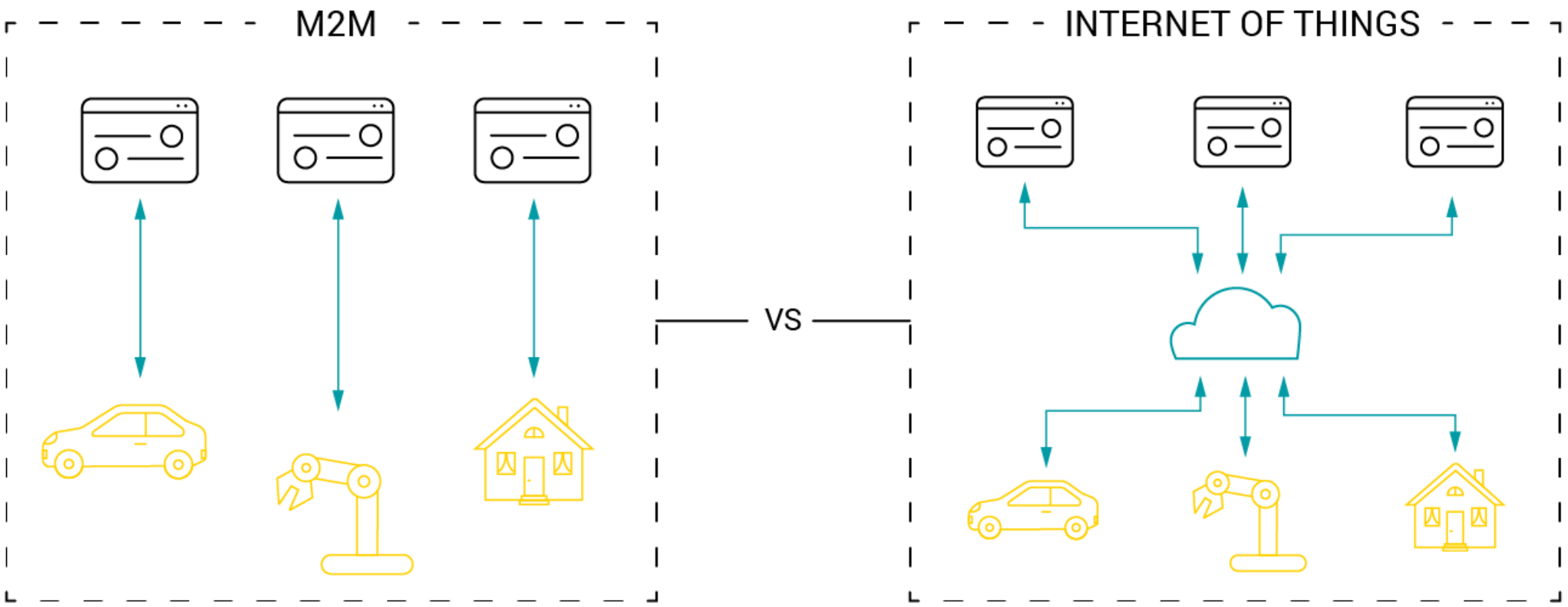
CoAP Protocol

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

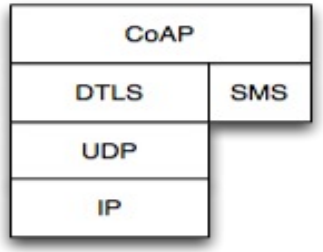
- Observe at new events happened on sensors or actuators.
- Device management and discoverability from external devices.
- Web protocol used in M2M with constrained requirements
- Asynchronous message exchange
- Low overhead and very simple to parse
- URI and content-type support
- Proxy and caching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

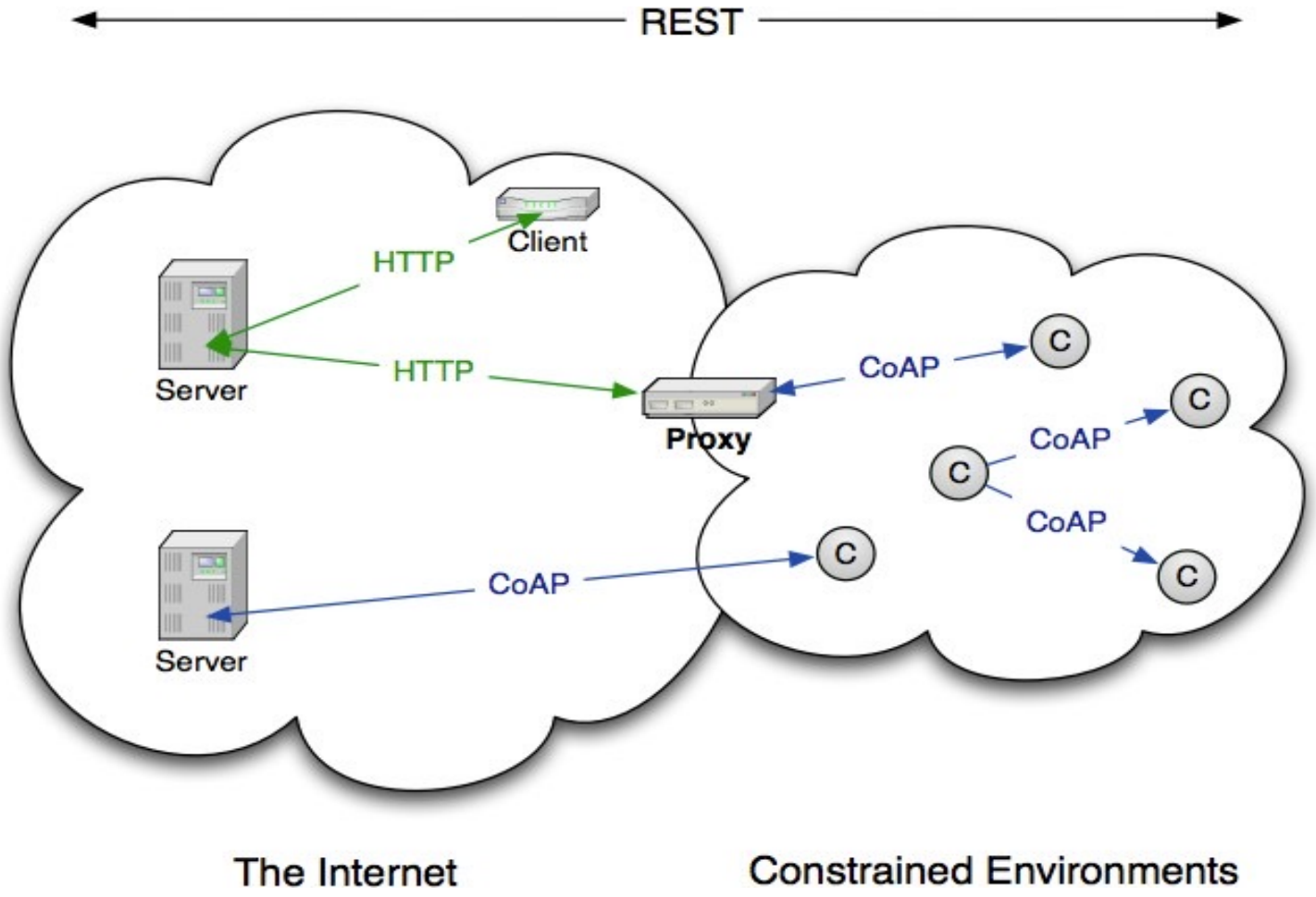
When to use CoAP?

- *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

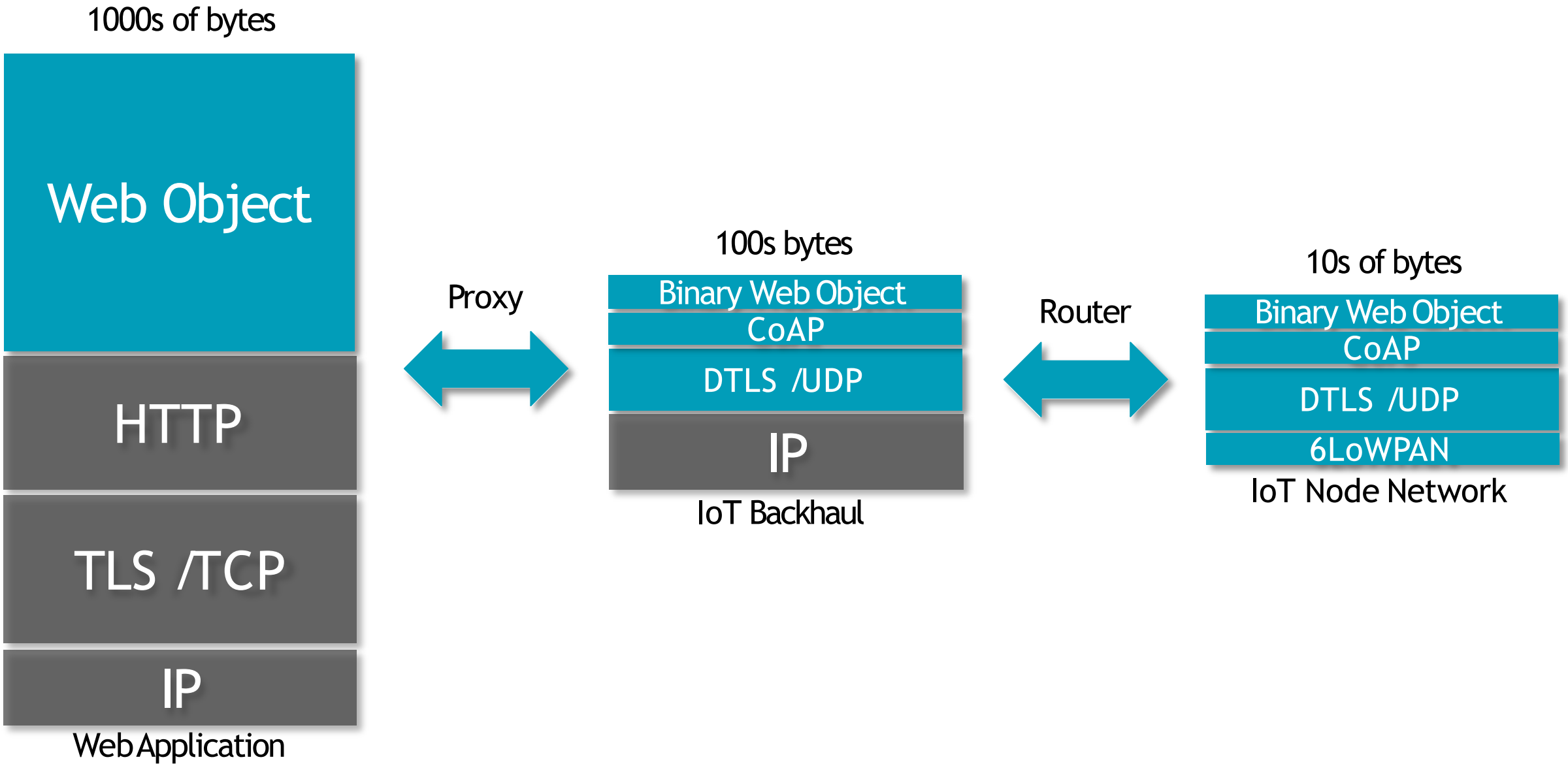
- Open IETF Standard
- Compact 4-byte Header
- UDP, SMS, (TCP) Support
- Strong DTLS Security
- Asynchronous Subscription
- Built-in Discovery



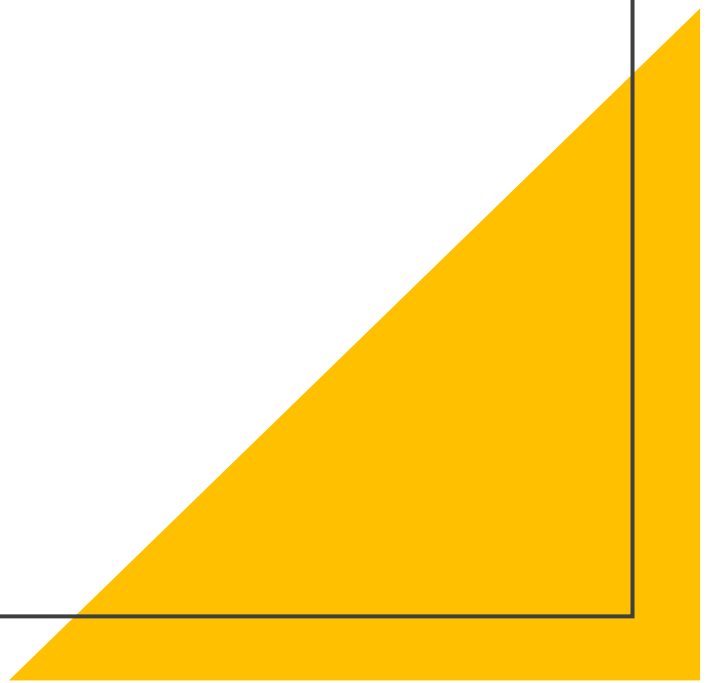


IETF

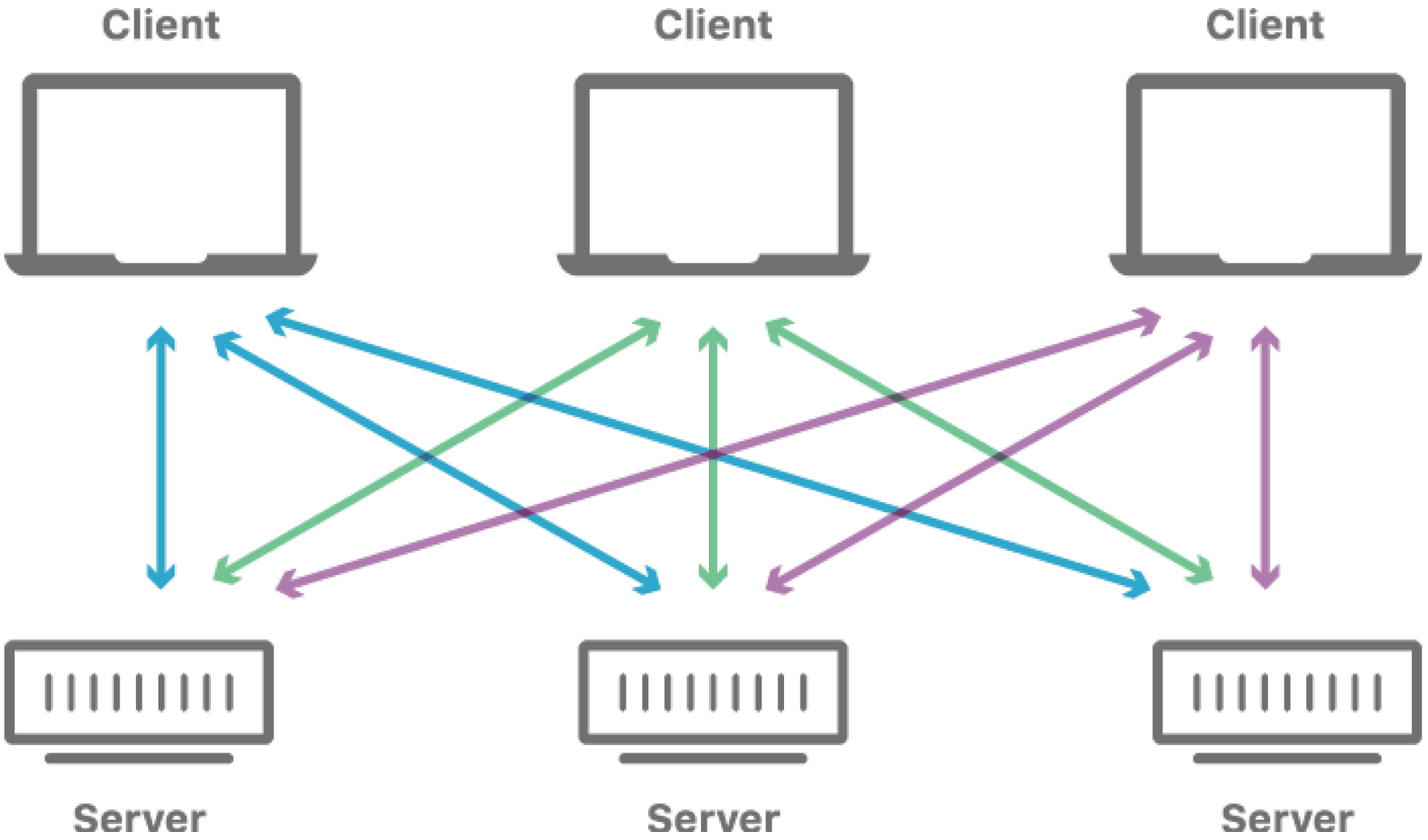
From Web Applications to IoT Nodes



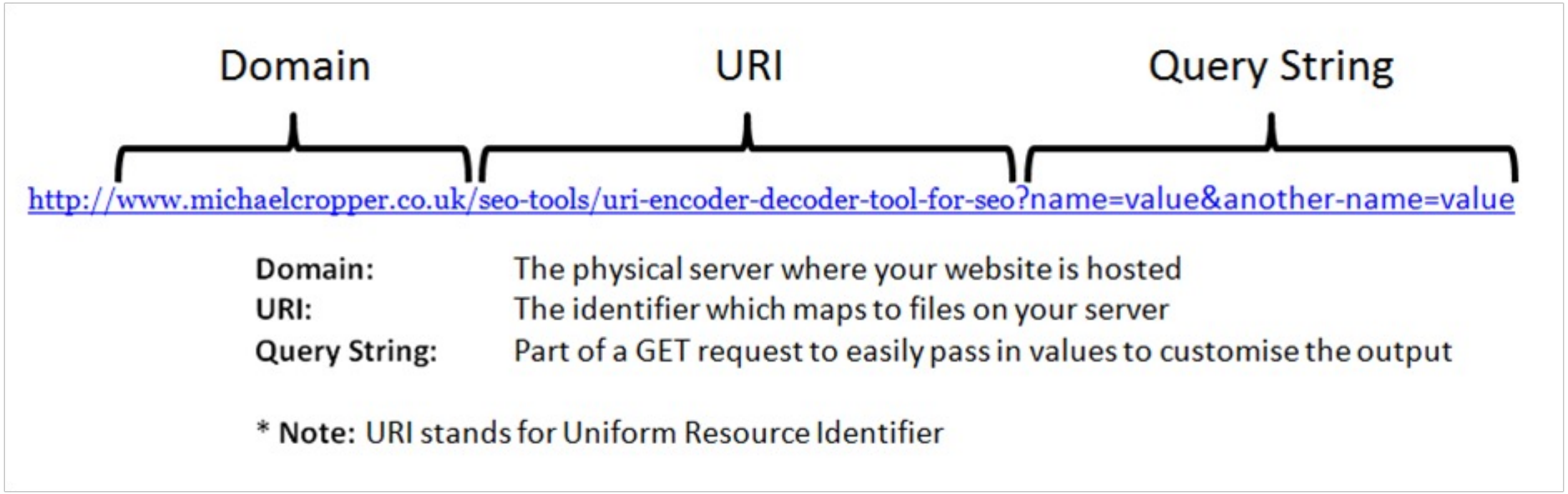
The Web and REST

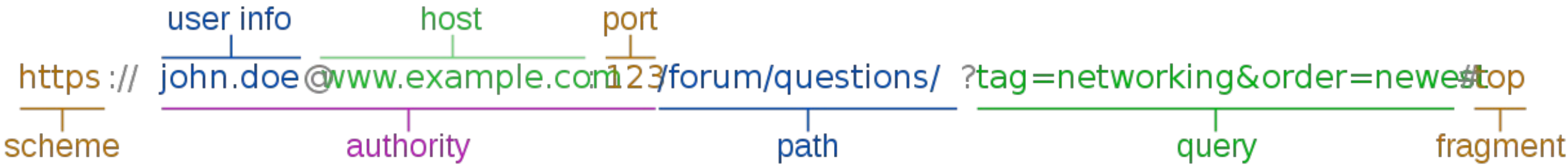


Web Architecture

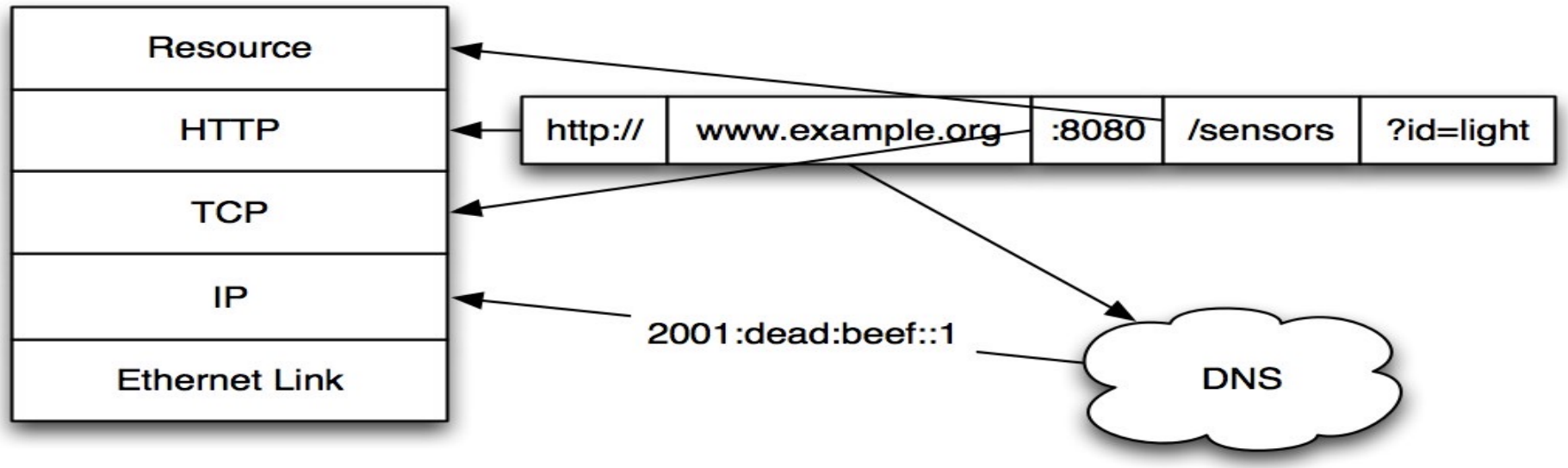


Web Naming

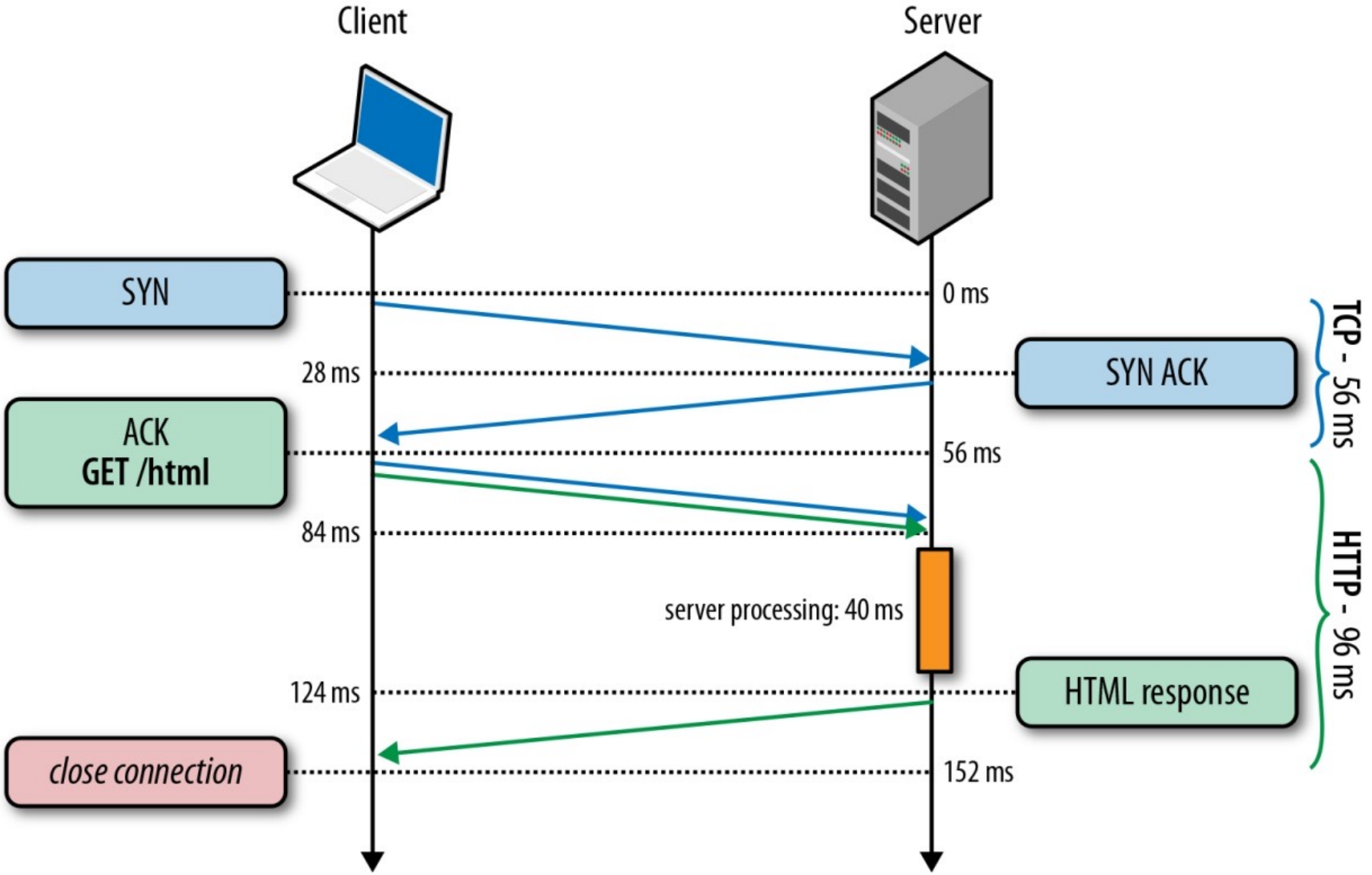

Domain
URI
Query String
http://www.michaelcropper.co.uk/seo-tools/uri-encoder-decoder-tool-for-seo?name=value&another-name=value
Domain: The physical server where your website is hosted
URI: The identifier which maps to files on your server
Query String: Part of a GET request to easily pass in values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

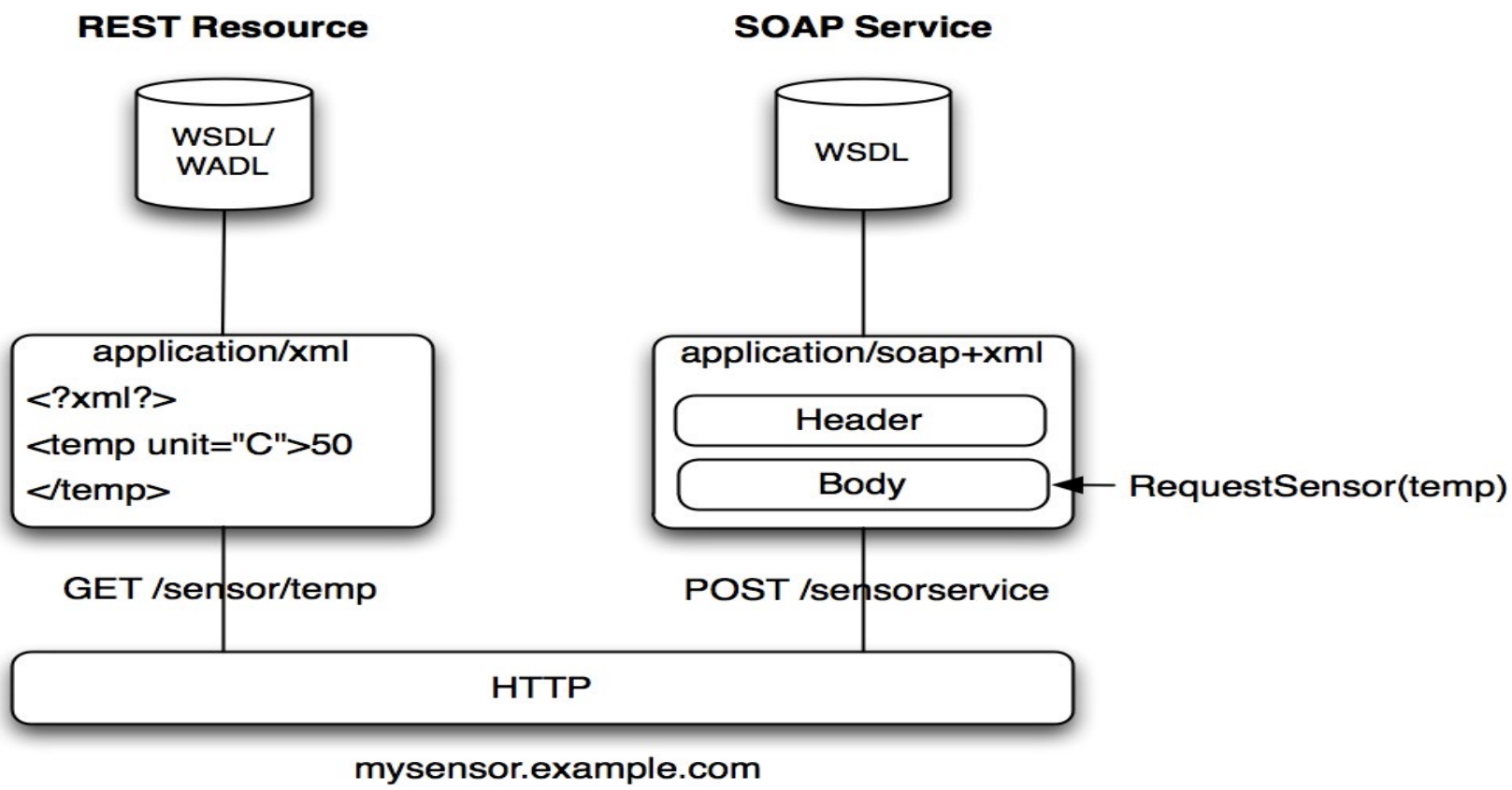
URL Resolution



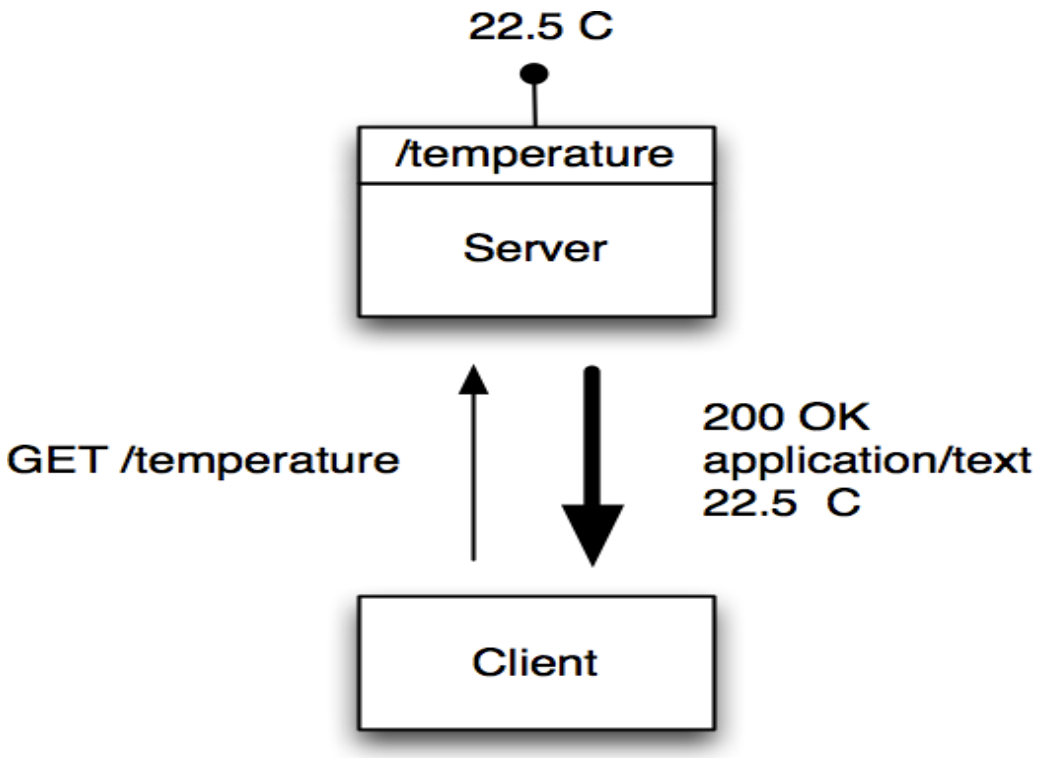
HTTP Request



Web Paradigms

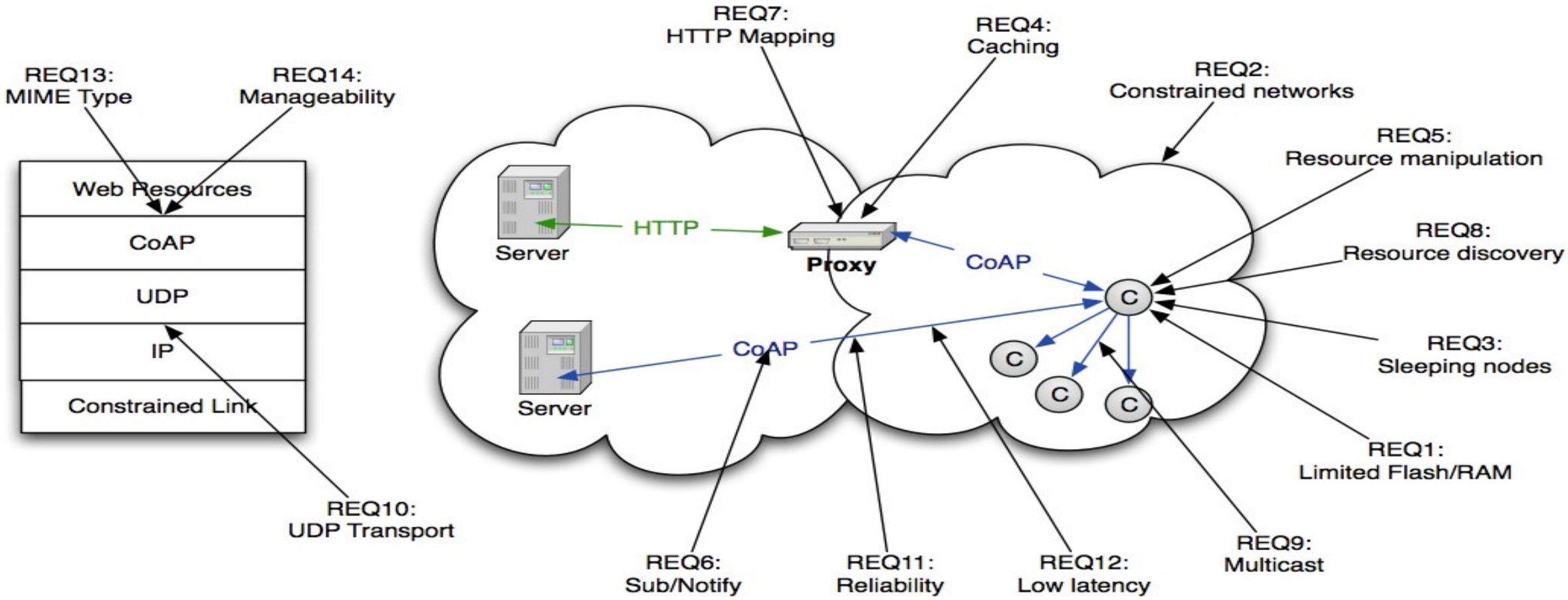


A REST Request

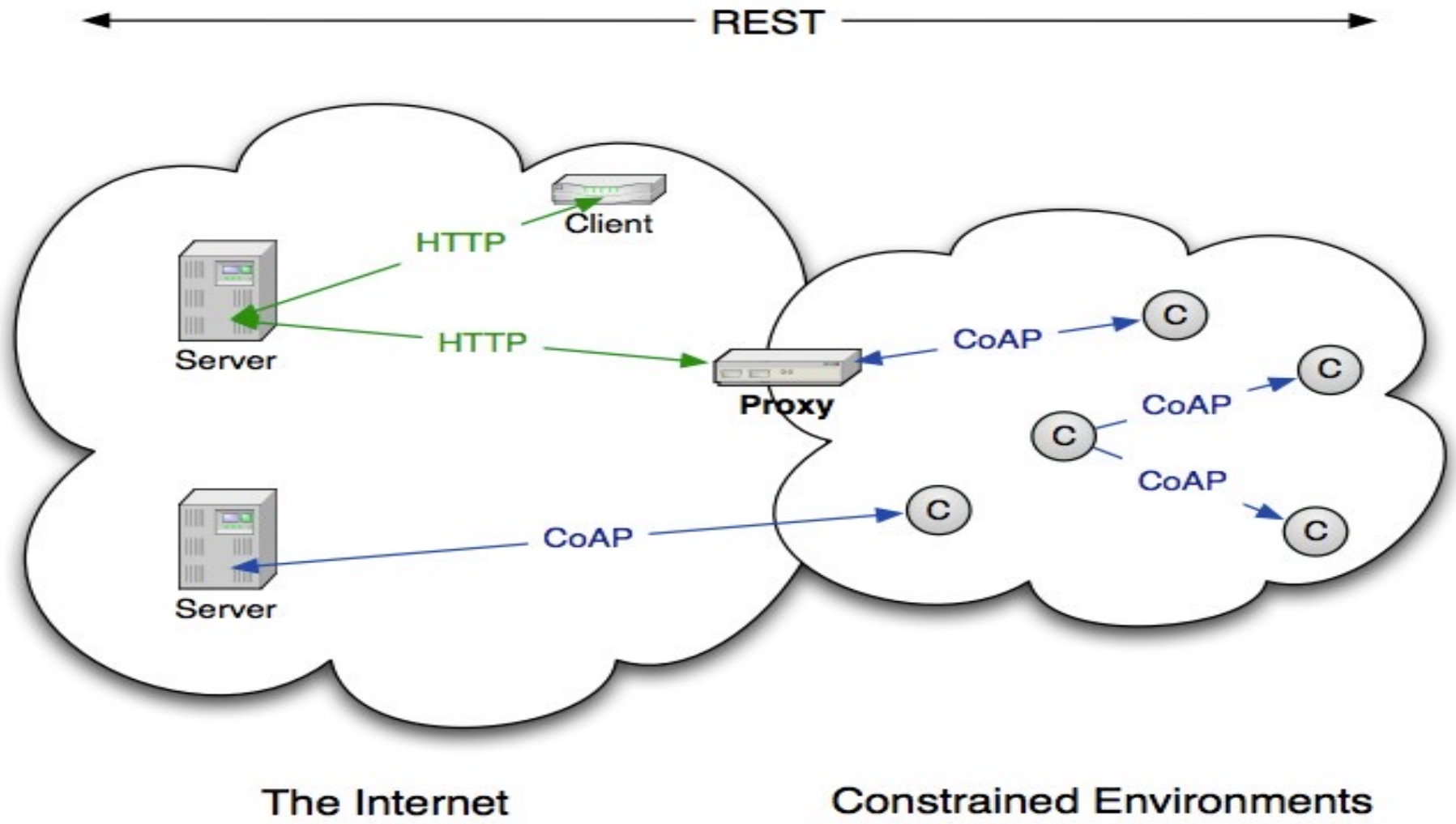


CoAP: Constrained Application Protocol

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

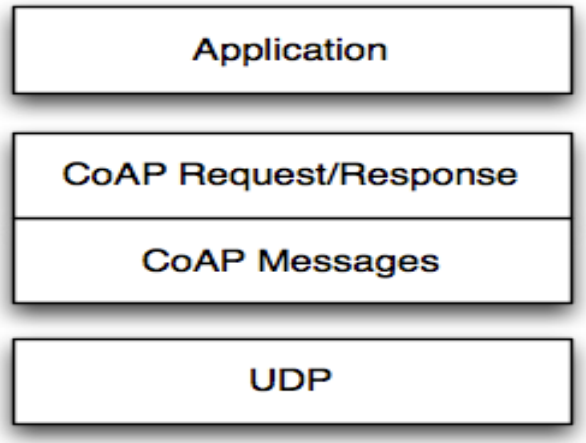
CoAP is not

- 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

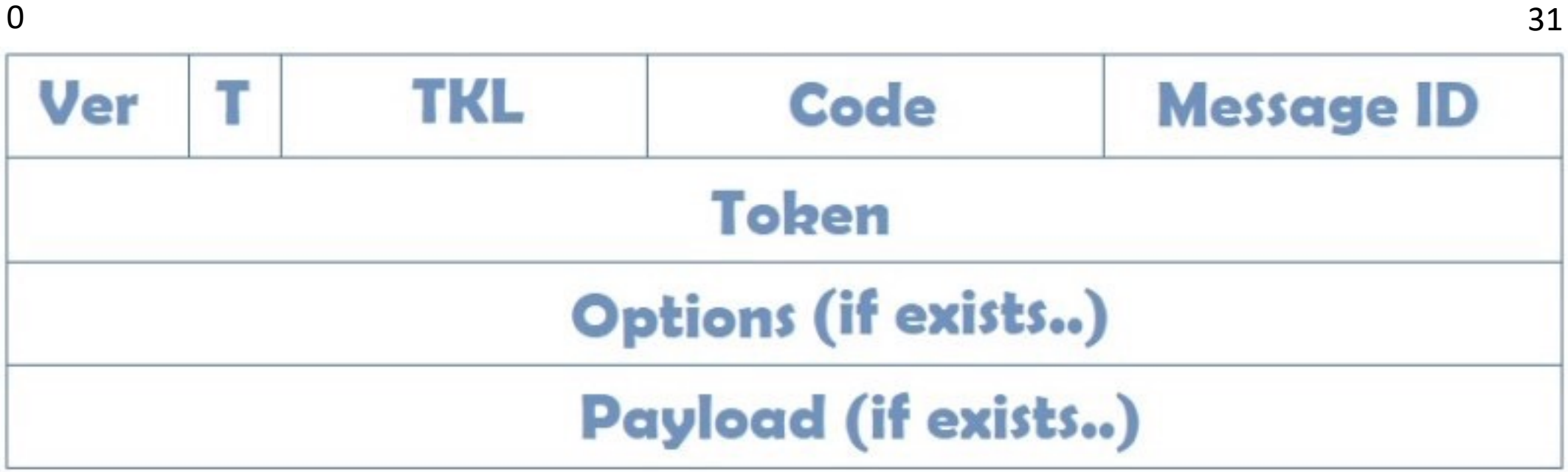
Base Messaging

- Simple message exchange between endpoints
- Confirmable or Non-Confirmable Message answered by Acknowledgement or Reset Message

REST Semantics

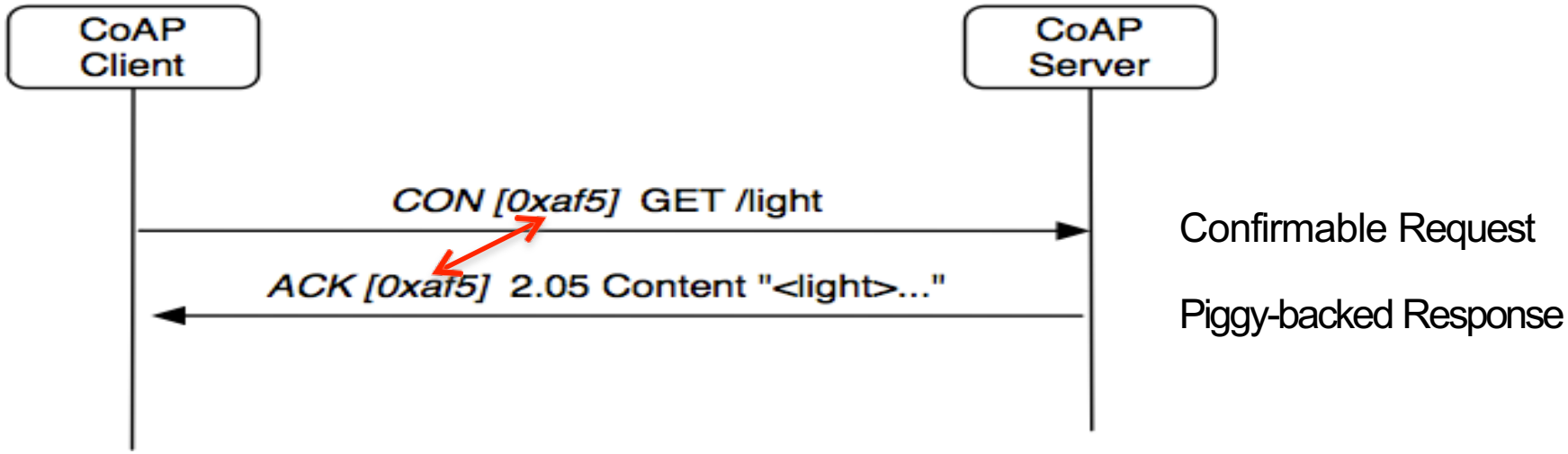
- REST Request/Response piggybacked on CoAP Messages
- Method, Response Code and Options (URI, content-type etc.)

Message Header (4 bytes)



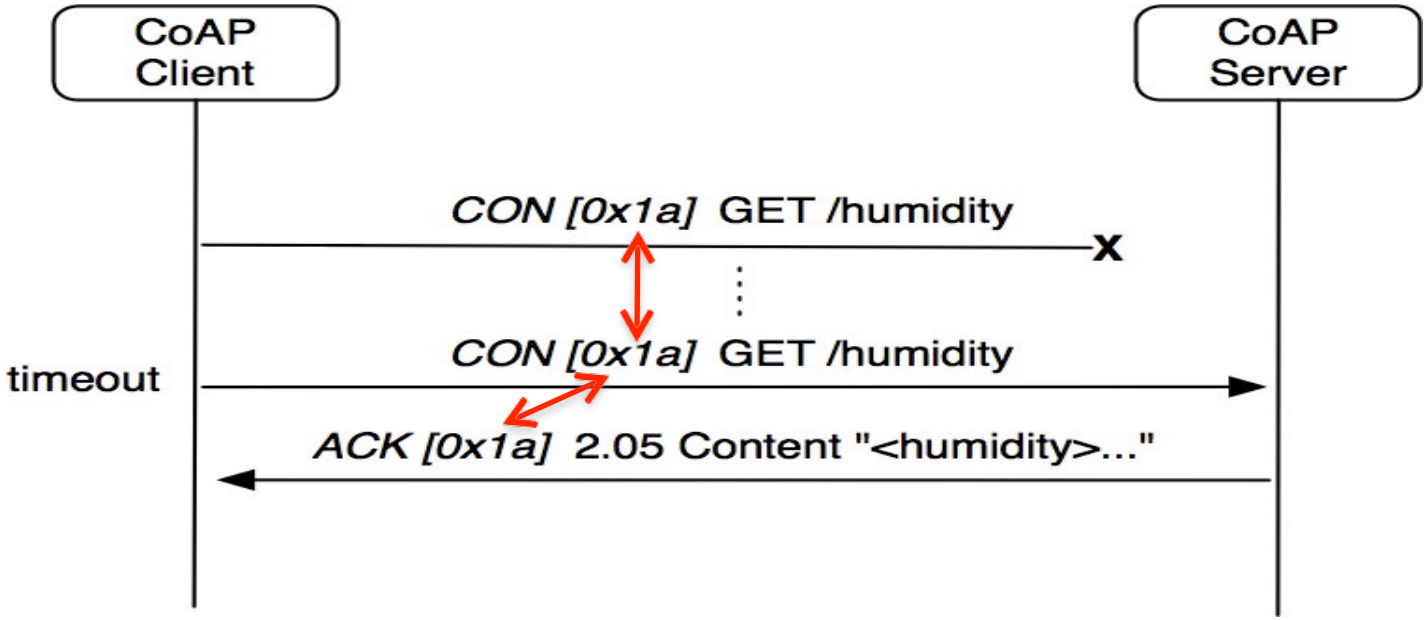
Ver: It is a 2 bit unsigned integer indicating the version
T: it is a 2 bit unsigned integer indicating the message type: 0 confirmable, 1 non-confirmable
TKL: Token Length is the token 4 bit length
Code: It is the code response (8 bit length)
Message ID: It is the message ID expressed with 16 bit

Request Example

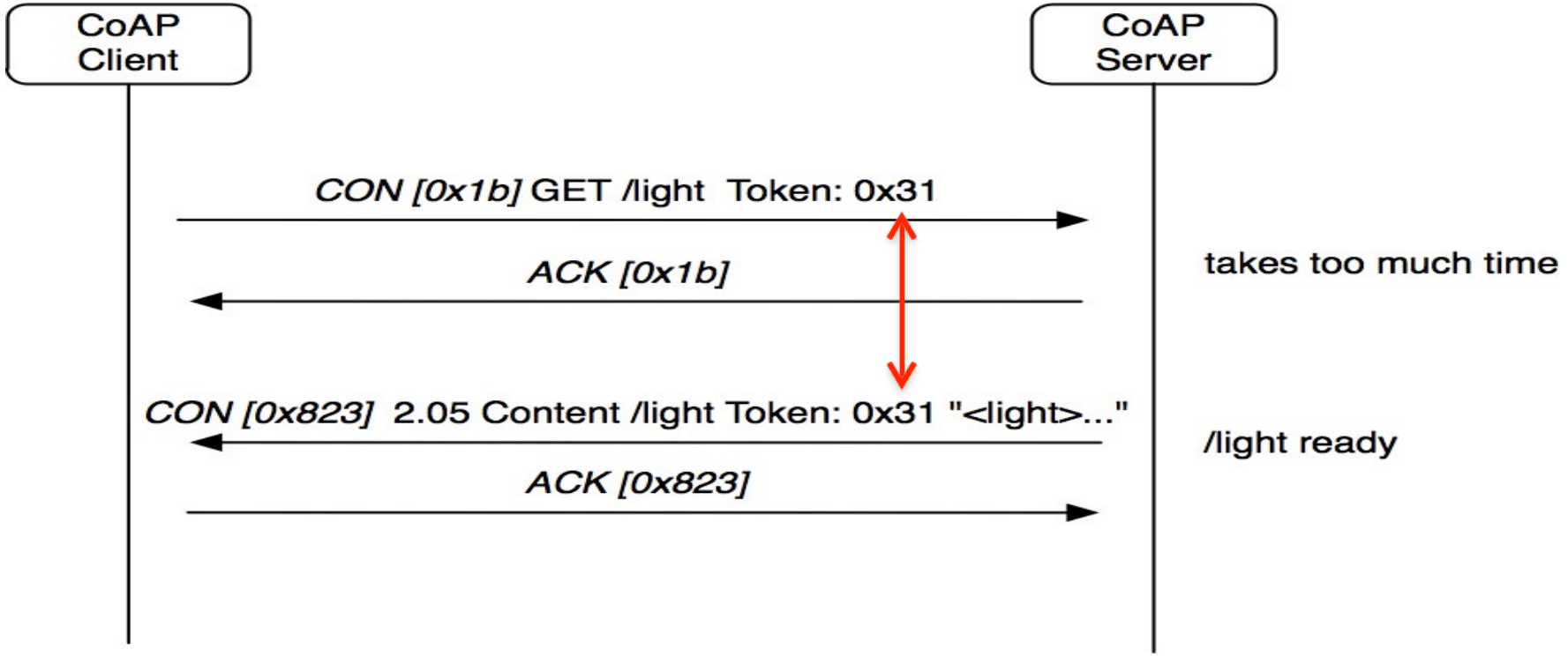


In the above diagram, you can see communication but If the server has troubles managing the incoming request it can send back a Rest message (RST) instead of the Acknowledge message (ACK).

Dealing with Packet Loss

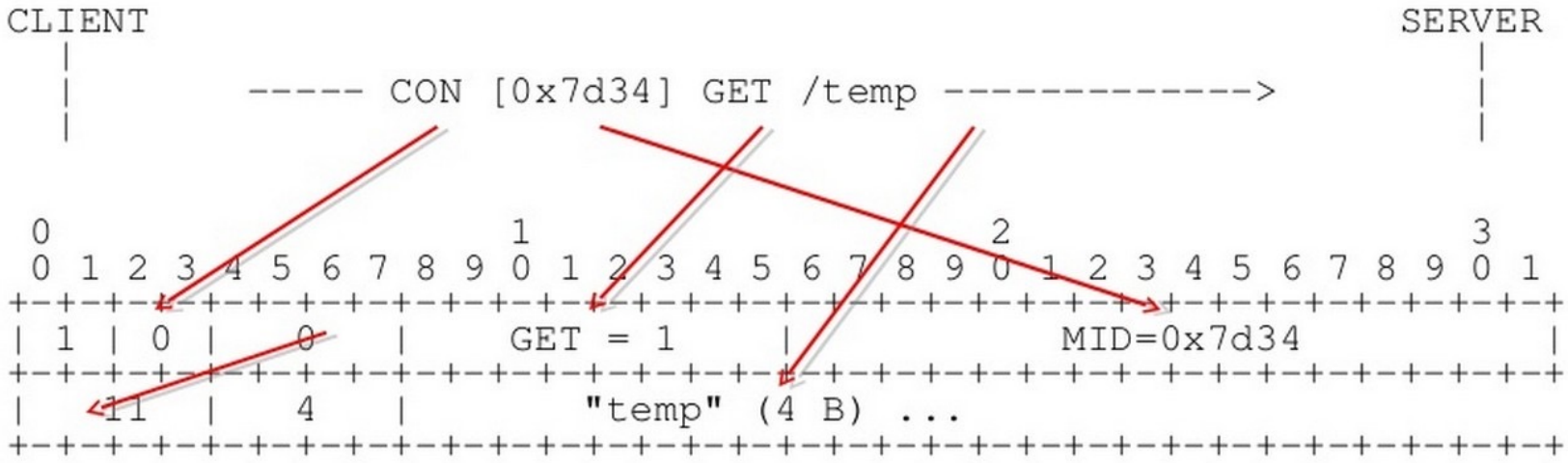


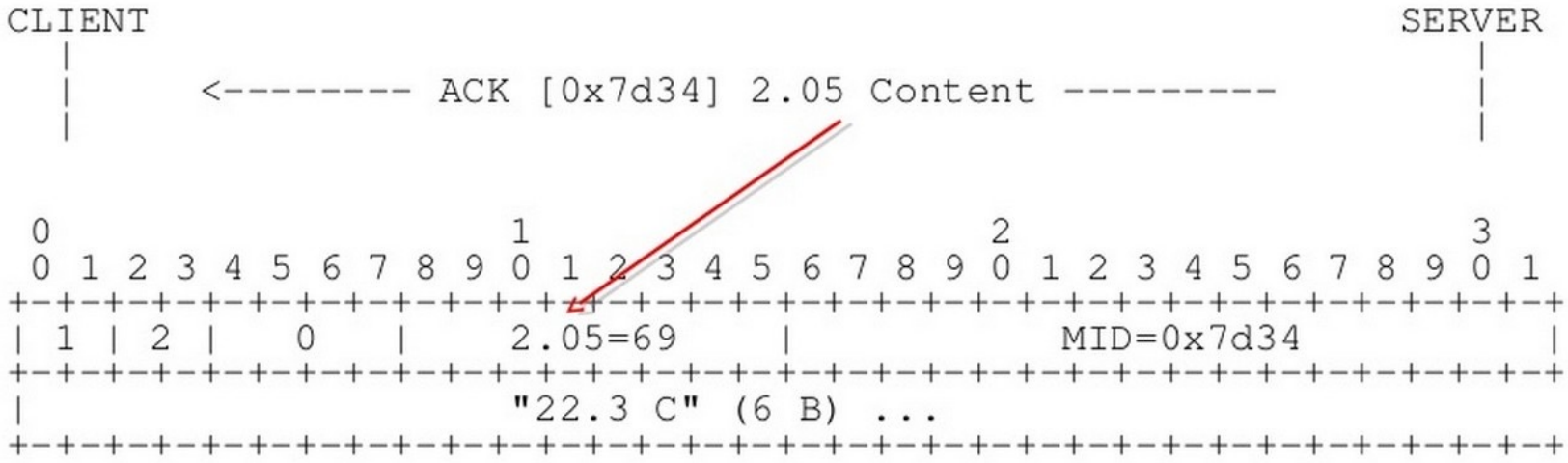
Separate Response



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...





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

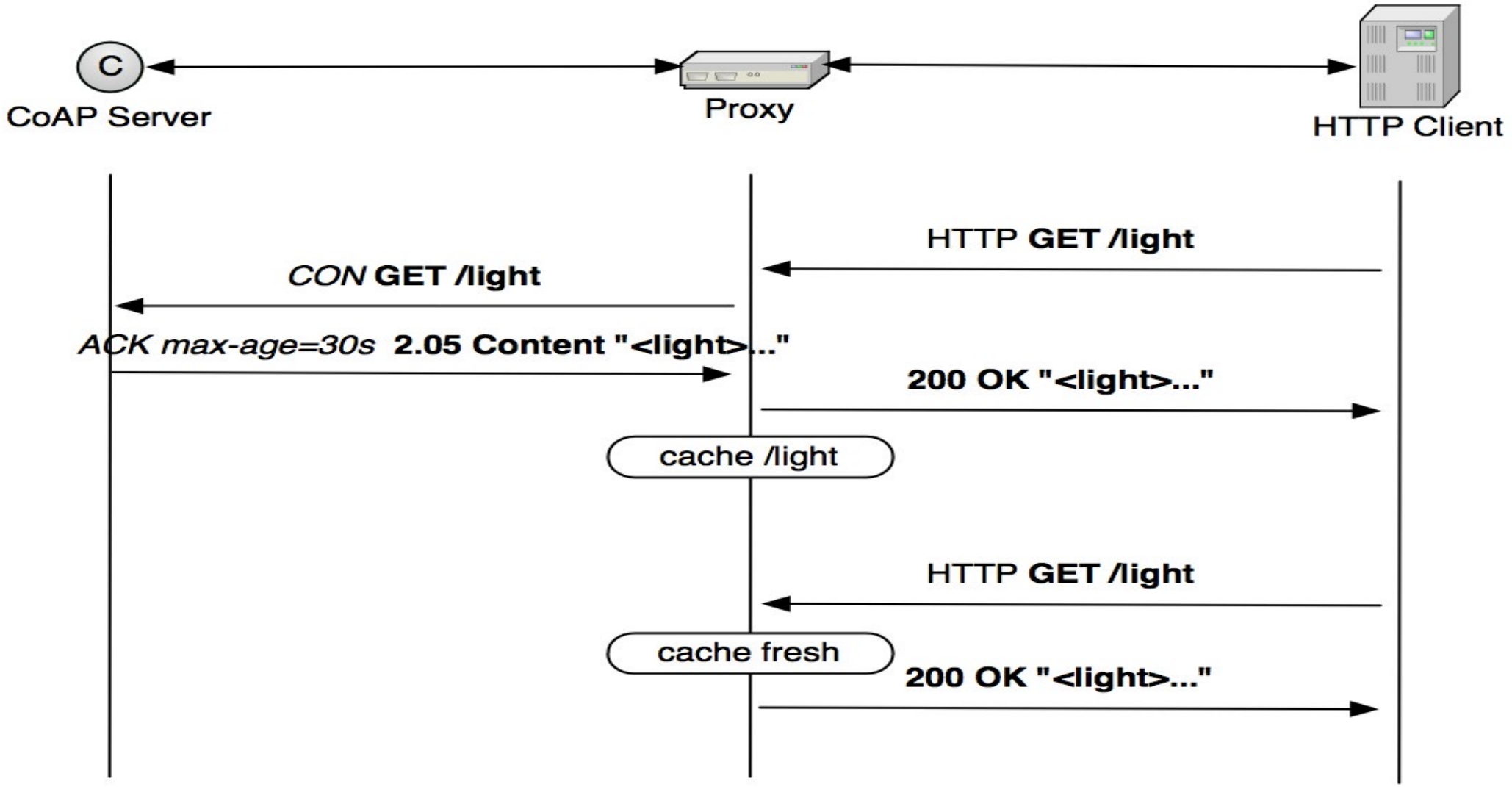
Validation model

- Validity checked using the Etag Option

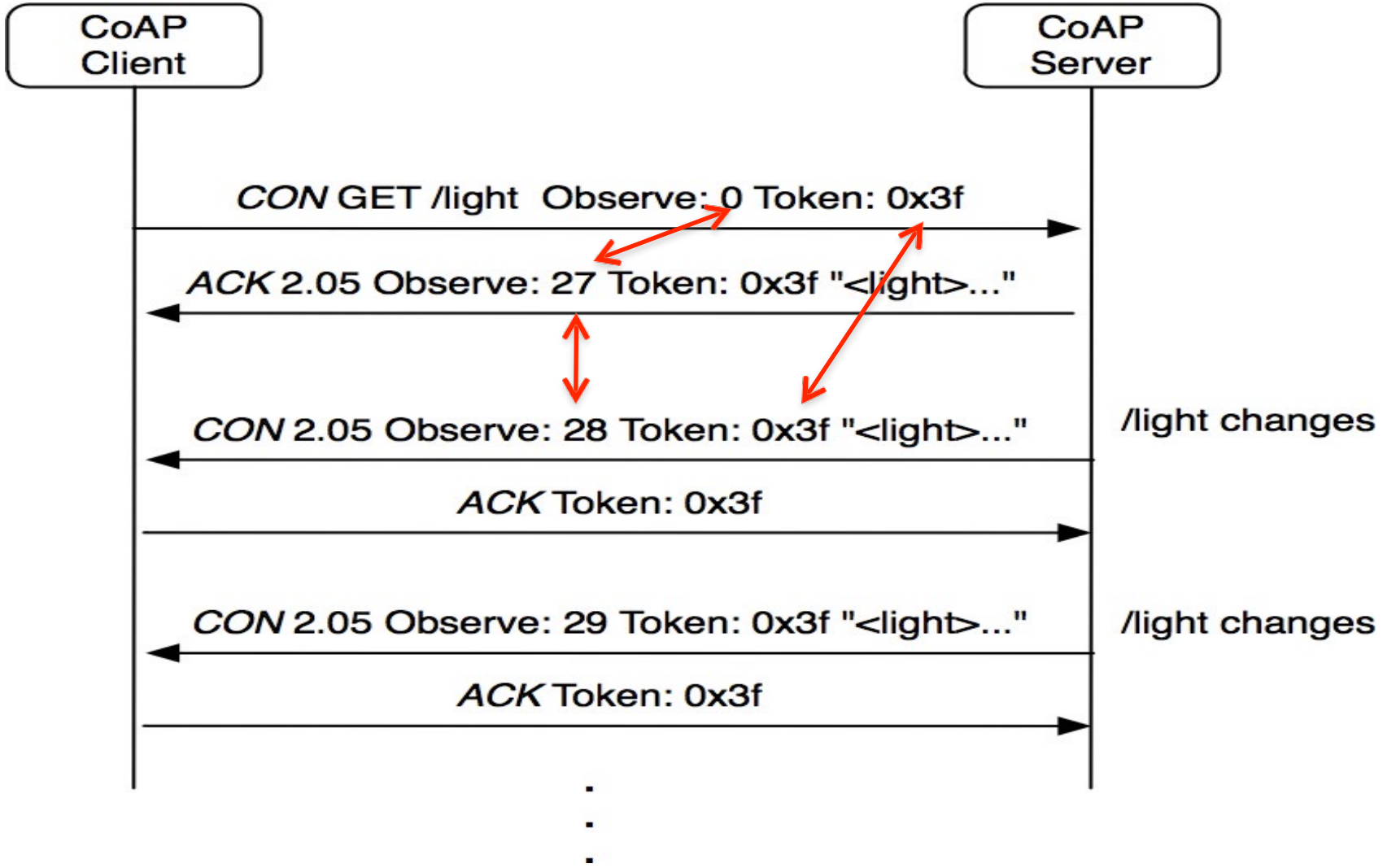
A proxy often supports caching

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

Proxying and caching

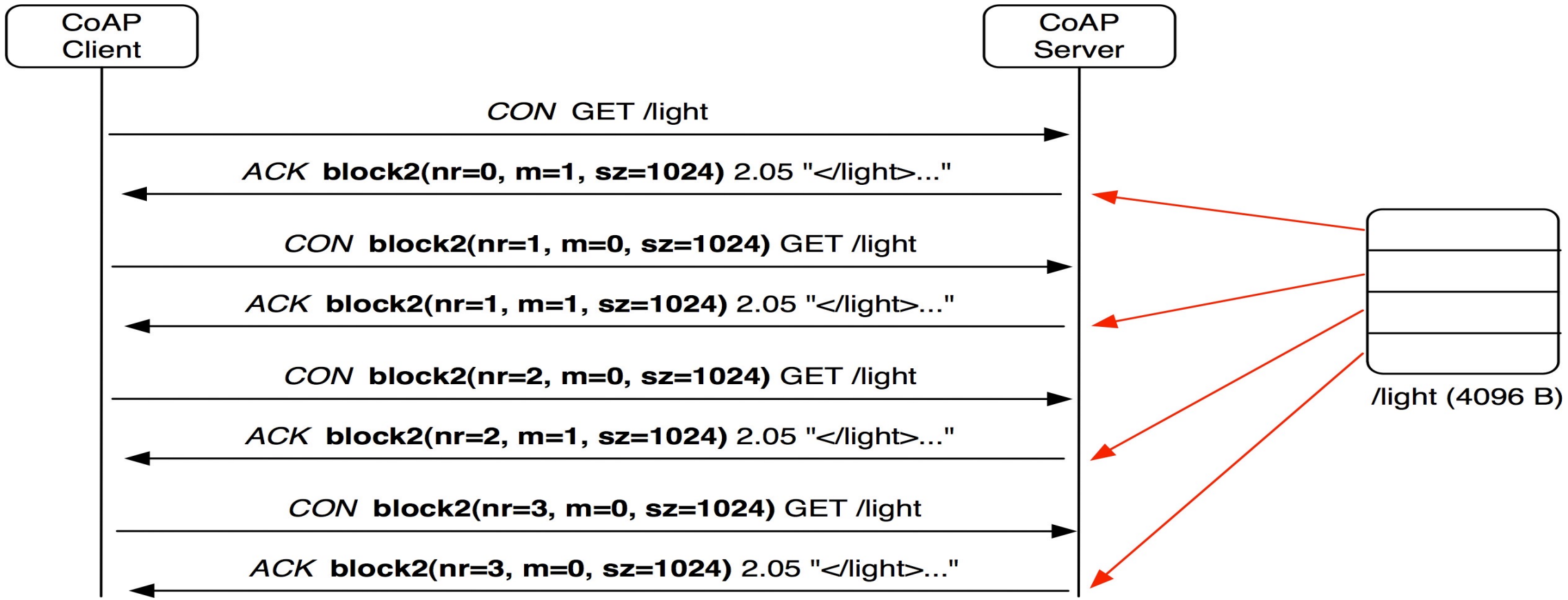


Observation



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

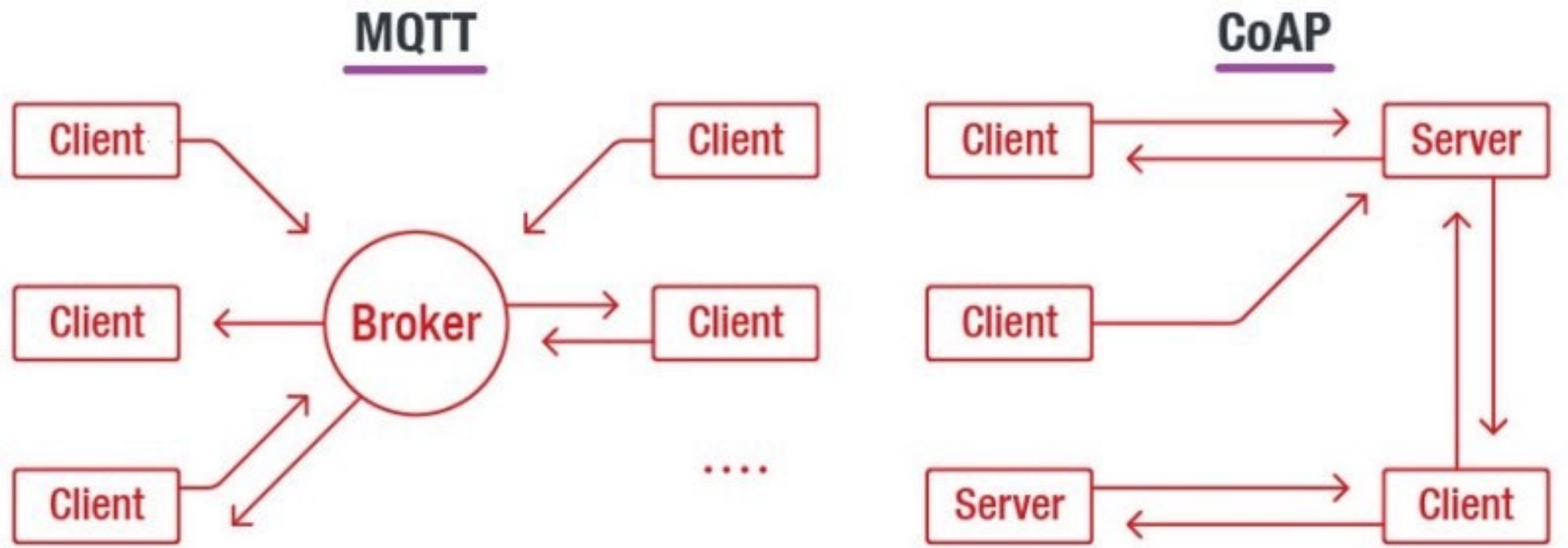
- ARM Sensinode NanoService
- RTX 4100 WiFi Module

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

MQTT Protocol

MQTT - *Message Queuing Telemetry Transport*

- Machine-to-machine (M2M)/"Internet of Things" connectivity protocol
- Invented by Dr. Andy Stanford-Clark of IBM and Arlen Nipper of Arcom (now Eurotech) in 1999
- ISO standard (ISO/IEC PRF 20922)
- Public and royalty-free license
- Used by Amazon Web Services, IBM WebSphere MQ, Microsoft Azure IoT, Adafruit, Facebook Messenger etc.

MQTT Features

- Small code footprint
- Ideal if processor or memory resources are limited
- Ideal if bandwidth is low or network is unreliable
- Publish/subscribe message exchange pattern
- Works on top of TCP/IP
- Quality of service levels: at most once, at least once, exactly once
- Client libraries for Android, Arduino, C, C++, C#, Java, JavaScript, .NET etc.
- Security: authentication using user name and password, encryption using SSL/TLS
- Support for persistent messages stored on the broker

Applications

- Home automation (e.g. smart lightning, smart metering)
- Healthcare
- Mobile phone apps (e.g. messaging, monitoring)
- Industrial automation
- Automotive
- General IoT applications

Publish/Subscribe

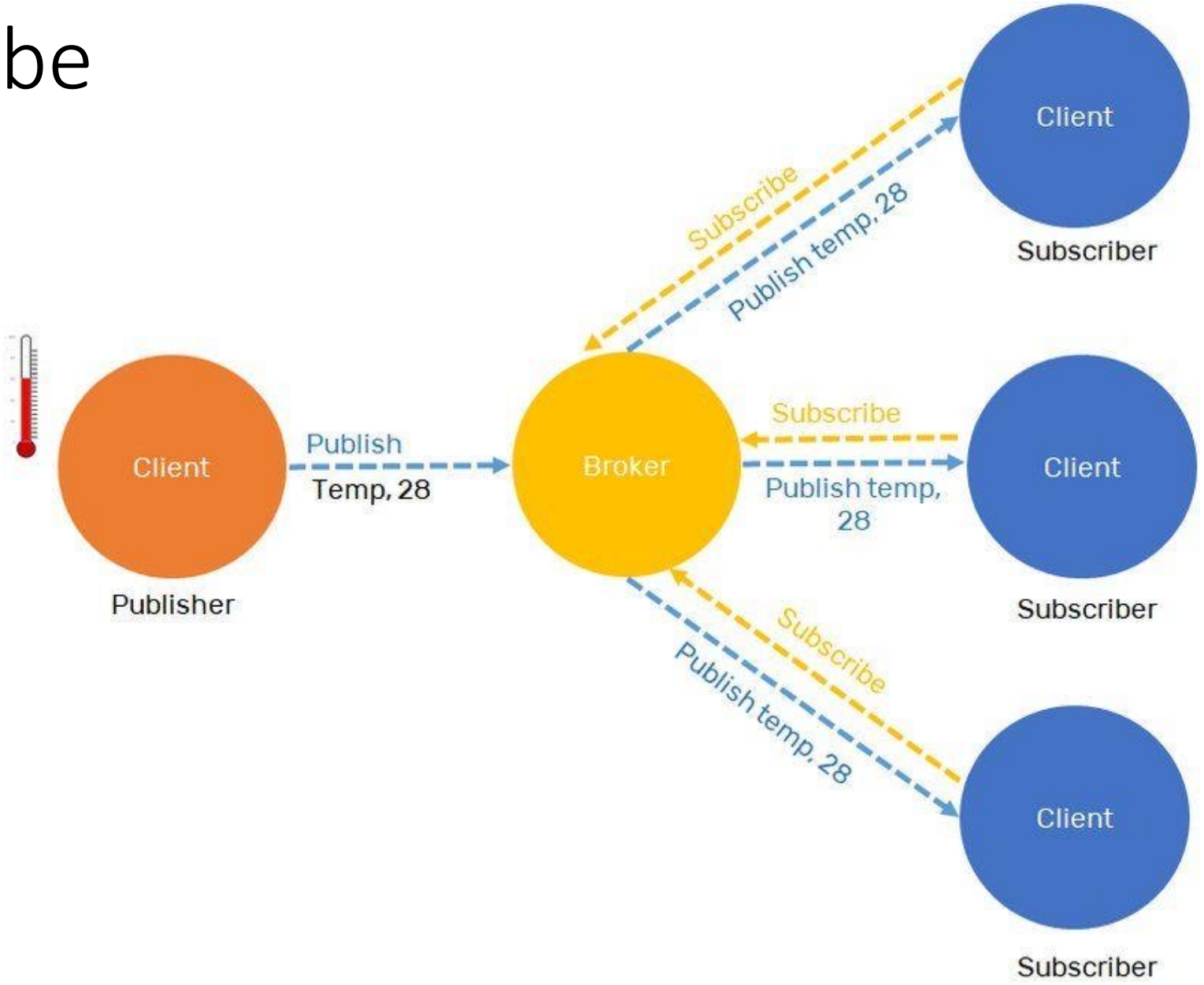
- Multiple clients connect to a broker and subscribe to topics that they are interested in
- Clients connect to the broker and publish messages to topics.
- Topics are treated as a hierarchy, using a slash (/) as a separator.
- Example: multiple sensor devices may publish temperature readings on the topic:

  sensors/DEVICE_NAME/temperature/NODE_ID

- Clients can receive messages by creating subscriptions. A subscription may be to an explicit topic, in which case only messages to that topic will be received, or it may include wildcards.
- Two wildcards are available: + or #
- MQTT clients can register a custom ‘last will testament’ message to be sent by the broker if they disconnect.
- This message can be used to signal to subscribers when a device disconnects

Publish/Subscribe

- **Topics/Subscriptions**: Messages are published to topics. Clients can subscribe to a topic or a set of related topics
- **Publish/Subscribe**: Clients can subscribe to topics or publish to topics



Request/Response

1) Client

- subscribe topic "function-xyz/response/<id>" //note: <id> is a client unique ID

2) Server

- subscribe topic "function-xyz/request/+" //note: "+" is a wildcard

3) Client

- publish topic "function-xyz/request/<id>" payload <input parameter>

4) Server

- receive notification "function-xyz/request/<id>" payload <input parameter>
- retrieve <id> from string
- process function-xyz(<input parameter>)
- publish topic "function-xyz/response/<id> " payload "<response>"

5) Client

- receive notification "function-xyz/response/<id>" payload "<response>"

QoS Levels

- 0 -> At most once (Best effort, No Ack)
- 1 -> At least once (Acked, retransmitted if ack not received)
- 2 -> Exactly once [Request to send (Publish), Clear-to-send (Pubrec), message (Pubrel), ack (Pubcomp)]

- **Retained Messages**: Server keeps messages even after sending them to all subscribers. New subscribers get the retained messages

MQTT Features

- **Clean Sessions** and **Durable Connections**
  - At connection set up: Clean session flag -> all subscriptions are removed on disconnect, otherwise subscriptions remain in effect after disconnection
  - Subsequent messages with high QoS are stored for delivery after reconnection
- **Wills**
  - At connection a client can inform that it has a will or a message that should be published if unexpected disconnection
  - Alarm if the client loses connection
- Periodic **keep alive** messages -> If a client is still alive
- **Topic Trees -** topics are organized as trees using the / character
  - /# matches all sublevels
  - /+ matches only one sublevel