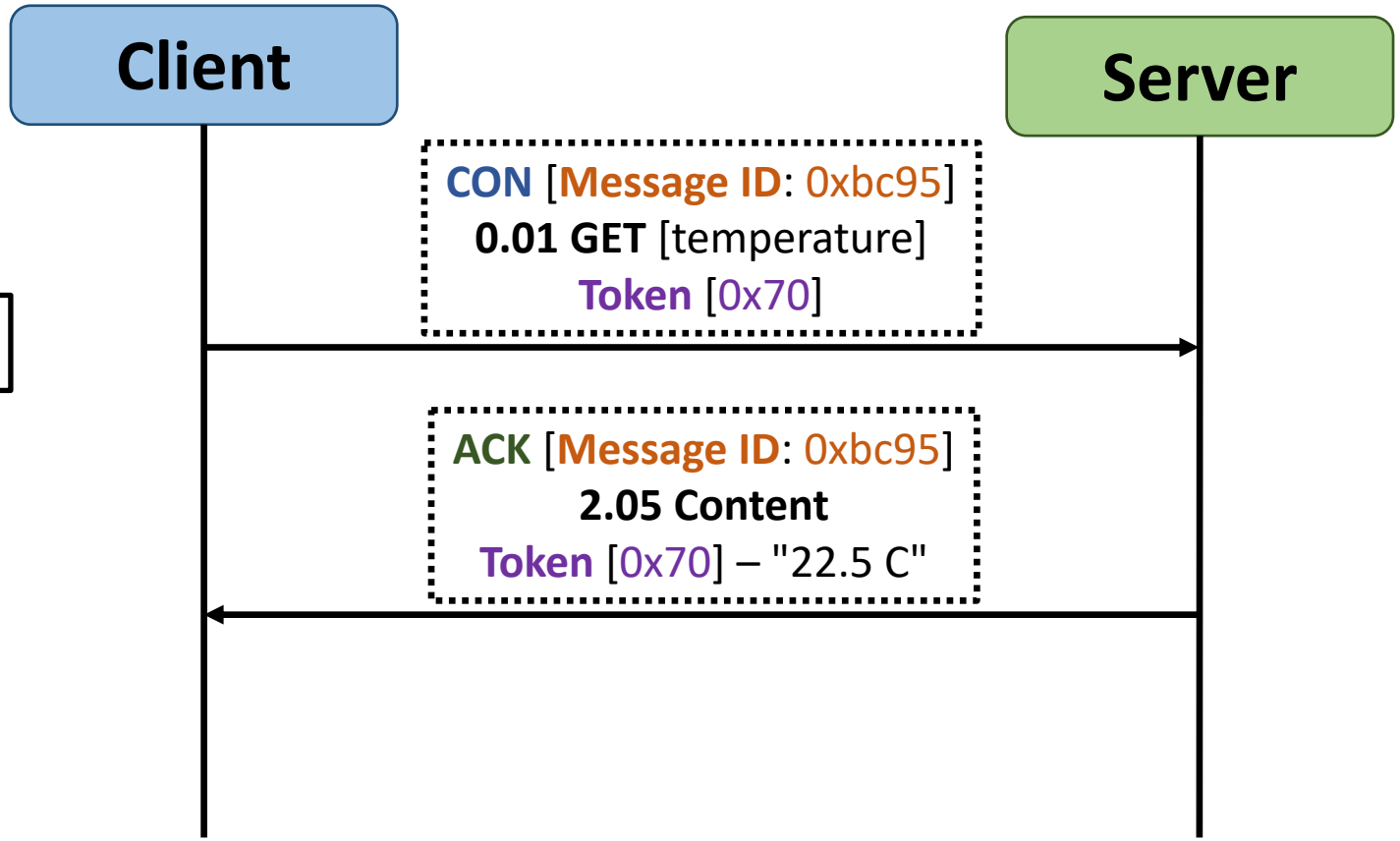
Abstract Layering of CoAP.



1. A Messaging layer employed to deal with UDP and the asynchronous nature of the messages.



Abstract Layering of CoAP.



2. and the Requests/Responses layer that manages request/response interactions.

4. CoAP Message Format

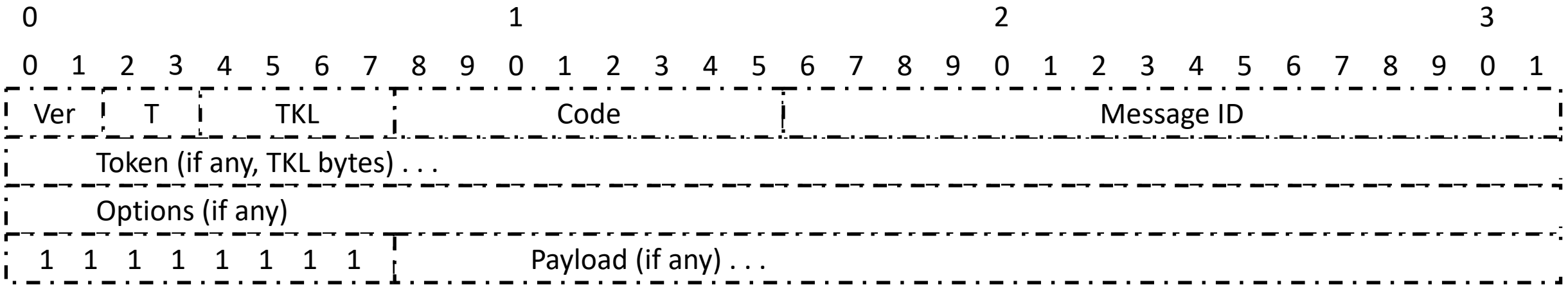
Click me!

as defined in **RFC 7252**!

IETF®

Context

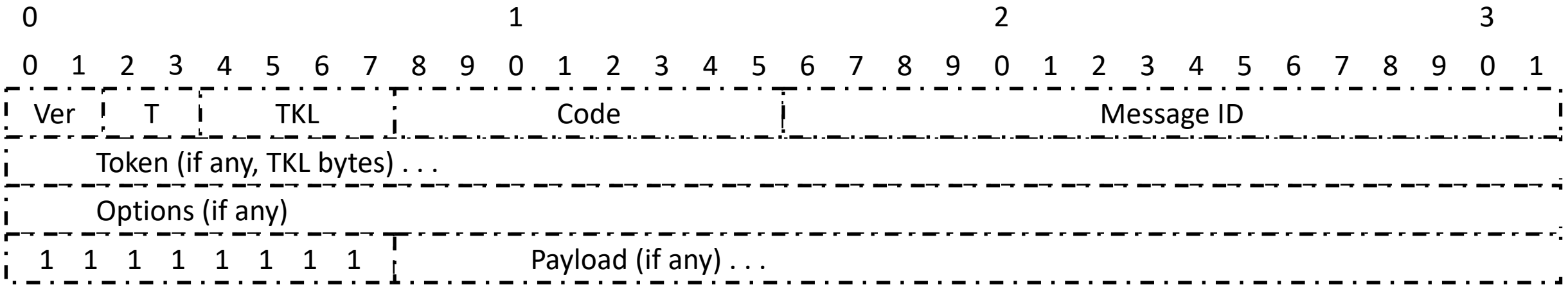
CoAP Message Format:



Context

CoAP Message Format:

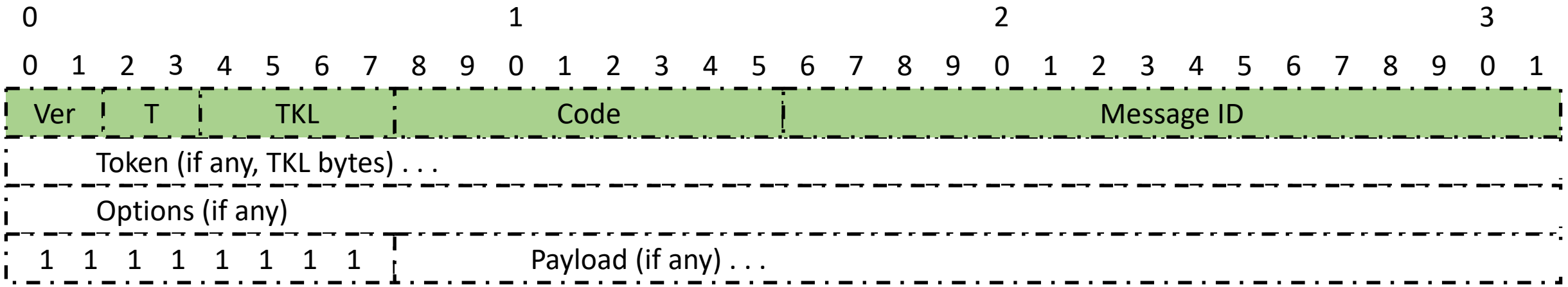
- A CoAP message is encoded in a **binary format**.



Context

CoAP Message Format:

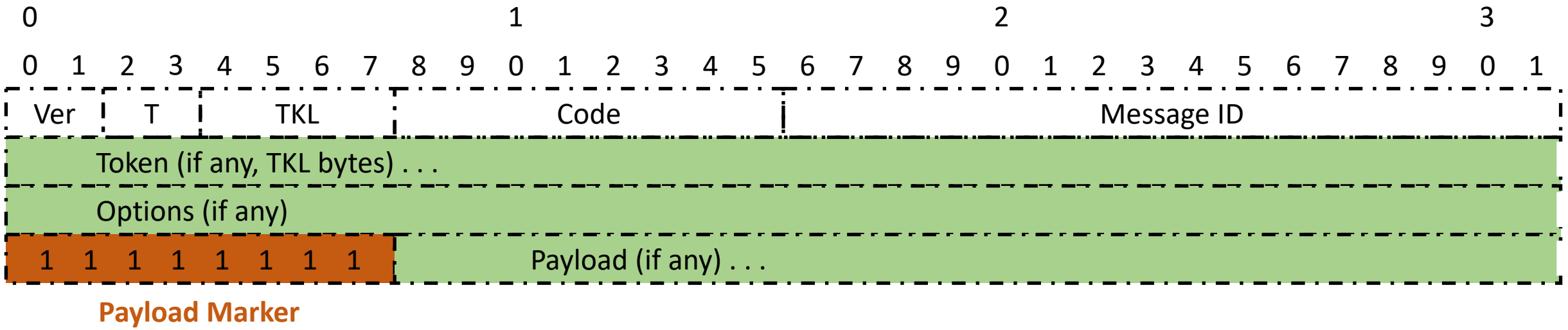
- A CoAP message is encoded in a **binary format**.
- Starts with **4-byte Header**.



Context

CoAP Message Format:

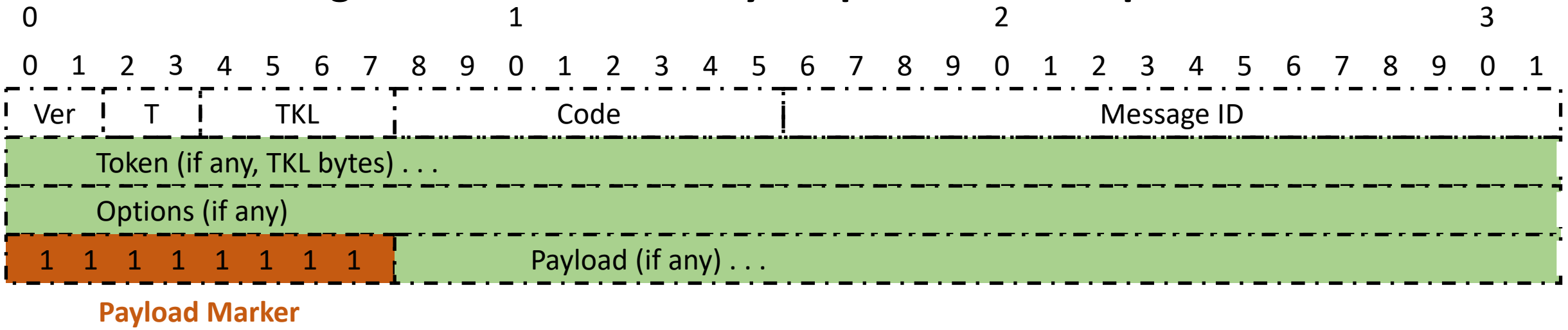
- A CoAP message is encoded in a **binary format**.
- Starts with **4-byte Header**.
- Followed by a **Token value,** a **CoAP Options**, and a **Payload**.



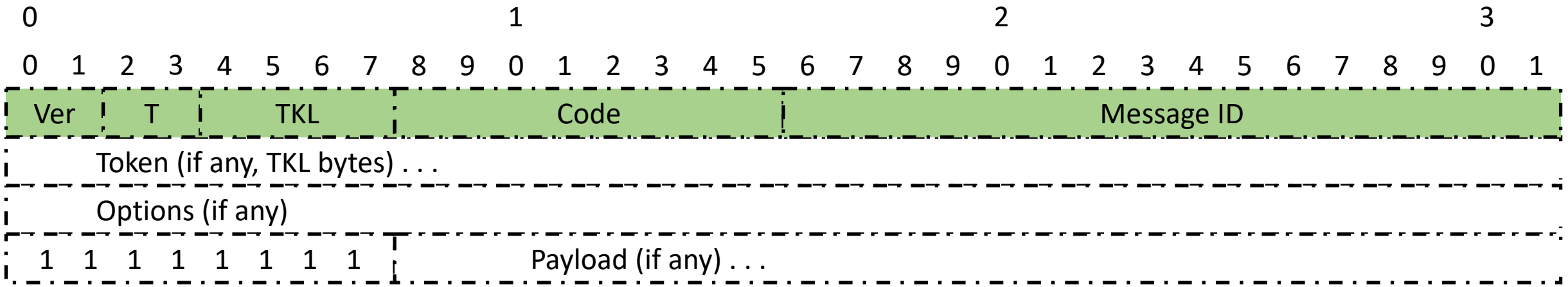
Context

CoAP Message Format:

- A CoAP message is encoded in a **binary format**.
- Starts with **4-byte Header**.
- Followed by a **Token value,** a **CoAP Options**, and a **Payload**.
- This message format is **shared by Requests and Responses**.

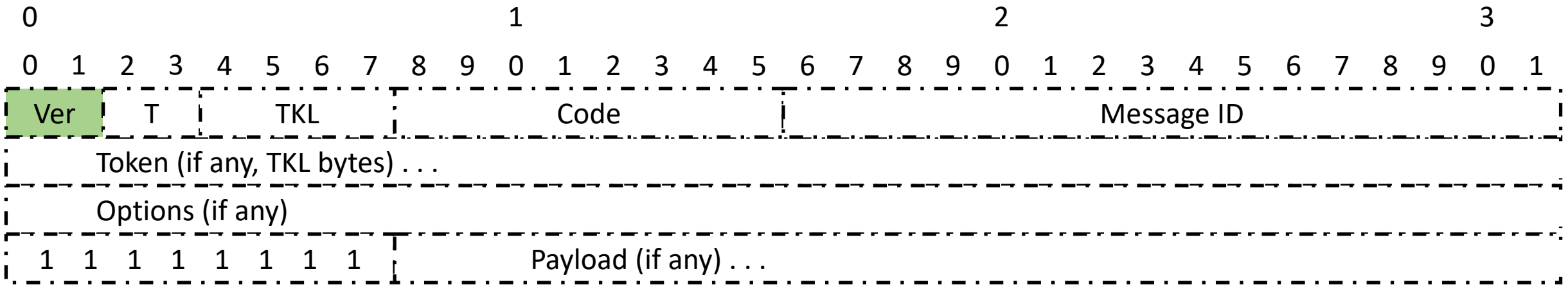


Message Format: Header



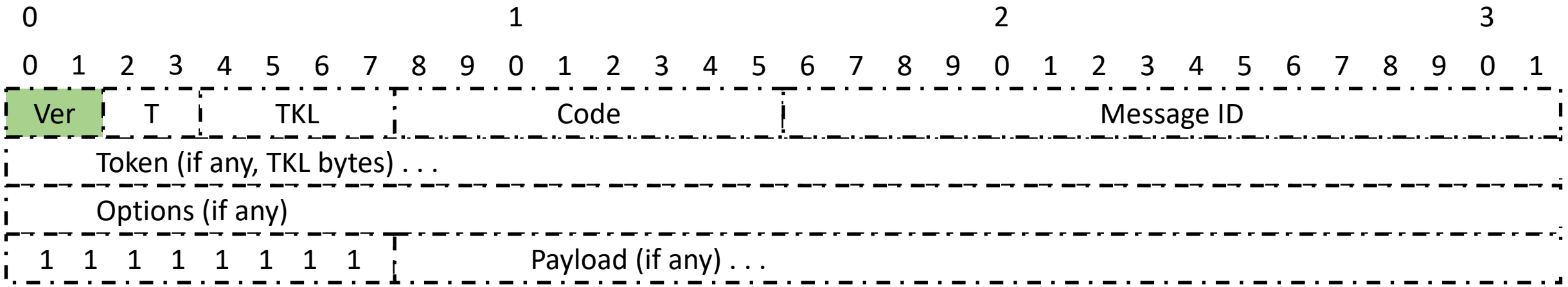
Message Format: Version

- **Version** (2 bits): *Indicates the CoAP Version Number*.



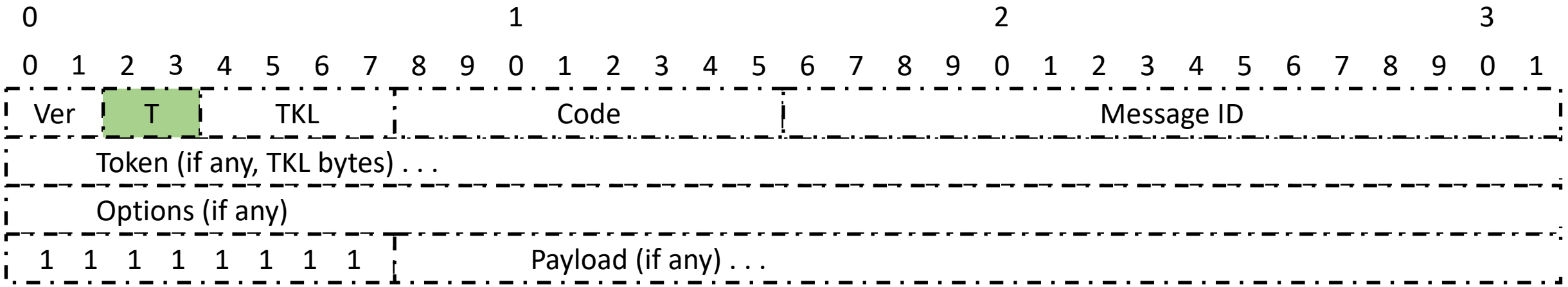
Message Format: Version

- **Version** (2 bits): *Indicates the CoAP Version Number*.
  - This field ***must*** be set to **01** when this **RFC** (i.e., RFC 7252) **is implemented**.
  - Rest of the values (**00**, **10**, **11**) are reserved for **future versions**.
  - Messages with **unknown** version numbers are **ignored**.



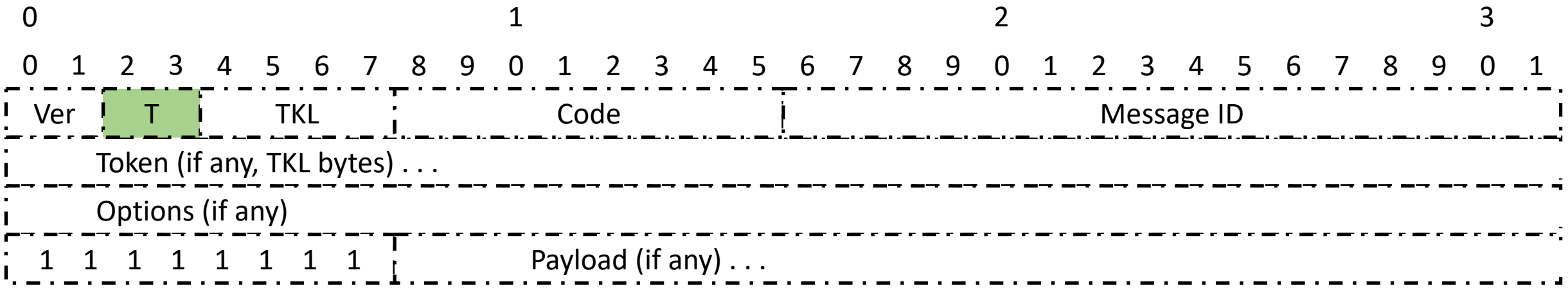
Message Format: Type

- **Type** (2 bits): *Indicates the Type of the Message*:



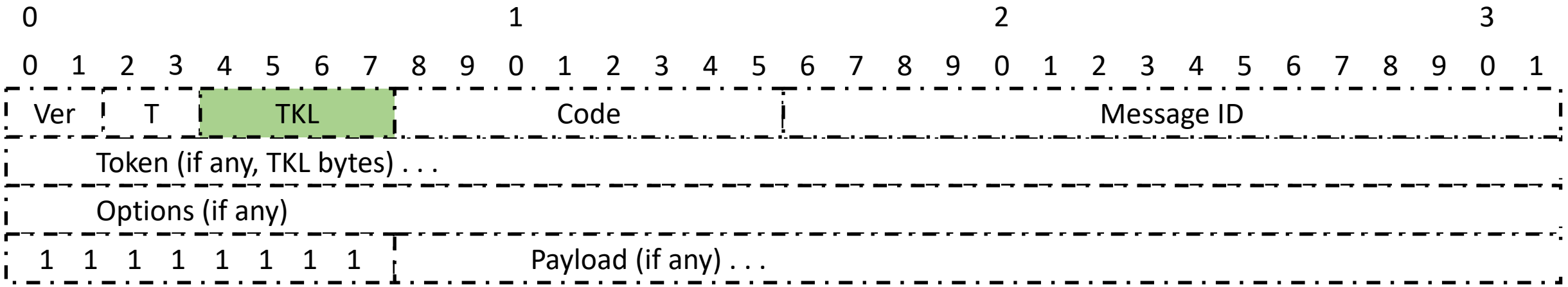
Message Format: Type

- **Type** (2 bits): *Indicates the Type of the Message*:
  - Confirmable: 0 (00 binary).
  - Non-confirmable: 1 (01 binary).
  - Acknowledgement: 2 (10 binary).
  - Reset: 3 (11 binary).



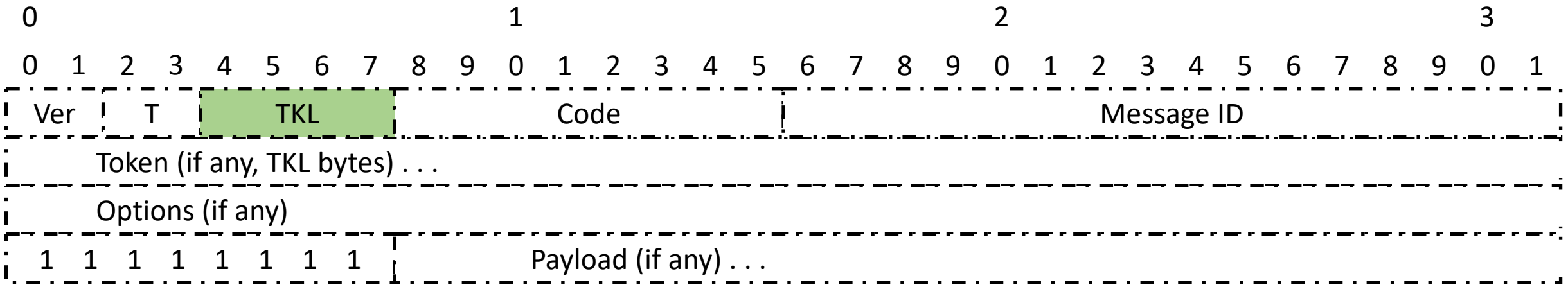
Message Format: Token Length

- **Token Length** (4 bits): *Indicates the Length of the Token field*.
  - The Token field comes with variable length (0-8 bytes).



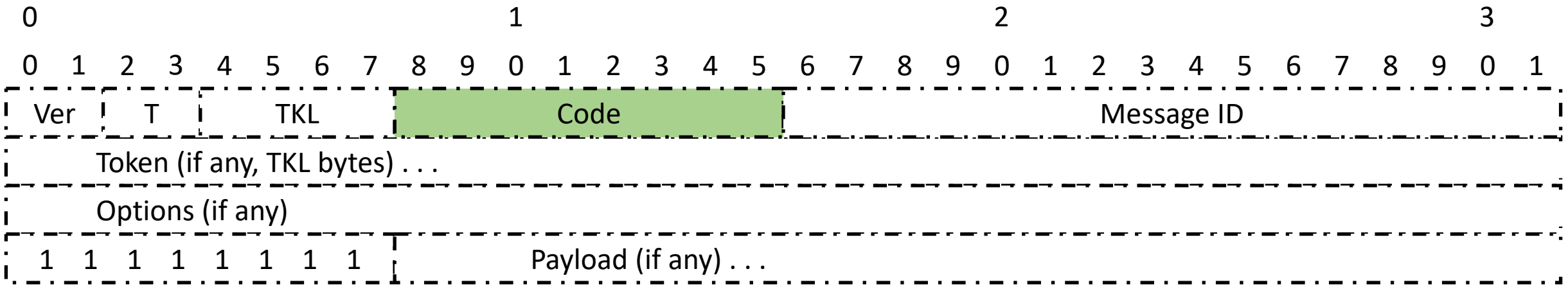
Message Format: Token Length

- **Token Length** (4 bits): *Indicates the Length of the Token field*.
  - The Token field comes with variable length (0-8 bytes).
  - The lengths 9-15 (1001 – 1111 binary) bytes are reserved.



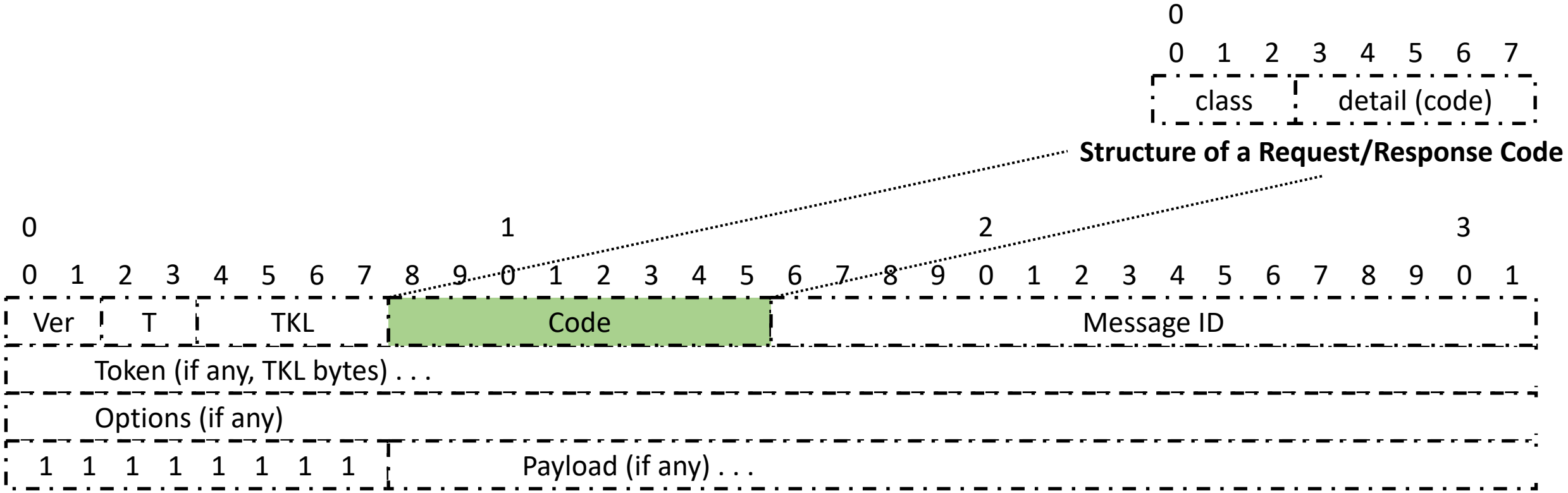
Message Format: Code

- **Code** (8 bits): *Indicates the Request/Response Code*.



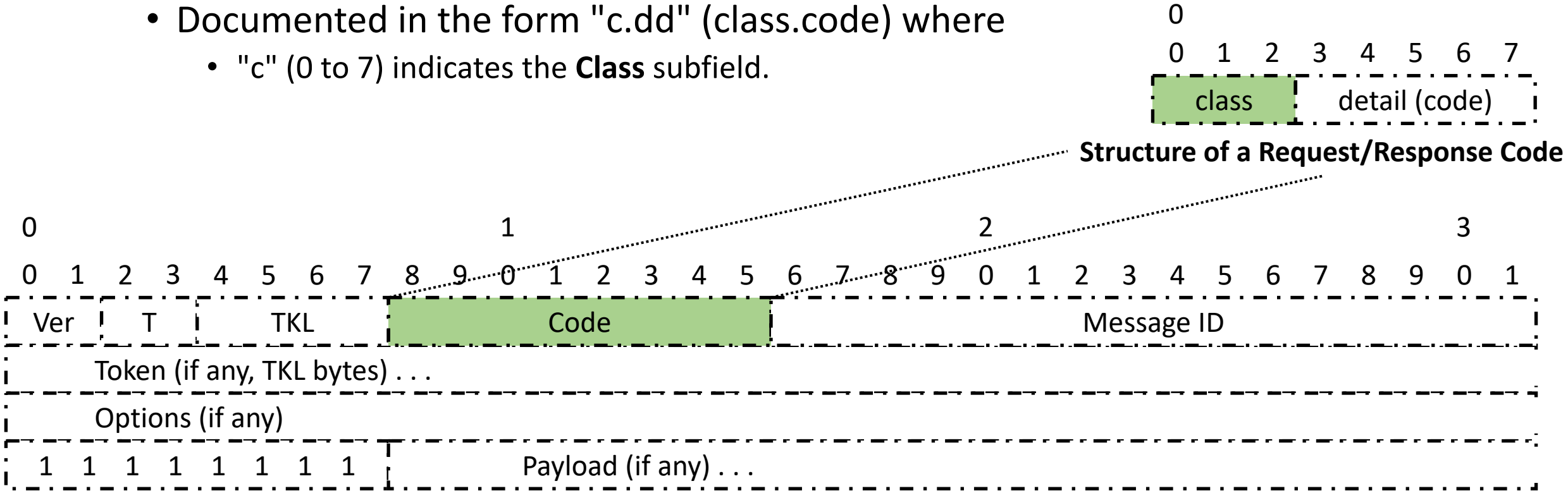
Message Format: Code

- **Code** (8 bits): *Indicates the Request/Response Code*.
  - Split into a 3-bit **Class** (MSB) and a 5-bit **Detail/Code** (LSB).



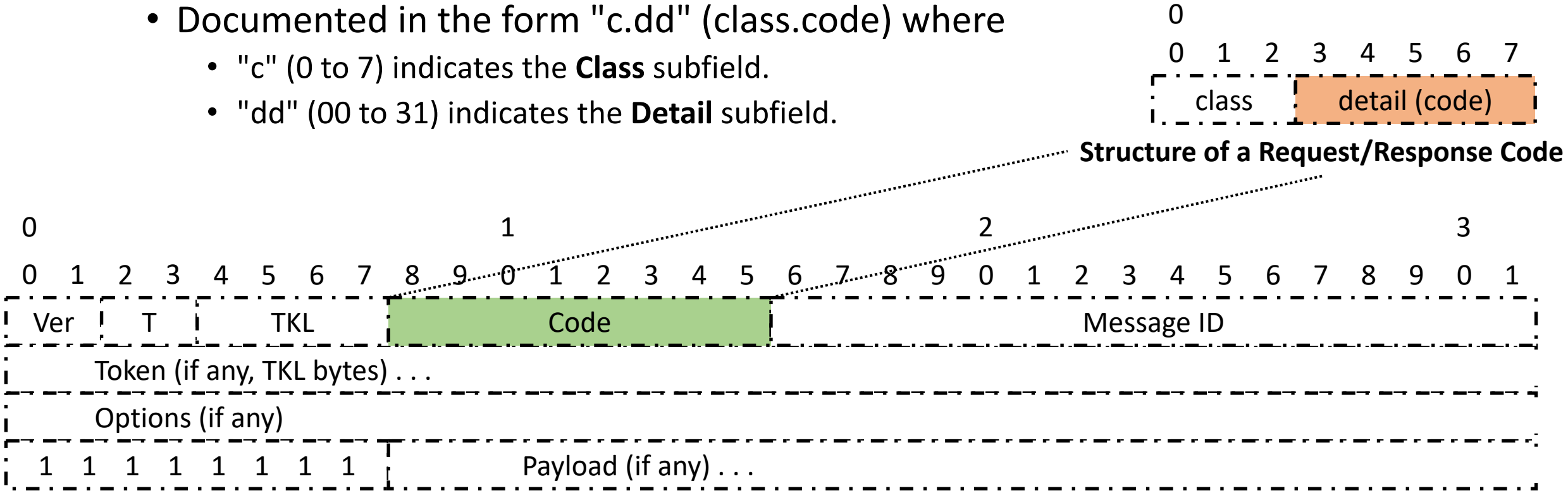
Message Format: Code

- **Code** (8 bits): *Indicates the Request/Response Code*.
  - Split into a 3-bit **Class** (MSB) and a 5-bit **Detail/Code** (LSB).
  - Documented in the form "c.dd" (class.code) where
    - "c" (0 to 7) indicates the **Class** subfield.



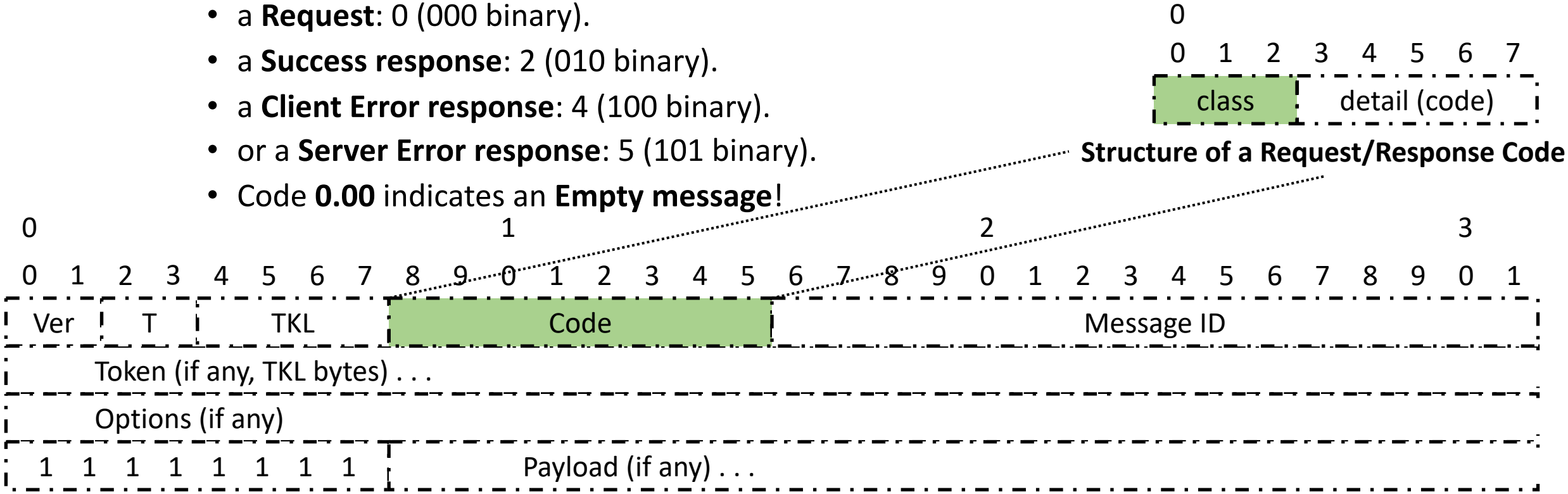
Message Format: Code

- **Code** (8 bits): *Indicates the Request/Response Code*.
  - Split into a 3-bit **Class** (MSB) and a 5-bit **Detail/Code** (LSB).
  - Documented in the form "c.dd" (class.code) where
    - "c" (0 to 7) indicates the **Class** subfield.
    - "dd" (00 to 31) indicates the **Detail** subfield.



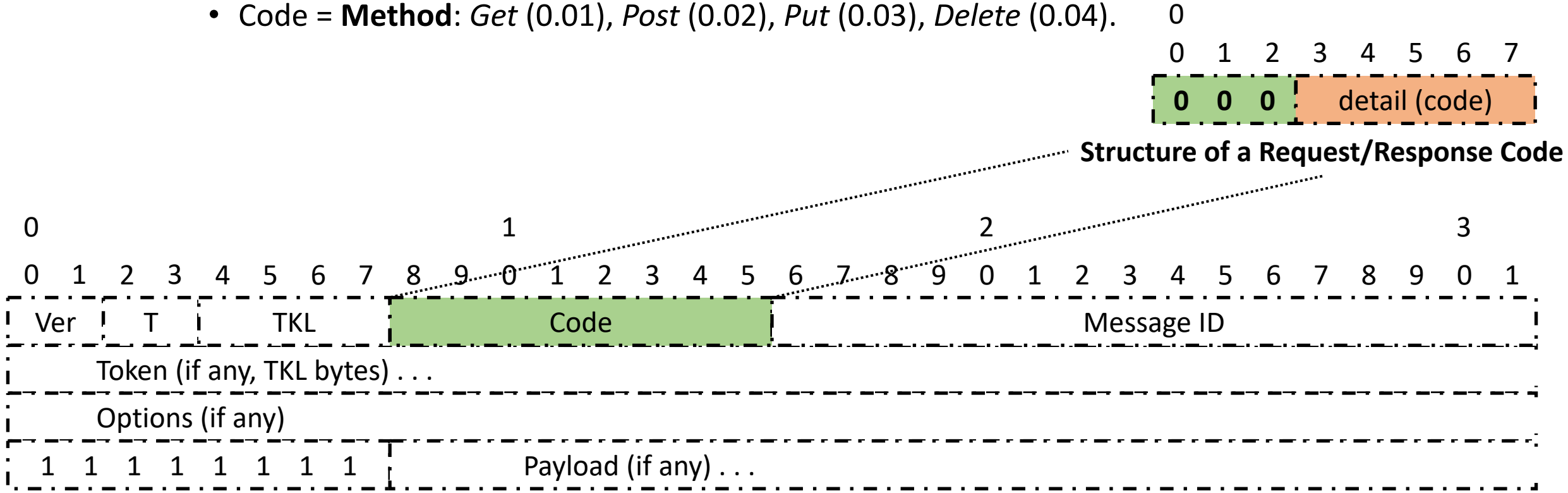
Message Format: Code

- **Code** (8 bits): *Indicates the Request/Response Code*.
  - The **Class** can indicate:
    - a **Request**: 0 (000 binary).
    - a **Success response**: 2 (010 binary).
    - a **Client Error response**: 4 (100 binary).
    - or a **Server Error response**: 5 (101 binary).
    - Code **0.00** indicates an **Empty message**!



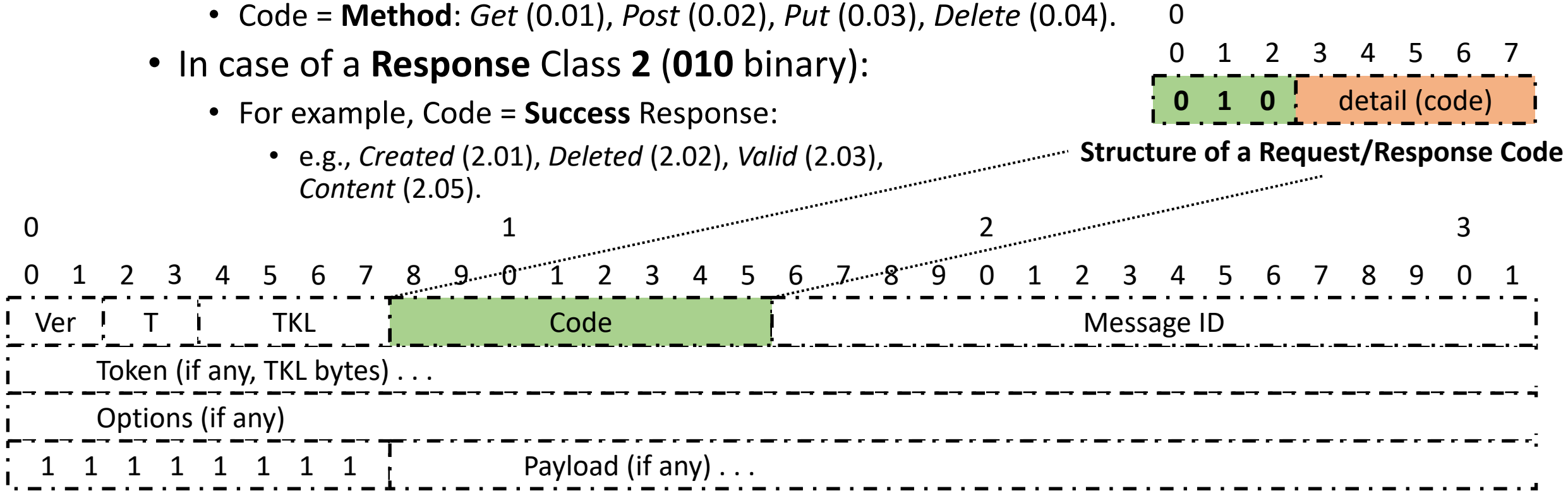
Message Format: Code

- **Code** (8 bits): *Indicates the Request/Response Code*.
  - In case of a **Request** Class **0** (**000** binary):
    - Code = **Method**: *Get* (0.01), *Post* (0.02), *Put* (0.03), *Delete* (0.04).



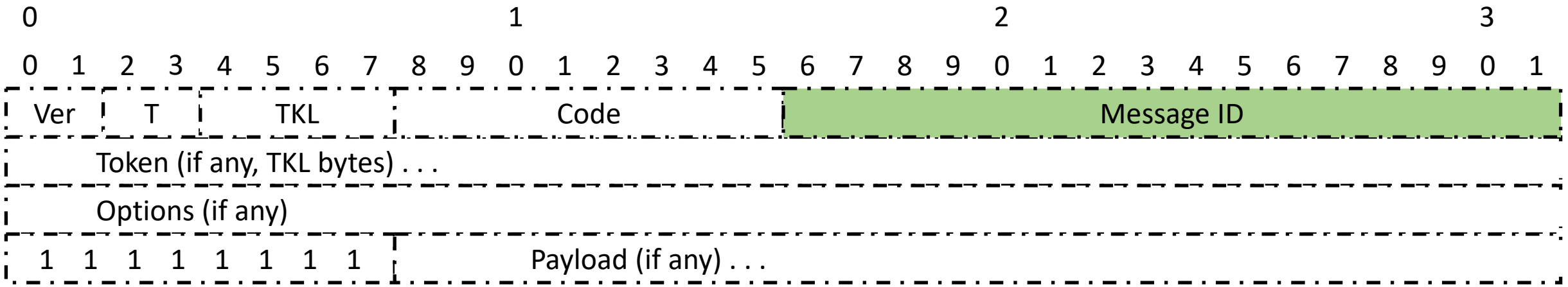
Message Format: Code

- **Code** (8 bits): *Indicates the Request/Response Code*.
  - In case of a **Request** Class **0** (**000** binary):
    - Code = **Method**: *Get* (0.01), *Post* (0.02), *Put* (0.03), *Delete* (0.04).
  - In case of a **Response** Class **2** (**010** binary):
    - For example, Code = **Success** Response:
      - e.g., *Created* (2.01), *Deleted* (2.02), *Valid* (2.03), *Content* (2.05).



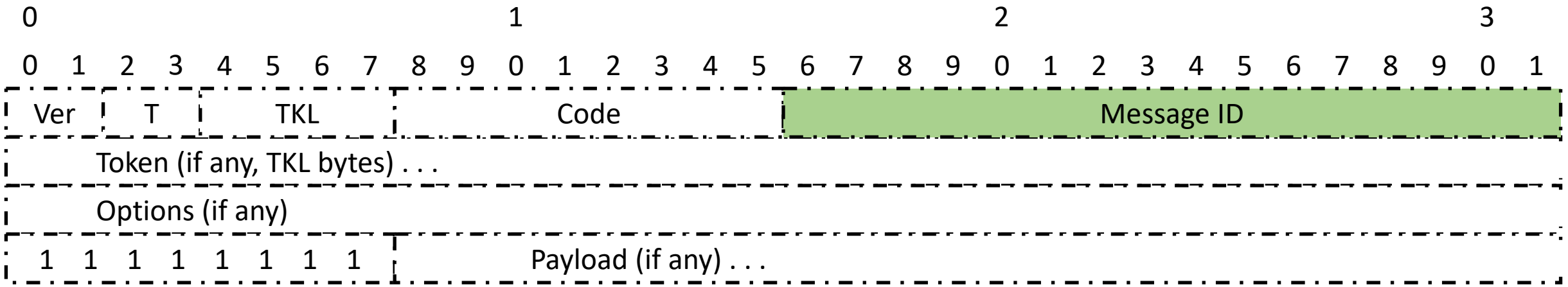
Message Format: Message ID

- Message ID (16 bits): ***Detects*** *Message Duplication and* ***enables*** *Reliability*.



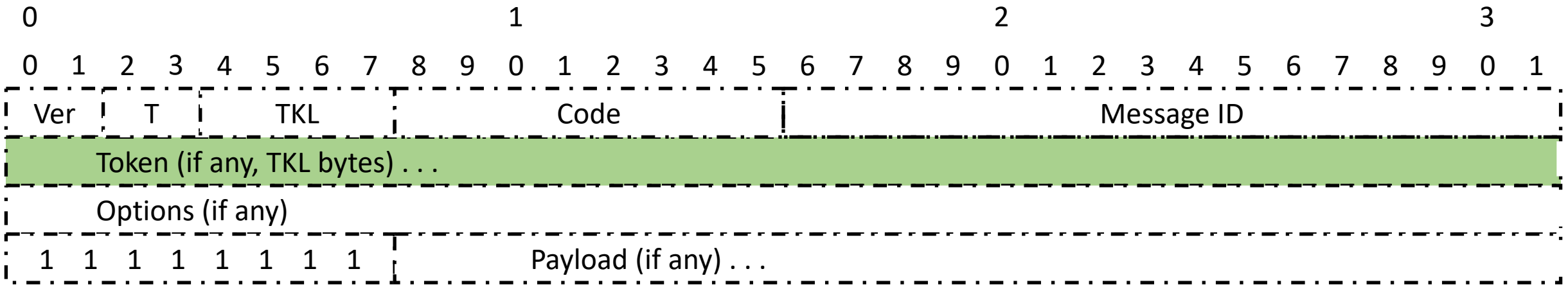
Message Format: Message ID

- Message ID (16 bits): ***Detects*** *Message Duplication and* ***enables*** *Reliability*.
  - **Matches** ACK/Reset messages to Confirmable/Non-confirmable.
  - For more, check out the “CoAP Type of Messages” Section!

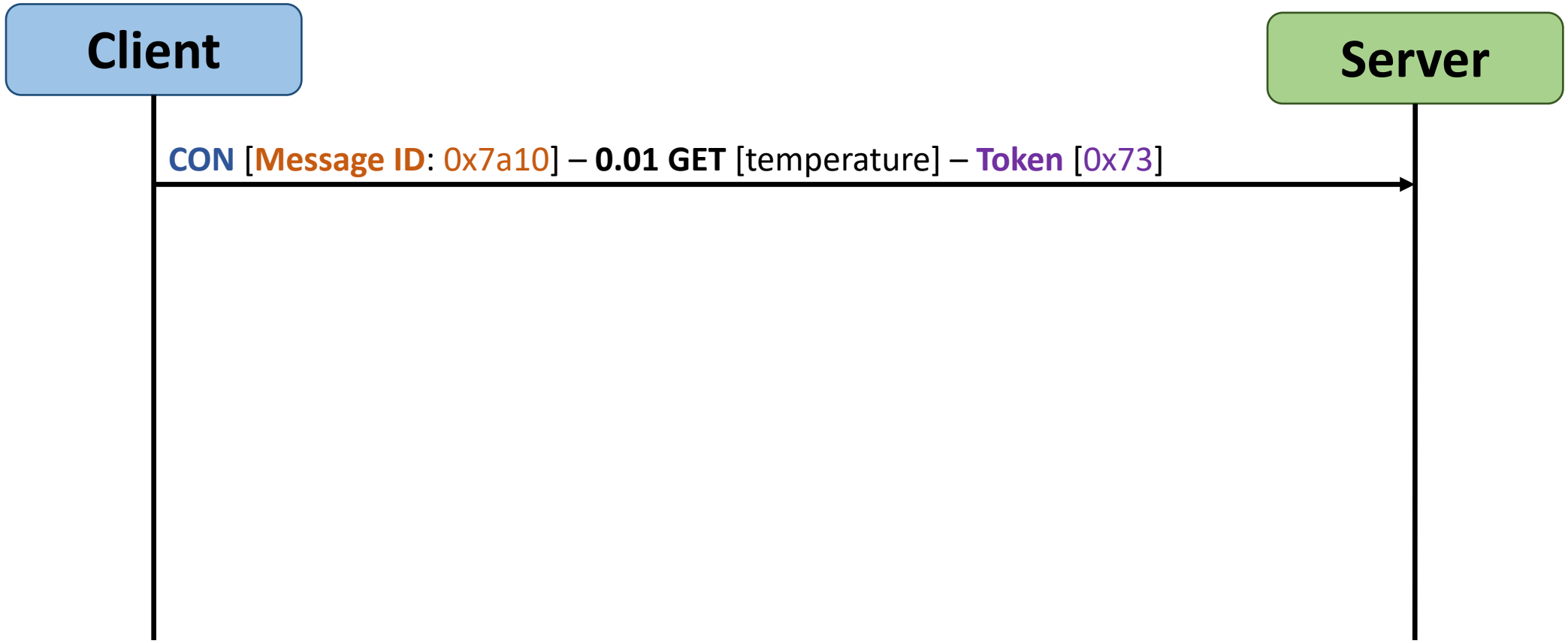


Message Format: Token

- Token (0 to 8 bytes, as given by the Token Length field): ***Correlates*** *Requests to Responses*.



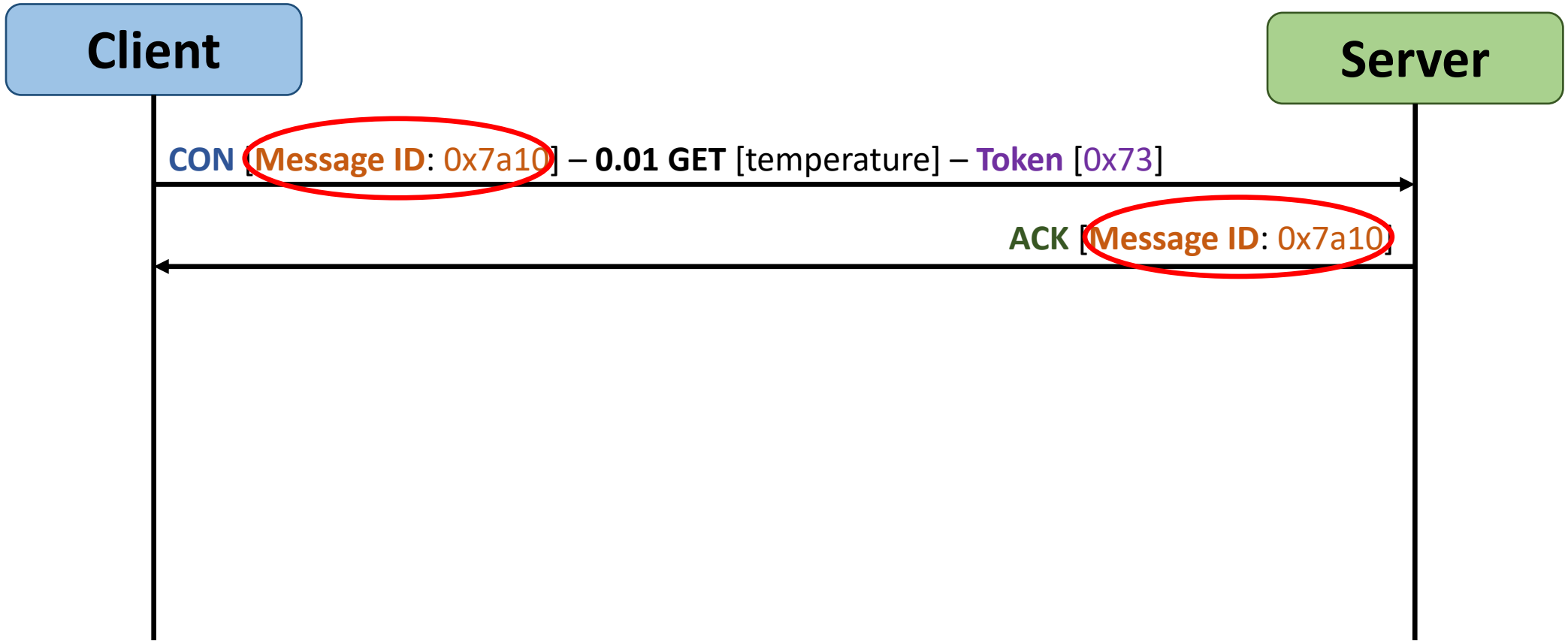
Token vs Message ID: A GET Request with a Separate Response example



To better understand the difference between Message ID and Token, here is an example of a Get Request with a Separate Response. **Every request carries a Token whose value was generated by the client**.

Source: ***RFC 7252***

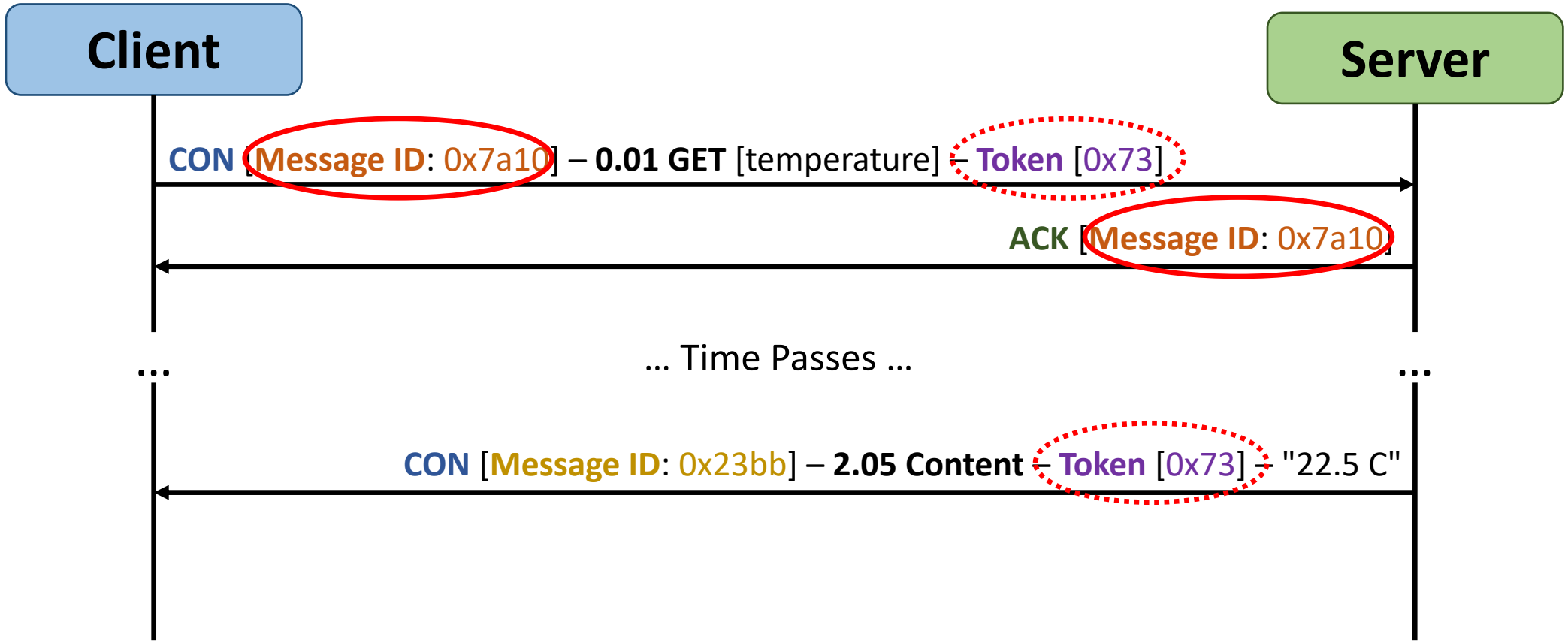
Token vs Message ID: A GET Request with a Separate Response example



If the server is not able to respond immediately to a request carried in a Confirmable message, **it responds with an Empty Acknowledgement** message **so** that **the client can stop retransmitting** the request.

Source: ***RFC 7252***

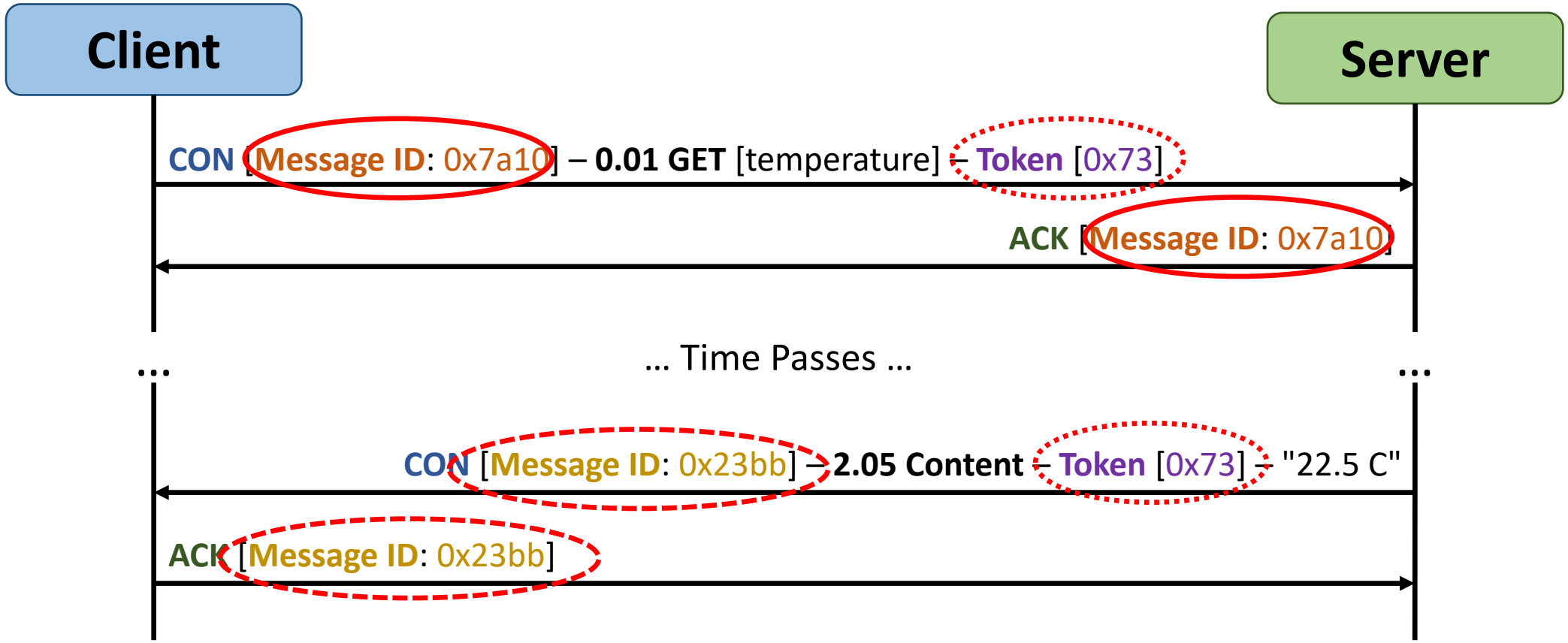
Token vs Message ID: A GET Request with a Separate Response example



When the response is ready, the **server sends it** in a **New Confirmable** message **without modifying** the **Token Value**!

Source: ***RFC 7252***

Token vs Message ID: A GET Request with a Separate Response example

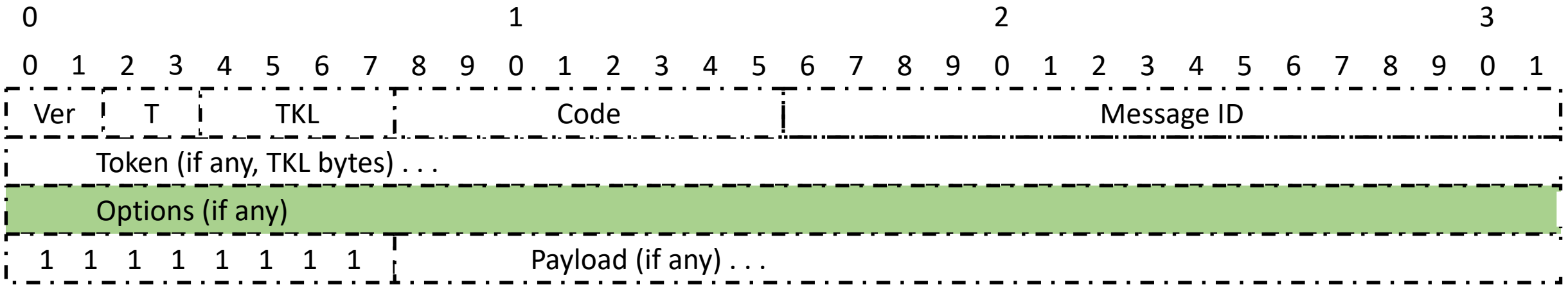


Finally, **this new Confirmable Message** in turn **needs to be acknowledged** by the client.

Source: ***RFC 7252***

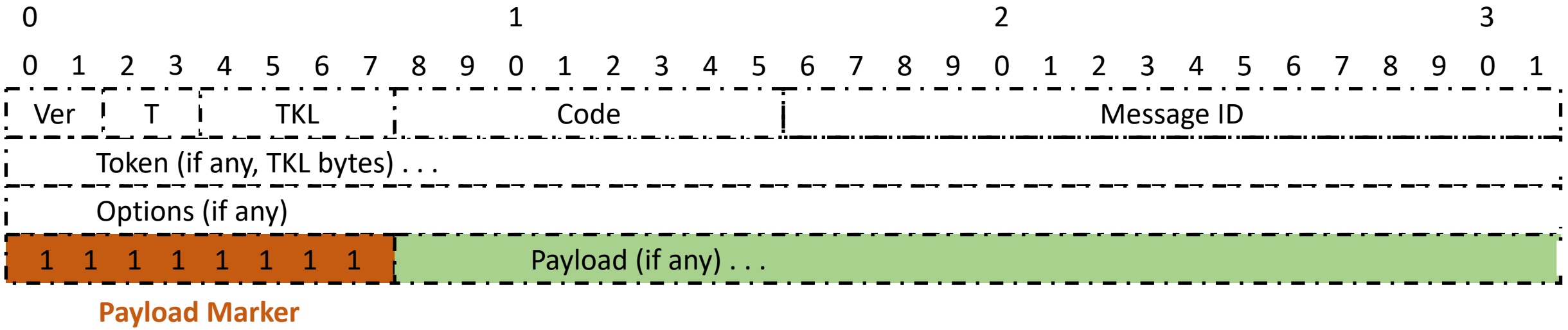
Message Format: Options

- Options: a sequence of zero or more CoAP Options in **Type-Length-Value** (TLV) **format**, in which different kind of information could be sent.



Message Format: Payload

- Payload (if any):
  - An Option can be followed by the end of the message, by another Option, or by the Payload.
  - It is Prefixed by the 1-byte "payload marker" (0xFF, or 11111111 in binary).
  - Its length is implied by the datagram length.



5. CoAP Type of Messages

Click me!

as defined in **RFC 7252**!

Type of Messages

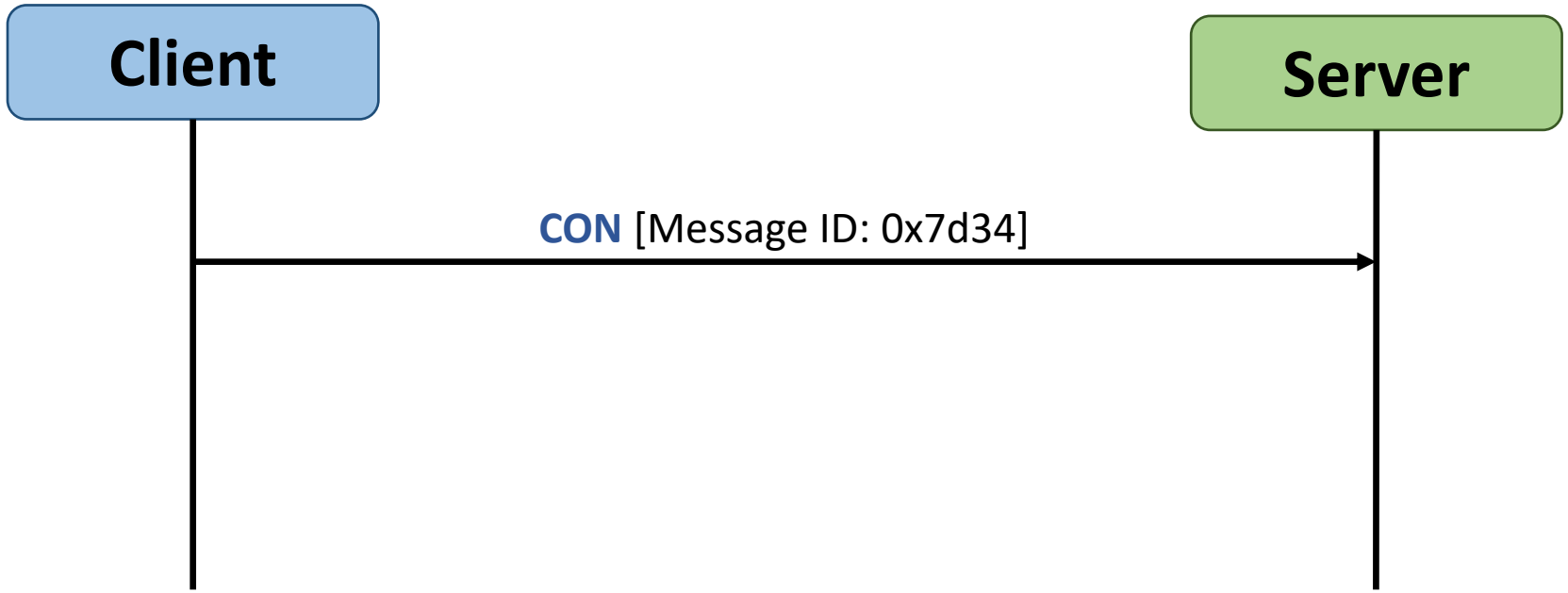
- **Confirmable** (CON)
- **Non-confirmable** (NON)
- **Acknowledgement** (ACK)
- **Reset** (RST)

CONFIRMABLE

Confirmable	• A message that **requires** an ACK.

A confirmable message elicits exactly one return message of type ACK or Reset.

CONFIRMABLE



An example of a Conrfirmable Message, a ***Reliable Message Transmission***:
“To provide a reliable message transmission, the Message is marked as Confirmable.”

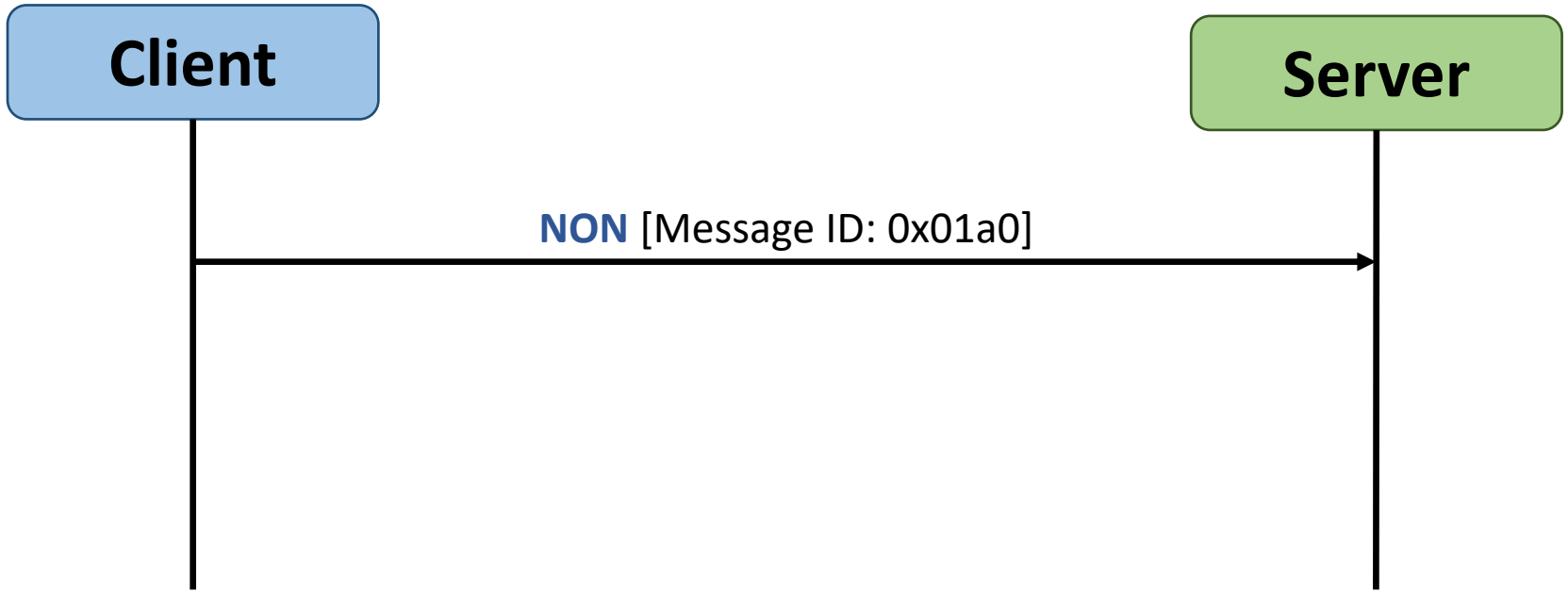
Source: ***RFC 7252***

NON-CONFIRMABLE

Confirmable	• A message that **requires** an ACK.
Non-confirmable	• A message that **does not** require an ACK.

For example, messages (measurements) that are repeated regularly from a sensor.

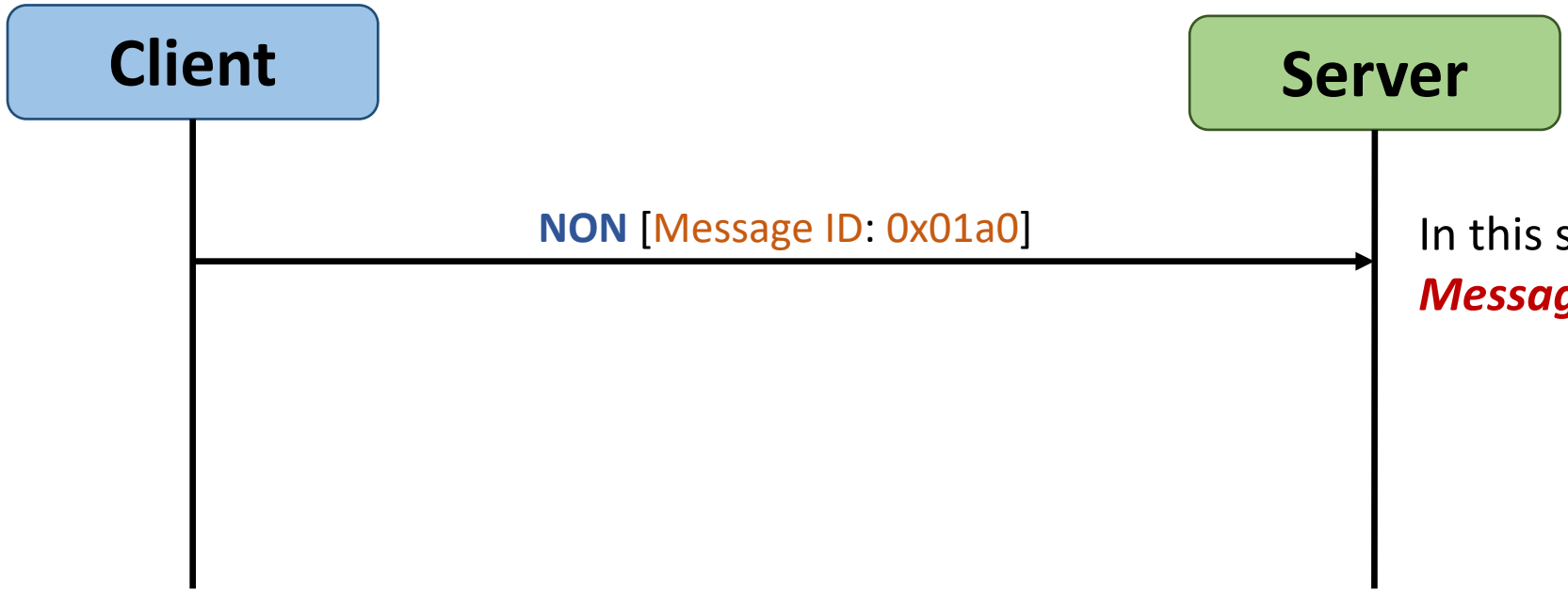
NON-CONFIRMABLE



An example of a Non-conrfirmable Message, an ***Unreliable Message Transmission***:
"A message that does not require a reliable transmission, then it is marked as Non-confirmable."

Source: ***RFC 7252***

NON-CONFIRMABLE



In this scenario, ***Message ID*** *is used for* ***Message Duplication Detection***!!!

An example of a Non-conrfirmable Message, an ***Unreliable Message Transmission***.

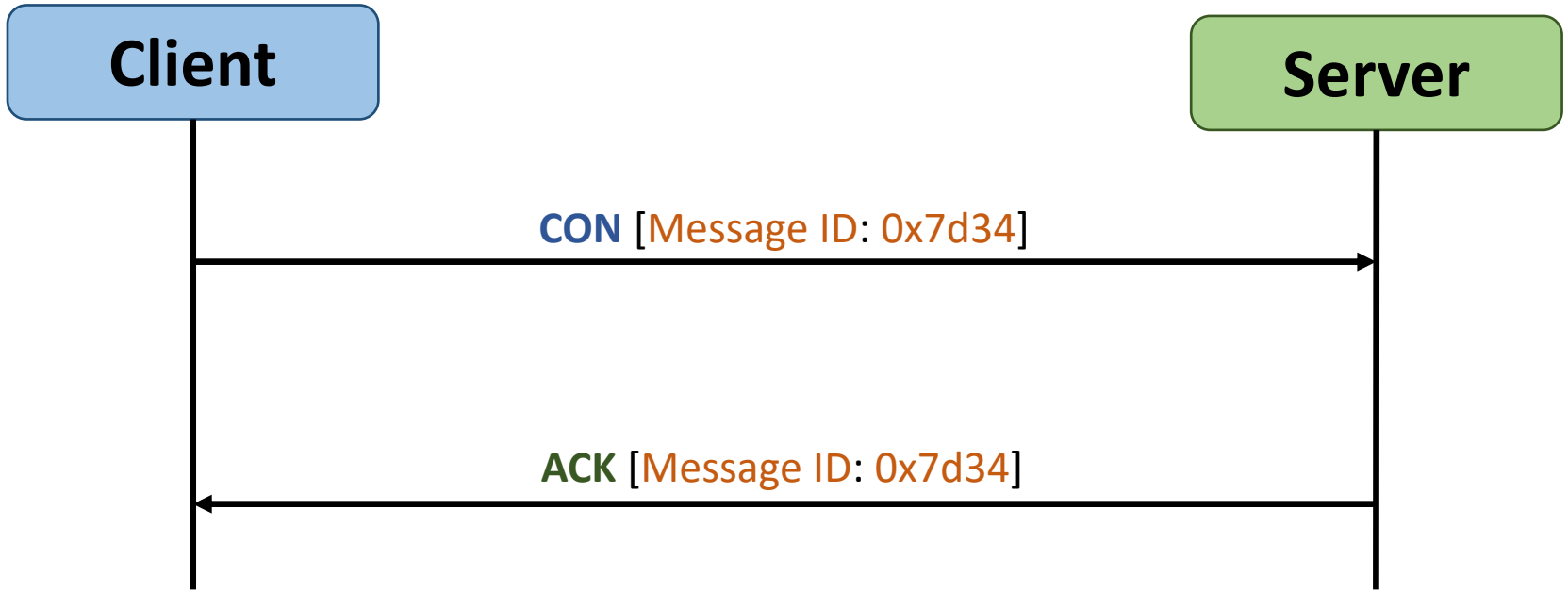
Source: ***RFC 7252***

ACKNOWLEDGEMENT

Confirmable	• A message that **requires** an ACK.
Non-confirmable	• A message that **does not** require an ACK.
Acknowledgement	• An ACK **acknowledges** a *Confirmable* message.

An ACK **does not** indicate success/failure of any ***request*** encapsulated in the Confirmable message!

ACKNOWLEDGEMENT



An example of an Acknowledgement Message:
"To confirm that a specific Confirmable message arrived, the Server transmits back an Acknowledgement with the same Message ID from the corresponding Client."

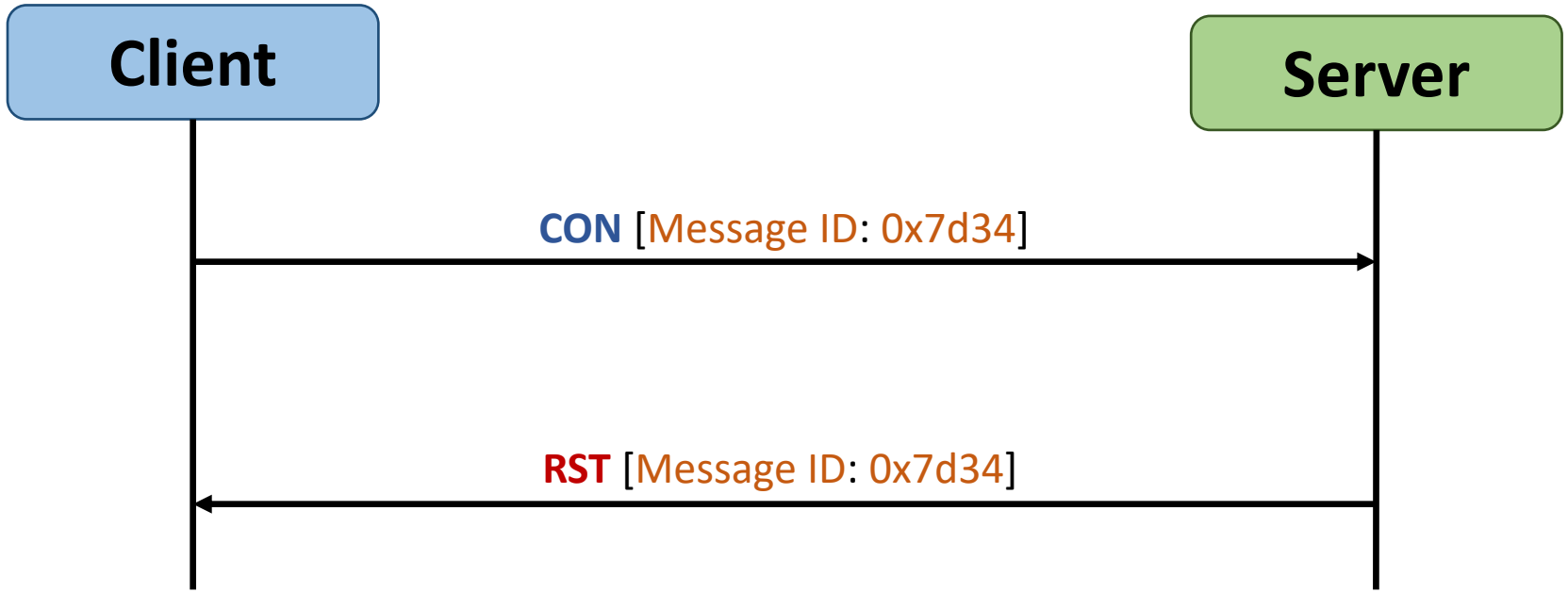
Source: ***RFC 7252***

Reset

Confirmable	• A message that **requires** an ACK.
Non-confirmable	• A message that **does not** require an ACK.
Acknowledgement	• An ACK **acknowledges** a *Confirmable* message.
Reset	• A Reset indicates that a Recipient **was not able to process** a Confirmable/Non-confirmable message.

For example, the receiving device has rebooted.

RESET



An example of a Reset Message:
"Here is a scenario where the Server is not at all able to process a Confirmable Message, and therefore, it replies with a Reset message instead of an Acknowledgement."

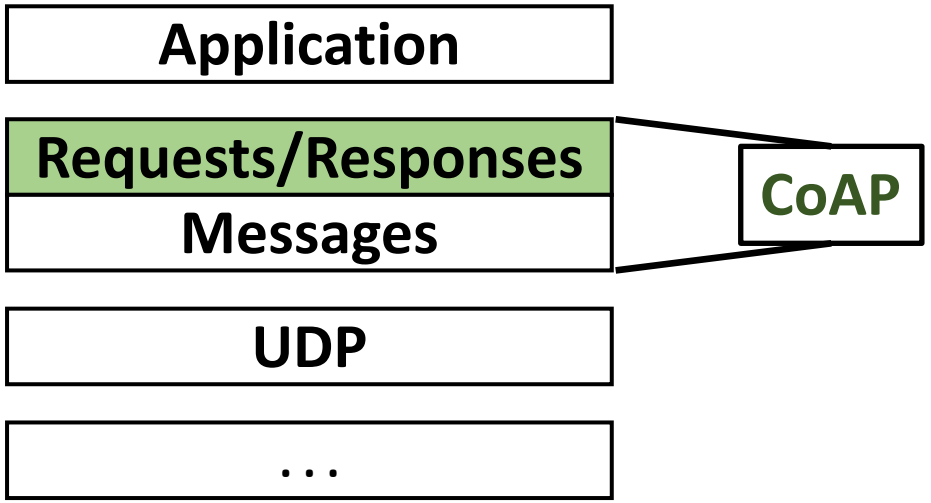
Source: ***RFC 7252***

6. CoAP Request/Response Model

Click me!

as defined in **RFC 7252**!

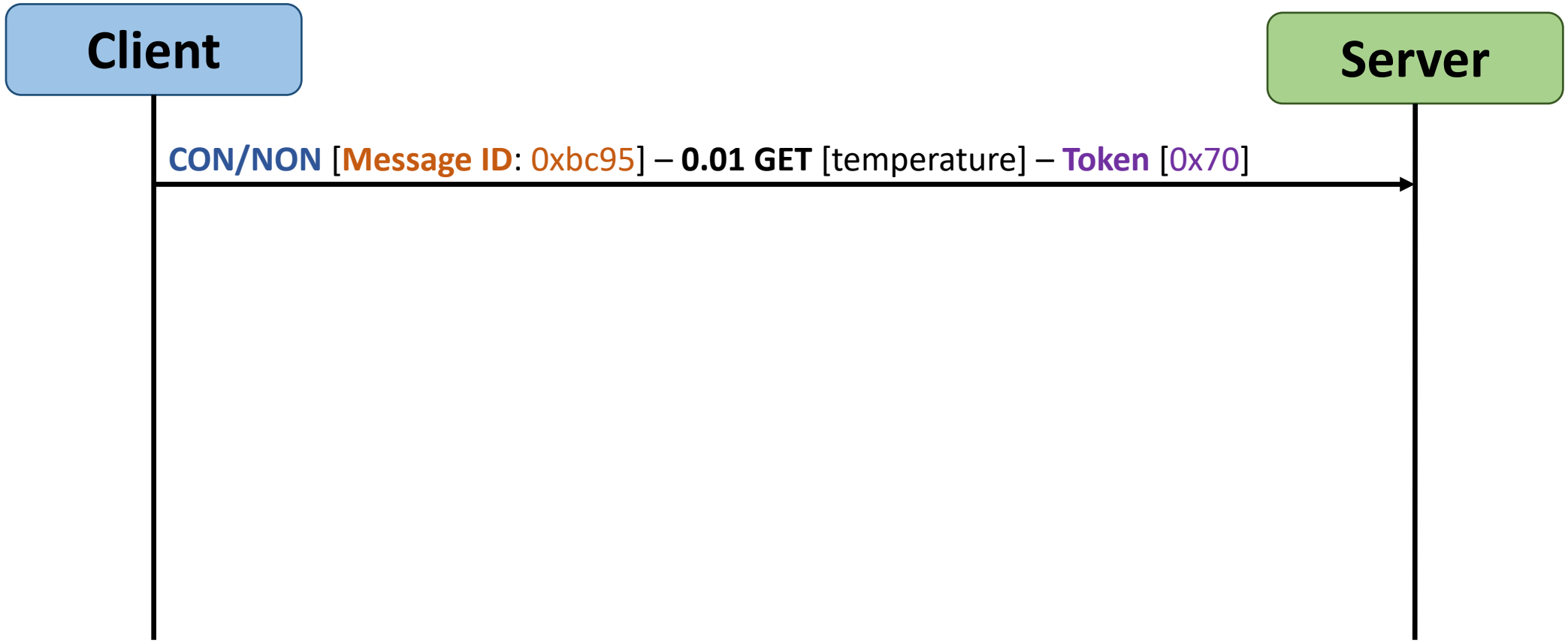
REQUEST/RESPONSE MODEL



Abstract Layering of CoAP.

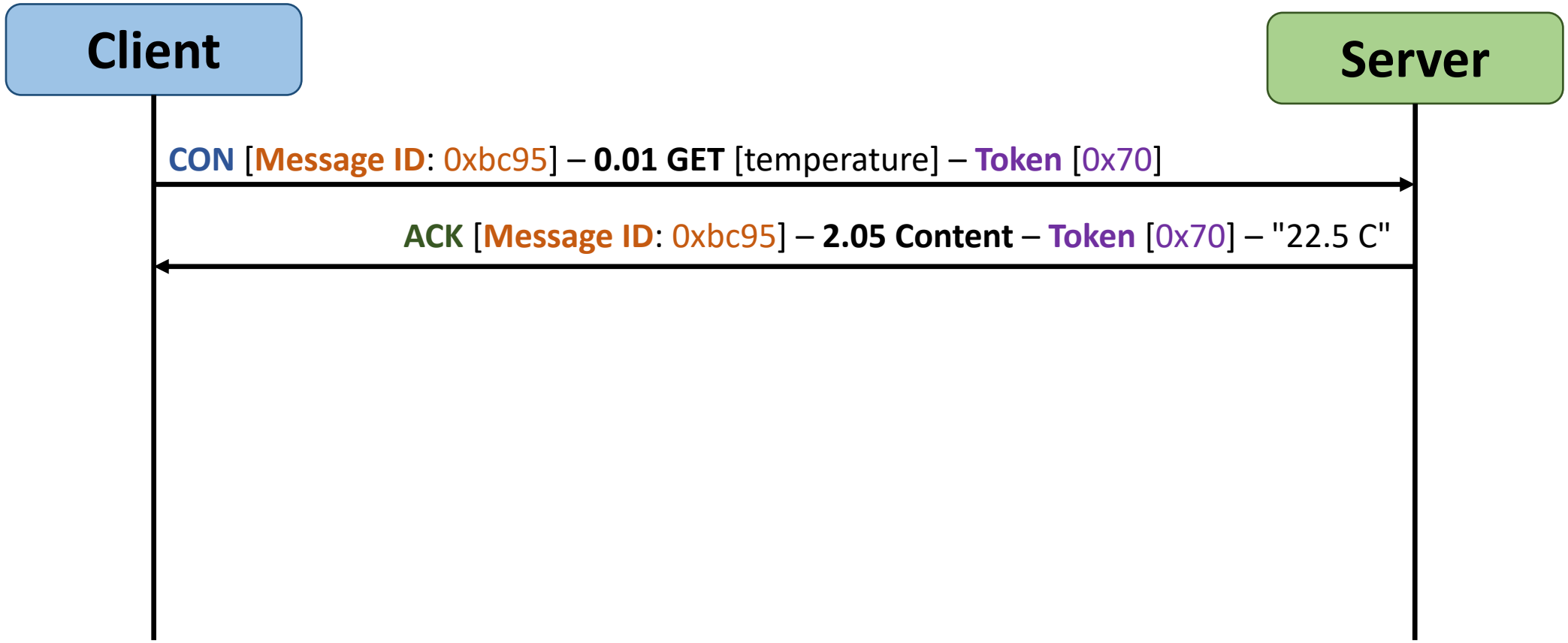
The CoAP Request/Response is the second layer in the CoAP abstraction layer.

REQUEST/RESPONSE MODEL



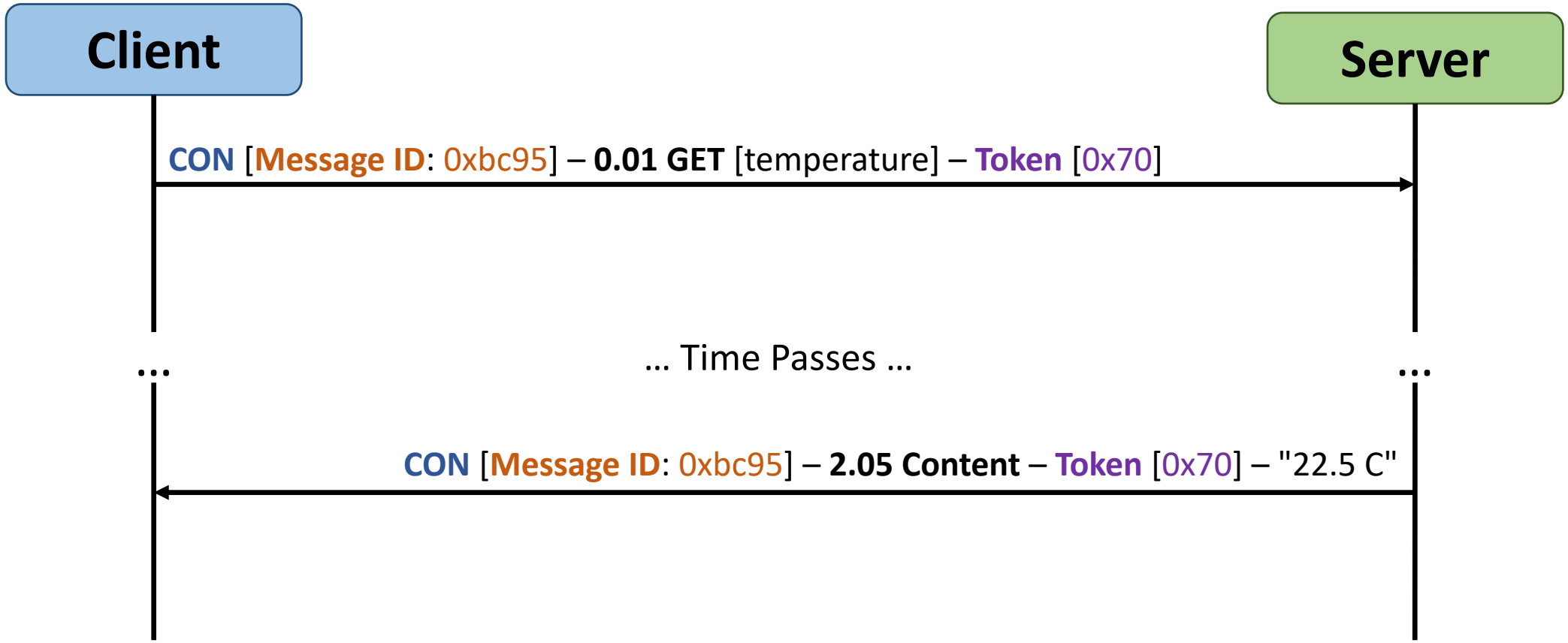
The Request is sent using a **Confirmable** or **Non-Confirmable** message.

REQUEST/RESPONSE MODEL



Then there are several scenarios depending on if the Server can respond immediately … OR …

REQUEST/RESPONSE MODEL



OR NOT to the Client's Request.

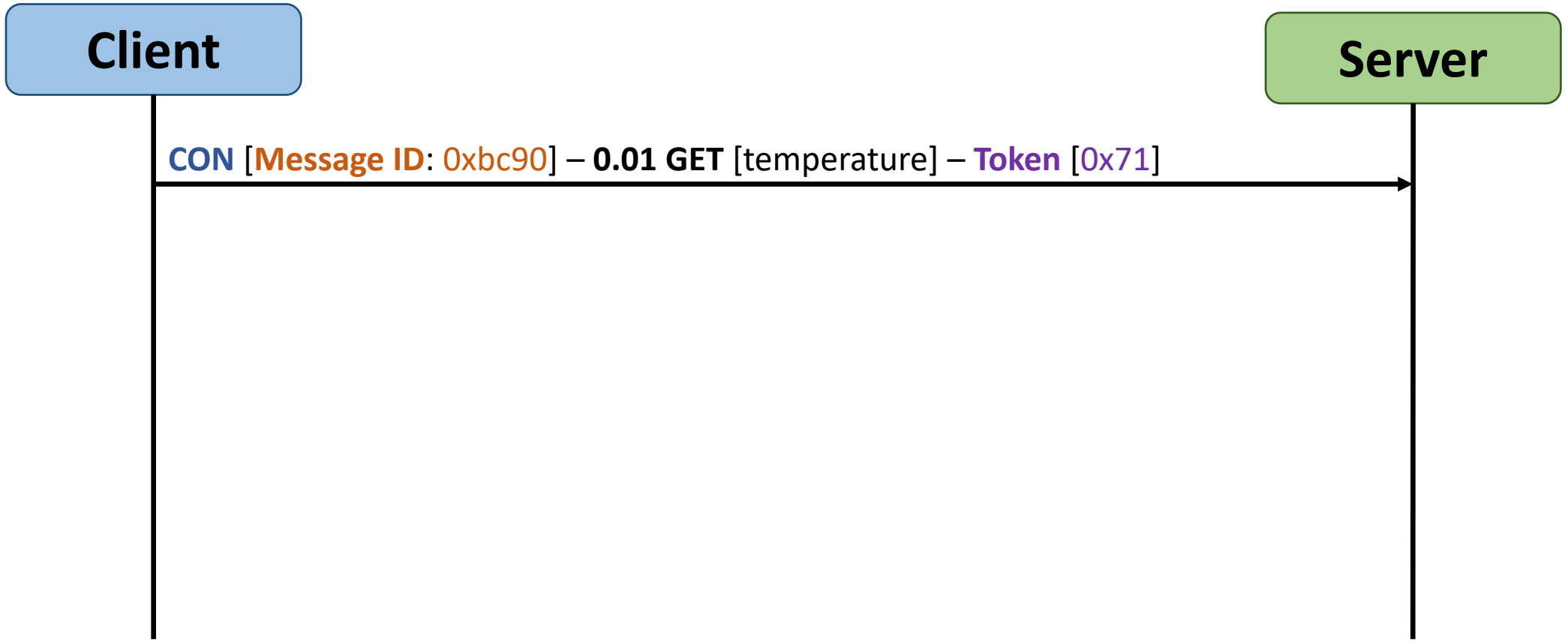
Request/Response Model

Piggybacked

- The *Response **is** immediately available*:
  - The **Respond is carried in** the **ACK** message.

Now, if the Response is immediately available, then it will be carried in the resulting Acknowledgement message.
→ This is called a **Piggybacked Response**.

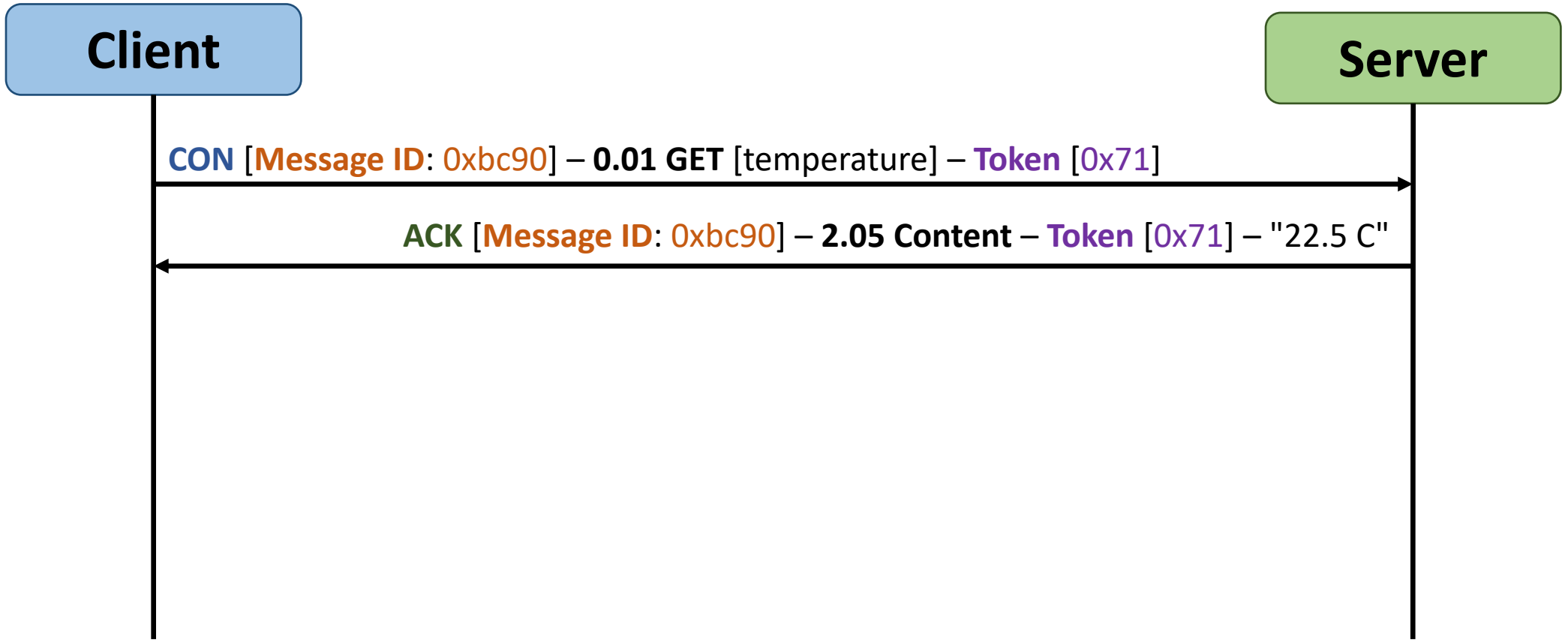
PIGGYBACKED RESPONSE [EXAMPLE 1]



A **Successful** example of a (temperature) GET Request with **Piggybacked Response**.

Source: ***RFC 7252***

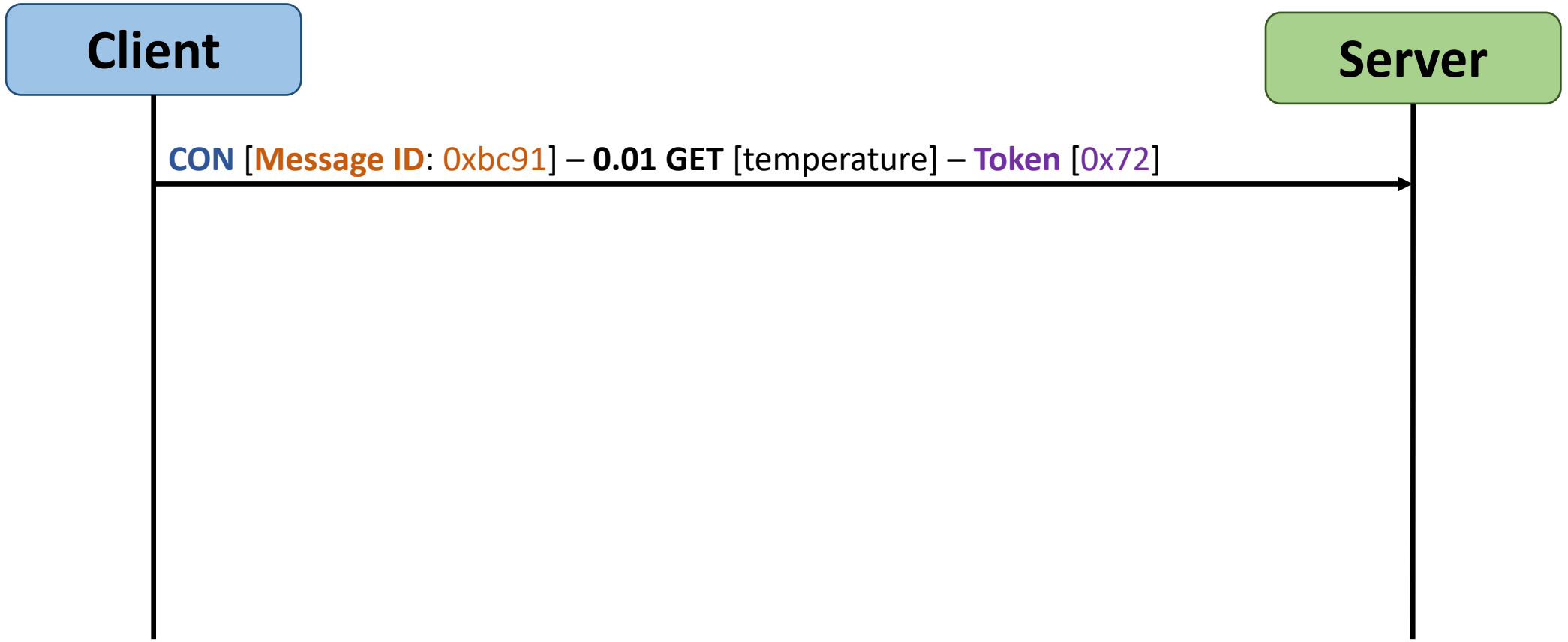
PIGGYBACKED RESPONSE [EXAMPLE 1]



The value of the temperature is immediately available, and therefore,
the server sends back to the client an Acknowledgement message containing the response.

Source: ***RFC 7252***

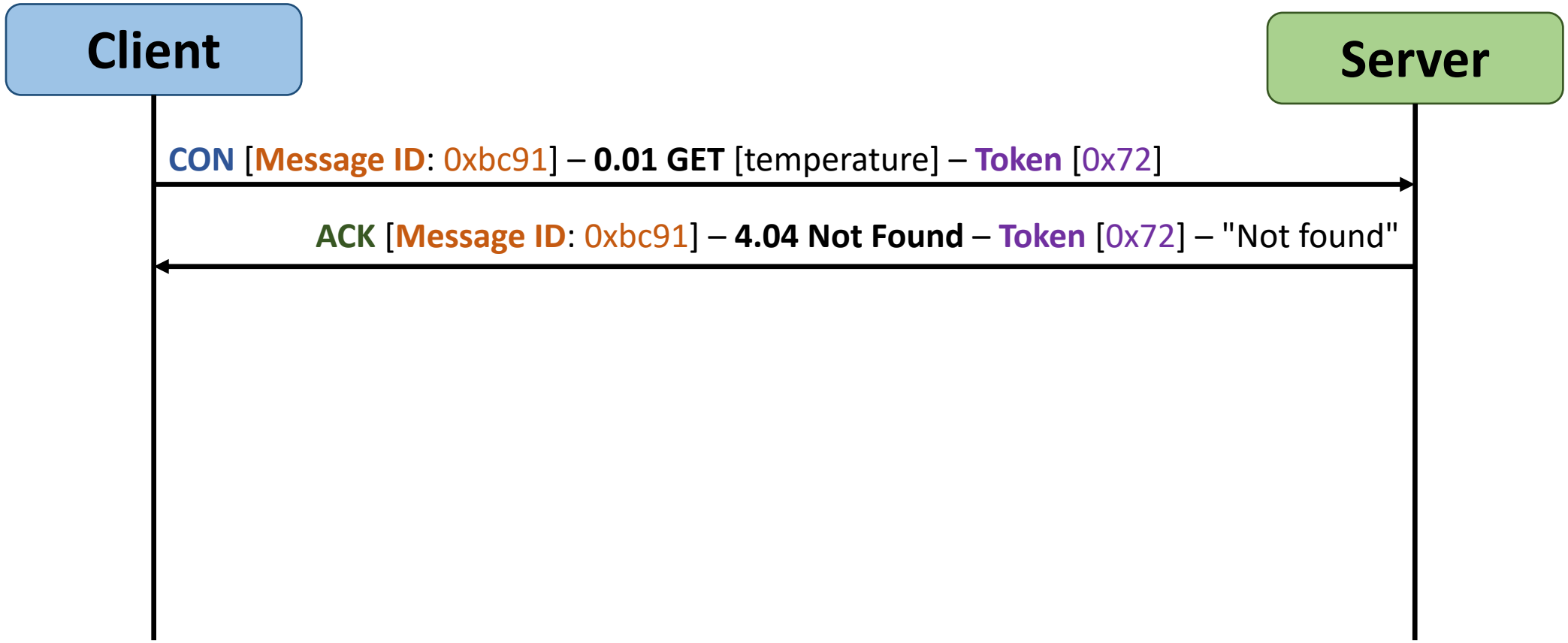
PIGGYBACKED RESPONSE [EXAMPLE 2]



A **Non-successful** example of a (temperature) GET Request with **Piggybacked Response**.

Source: ***RFC 7252***

PIGGYBACKED RESPONSE [EXAMPLE 2]



In this case, the server sends back to the client an Acknowledgement message containing the **Response Code Client Error** of **4 dot 04 Not Found**, which is like the HTTP 404 "Not Found".

Source: ***RFC 7252***

Request/Response Model

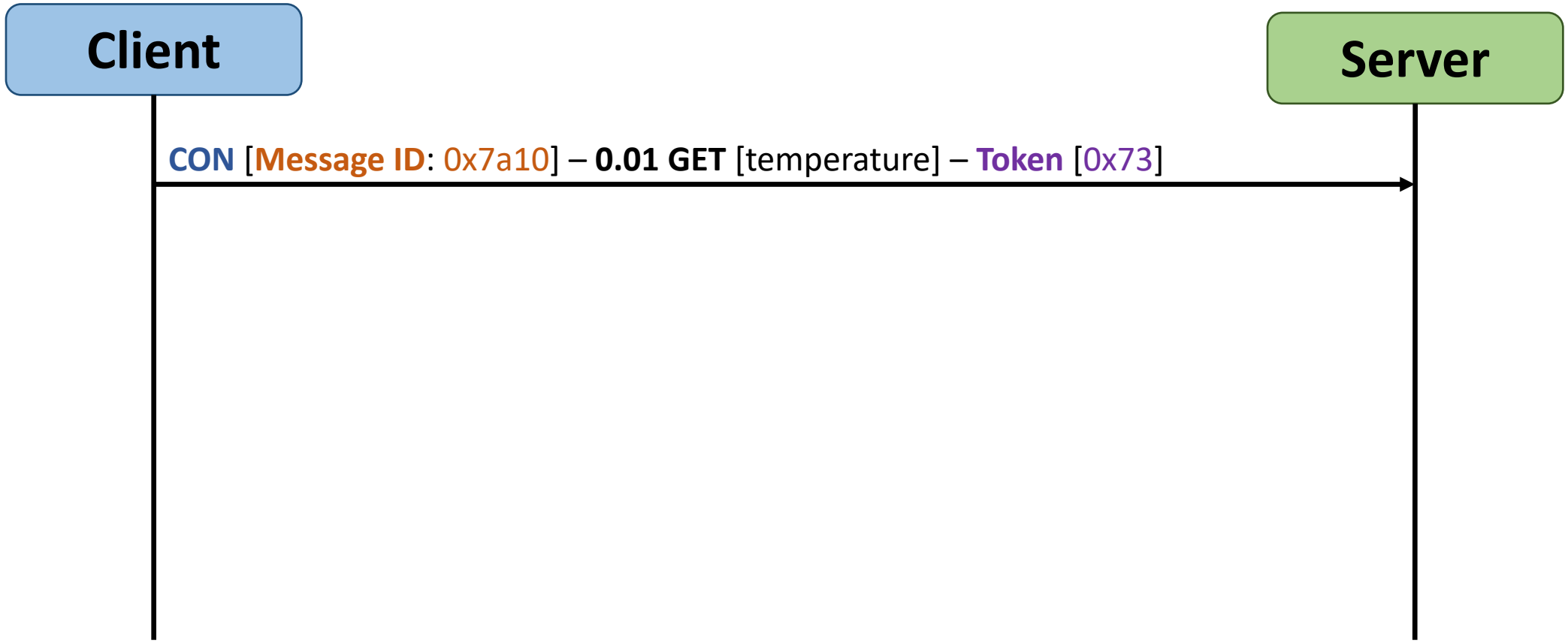
Piggybacked

- The *Response* ***is*** *immediately available*:
  - The **Respond is carried in** the **ACK** message.

Separate

- The *Response* ***is not*** *immediately available*:
  - The Server sends an Empty ACK.
  - Then, the **Respond is sent in** a **NEW CON** message!

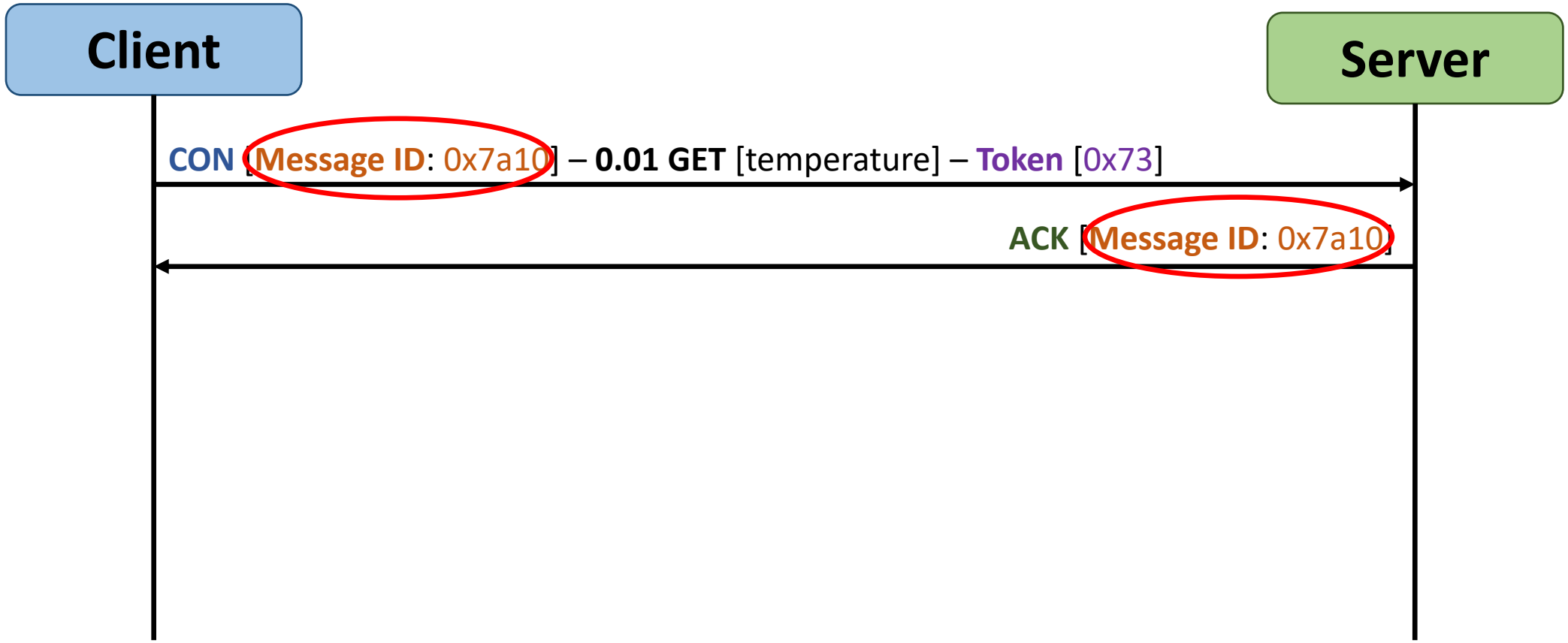
SEPARATE RESPONSE



An example of a GET Request with a **Separate Response**.

Source: ***RFC 7252***

SEPARATE RESPONSE



In this example, the **server is not able to respond immediately**. Therefore, it responds with an **Empty Acknowledgement message** in order the client to avoid retransmitting the Request.

Source: ***RFC 7252***

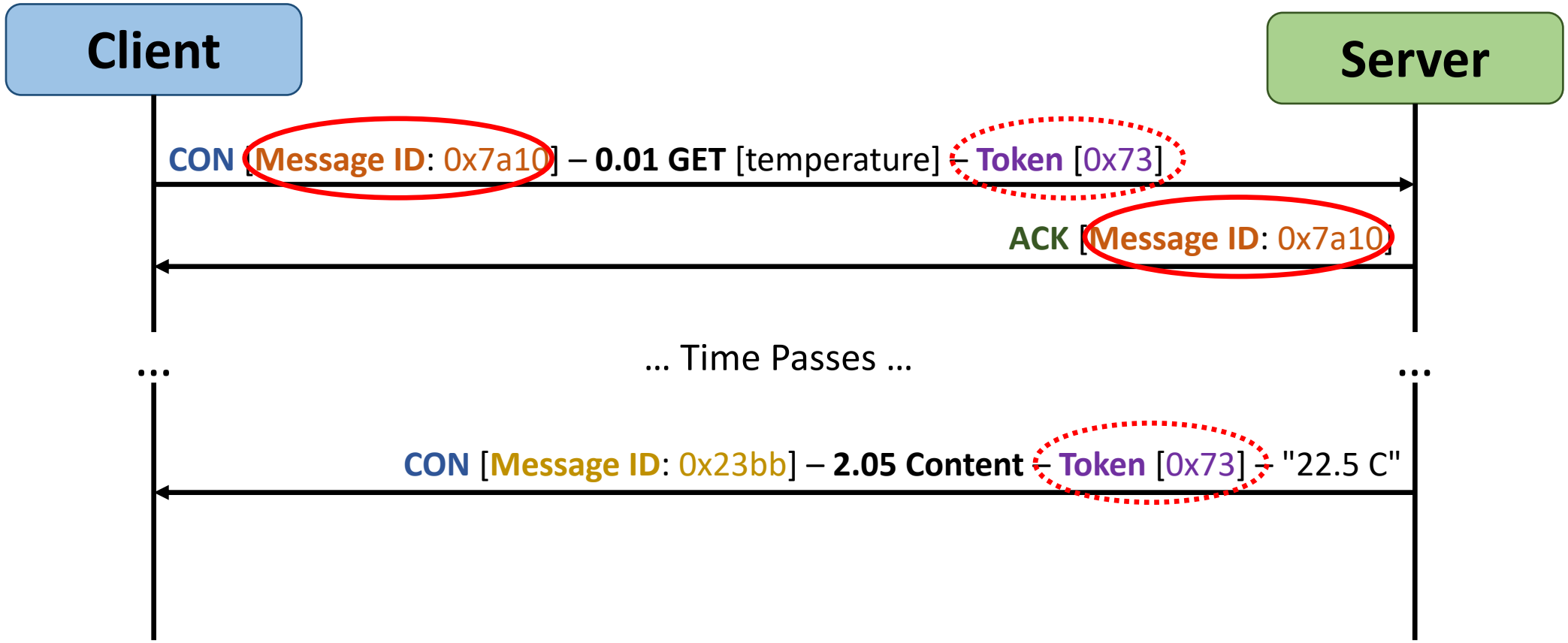
SEPARATE RESPONSE

ATTENTION!!!

- **Message ID** detects Message Duplication & enables Reliability.
- **Token** correlates Requests to Responses.

Source: ***RFC 7252***

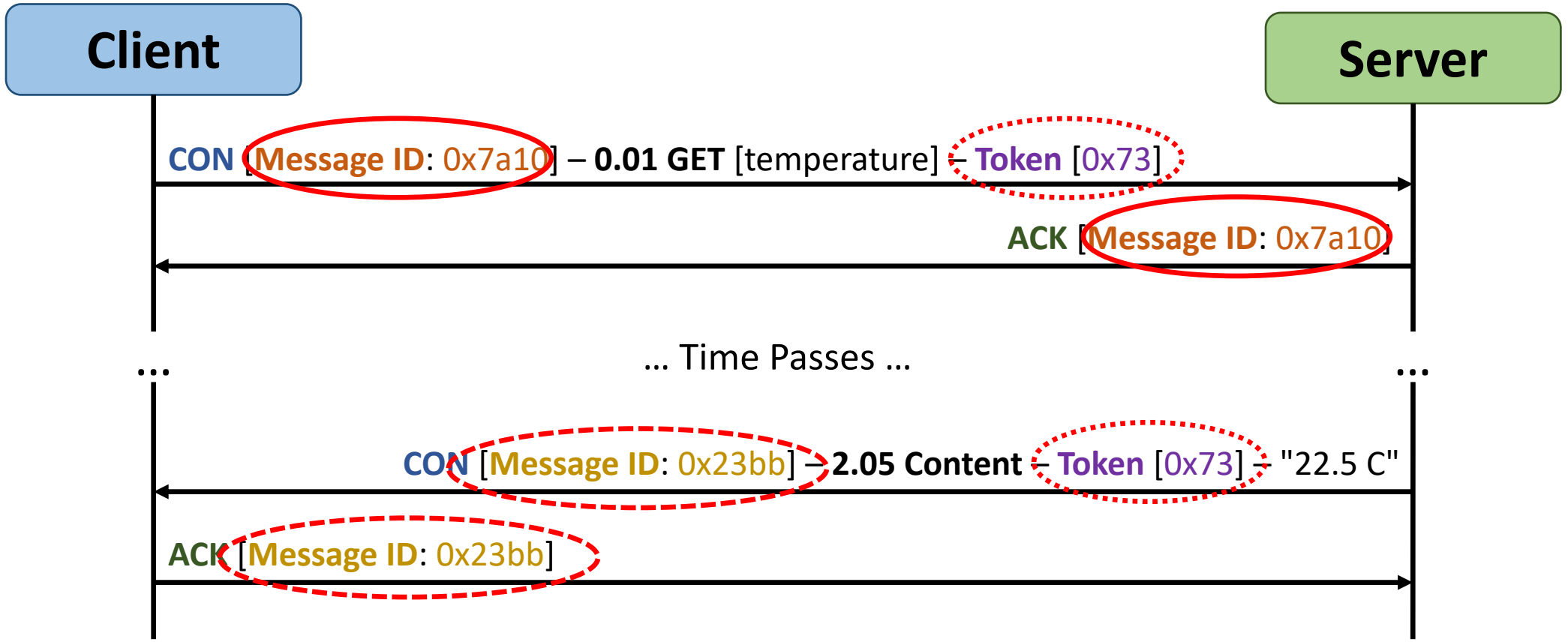
SEPARATE RESPONSE



Once the response is ready, the server sends it in a **New Confirmable message without modifying the Token Value**!

Source: ***RFC 7252***

SEPARATE RESPONSE



This **new Confirmable Message**, in turn needs to be acknowledged by the client.

Source: ***RFC 7252***

REQUEST/RESPONSE MODEL

Piggybacked

- The *Response* ***is*** *immediately available*:
  - The **Respond is carried in** the **ACK** message.

Separate

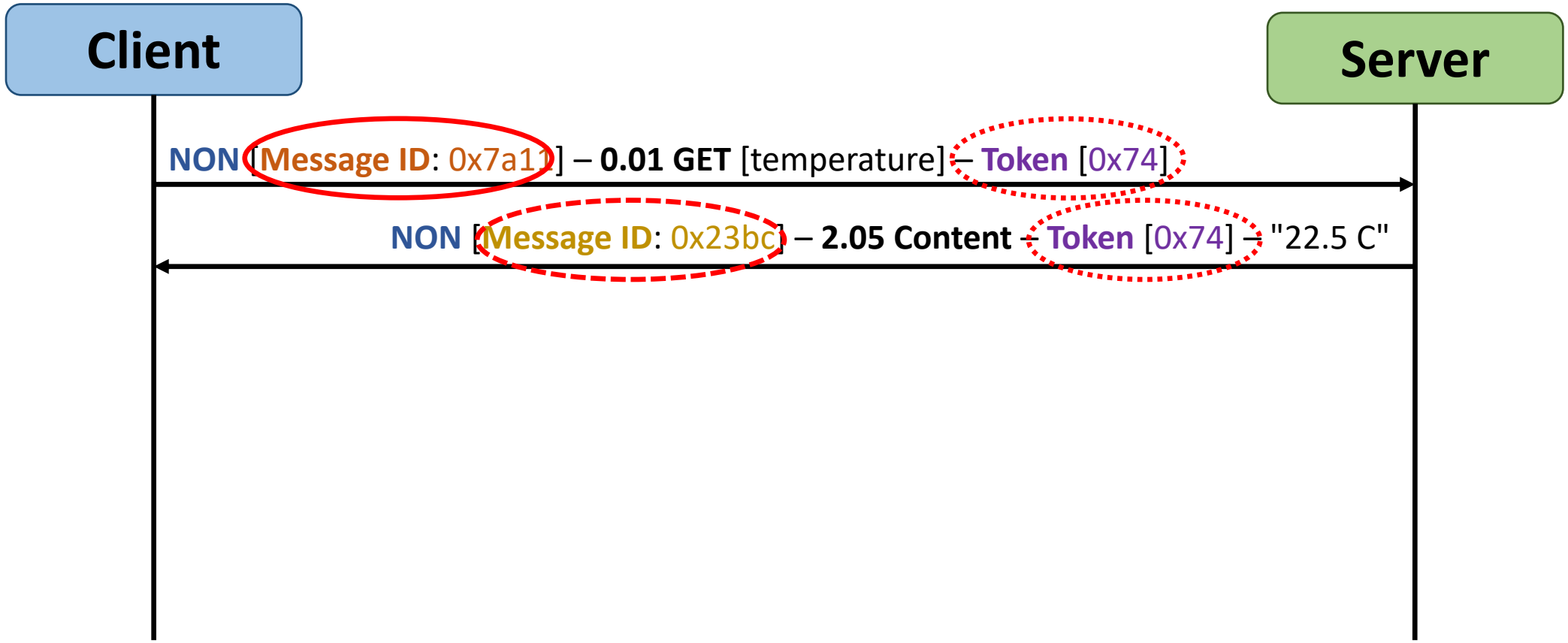
- The *Response* ***is not*** *immediately available*:
  - The Server sends an Empty ACK.
  - Then, the **Respond is sent in** a **NEW CON** message!

Non-confirmable

- Request/Response sent in Non-confirmable Messages

If a Request is sent in a Non-confirmable message, then the response is sent in a new Non-confirmable message as well.

Request/Response sent in Non-confirmable Messages



An example of a **Request** and a **Response carried** in **Non-confirmable** Messages.

Source: ***RFC 7252***

CoAP Tutorial

as defined in **RFC 7252**!

Georgios Z. PAPADOPOULOS, PhD, HDR
Professor at IMT Atlantique, campus of Rennes, France