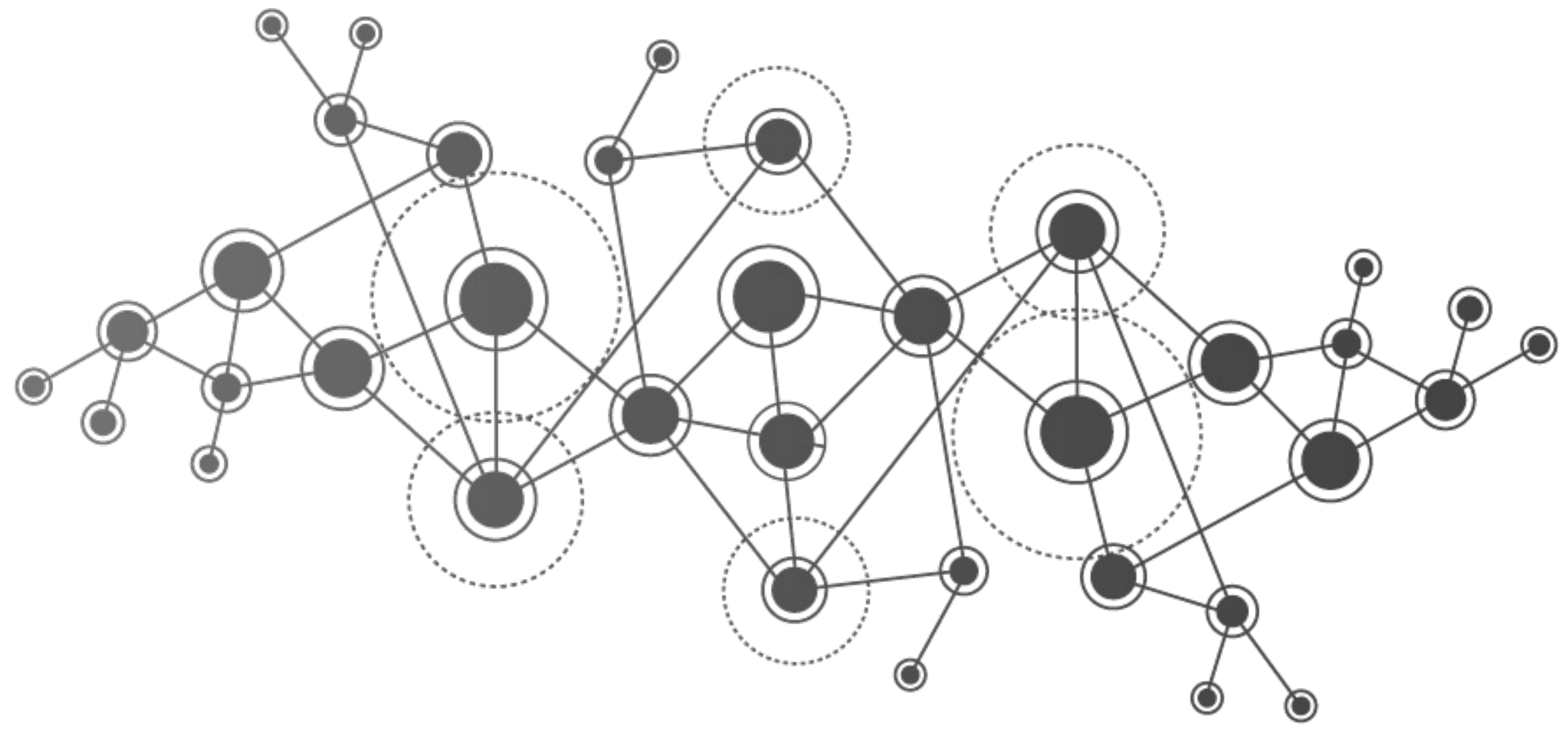
UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA



Intelligent Internet of Things

IoT Application Protocol - CoAP

Prof. Marco Picone

A.A 2023/2024

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

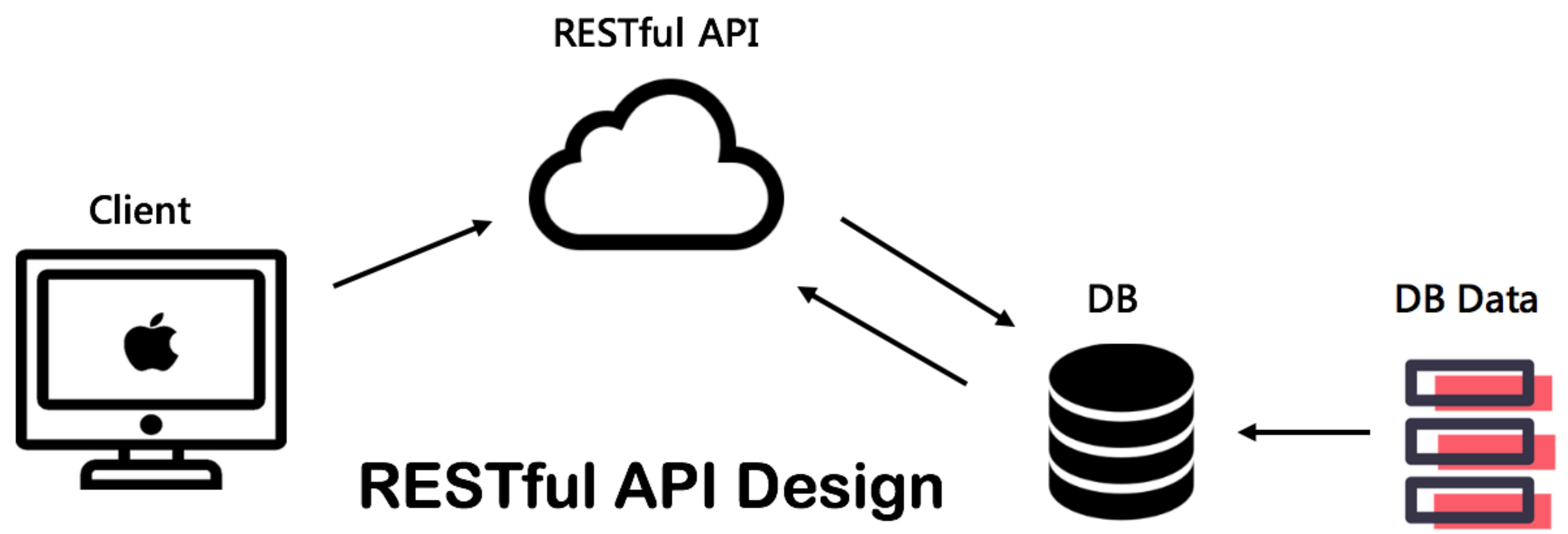
IoT Application Protocol - CoAP

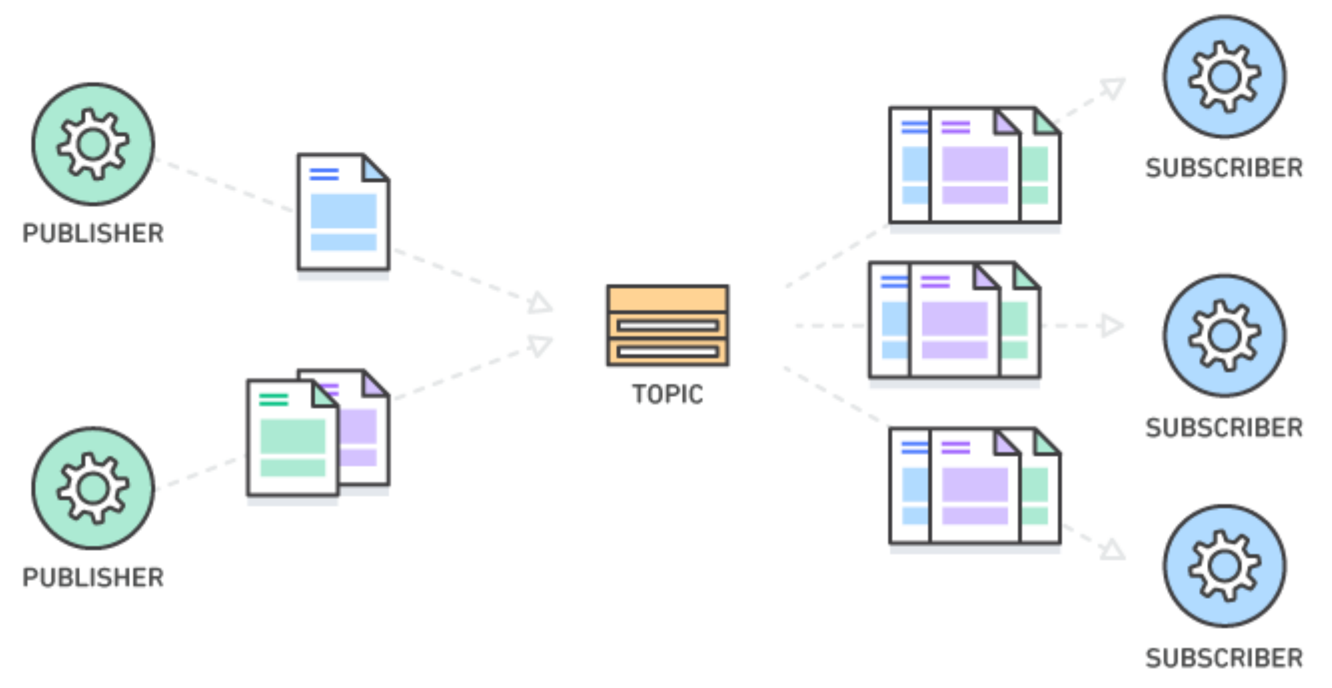
- CoAP
  - Protocol
  - Messages
  - Packets
  - Reliable Communications
  - Options
  - Observability
  - Additional Features

2023 - Marco Picone, Ph.D. (marco.picone@unimore.it)

Applications in the Internet of Things

- Protocols refer to specific communication paradigms:
  - request/response
  - publish/subscribe
- Each scenario might introduce requirements that drive the choice of suitable a communication paradigm and protocol... Which architectural style fits best in my scenario?
- While there is no final and "one-for-all" answer to this question, the IoT is being built around the REST architectural style, similarly to the Internet





REpresentational State Transfer (REST)

- REST is the architectural style behind the World Wide Web
- Defined in 2000 in Roy Fielding’s Ph.D. Dissertation “Architectural Styles and the Design of Network-based Software Architectures”*
- REST defines a set of rules and principles that all the elements of the architecture must conform to in order to build web applications that scale well, in terms of:
  - scalability (number of interacting clients)
  - robustness (long-term evolution of systems)

* R. T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” PhD thesis, University of California, 2000. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/ top.htm

Resource-oriented architectures

- REST is based on the concept of resource
- **A resource is any relevant entity in an application's domain and exposed in web** (e.g., a webpage, a video, an order in an e-commerce website, ...)
- A resource is anything with which a user interacts while progressing toward some goal
- Anything can be mapped into a resource
- Resources are characterized by some data (e.g., the title of the page, the items in an order, ...)

REpresentational State Transfer (REST)

- A REST architectures builds on:
  - **clients** (or user agents, such as browsers) are the application interface and initiate the interactions
  - **servers** (origin servers) host resources and serve client requests
  - **Intermediaries** act as clients and servers at the same time:
    - forward proxies (known to clients) are “exit points” for a request
    - reverse proxies appear as origin servers to a client, but actually relay requests
  - Uniform interfaces: all connectors within the system must conform to this interface's constraints
    - identification of resources
    - manipulation of resources through representations
    - self-descriptive messages
    - hypermedia as the engine of application state

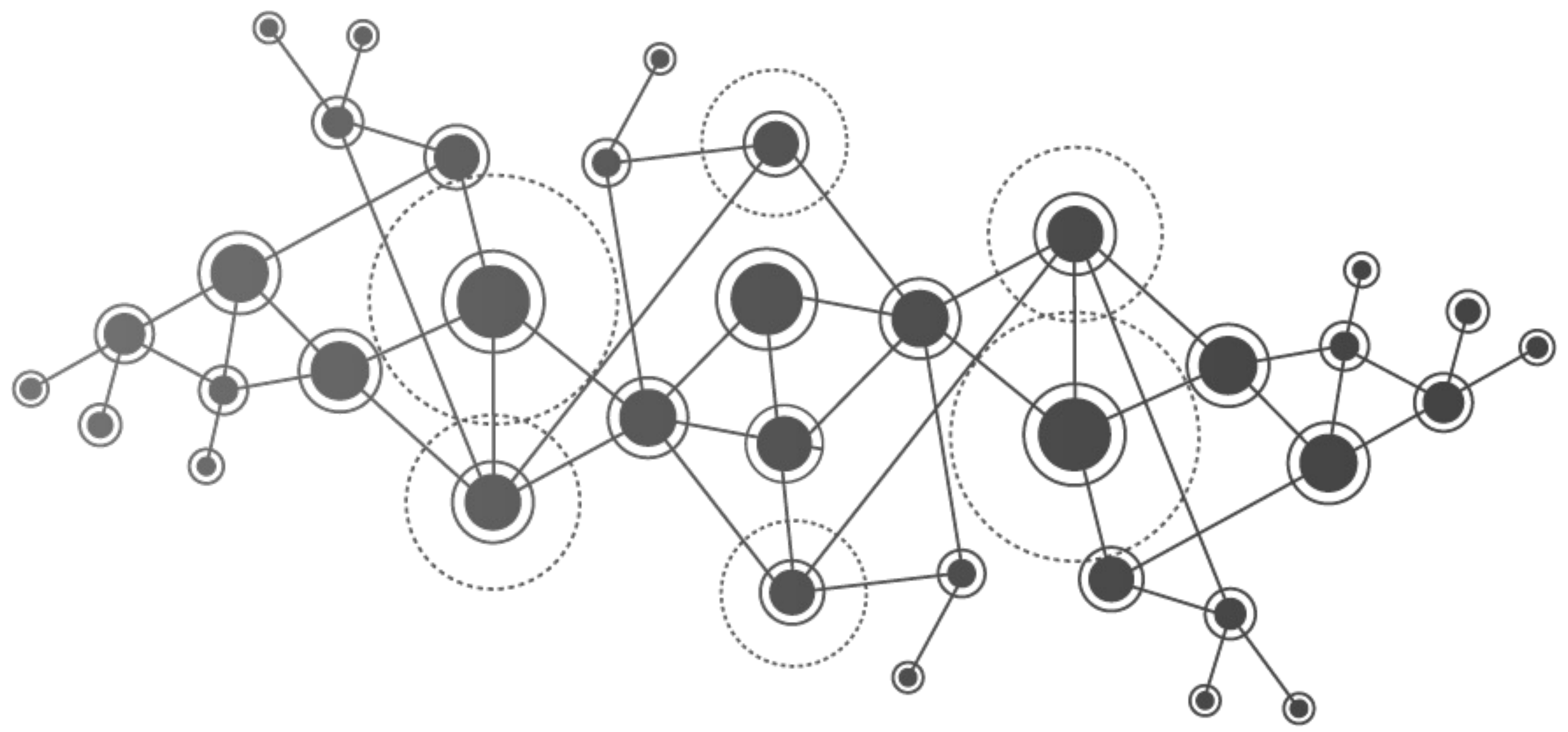
Characteristics of IoT Networks

- **Small packet size:** In constrained IoT networks the maximum physical layer packet can be quite limited according to the deployment and the communication technologies (e.g., 127 bytes)
- **Low bandwidth**
- **Low power:** according to the scenario and the deployment some or all devices are battery operated
- **Low cost:** devices are typically associated with sensors, switches, etc. This drives some of the other characteristics such as low processing, low memory, etc.
- **Unreliability:** uncertain radio connectivity, battery drain, device lockups, physical tampering, etc.
- **Duty cycling:** IoT devices may sleep for long periods of time in order to save energy, and are unable to communicate during these sleep periods

(REST) Application-Layer Protocols for the IoT

- Application-layer protocols must be designed carefully in order to take into account the constraints deriving from lower layers and the very nature of smart objects
- HTTP and TCP are note well suited
  - HTTP introduces overhead (text-based and verbose protocol)
  - TCP (connection-oriented transport)
    - setting up and maintaining connections is a heavy task
    - duty-cycling is not compatible with keeping connections alive Solution: CoAP
- Solution -> **CoAP**

An Application Layer for the IoT



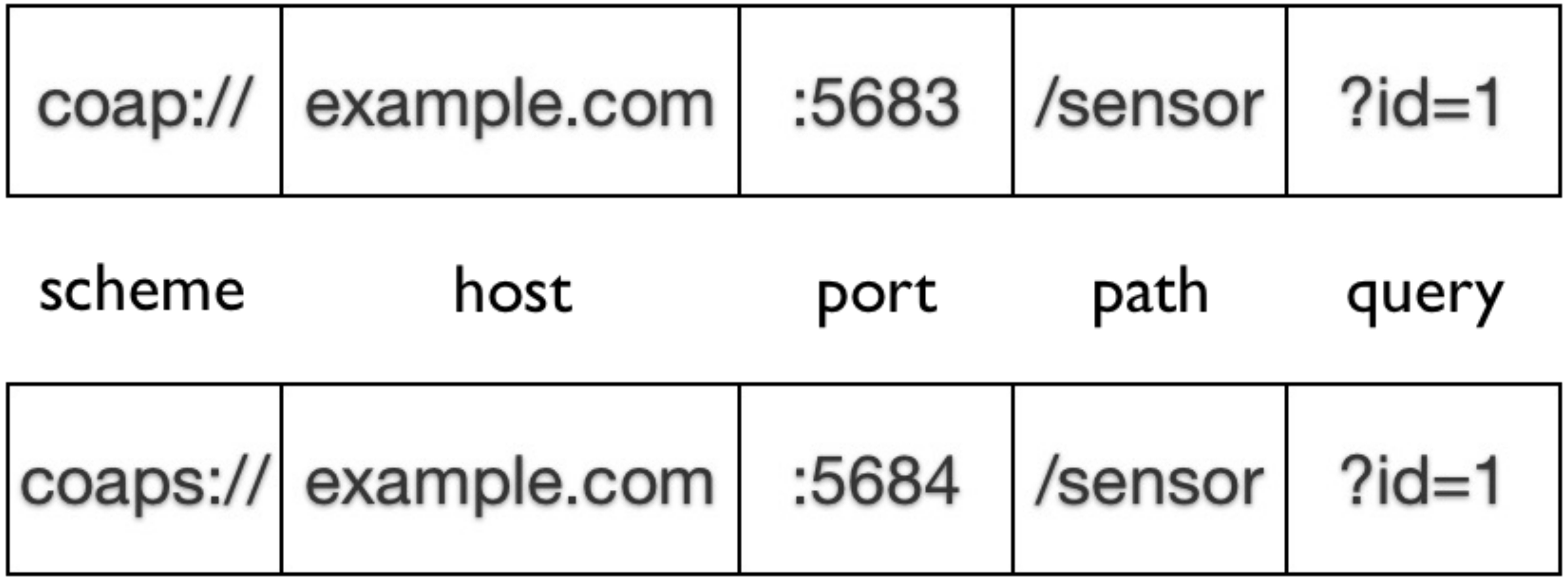
- The Web of Things (We will talk about that later …)
- Connected objects are organized in a web architecture and use web technologies to exchange information
- Integration with the World Wide Web (Extended Internet)
- The patterns of the web are well-known and can be reused effectively (REST)

Constrained Application Protocol

- Application-layer protocol designed to be used by constrained devices in terms of computational capabilities, which may feature limited battery and operate in constrained (low-power and lossy) networks
- Designed by the IETF Constrained RESTful Environments (CoRE) Working Group - RFC 7252
- **RESTful protocol**
- **Binary** protocol: small message size, easy to parse
- **Maps to HTTP easily in order to guarantee the integration with the WWW**
- CoAP runs on top of a lightweight transport, i.e. UDP
- IoT-oriented features:
  - asynchronous message exchange
  - multicast communication

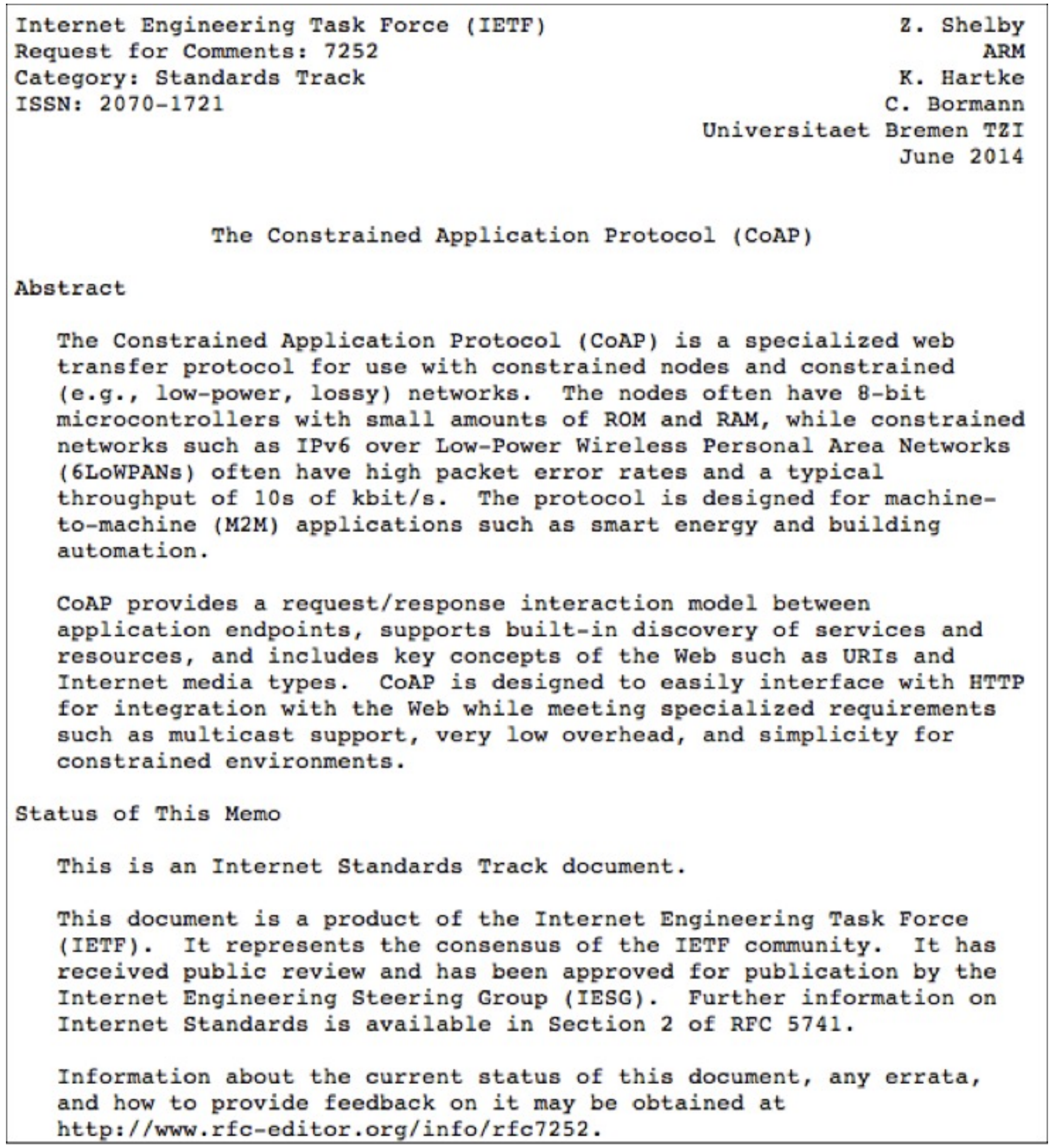
Constrained Application Protocol

- CoAP URIs use the **coap:** or **coaps:** scheme
- **coaps**: refers to secure CoAP (uses **DTLS** as a secure transport, similarly to HTTPS with TLS)
- Default ports:
  - coap: uses UDP port 5683
  - coaps: uses UDP port 5684



coap://	example.com	:5683	/sensor	?id=1
scheme	host	port	path	query
coaps://	example.com	:5684	/sensor	?id=1

Constrained Application Protocol



```
Internet Engineering Task Force (IETF)                         Z. Shelby
Request for Comments: 7252                                           ARM
Category: Standards Track                                      K. Hartke
ISSN: 2070-1721                                               C. Bormann
                                                 Universitaet Bremen TZI
                                                               June 2014


              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
   networks such as IPv6 over Low-Power Wireless Personal Area Networks
   (6LoWPANs) often have high packet error rates and a typical
   throughput of 10s of kbit/s.  The protocol is designed for machine-
   to-machine (M2M) applications such as smart energy and building
   automation.

   CoAP provides a request/response interaction model between
   application endpoints, supports built-in discovery of services and
   resources, and includes key concepts of the Web such as URIs and
   Internet media types.  CoAP is designed to easily interface with HTTP
   for integration with the Web while meeting specialized requirements
   such as multicast support, very low overhead, and simplicity for
   constrained environments.

Status of This Memo

   This is an Internet Standards Track document.

   This document is a product of the Internet Engineering Task Force
   (IETF).  It represents the consensus of the IETF community.  It has
   received public review and has been approved for publication by the
   Internet Engineering Steering Group (IESG).  Further information on
   Internet Standards is available in Section 2 of RFC 5741.

   Information about the current status of this document, any errata,
   and how to provide feedback on it may be obtained at
   http://www.rfc-editor.org/info/rfc7252.
```

https://tools.ietf.org/html/rfc7252

CoAP-related IETF CoRE WG documents

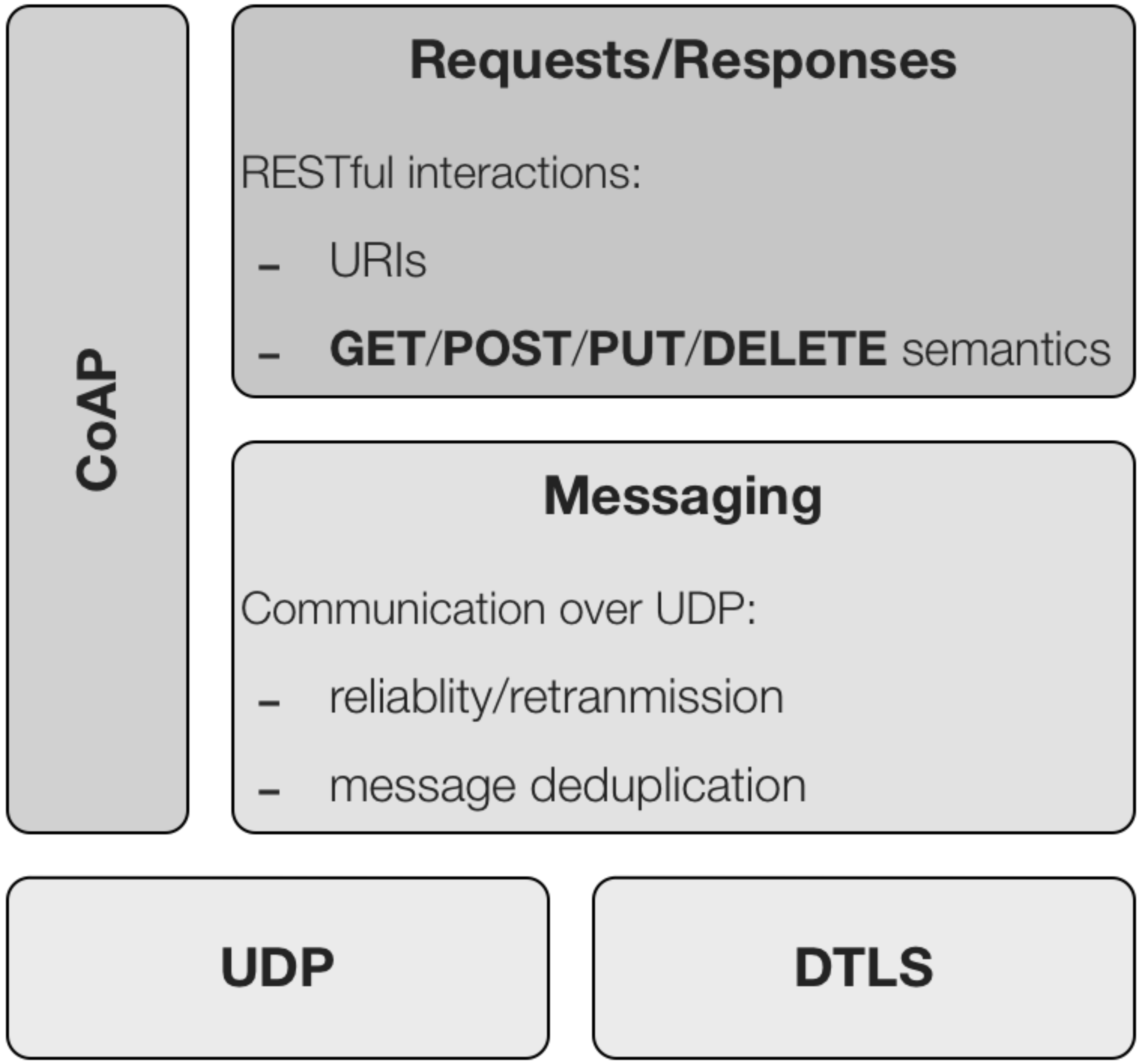
Documents	Type	no./version	Description
CoRE Link Format	RFC	6.690	“Web Linking” using a link format for use by constrained web servers to describe hosted resources
draft-ietf-core-block	RFC	7959	"Block" options for transferring multiple blocks of information from a resource representation in multiple request/response pairs
draft-ietf-core-groupcomm	RFC	7390	CoAP protocol in group communication context, on top of IP multicast
draft-ietf-core-http-mapping	RFC	8075	HTTP to CoAP protocol translation for proxy, focusing on the reverse proxy case
draft-ietf-core-interfaces	internet draft	11	REST interface descriptions for Batch, Sensor, Parameter and Actuator types for use in constrained web servers using the CoRE Link Format (RFC 6690)
draft-ietf-core-observe	RFC	7641	"Observe" option to enable a subscribe/notify paradigm in CoAP
draft-ietf-core-resource-directory	internet draft	13	Resource Directory (RD), which hosts descriptions of resources held on other servers, allowing lookups to be performed for those resources

CoAP vs. HTTP protocol stack

Application	CoAP	HTTP
Transport	UDP	TCP
Network	IPv6/6LoWPAN	IP
Link	MAC	MAC
Physical	PHY	PHY
Layers	*Internet of Things*	*Internet*

Constrained Application Protocol

- CoAP is based on a request/response communication model
- CoAP is based on UDP: reliability is not provided by the transport layer (retransmissions may be needed)
- Asynchronous communications may require matching between requests and responses
- CoAP is characterized by two layers
  - **Messaging layer:**
    - support for duplicate detection
    - (optional) reliability
  - **Requests/Responses layer:**
    - RESTful interactions
    - GET, POST, PUT, DELETE methods



CoAP Messaging

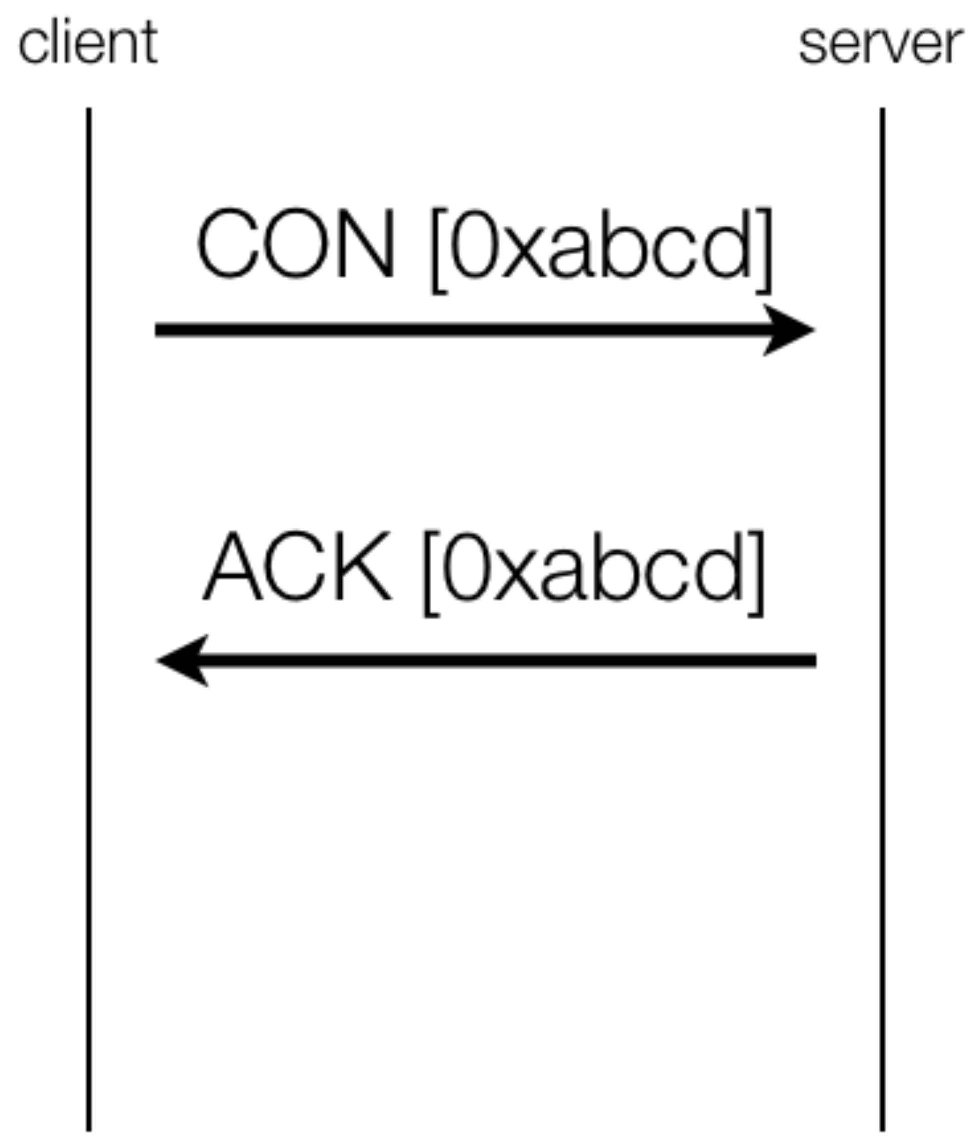
- CoAP messages are exchanged between UDP endpoints (default port is 5683)
- **CoAP messages, as in HTTP, are either requests or responses**
- Messages include a **Message ID** (16 bits) used to detect duplicates and for optional reliability (acknowledgements of received messages)
- Requests and responses are matched using a **Token** (0 to 8 bytes)
- **Message IDs** and **Tokens** are separate concepts and they actually work "orthogonally"

CoAP Messaging Reliability

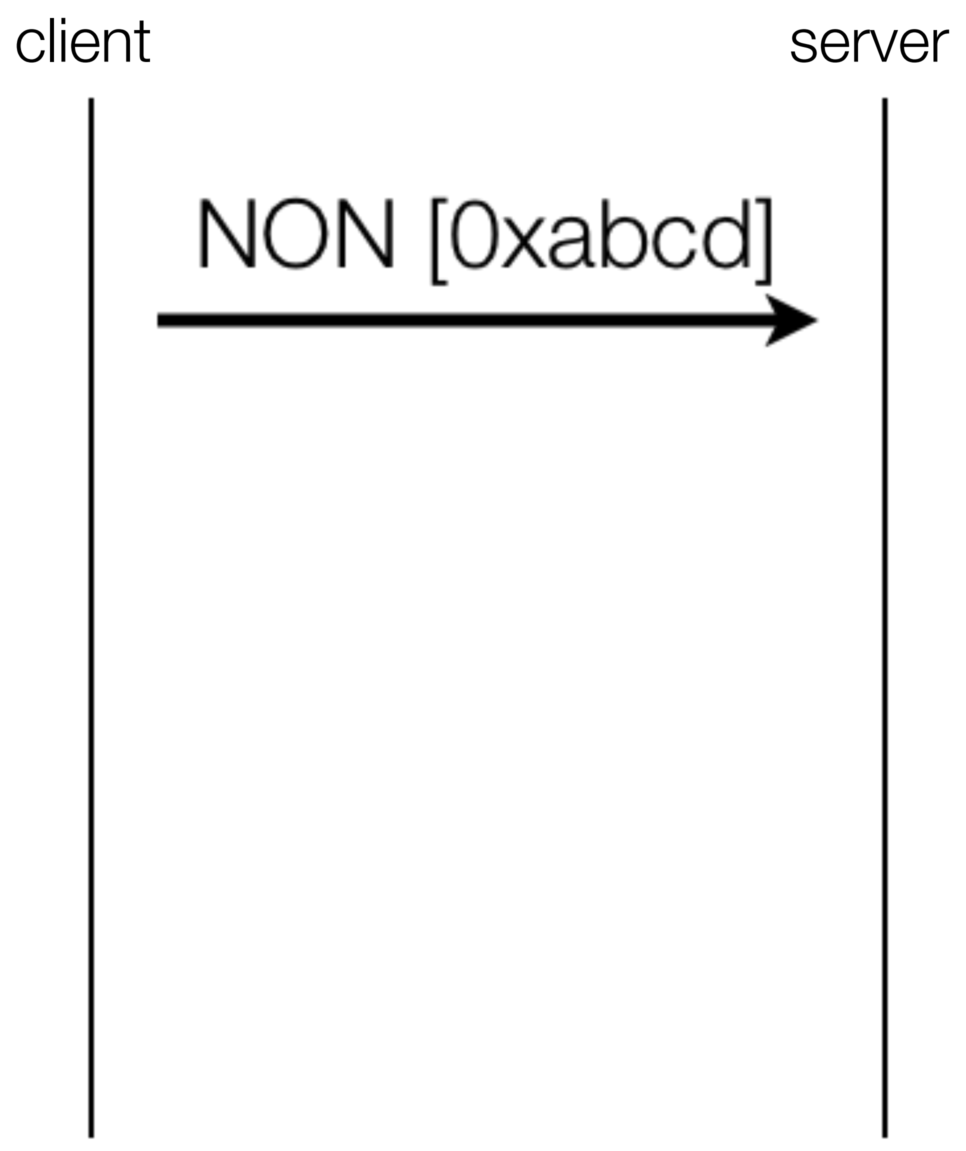
- The use of UDP, especially in LLNs, might lead to messages being lost
- **Reliability cannot be provided by transport layer (as if TCP was used) and must be ensured at application layer (CoAP over TCP is work in progress)**
- **Reliability is implemented by performing retransmissions with exponential back-off (timeout is doubled at each retransmission)**
- A maximum of 4 retransmissions can be performed
- There 4 kinds of messages:
    - **CON** (confirmable): used for messages that must be transmitted reliably
    - **NON** (non-confirmable): used for messages for which reliability is not needed
    - **ACK** (acknowledgment): used to acknowledge the reception of a CON message
    - **RST** (reset): used to cancel a reliable transmission

CoAP Messaging Reliability

- When a CON request is sent, an ACK response is required to be sent back to acknowledge the correct reception of the request
- When a NON request is sent, the response should be returned in a NON message
- CON messages provide reliability: if no ACK response is returned prior to a retransmission timeout (with exponential back-off), the message is retransmitted

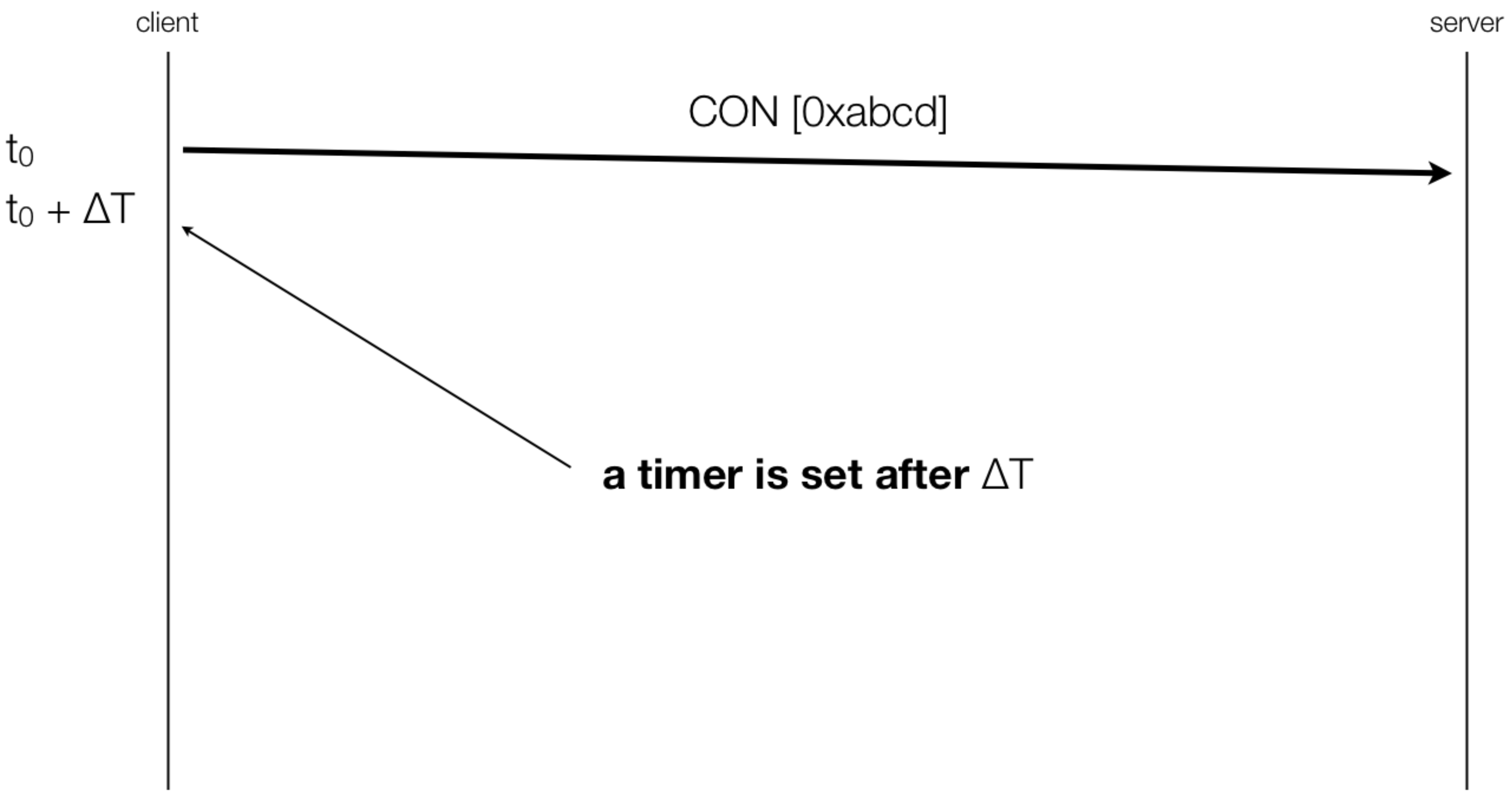


reliable transmission

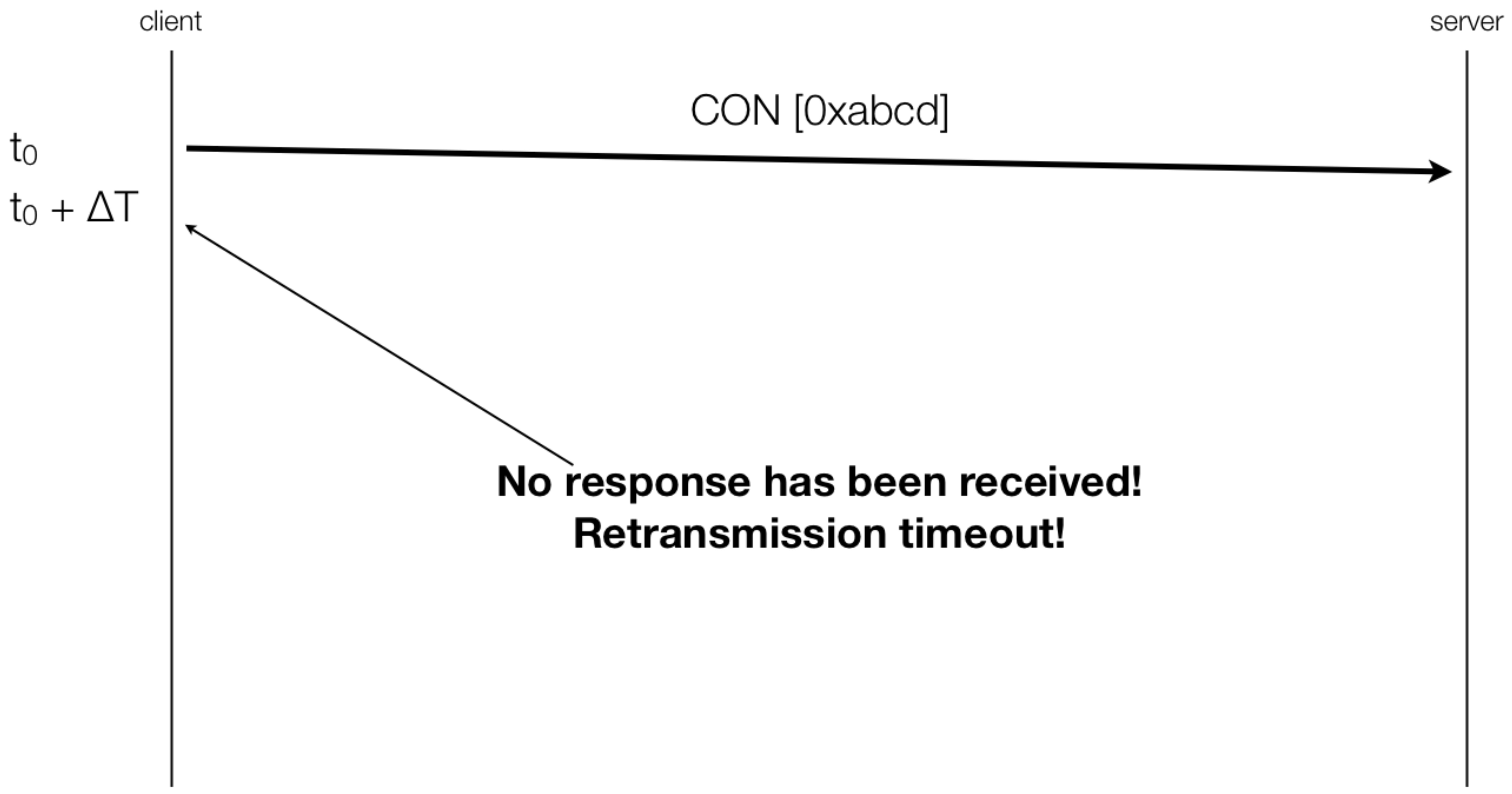


unreliable transmission

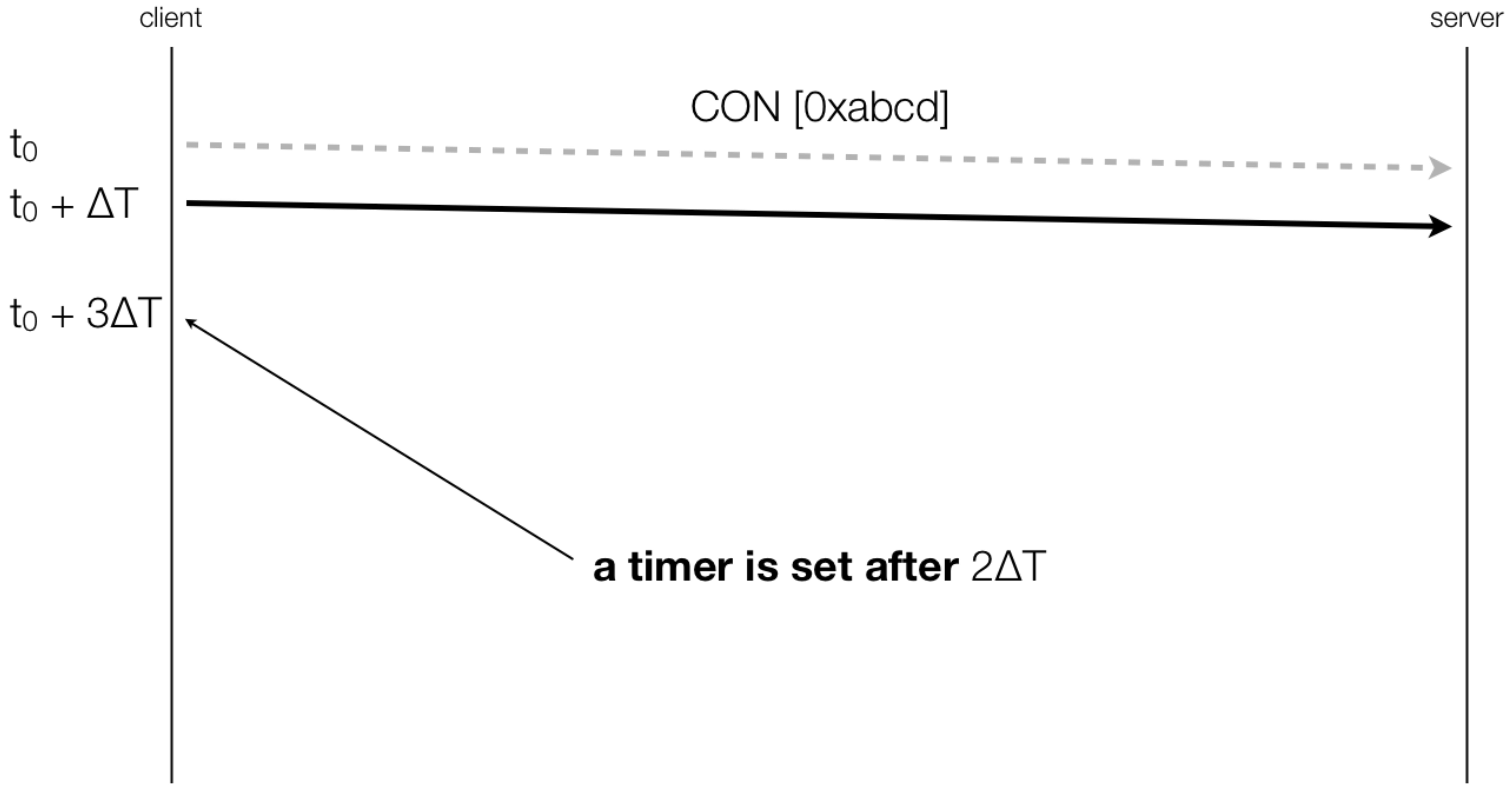
CoAP reliable transmission



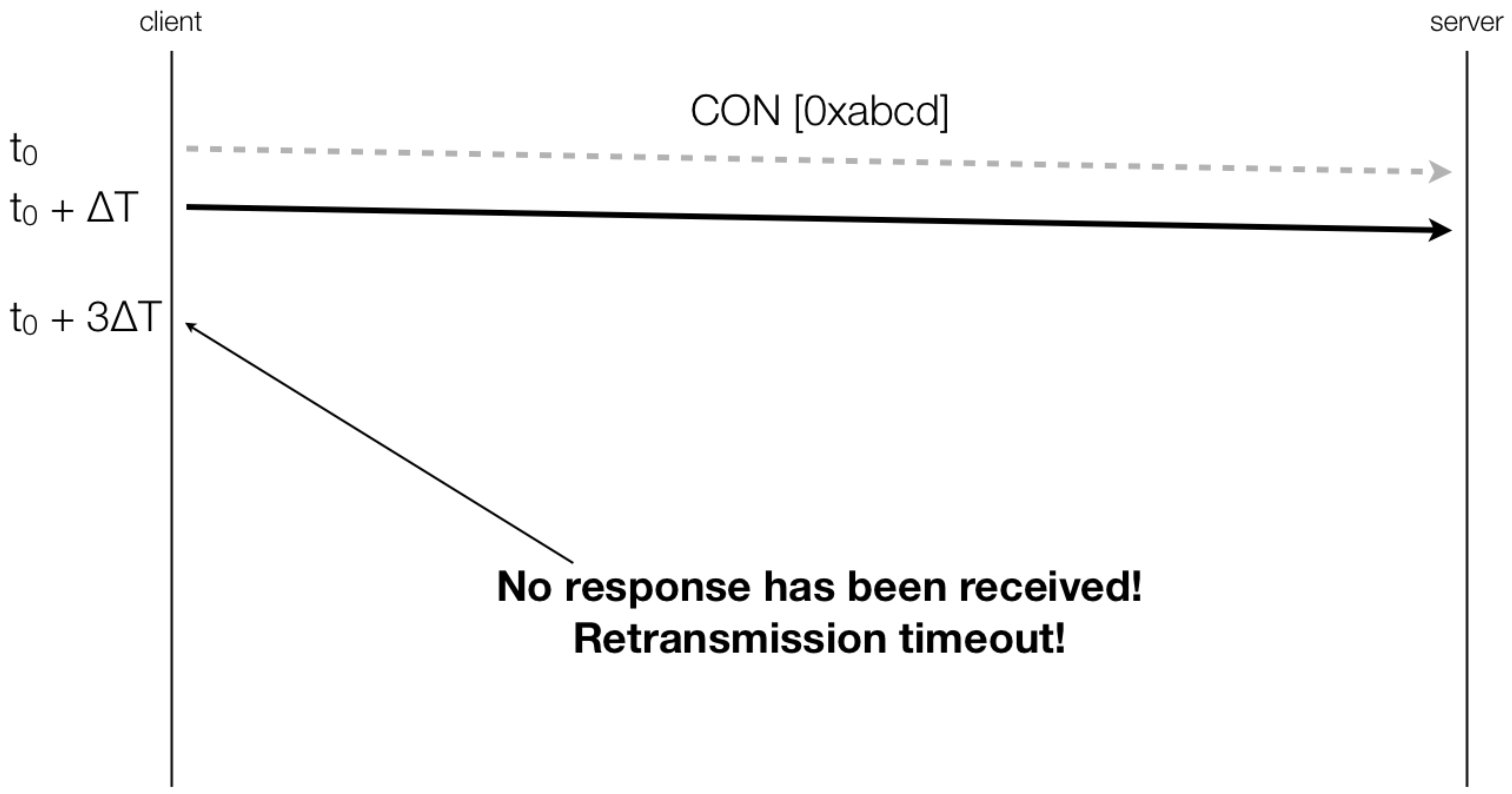
CoAP reliable transmission



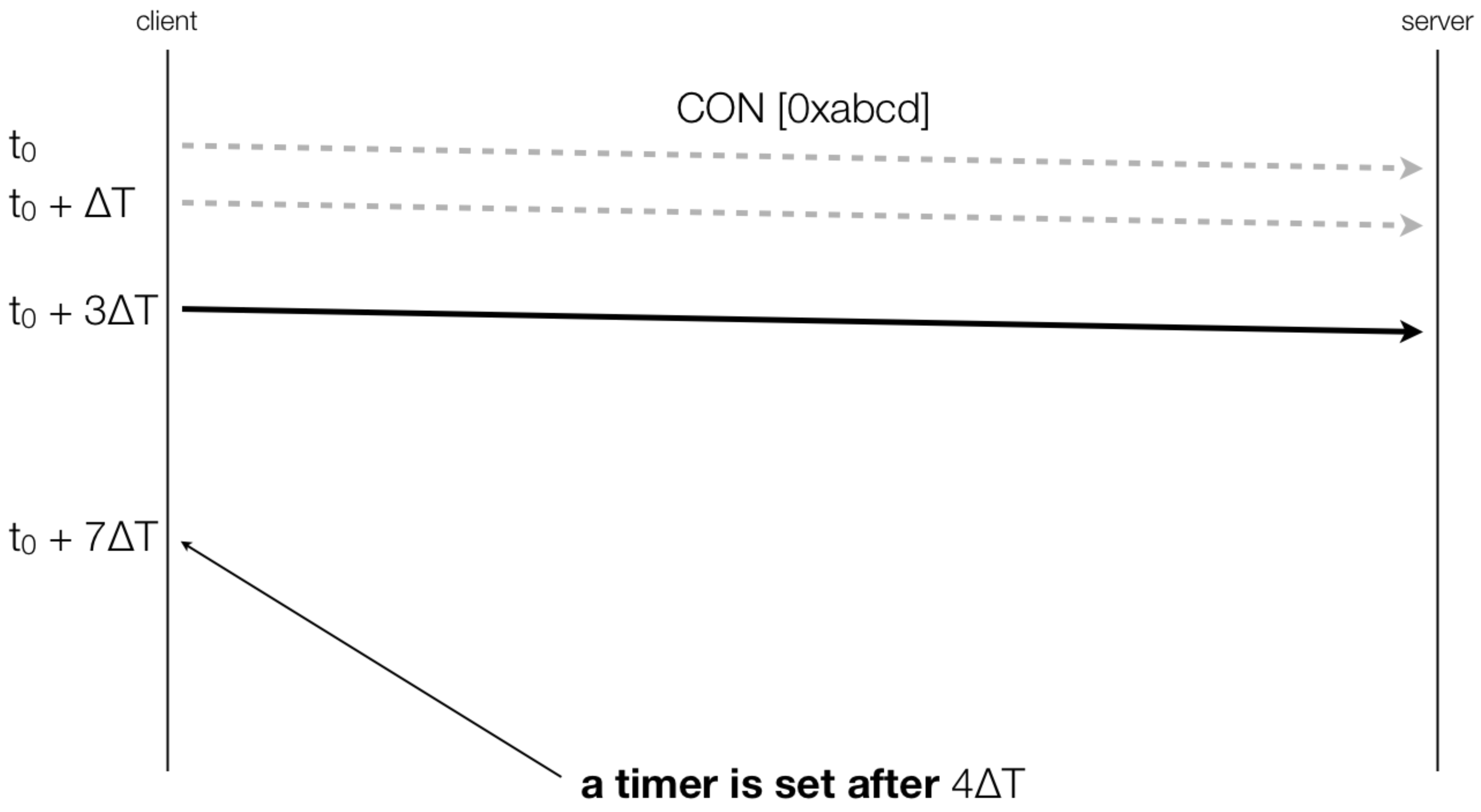
CoAP reliable transmission



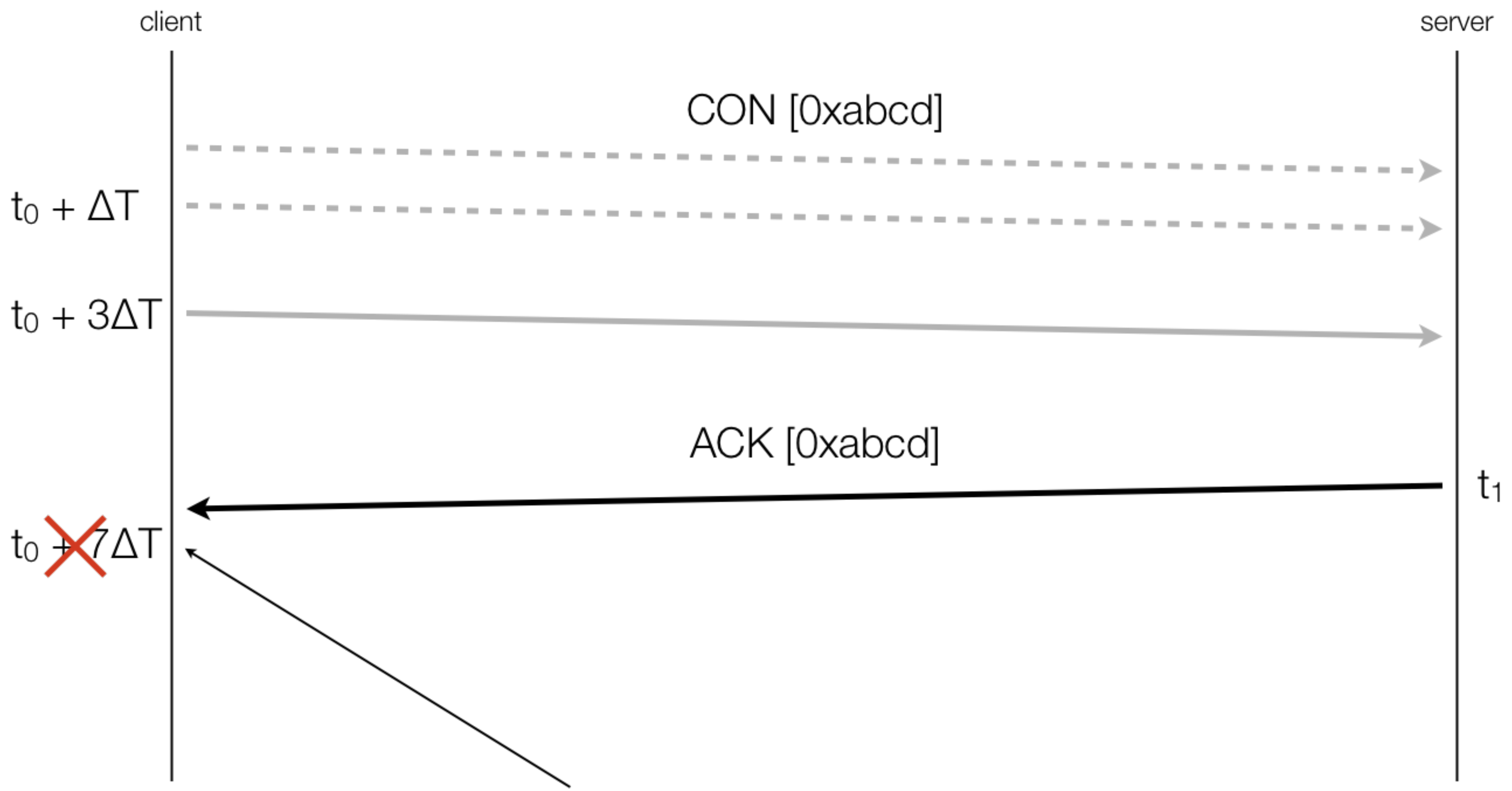
CoAP reliable transmission



CoAP reliable transmission



CoAP reliable transmission

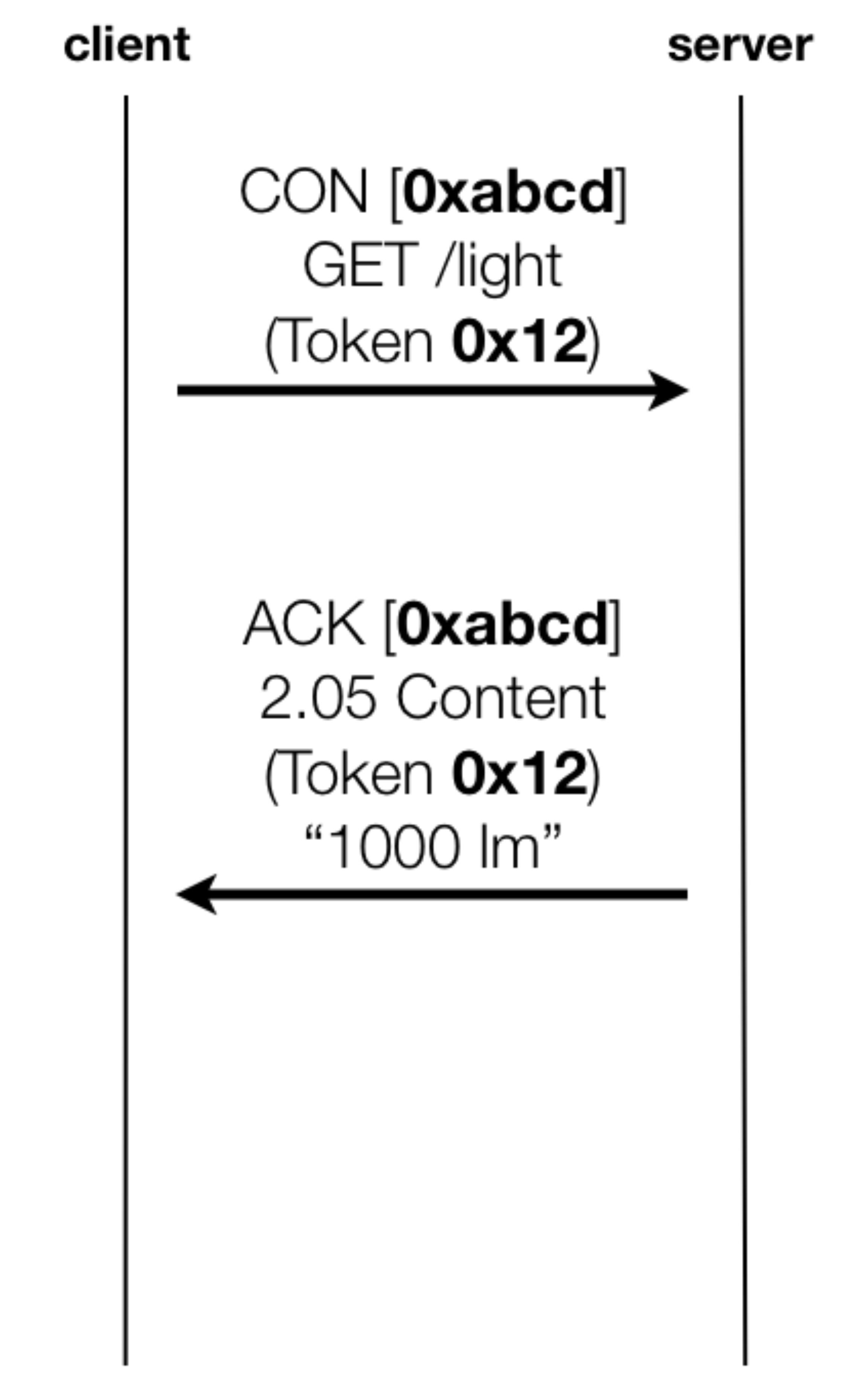


An ACK is received before timer expires
Timer is invalidated

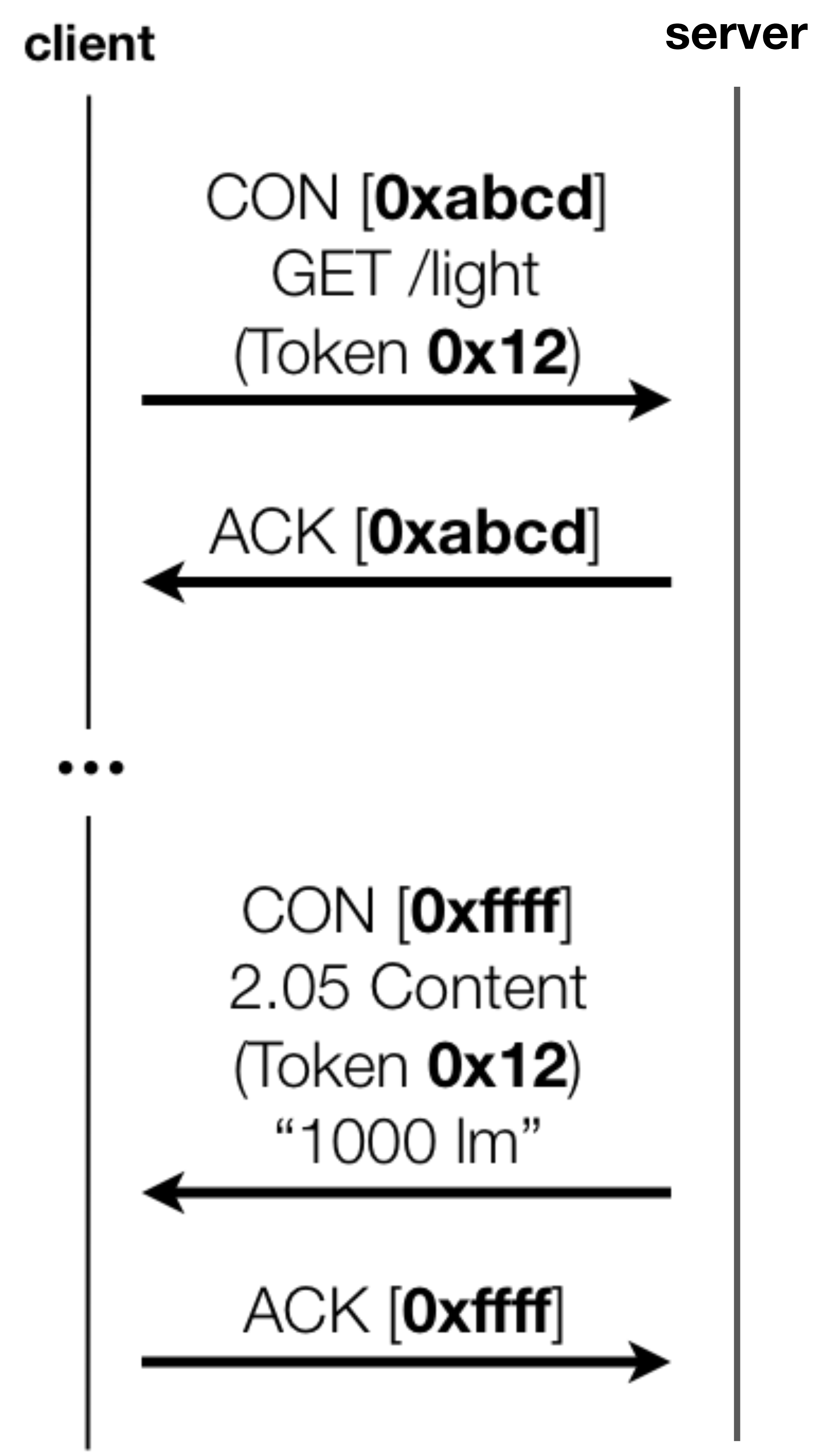
Piggy-backed and separate responses

- If a CON CoAP request is sent, CoAP responses can be either:
    - **piggy-backed**, if the server can respond immediately
    - **separate,** if the server cannot respond immediately
- With **piggy-backed responses**, the **responses are included in the ACK** message that acknowledges the request (implicit acknowledgement)
- With separate responses, **the server first sends an ACK message** to acknowledge the request; when the data is available, **the server sends a CON message containing the response, which is acknowledged by the client with a new ACK message**

Piggy-backed and separate responses

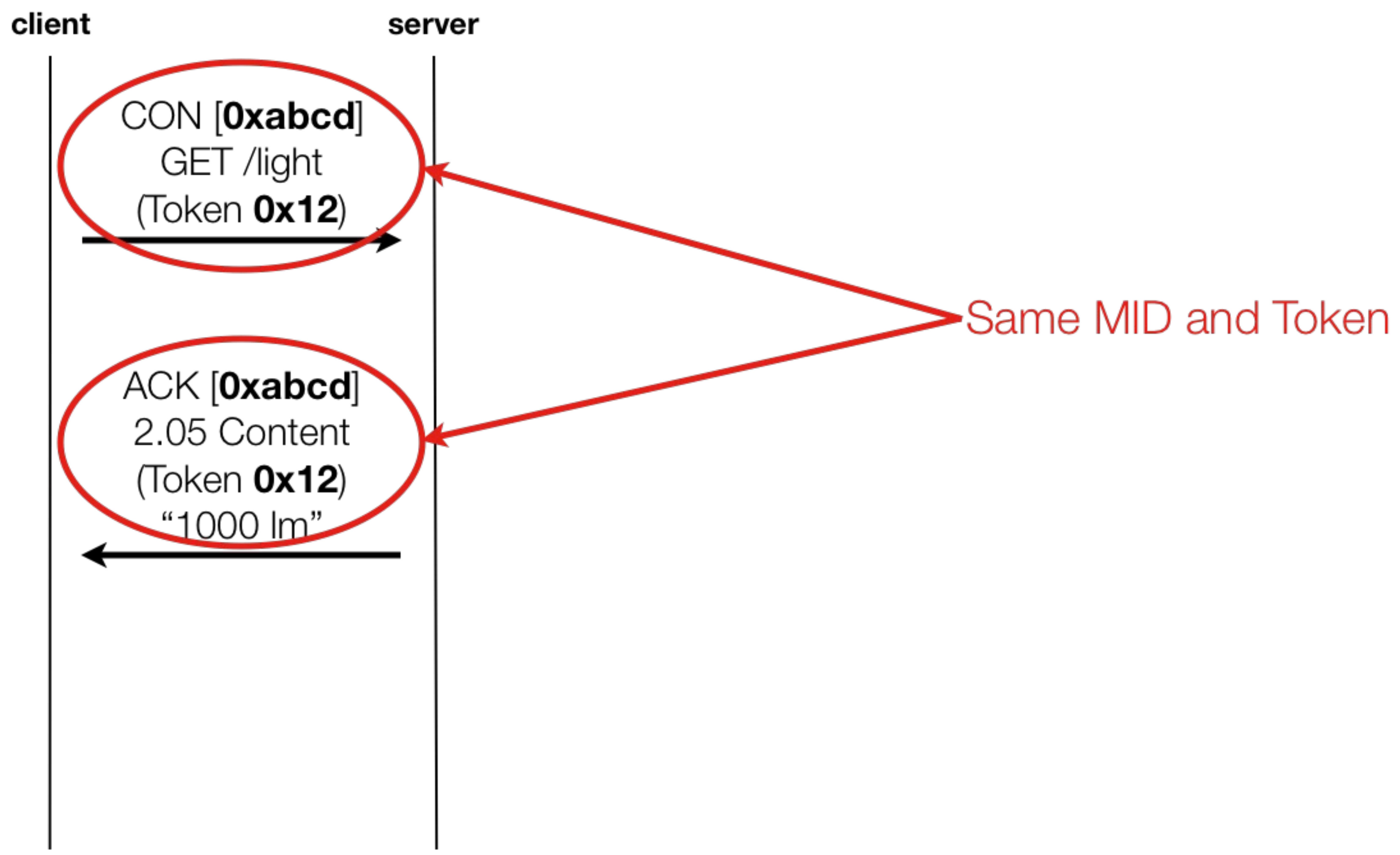


Piggy-backed response

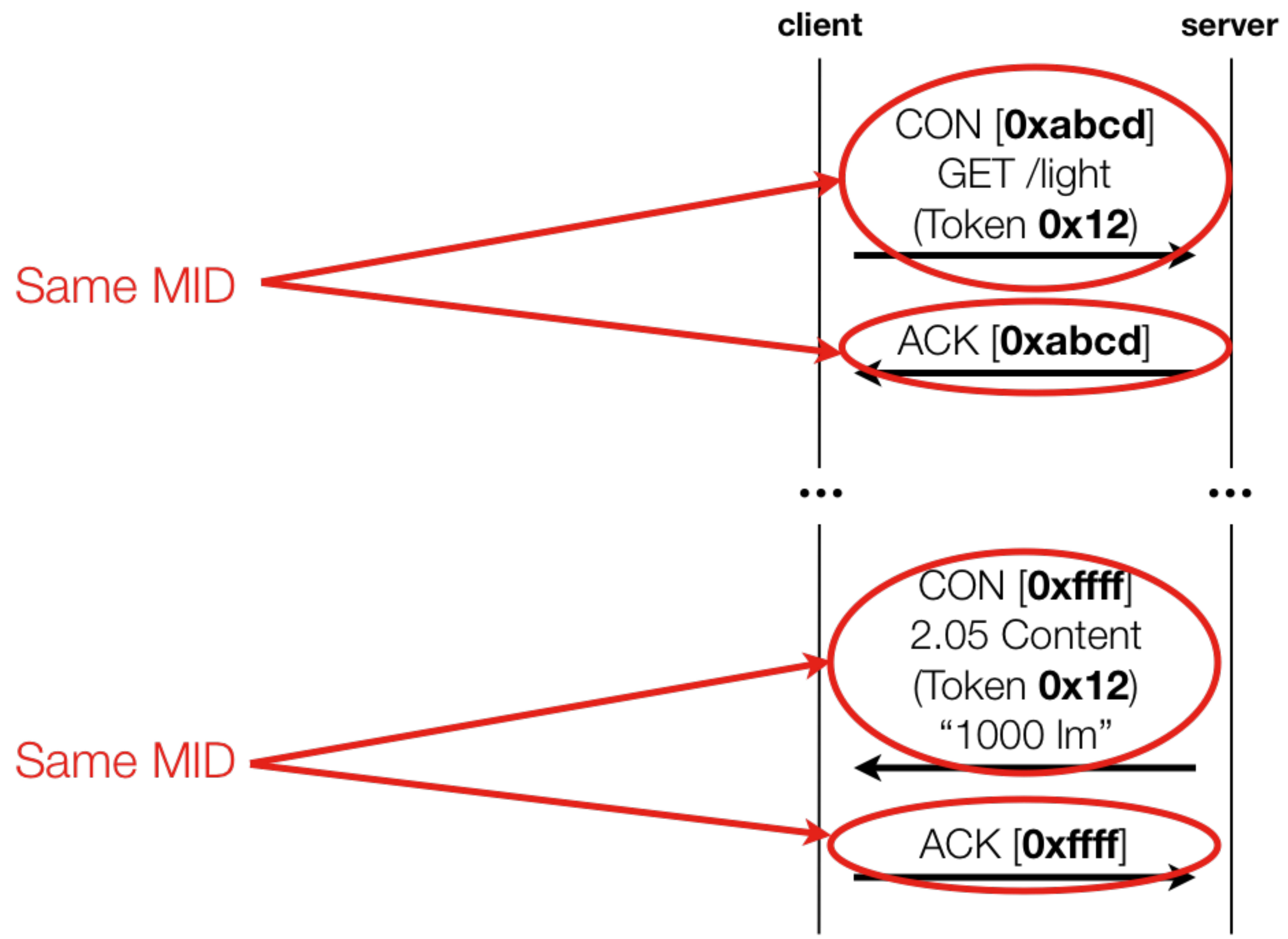


Separate response

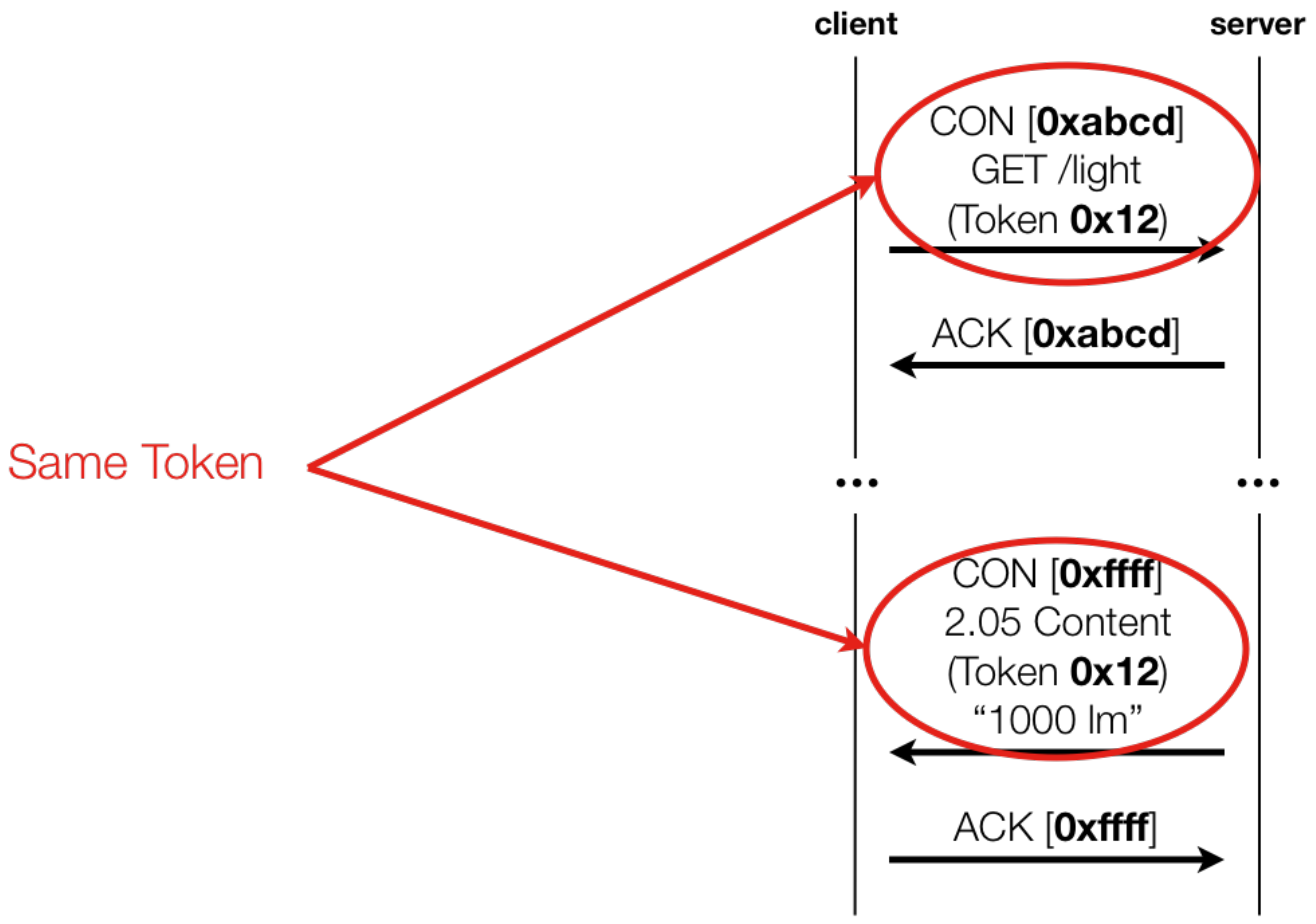
Piggy-backed responses



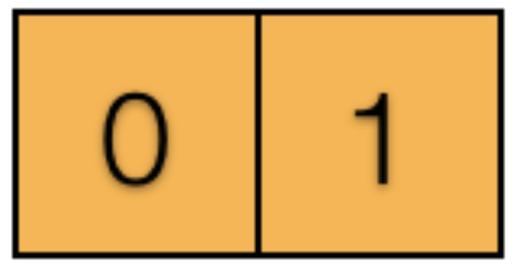
Separate responses



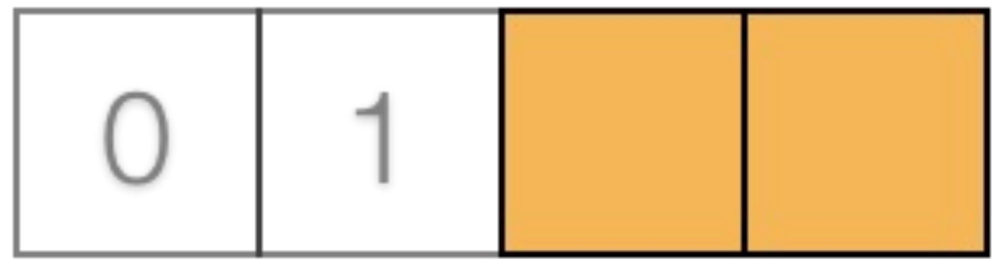
Separate responses



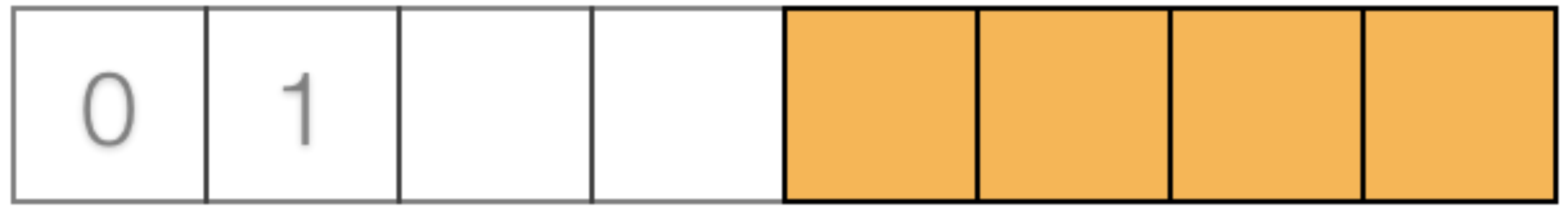
CoAP Message Format



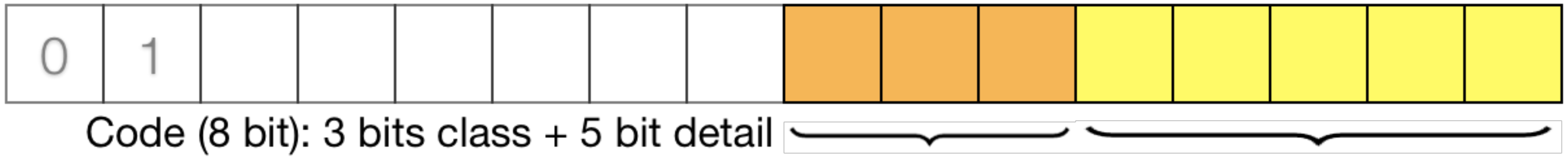
Ver (2 bit): Version (= 01)



T (2 bit): Type (0 = CON, 1 = NON, 2 = ACK, 3 = RST)



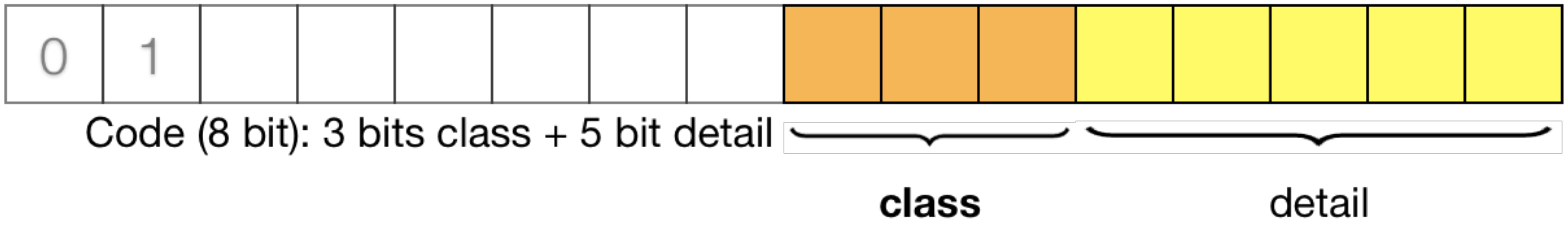
TKL (4 bit): Token length (0-8); lengths 9-15 are reserved



Code (8 bit): 3 bits class + 5 bit detail

class detail

CoAP Message Format

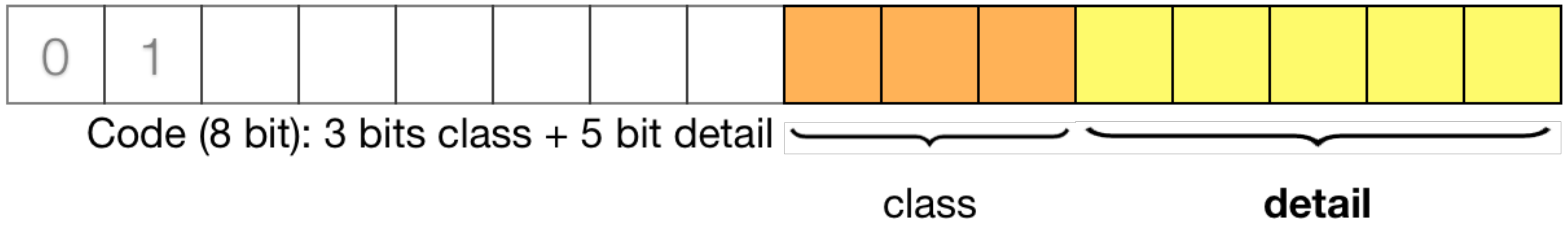


Class	Bits	Meaning
0	000	Request
2	010	Success
4	100	Client error
5	101	Server error

$\left.\begin{matrix}\\ \\ \\ \end{matrix}\right\}$ Response

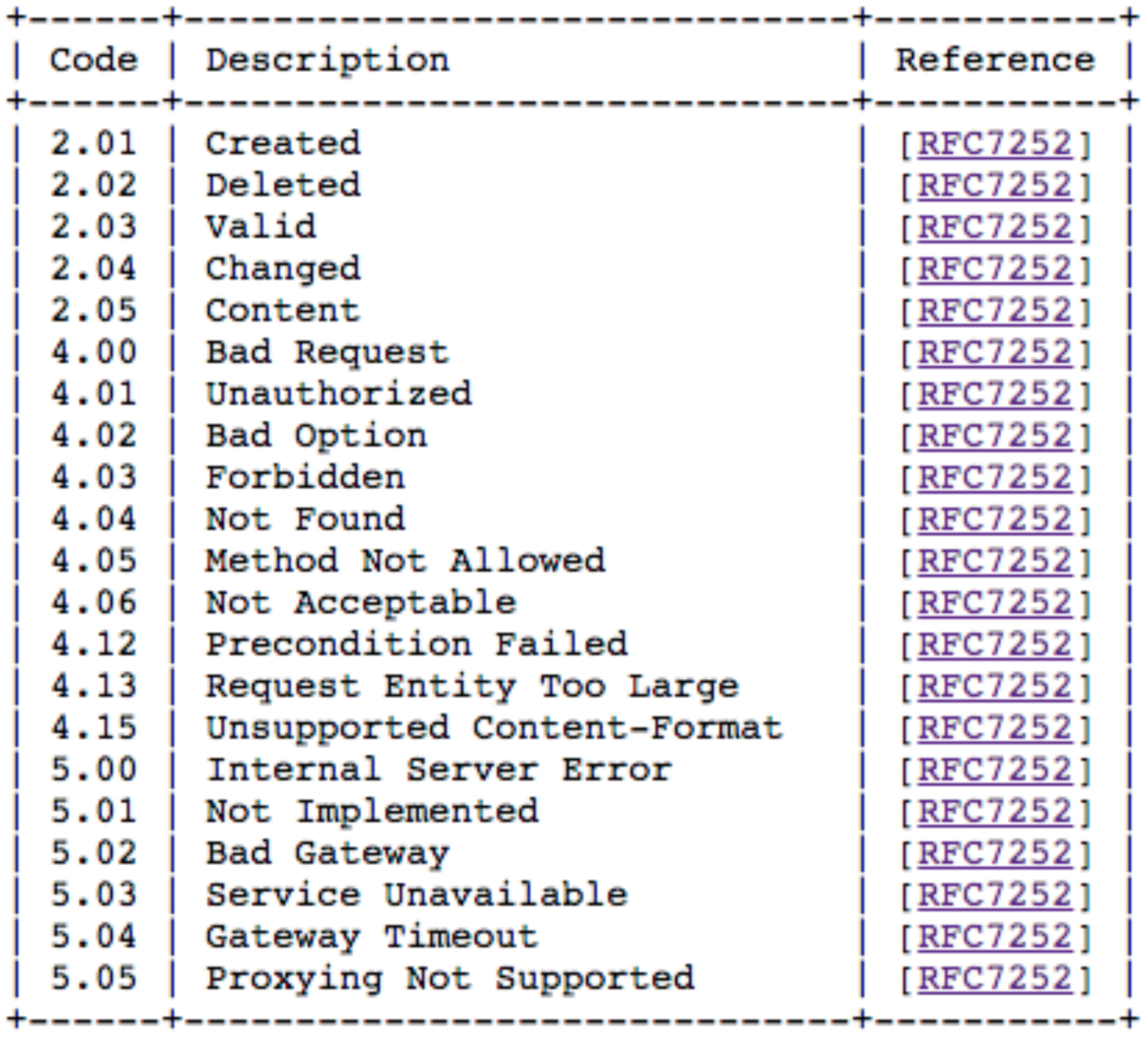
- If all bits are set to 0, it is a request
- The first bit indicates whether this is a success or error response
- The last bit indicates if the error is due to the client or to the server

CoAP Message Format



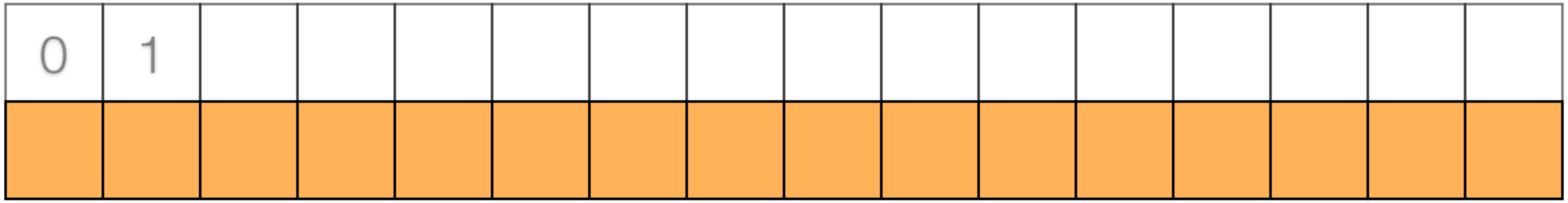
- In case of requests, the detail is used to indicate the type of request
  - 1 (GET), 2 (POST), 3 (PUT), 4 (DELETE)
  - special case: 0 indicates an empty message
- In case of responses, the detail is used to give additional information related to the response:
  - 2.01 Created, 2.02 Deleted, 2.04 Changed, 2.05 Content
  - 4.00 Bad Request, 4.02 Bad Option, 4.04 Not Found, 4.05 Method Not Allowed, ...
  - 5.00 Internal Server Error, 5.01 Not Implemented, ...

CoAP Message Format



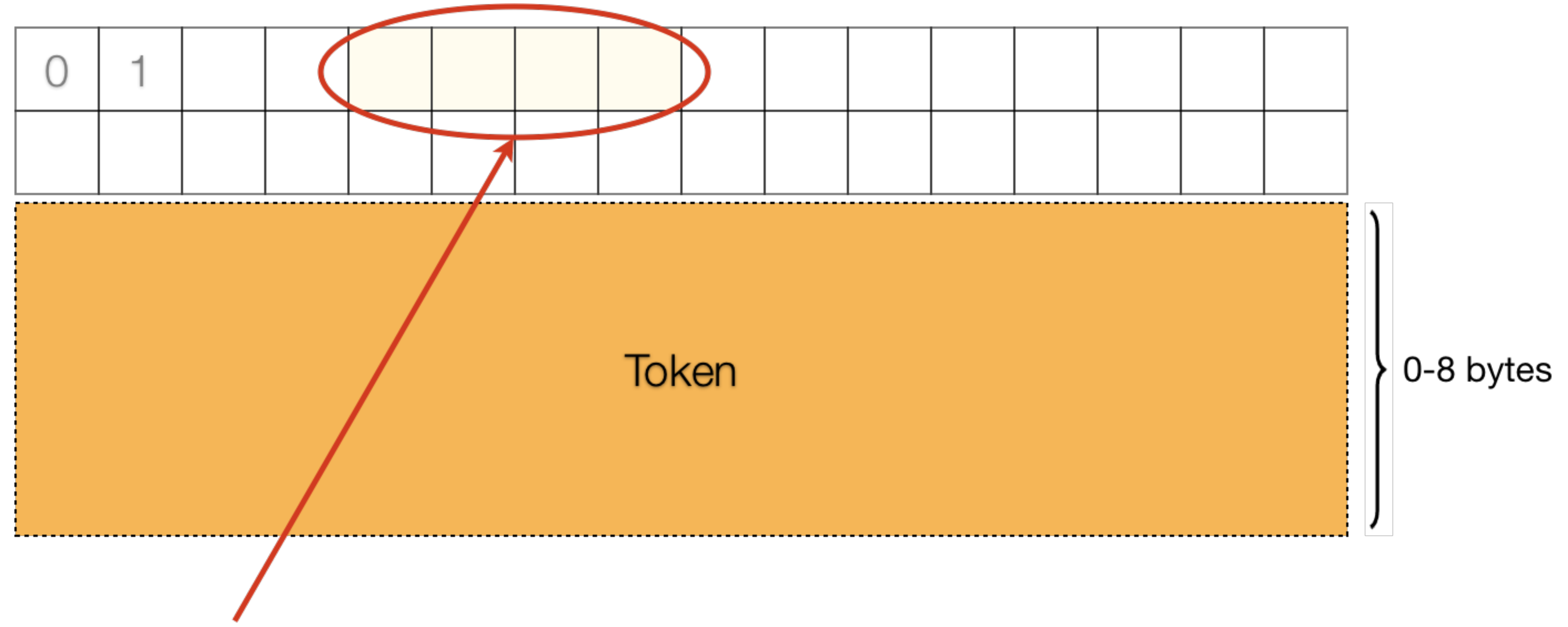
Code	Description	Reference
2.01	Created	[RFC7252]
2.02	Deleted	[RFC7252]
2.03	Valid	[RFC7252]
2.04	Changed	[RFC7252]
2.05	Content	[RFC7252]
4.00	Bad Request	[RFC7252]
4.01	Unauthorized	[RFC7252]
4.02	Bad Option	[RFC7252]
4.03	Forbidden	[RFC7252]
4.04	Not Found	[RFC7252]
4.05	Method Not Allowed	[RFC7252]
4.06	Not Acceptable	[RFC7252]
4.12	Precondition Failed	[RFC7252]
4.13	Request Entity Too Large	[RFC7252]
4.15	Unsupported Content-Format	[RFC7252]
5.00	Internal Server Error	[RFC7252]
5.01	Not Implemented	[RFC7252]
5.02	Bad Gateway	[RFC7252]
5.03	Service Unavailable	[RFC7252]
5.04	Gateway Timeout	[RFC7252]
5.05	Proxying Not Supported	[RFC7252]

CoAP Message Format



Message-ID (16 bit): it is used to detect duplicates and for optional reliability

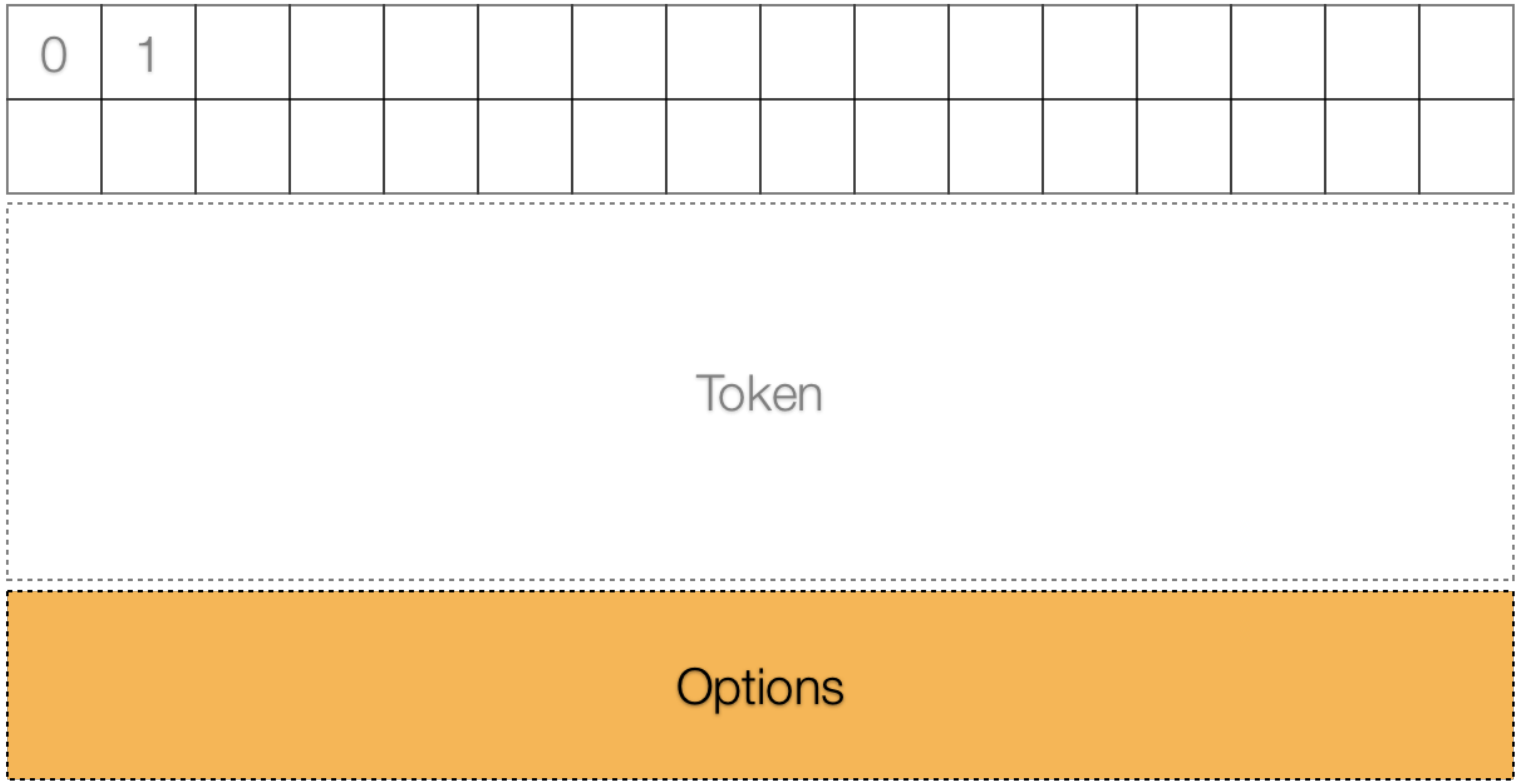
CoAP Message Format



Token (TKL bytes): it is used to match responses to requests independently from the underlying messages

This field is *optional* (if TKL = 0)

CoAP Message Format

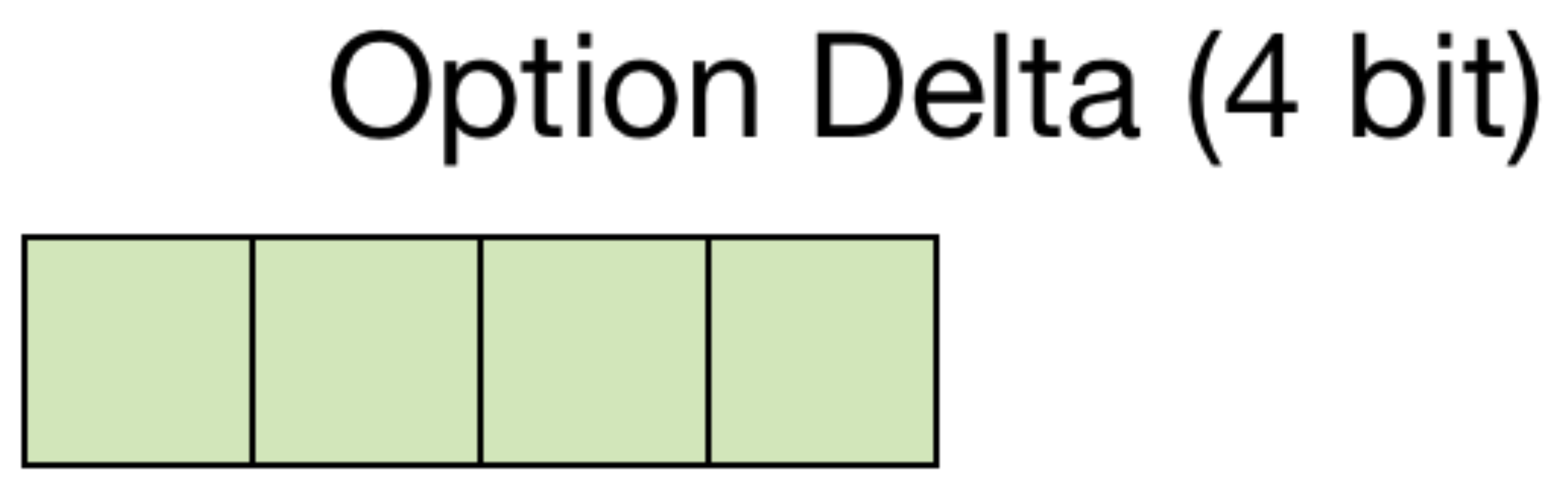


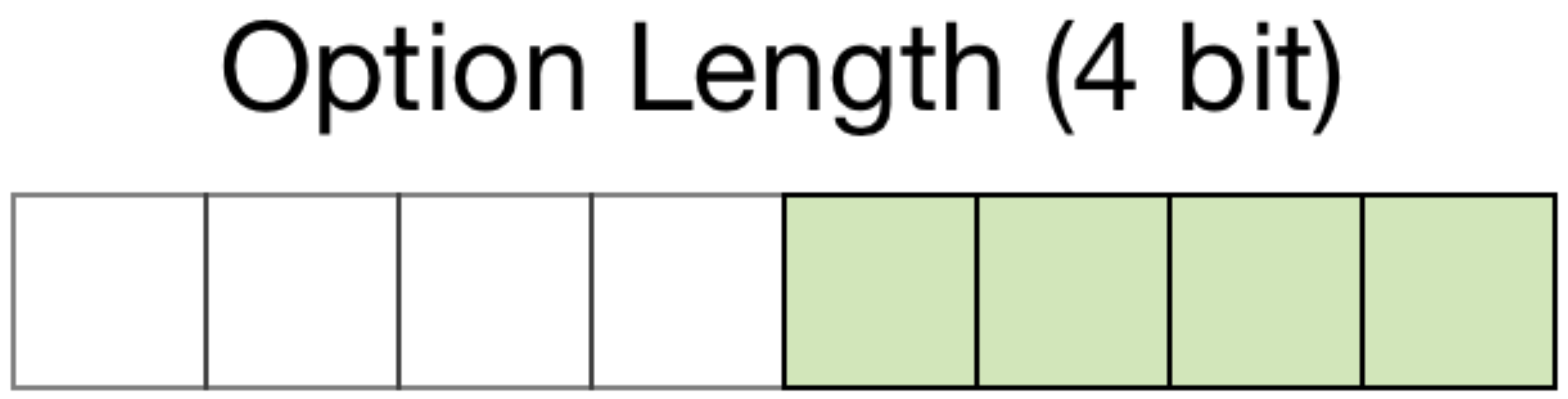
CoAP Options

- CoAP request and response semantics are carried in CoAP messages
- **Optional (or default) request and response information, such as the URI and payload media type are carried as CoAP options (similar to HTTP headers)**
- CoAP defines a number of options which can be included in a message
- Each option instance in a message specifies the **Option Number** of the defined CoAP option, the **length of the Option Value** and the **Option Value itself**
- In order to maximize compactness, options are encoded in a very efficient way, called **delta encoding**
    - CoAP Options are identified by numbers, registered on IANA registry
    - Options are kept in ascending order
    - Option number is calculated as a delta from the preceding option

CoAP Options

- The **Option Number** is calculated by simply summing the **Option Delta** values of this and all previous options before it
- Possible values:
  - **0** to **12** (no extended option)
  - **13** (extended option): value of the option ranges from 13 to 268
    - 1 byte for extended delta after option length
  - **14** (extended option): value of the option ranges from 269 to 65804
    - 2 bytes for extended delta after option length

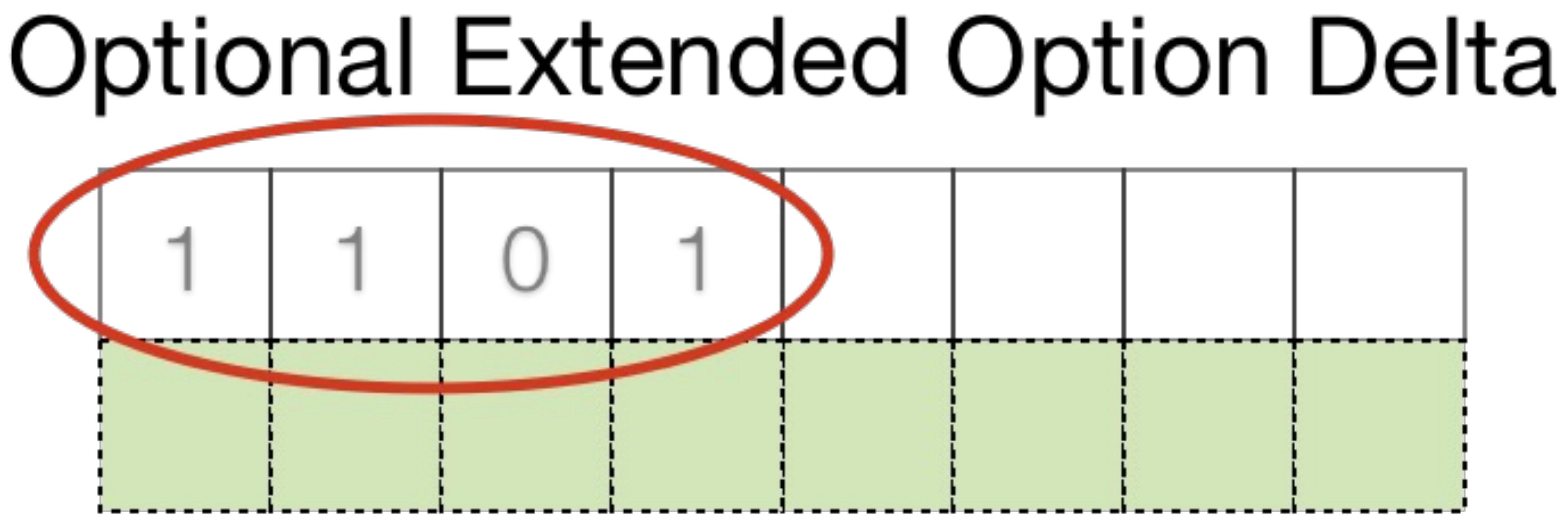




CoAP Options

If option delta is 13, 1 byte for extended delta

Option number is calculated as 13 + value of extended delta

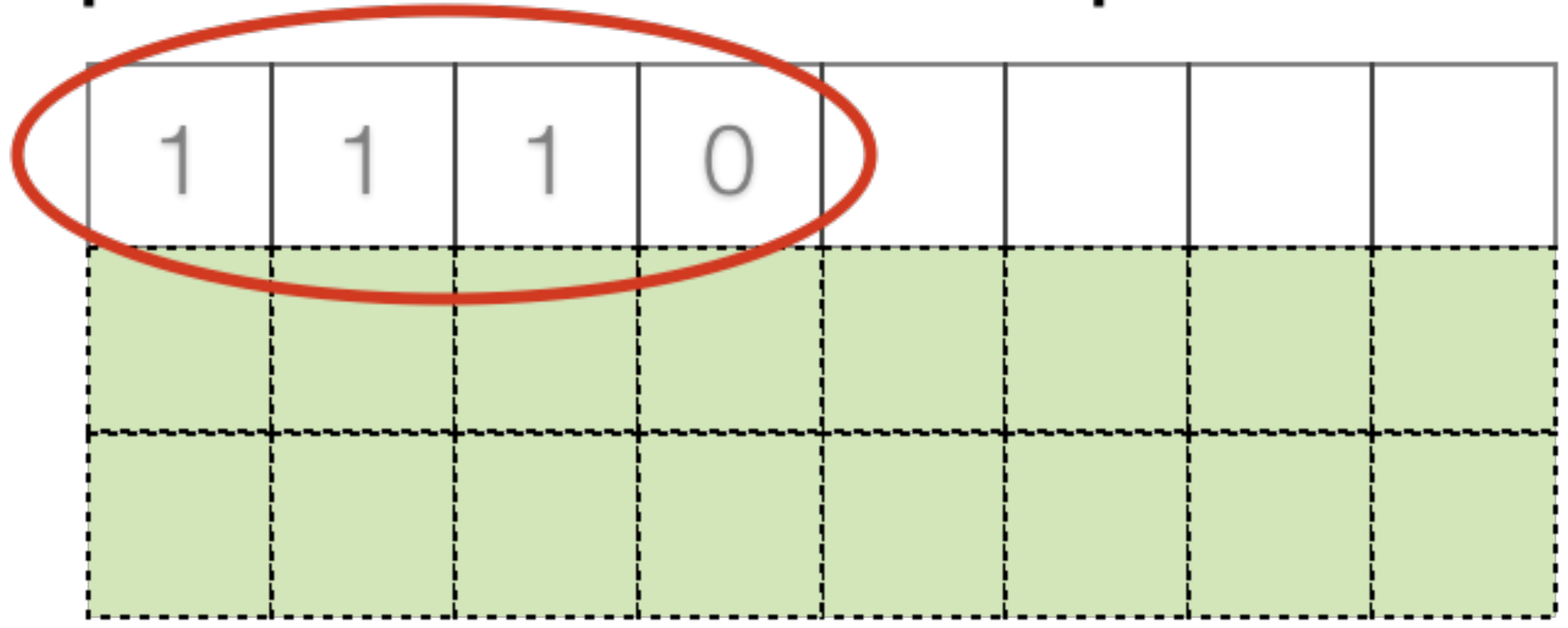


CoAP Options

If option delta is 14, 2 bytes for extended delta

Option number is calculated as 269 + value of extended delta

Optional Extended Option Delta



CoAP Options

- A sequence of exactly Option Length bytes
- The length and format of the Option Value depend on the respective option, which MAY define variable-length values
- Option values can be:
  - **Empty:** A zero-length sequence of bytes
  - **Opaque:** An opaque sequence of bytes
  - **Uint:** A non-negative integer that is represented in network byte order using the number of bytes given by the Option Length field.
  - **String:** A Unicode string that is encoded using UTF-8 [RFC3629] in Net-Unicode form [RFC5198]

CoAP Options

Option number	Option name	Format	Length (bits)
1	If-Match	opaque	0-8
3	Uri-Host	string	1-255
4	ETag	opaque	1-8
5	If-None-Match	empty	0
7	Uri-Port	uint	0-2
8	Location-Path	string	0-255
11	Uri-Path	string	0-255
12	Content-Format	uint	0-2
14	Max-Age	uint	0-4
15	Uri-Query	string	0-255
17	Accept	uint	0-2
20	Location-Query	string	0-255
35	Proxy-Uri	string	1-1034
39	Proxy-Scheme	string	1-255
60	Size1	uint	0-4

CoAP Options

Option number	Option name	Format	Length (bits)
1	If-Match	opaque	0-8
3	Uri-Host	string	1-255
4	ETag	opaque	1-8
5	If-None-Match	empty	0
7	Uri-Port	uint	0-2
8	Location-Path	string	0-255
11	Uri-Path	string	0-255
12	Content-Format	uint	0-2
14	Max-Age	uint	0-4
15	Uri-Query	string	0-255
17	Accept	uint	0-2
20	Location-Query	string	0-255
35	Proxy-Uri	string	1-1034
39	Proxy-Scheme	string	1-255
60	Size1	uint	0-4

Uri-Host, Uri-Port, Uri-Path, and Uri-Query

- The Uri-Host, Uri-Port, Uri-Path, and Uri-Query options are used to specify the target resource of a request to a CoAP origin server
- Uri-Path and Uri-Query options are repeatable
- The Uri-Host Option specifies the Internet host of the resource being requested
- The Uri-Port Option specifies the transport layer port number of the resource
- Each Uri-Path Option specifies one segment of the absolute path to the resource
- Each Uri-Query Option specifies one argument parameterizing the resource

```
Input:

   Destination IP Address = [2001:db8::2:1]
   Destination UDP Port = 5683
   Uri-Host = "example.net"

Output:

   coap://example.net/
```

```
Input:

   Destination IP Address = [2001:db8::2:1]
   Destination UDP Port = 5683
   Uri-Host = "example.net"
   Uri-Path = ".well-known"
   Uri-Path = "core"

Output:

   coap://example.net/.well-known/core
```

Uri-Host, Uri-Port, Uri-Path, and Uri-Query

```
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"
   |      |
```

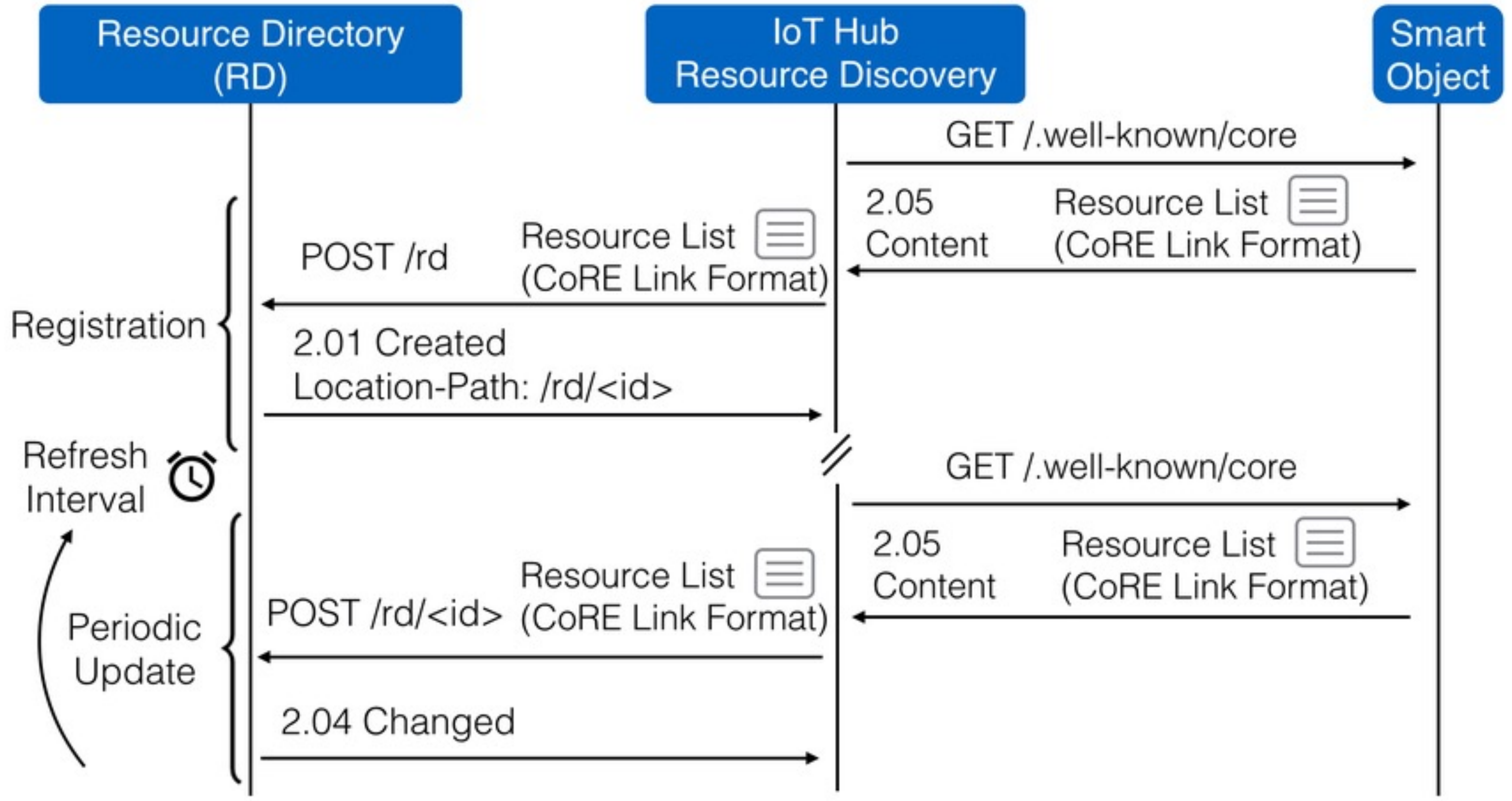
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| 1 | 0 |   0   |     GET=1     |          MID=0x7d34           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+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

 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           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 1 1 1 1 1 1|      "22.3 C" (6 B) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

CoAP Options

Option number	Option name	Format	Length (bits)
1	If-Match	opaque	0-8
3	Uri-Host	string	1-255
4	ETag	opaque	1-8
5	If-None-Match	empty	0
7	Uri-Port	uint	0-2
8	Location-Path	string	0-255
11	Uri-Path	string	0-255
12	Content-Format	uint	0-2
14	Max-Age	uint	0-4
15	Uri-Query	string	0-255
17	Accept	uint	0-2
20	Location-Query	string	0-255
35	Proxy-Uri	string	1-1034
39	Proxy-Scheme	string	1-255
60	Size1	uint	0-4

Location Path & Location Query

- The Location-Path and Location-Query options together indicate a relative URI that consists either of an absolute path, a query string or both
- A combination of these options is included in a 2.01 (Created) response to indicate the location of the resource created as the result of a POST request



CoAP Options

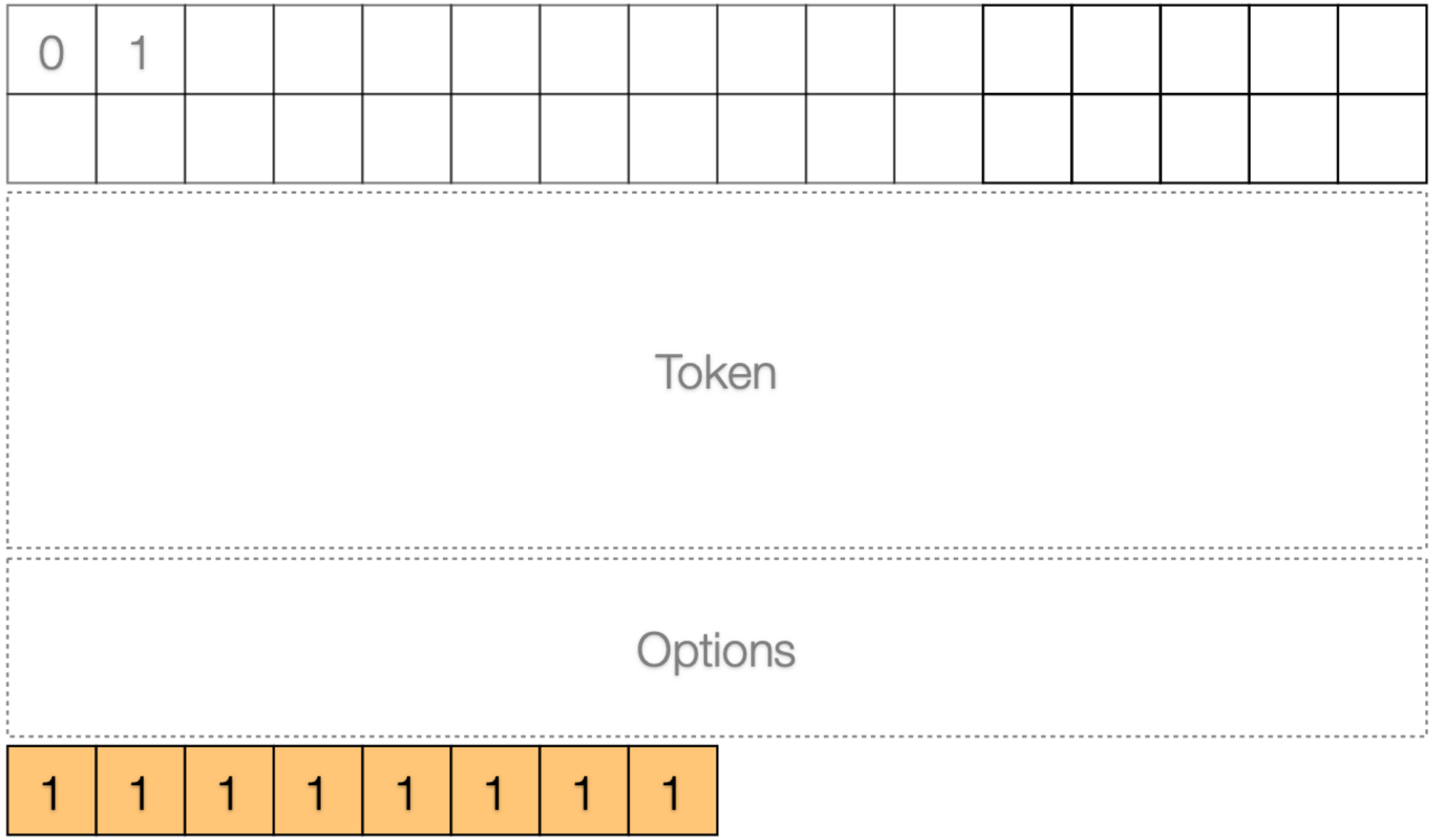
Option number	Option name	Format	Length (bits)
1	If-Match	opaque	0-8
3	Uri-Host	string	1-255
4	ETag	opaque	1-8
5	If-None-Match	empty	0
7	Uri-Port	uint	0-2
8	Location-Path	string	0-255
11	Uri-Path	string	0-255
12	Content-Format	uint	0-2
14	Max-Age	uint	0-4
15	Uri-Query	string	0-255
17	Accept	uint	0-2
20	Location-Query	string	0-255
35	Proxy-Uri	string	1-1034
39	Proxy-Scheme	string	1-255
60	Size1	uint	0-4

Content-Format and Accept

- The Content-Format option indicates the representation format of the message payload
- The Accept option can be used to indicate which Content-Format is acceptable to the client
- The representation format is given as a numeric content format identifier that is defined in the CoAP Content Format Registry

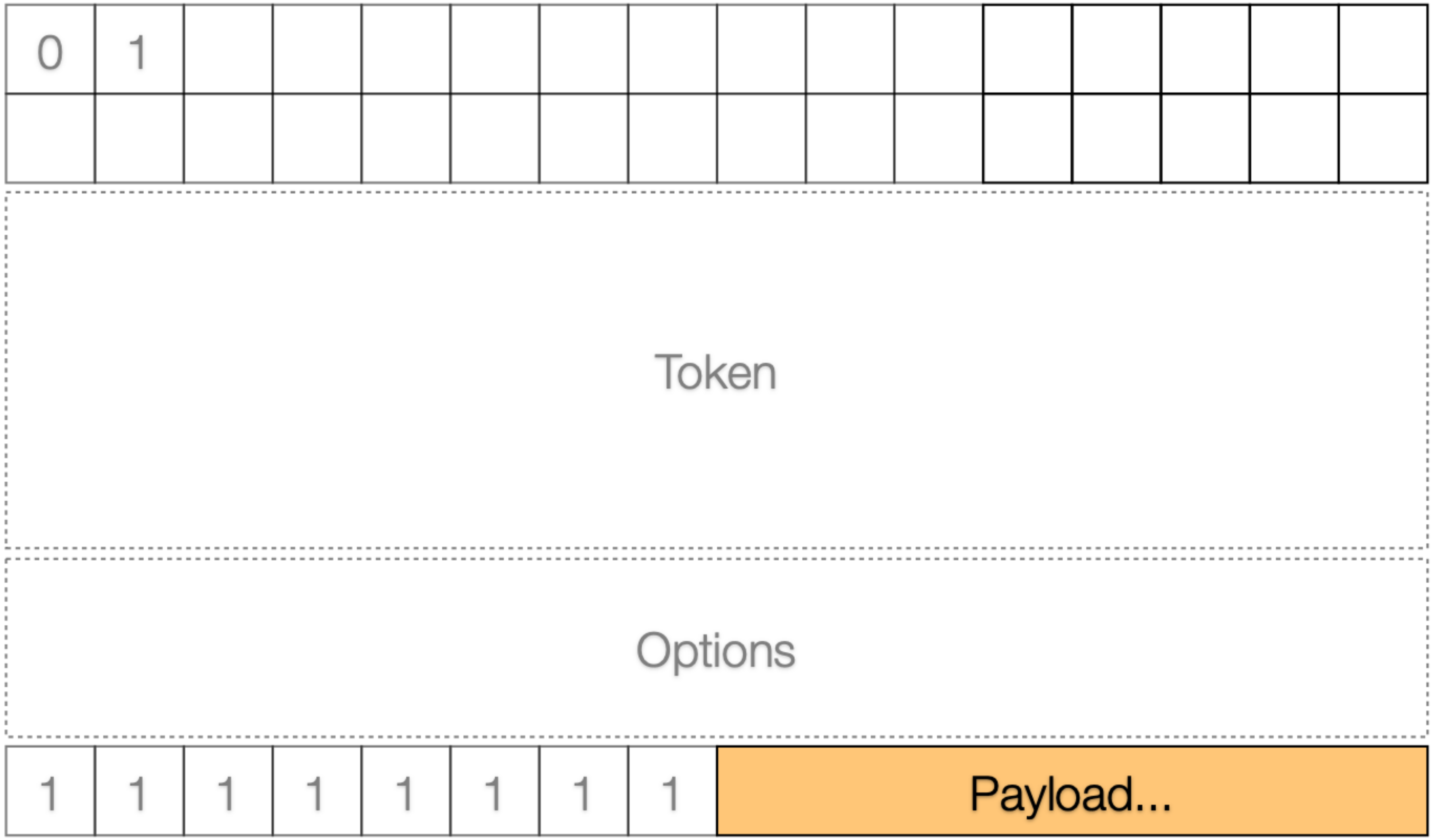
Media type	**Encoding**	**Content format identifier**
text/plain	charset=utf-8	0
application/link-format	-	40
application/xml	-	41
application/octet-stream	-	42
application/exi	-	47
application/json	-	50

CoAP Message Format



Payload Marker (0xFF): indicates the end of options and the start of the payload

CoAP Message Format



Payload: extends to the end of the UDP datagram

IoT Oriented Features

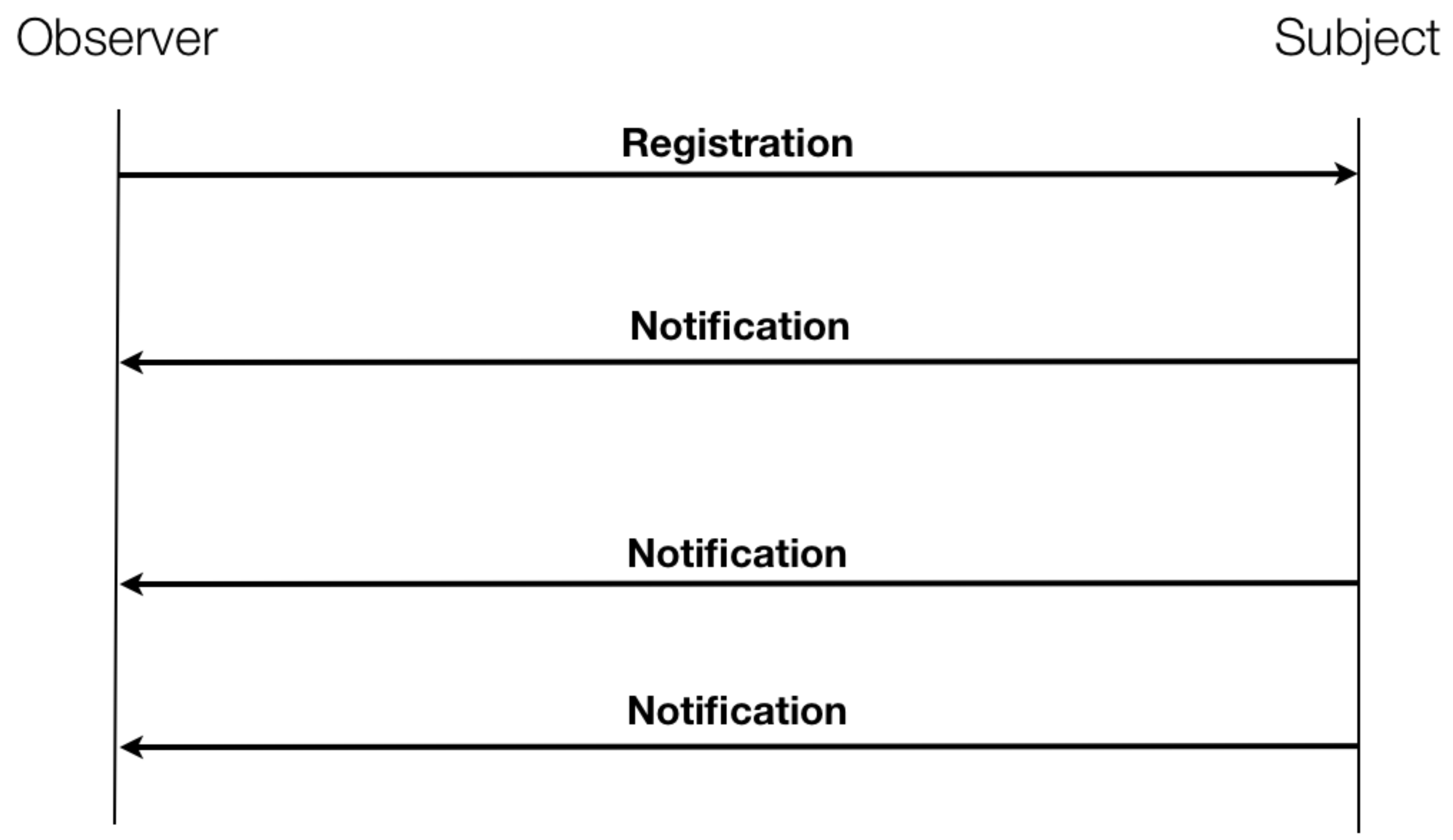
- The IETF CoRE Working Group is also defining other IoT-oriented mechanisms to extend CoAP:
  - resource observation
  - block-wise transfers
  - multicast communication
  - resource discovery
  - HTTP/CoAP proxying

Resource Observation

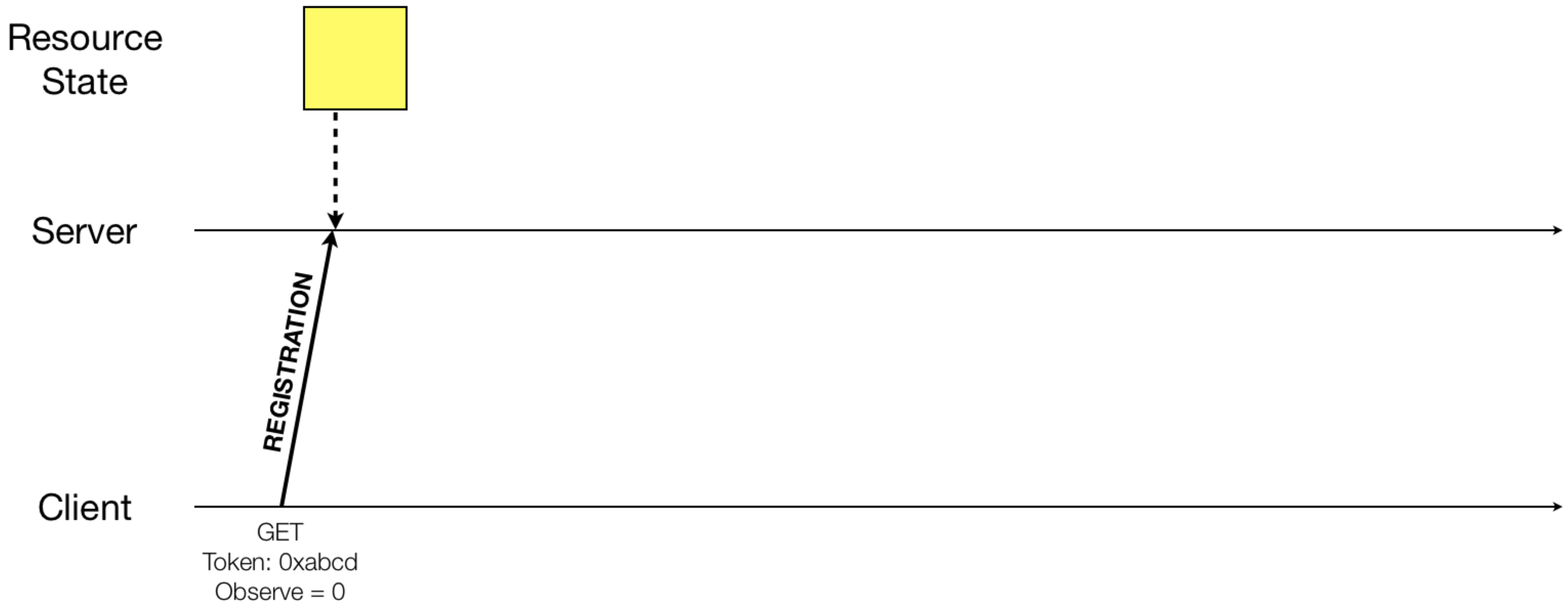
- The state of a resource on a CoAP server can change over time: how can clients always have the most recent state?
- **Polling: perform GET requests periodically**
- **Polling is inefficient:**
  - resource may not change state between two successive GET requests: unnecessary requests consume energy on the constrained CoAP server
  - the resource may have changed its state between two successive GET requests: for some time the client was not having the most updated state
- The Observe option has been introduced to avoid polling (RFC 7641)
- When included in a GET request, **tells the server to send notifications to the client whenever the state of the resource changes**
- **Similar to a Pub/Sub model, but slightly different: it is still request/response, but many responses are sent after a single request**
- Responses are linked to the request because they **report the same Token**

Resource Observation

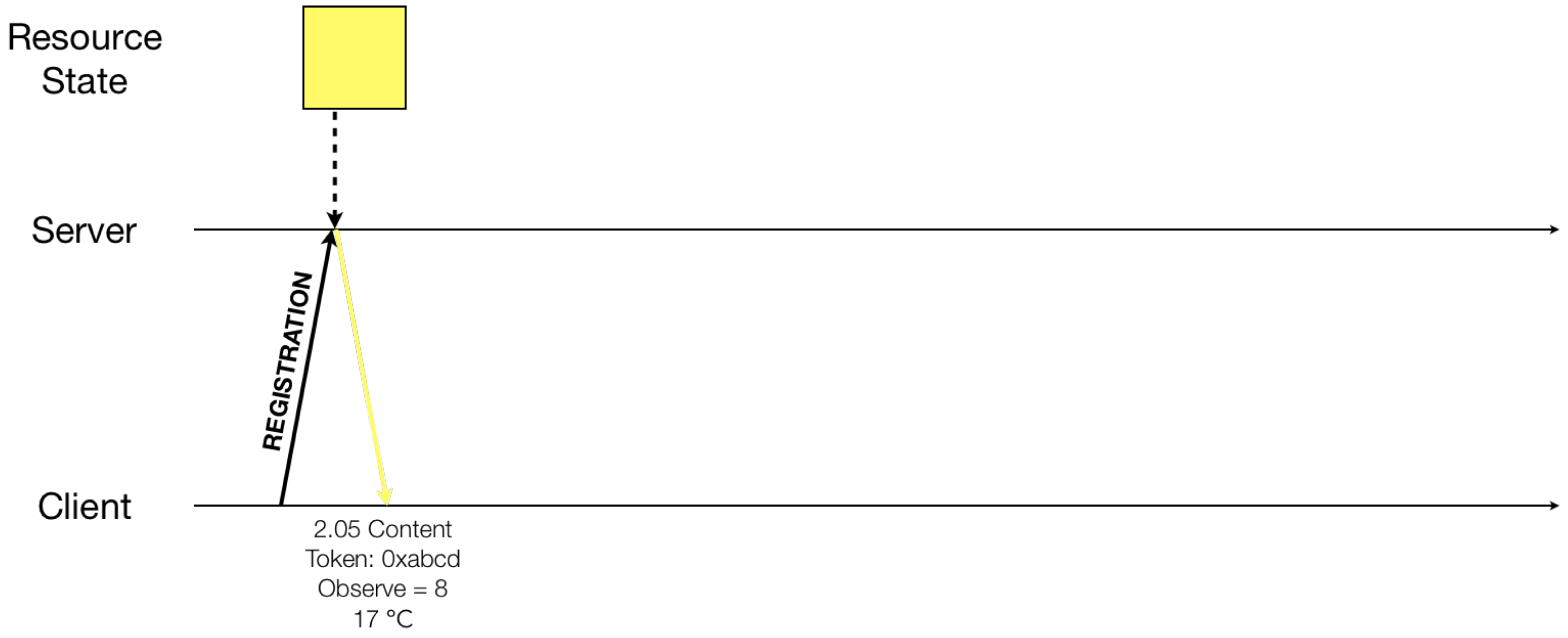
- The Observe Option implements the Observer Design Pattern



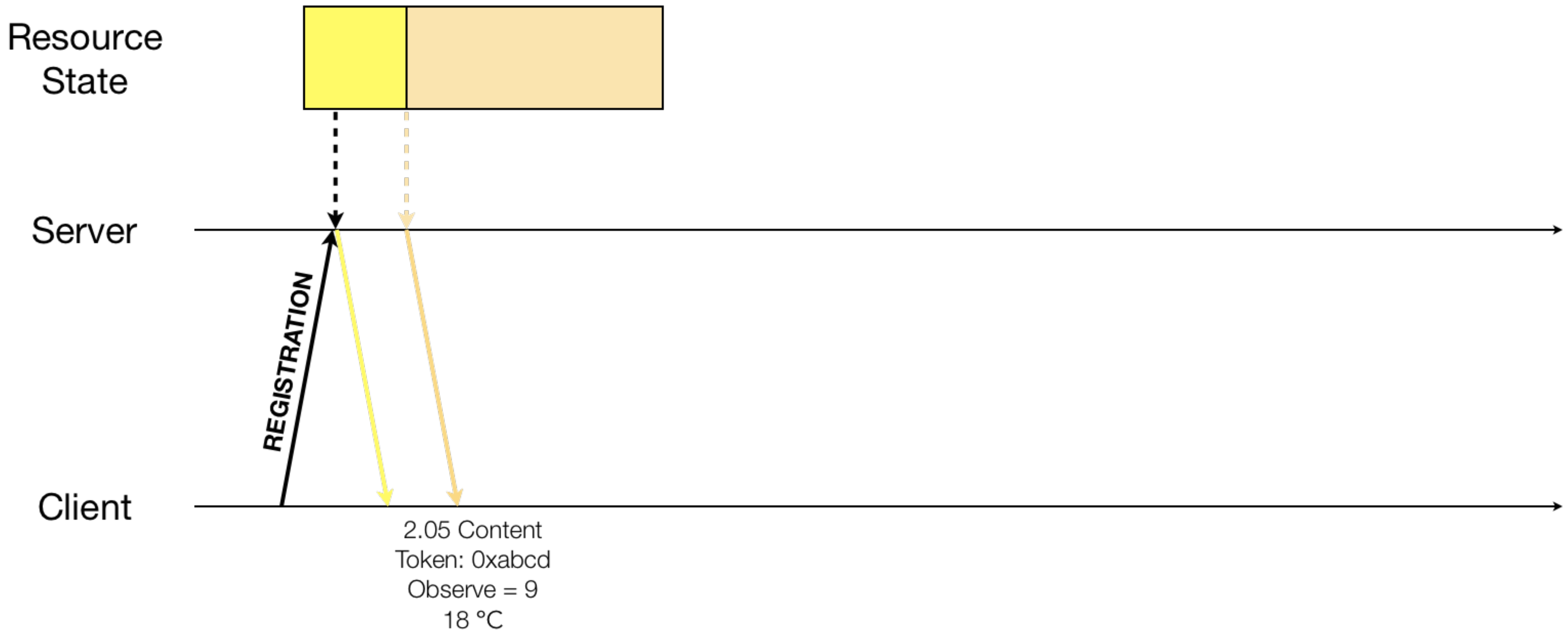
Resource Observation



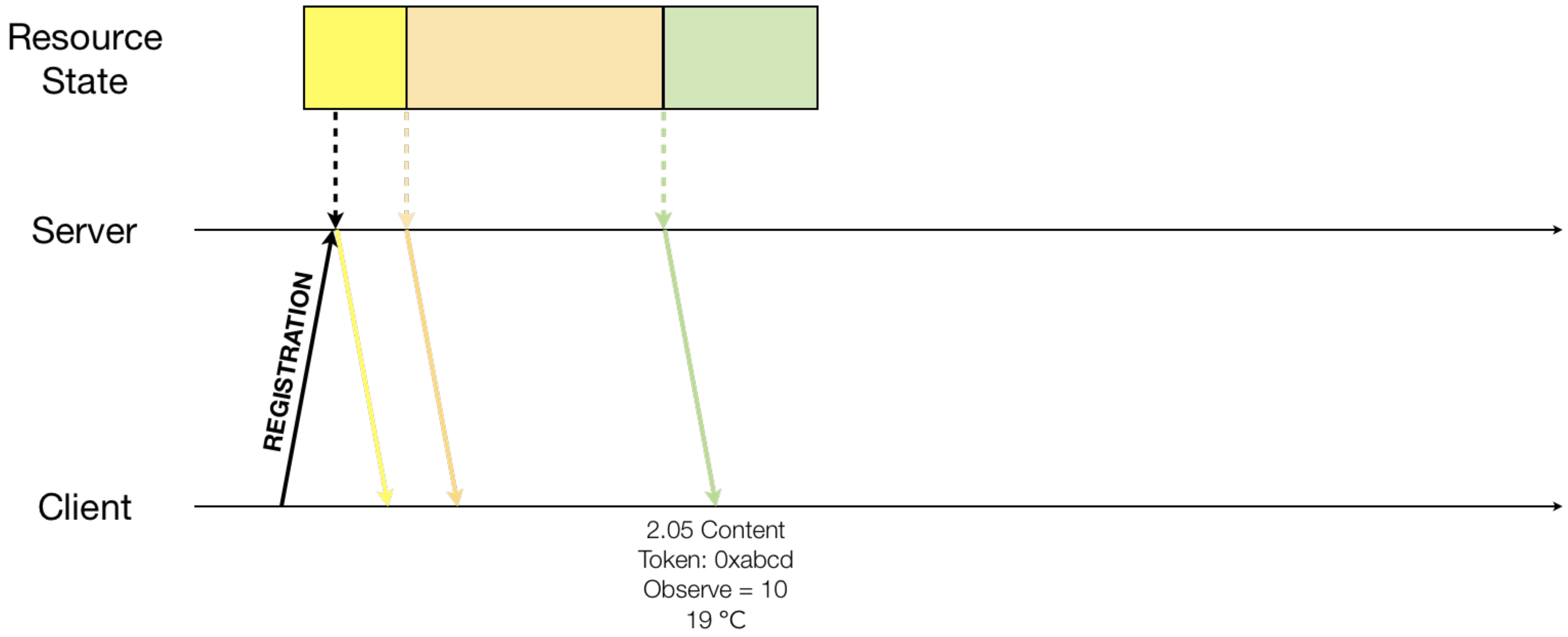
Resource Observation



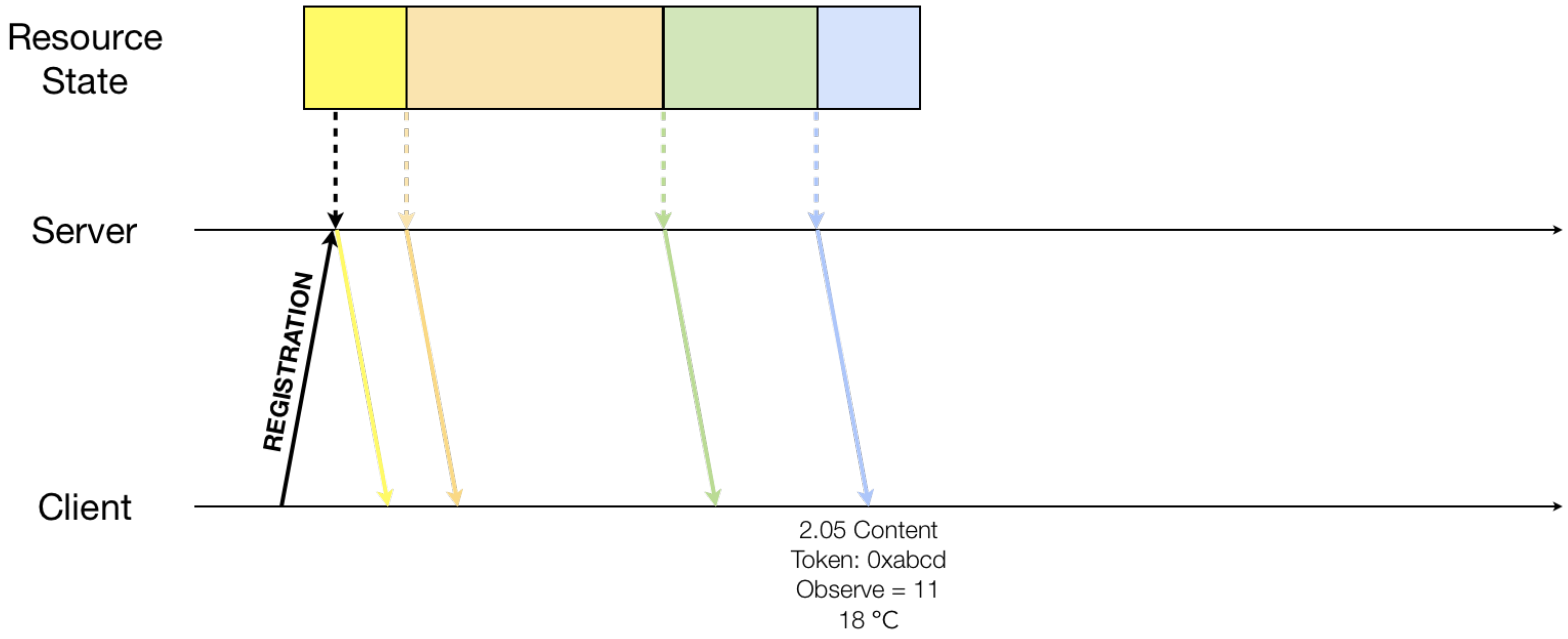
Resource Observation



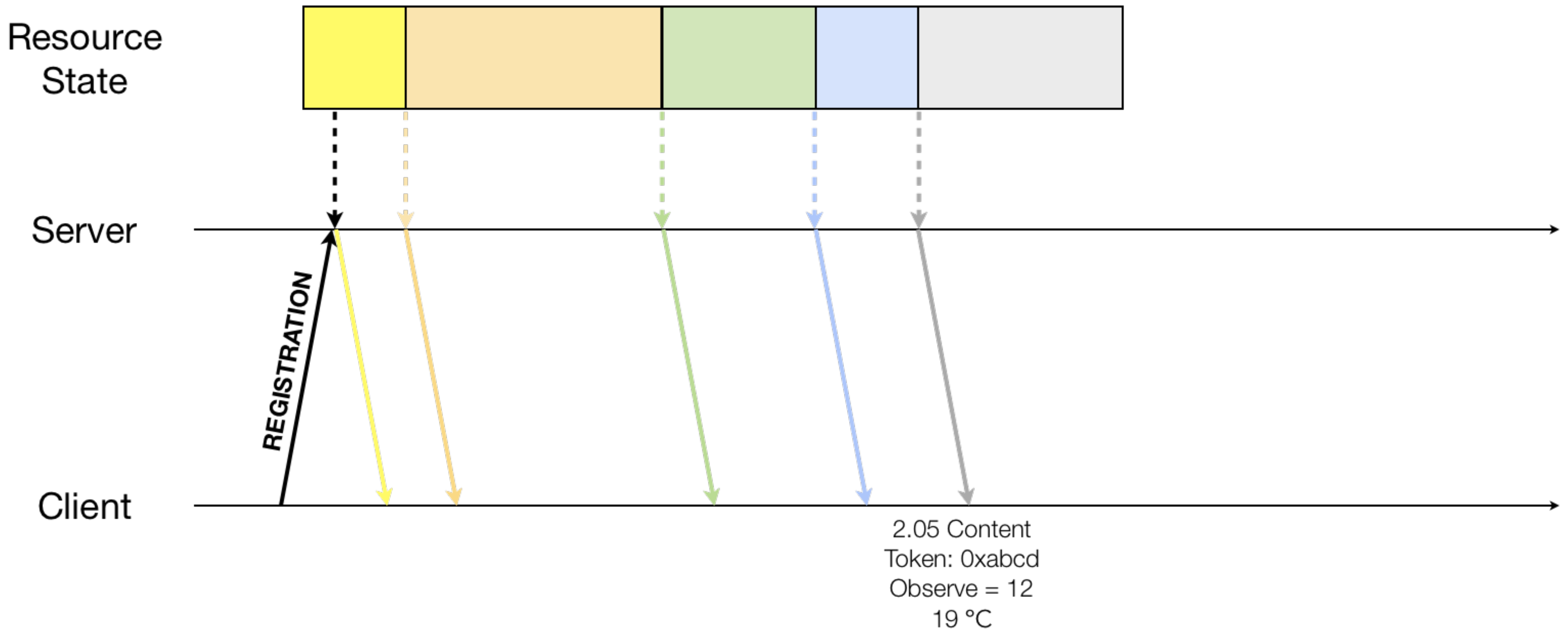
Resource Observation



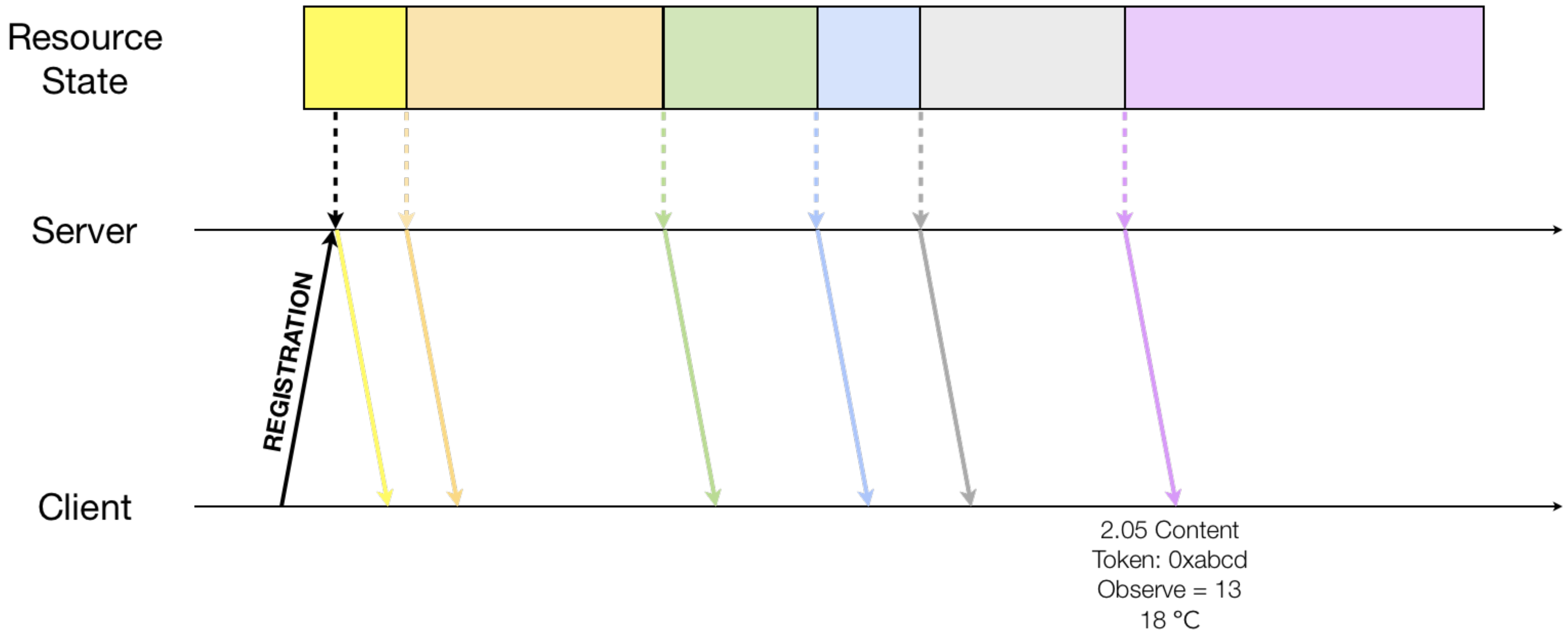
Resource Observation



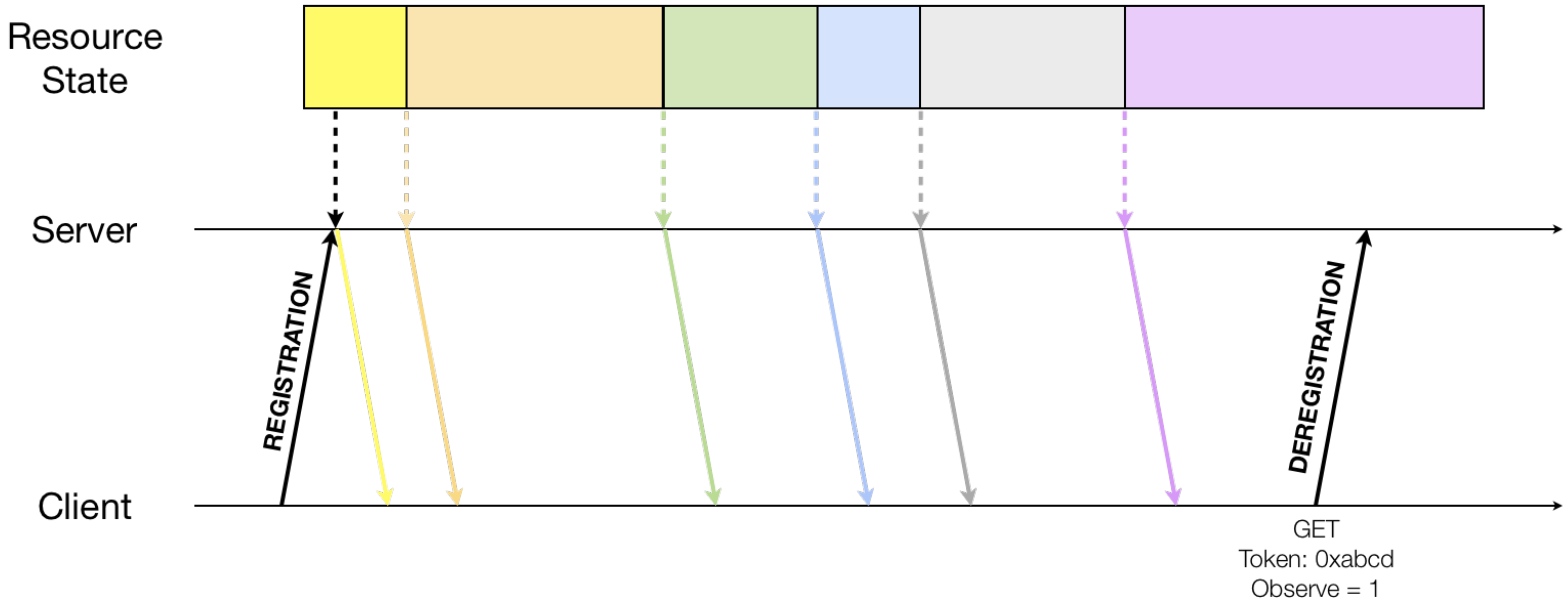
Resource Observation



Resource Observation



Resource Observation



Block-wise transfers

- CoAP messages work well for the small payloads
- The size of CoAP messages is limited due to the maximum size of UDP packets (65536 bytes)
- However:
  - Applications may need to transfer larger payloads
  - Constrained servers or clients may not be able to process payloads of arbitrary sizes (e.g., their buffer limits them to accept payloads of 128 bytes, but 1Kbyte of data needs to be transferred)
- Block-wise transfers have been defined to divide large (whatever this means) amounts of data into several chunks of a given size (draft-ietf-core-block-19)
- **Block1** and **Block2** options serve the need to provide a minimal way to transfer larger representations in a block-wise fashion

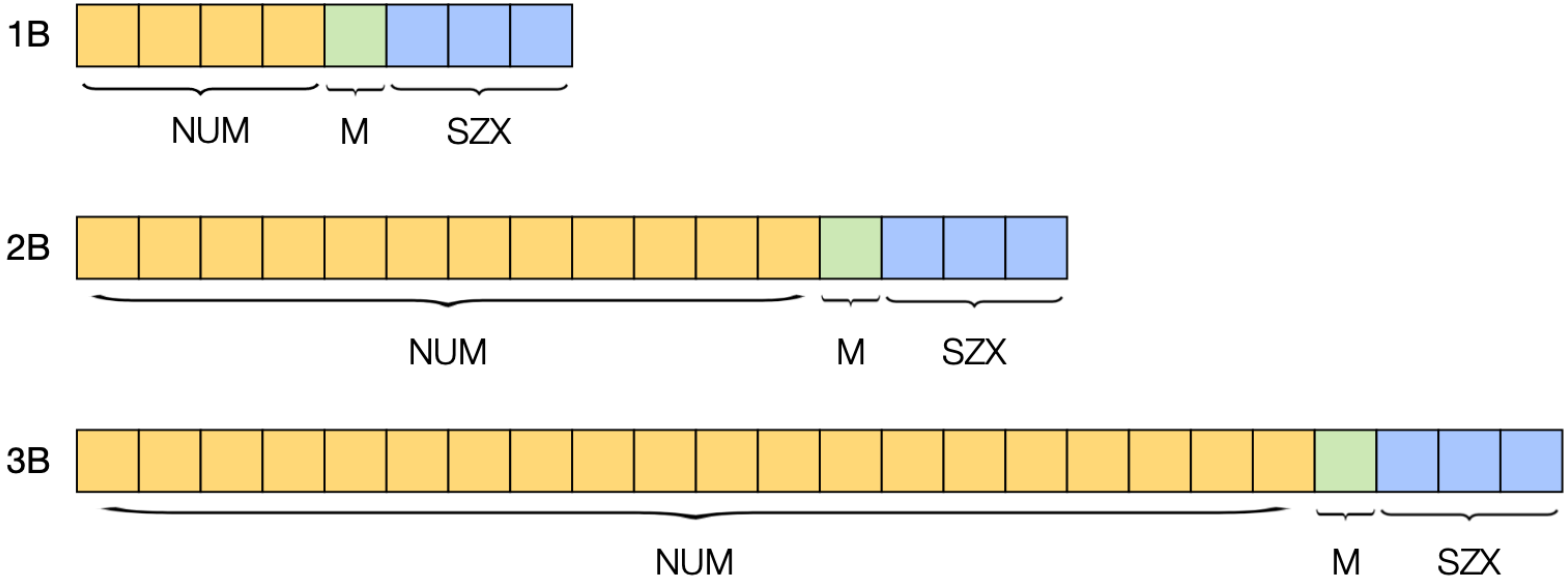
Block1 and Block2 Options

- Block1 (number 27): pertains to the request payload (useful with POST/PUT requests)
- Block2 (number 23): pertains to the response payload (useful with GET requests)
- If Block1 is in the request or Block2 is in the response, they describe how the payload of the message is a part of the a large body being transferred (**descriptive usage**)
- If Block1 is in the response or Block2 is in the request, they describe how the payload will be formed or has been processed (**control usage**)
- Clients and servers can negotiate a block size that can be used for all transfers, using the **Size1** and **Size2** options

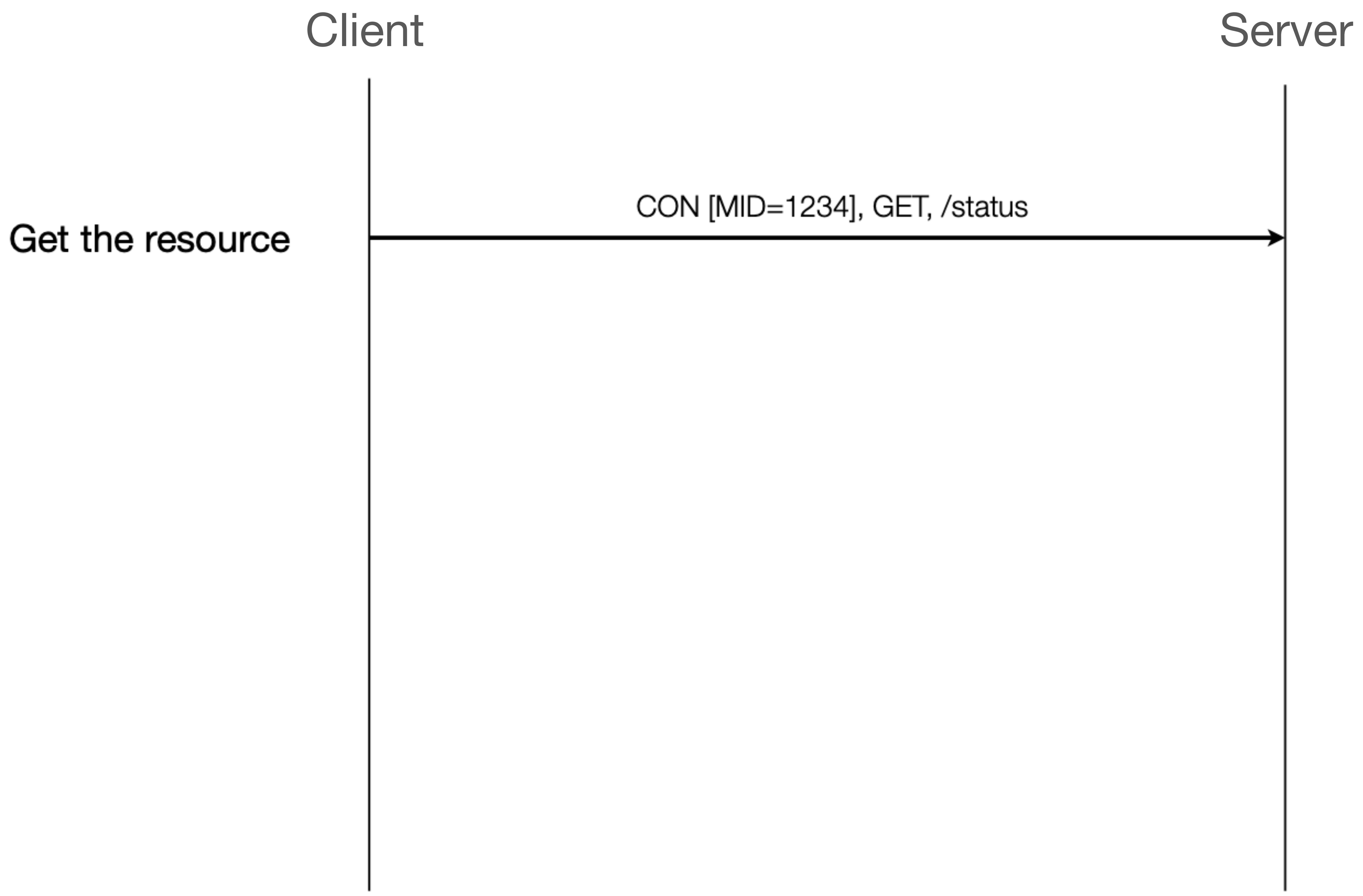
Anatomy of a Block Option Value

- The value of the Block Option is a variable-size (0 to 3 bytes)
- Contains 3 fields:
  - **NUM:** the relative number of the block within a sequence of blocks with a given size
  - **M (1 bit):** "more" bit; whether more blocks are following (i.e., whether this is the last block)
  - **SZX (3 bits):** size exponent of the block. SZX can be 0 to 6; 7 is reserved and cannot be used
  - The actual size of the block is computed as 2^{4+SZX}, which means that blocks can have a size from 2^4 to 2^{10} bytes

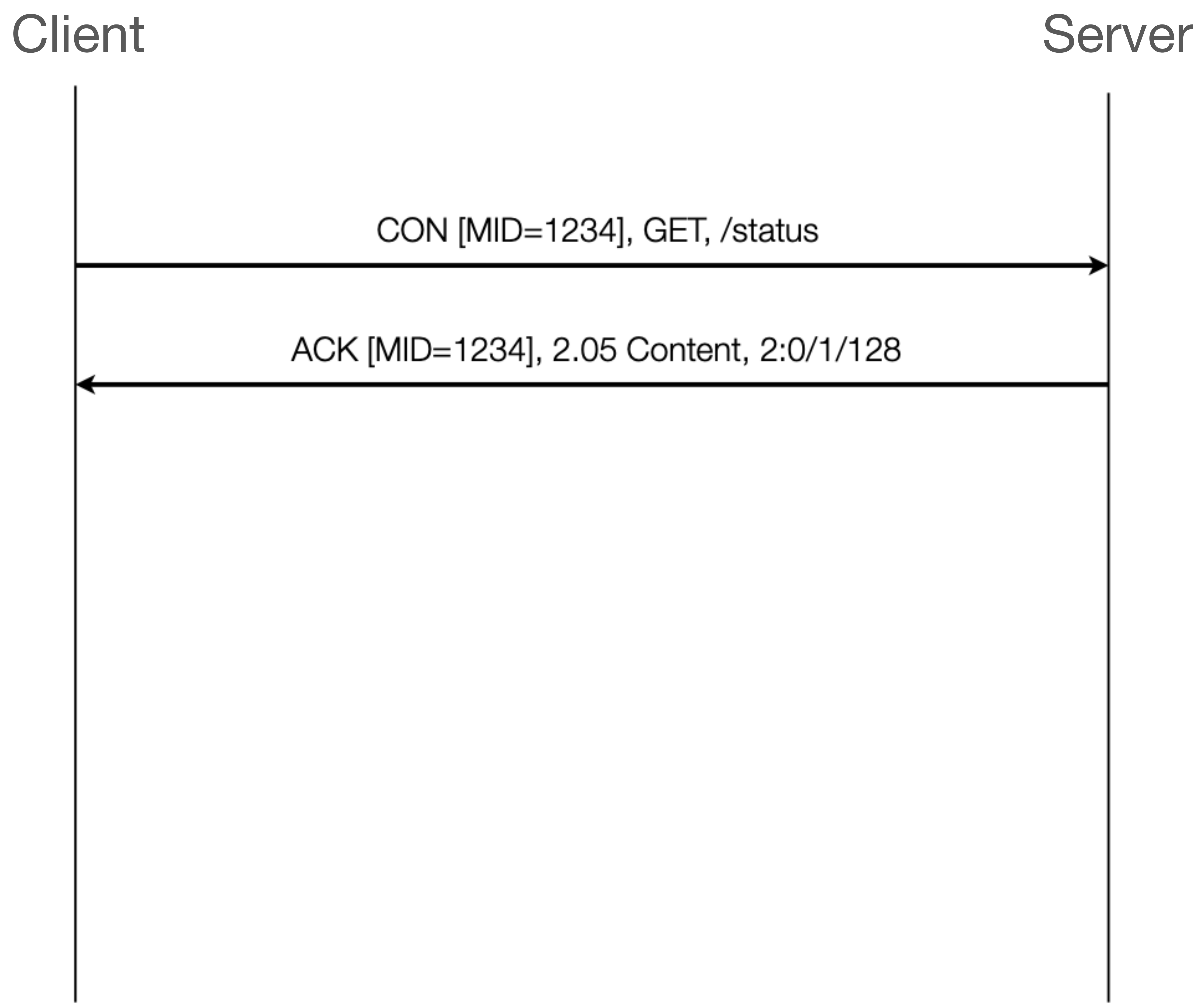
Anatomy of a Block Option Value



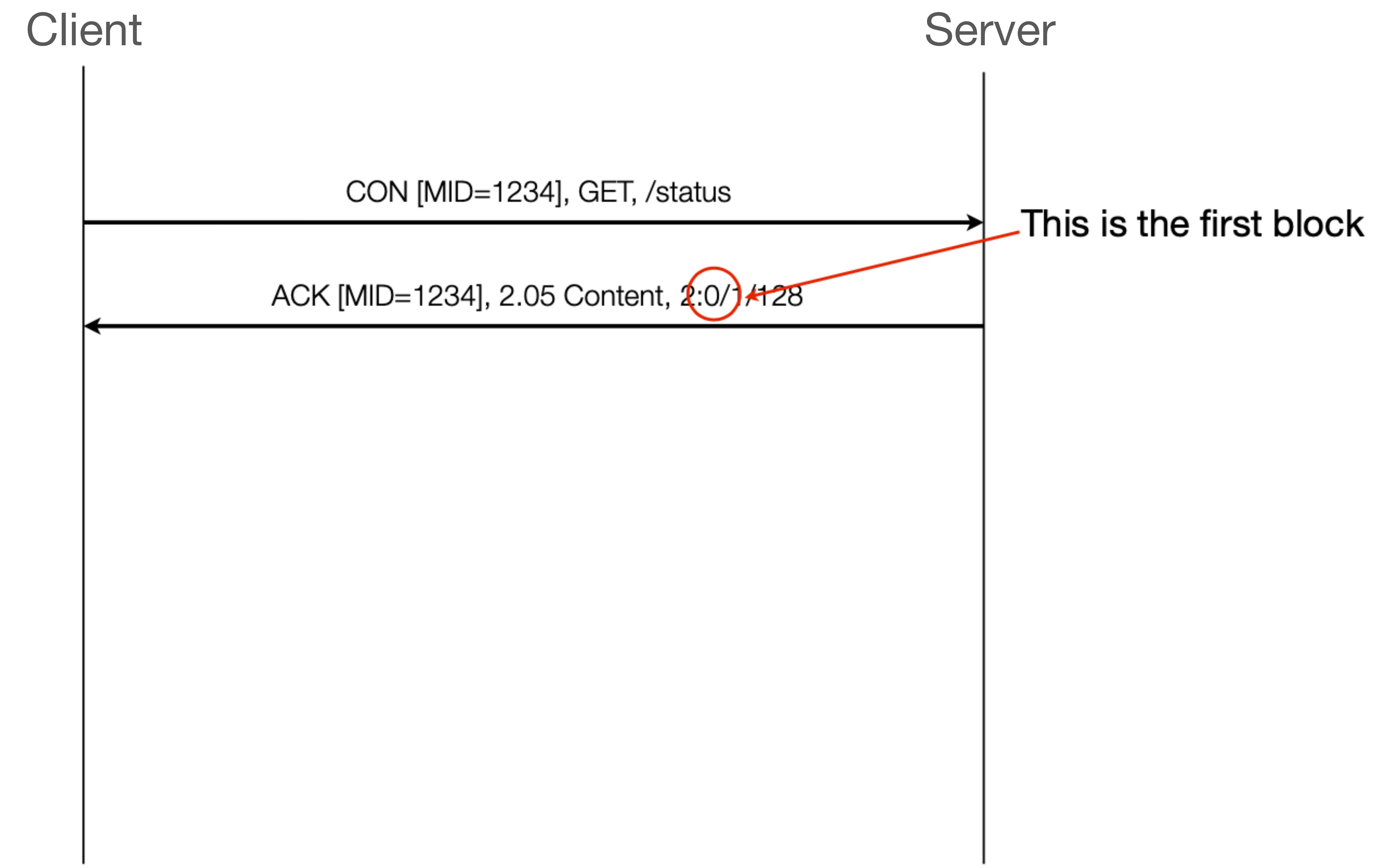
Block2 Option Example (Response Payload)



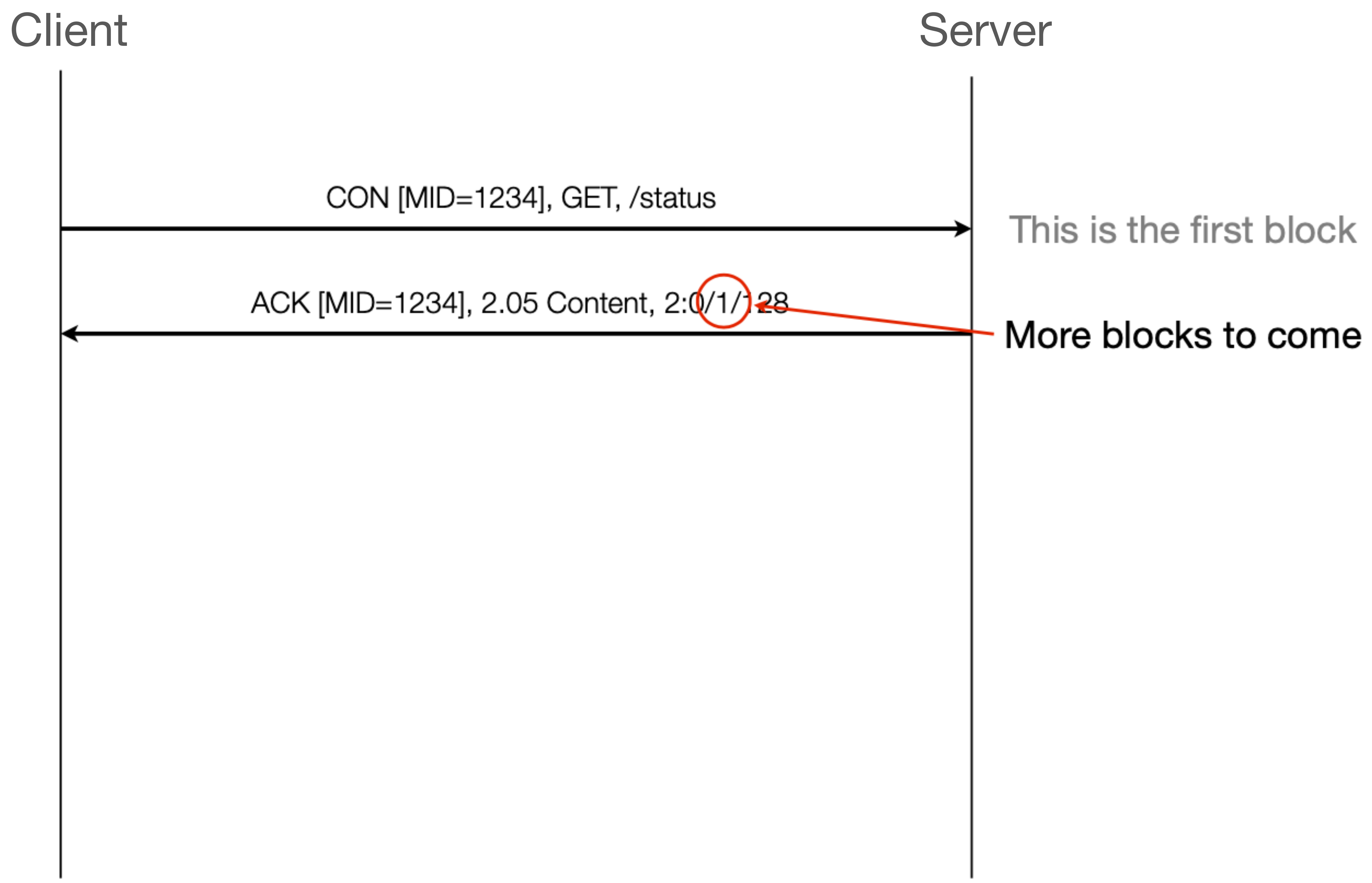
Block2 Option Example (Response Payload)


Client
Server
CON [MID=1234], GET, /status
ACK [MID=1234], 2.05 Content, 2:0/1/128

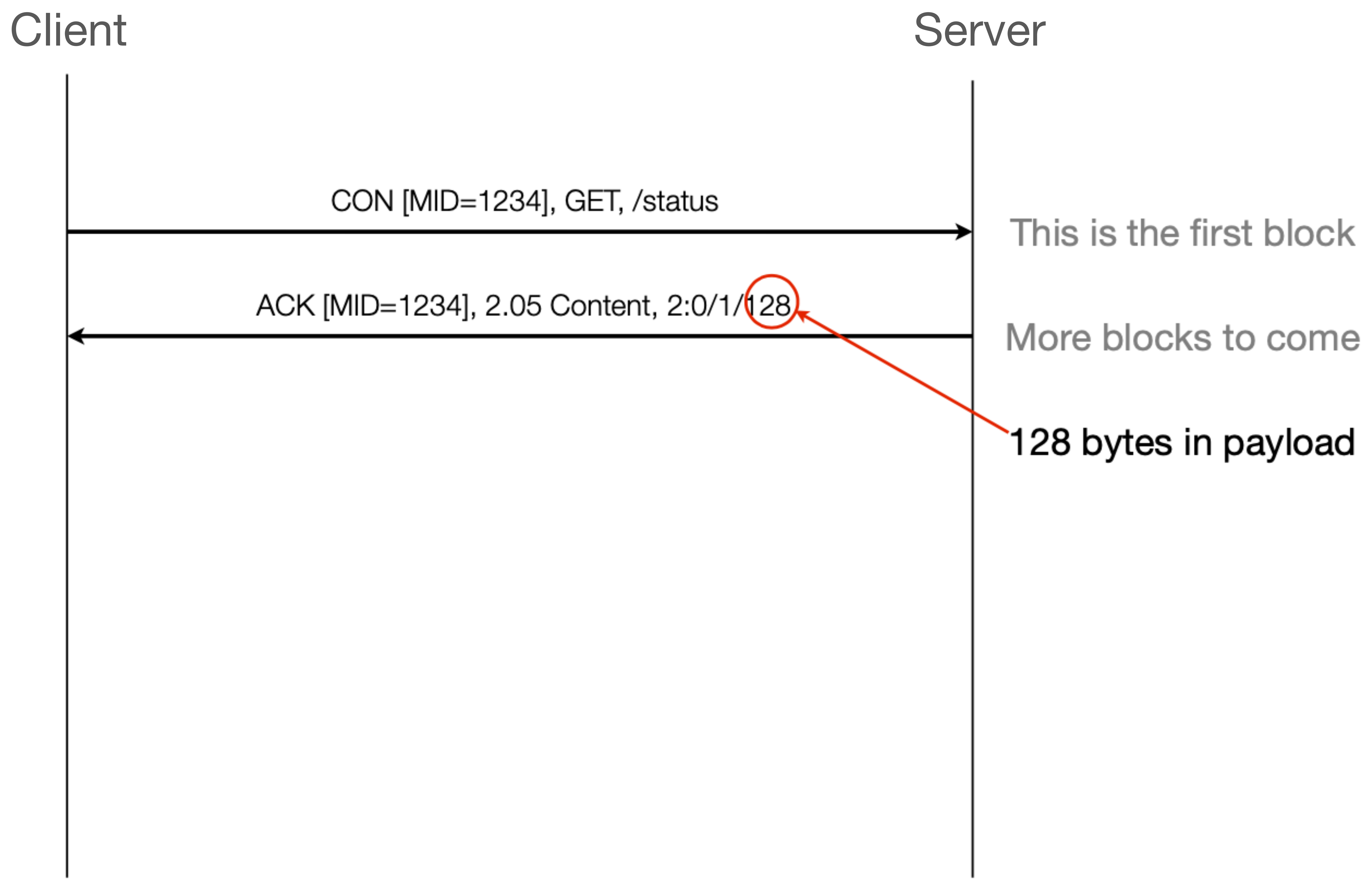
Block2 Option Example (Response Payload)


Client
Server
CON [MID=1234], GET, /status
ACK [MID=1234], 2.05 Content, 2:0/1/128
This is the first block

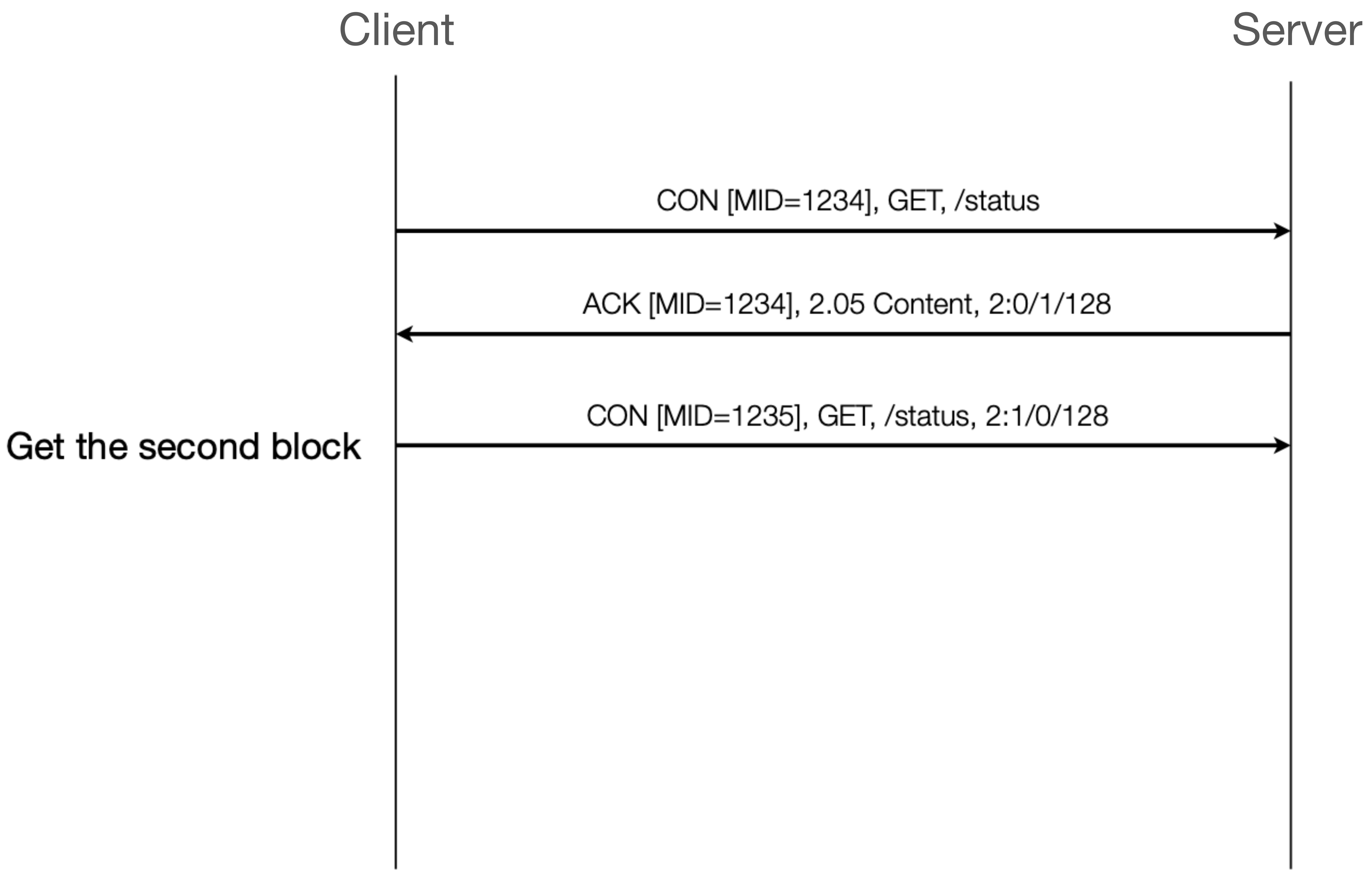
Block2 Option Example (Response Payload)


Client
Server
CON [MID=1234], GET, /status
ACK [MID=1234], 2.05 Content, 2:0/1/128
This is the first block
More blocks to come

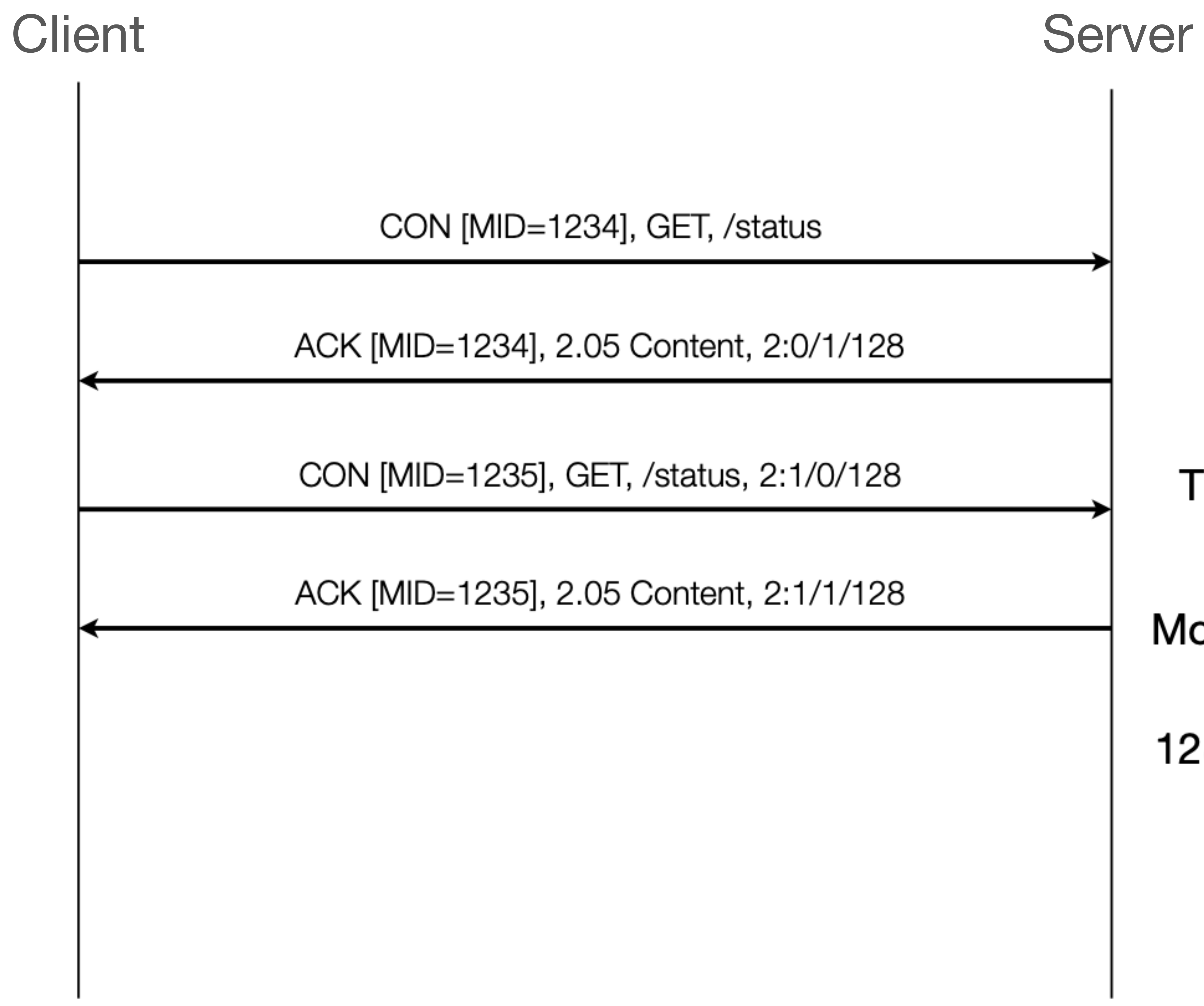
Block2 Option Example (Response Payload)



Block2 Option Example (Response Payload)


Client
Server
CON [MID=1234], GET, /status
ACK [MID=1234], 2.05 Content, 2:0/1/128
CON [MID=1235], GET, /status, 2:1/0/128
Get the second block

Block2 Option Example (Response Payload)


Client
Server
CON [MID=1234], GET, /status
ACK [MID=1234], 2.05 Content, 2:0/1/128
CON [MID=1235], GET, /status, 2:1/0/128
ACK [MID=1235], 2.05 Content, 2:1/1/128

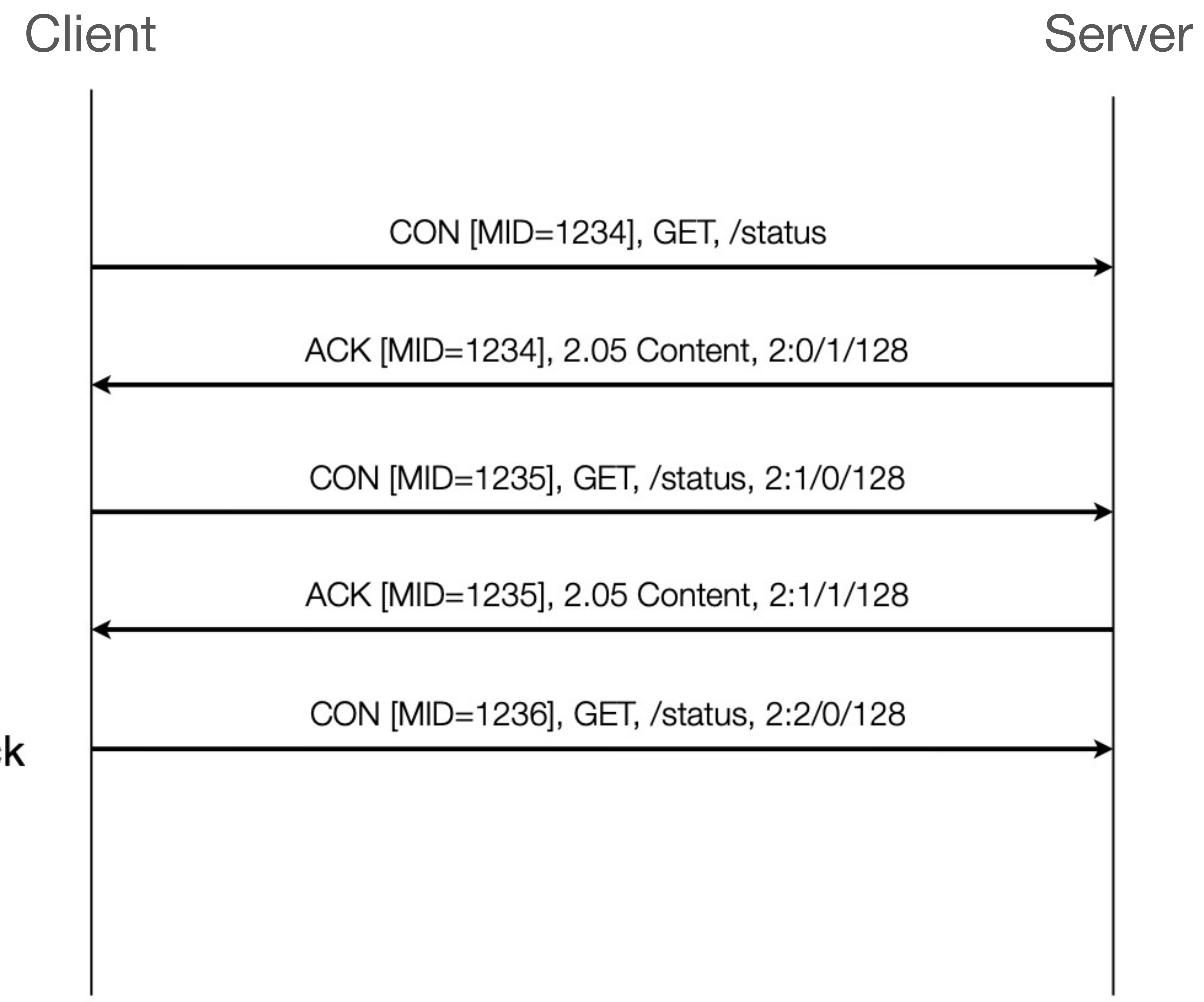
This is the second block

More blocks to come

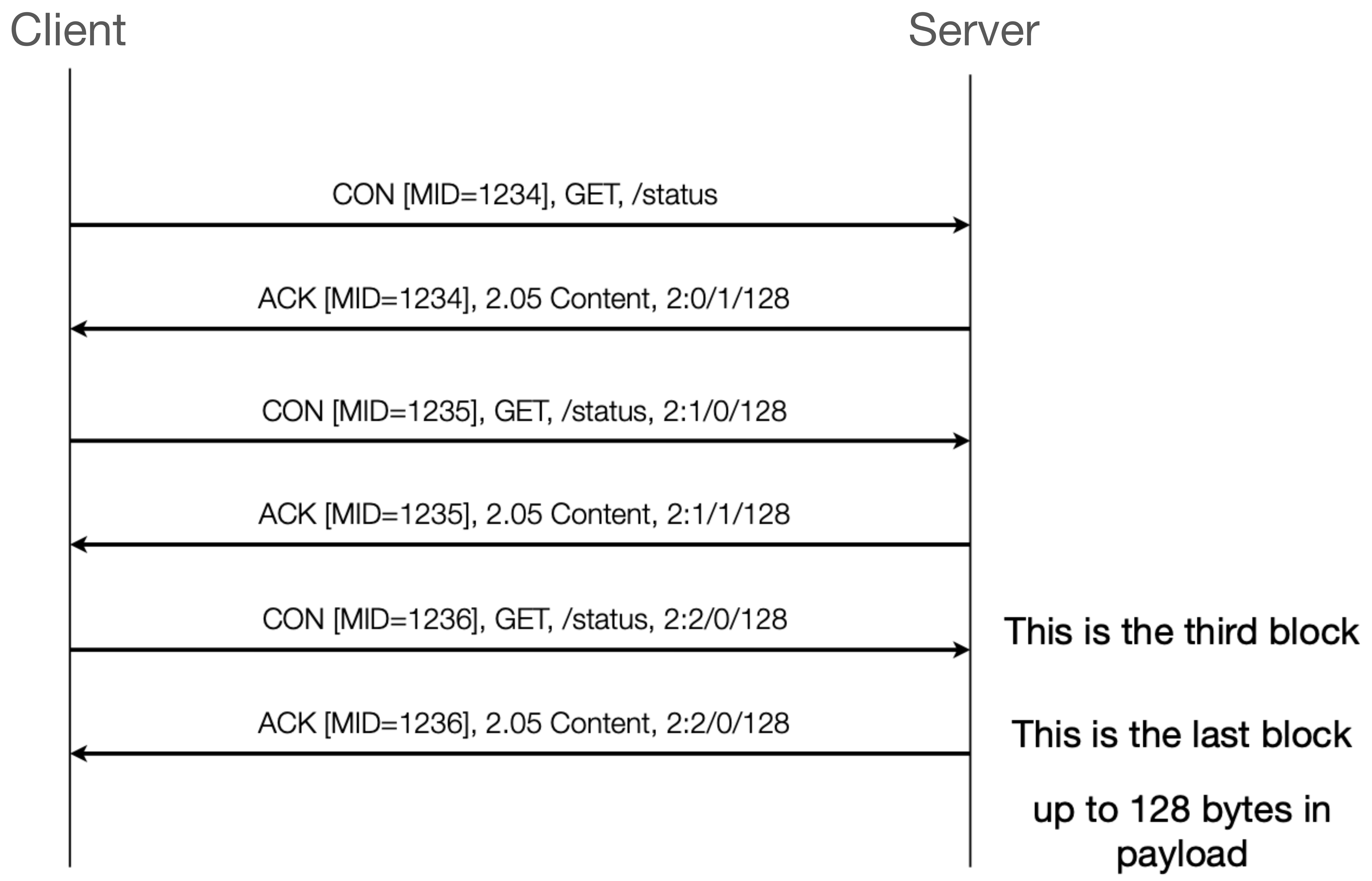
128 bytes in payload

Block2 Option Example (Response Payload)

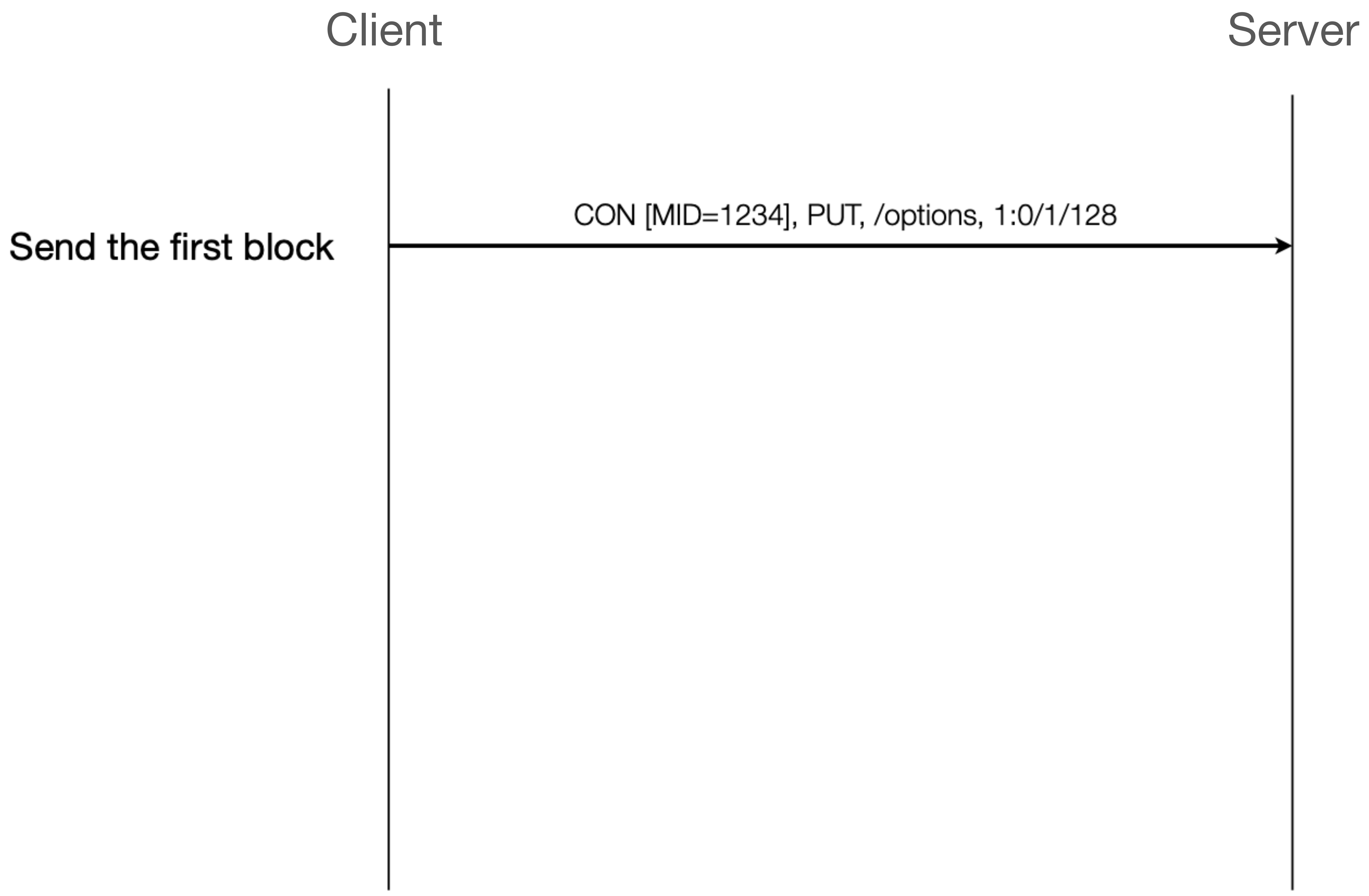
Get the third block


Client
Server
CON [MID=1234], GET, /status
ACK [MID=1234], 2.05 Content, 2:0/1/128
CON [MID=1235], GET, /status, 2:1/0/128
ACK [MID=1235], 2.05 Content, 2:1/1/128
CON [MID=1236], GET, /status, 2:2/0/128

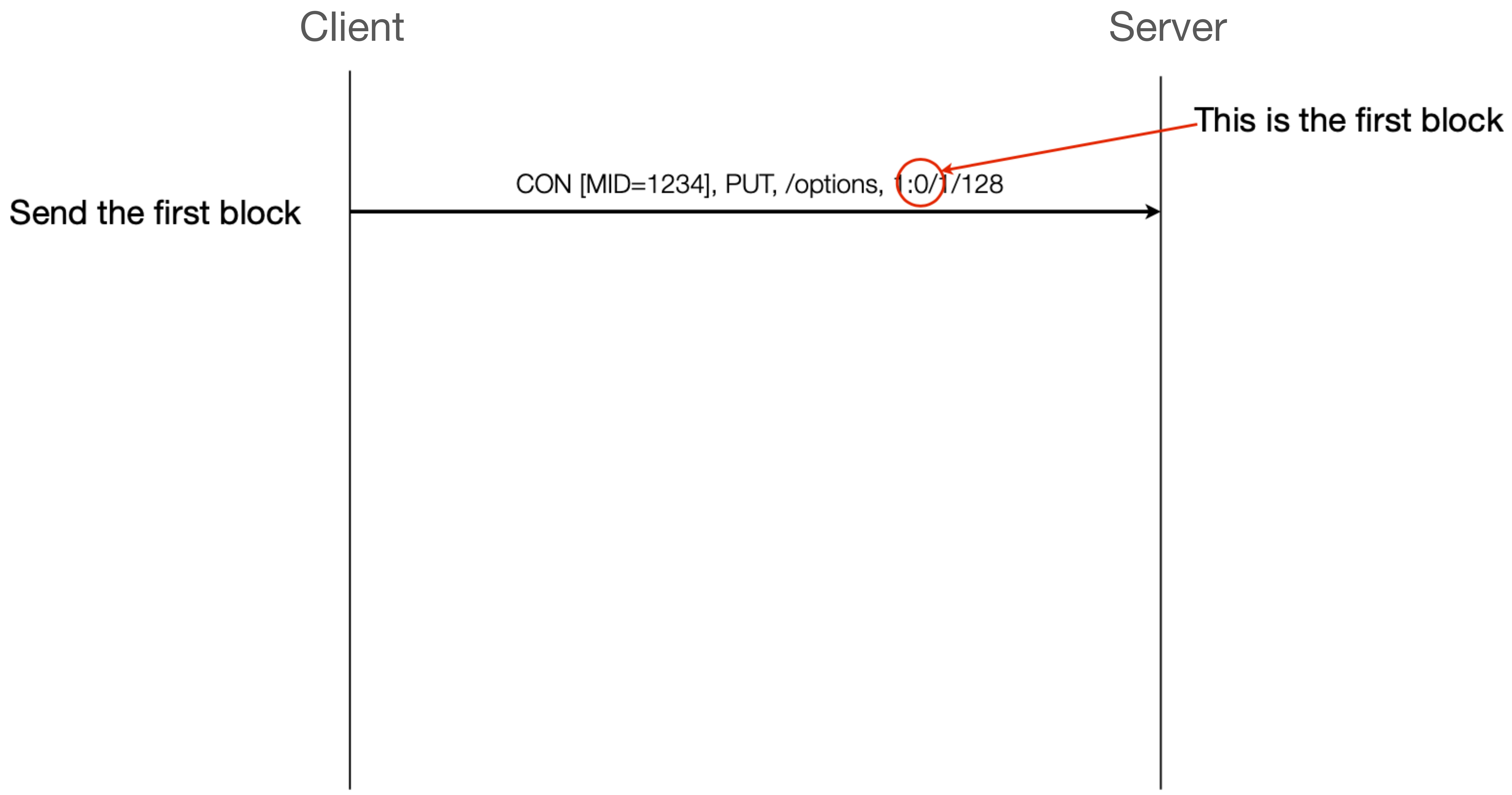
Block2 Option Example (Response Payload)


Client
Server
CON [MID=1234], GET, /status
ACK [MID=1234], 2.05 Content, 2:0/1/128
CON [MID=1235], GET, /status, 2:1/0/128
ACK [MID=1235], 2.05 Content, 2:1/1/128
CON [MID=1236], GET, /status, 2:2/0/128
This is the third block
ACK [MID=1236], 2.05 Content, 2:2/0/128
This is the last block
up to 128 bytes in payload

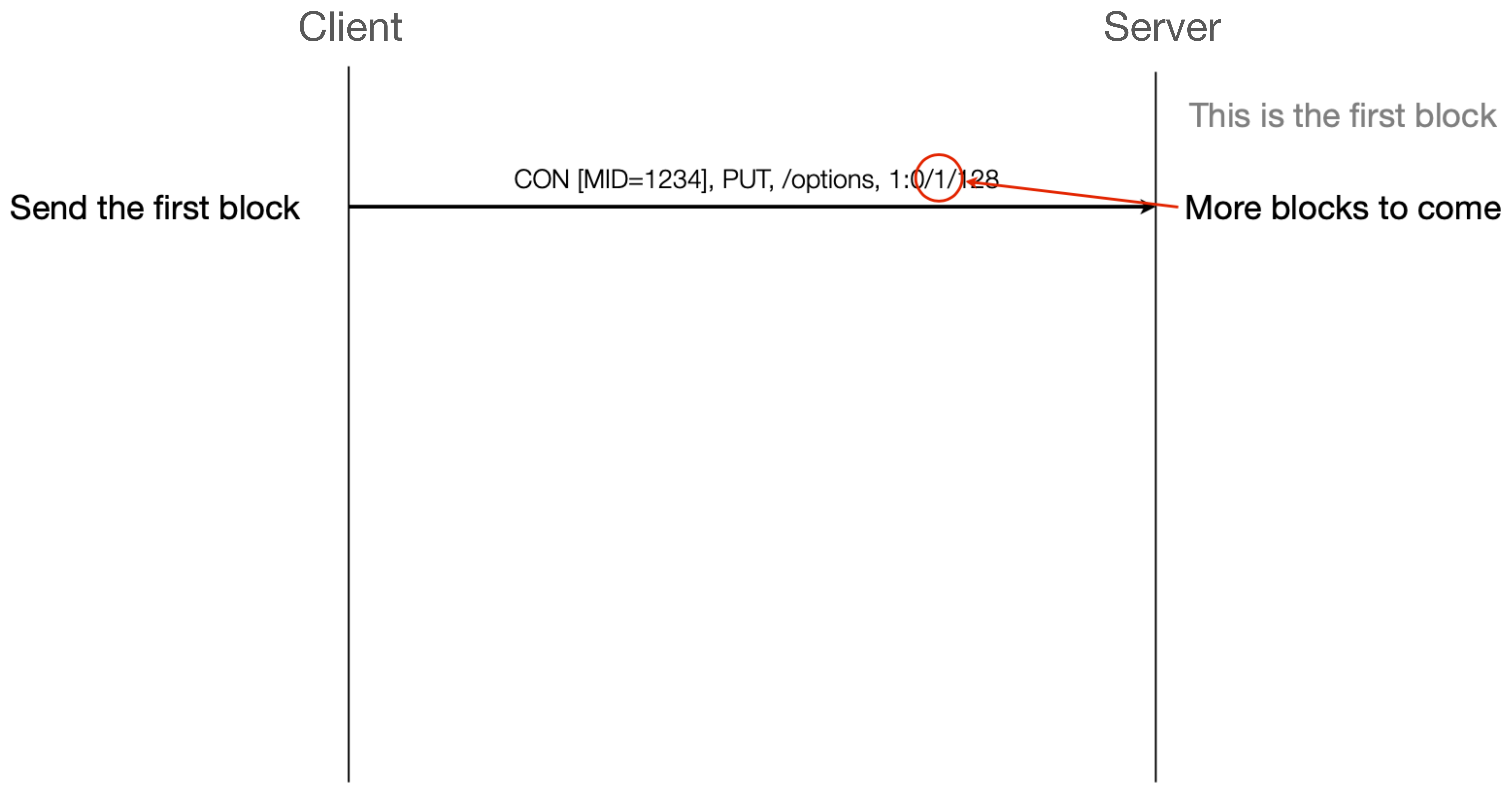
Block1 Option Example (Request Payload)



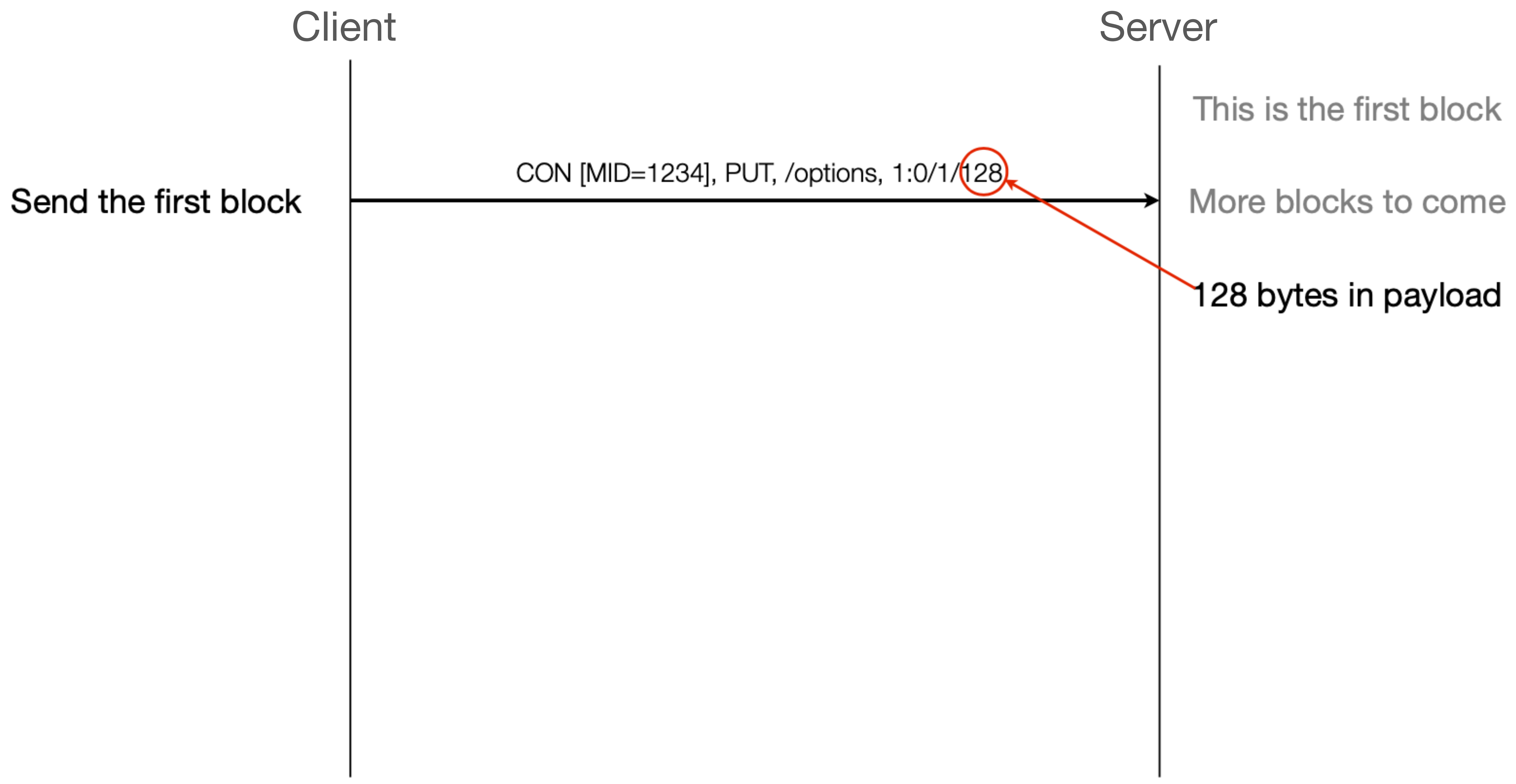
Block1 Option Example (Request Payload)



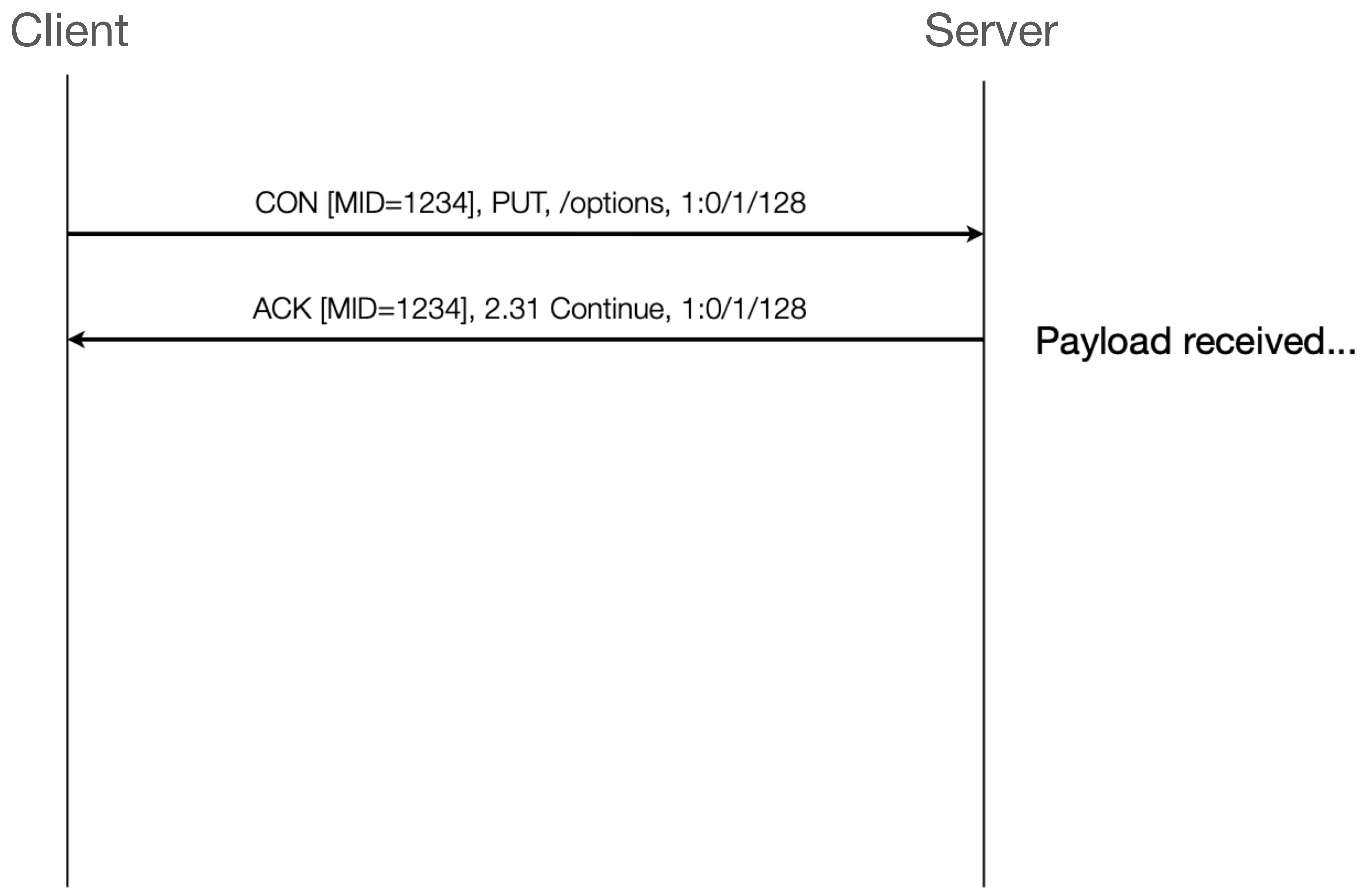
Block1 Option Example (Request Payload)



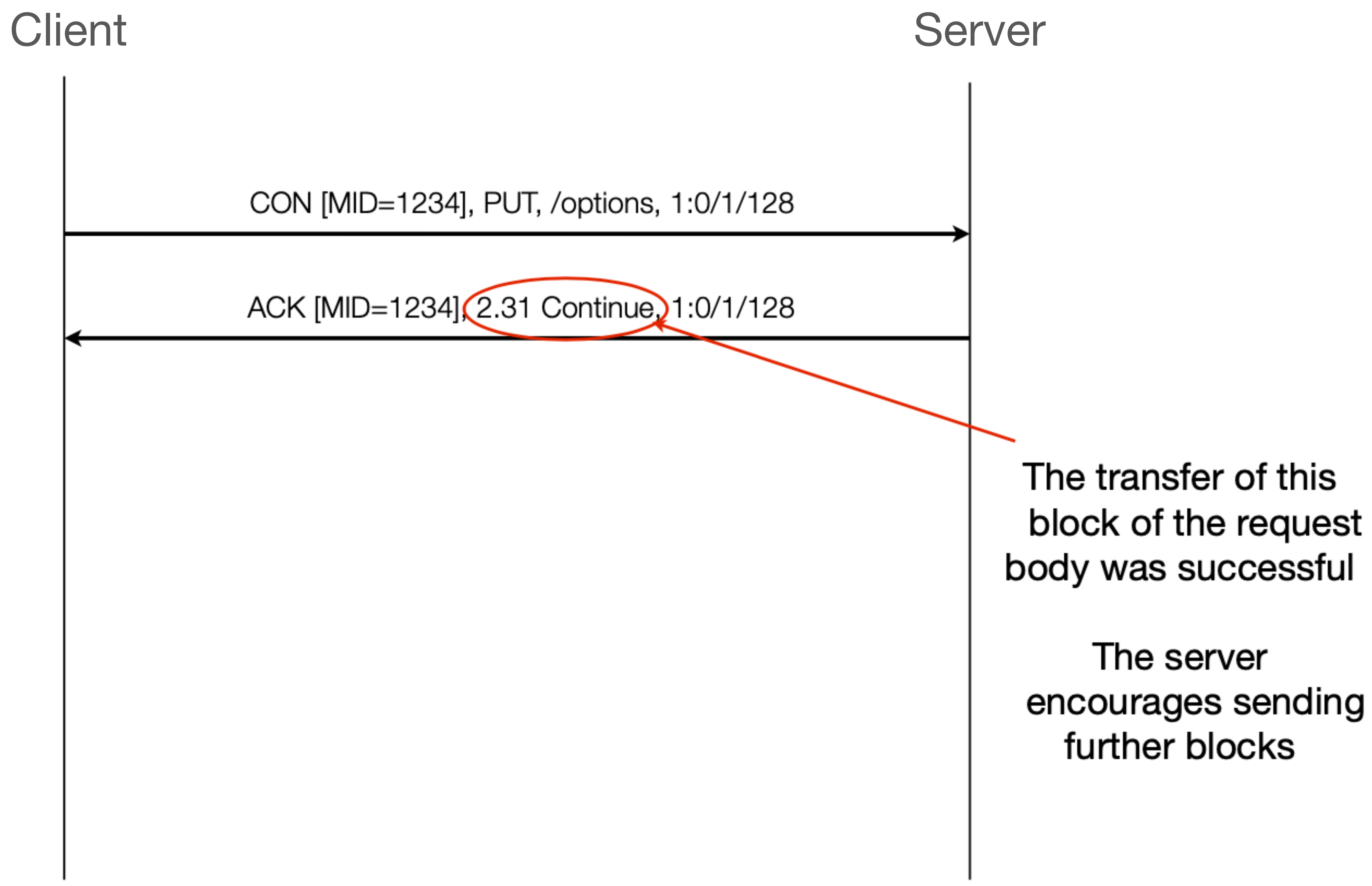
Block1 Option Example (Request Payload)



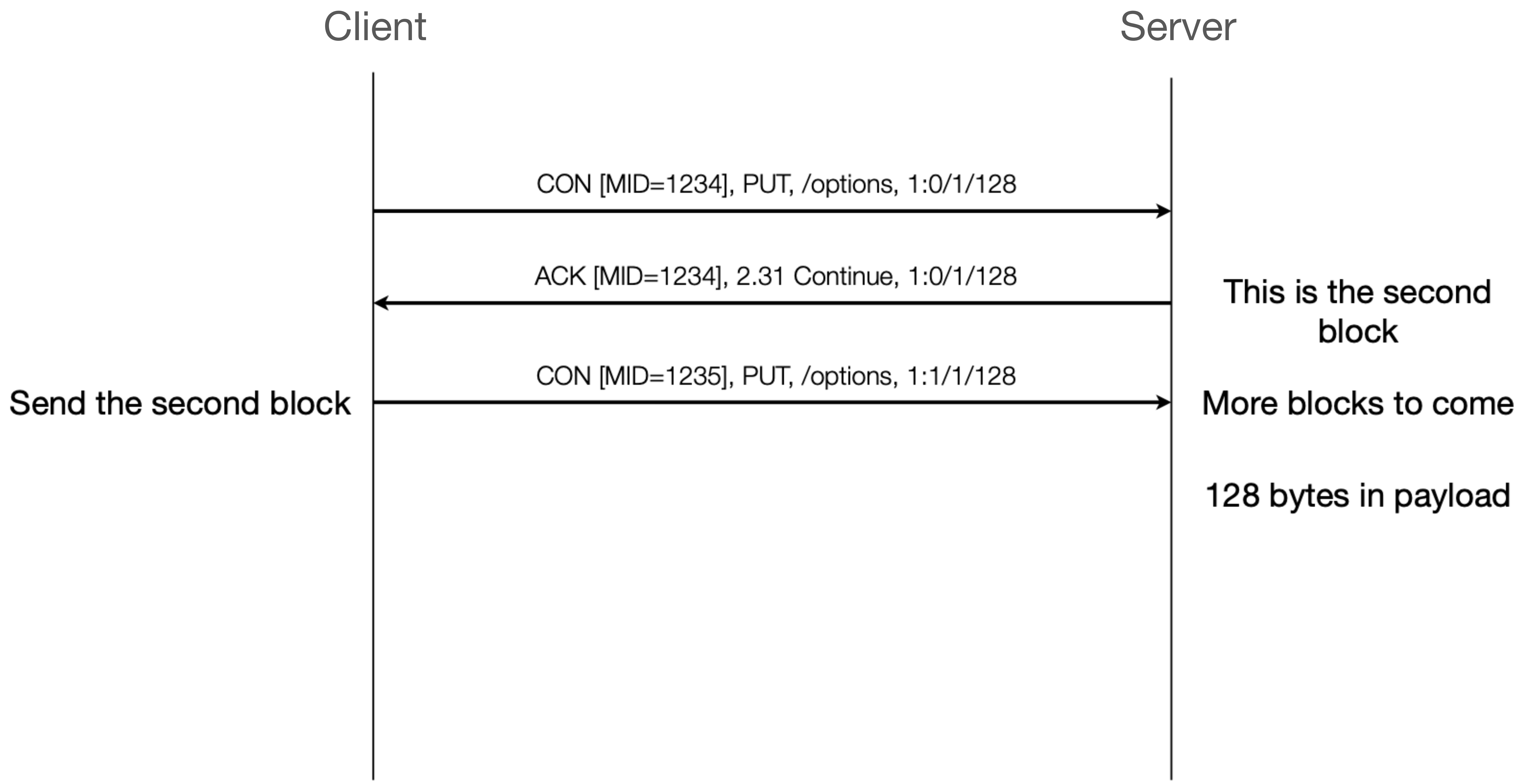
Block1 Option Example (Request Payload)


Client
Server
CON [MID=1234], PUT, /options, 1:0/1/128
ACK [MID=1234], 2.31 Continue, 1:0/1/128
Payload received...

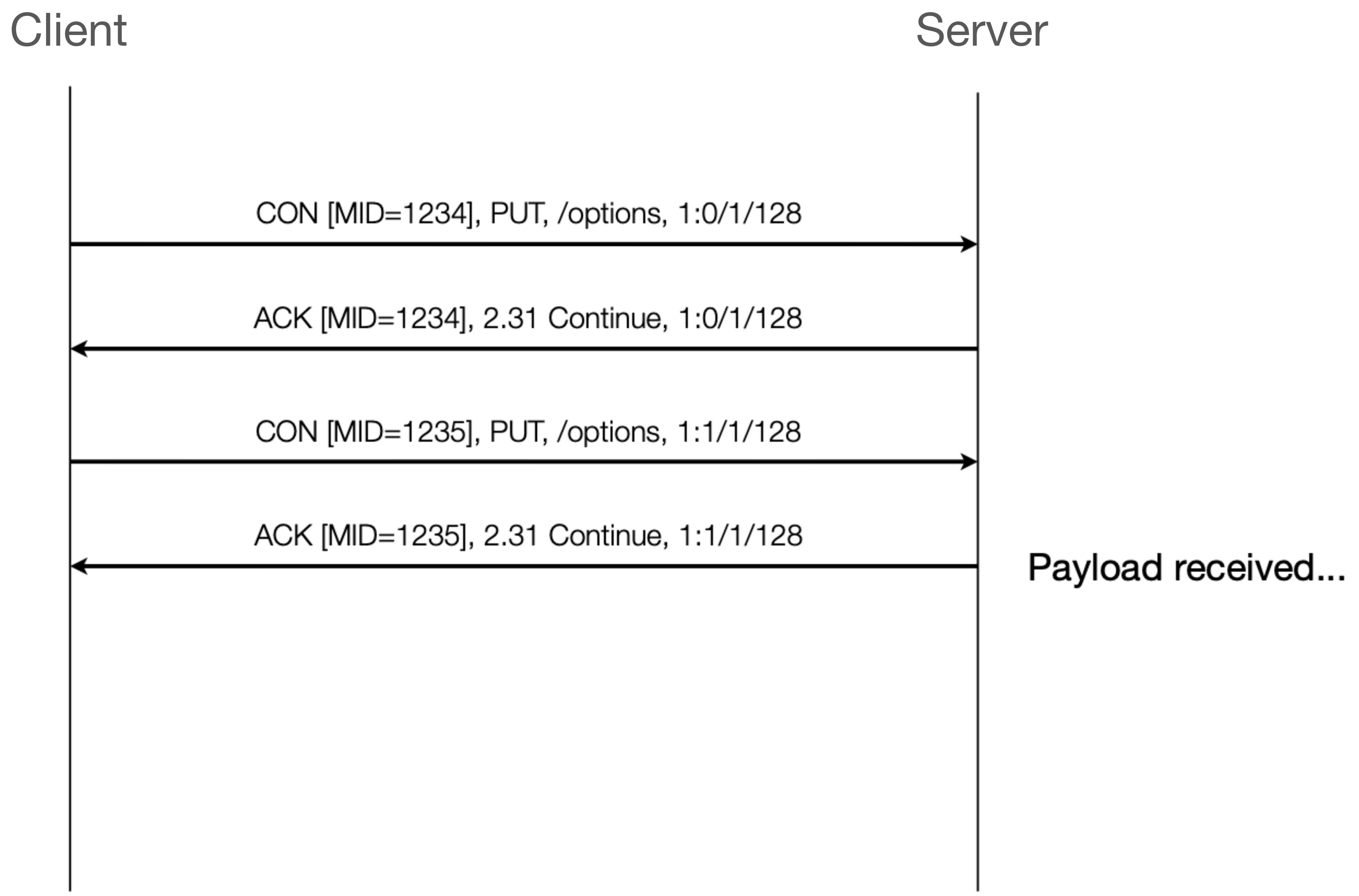
Block1 Option Example (Request Payload)


Client
Server
CON [MID=1234], PUT, /options, 1:0/1/128
ACK [MID=1234], 2.31 Continue, 1:0/1/128
The transfer of this block of the request body was successful
The server encourages sending further blocks

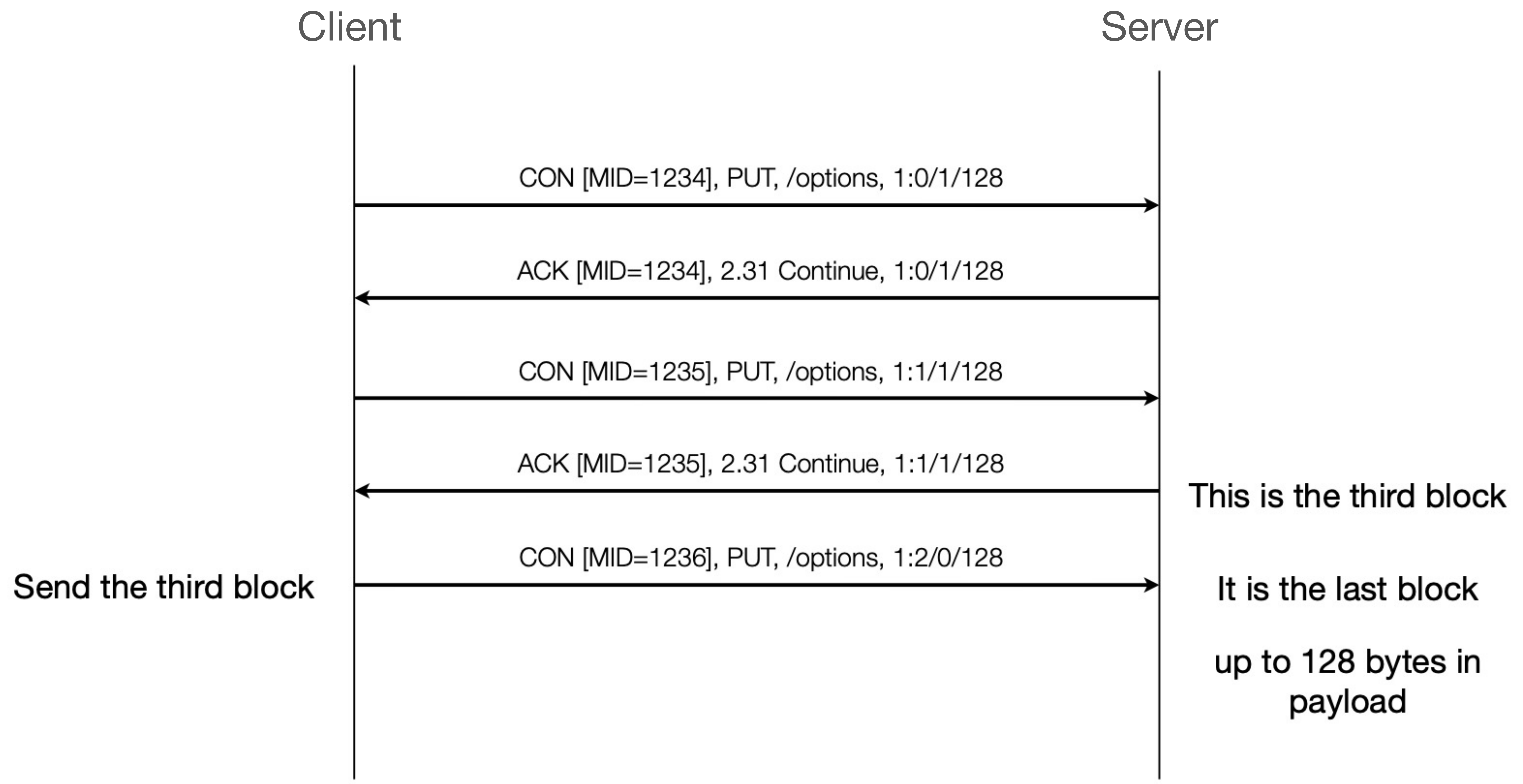
Block1 Option Example (Request Payload)


Client
Server
CON [MID=1234], PUT, /options, 1:0/1/128
ACK [MID=1234], 2.31 Continue, 1:0/1/128
CON [MID=1235], PUT, /options, 1:1/1/128
Send the second block
This is the second block
More blocks to come
128 bytes in payload

Block1 Option Example (Request Payload)

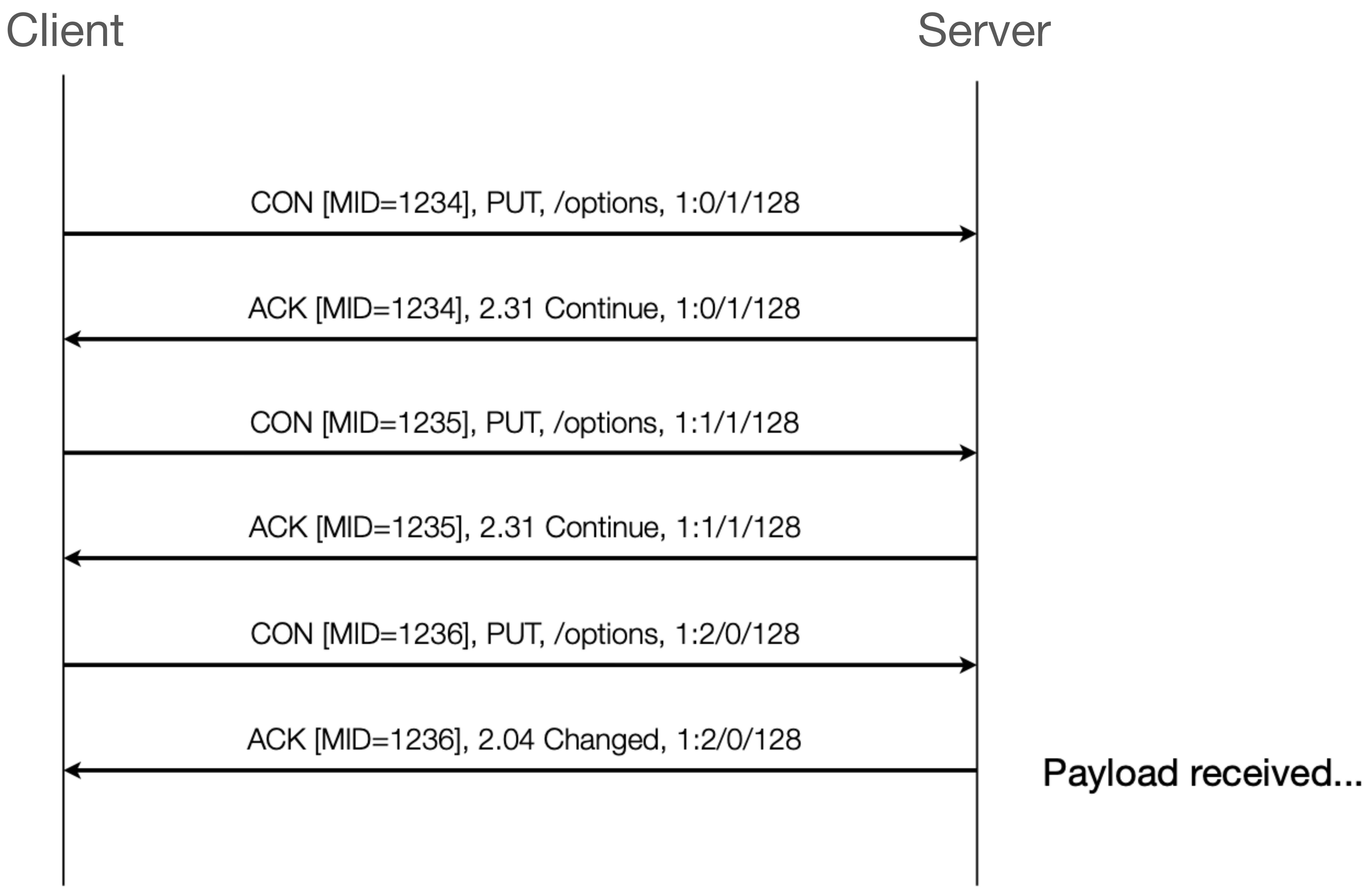

Client
Server
CON [MID=1234], PUT, /options, 1:0/1/128
ACK [MID=1234], 2.31 Continue, 1:0/1/128
CON [MID=1235], PUT, /options, 1:1/1/128
ACK [MID=1235], 2.31 Continue, 1:1/1/128
Payload received...

Block1 Option Example (Request Payload)



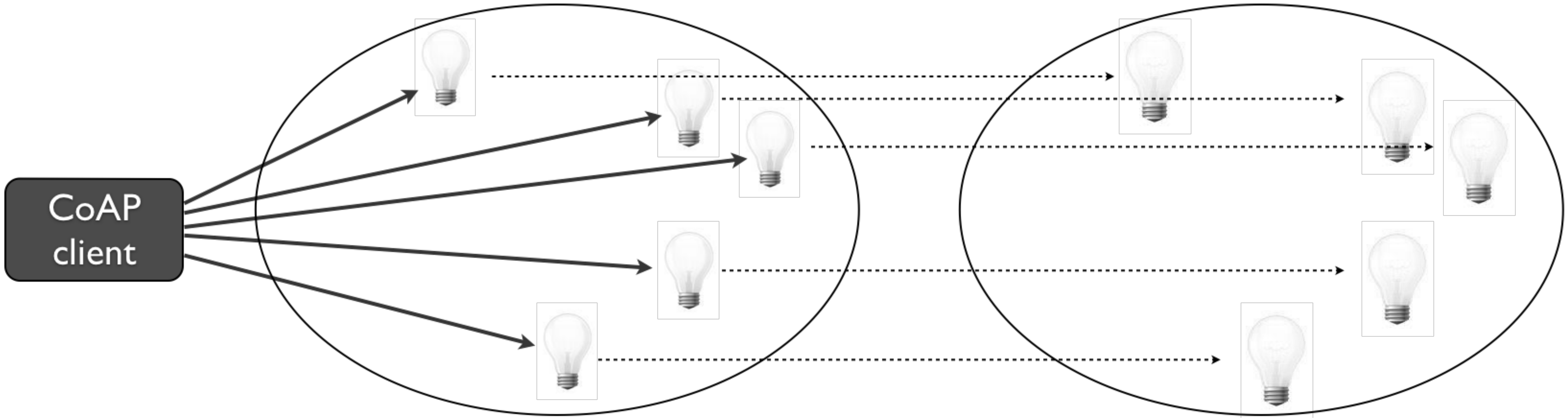
Client
Server
CON [MID=1234], PUT, /options, 1:0/1/128
ACK [MID=1234], 2.31 Continue, 1:0/1/128
CON [MID=1235], PUT, /options, 1:1/1/128
ACK [MID=1235], 2.31 Continue, 1:1/1/128
CON [MID=1236], PUT, /options, 1:2/0/128
Send the third block
This is the third block
It is the last block
up to 128 bytes in payload

Block1 Option Example (Request Payload)


Client
Server
CON [MID=1234], PUT, /options, 1:0/1/128
ACK [MID=1234], 2.31 Continue, 1:0/1/128
CON [MID=1235], PUT, /options, 1:1/1/128
ACK [MID=1235], 2.31 Continue, 1:1/1/128
CON [MID=1236], PUT, /options, 1:2/0/128
ACK [MID=1236], 2.04 Changed, 1:2/0/128
Payload received...

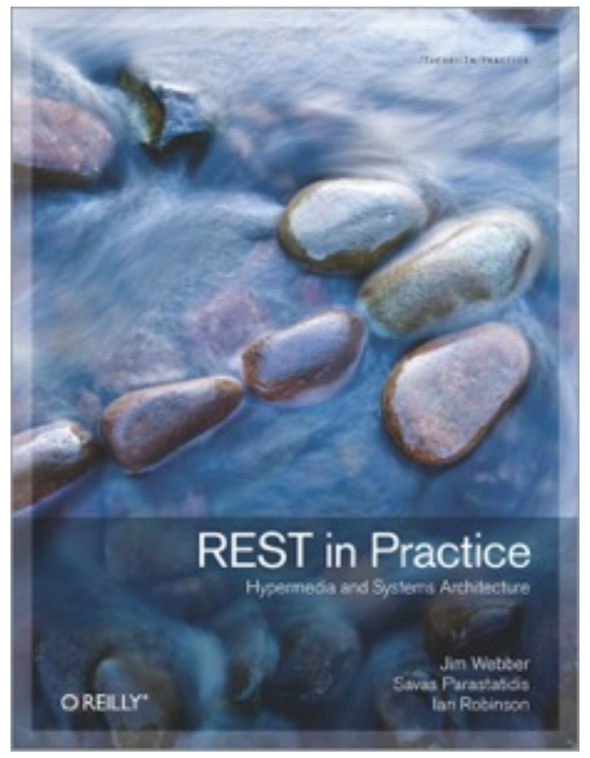
Group Communications

- CoAP supports multicast to target multiple endpoints with a single request
- Example: *POST All-Devices.floor1.west.bldg6.example.com/status/lights*
- draft-ietf-core-groupcomm-25 defines mechanisms to manage CoAP groups



References

1. Jim Webber, Savas Parastatidis, Ian Robinson, “REST in Practice,” O'Reilly Media, September 2010 [http://shop.oreilly.com/product/9780596805838.do]
2. “Internet of Things Applications - From Research and Innovation to Market Deployment”, edited by O. Vermesan and P. Friess, pp. 287-313, The River Publishers Series in Communications, Alborg, Denmark, 2014. ISBN: 9788793102941 [http://riverpublishers.com/view_details.php?book_id=245]
3. Matthias Kovatsch, “CoAP: The Web-based Application-layer Protocol for the Internet of Things,” Lecture COLLECT at the Internet of Things and Smart Cities Ph.D. School 2014, Sep. 10th, 2014, Lerici (SP), Italy [http://phdschool.tlc.unipr.it/iot/2014/index.php/welcome/downloads.html]
4. * R. T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” PhD thesis, University of California, 2000. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm



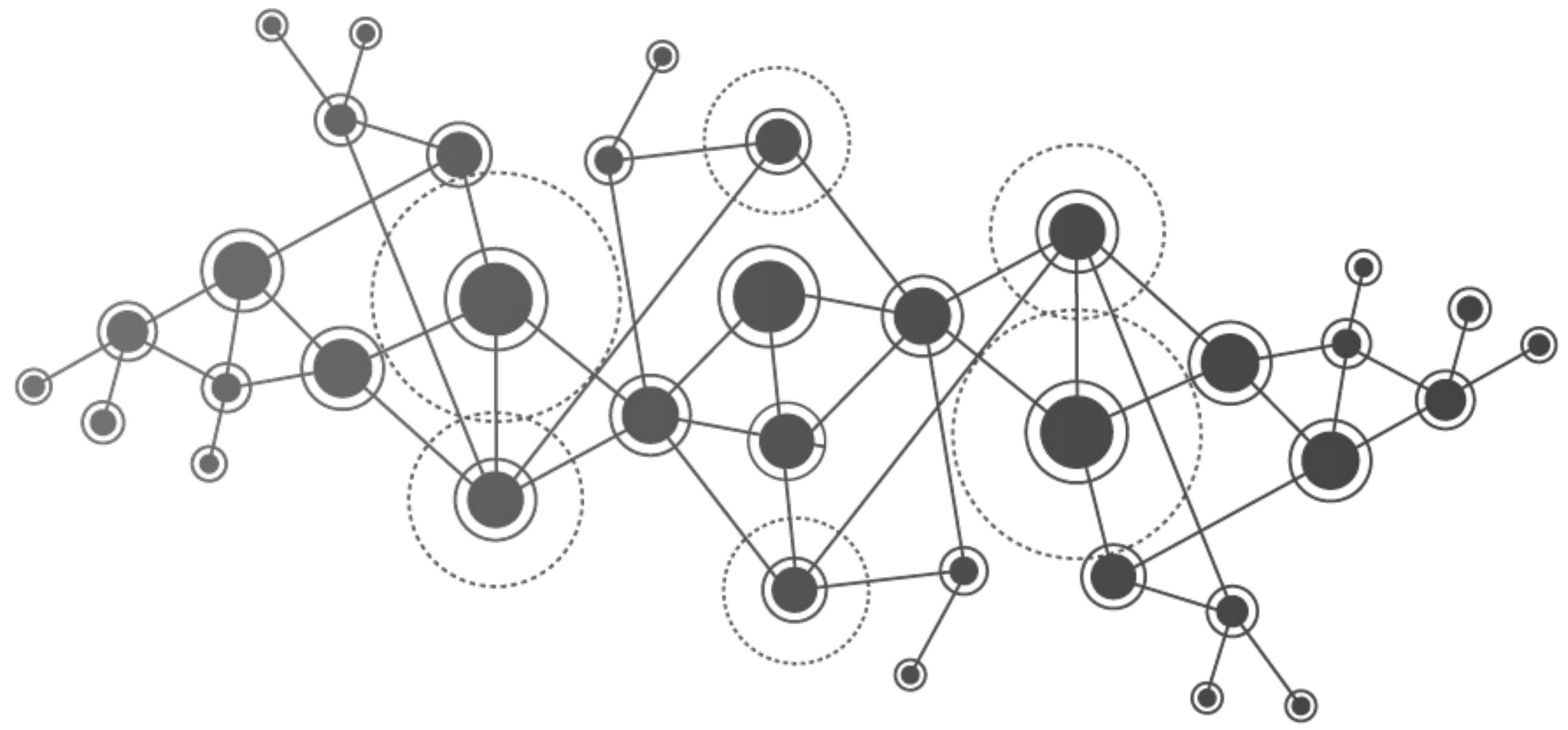
River Publishers Series in Communication
Internet of Things – From Research and Innovation to Market Deployment
Editors
Ovidiu Vermesan
Peter Friess
River Publishers

References

5. IETF CoRE Working Group Documents (https://tools.ietf.org/wg/core/)
6. Z. Shelby, K. Hartke, and C. Bormann, The Constrained Application Protocol (CoAP), RFC 7252, IETF, June 2014
7. Z. Shelby, Constrained RESTful Environments (CoRE) Link Format, RFC 6690, IETF, August 2012
8. K. Hartke, Observing Resources in CoAP, RFC 7641, IETF, September, 2015
9. C. Bormann and Z. Shelby, Block-wise transfers in CoAP, I-d draft-ietf-core-block-19, IETF, March 21, 2016
10. Z. Shelby and C. Bormann, CoRE Resource Directory, I-d draft-ietf-core-resource-directory-07, IETF, March 21, 2016
11. Z. Shelby and M. Vial, CoRE Interfaces, I-d draft-ietf-core-interfaces-04, IETF, October 10, 2015
12. A. Castellani, S. Loreto, A. Rahman, T. Fossati and E. Dijk, Guidelines for HTTP-CoAP Mapping Implementations, I-d draft-ietf-core-http-mapping-09, IETF, April 6, 2016
13. CoAP - Z. Shelby - https://www.iab.org/wp-content/IAB-uploads/2011/04/Shelby.pdf

UNIMORE

UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA



Intelligent Internet of Things

IoT Application Protocol - CoAP

Prof. Marco Picone

A.A 2023/2024