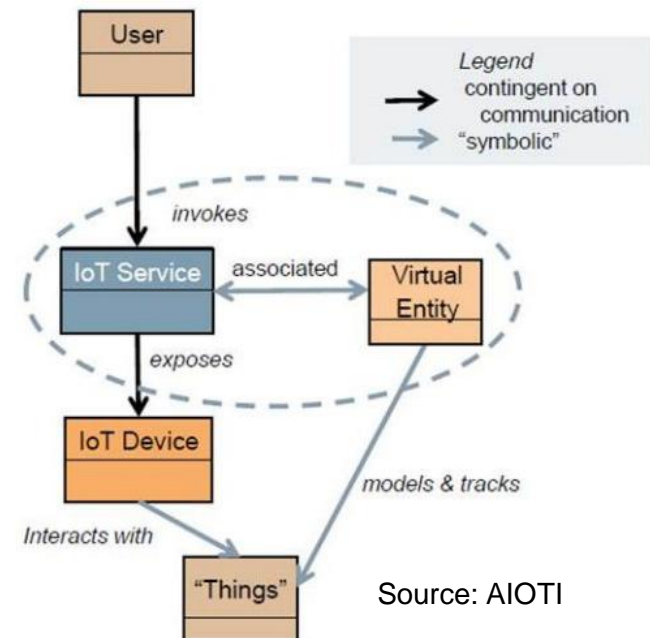


# EIOT

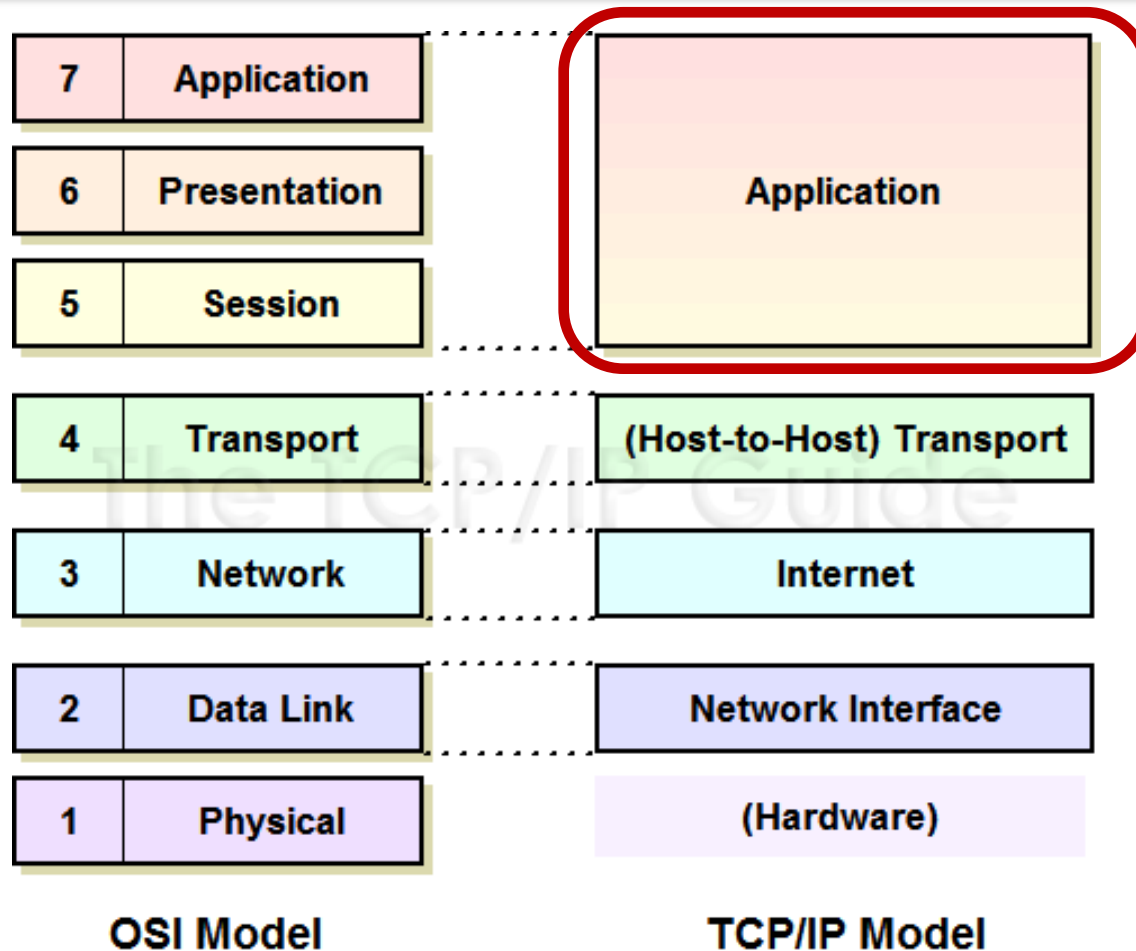
## Constrained Application Protocol (CoAP)

Jarosław Domaszewicz

Instytut Telekomunikacji Politechniki Warszawskiej



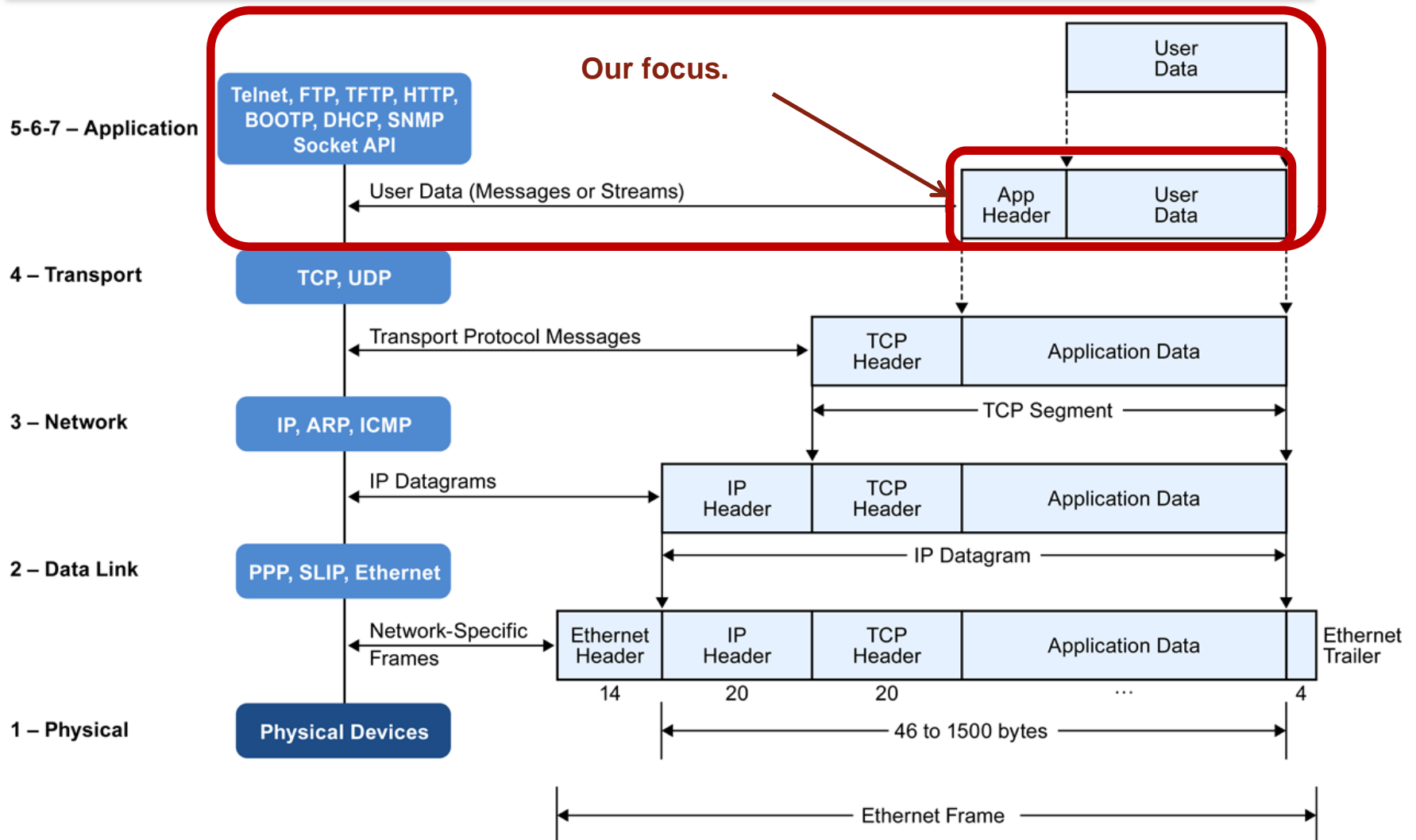
# APPLICATION LAYER (1/3)



Źródło: <http://www.tcpipguide.com>

**Innovations ? In the application layer!**

# APPLICATION LAYER (2/3)



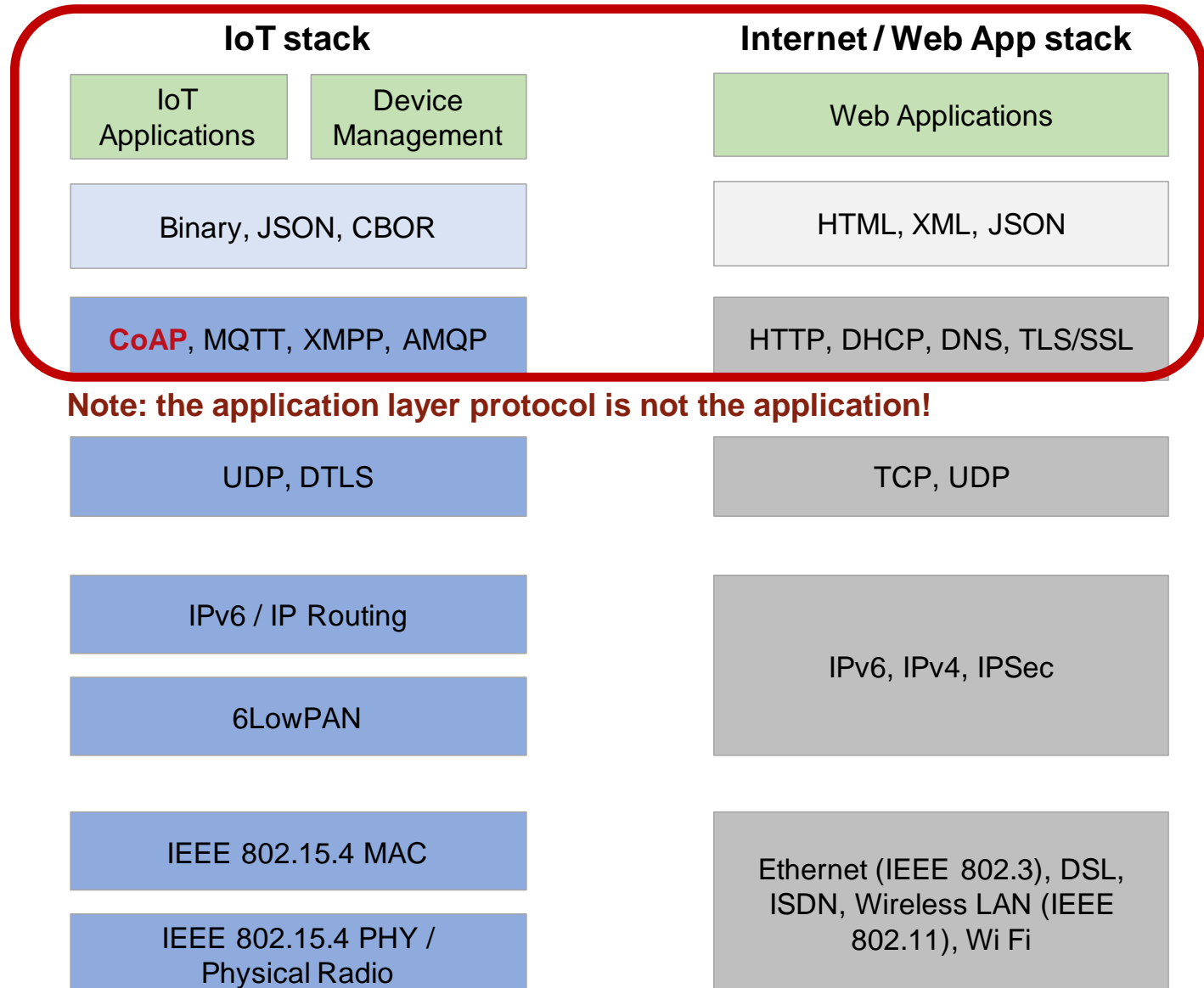
Source: <https://www.micrium.com/iot/internet-protocols>

# APPLICATION LAYER (3/3)

Source:

Constrained Application Protocol (Web Protocol for IoT)

A. Chakrabarti, [www.slideshare.net](http://www.slideshare.net)



**Note: the application layer protocol is not the application!**

# APPLICATION LAYER COMPETITORS (1/2)

- CoAP (Constrained Application Protocol)

- developed by CoRE, Constrained RESTful Environments WG of IETF
- an Internet (IETF) standard
- runs on top of UDP
- enables HTTP-like interactions in IoT: client/server, restful APIs



- MQTT (formerly Message Queuing Telemetry Transport, now MQTT)

- developed by industry (IBM, Arcom)
- supported by a major IBM product (MQ series)
- now an OASIS standard and ISO standard
- runs on top of TCP
- based on the publish/subscribe interaction paradigm



# APPLICATION LAYER COMPETITORS (2/2)

	MQTT	CoAP
Application Layer	Single Layered completely	Single Layered with 2 conceptual sub layers ( Messages Layer and Request Response Layer )
Transport Layer	Runs on TCP	Runs on UDP
Reliability Mechanism	3 Quality of Service levels	<u>Confirmable messages</u> , Non-confirmable messages, <u>Acknowledgements and retransmissions</u>
Supported Architectures	Publish-Subscribe	Request-Response, <span>Resource observe/Publish-Subscribe</span>

Message Layer (reliability)

Observe option

Source: *Performance Evaluation of MQTT and CoAP via a Common Middleware*,  
D. Thangavel et al., 2014 IEEE Ninth Intl. Conf. on Intelligent Sensors, Sensor Networks and Information Processing, 2014

# CoRE (CONSTRAINED RESTFUL E.) *CONSTRAINED?*

---

Name	data size (e.g., RAM)	code size (e.g., Flash)
Class 0, C0	<< 10 KiB	<< 100 KiB
Class 1, C1	~ 10 KiB	~ 100 KiB
Class 2, C2	~ 50 KiB	~ 250 KiB

**Table 1: Classes of Constrained Devices (KiB = 1024 bytes) [RFC7228]**

Source: *Terminology for Constrained-Node Networks*, RFC7228  
C. Bormann, M. Ersue, A. Keranen , May 2014

# CoRE (CONSTRAINED RESTFUL E.) *RESTFUL?*

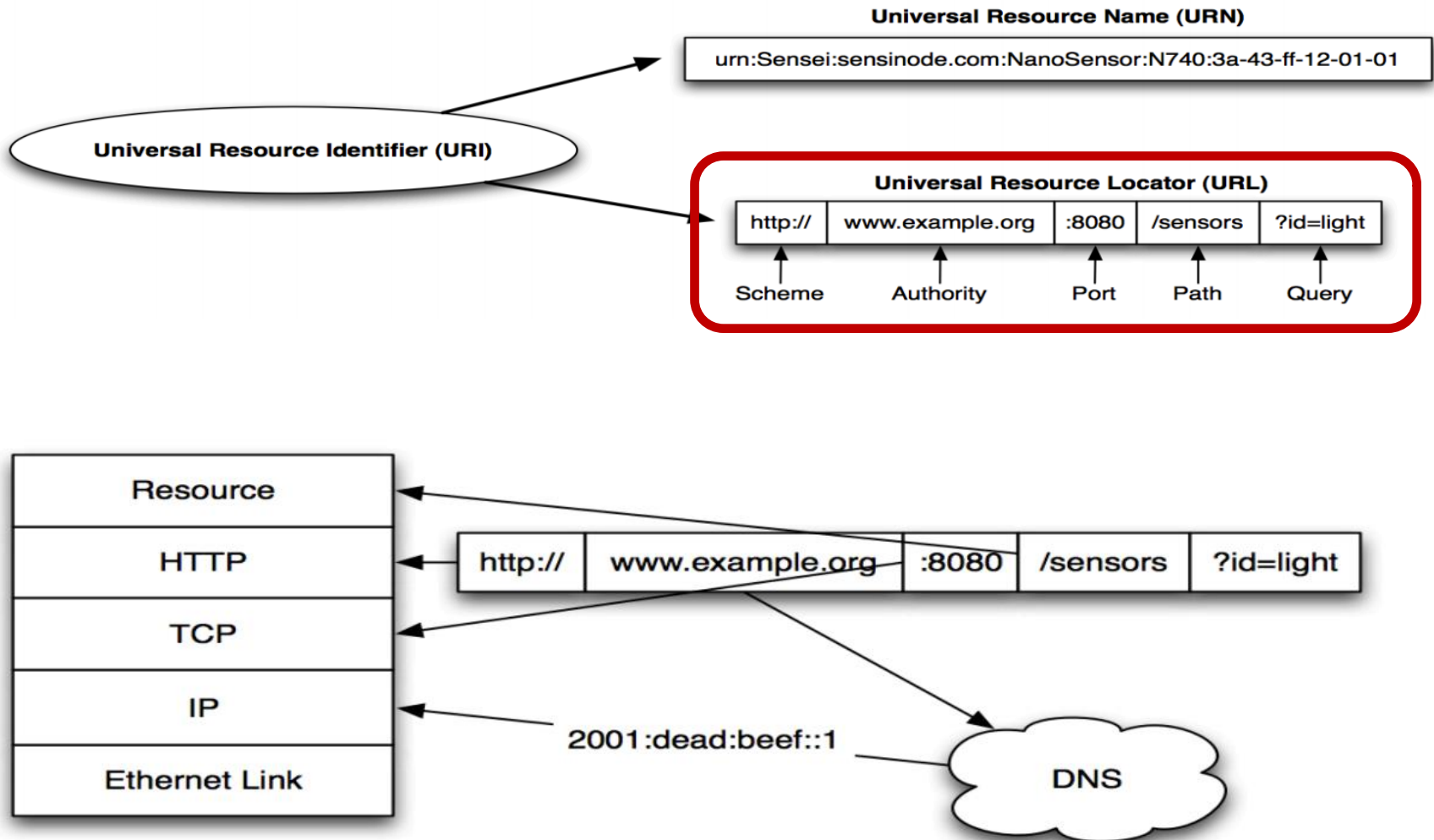
---

- note: the description below is (somewhat) simplified
- there are resources (e.g., data items, sensor readings, ..., whatever)
- a resource has its URI
- a resource is hosted on a server
- a resource has its (possibly multiple) representation(s)
  - a resource representation has its media type
- the client uses the CRUD "verbs" (Create, Retrieve, Update, Delete) to transfer (work with) resource representations
  - these verbs are resource and application neutral
- no per-client state on the server (statelessness)
  - a request from a client must be understood by itself
  - the state is kept only on clients

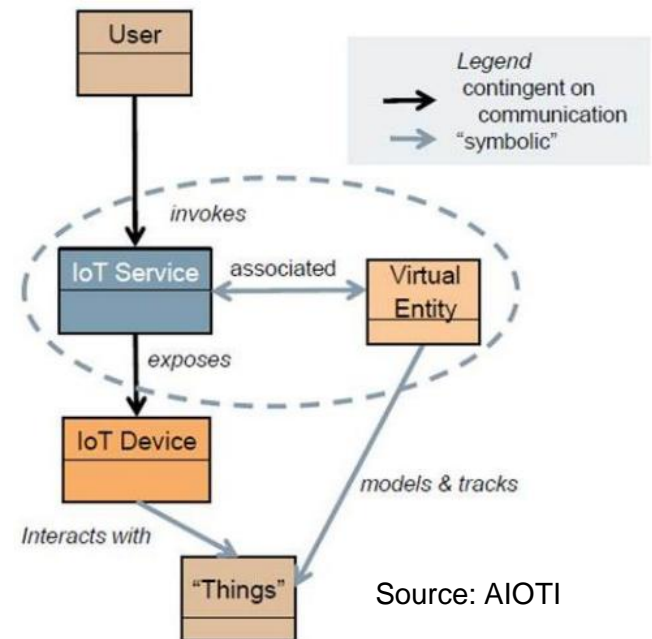


# URIs

Source: *CoAP: The Web of Things Protocol*, ARM IoT Tutorial, Z. Shelby, 2014



# CoAP – fundamentals



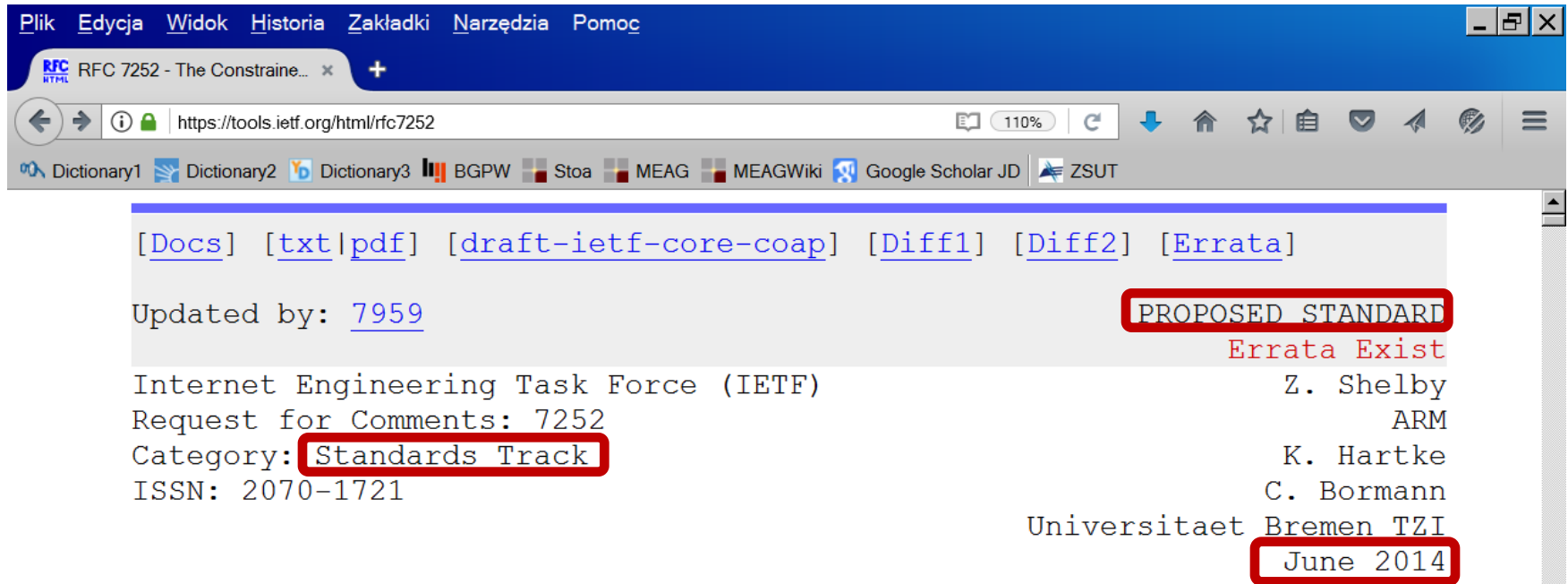
# CoAP: KEY RFCs

---

- [RFC7252] "The Constrained Application Protocol (CoAP)"
  - Z. Shelby, K. Hartke, C. Bormann, June 2014**the main CoAP specification, 112 pages**
- [RFC7641] "Observing Resources in CoAP"
  - K. Hartke, September 2015**how to be up to date about the state of a resource without too many requests**
- [RFC7959] "Blockwise Transfers in CoAP"
  - C. Bormann, Z. Shelby, August 2016**how to transfer big resource representations**
- [RFC6690] "Constrained RESTful Environments (CoRE) Link Format"
  - Z. Shelby, August 2012**how to discover resources hosted by a server**

# RFC 7252 (1/2)

- Document: <https://tools.ietf.org/html/rfc7252>



The screenshot shows a web browser window with the URL <https://tools.ietf.org/html/rfc7252>. The page content includes navigation links at the top: [Docs], [txt|pdf], [draft-ietf-core-coap], [Diff1], [Diff2], and [Errata]. Below these, it states "Updated by: 7959" and "PROPOSED STANDARD" (highlighted with a red box). The text "Errata Exist" is also visible. The main body of the page identifies the document as "Internet Engineering Task Force (IETF) Request for Comments: 7252" and "Category: Standards Track" (highlighted with a red box). The ISSN is listed as "2070-1721". On the right side, the authors are listed: "Z. Shelby", "ARM", "K. Hartke", "C. Bormann", and "Universitaet Bremen TZI". The date "June 2014" is also highlighted with a red box.

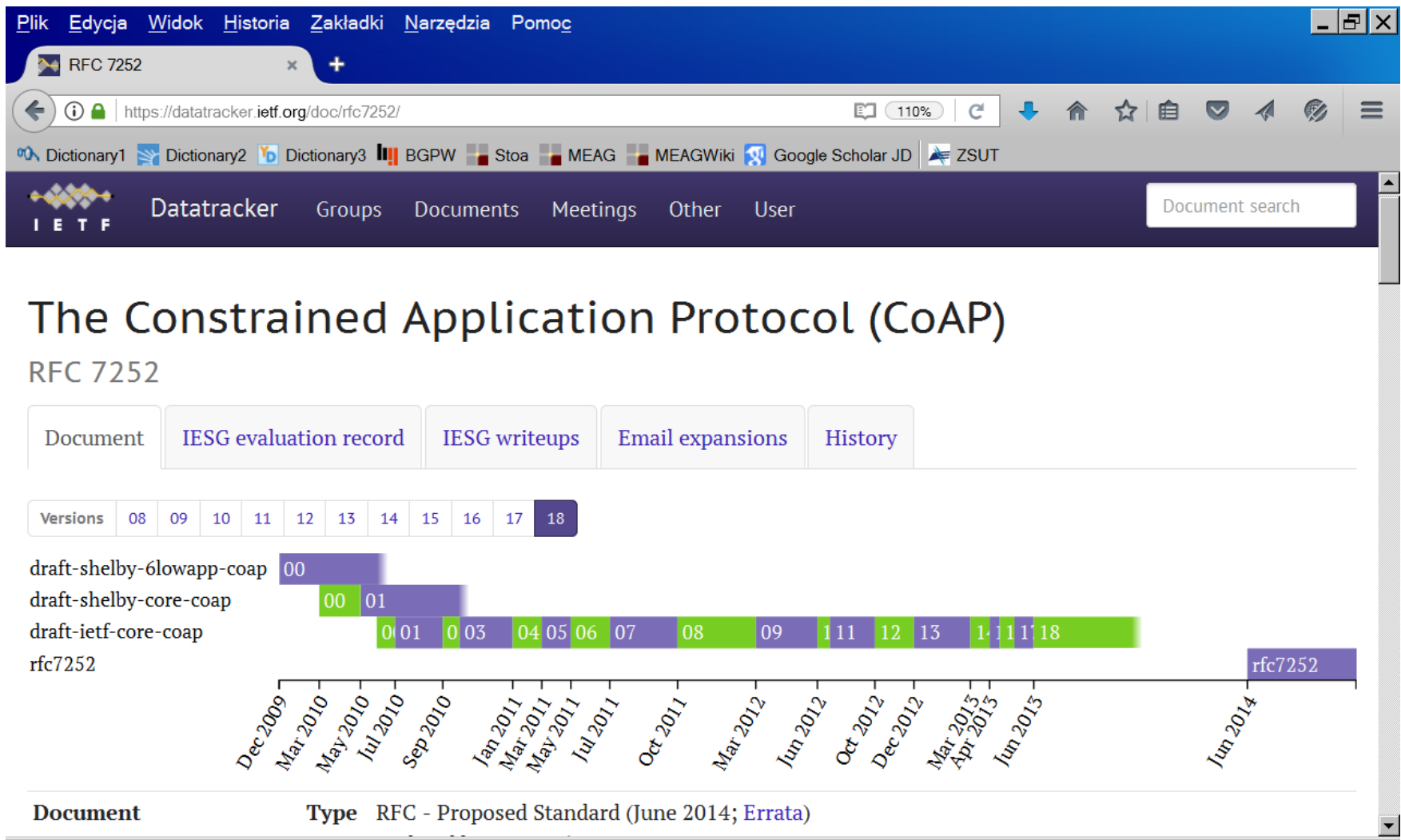
## The Constrained Application Protocol (CoAP)

### Abstract

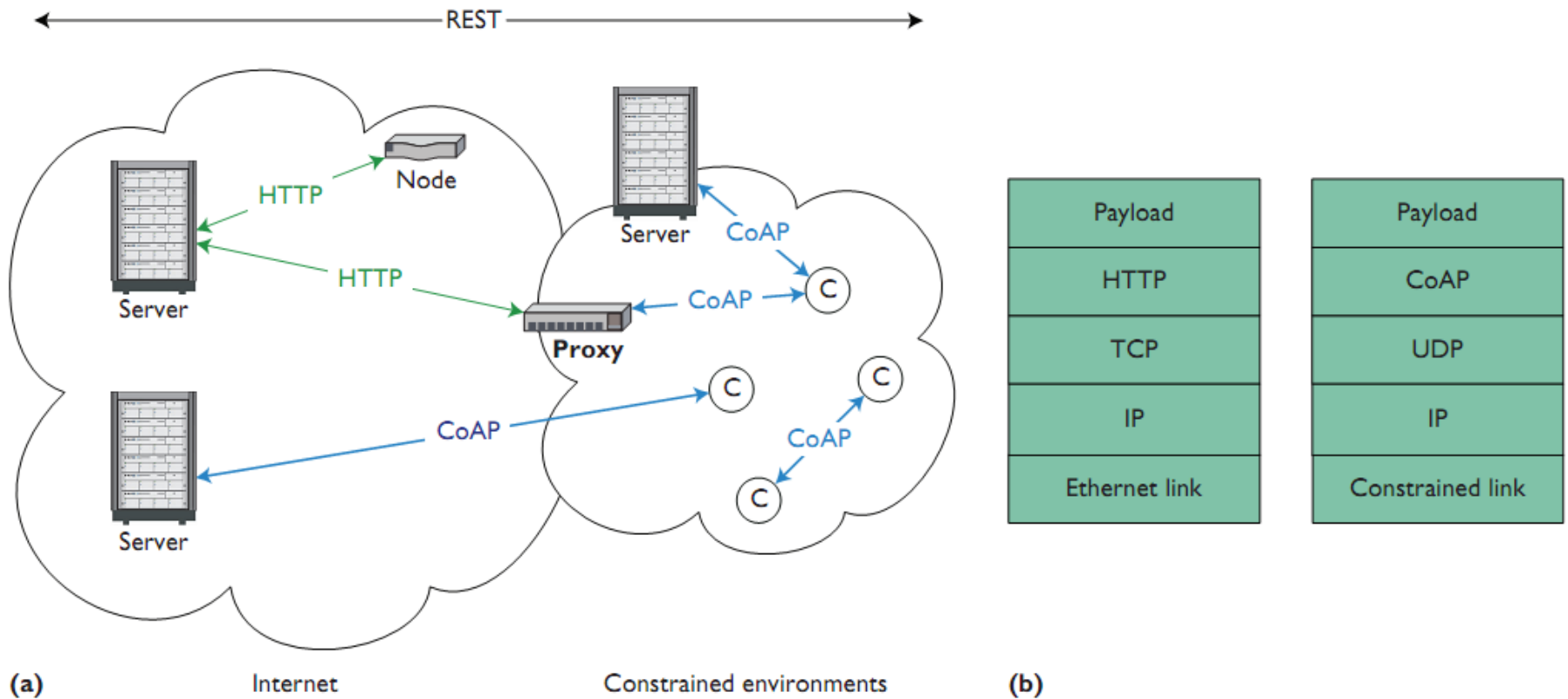
The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained (e.g., low-power, lossy) networks. The nodes often have 8-bit microcontrollers with small amounts of ROM and RAM, while constrained

# RFC 7252 (2/2)

- History: <https://datatracker.ietf.org/doc/rfc7252/>

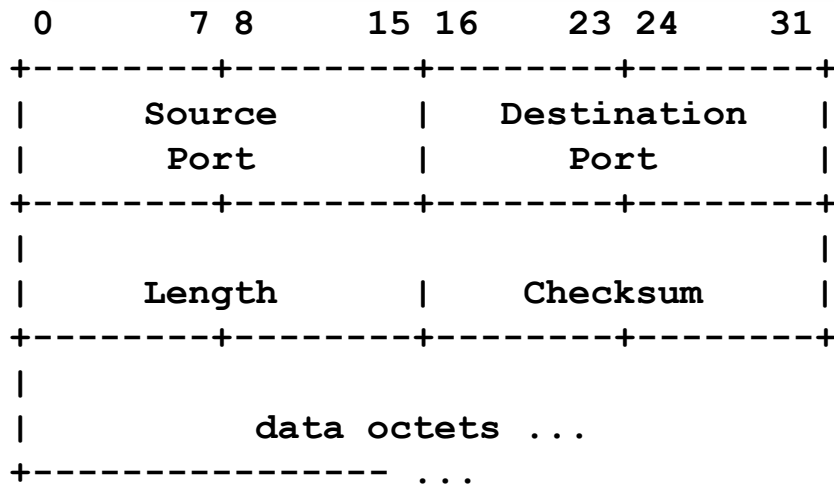


# CoAP SYSTEM ARCHITECTURE

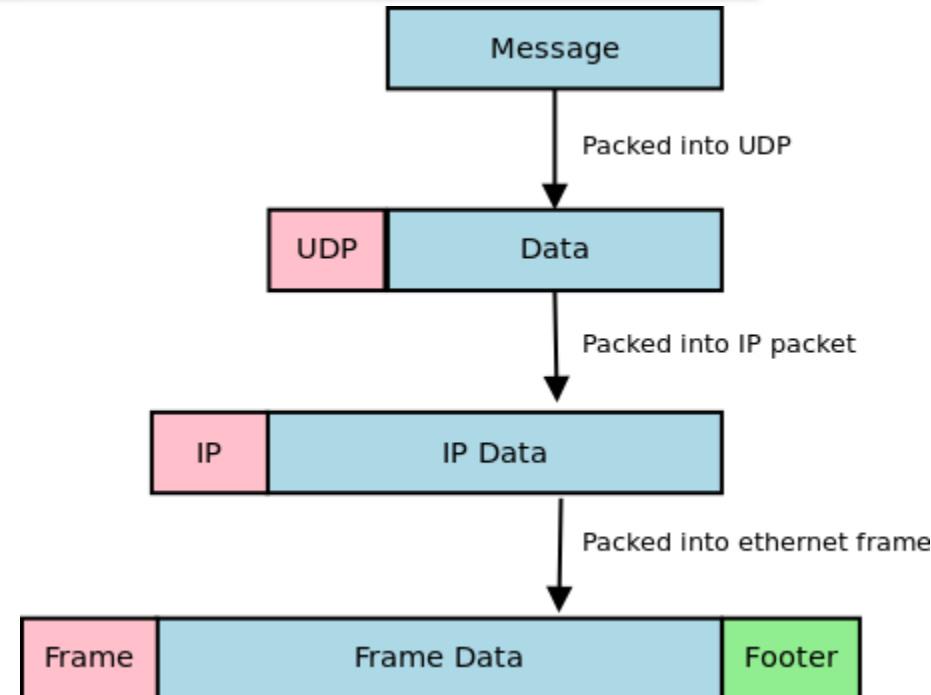


Source: *CoAP: An Application Protocol for Billions of Tiny Internet Nodes*  
C. Bormann, A. P. Castellani, Z. Shelby  
IEEE INTERNET COMPUTING, 2012

# UDP BASICS



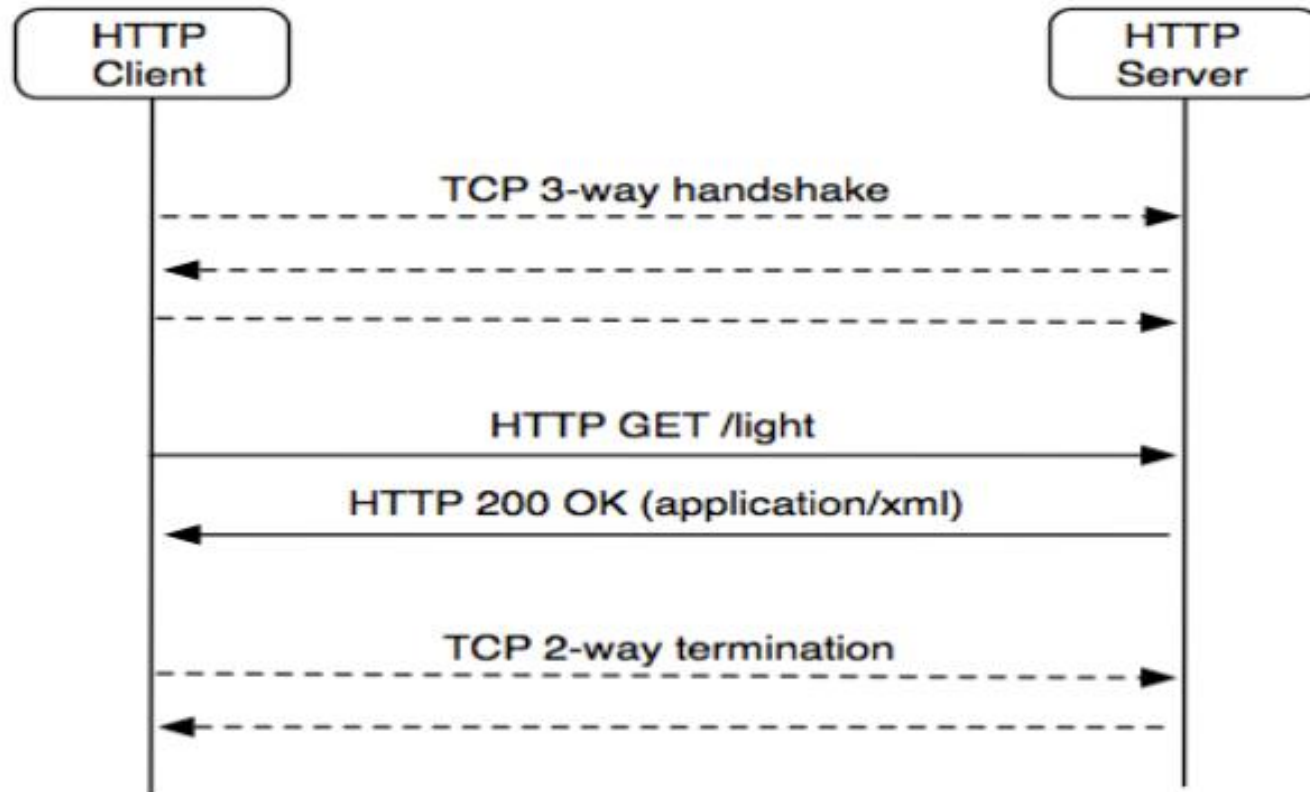
User Datagram Header Format [RFC768]



<http://jamesslocum.com/post/77759061182>

- connectionless
- each user datagram results in a single IP datagram
- delivery: out-of-order, duplicated, missing
- offers the port abstraction
- aside: why would anybody want to use UDP?

# WHY NOT TCP?



Source: *CoAP: The Web of Things Protocol*, ARM IoT Tutorial, Z. Shelby, 2014



# CoAP IN THE PROTOCOL STACK

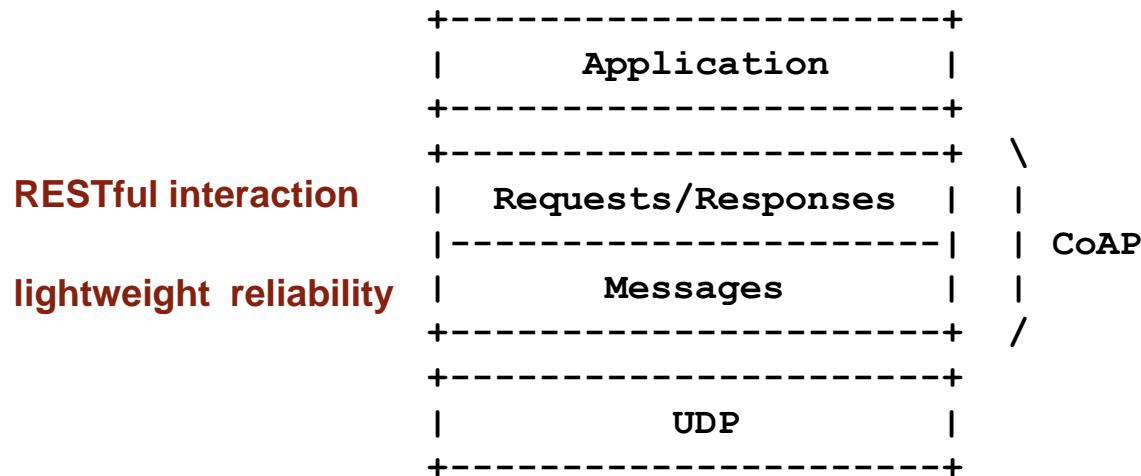


Figure 1: Abstract Layering of CoAP [RFC7252]

- CoAP endpoint = IP address + UDP port number, port 5683
- each CoAP message occupies the data section of one UDP datagram
- CoAP over TCP (RFC 8323) is also possible
- CoAP is not the application itself (the application logic is up to you!)

# CoAP MESSAGES

---

- CoAP client and server (one node may play both roles)
- requests/responses:
  - requests: from client to server  
method code (which action to perform on the resource): GET, PUT, POST, DELETE
  - responses: from server to client  
response code (similar to the HTTP status code)
- CON (confirmable)/NON (non-confirmable)/ACK/RST
  - CON+ACK: lightweight reliability
  - RST: recipient unable to process the message

# CoAP MESSAGE FORMAT

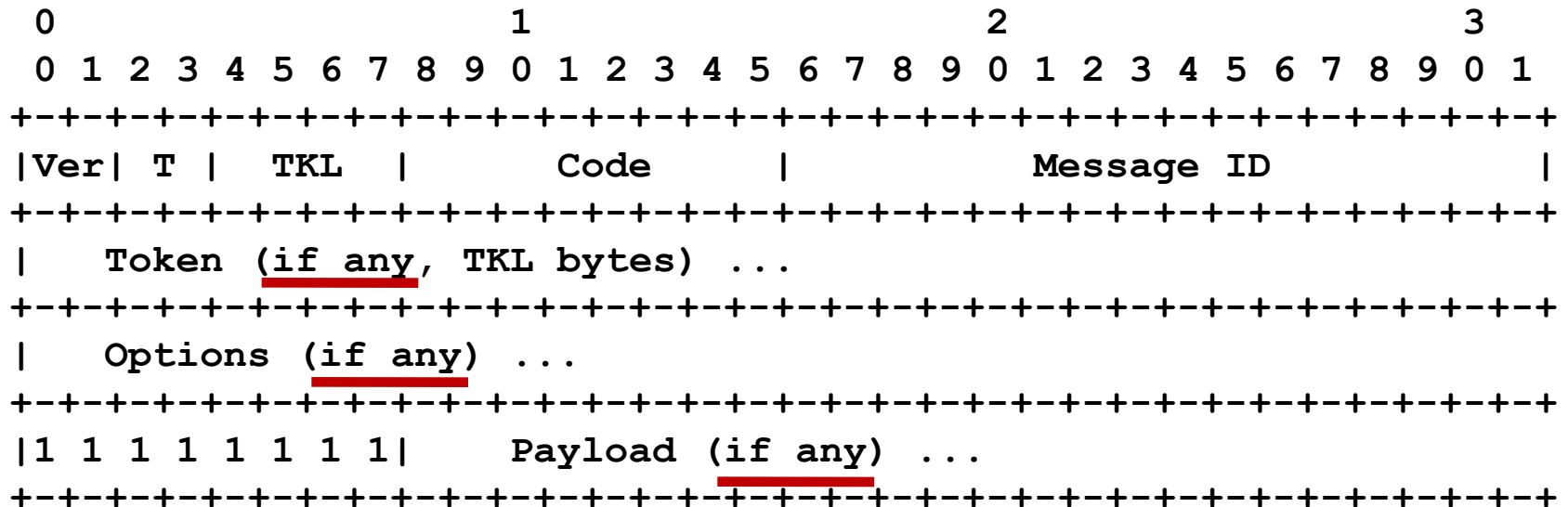


Figure 7: Message Format[RFC7252]

shortest CoAP message: 4B

# CoAP MESSAGE FORMAT

## Consider a GET on a resource. Here is what different fields are for.

The diagram illustrates the structure of an HTTP message and how it is matched with an ACK and a response. The message is divided into several sections, each with a specific purpose:

- GET (in the request) and Content (in the response)**: This section contains the GET method and the Content-Type header. It is matched with the ACK and the response.
- to match an ACK with the message**: This section contains the Content-Length header, which is used to match the ACK with the message.
- to match the response with the request**: This section contains the Token (if any, TKL bytes) and the Options (if any) fields, which are used to match the response with the request.
- URI (in the request) goes here**: This section contains the URI, which is the resource being requested.
- a resource representation (in the response)**: This section contains the Payload (if any), which is the resource representation.

The diagram shows the following structure:

```

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) ...   to match the response with the request
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|   Options (if any) ...   URI (in the request) goes here
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|1 1 1 1 1 1 1|   Payload (if any) ... a resource representation (in the response)
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
  
```

### Figure 7: Message Format[RFC7252]

# CoAP MESSAGE FORMAT

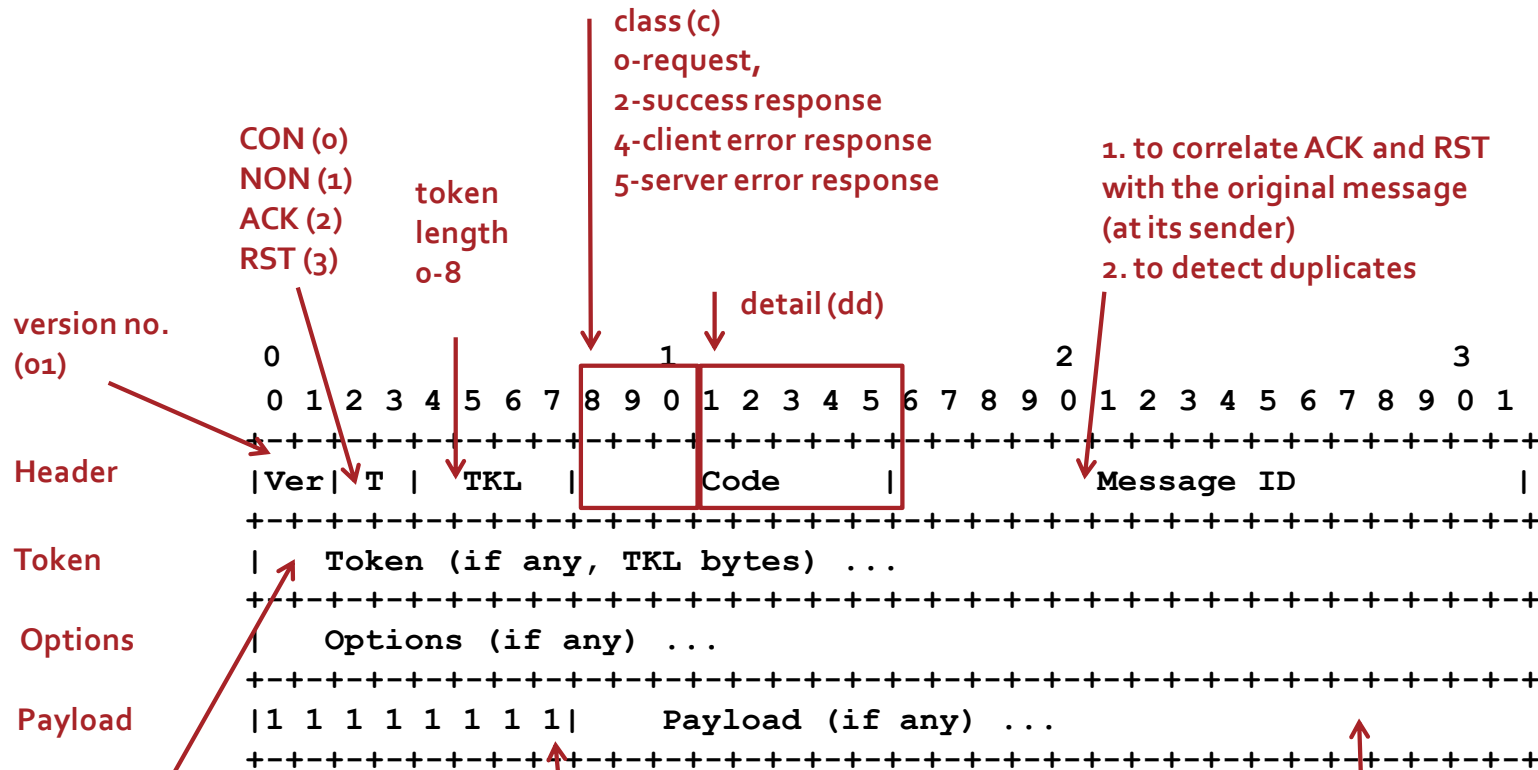


Figure 7: Message Format[RFC7252]

c.dd indicates a Request Method or a Response Code

0.00 Empty message

0.01 GET

0.02 POST

0.03 PUT

0.04 DELETE

2.dd success

4.dd client error

5.dd server error

# NO RELIABILITY: NON-CONFIRMABLE MESSAGES

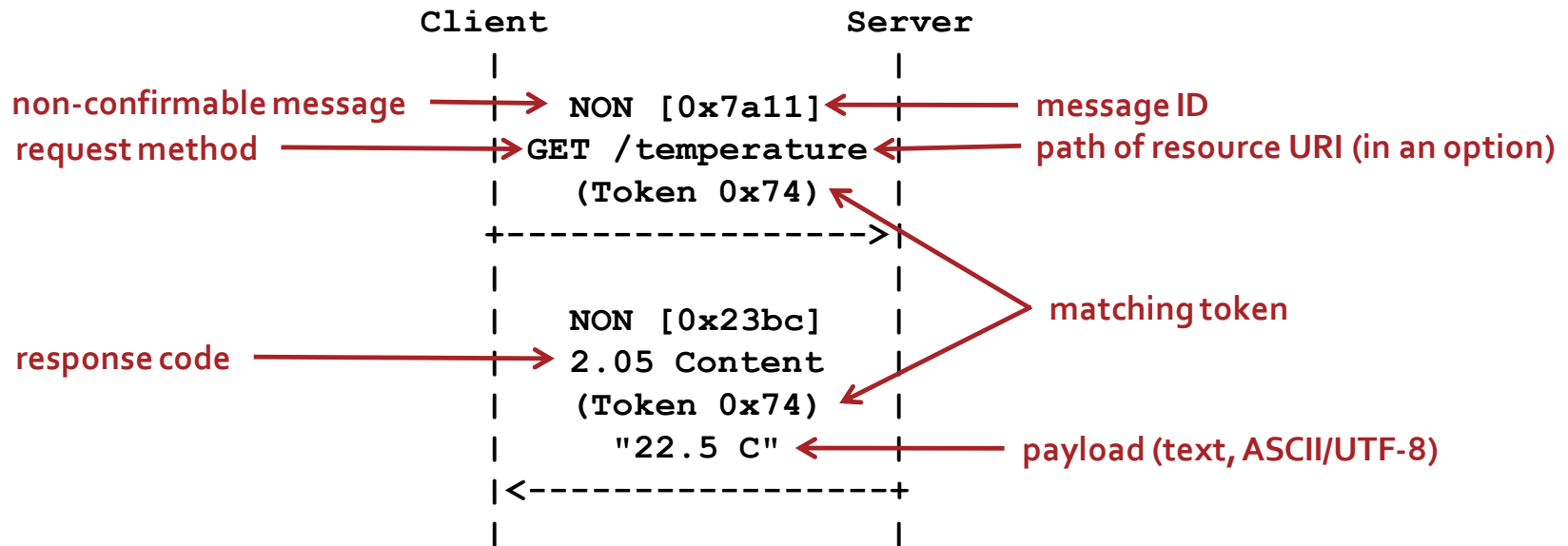


Figure 6: A Request and a Response Carried in Non-confirmable Messages [RFC7252]

- reception not acknowledged
- the token is used to match a response with its request
- RST when the recipient unable to process a non-confirmable message

# WITH RELIABILITY: CONFIRMABLE MESSAGES

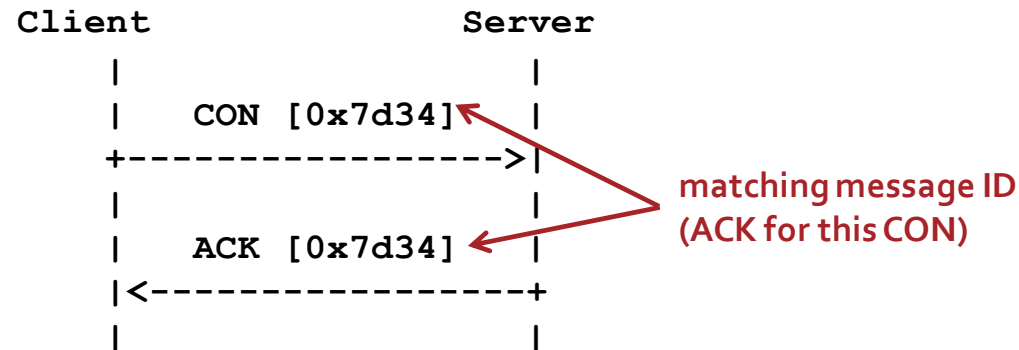


Figure 2: Reliable Message Transmission [RFC7252]

- simple stop-and-wait
- wait for ACK (or RST) with timeout
- if no ACK, retransmit
- exponential back-off: timeout doubled each time
- continue until you run out of attempts (MAX\_RETRANSMIT)
- RST when the recipient unable to process a confirmable message
- note: ACK (by itself) is not a response

# CONFIRMABLE: PIGGYBACKED RESPONSE

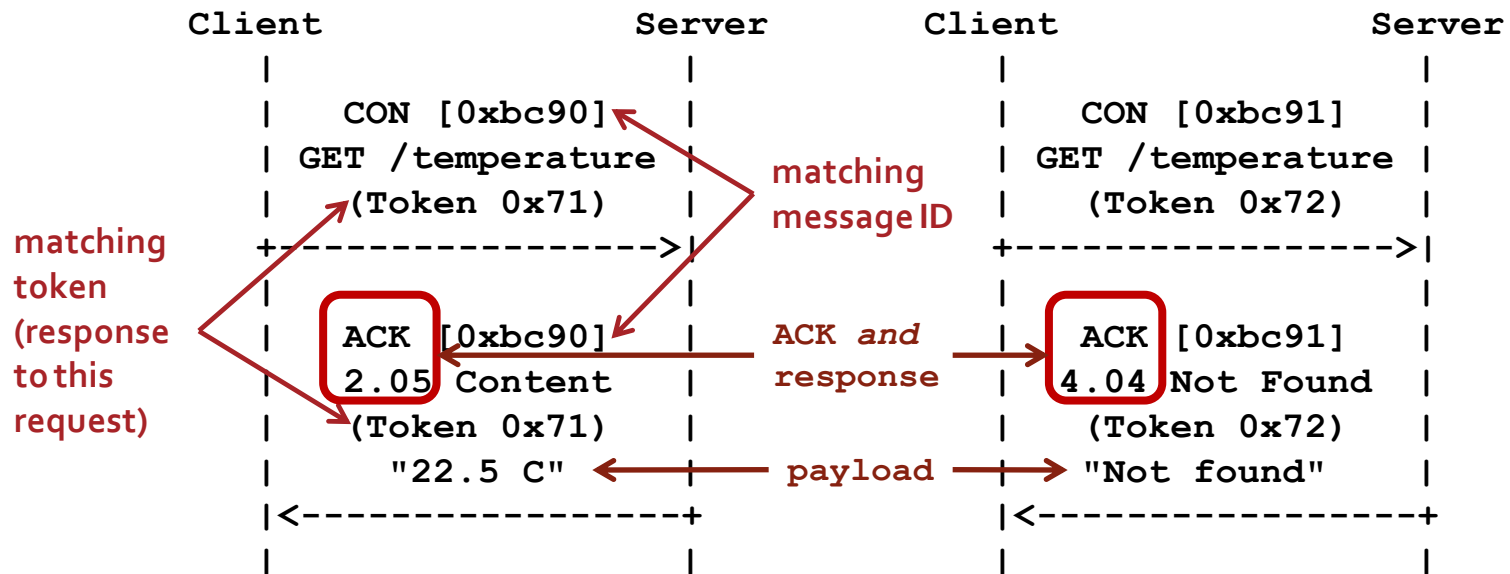


Figure 4: Two GET Requests with Piggybacked Responses[RFC7252]

- the response carried in ACK (if available immediately)



# CONFIRMABLE: EMPTY ACK AND SEPARATE RESPONSE

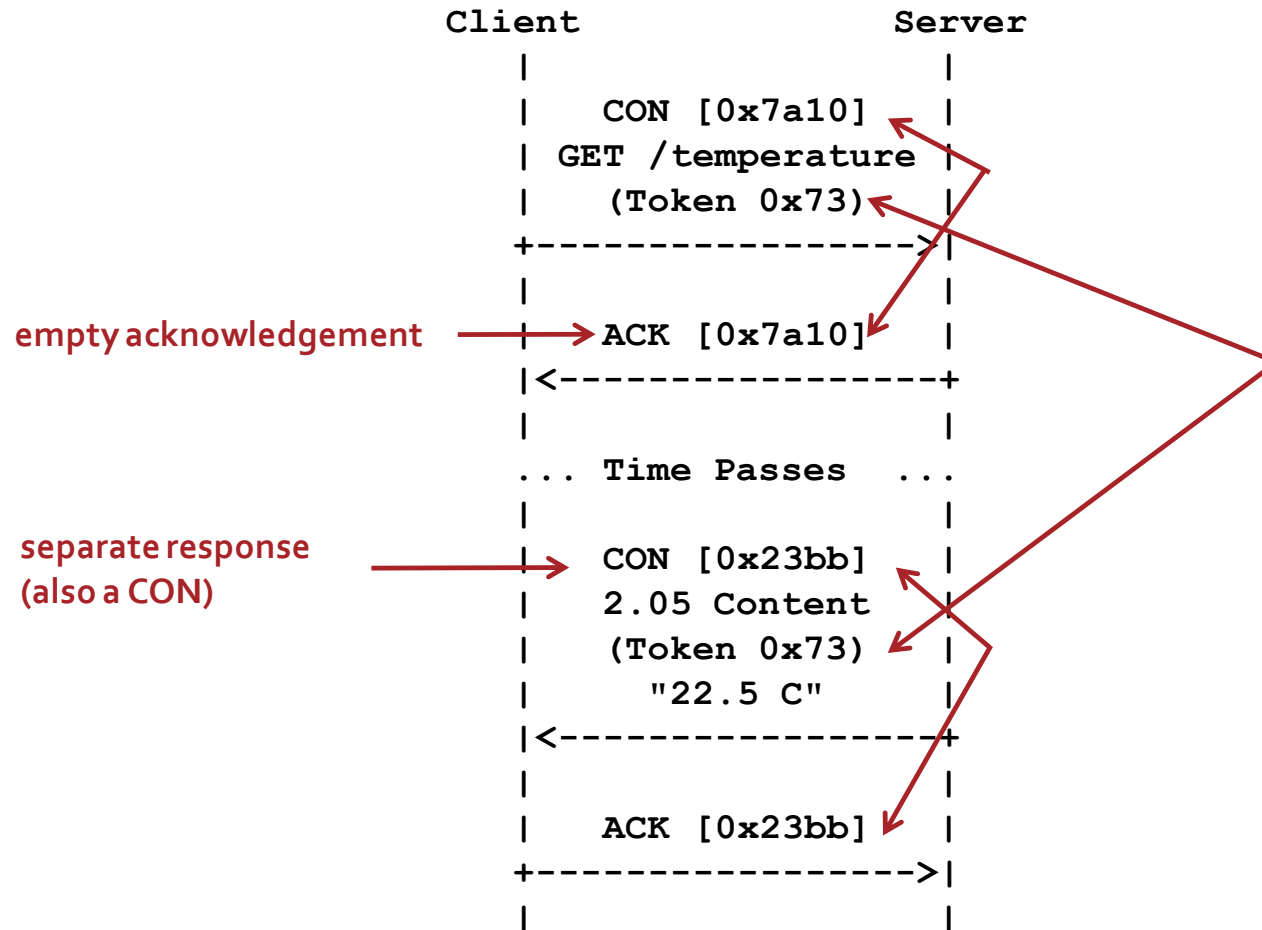


Figure 5: A GET Request with a Separate Response[RFC7252]

- if the response not available immediately (say, it takes some time to take a sensor reading)

# USAGE OF MESSAGE TYPES

message ID must be echoed

T field

Code field

piggybacked response

CoAP ping

empty ACK

	CON	NON	ACK	RST
Request	X	X	-	-
Response	X	X	X	-
Empty	*	-	X	X

Table 1: Usage of Message Types [RFC7252]

- CoAP ping: to elicit a reset message (RST), not in normal operation

# CON, NON, ACK, RST, MESSAGE ID, TOKEN IN MESSAGE

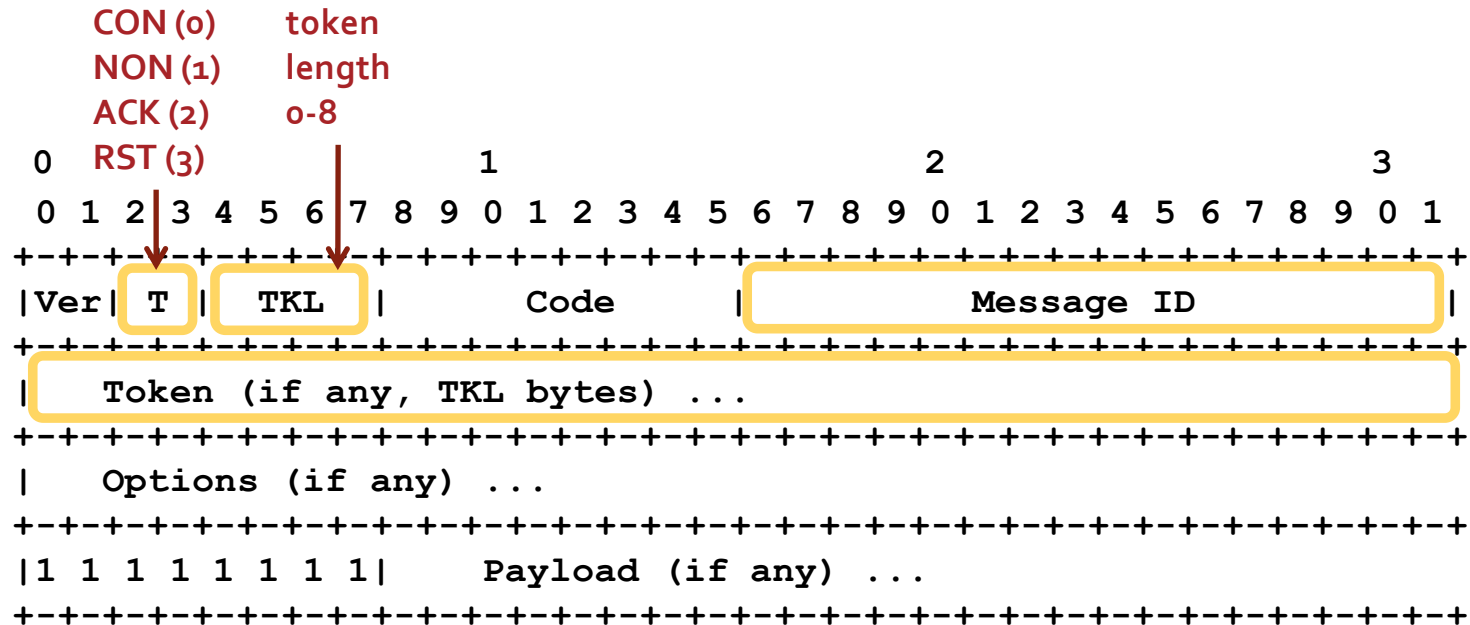


Figure 7: Message Format[RFC7252]

# REQUEST METHODS

---

- GET, PUT, POST, and DELETE
- these are similar to those of HTTP
- an URI (partially given in options) identifies a resource
- GET: retrieves a representation of the identified resource
- POST: requests that the representation enclosed in the request be processed
  - the actual function performed by the POST method is determined by the server and dependent on the target resource
  - it usually results in a new resource being created or the target resource being updated (the target resource may also be deleted)
- PUT: requests that the identified resource be updated or created with the enclosed representation
- DELETE: requests that the identified resource be deleted

# METHOD CODES IN MESSAGE

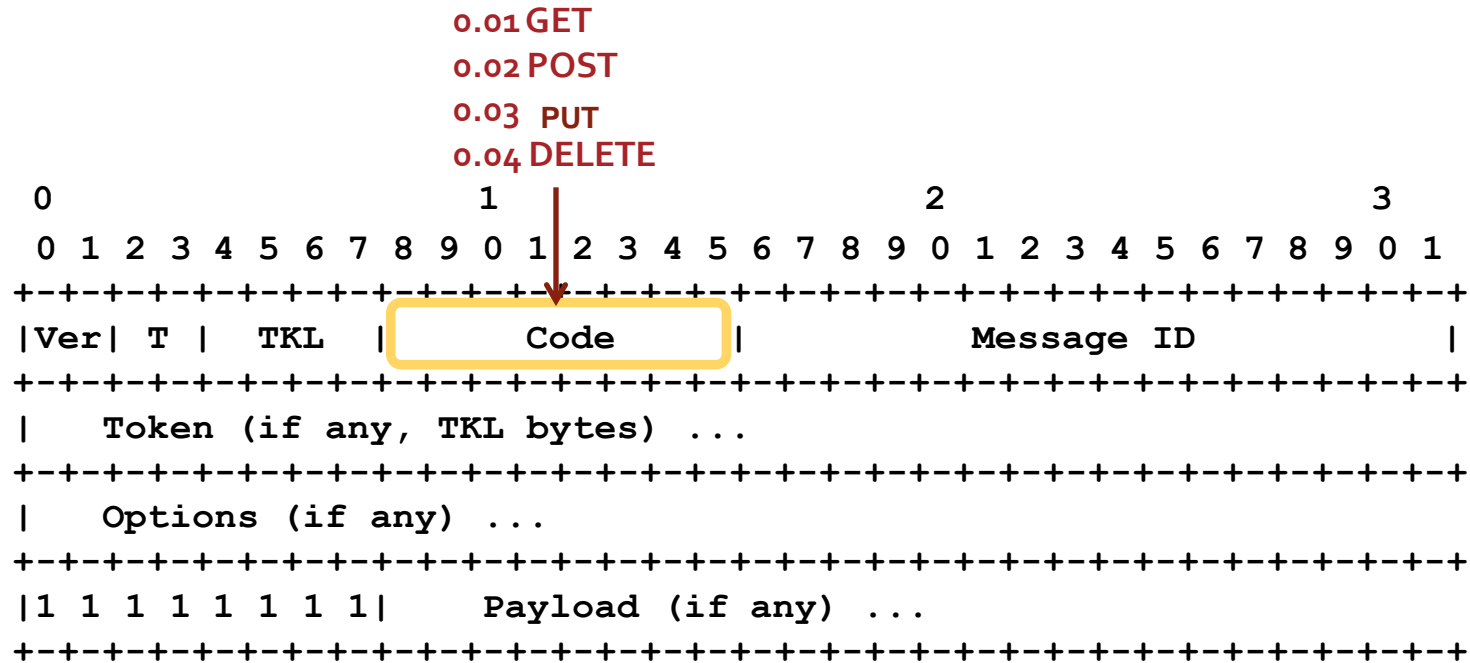


Figure 7: Message Format[RFC7252]

# RESPONSES

---

- a response is matched to the request by means of a client-generated token
- three classes of Response Codes: **kody odpowiedzi**
  - 2 - Success: the request was successfully received, understood, and accepted
  - 4 - Client Error: the request contains bad syntax or cannot be fulfilled
  - 5 - Server Error: the server failed to fulfill an apparently valid request

# RESPONSE CODES IN MESSAGE: SUCCESS 2.XX

2.01	Created	POST and PUT
2.02	Deleted	DELETE and POST
2.03	Valid	the response identified by the entity-tag is valid (used in validation for caching purposes)
2.04	Changed	PUT and POST
2.05	Content	GET
2.31	Continue	in block-wise transfers; a block has been received successfully, but the total update has not been completed yet

```

0
0 1 2 3 4 5 6 7
+---+---+---+---+---+---+
|class| detail |
+---+---+---+---+---+---+

```

c.dd

2.05 ↔ binary: 010.00101 ↔ decimal: 64+4+1=69

Figure 9: Structure of a Response Code

```

0
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| Payload (if any) ...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

# RESPONSE CODES IN MESSAGE: CLIENT ERROR 4.XX

4.00	Bad Request (generic response code)
4.01	Unauthorized
4.02	Bad Option
4.04	Not Found
4.05	Method Not Allowed
4.15	Unsupported Content-Format
...	...

```

0
0 1 2 3 4 5 6 7
+---+---+---+---+---+---+
|class| detail | c.dd
+---+---+---+---+---+---+

```

Figure 9: Structure of a Response Code

```

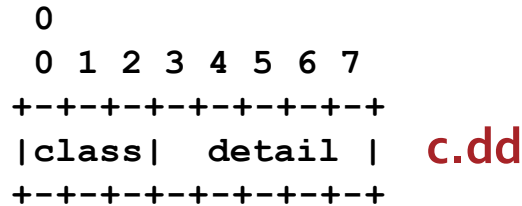
0
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|Ver| T | TKL | Code | Message ID |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| Token (if any, TKL bytes) ...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+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

```

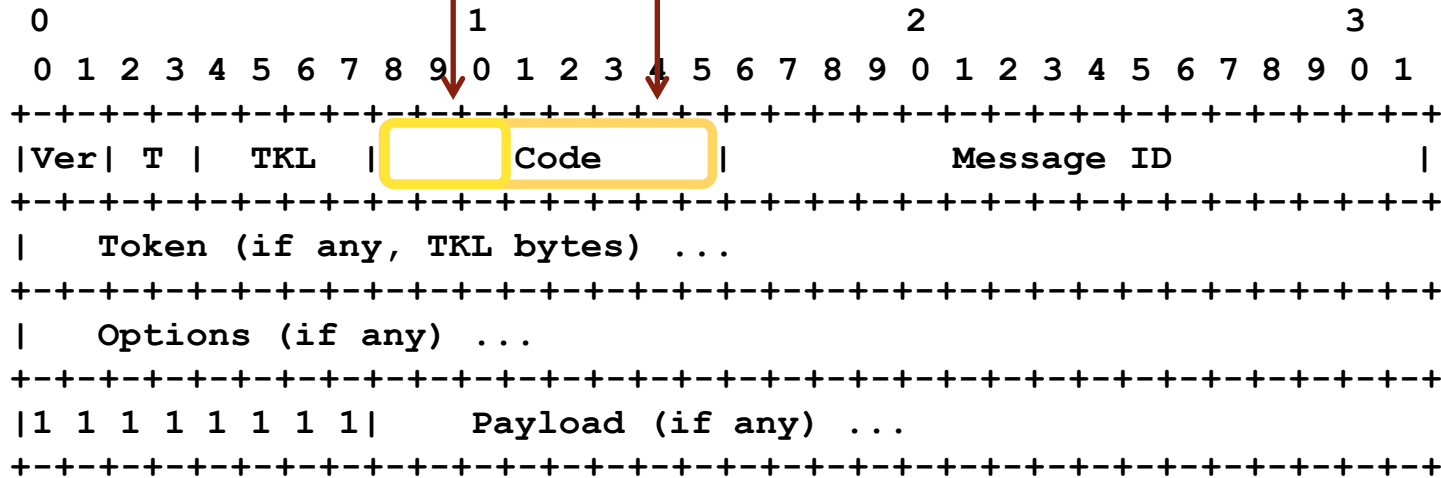


# RESPONSE CODES IN MESSAGE: SERVER ERROR 5.XX

5.00	Internal Server Error (generic response code)
5.01	Not Implemented
5.03	Service Unavailable (uses the Max-Age Option to indicate the number of seconds after which to retry)
...	...



### Figure 9: Structure of a Response Code



# CoAP OPTIONS

option = (option number, option value)

option number	... of option value				
	No.	Name	Format	Length	Default
	1	If-Match	opaque	0-8	(none)
	3	Uri-Host	string	1-255	(see below)
	4	ETag	opaque	1-8	(none)
	5	If-None-Match	empty	0	(none)
	7	Uri-Port	uint	0-2	(see below)
	8	Location-Path	string	0-255	(none)
	11	Uri-Path	string	0-255	(none)
	12	Content-Format	uint	0-2	(none)
	14	Max-Age	uint	0-4	60
	15	Uri-Query	string	0-255	(none)
	17	Accept	uint	0-2	(none)
	20	Location-Query	string	0-255	(none)
	35	Proxy-Uri	string	1-1034	(none)
	39	Proxy-Scheme	string	1-255	(none)
	60	Size1	uint	0-4	(none)

Table 4: Options [RFC7252]

# SELECTED OPTIONS (1/2)

---

- **Content-Format**
  - the representation format of the payload
- **Etag**
  - an entity-tag is intended for use as a resource-local identifier for a specific representation of a resource; generated by the server providing the resource; used for validation
- **Max-Age**
  - the maximum time a response may be cached before it is considered not fresh, default: 60s
- **Accept**
  - in a request, the client can indicate which content-format it prefers to receive

# SELECTED OPTIONS (2/2)

---

**coap-URI = "coap:" "://" host [ ":" port ] path [ "?" query ]**

- **Uri-Host**
  - default: the IP address of the request message
- **Uri-Path**      **Uri-Path is heavily used**
- **Uri-Port**
  - default: the destination UDP port
- **Uri-Query**

# OPTIONS IN MESSAGE

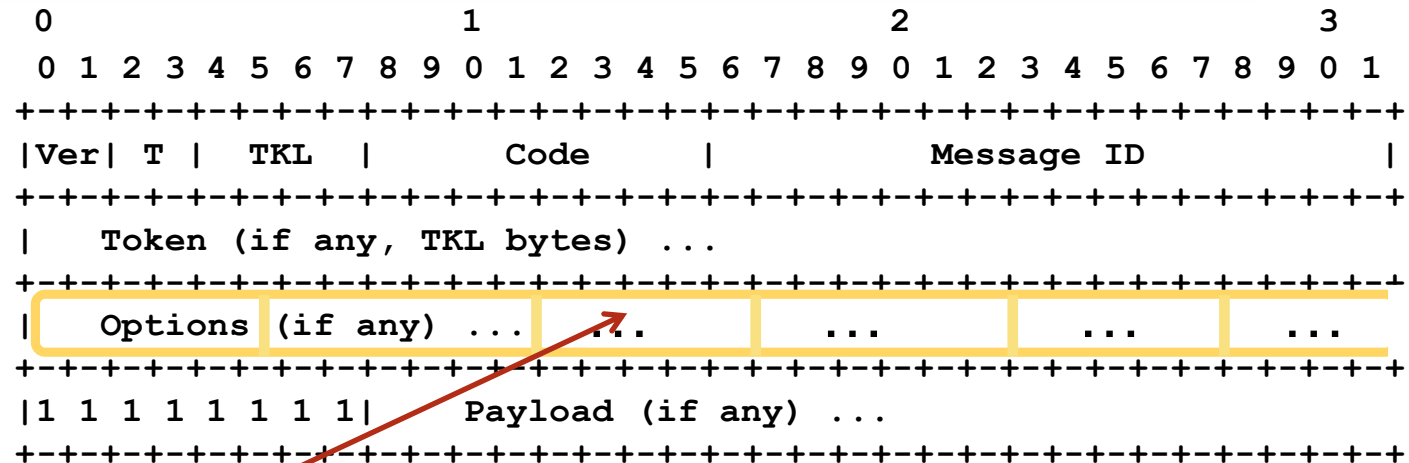


Figure 7: Message Format[RFC7252]

one option:

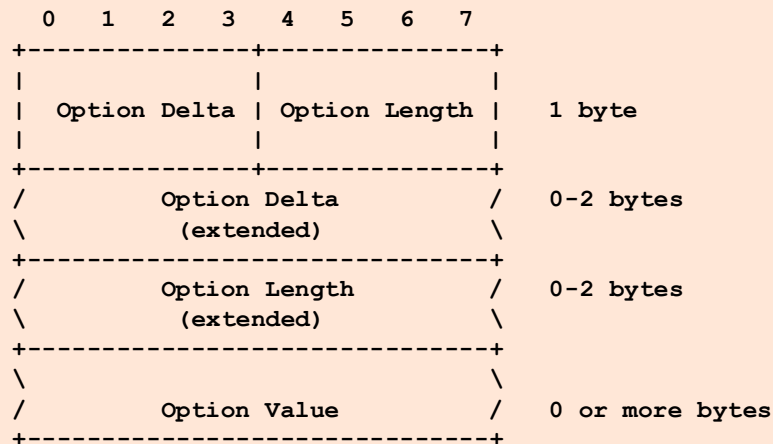


Figure 8: Option Format RFC[7252]

# OPTION FORMAT: NUMBER + VALUE

If 13, one byte extension = Option Delta - 13  
If 14, two byte extension = Option Delta - 269  
15 reserved for payload marker

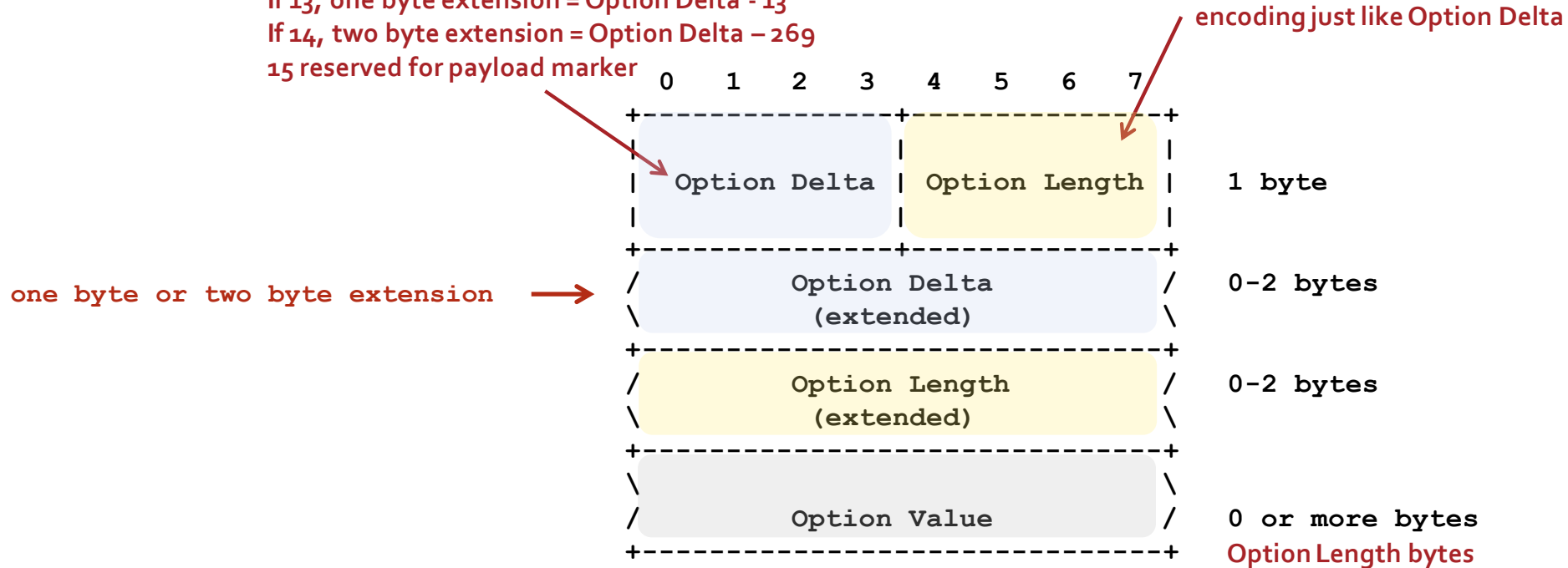


Figure 8: Option Format RFC[7252]

- each option has a number
- a message may contain a sequence of options
- options are ordered according to their numbers (increasing order)
- Option Delta = no. of the current option – no. of the previous one
  - for the first option, Option Delta = no of the first option

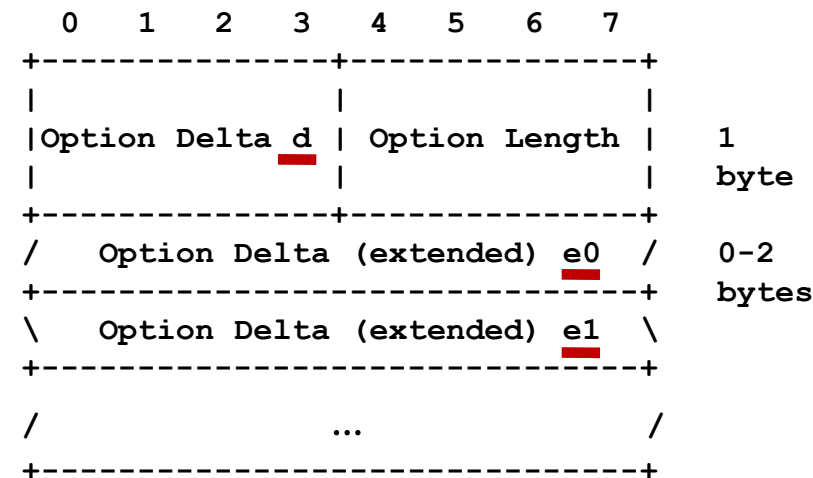
# OPTION FORMAT: DECODING OPTION DELTA

- let  $D$  = Option Delta (to be determined when parsing a message)
- let  $d$  = the Option Delta field in the first byte of the option
- let  $e0$  = the first byte of the Option Delta extended (if present)
- let  $e1$  = the second byte of the Option Delta extended (if present)

- if  $d \leq 12$ 
  - $D=d$ ,  $e0$  missing,  $e1$  missing

- if  $d == 13$ 
  - $D=13+e0$ ,  $e1$  missing  
(so  $13 \leq D \leq 268$ )

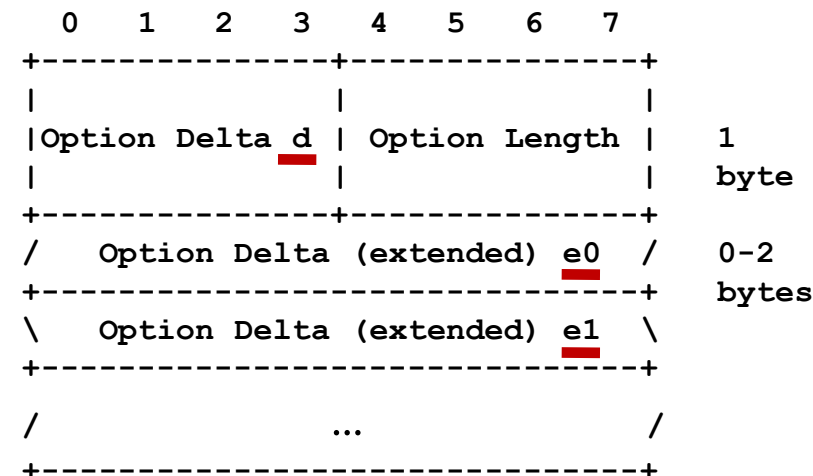
- if  $d == 14$ 
  - $D=269+e0*256+e1$   
(so  $D \geq 269$ )



network byte order: the first byte,  $e0$ , is more significant

# OPTION FORMAT: ENCODING OPTION DELTA

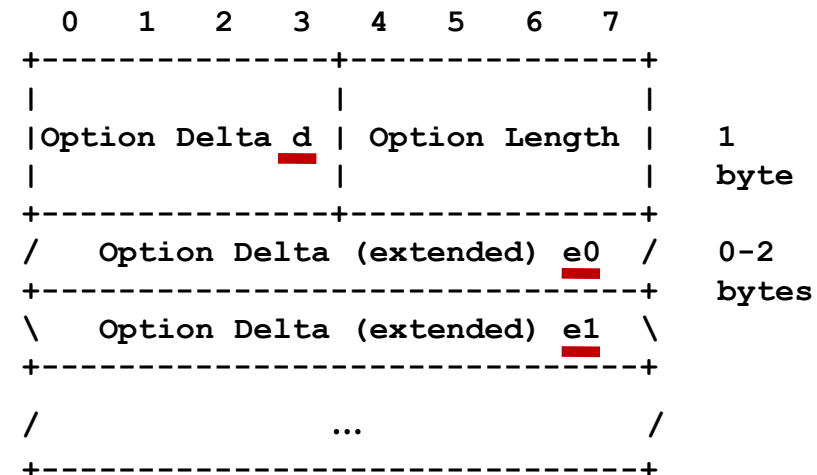
- let  $D$  = Option Delta (to be encoded when assembling a message)
- let  $d$  = the Option Delta field in the first byte of the option
- let  $e0$  = the first byte of the Option Delta extended (if present)
- let  $e1$  = the second byte of the Option Delta extended (if present)
- if  $D \leq 12$ 
  - $d=D$ ,  $e0$  missing,  $e1$  missing
- if  $13 \leq D \leq 268$ 
  - $d=13$ ,  $e0=D-13$ ,  $e1$  missing
- if  $D \geq 269$ 
  - $d=14$ ,  $e0=(D-269)/256$ ,  $e1=(D-269)\%256$





# OPTION FORMAT: OPTION DELTA EXAMPLES

D	d	e0	e1
7	7	-	-
13	13	0	-
17	13	4	-
268	13	255	-
269	14	0	0
270	14	0	1
524	14	0	255
525	14	1	0



# OPTIONS: ASSORTED CAVEATS

---

- If the option value format is `uint`, leading zero bytes should not be included in the value.
  - in particular, if the option value is zero, the value should be empty (no bytes, Option Length equal to zero)
- "each Uri-Path Option specifies one segment of the absolute path to the resource" RFC[7252]
  - one Uri-Path option per segment of the path (not the entire path)
  - for a path that includes multiple segments, a CoAP request will include multiple Uri-Path options (in the "path order")
  - the Option Delta for the second and following Uri-Path options is zero
  - we say that the Uri-Path option is **repeatable** (RFC[7262])

# PAYLOAD

---

- possible payloads:
  - a resource representation
  - diagnostic payload (in case of error)
- resource representation
  - format is specified by the Internet media type given by the **Content-Format** Option
- diagnostic payload (when no **Content-Format** option is given)
  - the payload of responses indicating a client or server error is a brief human-readable diagnostic message, explaining the error situation

# CONTENT FORMATS (CONTENT-FORMAT OPTION)

used for CoAP resource discovery

Media type	Encoding	ID	Reference
text/plain;	-	0	[RFC2046] [RFC3676]
charset=utf-8			[RFC5147]
application/link-format	-	40	[RFC6690]
application/xml	-	41	[RFC3023]
application/octet-stream	-	42	[RFC2045] [RFC2046]
application/exi	-	47	[REC-exi-20140211]
application/json	-	50	[RFC7159]

Table 9: CoAP Content-Formats [RFC7252]

Efficient XML Interchange (binary)

Concise Binary  
Object  
Representation

## 7.4. CoAP Content-Format

Media Type: application/cbor

Id: 60

Source: *Concise Binary ObjectRepresentation (CBOR)* , [RFC7049]

# PARSING EXAMPLE: MESSAGE

coap://coap.me:5683/location1

press here to see the log

The screenshot shows the CoAP client interface with the URL `coap://coap.me:5683/location1`. The main display shows a successful **2.05 Content (Blockwise) (Download finished)** response. The response details are as follows:

Header	Value	Option	Value	Info
Type	ACK	ETag	0xE6FAA2746E460698	
Code	2.05 Content	Content-Format	application/link-format	
MID	21199	Block2	0 (128 B/block)	
Token	0xC EE5			

The payload is 43 bytes, and the response is rendered. The CoAP Message Log at the bottom shows the following messages:

Time	CoAP Message	MID	Token	Options	Payload
13:10:04	CON-GET	21199 (0)	0xC EE5	If-Match: 0xE6FAA2746E460698, Uri-Path: location1, Block2: 0/0/128	
13:10:04	ACK-2.05 Content	21199	0xC EE5	ETag: 0xE6FAA2746E460698, Content-Format: 40, Block2: 0/0/128	</location1/location2>;rt="location2";ct=40

Red annotations highlight the 'Log' button in the top right and the second message in the log, which is the ACK response to be parsed.

this is the message we are going to parse  
(it's a piggybacked response)

# PARSING EXAMPLE: WHAT LOG SAYS

---

UDP: Received 63 bytes

PACKET (hex):

62,45,52,CF,CE,E5,48,E6,FA,A2,74,6E,46,6,98,81,28,B1,3,FF,  
3C,2F,6C,6F,63,61,74,69,6F,6E,31,2F,6C,6F,63,61,74,69,6F,6  
E,32,3E,3B,72,74,3D,22,6C,6F,63,61,74,69,6F,6E,32,22,3B,63  
,74,3D,34,30

...

# PARSING EXAMPLE: HEADER, TOKEN, PAYLOAD

```

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) ...
+-----+-----+-----+-----+

```

UDP: Received 63 bytes

MID=  
 $5 \times 16^3 + //5$   
 $2 \times 16^2 + //2$   
 $12 \times 16 + //C$   
 $15 = //F$   
 21199

T=2  
 Ver=1 (ACK) TKL=2 response code= 2.05 (Content)

PACKET (hex):

4B+ 62, 45, 52, CF, **header** 0110 0010, 0100 0101, 0101 0010, 1100 1111  
 2B+ CE, E5, **token**  
 13B+ 48, E6, FA, A2, 74, 6E, 46, 6, 98, 81, 28, B1, 3, **options (next slide)**  
 1B+ FF, **payload marker**  
 43B 3C, 2F, 6C, 6F, 63, 61, 74, 69, 6F, 6E, 31, 2F, 6C, 6F, 63, 61,  
 = 74, 69, 6F, 6E, 32, 3E, 3B, 72, 74, 3D, 22, 6C, 6F, 63, 61, 74,  
 63B 69, 6F, 6E, 32, 22, 3B, 63, 74, 3D, 34, 30

**payload:**

0x30-ASCII '0'

0x3C-ASCII '<' ~~</location1/location2>;rt="location2";ct=40~~

EIOT, 2020L

# PARSING EXAMPLE: OPTIONS

No.	Name	Format	Length	Default
4	ETag	opaque	1-8	(none)
12	Content-Format	uint	0-2	(none)
23	Block2	uint	0-3	(none)

option delta  
option no.  $0+4=4$  (ETag) **48**, option length

**E6, FA, A2, 74, 6E, 46, 6, 98**, option value (8B)

option delta  
option no.  $4+8=12$  (Content-F) **81**, option length

**28**, option value (1B),  $0x28=40$  application/link-format

option delta  
option no.  $12+11=23$  (Block2) **B1**, option length

**3**, option value (1B), NUM/M/size= 0/0/128

**FF** payload marker – no more options

M=0

NUM=0

**0000**

**0011**

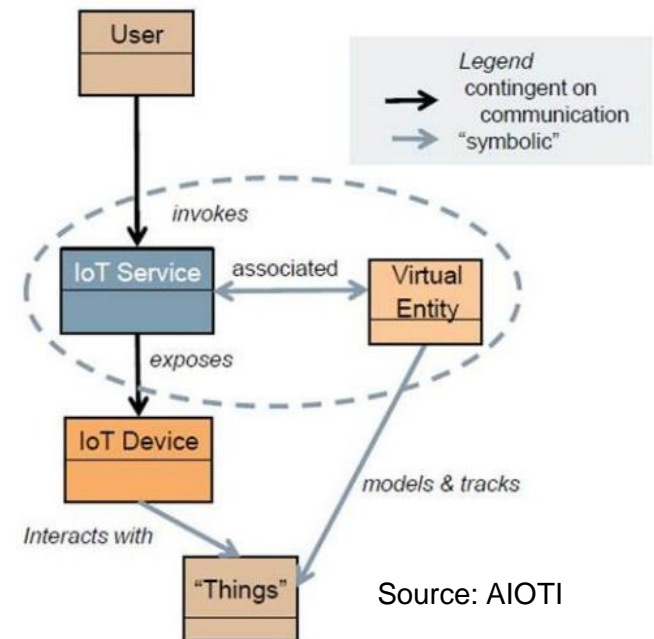
SZX=3, block size  $2^{*(3+4)}=128$

Note: the Block2 option is covered below.



# Examples

49



# CON REQUEST; PIGGYBACKED RESPONSE

Client	Server
+----->	Header: GET (T=CON, Code=0.01, MID=0x7d34)
GET	Uri-Path: "temperature"
<-----+	Header: 2.05 Content (T=ACK, Code=2.05, MID=0x7d34)
2.05	Payload: "22.3 C"

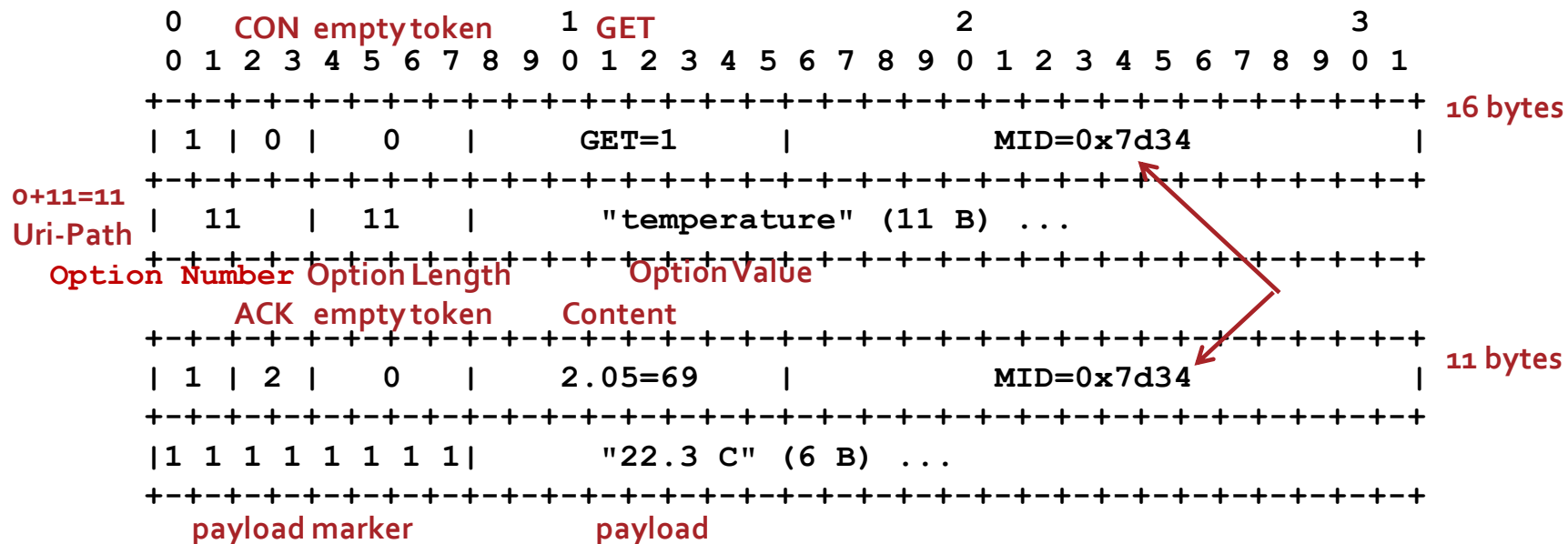


Figure 16: Confirmable Request; Piggybacked Response [RFC7252]

# CON REQUEST; PIGGYBACKED RESPONSE, WITH TOKEN

Client    Server

```
|      |
+----->|      Header: GET (T=CON, Code=0.01, MID=0x7d35)
| GET   |      Token: 0x20
|      |      Uri-Path: "temperature"
|      |
|<-----+      Header: 2.05 Content (T=ACK, Code=2.05, MID=0x7d35)
| 2.05  |      Token: 0x20
|      |      Payload: "22.3 C"
|      |
```

```
0  CON tokenlength  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=0x7d35   | 17 bytes
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   0x20 token |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 11   | 11   |   "temperature" (11 B) ...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

ACK tokenlength
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 1 | 2 |   1 |   2.05=69   |   MID=0x7d35   | 12 bytes
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   0x20 token |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|1 1 1 1 1 1 1| "22.3 C" (6 B) ...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Figure 17: Confirmable Request; Piggybacked Response, with token [RFC7252]

# CON REQ. RETRANSMITTED; PIGGYBACKED RESPONSE

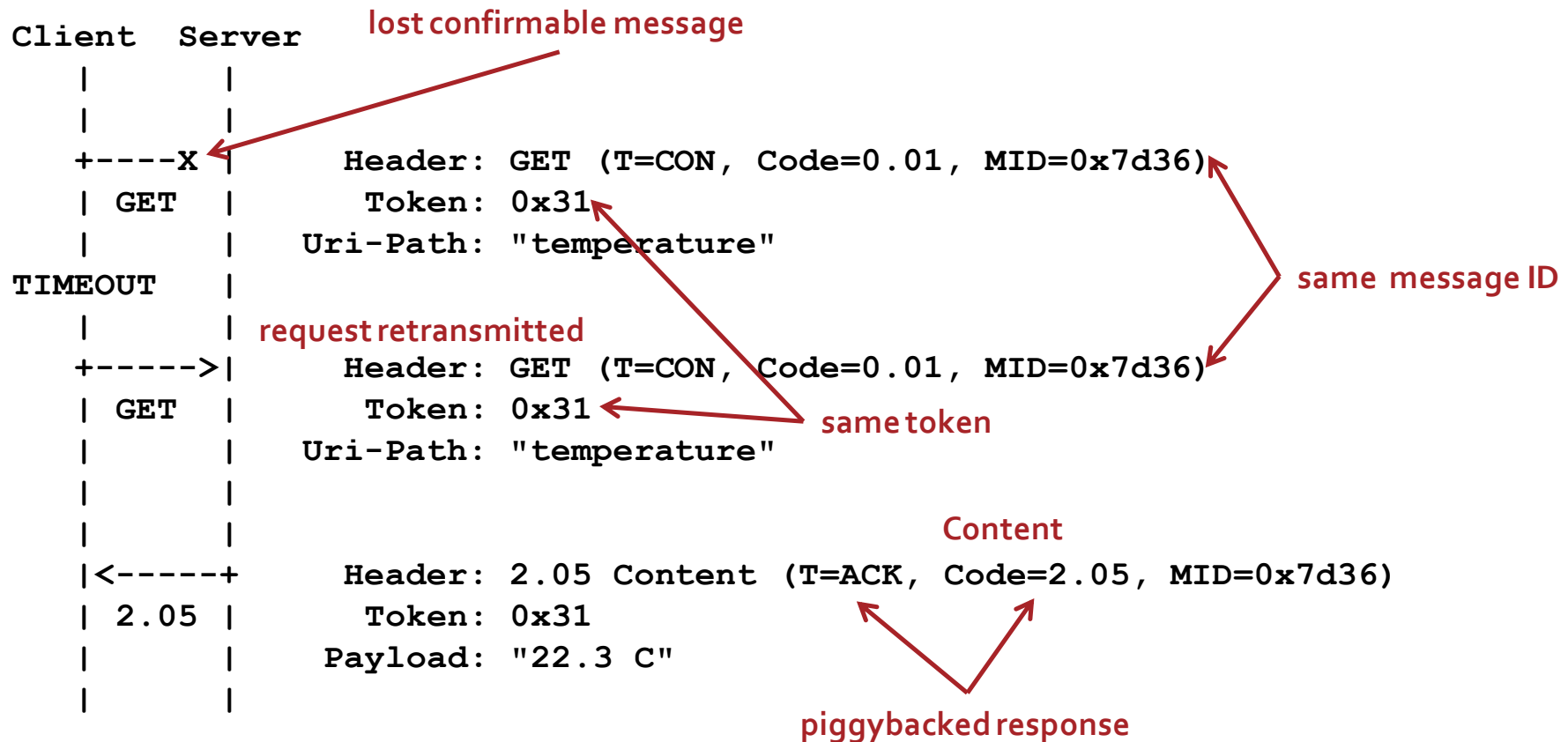


Figure 18: Confirmable Request (Retransmitted) ; Piggybacked Response [RFC7252]

# CON REQ.; PIGGYBACKED RESPONSE RETRANSMITTED

Client	Server
+----->	Header: GET (T=CON, Code=0.01, MID=0x7d37)
GET	Token: 0x42
	Uri-Path: "temperature"
X-----+	Header: 2.05 Content (T=ACK, Code=2.05, MID=0x7d37)
2.05	Token: 0x42
	Payload: "22.3 C"
TIMEOUT	
	request retransmitted
	all messages: same message ID, same token
+----->	Header: GET (T=CON, Code=0.01, MID=0x7d37)
GET	Token: 0x42
	Uri-Path: "temperature"
<-----+	Header: 2.05 Content (T=ACK, Code=2.05, MID=0x7d37)
2.05	Token: 0x42
	Payload: "22.3 C"



Figure 19: Confirmable Request; Piggybacked Response (Retransmitted) RFC[7252]

# CON REQUEST; SEPARATE RESPONSE

Client      Server

```
Client      Server
|           |
|           |
+----->|   Header: GET (T=CON, Code=0.01, MID=0x7d38)
| GET      |   Token: 0x53
|           |   Uri-Path: "temperature"
|           |
|           |   empty ACK
|<-- --+   Header: (T=ACK, Code=0.00, MID=0x7d38)
|           |
|           |   separate, confirmable response
|<-----+   Header: 2.05 Content (T=CON, Code=2.05, MID=0xad7b)
| 2.05     |   Token: 0x53
|           |   Payload: "22.3 C"
|           |
|           |   empty ACK to confirm confirmable response
+-- -->|   Header: (T=ACK, Code=0.00, MID=0xad7b)
```

matching ACK with its CON

empty ACK

separate, confirmable response

matching a response with its request

empty ACK to confirm confirmable response

Figure 20: Confirmable Request; Separate Response [RFC7252]

# CON REQUEST; SEPARATE RESPONSE (UNEXPECTED)

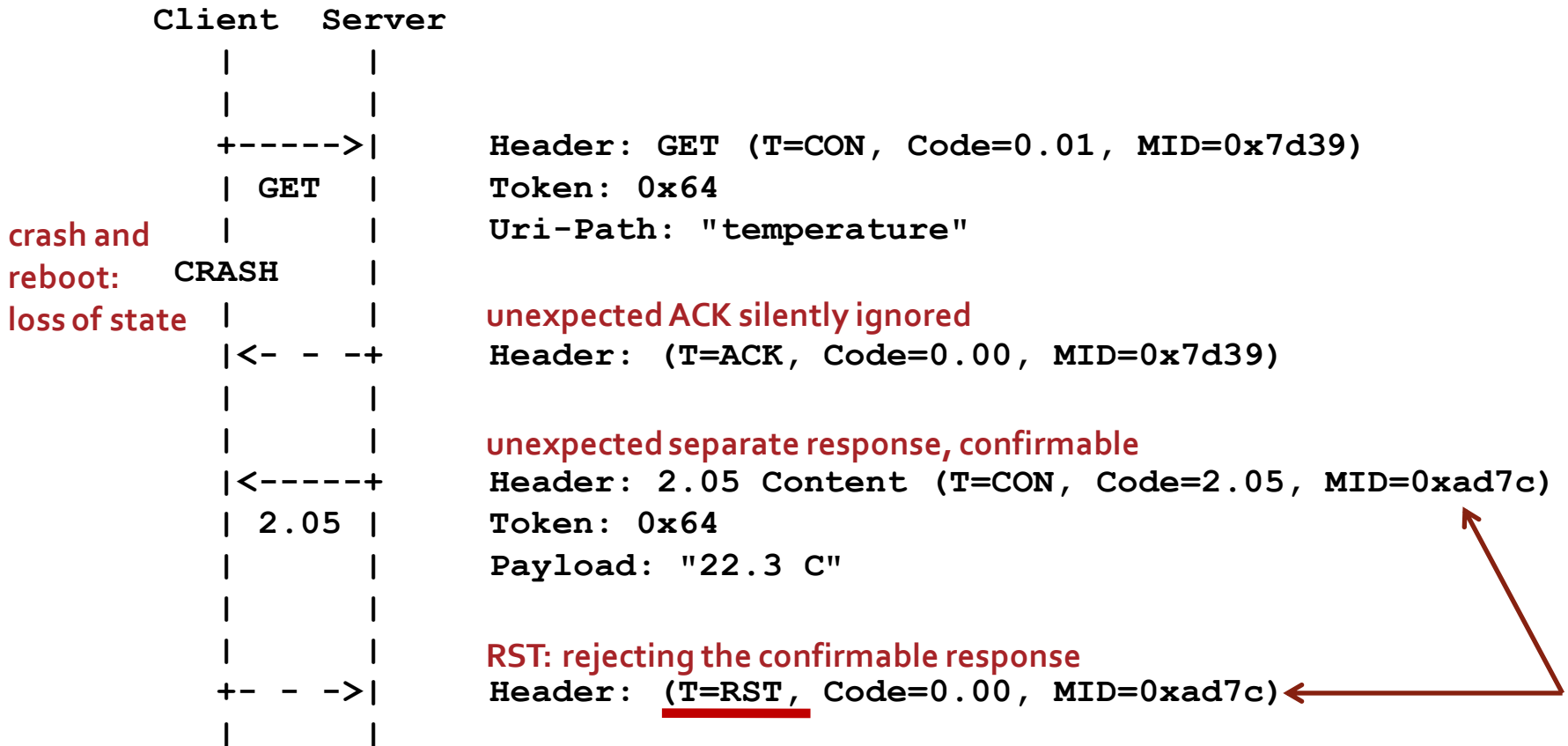
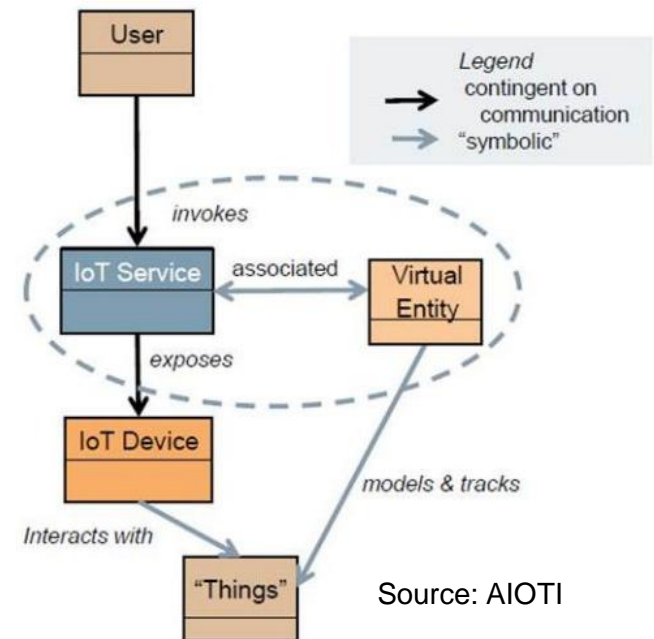


Figure 21: Confirmable Request; Separate Response (Unexpected) [RFC7252]

# Caching and proxying

56





# CACHING

---

- CoAP clients may cache responses in order to reduce the response time and network bandwidth consumption on future, equivalent requests
- reuse a prior response message (a **stored response**)
- two mechanisms: freshness and validation

# CACHING: FRESHNESS OF A STORED RESPONSE

---

- a stored response is reused without contacting the server
  - the Max-Age Option indicates how long the response is fresh
  - default: 60s

# CACHING: VALIDATING A STORED RESPONSE

---

- a client has some stored responses ... but none of them is fresh
- maybe one of the stored responses is can be reused after all?
- need to validate stored responses
- to do so, a new GET is required, but it may turn out that transmitting a payload from the server is not needed
  - responses may be tagged by the server, with the ETag Option
  - the client can inquire if a stored response is valid by sending a **GET** with its **Etag**
  - multiple **Etag**'s may be included if the client has multiple stored responses
  - the server may respond with 2.03 Valid and one of the **Etag**'s (without a payload, but possibly with a **Max-Age** option) ...
  - ... or the server may respond with 2.05 Content and include a new payload

# PROXYING

---

- a proxy is tasked by clients to perform requests on their behalf
- proxy classification 1:
  - **forward proxy**: explicitly selected by clients (as a proxy)
  - **reverse proxy**: the client is not aware that it talks to a proxy
- proxy classification 2:
  - **CoAP-to-CoAP** proxy
  - **cross proxy**: translates from or to a different protocol
- proxies can cache responses

# NO PROXY – JUST ORIGIN SERVER

URI split into the Uri-Host (has a default), Uri-Port (has a default), Uri-Path, and Uri-Query Options

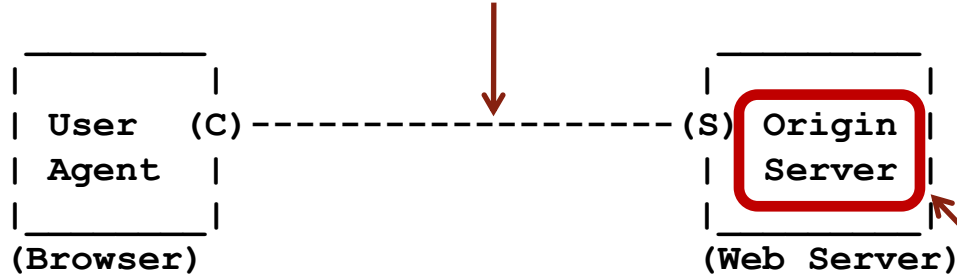


Figure 1: Client-Server Communication

**that's where the resource really is**

Source: *RESTful Design for Internet of Things Systems*  
A. Keranen, M. Kovatsch, K. Hartke,  
Internet -Draft, draft-keranen-t2trg-rest-iot-03, 2016

# REVERSE PROXY: PRETENDS IT IS AN ORIGIN SERVER

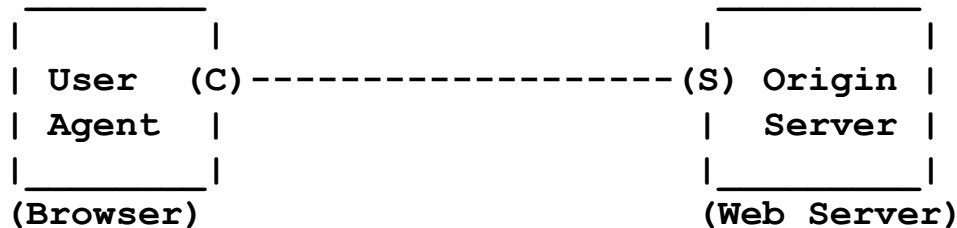


Figure 1: Client-Server Communication

Source: *RESTful Design for Internet of Things Systems*  
A. Keranen, M. Kovatsch, K. Hartke,  
Internet -Draft, draft-keranen-t2trg-rest-iot-03, 2016

the client talks as if to an origin server

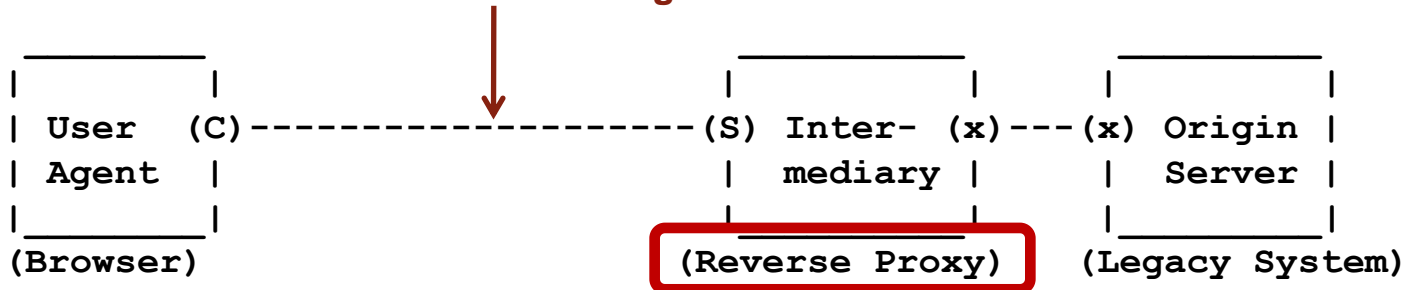


Figure 3: Communication with Reverse Proxy

need an URI mapping for the proxy

# FORWARD PROXY: THE CLIENT KNOWS IT'S A PROXY

Source: *RESTful Design for Internet of Things Systems*  
A. Keranen, M. Kovatsch, K. Hartke,  
Internet -Draft, draft-keranen-t2trg-rest-iot-03, 2016

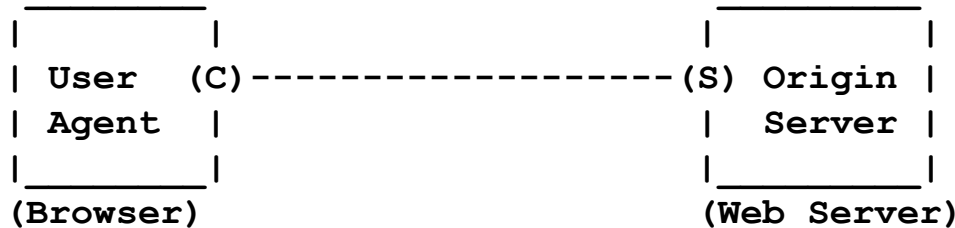


Figure 1: Client-Server Communication

the request URI in a proxy request is in the Proxy-Uri Option

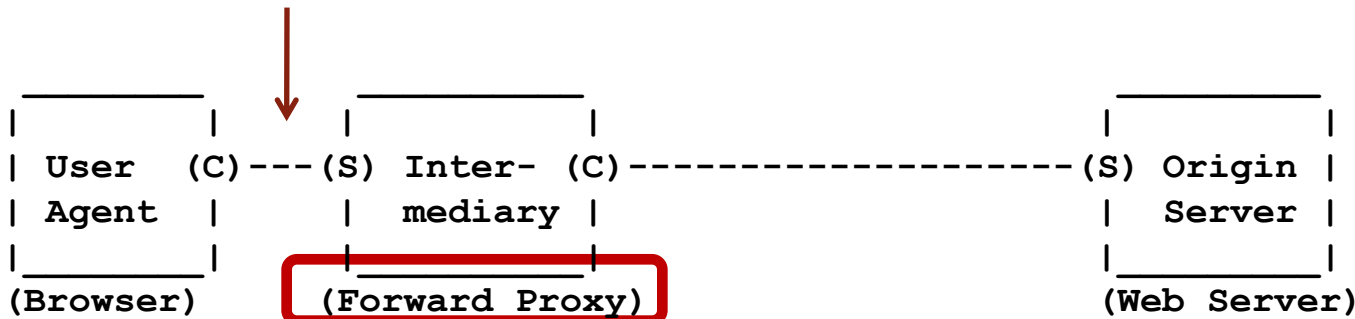


Figure 2: Communication with Forward Proxy

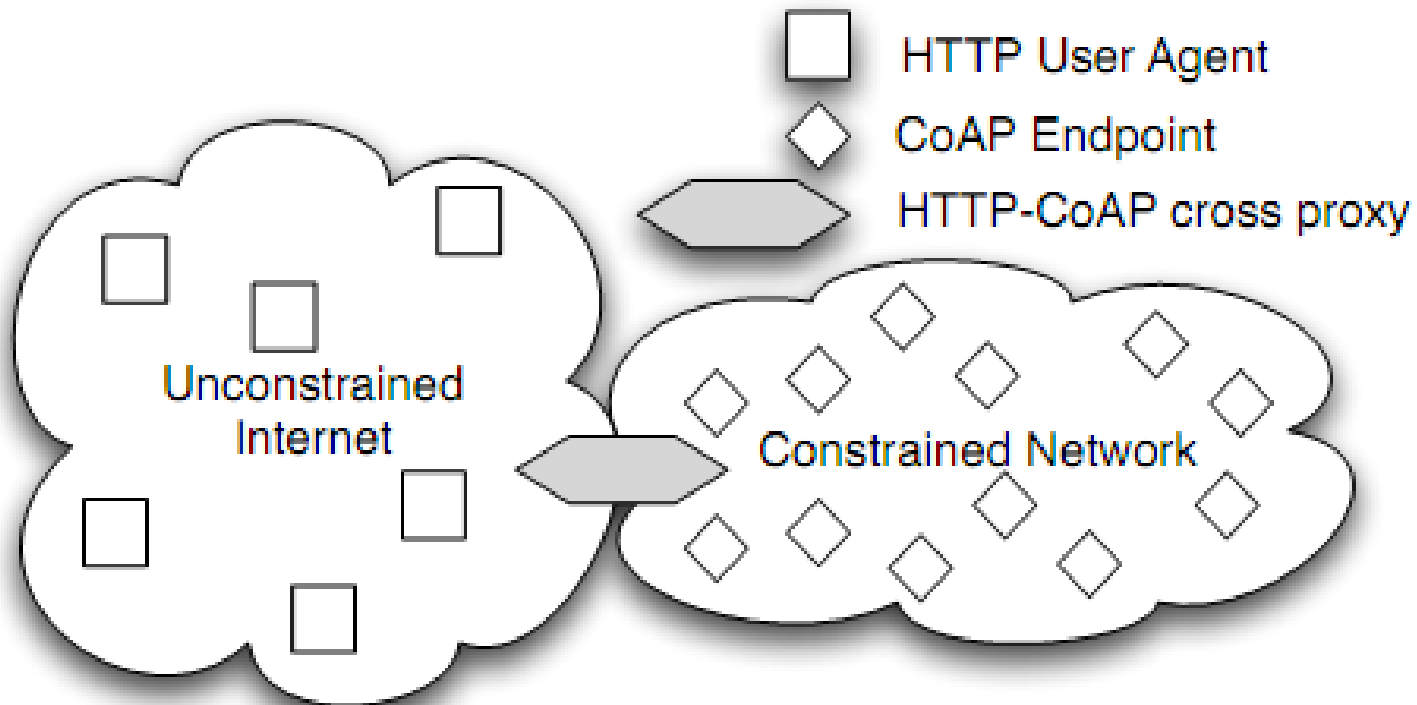
# HTTP-TO-CoAP PROXY (1/2)

Source:

*HTTP-CoAP Cross Protocol Proxy: An Implementation Viewpoint*

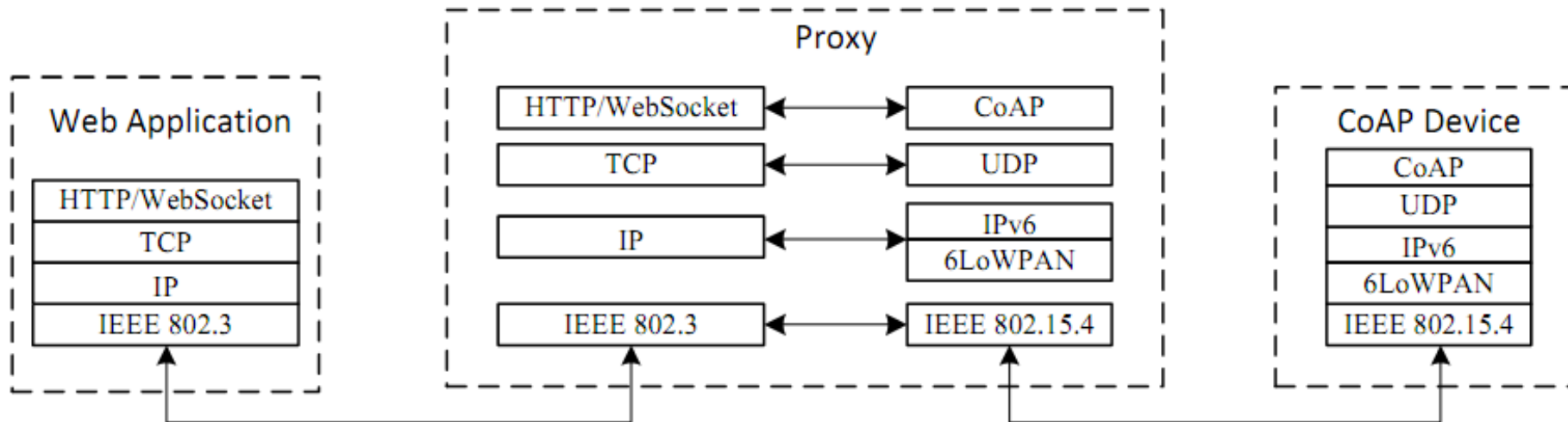
A. P. Castellani, Th. Fossati, S. Loreto

MASS 2012





# HTTP-TO-CoAP PROXY (2/2)



Source:

*A Proxy Design to Leverage the Interconnection of CoAP  
Wireless Sensor Networks with Web Applications*

A. Ludovici, A. Calveras

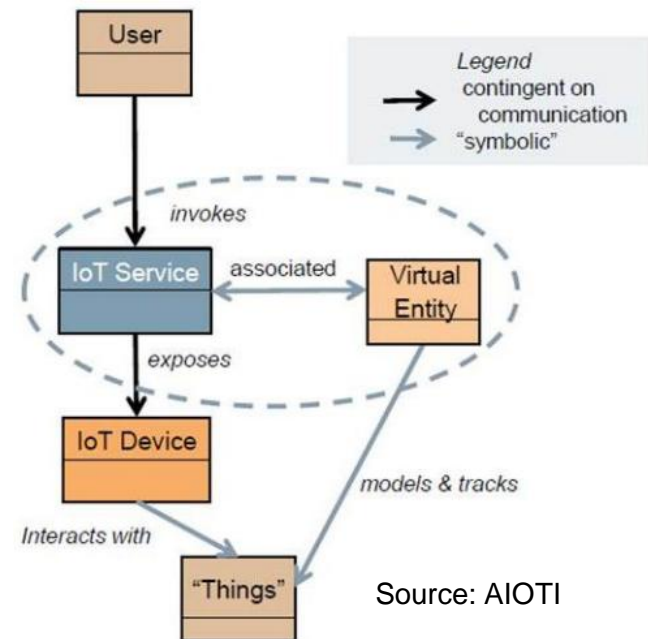
Sensors, 2015

- CoAP is designed with HTTP-to-CoAP proxies in mind

# „Observing” resources

how to be up to date about the state of a resource without too many requests

[RFC7641] "Observing Resources in CoAP" K. Hartke, September 2015



# WHAT'S THE PROBLEM?

- the client/server model does not work well when a client wants to have an up-to-date representation of a resource over a period of time.
- HTTP: polling, long polling
- what's bad: timers, traffic, delays ...
- polling vs. interrupts, pull vs. push

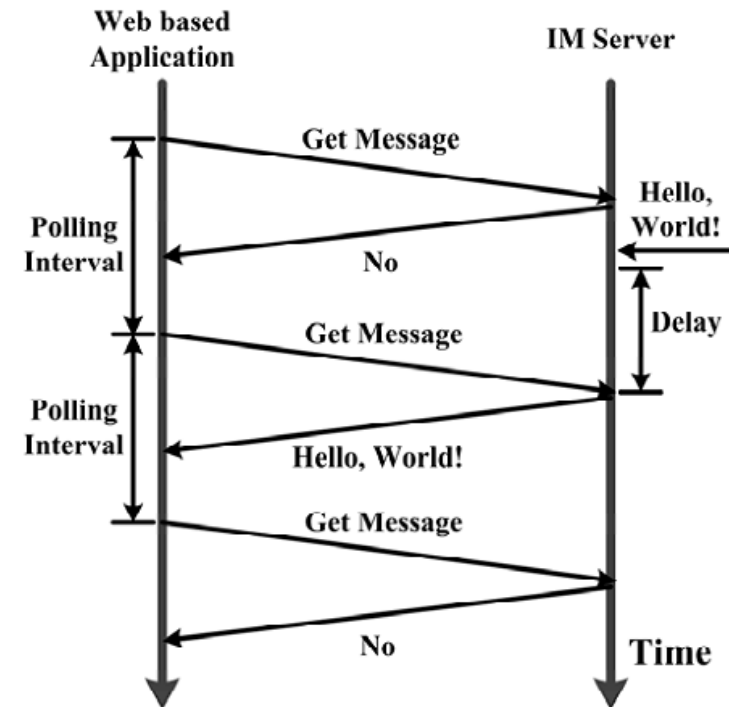


Figure 1. Workflow of HTTP Polling  
Source:  
*Research on Server Push Methods in Web Browser based Instant Messaging Applications*  
Kai Shuang, Feng Kai  
Journal of Software, Vol 8, No 10 (2013)

# OBSERVER DESIGN PATTERN

---

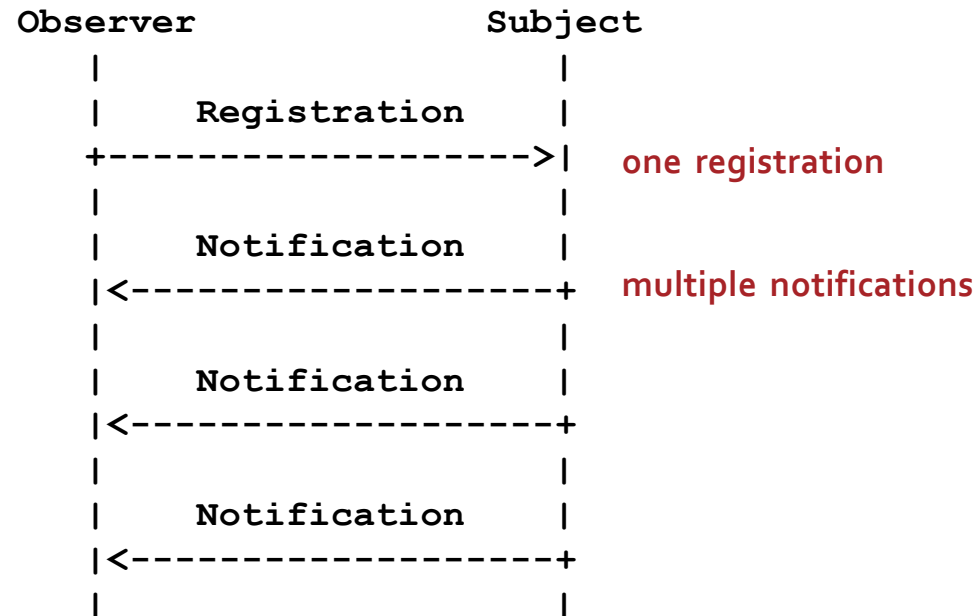


Figure 1: The Observer Design Pattern [RFC7641]

# OBSERVE OPTION

+	-----	+	-----	+	-----	+	-----	+	-----	+
	No.		Name		Format		Length		Default	
+	-----	+	-----	+	-----	+	-----	+	-----	+
	6		Observe		uint		0-3 B		(none)	
+	-----	+	-----	+	-----	+	-----	+	-----	+

Table 1: The Observe Option [RFC7641]

value of the option in GET:

0 add to the list of observers

1 remove from the list of observers

value in response/notification:

sequence number

- a GET request with the Observe Option:
  - retrieves a current representation, but also ...
  - requests the server to add/remove an entry in the **list of observers** of the resource
  - 0 (register) adds the entry to the list, if not present
  - 1 (deregister) removes the entry from the list, if present
- a response with the Observe Option
  - the original response and each subsequent notification
  - the option value is a **sequence number** for reordering detection
  - in every notification the token is as in the original request

# OBSERVING A RESOURCE: EXAMPLE

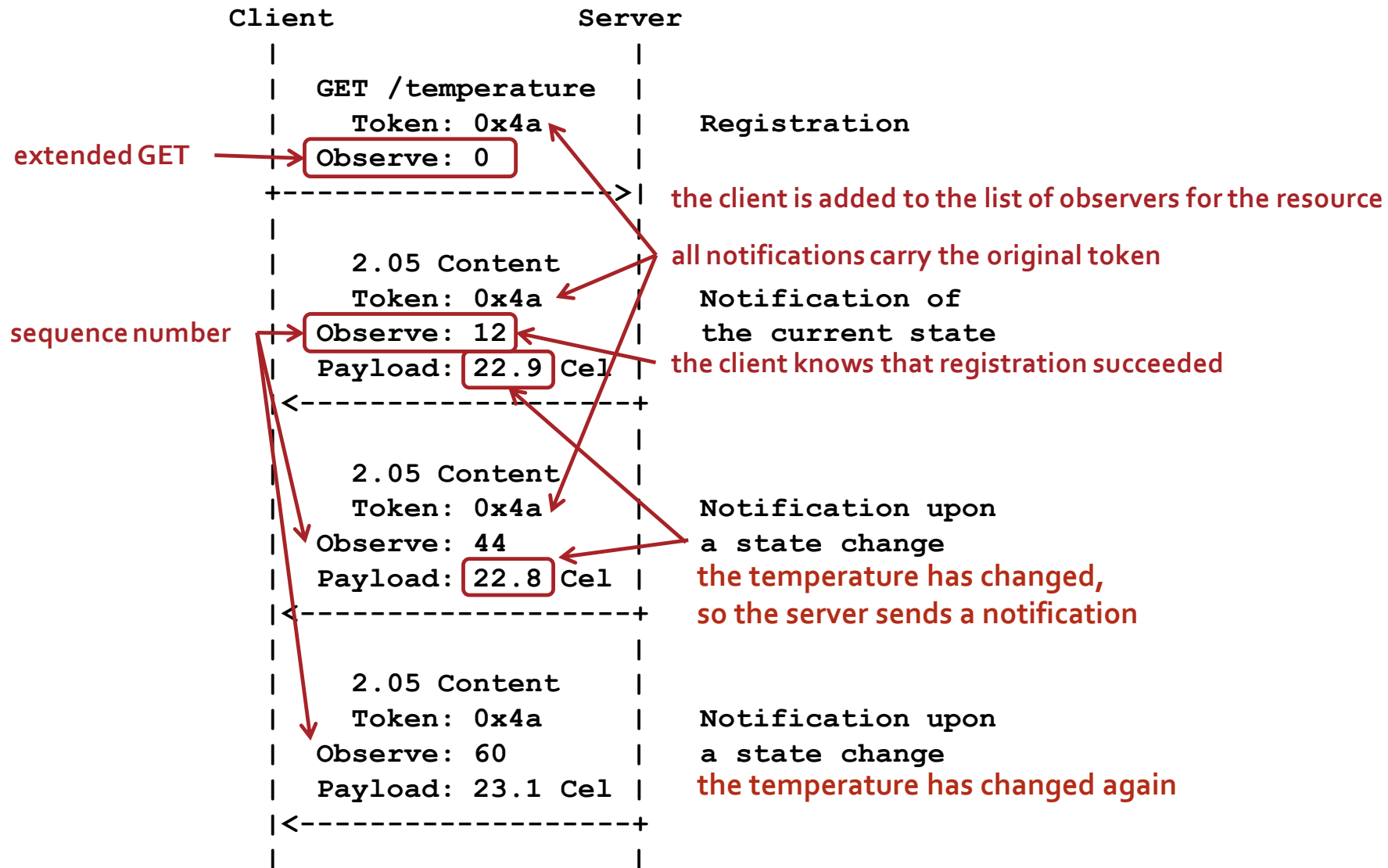


Figure 2: Observing a resource in CoAP [RFC7641]

# LIST OF OBSERVERS

---

- created for a given resource by the server when it receives a GET request with an Observe Option set to 0 (register)
- the list entry consists of the client endpoint and the token specified by the client in the request
- how long a client remains on the list?
  - the server can send a confirmable notification; if it does not receive ACK, it will assume that the client is no longer interested
  - the client may send RST in reaction to a notification
  - the client may deregister with Observe Option set to 1

# CONSISTENCY MODEL: EVENTUAL CONSISTENCY

---

- the goal is to keep the client in sync with the changes in the state of the resource
- sometimes, however, the client gets out-of-sync
  - the server may skip some notifications if changes occur too often
  - notification latency
  - lost notifications
  - the server may decide to drop the client from the list of observers
- the approach in CoAP
  - best effort
  - notifications are labeled with maximum duration
  - the **eventual consistency** model



# WHAT DOES A CHANGE OF STATE MEAN?

---

- the server decides what it means for a resource to change its state (in other words, how to expose an observable resource in a useful way)
- consider temperature (a temperature sensor):
  - **<coap://server/temperature>**
    - changes its state every few seconds to a current reading of the sensor
  - **<coap://server/temperature/felt>**
    - changes its state to "COLD" or "WARM", depending on some thresholds
  - **<coap://server/temperature/critical?above=42>**
    - changes its state either every few seconds to the current temperature reading if the temperature exceeds the client-specified threshold, or to "OK" when the reading drops below

# OBSERVING A RESOURCE: EXAMPLE ...

t	<u>Observed State</u>	CLIENT	SERVER	<u>Actual State</u>
1				
2	unknown			18.5 Cel
3		+----->		Header: GET 0x41011633
4		GET		Token: 0x4a
5				Uri-Path: temperature
6				<u>Observe: 0 (register)</u>
7				
8				
9		<-----+		Header: 2.05 0x61451633
10		2.05		Token: 0x4a
11	18.5 Cel			Observe: 9 <i>yes, you've been registered</i>
12				<b>Max-Age: 15</b>
13				Payload: "18.5 Cel"
14				
15				
16		<-----+		Header: 2.05 0x51457b50
17		2.05	19.2 Cel	Token: 0x4a
18	19.2 Cel			Observe: 16
19				<b>Max-Age: 15</b>
20				Payload: "19.2 Cel"
21				

*freshness (caching)* → **Max-Age: 15**

*sequence numbers* → **Max-Age: 15**

**aside: decode 0x61451633**

Figure 3: A Client Registers and Receives One Notification of the Current State and One of a New State upon a State Change [RFC7641]

t	Observed State	CLIENT	SERVER	Actual State
22				
23	19.2 Cel			19.2 Cel
24				
25	lost notification	X-----+		
26		2.05		19.7 Cel
27				
28				
29	Max-Age seconds			
30	have passed			
31	16+15 = 31			
32				
33	19.2 Cel			
34	(stale)			
35				
36				
37				
38		+----->		
39		GET		
40				
41				
42				
43				
44		<-----+		
45		2.05		
46	19.7 Cel			
47				
48				
49				

a new state

Header: 2.05 0x51457b51  
Token: 0x4a  
Observe: 25  
Max-Age: 15  
Payload: "19.7 Cel"

re-registration

Header: GET 0x41011634  
Token: 0xb2  
Uri-Path: temperature  
Observe: 0 (register)

tagging a response

Header: 2.05 0x61451634  
Token: 0xb2  
Observe: 44  
Max-Age: 15  
ETag: 0x78797a7a79  
Payload: "19.7 Cel"

Figure 4: The Client Re-registers after Max-Age Ends [RFC7641]

t	Observed State	CLIENT	SERVER	Actual State
51				
52	19.7 Cel			19.7 Cel
53				
54				
55			crash	
56				
57	Max-Age seconds			
58	have passed			
59	$44+15 = 59$			
60				
61	19.7 Cel			
62	<u>(stale)</u>			
63			reboot	

Figure 5: The Client Re-registers and Gives the Server the Opportunity to Select a Stored Response (1/2) [RFC7641]

t	Observed State	CLIENT	SERVER	Actual State
63				
64				
65			reboot	
66				20.0 Cel
67		+----->		Header: GET 0x41011635
68		GET		Token: 0xf9
69				Uri-Path: temperature
70				re-registration Observe: 0 (register)
71				ETag: 0x78797a7a79
72				is response with this Etag valid?
73				Content
74		<-----+		Header: 2.05 0x61451635
75		2.05		Token: 0xf9
76	20.0 Cel			Observe: 74
77				no, need to send Max-Age: 15
78				notification Payload: "20.0 Cel"
79				with payload
80				Valid
81		<-----+		Header: 2.03 0x5143aa0c
82		2.03		Token: 0xf9
83	19.7 Cel			Observe: 81
84				ETag: 0x78797a7a79
85				Max-Age: 15
86	stored response			no payload!

Figure 5: The Client Re-registers and Gives the Server the Opportunity to Select a Stored Response (2/2) [RFC7641]

ETag

t	Observed State	CLIENT	SERVER	Actual State
87				
88	19.7 Cel			19.7 Cel
89				
90				
91		<-----+		
92		2.05		19.3 Cel
93	19.3 Cel			
94				
95				
96				
97				
98				
99				
100				
101				
102				
103				
104				19.0 Cel
105				
106				
107				
108	19.3 Cel			
109	(stale)			
110				

regular notification

Header: 2.05 0x4145aa0f

Token: 0xf9

Observe: 91

Max-Age: 15

Payload: "19.3 Cel"

rejecting a notification and thus canceling the observation

Header: 0x7000aa0f

0111 0000 0000 0000

RST

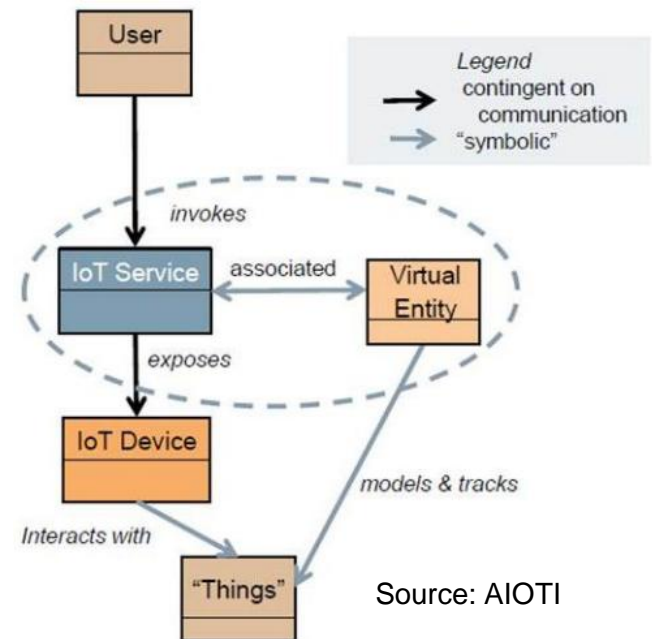
client dropped from list of observers

Figure 6: The Client Rejects a Notification and Thereby Cancels the Observation [RFC7641]

# CoRE Link Format

how to discover resources hosted by a server

[RFC6690] "Constrained RESTful Environments (CoRE) Link Format" Z. Shelby, August 2012



# WEB LINKING

---

- a means of indicating the relationships between Web resources
- **link, typed link** = a typed connection between two resources
- a typed link consists of
  - a **context URI** (by default, the requested resource)
  - a **link relation type** (semantics of a link: how the two resources are related)
  - a **target URI**, and
  - optionally, **target attributes** (key/value pairs).
- a link is the following statement:  
"{context URI} has a {relation type} with resource at {target URI},  
which has {target attributes}"
- HTTP Link header is a serialization of typed links



# WEB LINKING: "REGULAR" USAGE

---

- link relation types registry

- <http://www.iana.org/assignments/link-relations/link-relations.xhtml>

- excerpt:

preview	Refers to a resource that provides a preview of the link's context.	[ <a href="#">RFC6903</a> ], section 3
previous	Refers to the previous resource in an ordered series of resources. Synonym for "prev".	[ <a href="http://www.w3.org/TR/1999/REC-html401-19991224">http://www.w3.org/TR/1999/REC-html401-19991224</a> ]

- more examples of link relations:

- **describedby** refers to a resource providing information about the link's context
  - **start** refers to the first resource in a collection of resources
  - **next** indicates that the link's context is a part of a series, and that the next in the series is the link target
  - **copyright** refers to a copyright statement that applies to the link's context

# WEB LINKING IN CoAP: CoRE RESOURCE DISCOVERY

- a default URI to discover resources hosted by a constrained server: **/.well-known/core**
- the resource at /.well-known/core contains typed links
- the resource at /.well-known/core is serialized using the **CoRE Link Format**
  - carried as payload (in HTTP this is a header)
  - assigned an Internet media type: **application/link-format**

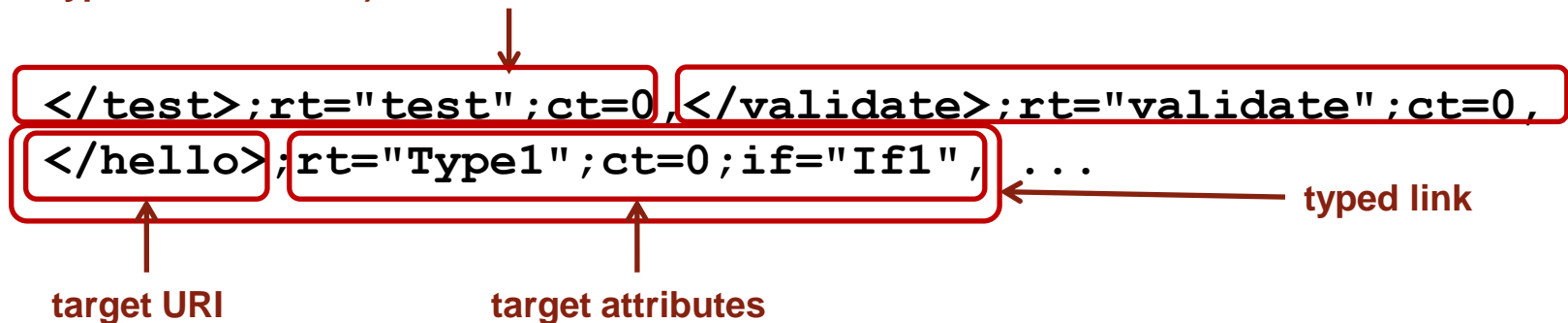
Media type	Encoding	ID	Reference
text/plain;	-	0	[RFC2046] [RFC3676]
charset=utf-8			[RFC5147]
application/link-format	-	40	[RFC6690]
application/xml	-	41	[RFC3023]
application/octet-stream	-	42	[RFC2045] [RFC2046]
application/exi	-	47	[REC-exi-20140211]
application/json	-	50	[RFC7159]

Table 9: CoAP Content-Formats [RFC7252]

# TYPED LINK IN CoRE

- in CoRE, a typed link consists of
  - a context URI (by default: the constrained server)
  - a link relation type (by default: "hosts")
    - "hosts" indicates that the target resource is hosted by the constrained server
  - a target URI (the URI of a resource hosted by the constrained server)
  - target attributes (key/value pairs)

a part of the payload received from `coap://coap.me:5683` in response to `GET /.well_known/core` (three typed links shown)



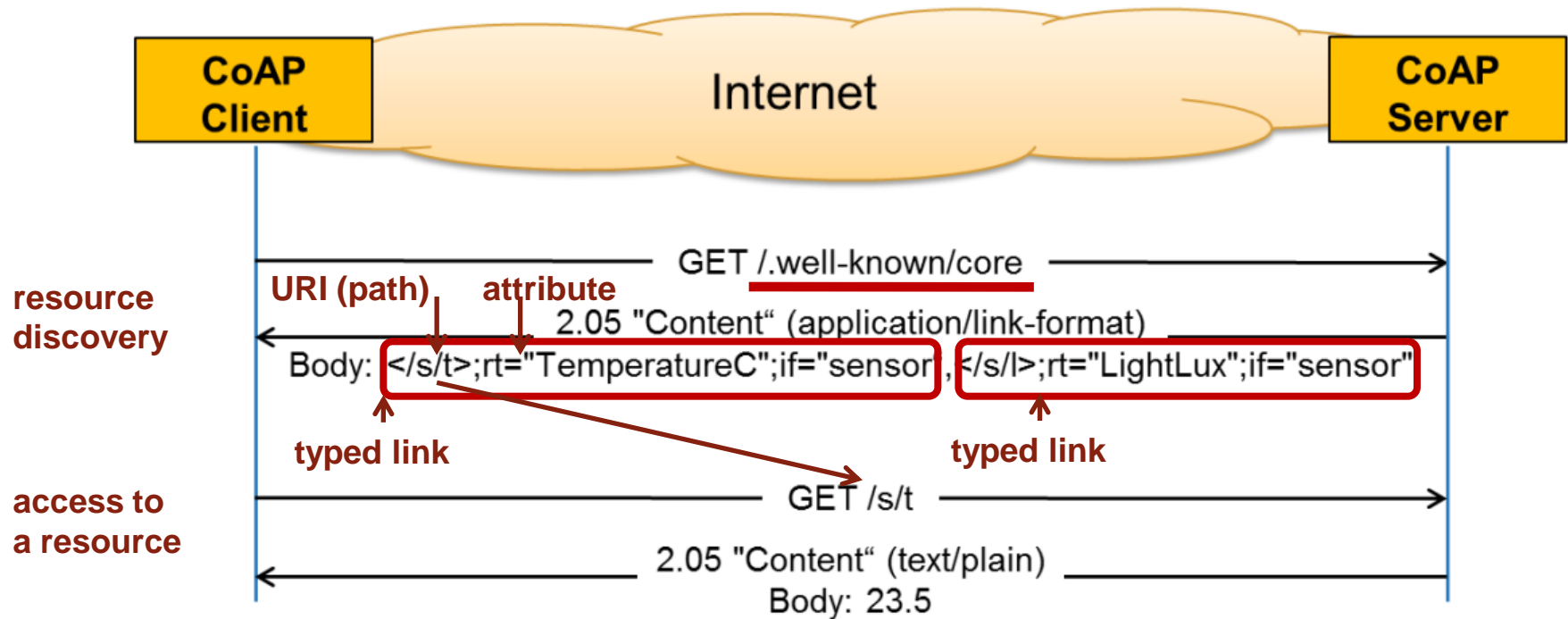
the typed link says: „This server hosts a resource with the URI path `/hello` .  
The resource is characterized by the following values of the attributes `rt`, `ct`, and `if`.”

# TARGET ATTRIBUTES

---

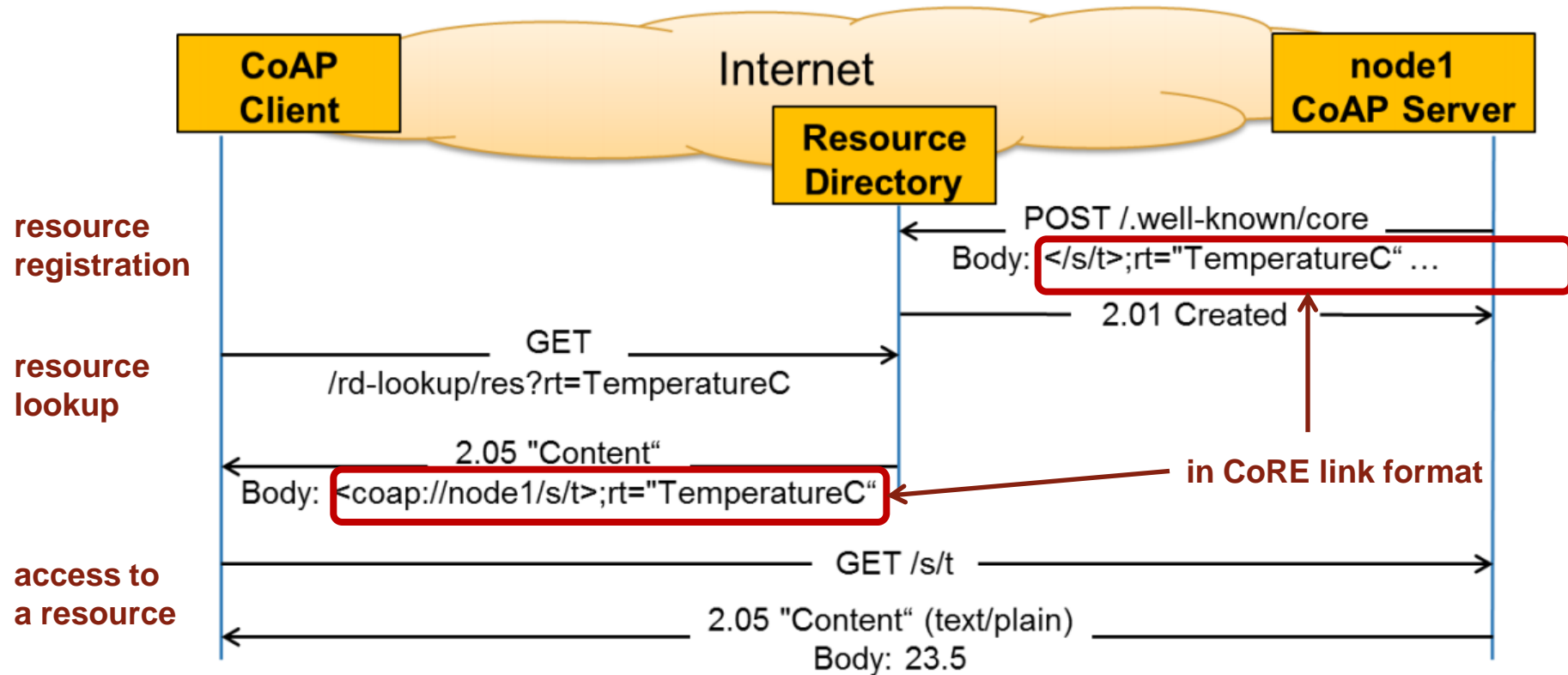
- **Resource Type, rt:** application-specific semantic type of a resource
  - example: outdoor-temperature
  - example: <http://sweet.jpl.nasa.gov/2.0/phys.owl#Temperature>
- **Interface Description, if:** describes the REST interface to interact with a resource
  - example: sensor
  - example: <http://www.example.org/myapp.wadl#sensor>
- **Maximum Size Estimate, sz:** an indication of the maximum size of the resource representation returned by performing a GET
- **Content type, ct:** a hint about the Content-Format this resource returns
- **Observable, obs:** a hint indicating that the resource is useful for observation (not a promise that the Observe Option can be used)

# CoRE RESOURCE DISCOVERY: EXAMPLE



Source: *Flexible Unicast-Based Group Communication for CoAP-Enabled Devices*  
I.Ishaq et al., Sensors, 2014, 14

# CoRE RESOURCE DIRECTORY



Source: *Flexible Unicast-Based Group Communication for CoAP-Enabled Devices*  
I. Ishaq et al., Sensors, 2014, 14

- See [draft-ietf-core-resource-directory-07] Z. Shelby et al. "CoRE Resource Directory", March 2016

# SOME QUESTIONS ON RESOURCE DISCOVERY

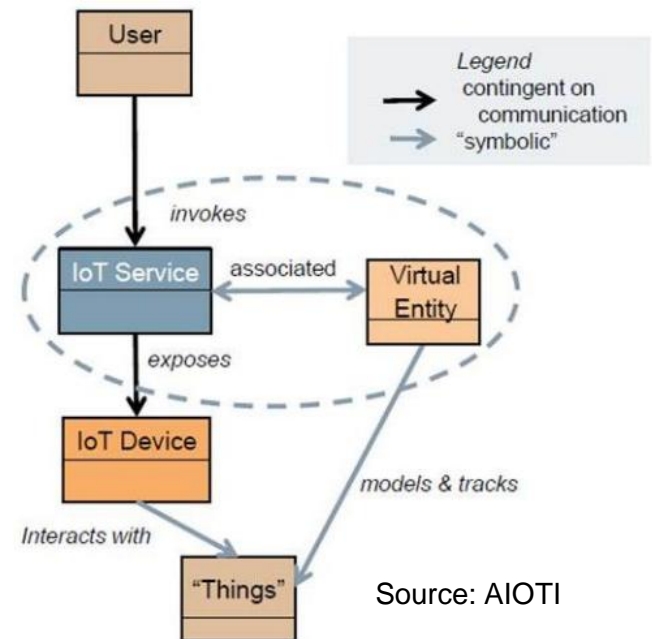
---

- how to identify resource types (rt=) and describe interfaces (if=)?
- how to ensure that the client "understands" these attributes?
- aside: what does it mean to "understand"?
- the resource discovery mechanism described so far may not be enough
- answer: **semantics**

# Block-Wise Transfers in CoAP

how to transfer big resource representations

[RFC7959] "Blockwise Transfers in CoAP" C. Bormann, Z. Shelby, August 2016

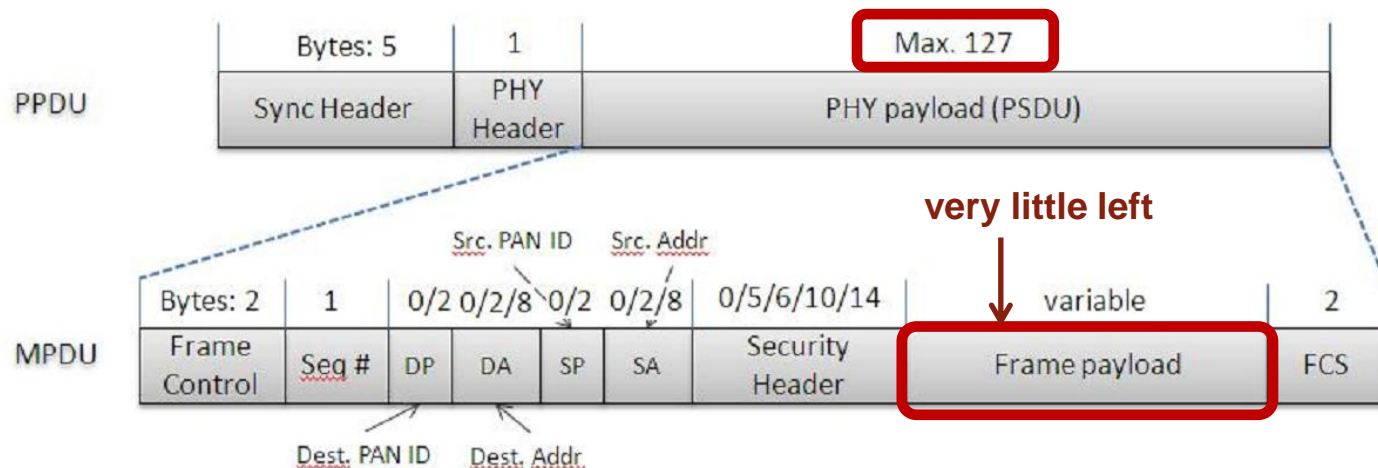




# BLOCK-WISE TRANSFERS (1/2)

- Problem? Consider a popular IoT connectivity option (IEEE 802.15.4)

**Figure 3.** IEEE 802.15.4 The *Physical Protocol Data Unit (PPDU)* and *MAC Protocol Data Unit (MPDU)* formats.



Source: *IETF Standardization in the Field of the Internet of Things (IoT): A Survey*  
Isam Ishaq et al., J. Sens. Actuator Netw. 2013, 2, 235-287

- Objective: avoid fragmentation in lower layers.

# BLOCK-WISE TRANSFERS (2/2)

---

- Solution: let the application layer protocol send a large resource representation piece-by-piece, one piece (block) at a time.

a large resource representation



a large resource representation split into blocks



# Block1, Block2 OPTIONS

---

	No.	Name	Format	Length	Default
GET (payload in response)	23	Block2	uint	0-3	(none)
POST, PUT (payload in request)	27	Block1	uint	0-3	(none)

Table 1: Block Option Numbers [RFC7959]

## OPTIONS Block1, Block2: OPTION VALUE

```

0
0 1 2 3 4 5 6 7
+--+--+--+--+--+--+--+
|  NUM  |M|  SZX  |
+--+--+--+--+--+--+--+

```

**SZX "exponent" for the size of the block**

## M are there more blocks?

**NUM** block sequence number (requested or provided)

```

0                                     1
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|          NUM          |M|  SZX  |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

```

[illegible]

### Figure 1: Block Option Value [RFC7959]

- size of the block =  $2^{*(SZX + 4)}$ 
  - $SZX = 0, 1, \dots, 6$  (7 is reserved)
  - size = 16, 32, ..., 1024
- in examples, option **Block<n>** is shown as **n: NUM/M/<size>**

# OPCJA Block2: EXAMPLE (SIMPLE BLOCKWISE GET)

Block2 – has to do with response payload (transferred from the server to the client)

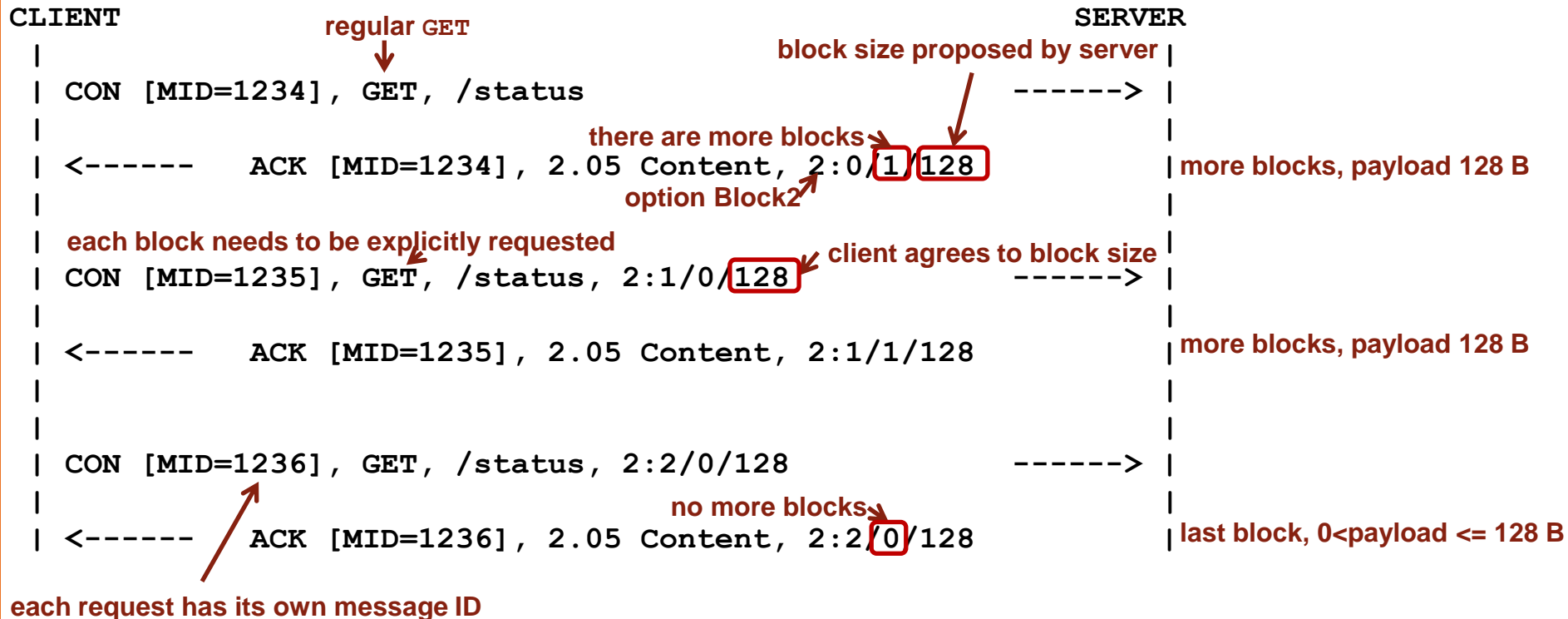


Figure 2: Simple Block-Wise GET [RFC7959]

# OPCJA BLOCK2: EXAMPLE (EARLY NEGOTIATION)

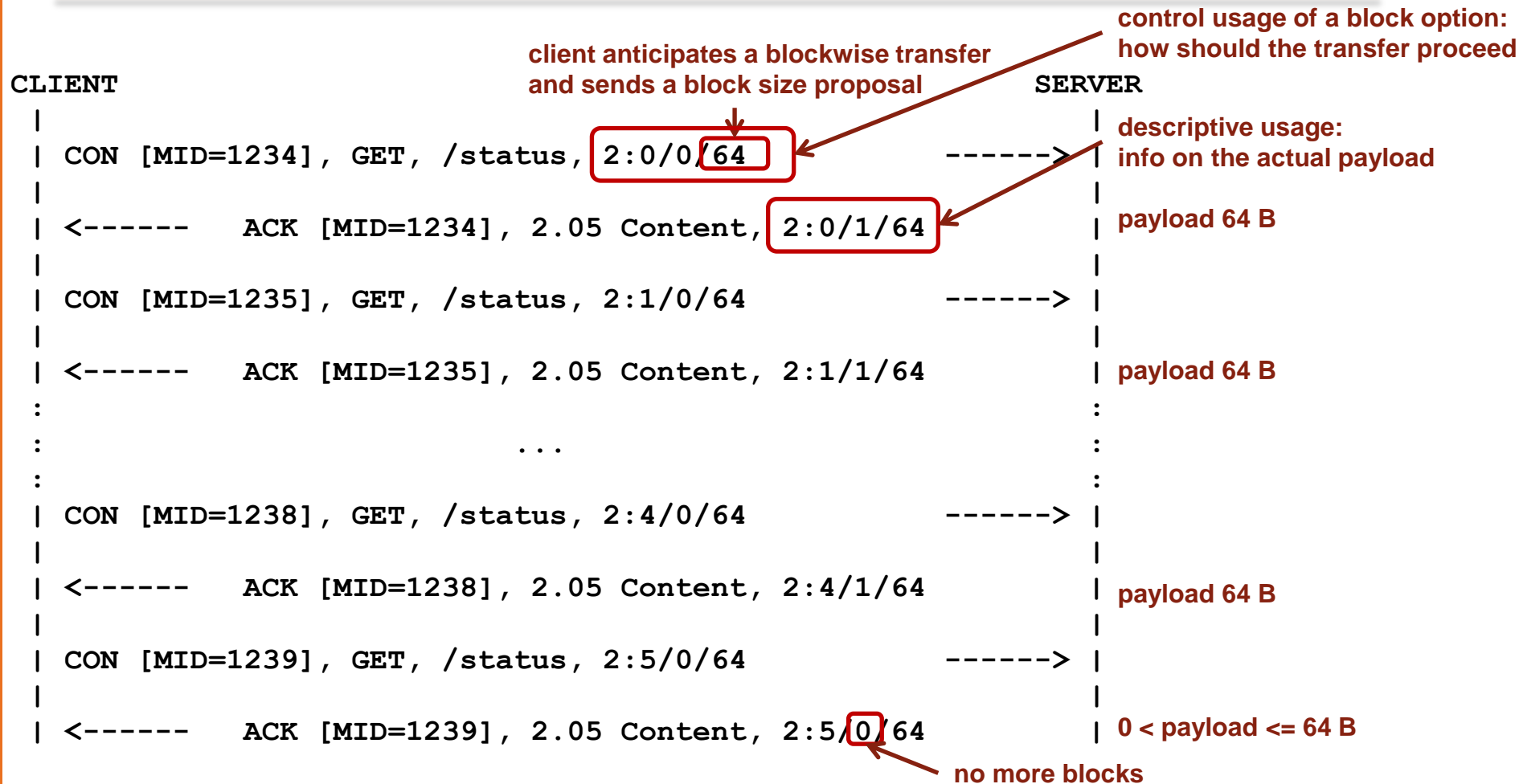


Figure 3: Block-Wise GET with Early Negotiation [RFC7959]

# OPCJA Block2: EXAMPLE (LATE NEGOTIATION)

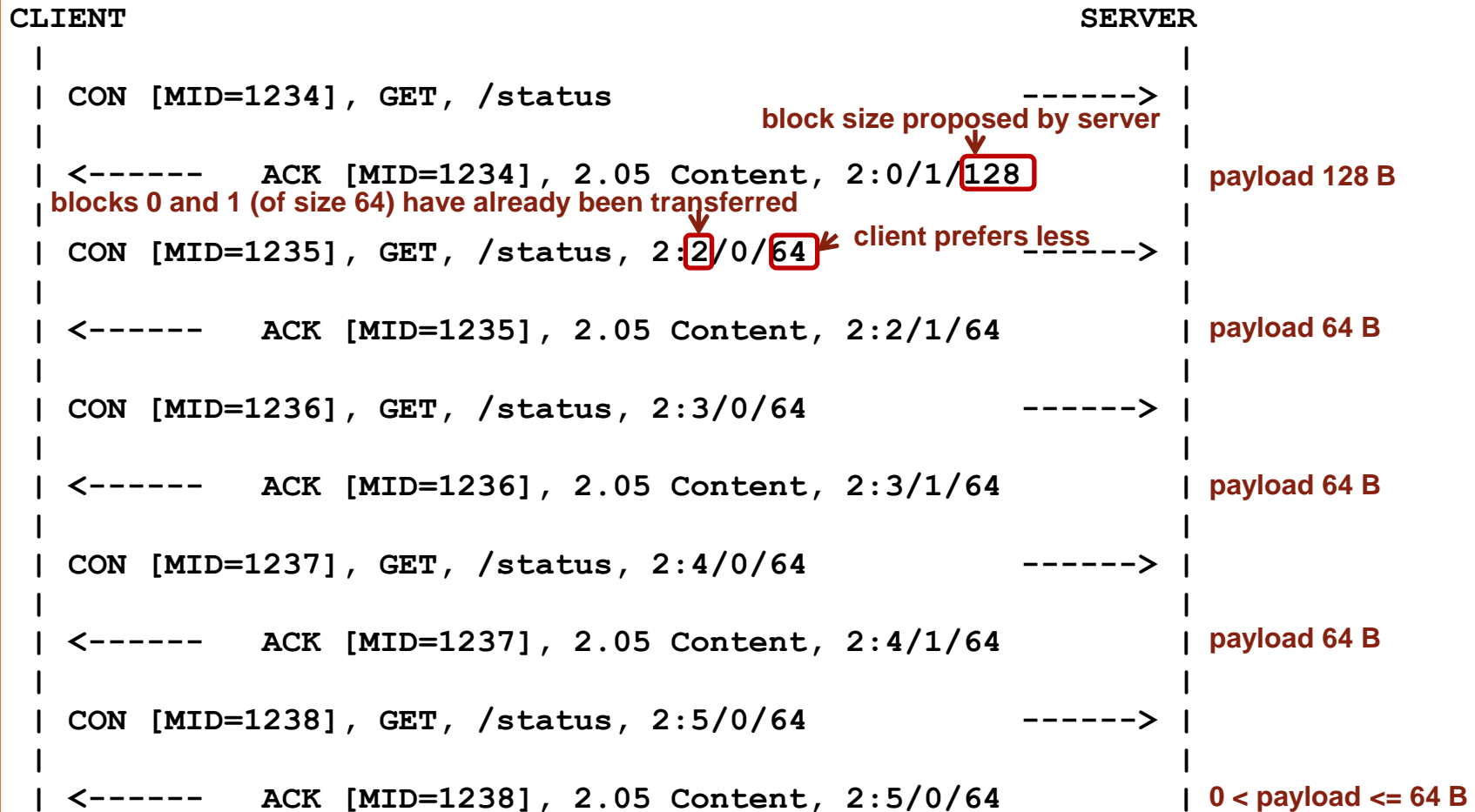


Figure 4: Block-Wise GET with Late Negotiation [RFC7959]

# OPCJA BLOCK2: EXAMPLE (LOST CON)

- Retransmisje bez zmian

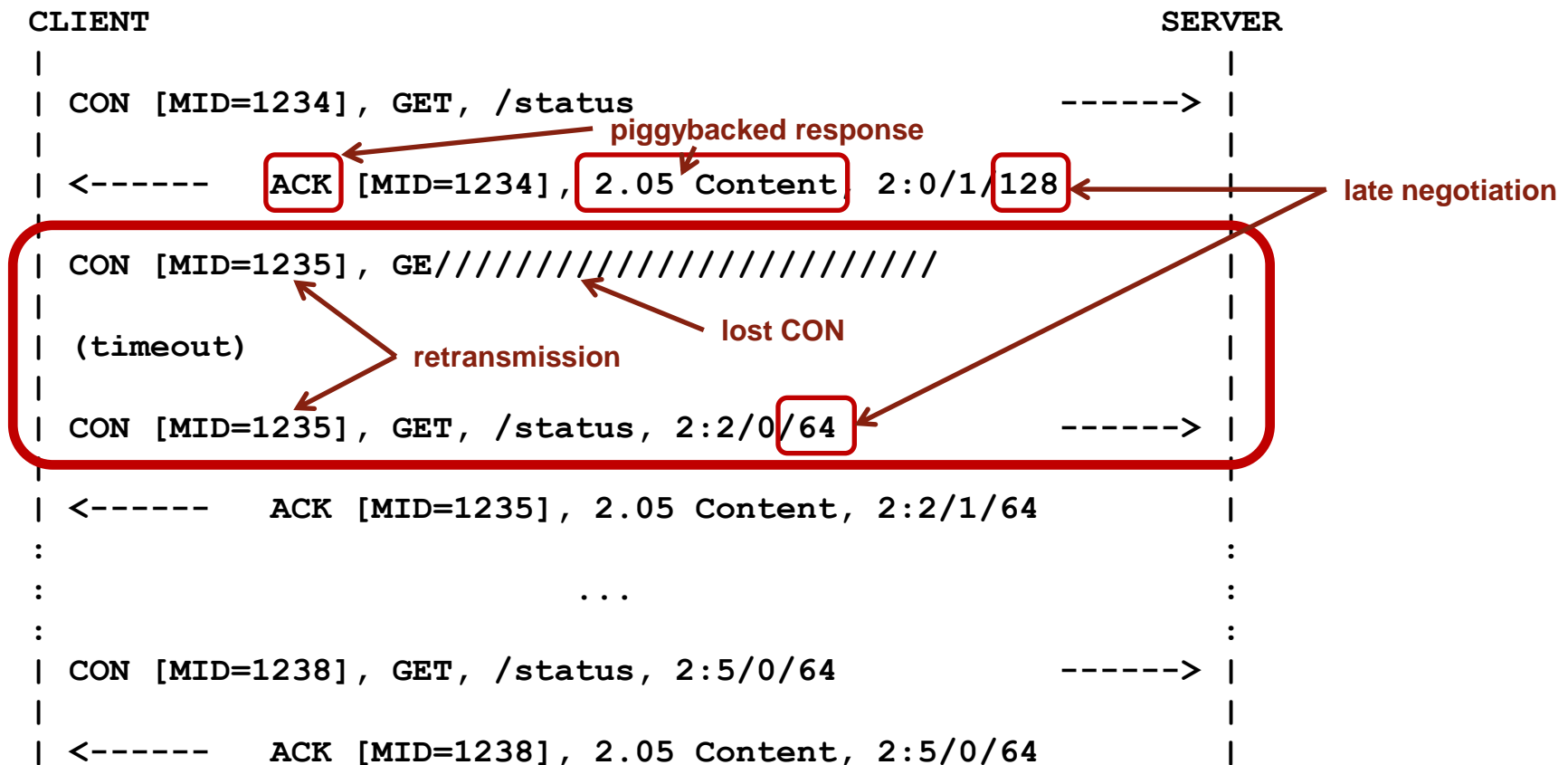


Figure 5: Block-Wise GET with Late Negotiation and Lost CON [RFC7959]



# OPCJA BLOCK2: EXAMPLE (LOST ACK)

- Retransmisje bez zmian

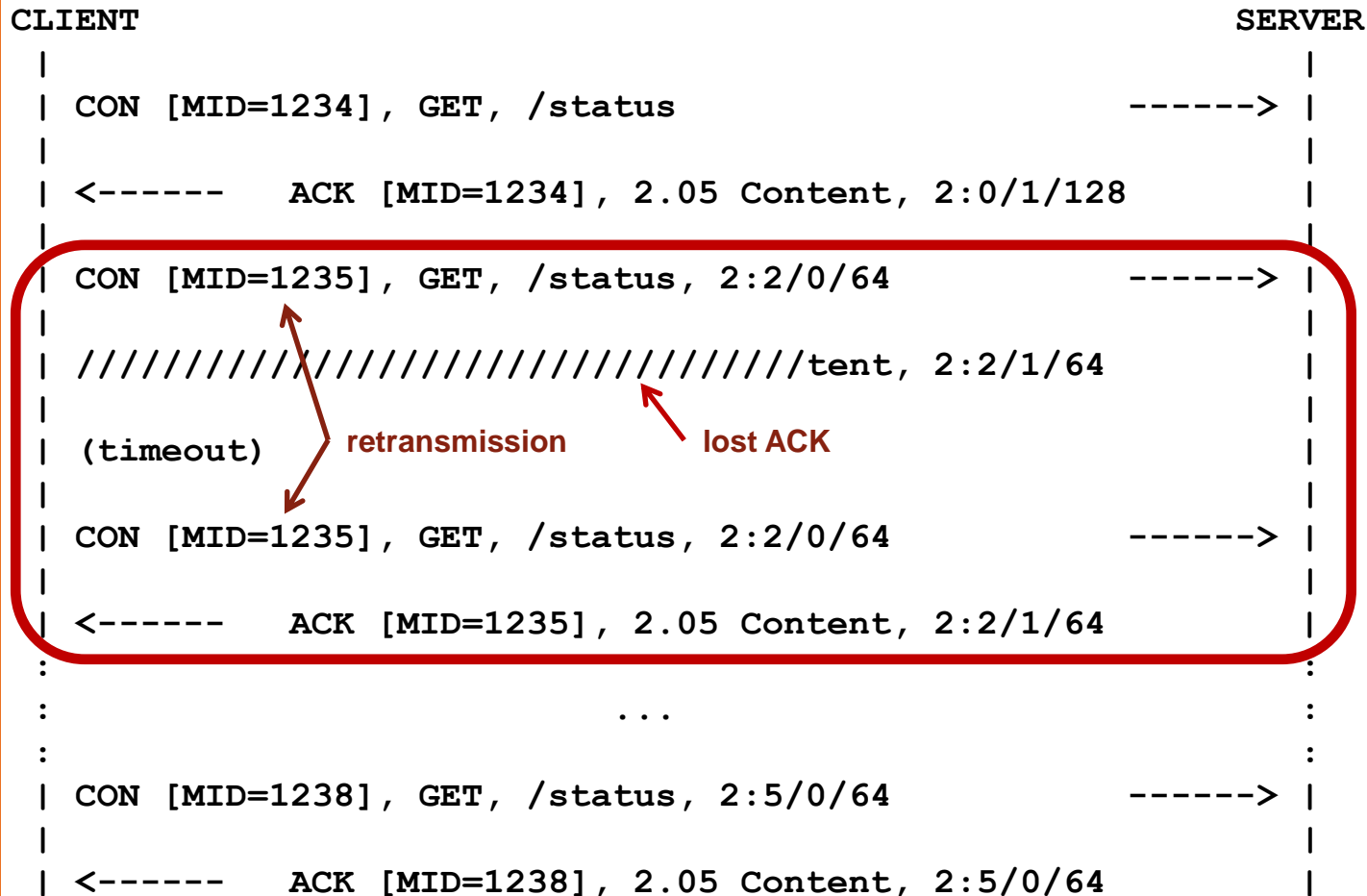


Figure 6: Block-Wise GET with Late Negotiation and Lost ACK [RFC7959]

# OPCJA Block1: EXAMPLE

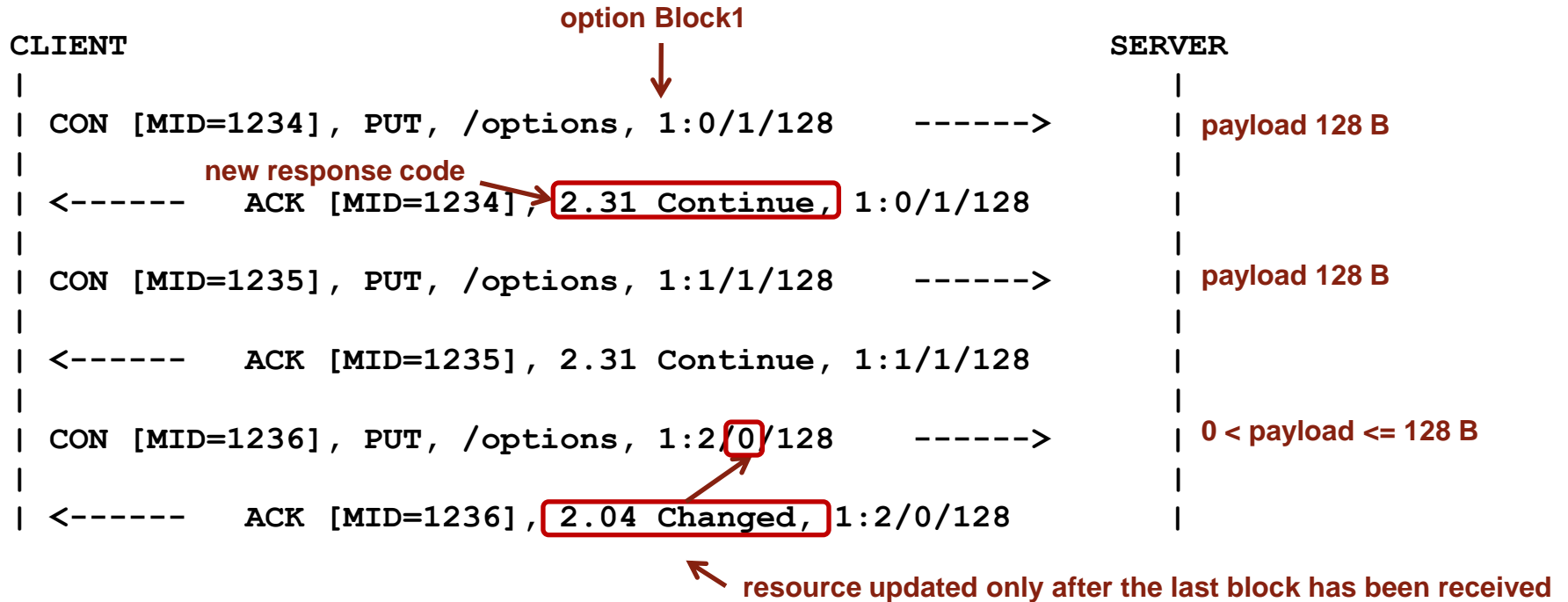


Figure 7: Simple Atomic Block-Wise PUT [RFC7959]

# OPCJE `SIZE1`, `SIZE2`

---

- Całkowity rozmiar reprezentacji zasobu
- `size1` – gdy reprezentacja przesyłana w zapytaniach
- `size2` – gdy reprezentacja przesyłana w odpowiedziach

# Thank you!

