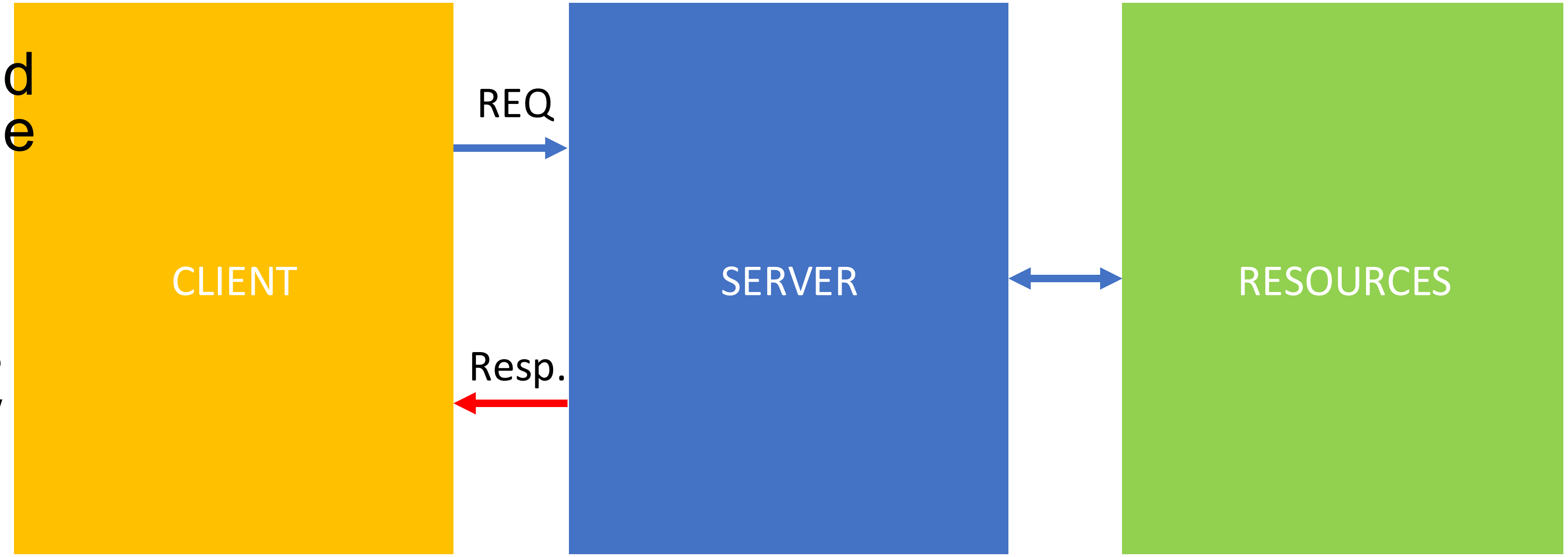
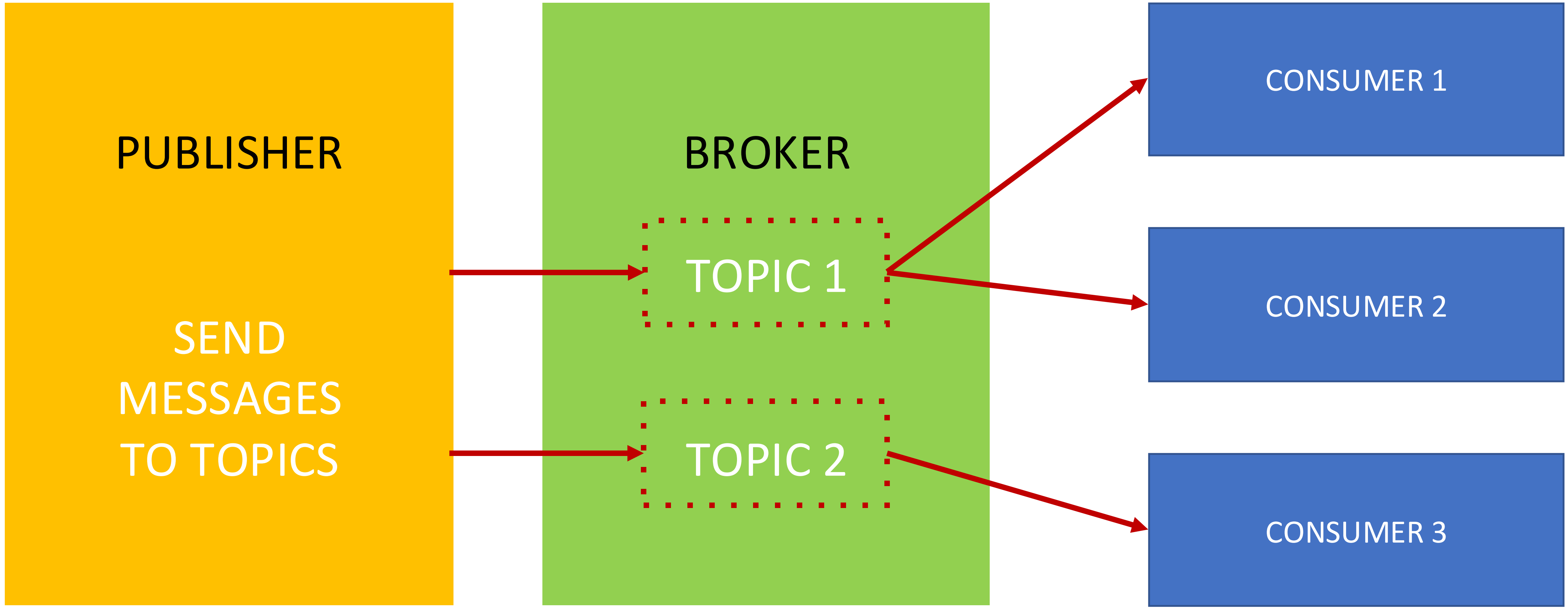
Communication Models of IoT

Client-Server Model

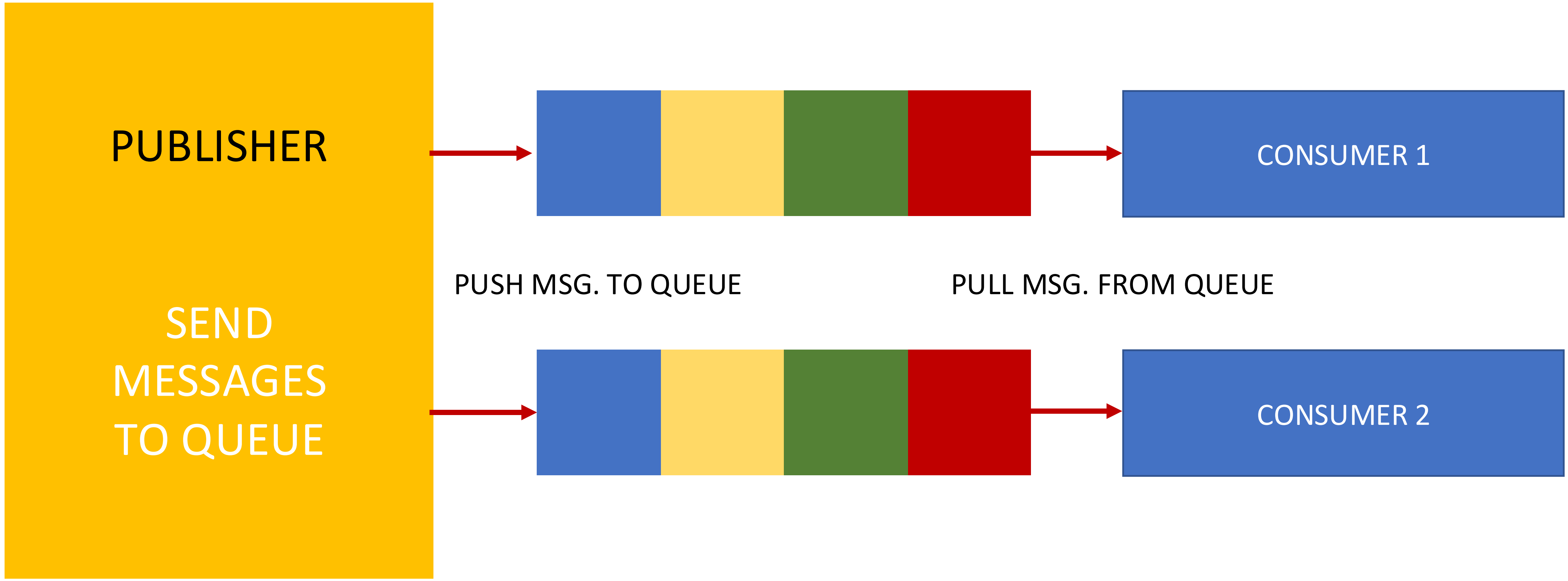
- Client-server is a communication model in which the client sends requests to the server and the server responds to the requests.
- When the server receives a request, it decides how to respond, fetches the data, retrieves resource representations, prepares the response, and then sends the response to the client.



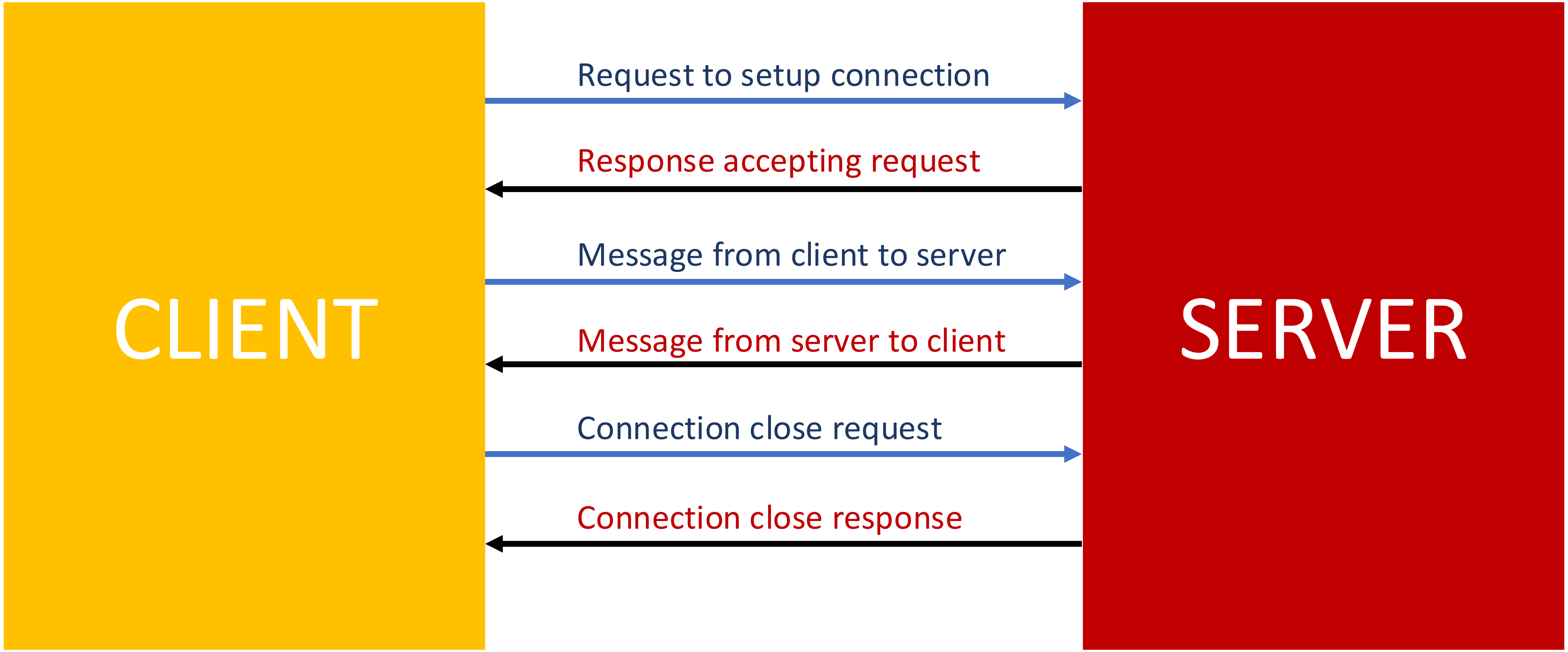
Publish-Subscribe Model



Push-Pull Model



Exclusive Pair Model



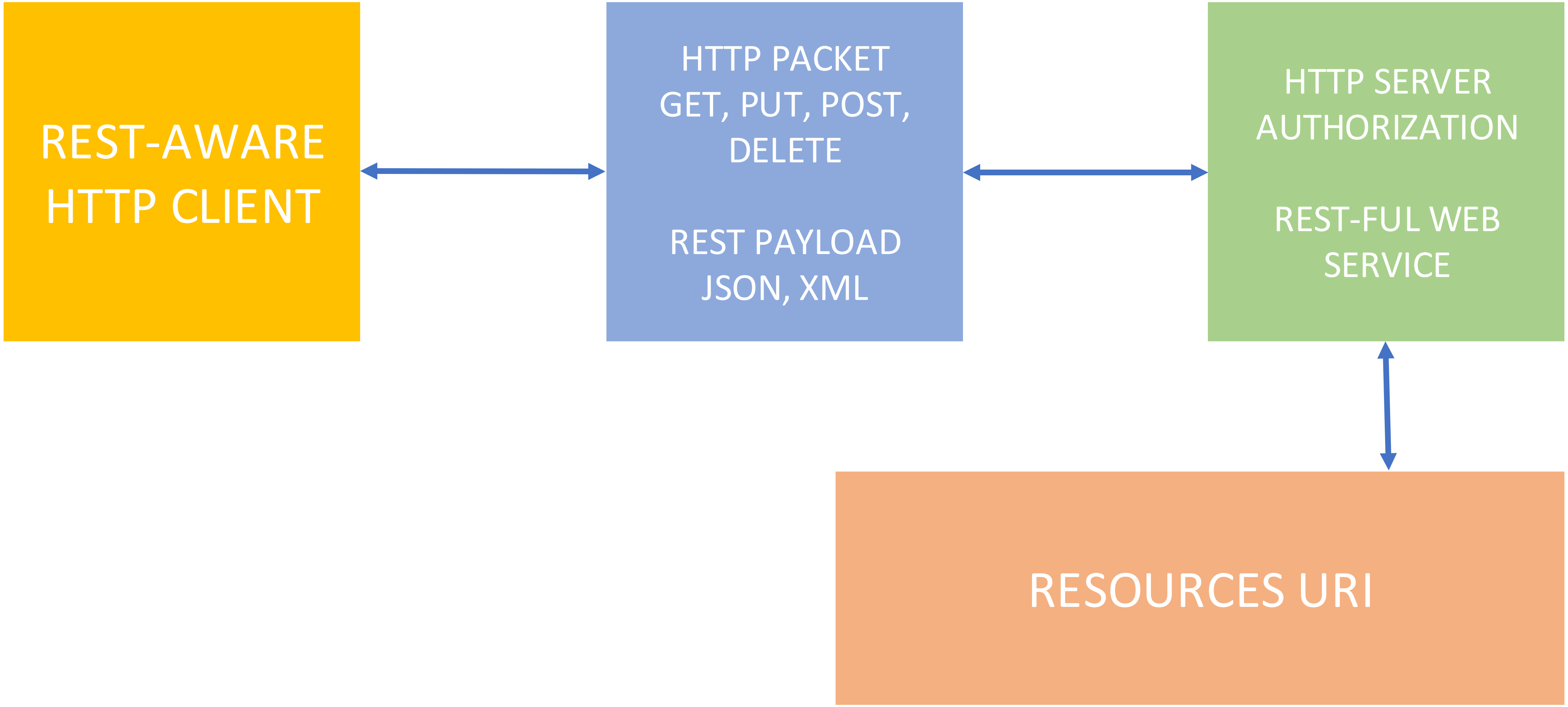
IoT Communication API

REST Communication APIs

- REpresentational State Transfer
- REST API is a way for two computer systems to communicate over HTTP in a similar way to web browsers and servers
- Client Server considerations
  - Client does not care about how data is stored at the server
  - Server does not care about the user interface at the client
- Stateless - the client request should contain all the information necessary to respond to a request
- Cache-able
- Layered - requesting client need not know whether it’s communicating with the actual server, a proxy, or any other intermediary
- Uniform interface
- Code on demand

https://www.sitepoint.com/developers-rest-api/

REST-Based APIs



RESTful Web Service Request

1. An Endpoint URL `https://mydomain/user/123?format=json`

2. The HTTP method

HTTP method	CRUD	Action
GET	read	returns requested data
POST	create	creates a new record
PUT or PATCH	update	updates an existing record
DELETE	delete	deletes an existing record

3. HTTP headers

4. Body Data

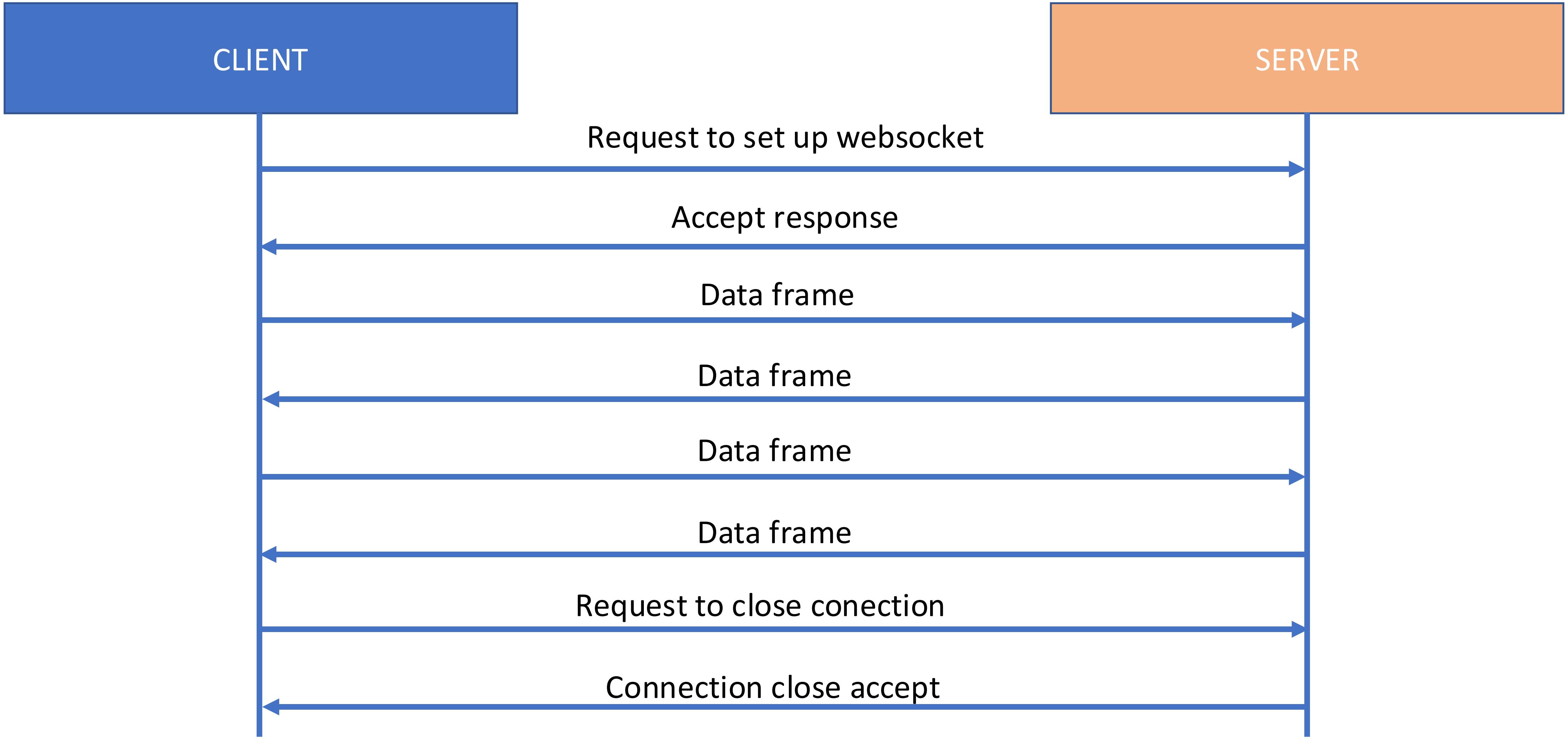
Examples

- a GET request to `/user/` returns a list of registered users on a system
- a POST request to `/user/123` creates a user with the ID 123 using the body data
- a PUT request to `/user/123` updates user 123 with the body data
- a GET request to `/user/123` returns the details of user 123
- a DELETE request to `/user/123` deletes user 123

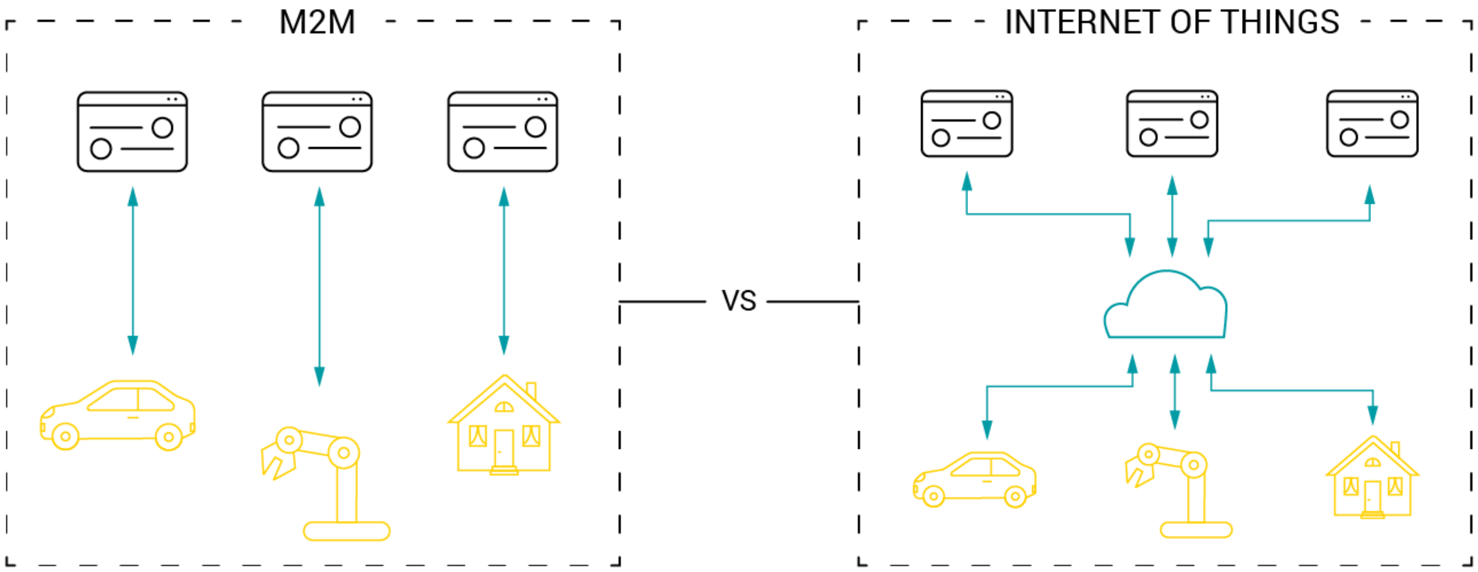
RESTful Web Service Reply

- **Response** payload can be whatever is practical: data, HTML, an image, an audio file, etc.
  - Typically JSON-encoded, but XML, CSV, simple strings, or any other format can be used
- An appropriate HTTP status code should also be set in the response header
  - 200 OK is most often used for successful requests
  - 201 Created may also be returned when a record is created
  - Errors should return an appropriate code (400 Bad Request, 404 Not Found, 401 Unauthorized)

WebSocket Based Communication



CoAP Protocol



M2M vs. IoT

M2M	IoT
Simple device-to-device communication usually within an embedded software at client site	Grand-scale projects and want-it-all approach
Isolated systems of devices using same standards	Integrates devices, data and applications across varying standards
Limited scalability options	Inherently more scalable
Wired or cellular network used for connectivity	Usually devices require active Internet connection
Extensive background of historical applications	State-of-the-art approach with roots in M2M

CoAP Features

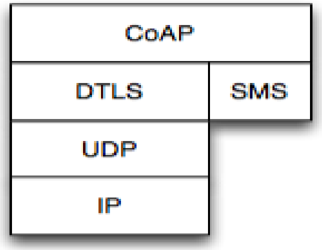
- Observe at new events happened on sensors or actuators.
- Device management and discoverability from external devices.
- Web protocol used in M2M with constrained requirements
- Asynchronous message exchange
- Low overhead and very simple to parse
- URI and content-type support
- Proxy and caching capabilities

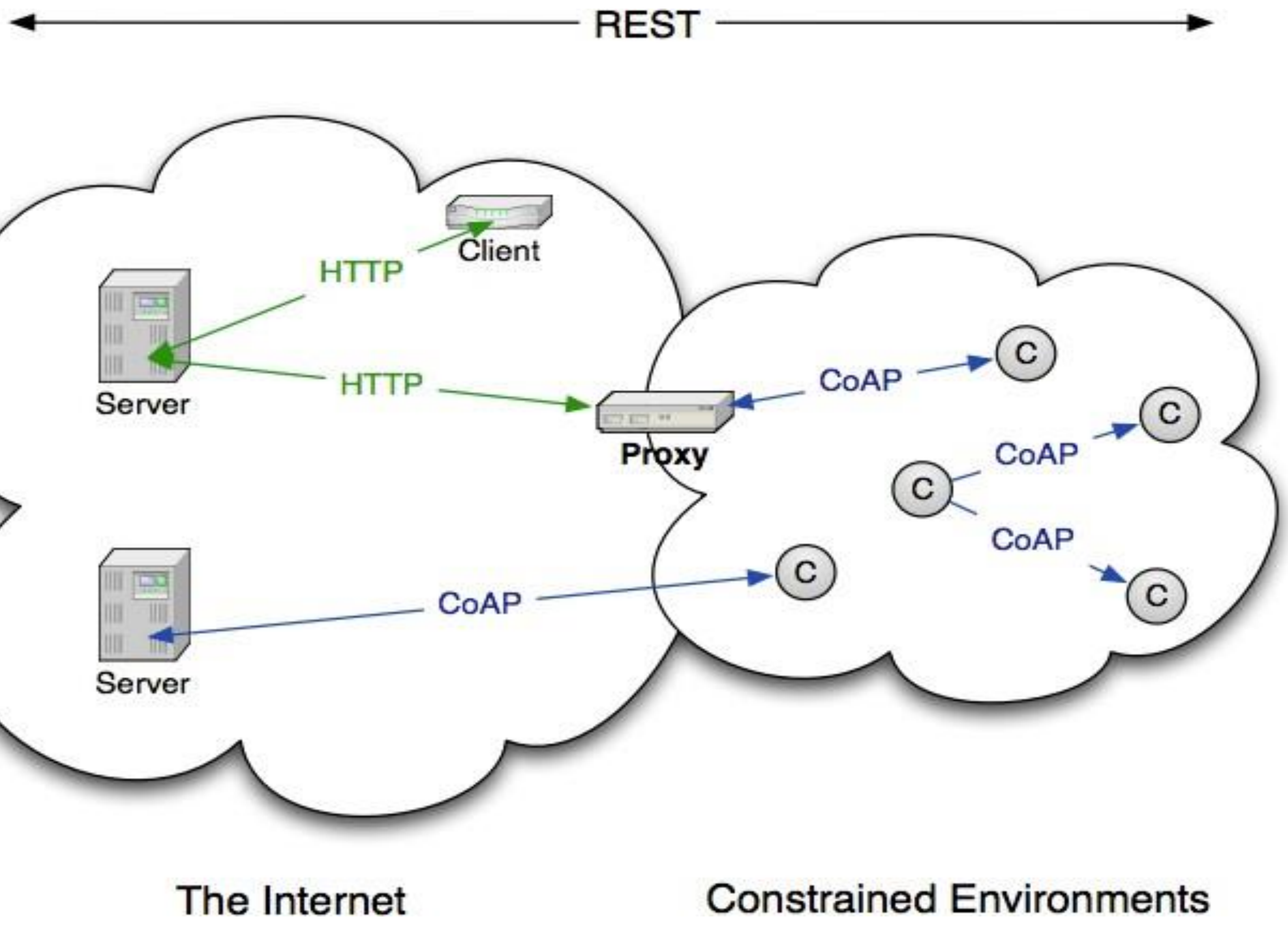
When to use CoAP?

- *Your hardware cannot run HTTP or TLS*
  - Running CoAP and DTLS can practically do the same as HTTP. If one is an expert on HTTP APIs, then the migration will be simple. You receive GET for reading and POST, PUT and DELETE for mutations and the security runs on DTLS.
- *Your hardware uses battery*
  - Running CoAP will improve the battery performance when compared with HTTP over TCP/IP. UDP saves some bandwidth and makes the protocol more efficient.
- *A subscription is necessary*
  - If one cannot run MQTT and HTTP polling is impossible then CoAP is a solution

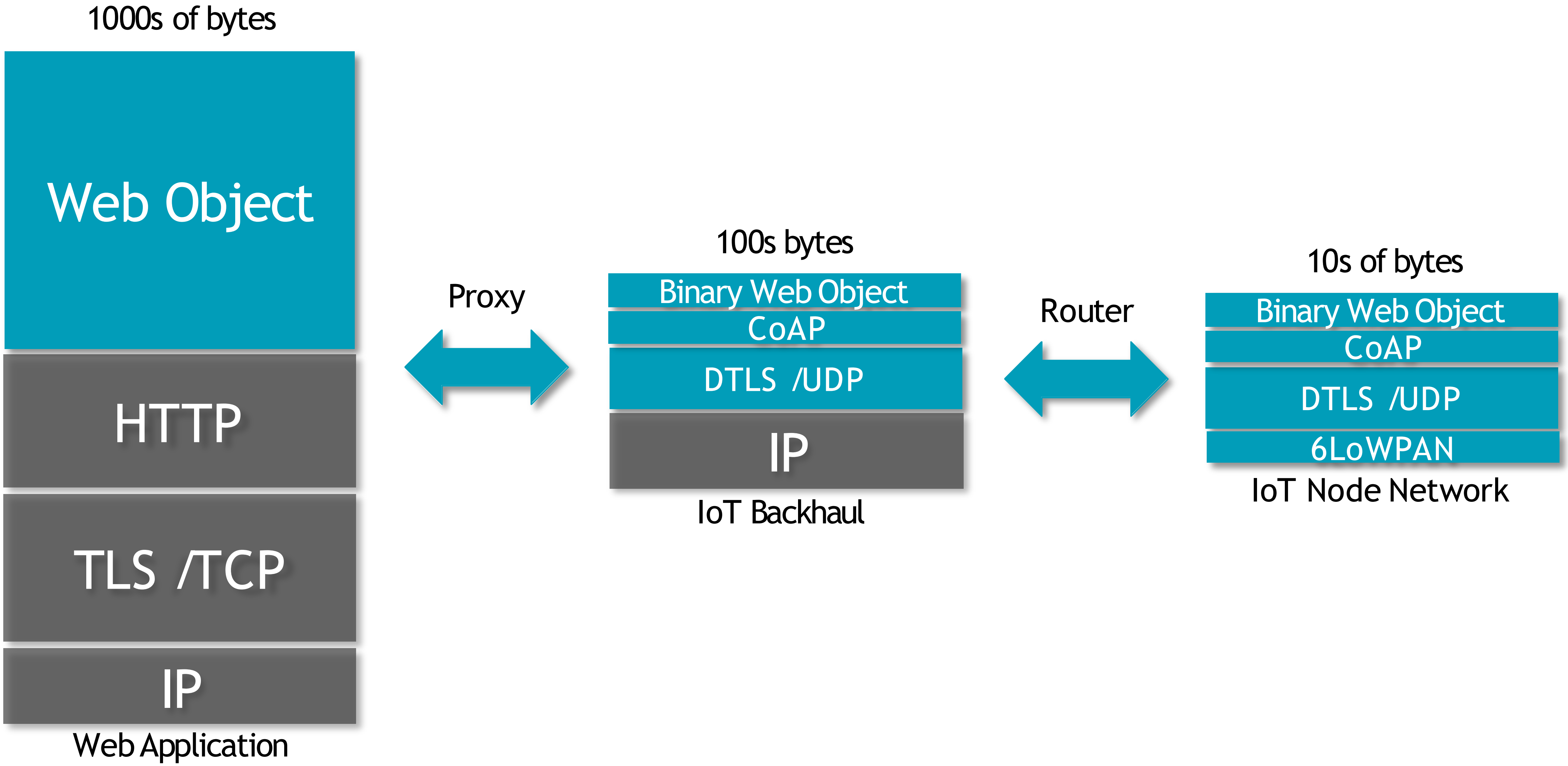
CoAP:TheWeb of Things Protocol

- Open IETF Standard
- Compact 4-byte Header
- UDP, SMS, (TCP) Support
- Strong DTLS Security
- Asynchronous Subscription
- Built-in Discovery



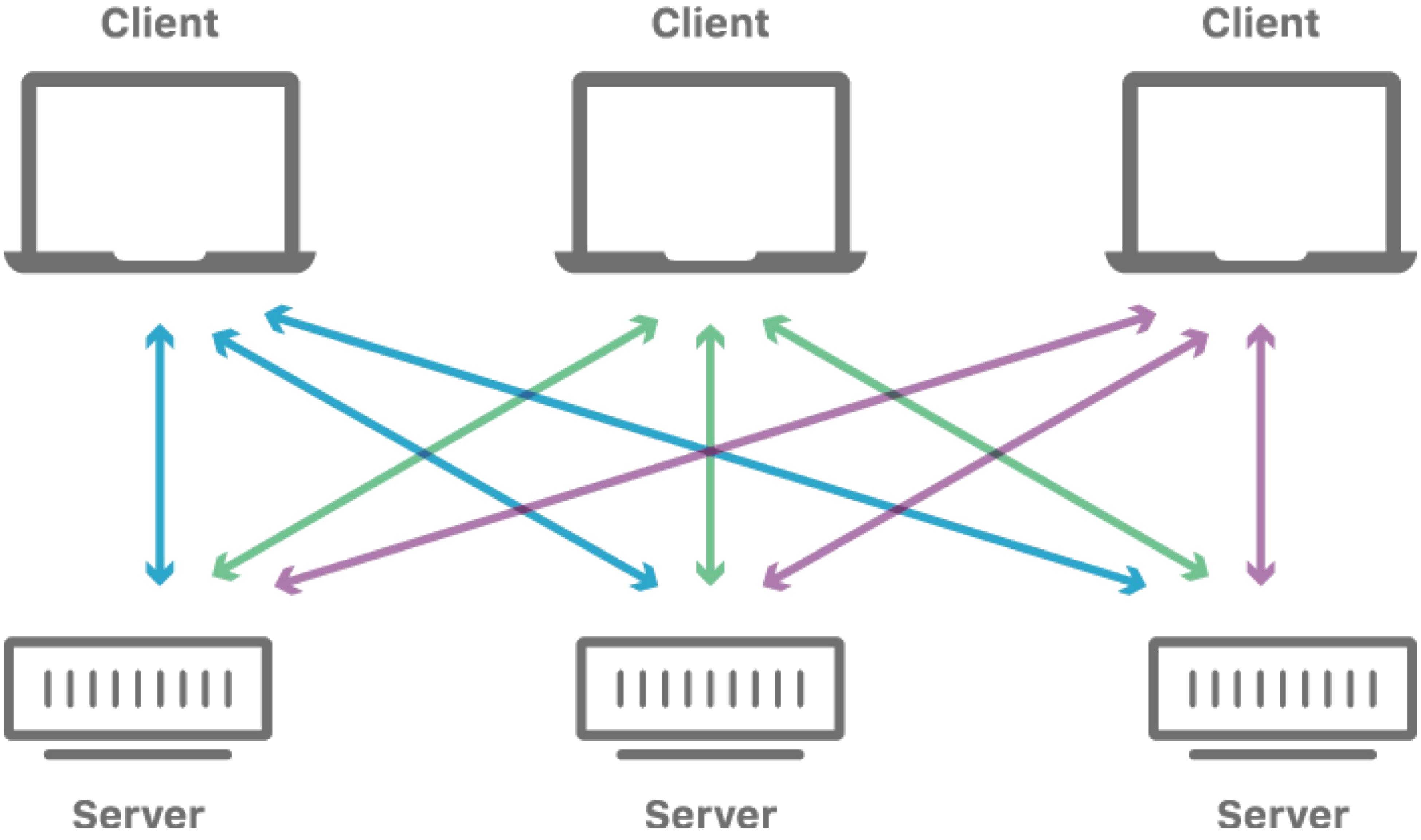


From Web Applications to IoT Nodes

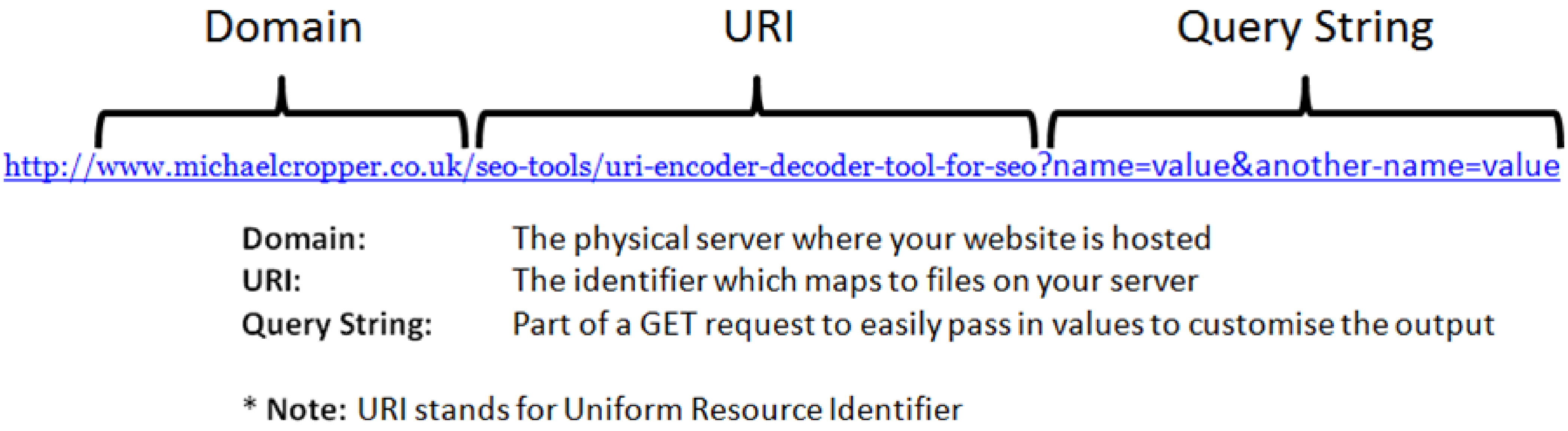


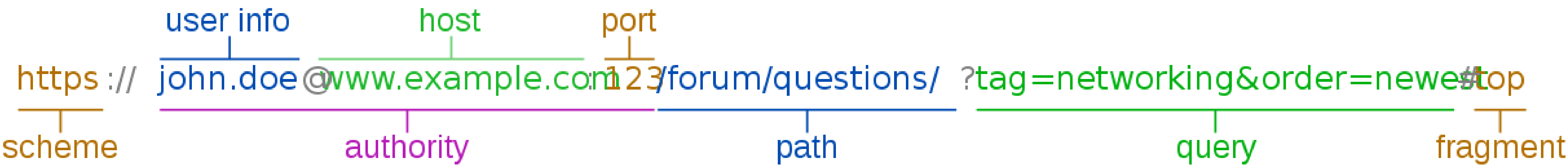
The Web and REST

Web Architecture

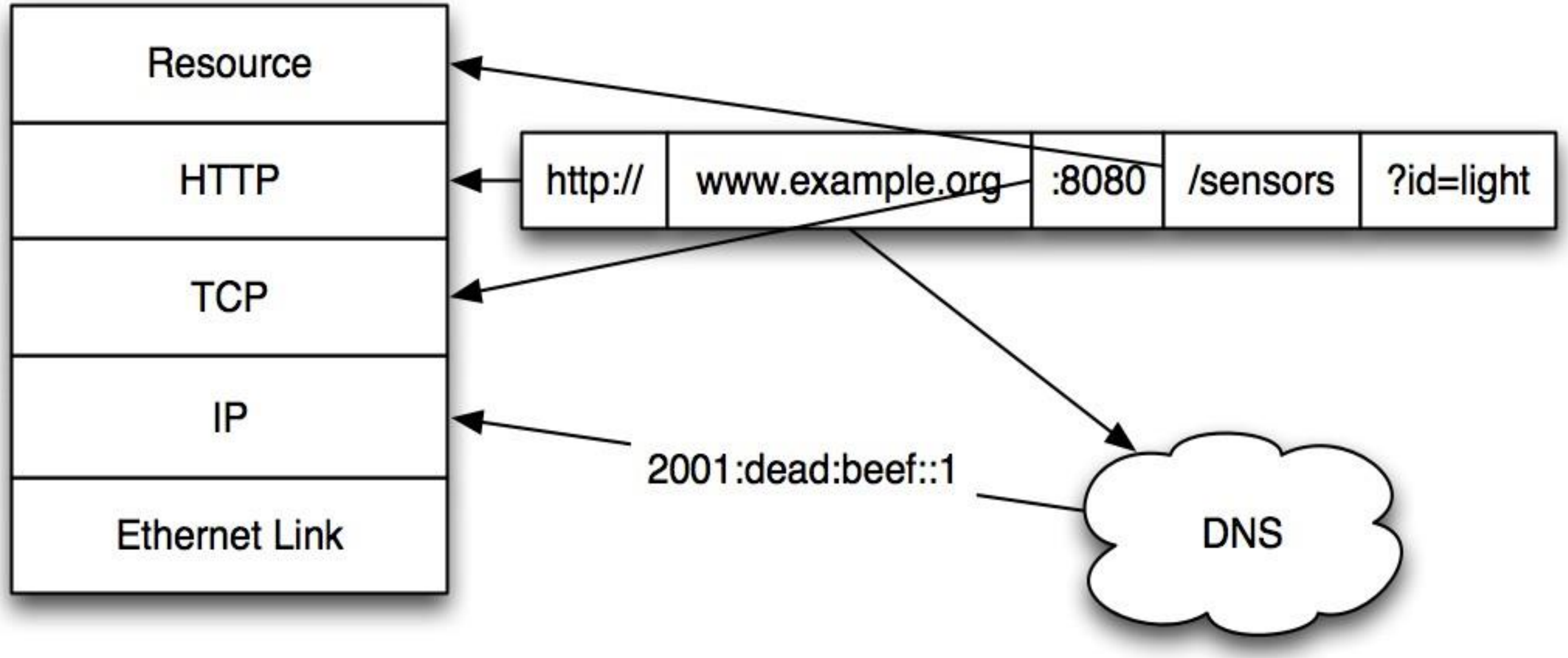


Web Naming

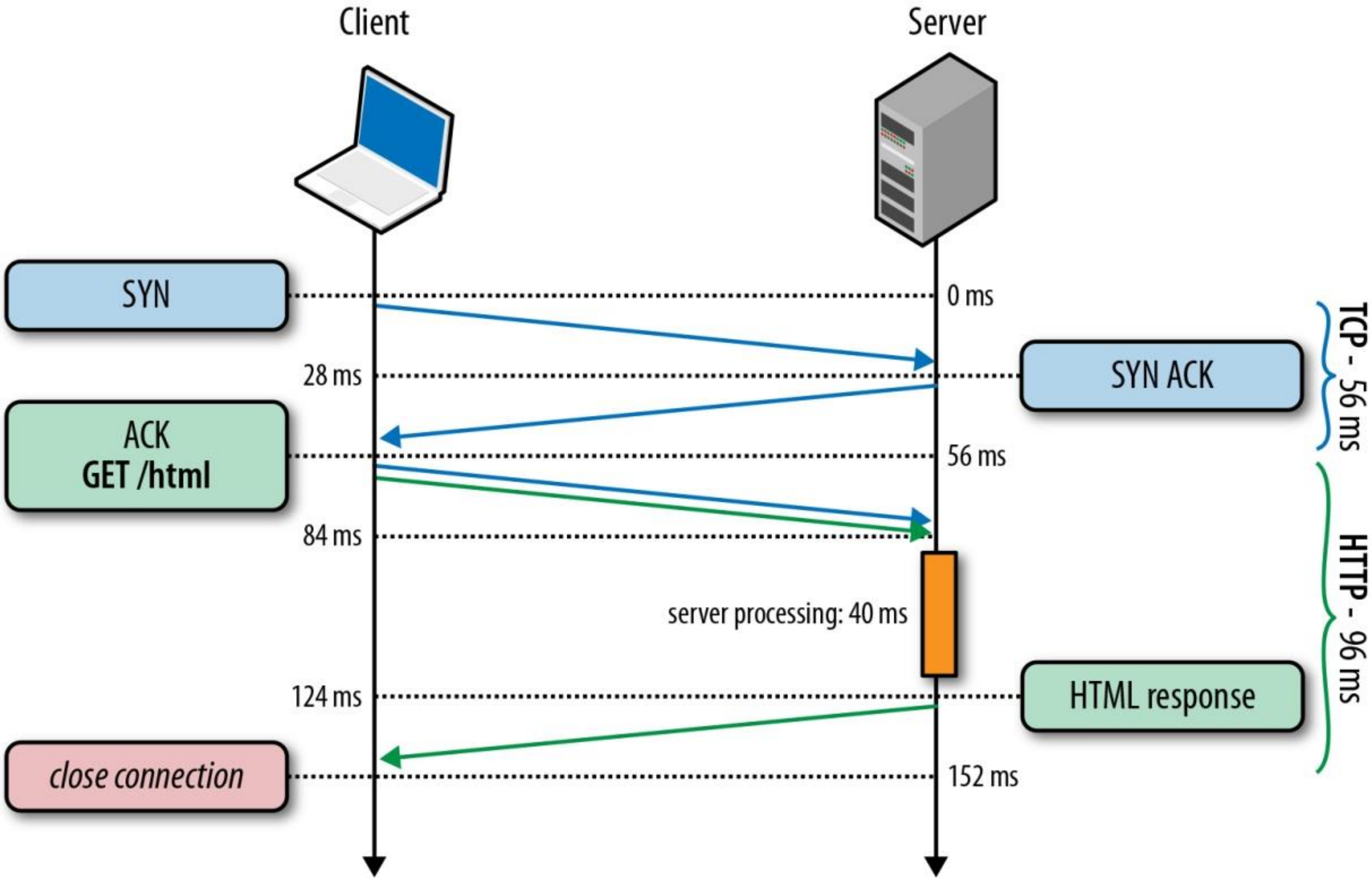

Domain
URI
Query String
http://www.michaelcropper.co.uk/seo-tools/uri-encoder-decoder-tool-for-seo?name=value&another-name=value
Domain: The physical server where your website is hosted
URI: The identifier which maps to files on your server
Query String: Part of a GET request to easily pass in values to customise the output
* Note: URI stands for Uniform Resource Identifier


user info
host
port
https :// john.doe @ www.example.com : 123 /forum/questions/ ?tag=networking&order=newest #top
scheme
authority
path
query
fragment

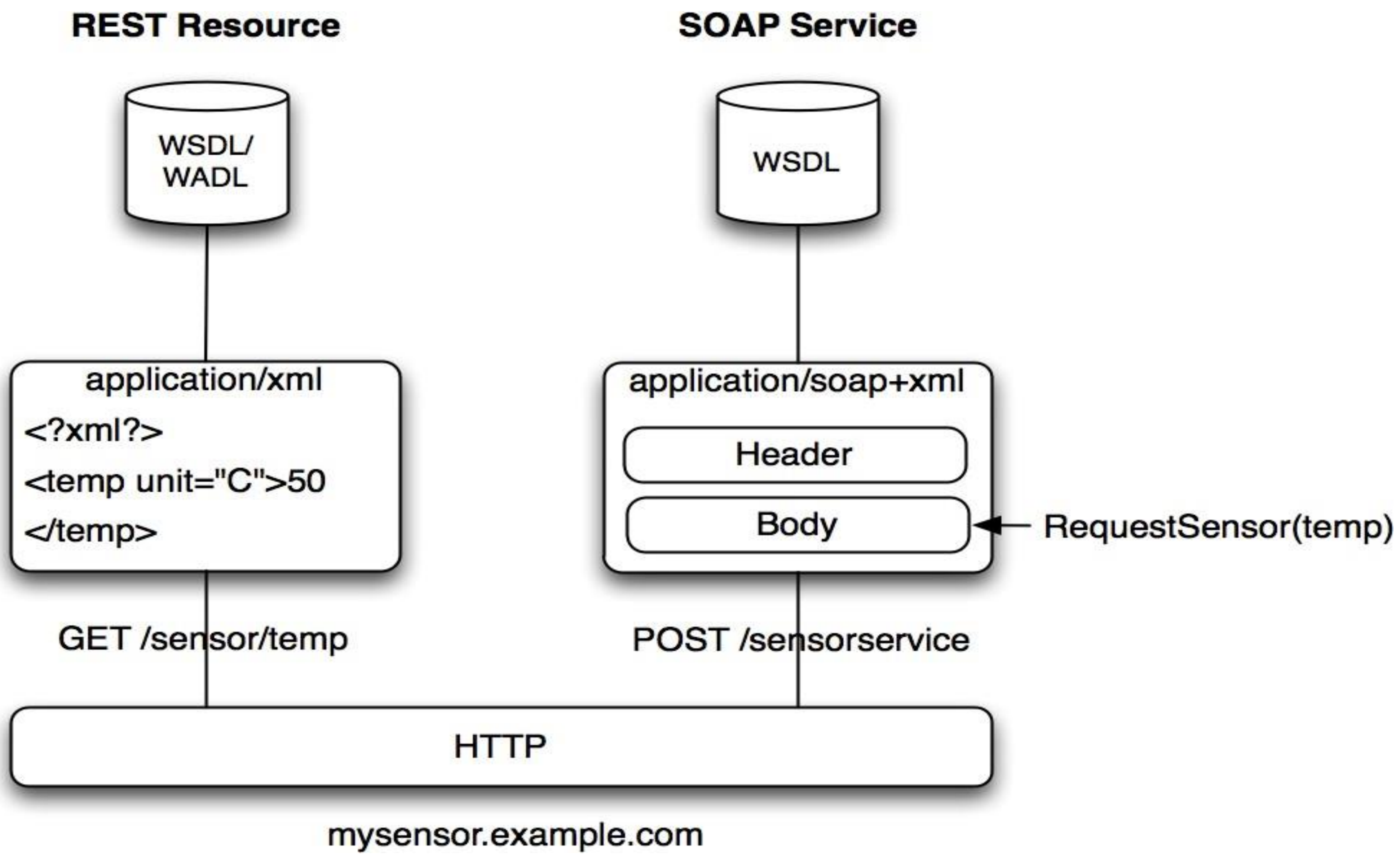
URL Resolution



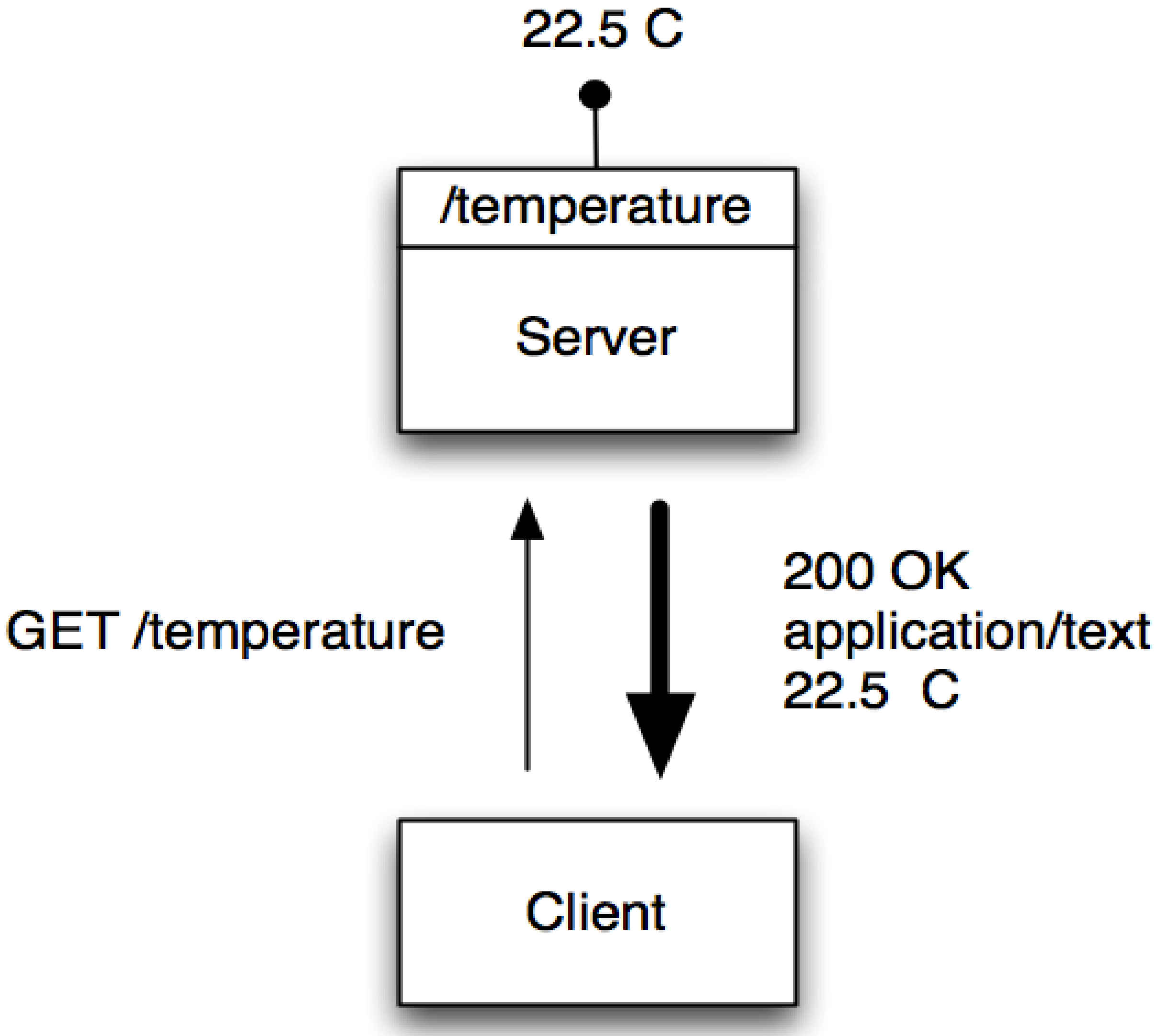
HTTP Request



Web Paradigms

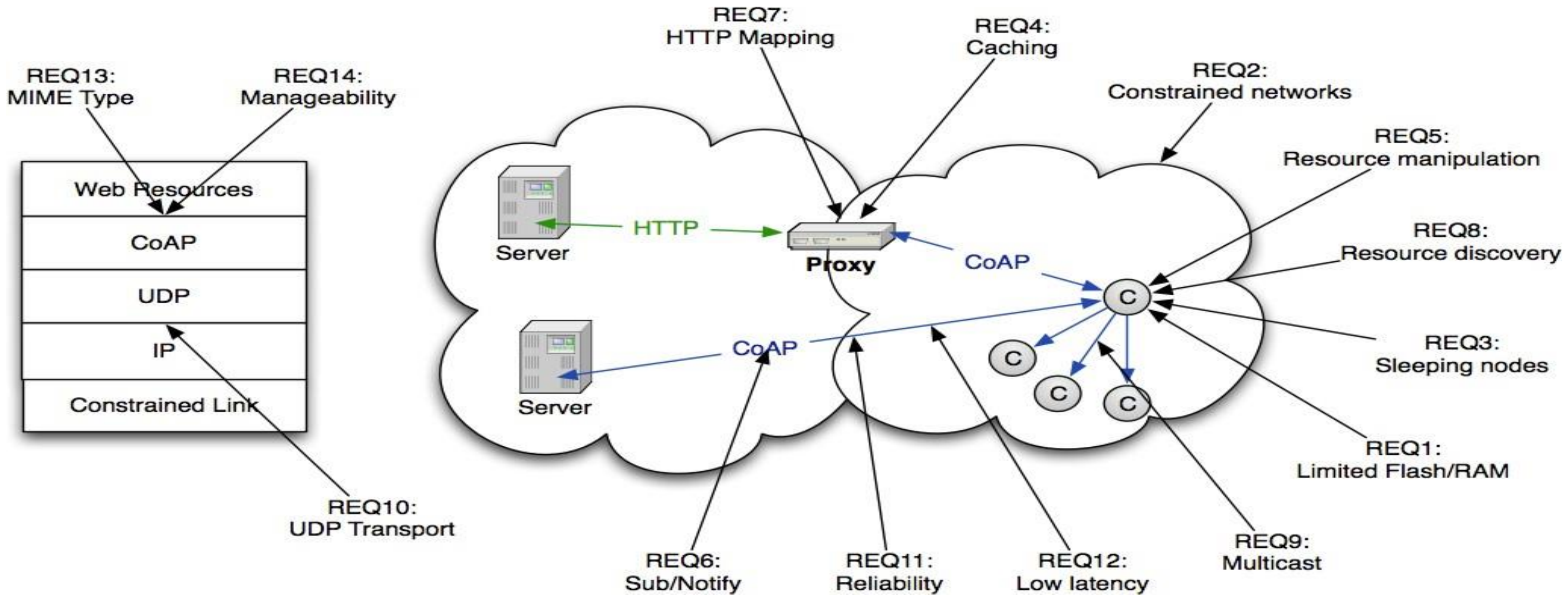


A REST Request

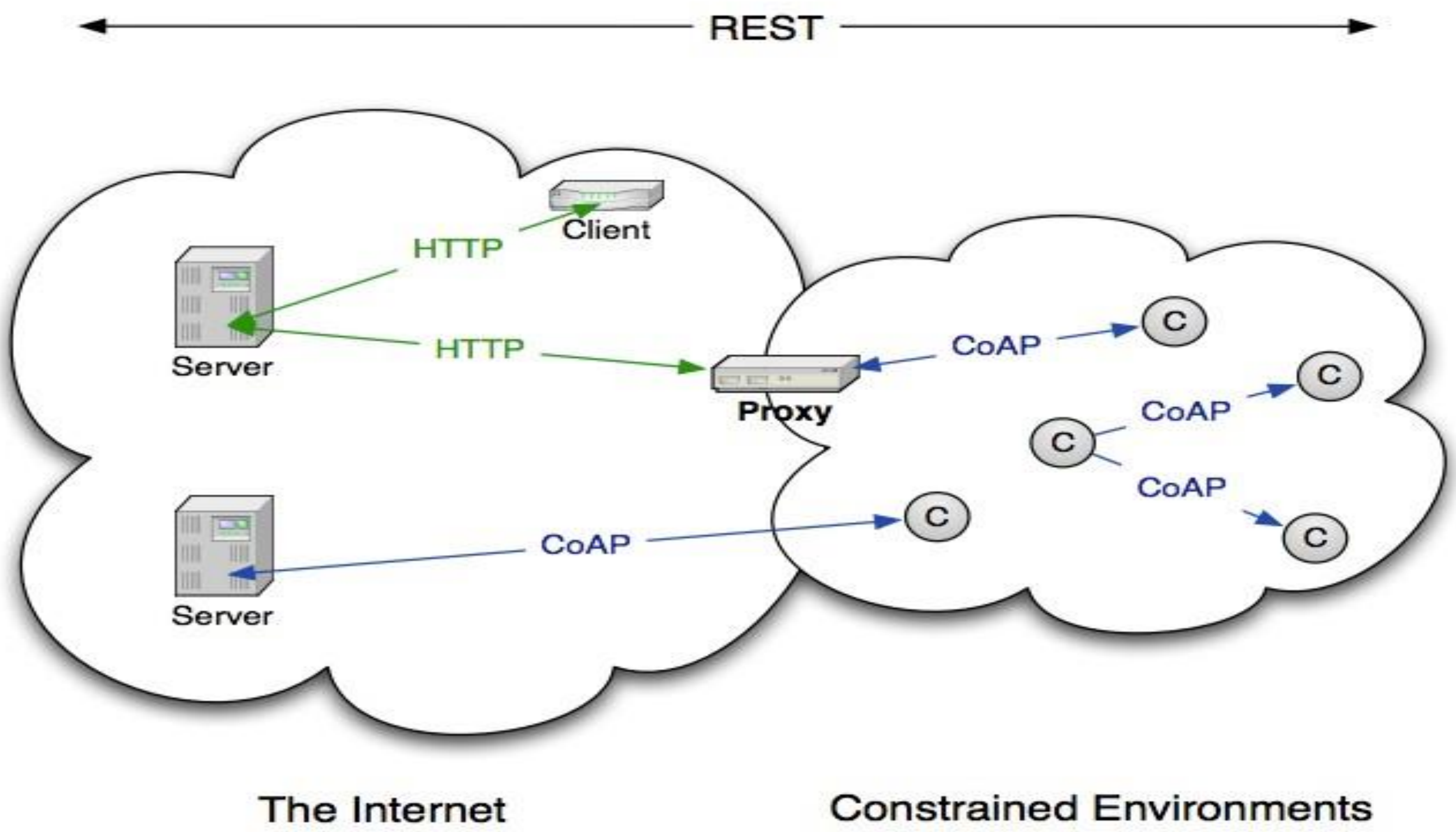


CoAP: Constrained Application Protocol

CoAP Design Requirements



The CoAP Architecture



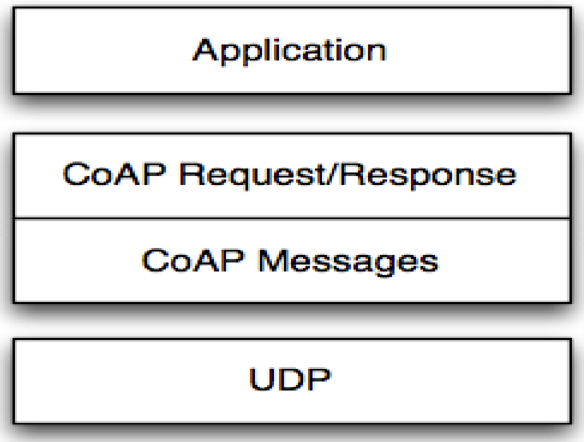
What CoAP is (and is not)

- CoAP is
  - A very efficient RESTful protocol
  - Ideal for constrained devices and networks
  - Specialized for M2M applications
  - Easy to proxy to/from HTTP
- CoAP is not
  - A general replacement for HTTP
  - HTTP compression
  - Restricted to isolated “automation” networks

CoAP Features

- Embedded web transfer protocol (coap://)
- Asynchronous transaction model
- UDP binding with reliability and multicast support
- GET, POST, PUT, DELETE methods
- URI support
- Small, simple 4 byte header
- DTLS based PSK, RPK and Certificate security
- Subset of MIME types and HTTP response codes
- Built-in discovery
- Optional observation and block transfer

Transaction Model



Transport

CoAP currently defines:

UDP binding with DTLS security

CoAP over SMS or TCP possible

Base Messaging

Simple message exchange between endpoints

Confirmable or Non-Confirmable Message answered by Acknowledgement or Reset Message

REST Semantics

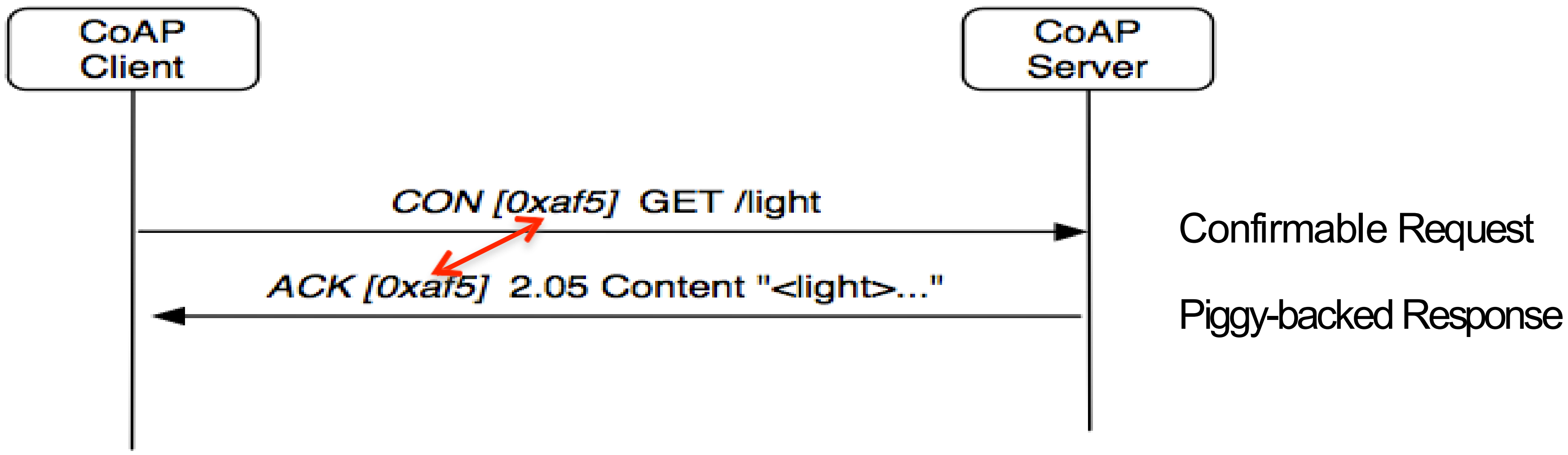
REST Request/Response piggybacked on CoAP Messages

Method, Response Code and Options (URI, content-type etc.)

Message Header (4 bytes)

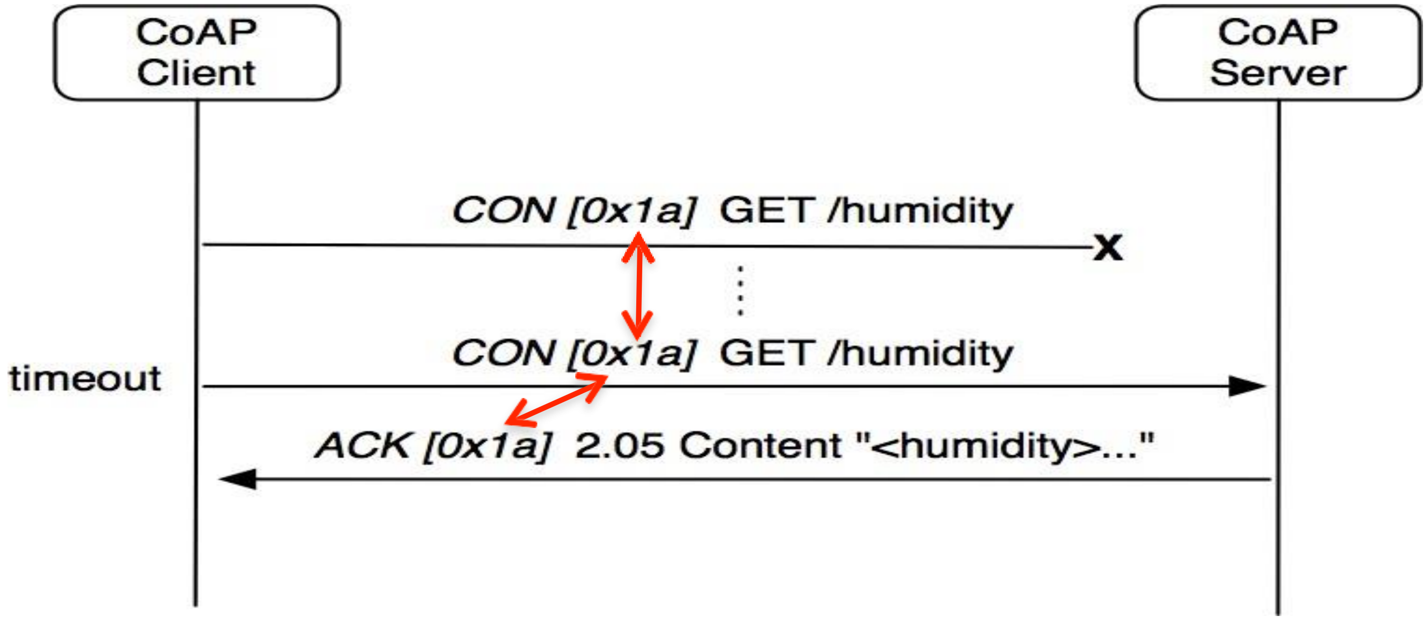
Ver: It is a 2 bit unsigned integer indicating the version
T: it is a 2 bit unsigned integer indicating the message type: 0 confirmable, 1 non-confirmable
TKL: Token Length is the token 4 bit length
Code: It is the code response (8 bit length)
Message ID: It is the message ID expressed with 16 bit

Request Example

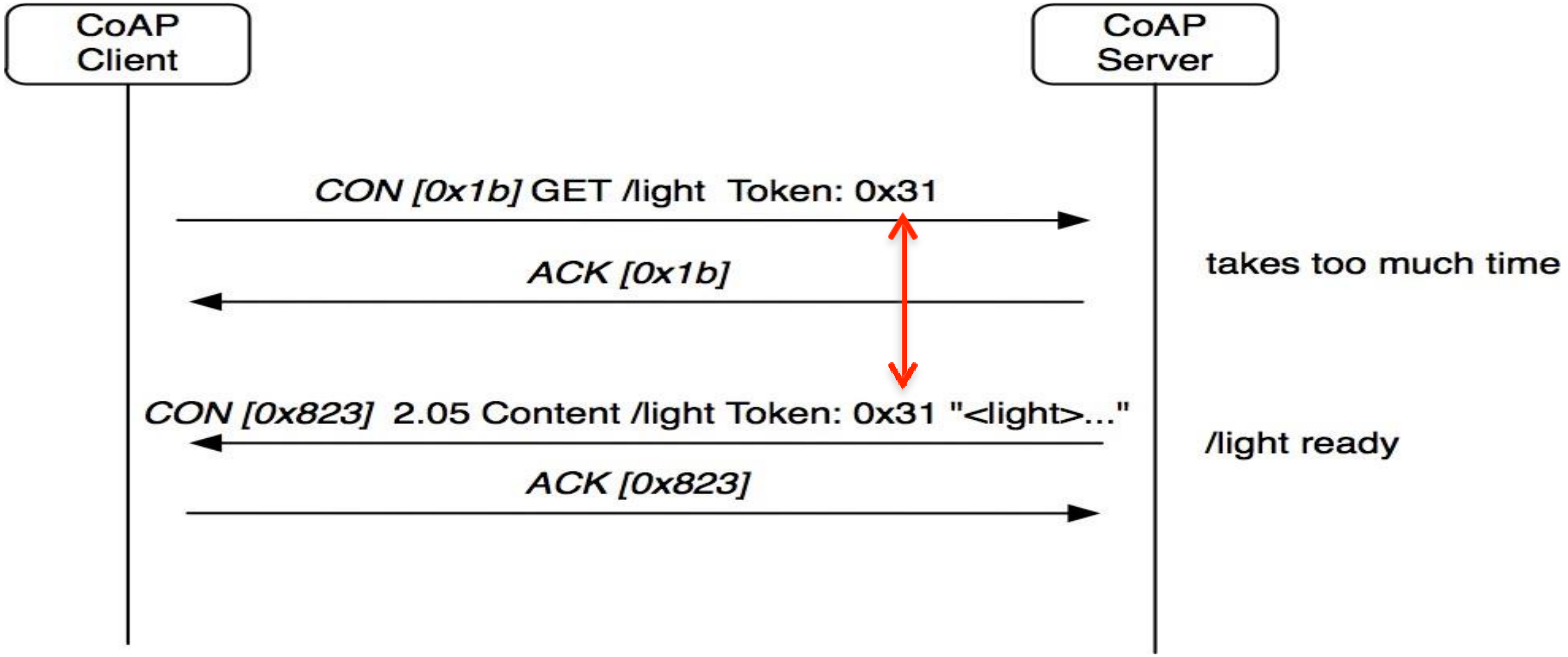


In the above diagram, you can see communication but If the server has troubles managing the incoming request it can send back a Rest message (RST) instead of the Acknowledge message (ACK).

Dealing with Packet Loss



Separate Response



If the server can’t answer to the request, then server sends an Acknowledge with an empty response. As soon as the response is available then the server sends a new Confirmable message to the client containing the response. At this point the client sends back an Acknowledge message.

Bits and bytes...

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

 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   |   4   |       "temp" (4 B) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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

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

Caching

CoAP includes a simple caching model

- Cacheability determined by response code
- An option number mask determines if it is a cache key

Freshness model

- Max-Age option indicates cache lifetime

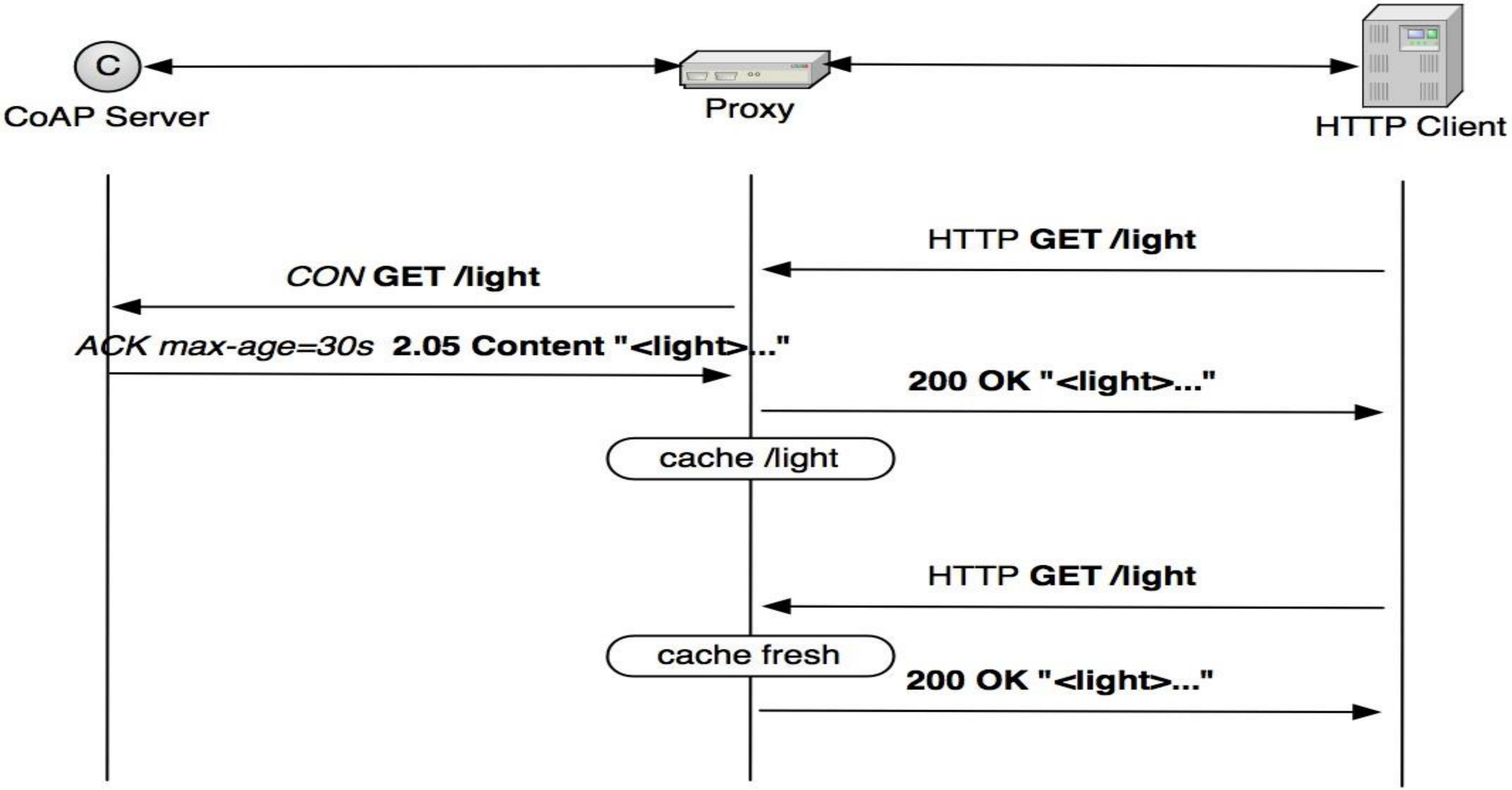
Validation model

- Validity checked using the Etag Option

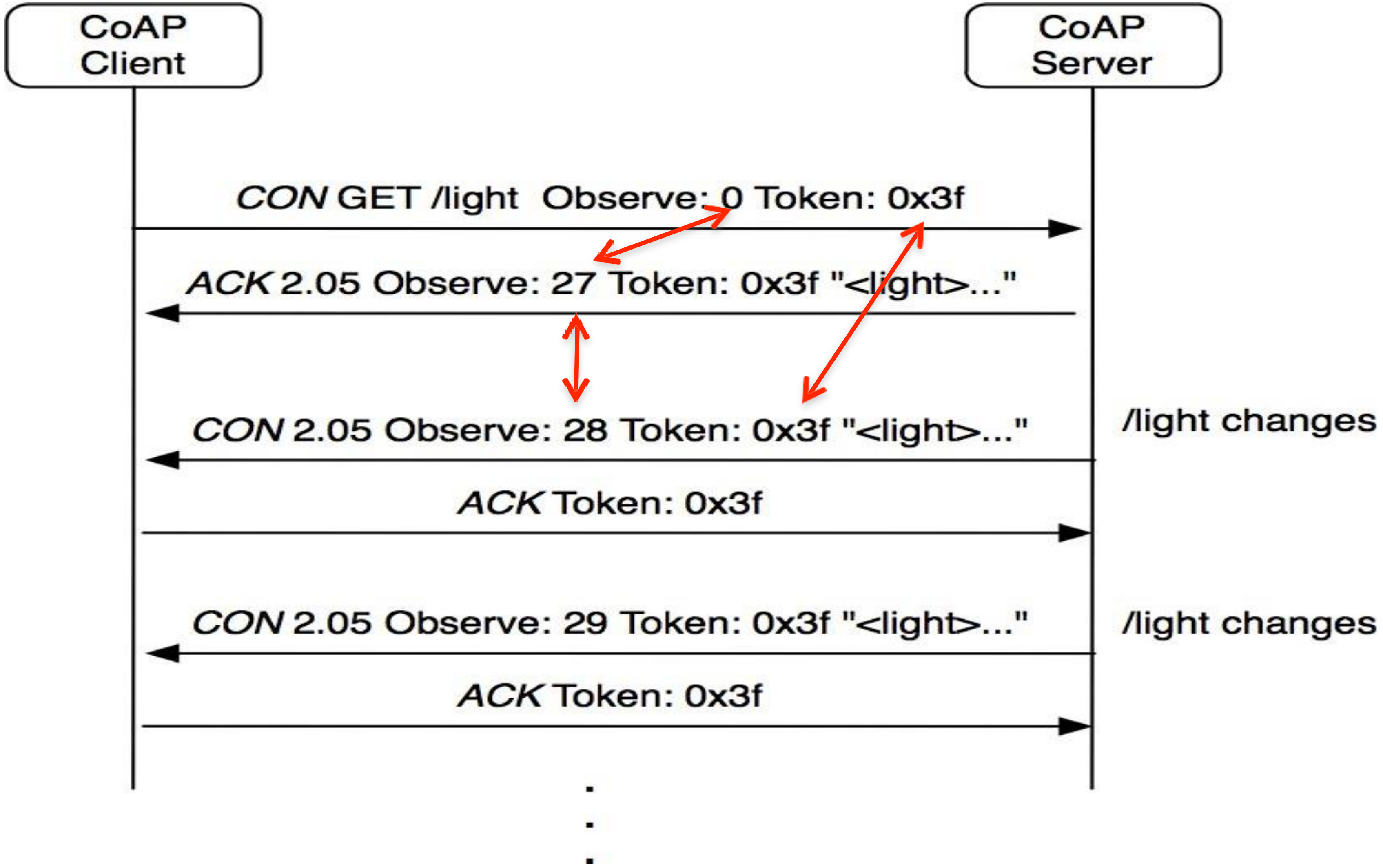
A proxy often supports caching

- Usually on behalf of a constrained node,
- a sleeping node,
- or to reduce network load

Proxying and caching

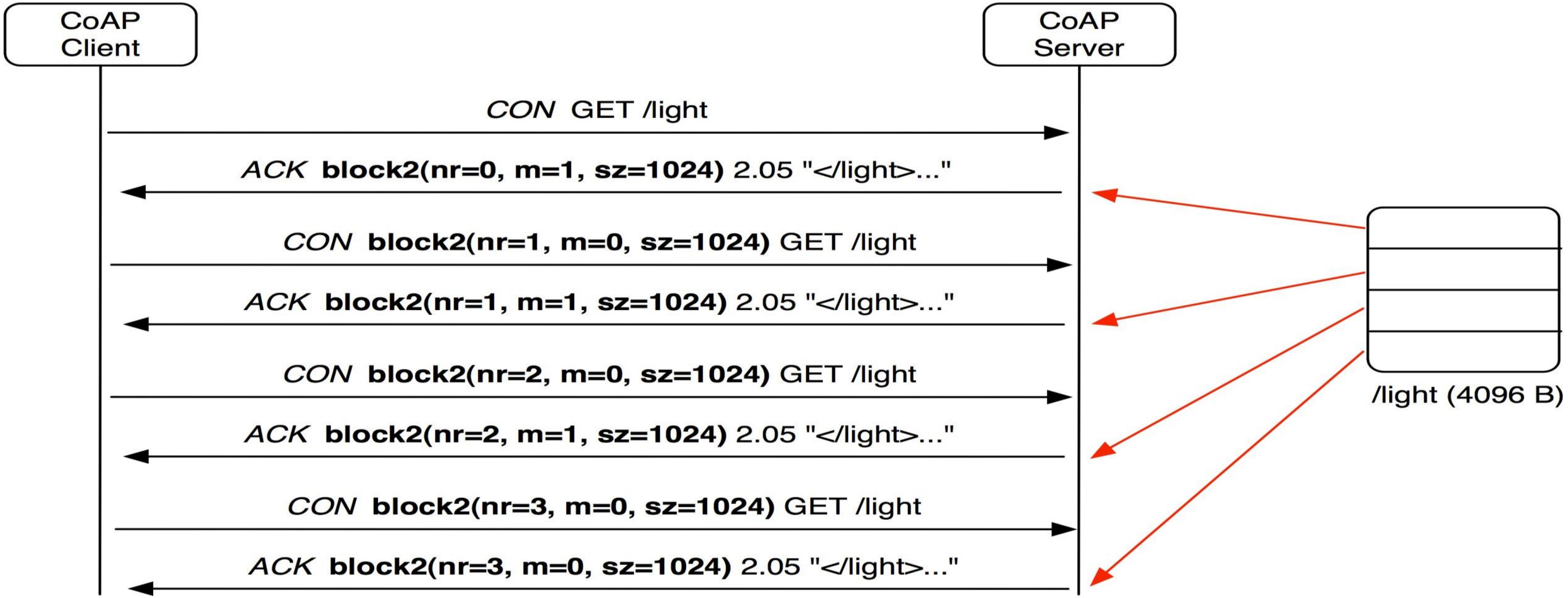


Observation



See draft-ietf-core-observe

Block transfer



See draft-ietf-core-block

Getting Started with CoAP

There are many open source implementations available

- mbed includes CoAP support
- Java CoAP Library Californium
- C CoAP Library Erbium
    - libCoAP C Library
    - jCoAP Java Library
- OpenCoAP C Library
- TinyOS and Contiki include CoAP support

CoAP is already part of many commercial products/systems

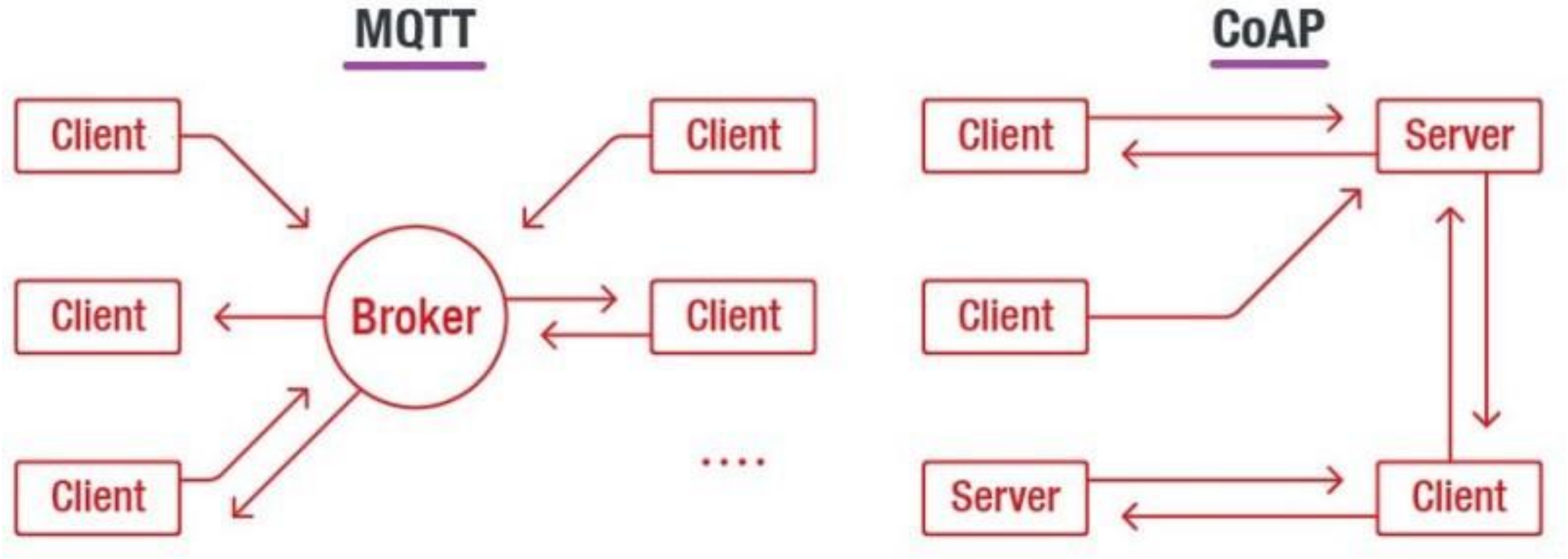
- ARM Sensinode NanoService
- RTX 4100 WiFi Module

Firefox has a CoAP plugin called Copper

Wireshark has CoAP dissector support

Implement CoAP yourself, it is not that hard!

CoAP vs. MQTT



MQTT
Client
Client
Client
Broker
Client
Client
....
CoAP
Client
Server
Client
Server
Client