Introduction to CoAP

15CSE480-Internet of Things

Objective

- What Is CoAP Protocol?
- The main features of CoAP protocols
- Http Vs. CoAP protocols
- CoAP Structure Model
- Message Layer model

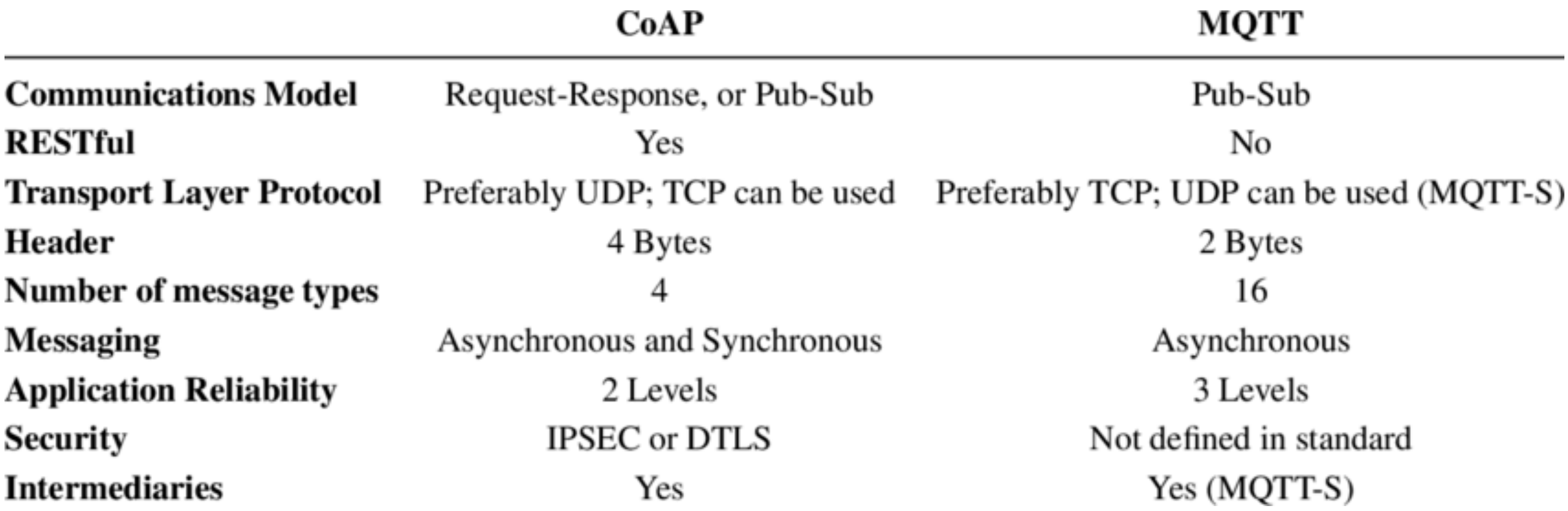
What Is CoAP Protocol?

- CoAP is an IoT protocol.
- CoAP stands for Constrained Application Protocol
- CoAP is a simple protocol with low overhead specifically designed for constrained devices (such as microcontrollers) and constrained networks.
- This protocol is used in Machine to Machine (M2M) data exchange and is very similar to HTTP.

The main features of CoAP protocols

- It is open IETF standard
- It is Embedded web transfer protocol (coap://)
- It uses asynchronous transaction model.
- GET, POST, PUT and DELETE methods are used.
- It uses small and simple 4 byte header.
- Supports binding to UDP, SMS and TCP.
- Datagram Transport Layer Security (**DTLS**) based Pre-Shared Key (**PSK**), Raw Public Key (**RPK**) and certificate security is used.
- Uses subset of MIME types and HTTP response codes.
- Uses built in discovery mechanism.

CoAP vs. MQTT

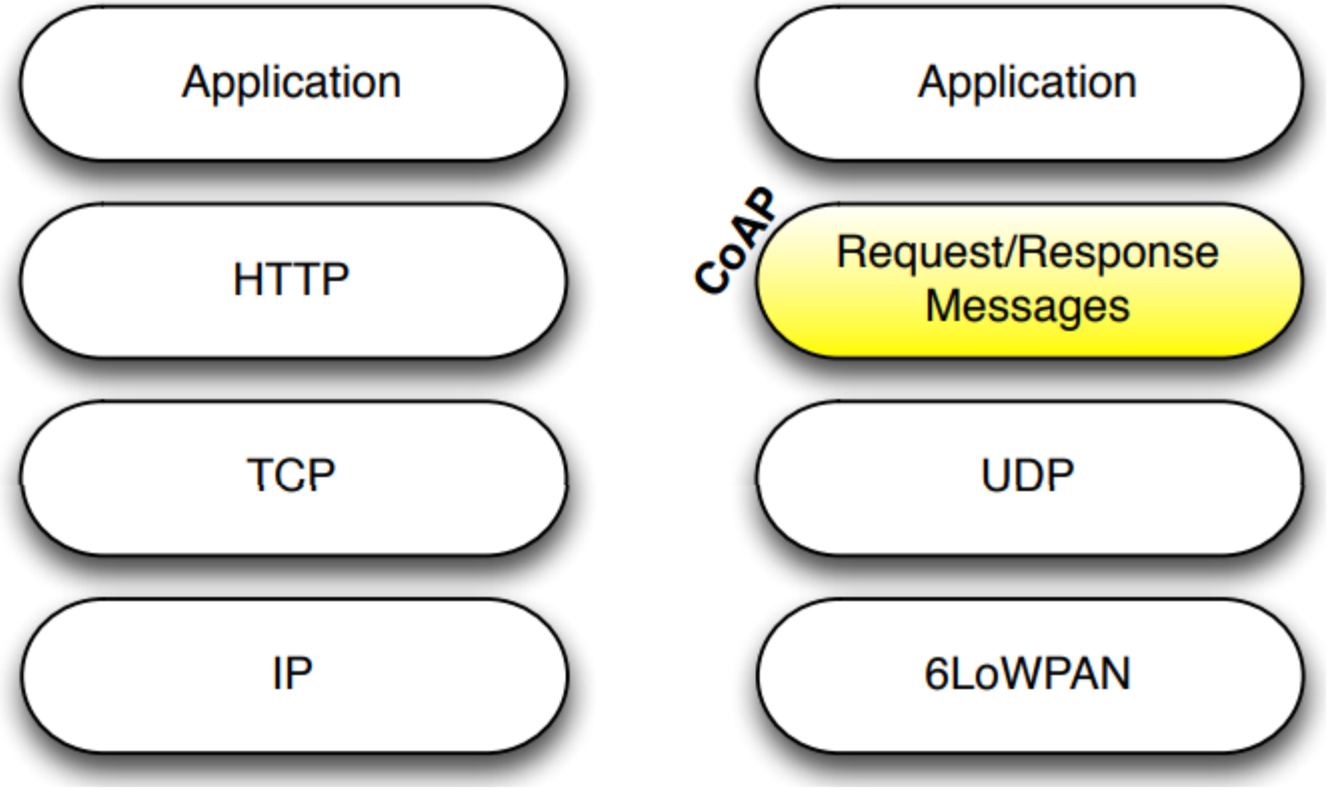


	CoAP	MQTT
Communications Model	Request-Response, or Pub-Sub	Pub-Sub
RESTful	Yes	No
Transport Layer Protocol	Preferably UDP; TCP can be used	Preferably TCP; UDP can be used (MQTT-S)
Header	4 Bytes	2 Bytes
Number of message types	4	16
Messaging	Asynchronous and Synchronous	Asynchronous
Application Reliability	2 Levels	3 Levels
Security	IPSEC or DTLS	Not defined in standard
Intermediaries	Yes	Yes (MQTT-S)

CoAP vs. HTTP

Feature	CoAP	HTTP
Protocol	It uses UDP, TCP can be used	It uses TCP.
Network layer	It uses IPv6 along with 6LoWPAN.	It uses IP layer.
Multicast support	It supports.	It does not support.
Architecture model	CoAP uses both client-Server & Publish-Subscribe models.	HTTP uses client and server architecture.
Synchronous communication	CoAP does not need this.	HTTP needs this.
Overhead	Less overhead and it is simple.	More overhead compare to CoAP and it is complex.
Application	Designed for resource constrained networking devices such as WSN/IoT/M2M.	Designed for internet devices where there is no issue of any resources.

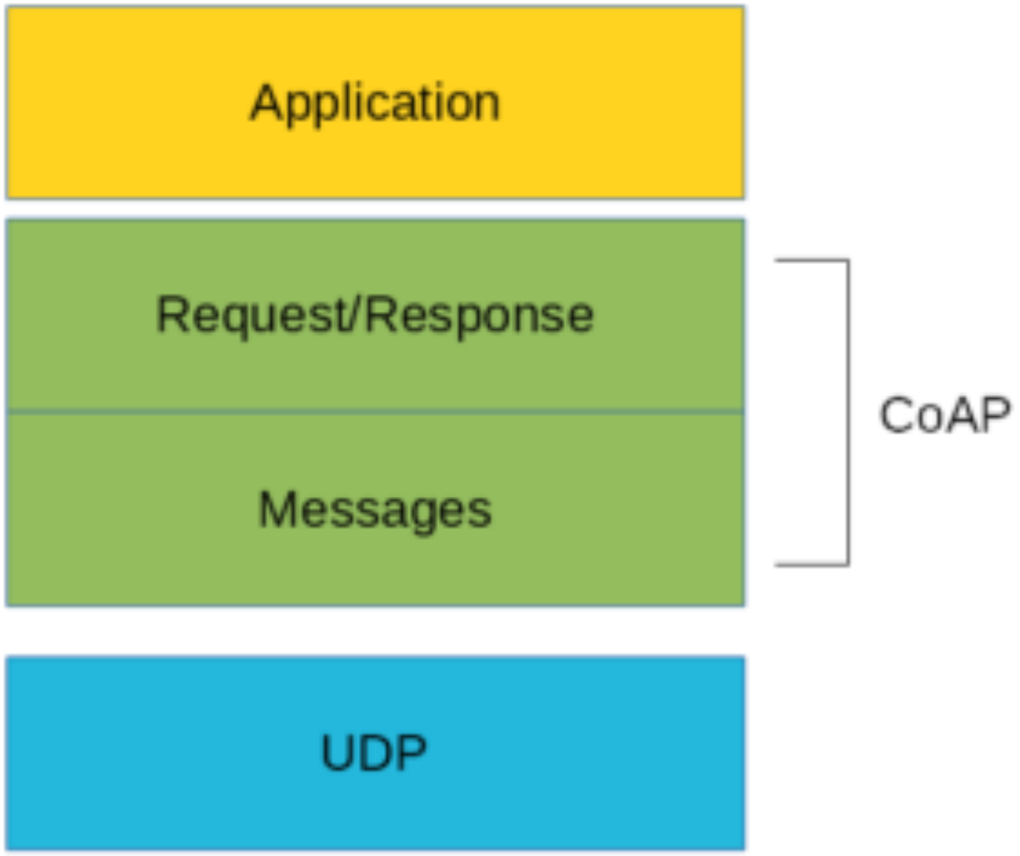
CoAP vs. HTTP


Application
HTTP
TCP
IP
Application
CoAP
Request/Response Messages
UDP
6LoWPAN

Terminology

- Before going deeper into the CoAp protocol, structure is useful to define some terms that we will use later:
- **Endpoint**: An entity that participates in the CoAP protocol. Usually, an Endpoint is identified with a host
- **Sender**: The entity that sends a message
- **Recipient**: The destination of a message
- **Client**: The entity that sends a request and the destination of the response
- **Server**: The entity that receives a request from a client and sends back a response to the client

CoAP Structure Model



CoAP Structure Model

- CoAP employs a two layers structure:
  - Messages
  - Request/Response
- The Messages layer deals with UDP and with asynchronous messages.
- The Request/Response layer manages request/ response interaction based on request/response messages.

Message Layer model

- CoAP supports four different message types:
  - CON (Confirmable)
  - NON (Non-confirmable)
  - ACK (Acknowledgment)
  - RST (Reset)

- **CoAP Messages Model**
- This is the lowest layer of CoAP.
- This layer deals with UDP exchanging messages between endpoints.
- Each CoAP message has a unique ID

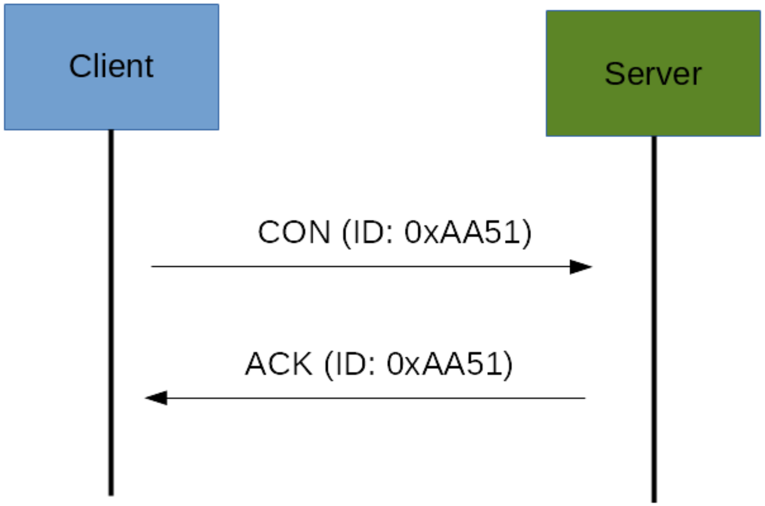
Message Layer model

- A CoAP message is built by these parts:
  - A binary header
  - A compact options
  - Payload
- CoAP protocol uses two kinds of messages:
  - Confirmable message
  - Non-confirmable message

Message Layer model: A confirmable message

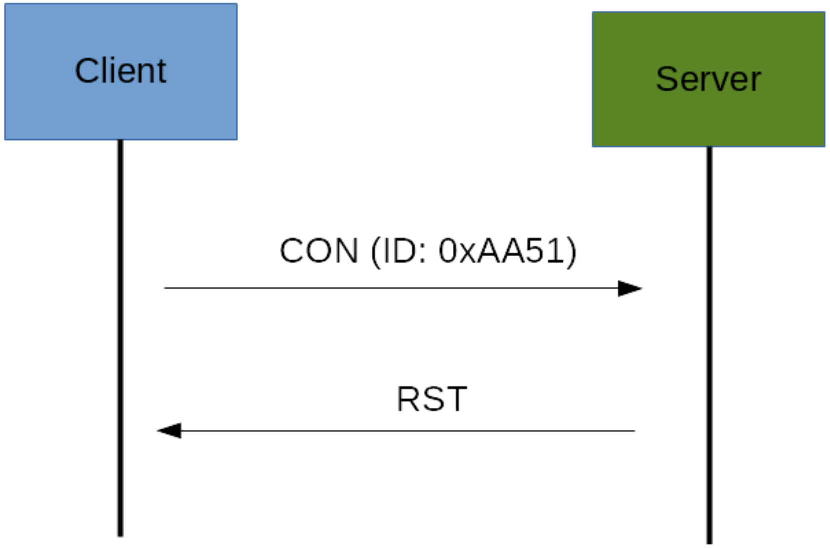
- A confirmable message is a reliable message. When exchanging messages between two endpoints, these messages can be reliable.
- In CoAP, a reliable message is obtained using a Confirmable message (CON).
- Using this kind of message, the client can be sure that the message will arrive at the server.
- A Confirmable message is sent again and again until the other party sends an acknowledge message (ACK).
- The ACK message contains the same ID of the confirmable message (CON).

Message Layer model: A confirmable message



The picture shows the message exchange process

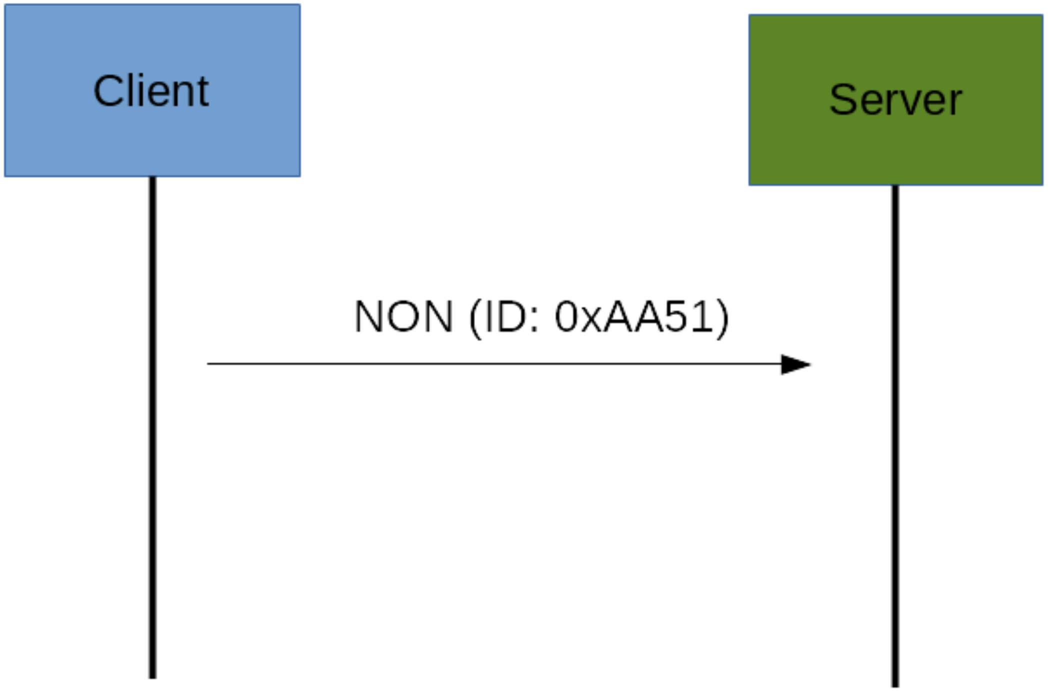
If the server has troubles managing the incoming request, it can send back a Rest message (RST) instead of the Acknowledge message (ACK)



Message Layer model: A Non-confirmable message

- The other message category is the Non-confirmable (NON) messages.
- These are messages that don’t require an Acknowledge by the server.
- They are unreliable messages or in other words messages that do not contain critical information that must be delivered to the server.
- It doesn't need to be ACKed, but has to contain message ID for supervising in case of retransmission
- To this category belongs messages that contain values read from sensors.

Message Layer model: A Non-confirmable message



CoAp Request/Response Model

- The CoAP Request/Response is the second layer in the CoAP abstraction layer.
- The request is sent using a Confirmable (CON) or Non-Confirmable (NON) message.
- There are several scenarios depending on if the server can answer immediately to the client request or the answer if not available.
  - Piggy-backed
  - Separate response
  - Non confirmable request and response

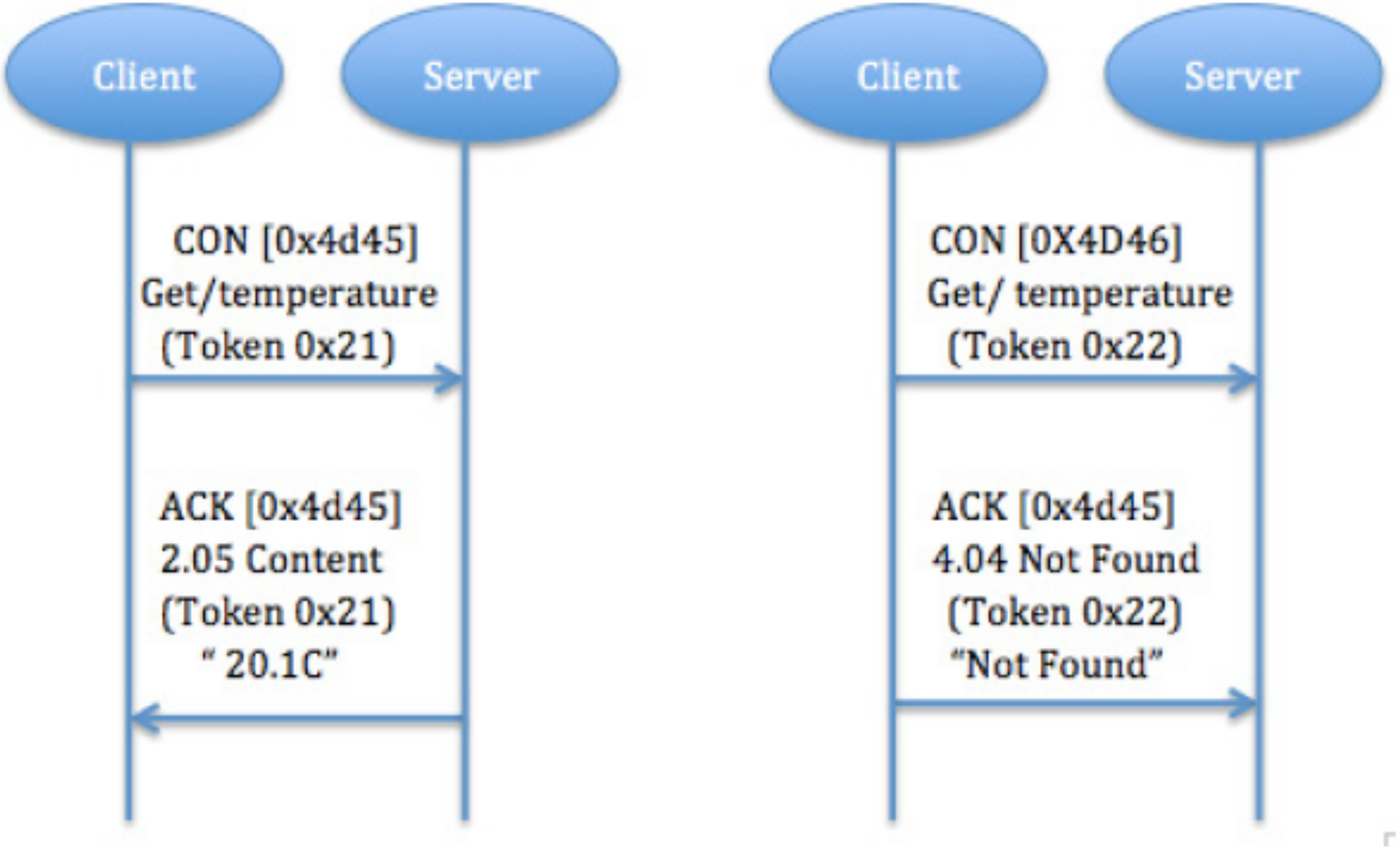
CoAp Request/Response Model

Piggy-backed

- Client sends request using CON type message and receives response
- ACK with confirmable message immediately for successful response, ACK contain response message (identify by using token),
- for failure response, ACK contain failure response code.
- The Token is different from the Message-ID and it is used to match the request and the response.

CoAp Request/Response Model

Piggy-backed

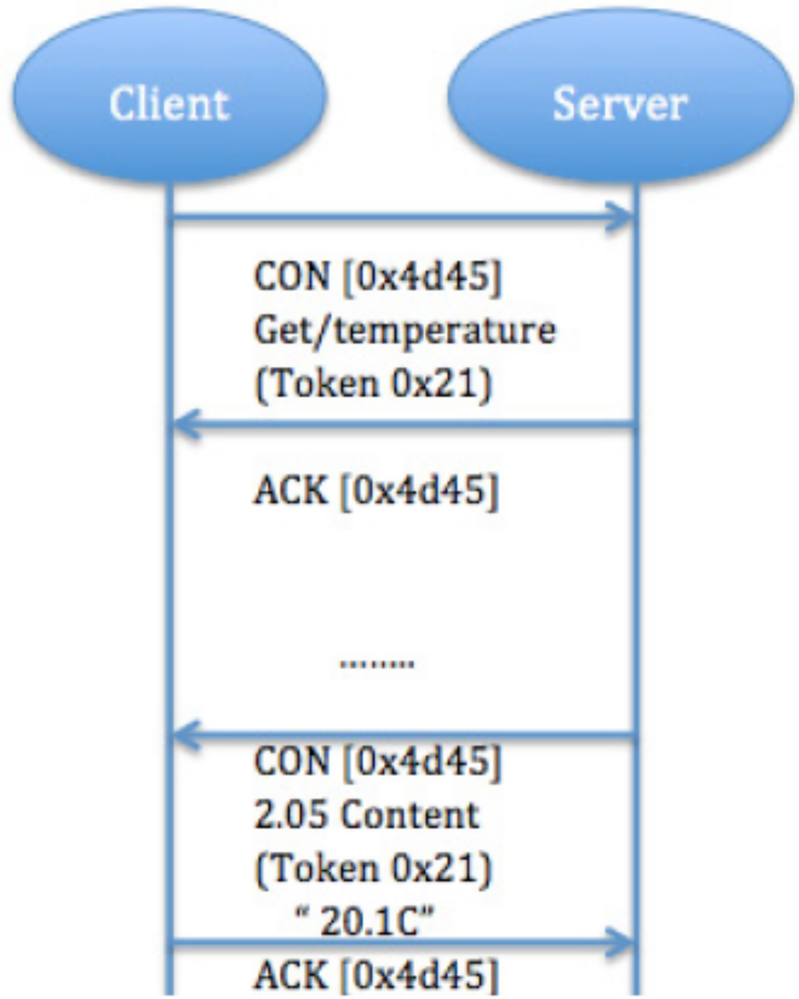


CoAp Request/Response Model

Separate response

- If server receive a CON type message but not able to response this request immediately, it will send an empty ACK in case of client resend this message.
- When server ready to response this request, it will send a new CON to client and client reply a confirmable message with acknowledgment.
- ACK is just to confirm CON message, no matter CON message carry request or response

CoAp Request/Response Model



CON [0x4d45]
Get/temperature
(Token 0x21)

ACK [0x4d45]

....

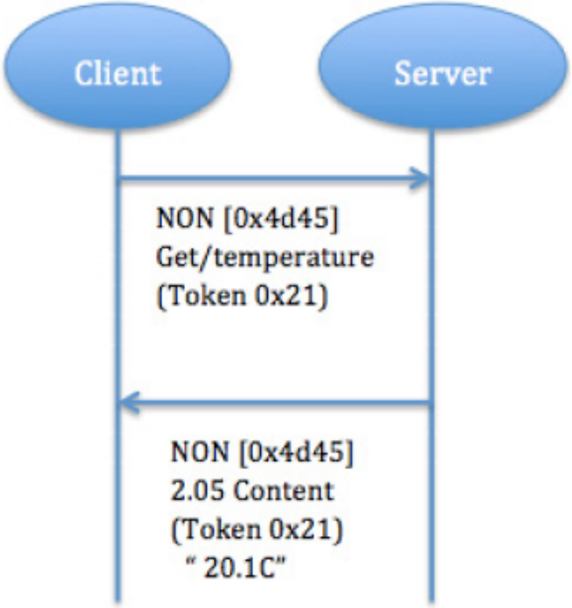
CON [0x4d45]
2.05 Content
(Token 0x21)
" 20.1C"

ACK [0x4d45]

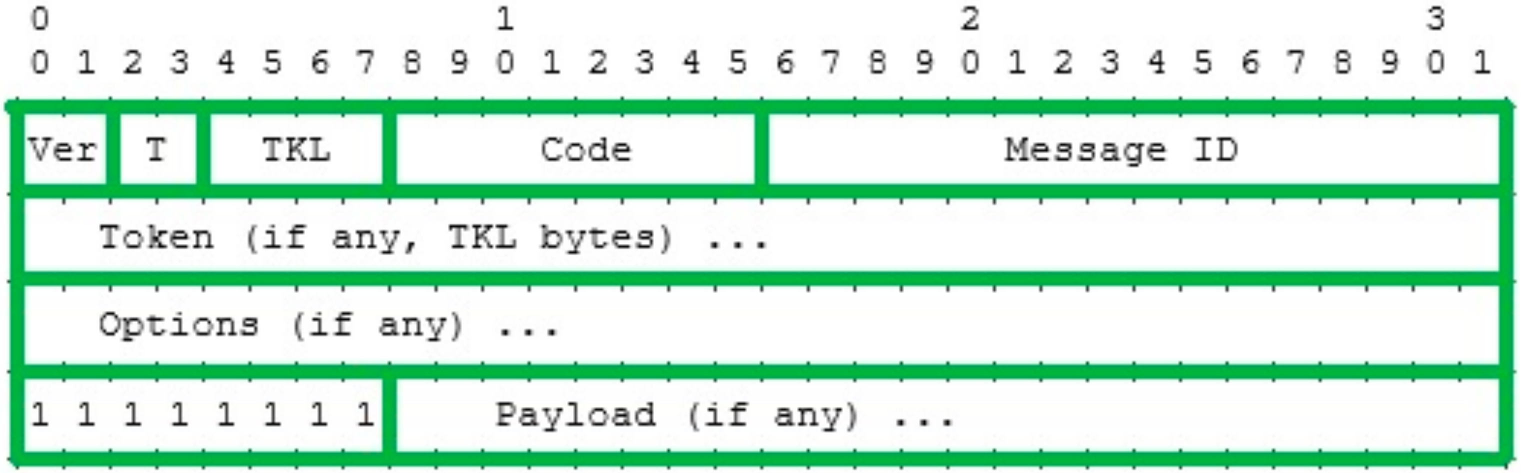
CoAp Request/Response Model

Non confirmable request and response

- Unlike Piggy-backed response carry confirmable message, in Non confirmable request client send NON type message indicate that Server don't need to confirm.
- Server will resend a NON type message with response



CoAP Message Format



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
|Ver| T |  TKL  |      Code     |          Message ID           |
|   Token (if any, TKL bytes) ...                               |
|   Options (if any) ...                                        |
|1 1 1 1 1 1 1 1|    Payload (if any) ...                       |
```

CoAP Message Format

Ver - Version (1) Indicates the version of CoAP

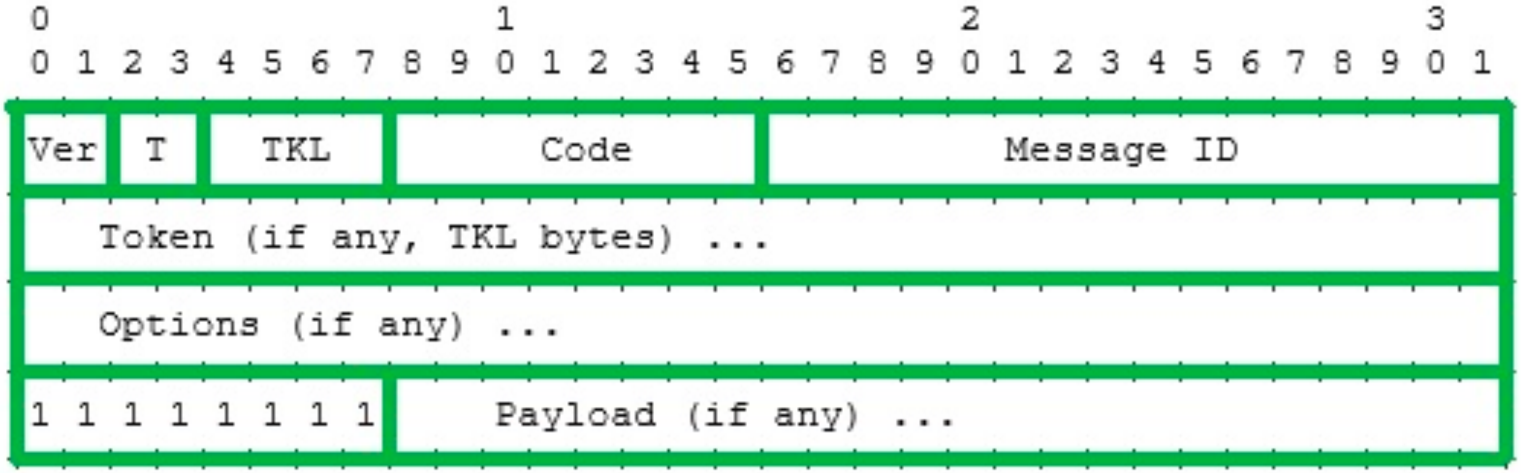
T – Message Type (Confirmable, Non-Confirmable, Acknowledgement, Reset)

TKL- Token Length, if any, the number of Token bytes after this header

Code - Request Method or Response Code

Message ID – 16-bit identifier for matching responses

Token – Optional response matching token

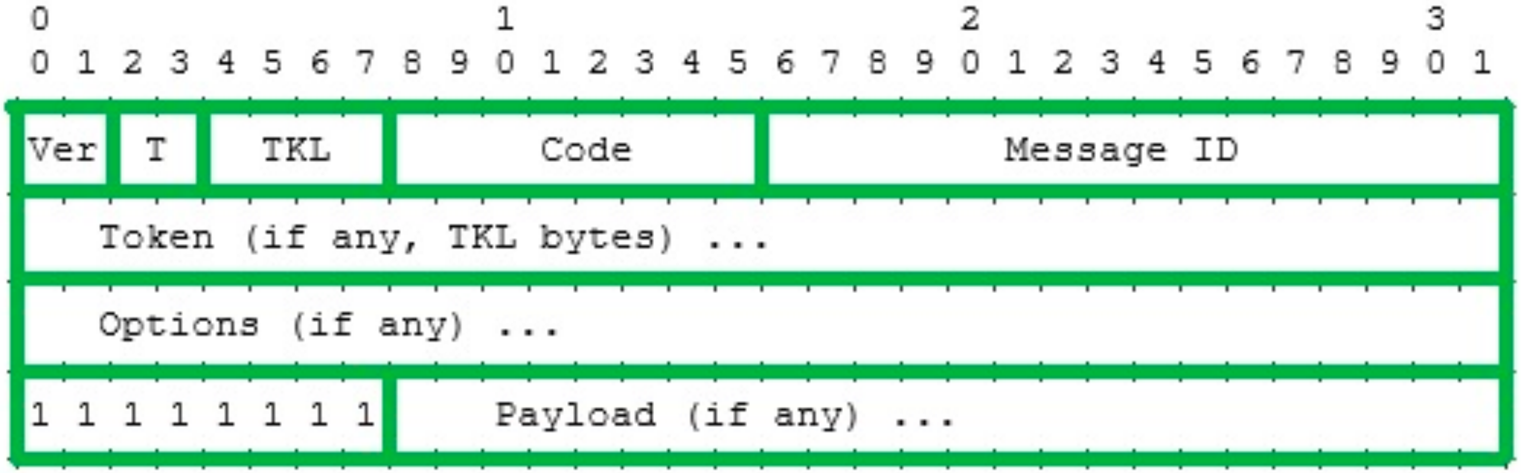


```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
|Ver| T |  TKL  |      Code     |          Message ID           |
   Token (if any, TKL bytes) ...
   Options (if any) ...
|1 1 1 1 1 1 1 1|    Payload (if any) ...
```

CoAP Message Format

OC/TKL:

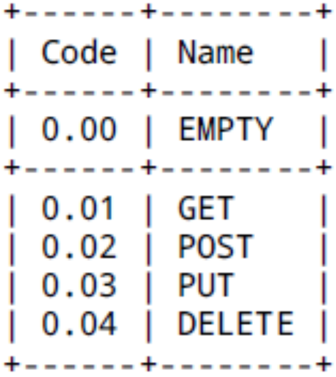
- 4-bit unsigned integer Option Count field. Indicates if there are Option Headers following the base header.
- If set to 0 the payload (if any) immediately follows the base header.
- If greater than zero the field indicates the number of options to immediately follow the header.



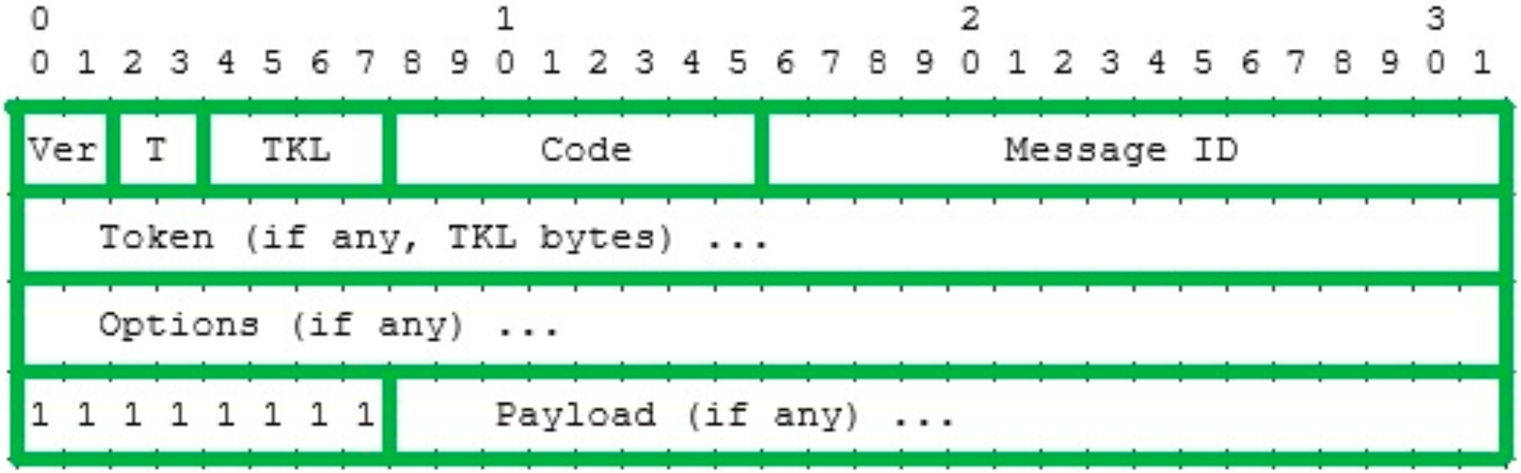
CoAP Message Format

Code:

- 8-bit unsigned integer. This field indicates the Method or Response Code of a message.
- The value 0 indicates no code.
- The values 1-10 are used for Method Codes
- The values 11-39 are reserved for future use.
- The values 40-255 are used for Response Codes



Code	Name
0.00	EMPTY
0.01	GET
0.02	POST
0.03	PUT
0.04	DELETE



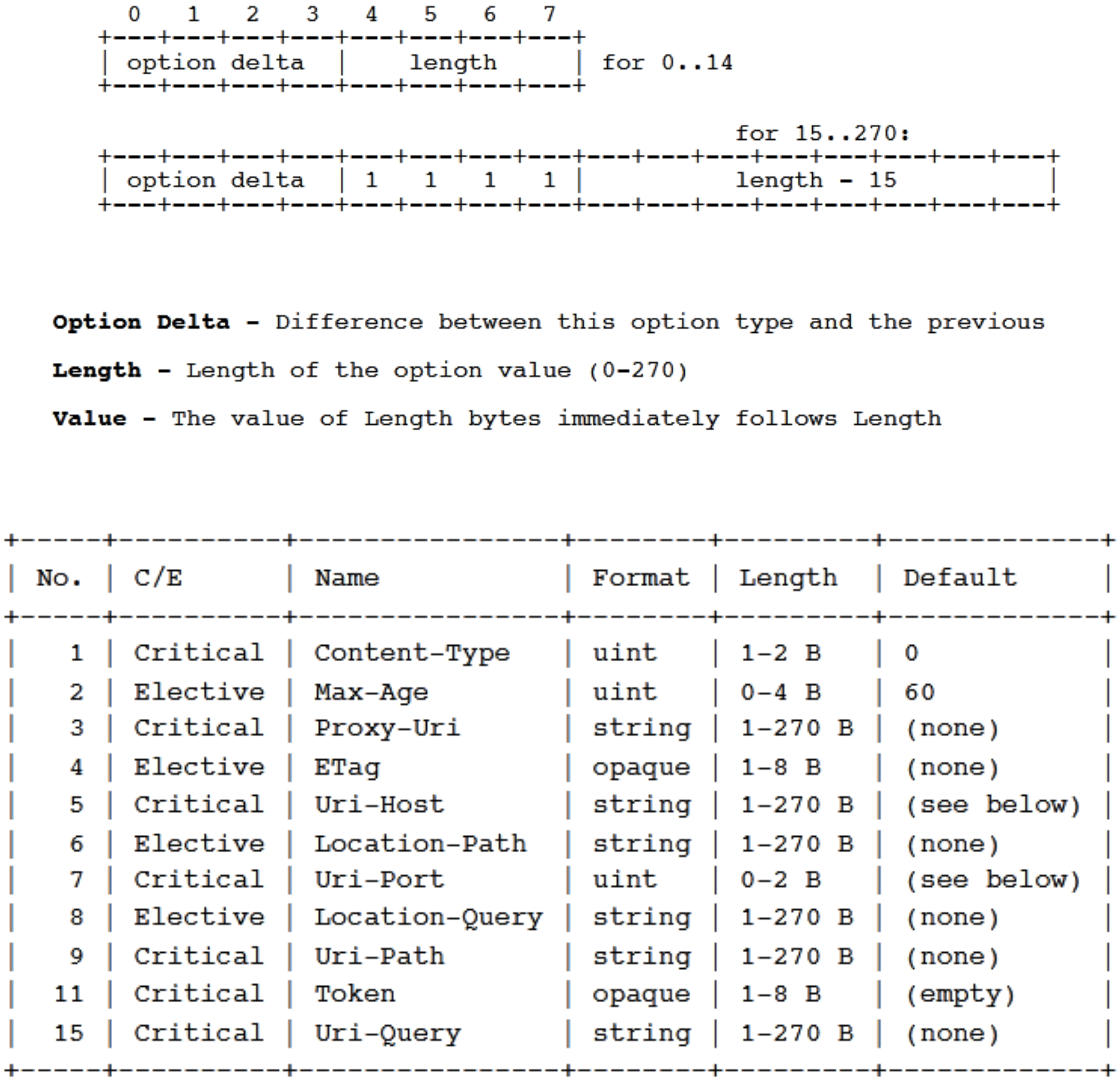
```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
|Ver| T |  TKL  |      Code     |          Message ID           |
   Token (if any, TKL bytes) ...
   Options (if any) ...
|1 1 1 1 1 1 1 1|    Payload (if any) ...
```

CoAP Message Format

Transaction ID:

- 16-bit unsigned integer. A unique Transaction ID assigned by the source and used to match responses.
- The Transaction ID MUST be changed for each new request (regardless of the end-point)
- MUST NOT be changed when retransmitting a request

Option Header



```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| option delta  |    length     | for 0..14
+---+---+---+---+---+---+---+---+

                                    for 15..270:
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| option delta  | 1   1   1   1 |          length - 15          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

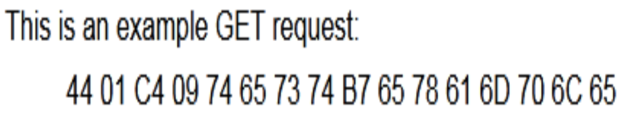
Option Delta - Difference between this option type and the previous

Length - Length of the option value (0-270)

Value - The value of Length bytes immediately follows Length

No.	C/E	Name	Format	Length	Default
1	Critical	Content-Type	uint	1-2 B	0
2	Elective	Max-Age	uint	0-4 B	60
3	Critical	Proxy-Uri	string	1-270 B	(none)
4	Elective	ETag	opaque	1-8 B	(none)
5	Critical	Uri-Host	string	1-270 B	(see below)
6	Elective	Location-Path	string	1-270 B	(none)
7	Critical	Uri-Port	uint	0-2 B	(see below)
8	Elective	Location-Query	string	1-270 B	(none)
9	Critical	Uri-Path	string	1-270 B	(none)
11	Critical	Token	opaque	1-8 B	(empty)
15	Critical	Uri-Query	string	1-270 B	(none)

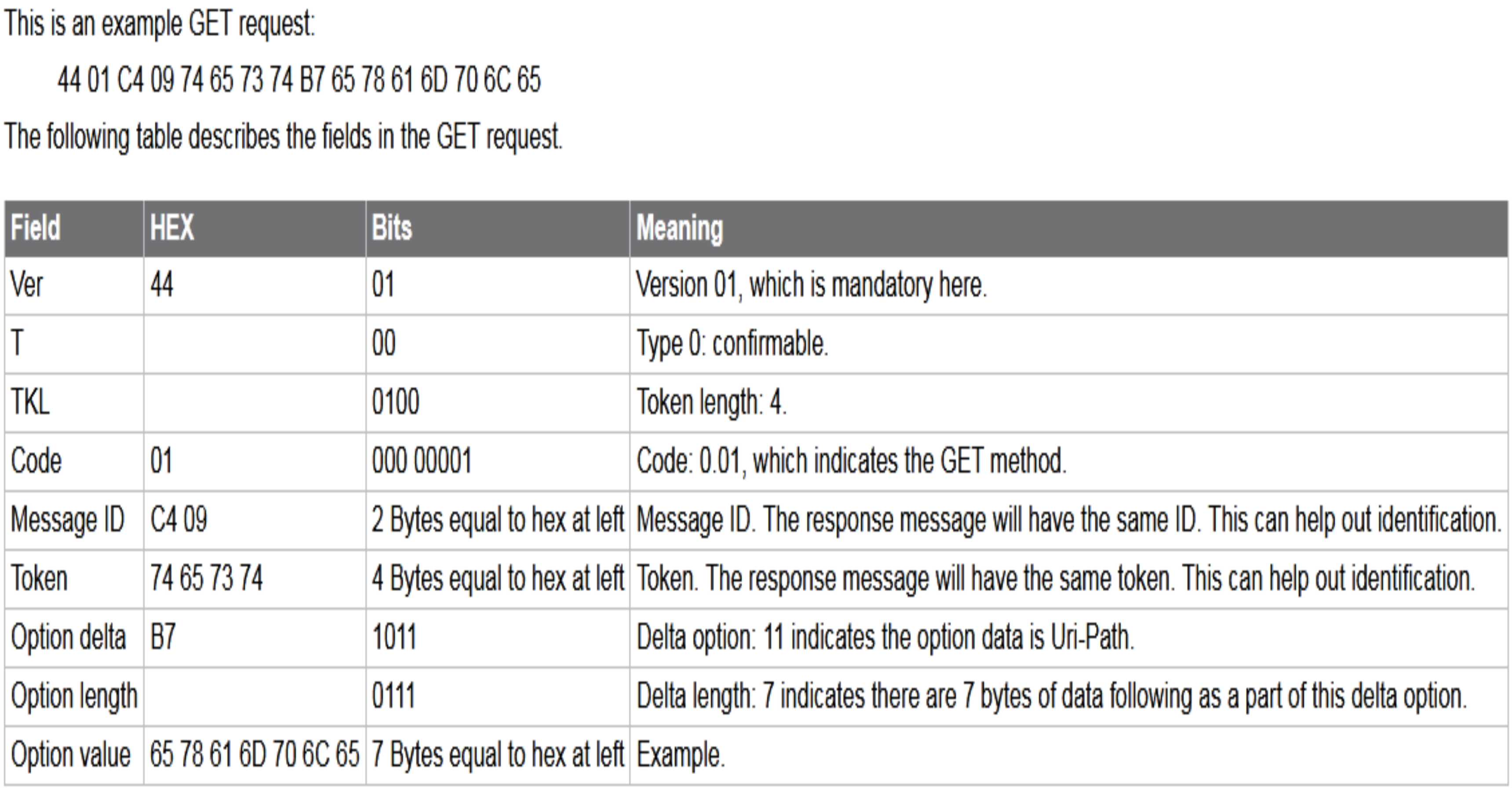
An Example GET Request



This is an example GET request:

```
44 01 C4 09 74 65 73 74 B7 65 78 61 6D 70 6C 65
```

An Example GET Request



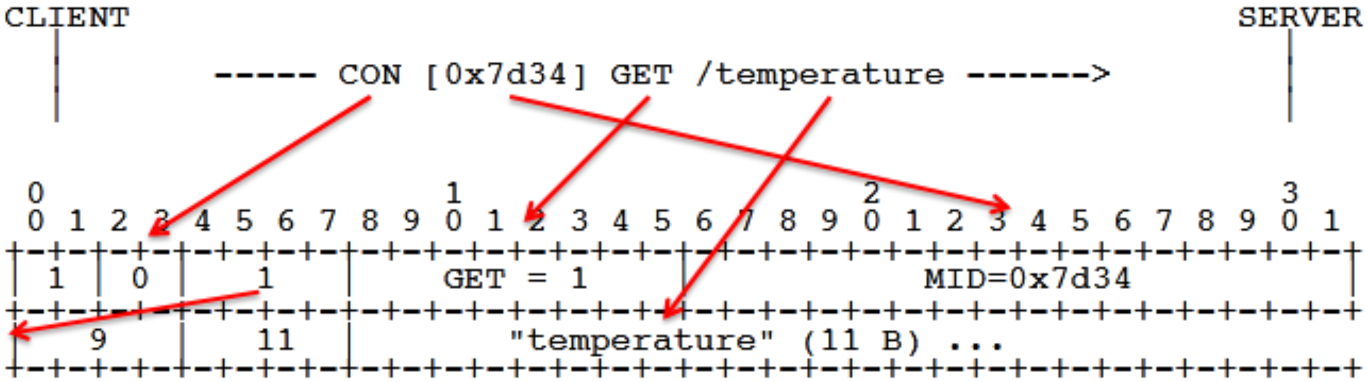
This is an example GET request:

```
    44 01 C4 09 74 65 73 74 B7 65 78 61 6D 70 6C 65
```

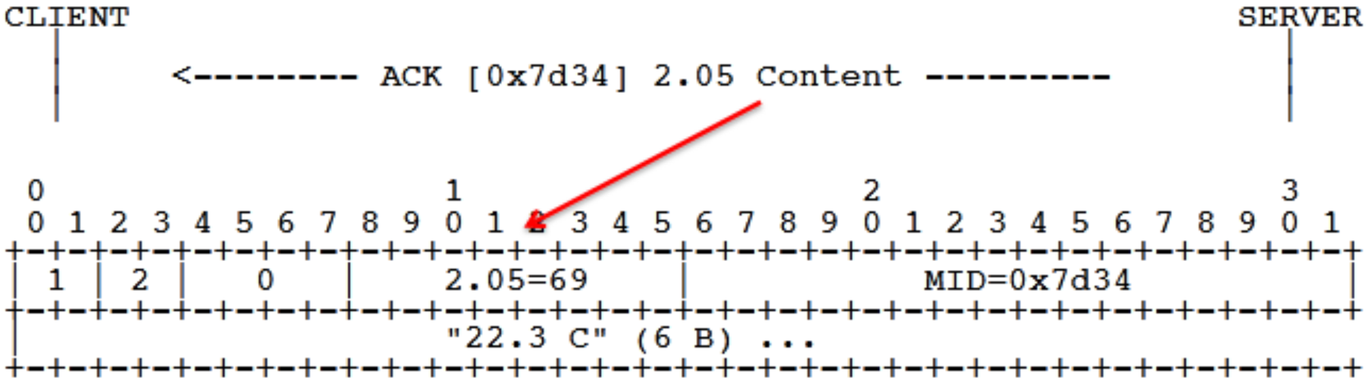
The following table describes the fields in the GET request.

Field	HEX	Bits	Meaning
Ver	44	01	Version 01, which is mandatory here.
T		00	Type 0: confirmable.
TKL		0100	Token length: 4.
Code	01	000 00001	Code: 0.01, which indicates the GET method.
Message ID	C4 09	2 Bytes equal to hex at left	Message ID. The response message will have the same ID. This can help out identification.
Token	74 65 73 74	4 Bytes equal to hex at left	Token. The response message will have the same token. This can help out identification.
Option delta	B7	1011	Delta option: 11 indicates the option data is Uri-Path.
Option length		0111	Delta length: 7 indicates there are 7 bytes of data following as a part of this delta option.
Option value	65 78 61 6D 70 6C 65	7 Bytes equal to hex at left	Example.

An Example GET Request

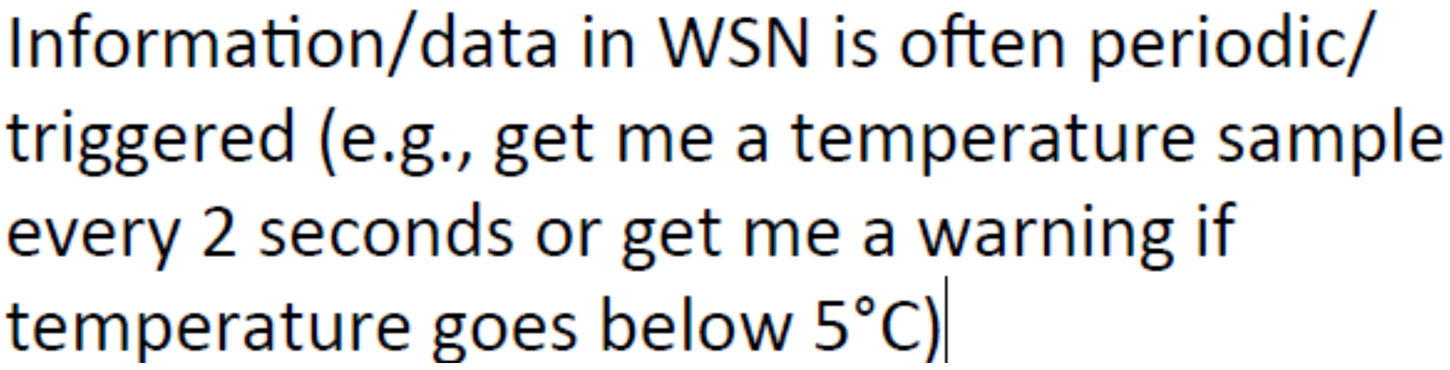


```
CLIENT                                                     SERVER
   |                                                          |
   |       ----- CON [0x7d34] GET /temperature ------>        |
   |                                                          |

 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| 1 | 0 |   1   |     GET = 1     |          MID=0x7d34           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   9   |  11   |       "temperature" (11 B) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

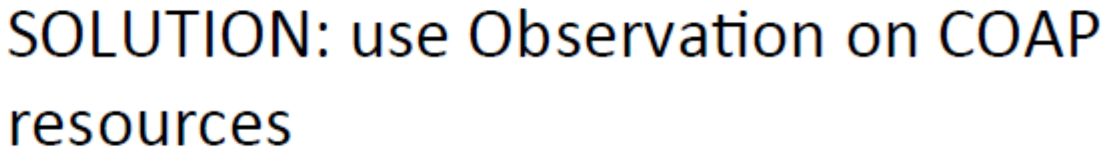


```
CLIENT                                                     SERVER
   |                                                          |
   |    <-------- ACK [0x7d34] 2.05 Content ---------         |
   |                                                          |

 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| 1 | 2 |   0   |    2.05=69      |          MID=0x7d34           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  "22.3 C" (6 B) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

COAP Observer



Information/data in WSN is often periodic/ triggered (e.g., get me a temperature sample every 2 seconds or get me a warning if temperature goes below 5°C)

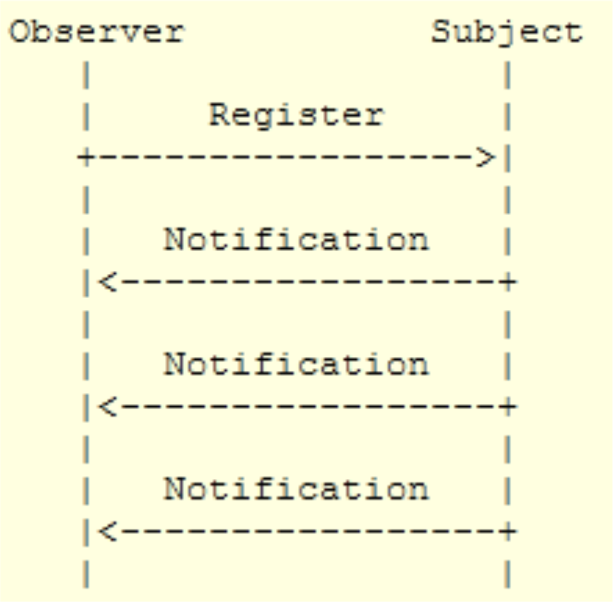


SOLUTION: use Observation on COAP resources

COAP Observer-Terminology

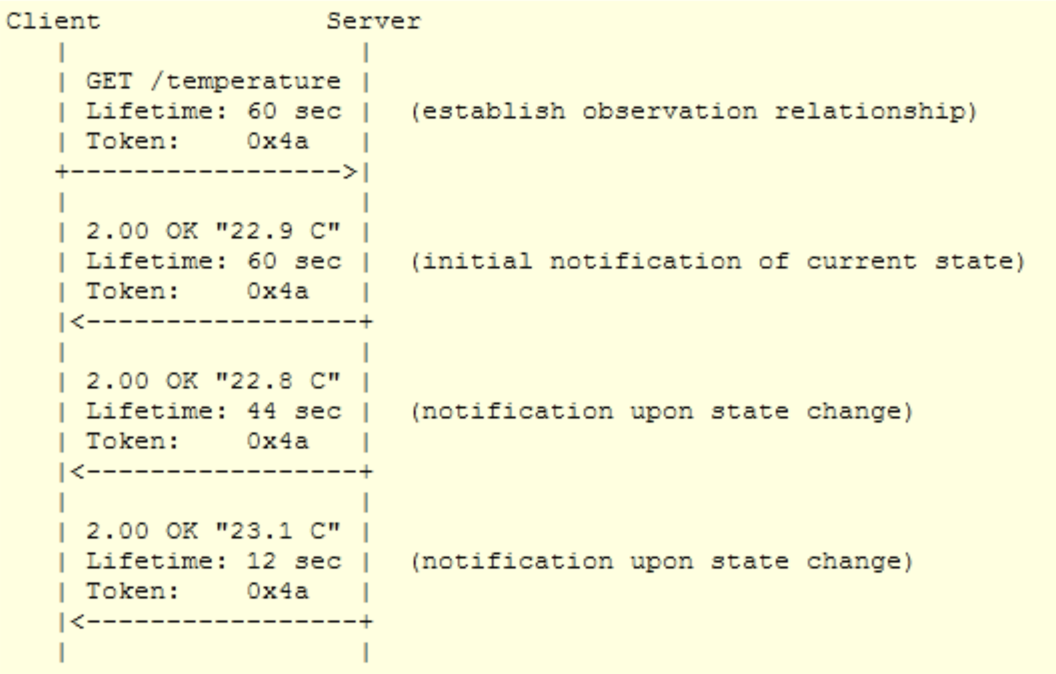
- **Subject:** In the context of CoAP, the subject is a resource located at some CoAP server. The state of the resource may change over time, ranging from infrequent changes to continuous state updates.
- **Observer:** The observer is a CoAP client that is interested in the current state of the resource at any given time.
- **Observation Relationship:** A client registers itself with a resource by sending a modified GET request to the server hosting the resource. The request causes the server to establish an observation relationship between the client and the resource. The response to the GET request supplies the client with a representation of the current resource state.
- **Notification:** Whenever the state of a resource changes, the server notifies each client that has an observation relationship to the resource. The notification is an additional response to the GET request; it supplies the client with a representation of the new resource state. The response echoes the token specified by the client in the request, so the client can easily correlate notifications.
- **Lifetime:** For robustness, an observation relationship is automatically ended after a negotiated duration of time. A client needs to refresh the relationship before the lifetime ends if it wants to be kept in the list of observers. The server includes the remaining lifetime duration in each notification.

COAP Observer



```
Observer          Subject
    |                 |
    |     Register    |
    +---------------->|
    |                 |
    |   Notification  |
    |<----------------+
    |                 |
    |   Notification  |
    |<----------------+
    |                 |
    |   Notification  |
    |<----------------+
    |                 |
```

an example of a CoAP client establishing an observation relationship with a resource on a CoAP server and then being notified, once upon registration and then whenever the state of the resource changes.



```
Client              Server
   |                  |
   | GET /temperature |
   | Lifetime: 60 sec |   (establish observation relationship)
   | Token:     0x4a  |
   +----------------->|
   |                  |
   | 2.00 OK "22.9 C" |
   | Lifetime: 60 sec |   (initial notification of current state)
   | Token:     0x4a  |
   |<-----------------+
   |                  |
   | 2.00 OK "22.8 C" |
   | Lifetime: 44 sec |   (notification upon state change)
   | Token:     0x4a  |
   |<-----------------+
   |                  |
   | 2.00 OK "23.1 C" |
   | Lifetime: 12 sec |   (notification upon state change)
   | Token:     0x4a  |
   |<-----------------+
   |                  |
```