APPLICATION LAYER

COAP

COAP

- Application layer protocol over UDP (User Datagram Protocol)
- An extended tiny version of HTTP
  - Not all HTTP methods are supported in CoAP
  - CoAP supports non-HTTP features (push and multicast)
  - CoAP is optimized for bandwidth and processing efficiency
- RESTful (Representational State Transfer: internet protocol) for easy interfacing with HTTP
- Designed for datagram transport protocols (UDP instead of TCP)
  - Requests and responses are matched through message tokens
  - Multicast can be used to request responses from various resources
- Designed to be used with constrained nodes and lossy networks integration with 6LoWPAN (a wireless extension of the internet protocol)
- Built-in resource discovery and observation
- Because CoAP is datagram based, it may be used on top of SMS and other packet based communications protocols.
- Standardized in IETF RFC 7252

COAP (RESOURCE DISCOVERY)

- CoAP defines a standard mechanism for resource discovery. Servers provide a list of their resources (along with metadata about them) at /.well-known/core. A well-known URI is defined as a default entry-pint for requesting links hosted by the server. These links are in the application/link-format media type and allow a client to discover what resources are provided and what media types they are.

- Constrained RESTful Environments (CoRE) Link Format used as resource discovery representation for CoAP
- CoAP servers publish CoRE as one way for discovering resources
- CoRE supports filters when discovering resources

REQ: GET /sensors
RES: 2.05 Content
</sensors/temp>;rt="temperature-c";if="sensor",
</sensors/light>;rt="light-lux";if="sensor"

METHOD DEFINITIONS

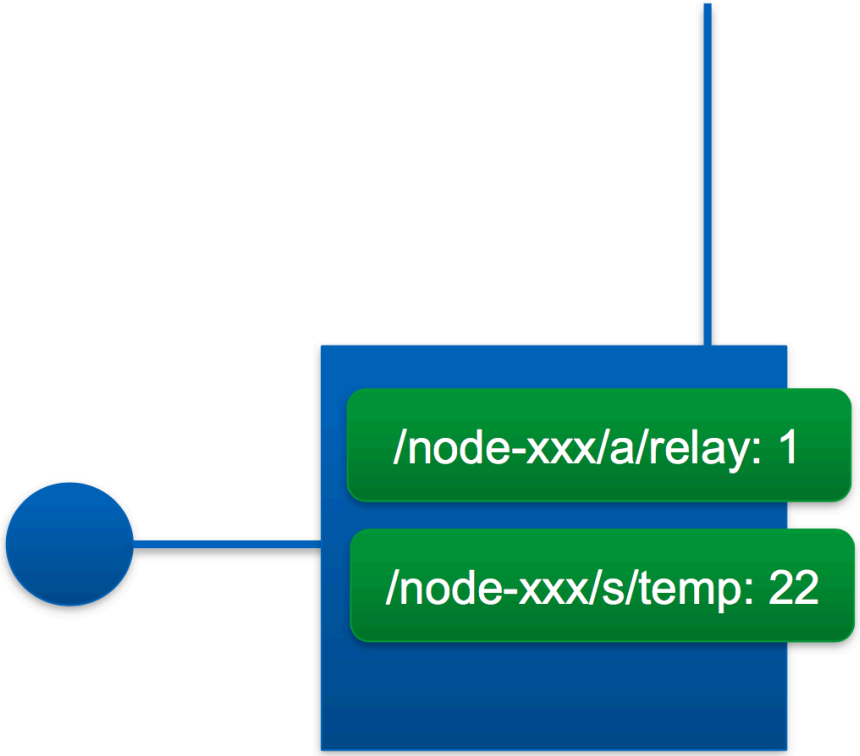
The GET method retrieves a representation for the information that currently corresponds to the resource identified by the reques URI. An implementation MAY relax the requirement to answer all retransmissions of a request with the same response, obviating the need to maintain state for Message IDs.

The POST method requests that the representation enclosed in the request be processed. The actual function performed by the POST method is determined by the origin server and dependent on the target resource. It usually results in a new resource being created or the target resource being updated.

The PUT method requests that the resource identified by the request URI be updated or created with the enclosed representation The representation format is specified by the media type given in the Content-Type Option.

The DELETE method requests that the resource identified by the request URI be deleted. A (Deleted) response SHOULD be sent o success or in case the resource did not exist before the request.

COAP RESOURCE DISCOVERY



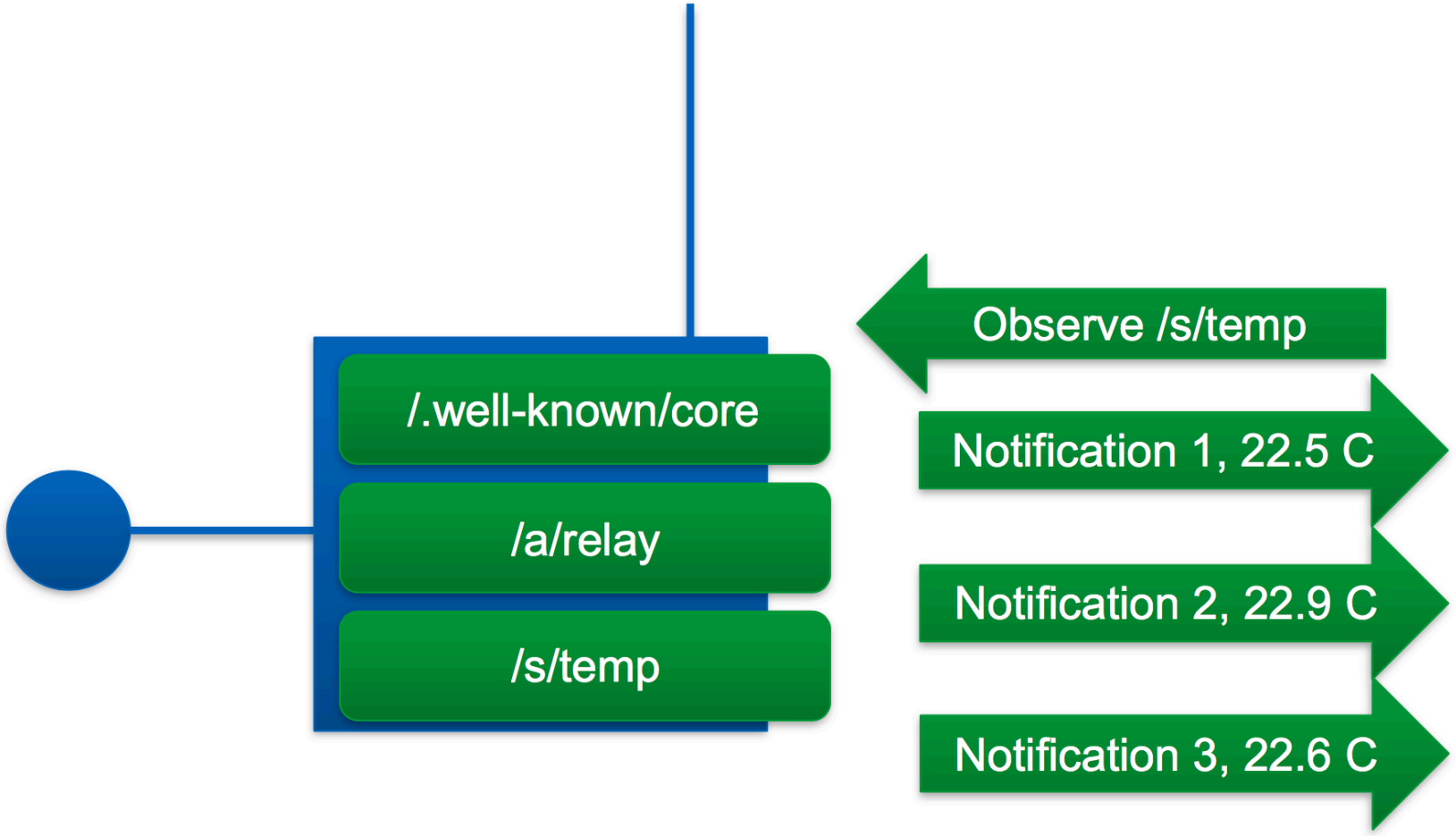
GET /node-xxx/a/relay
-> 1

GET /node-xxx/s/temp
-> 22

POST /node-xxx/a/relay/1
-> OK

DELETE /node-xxx/a/relay
-> OK

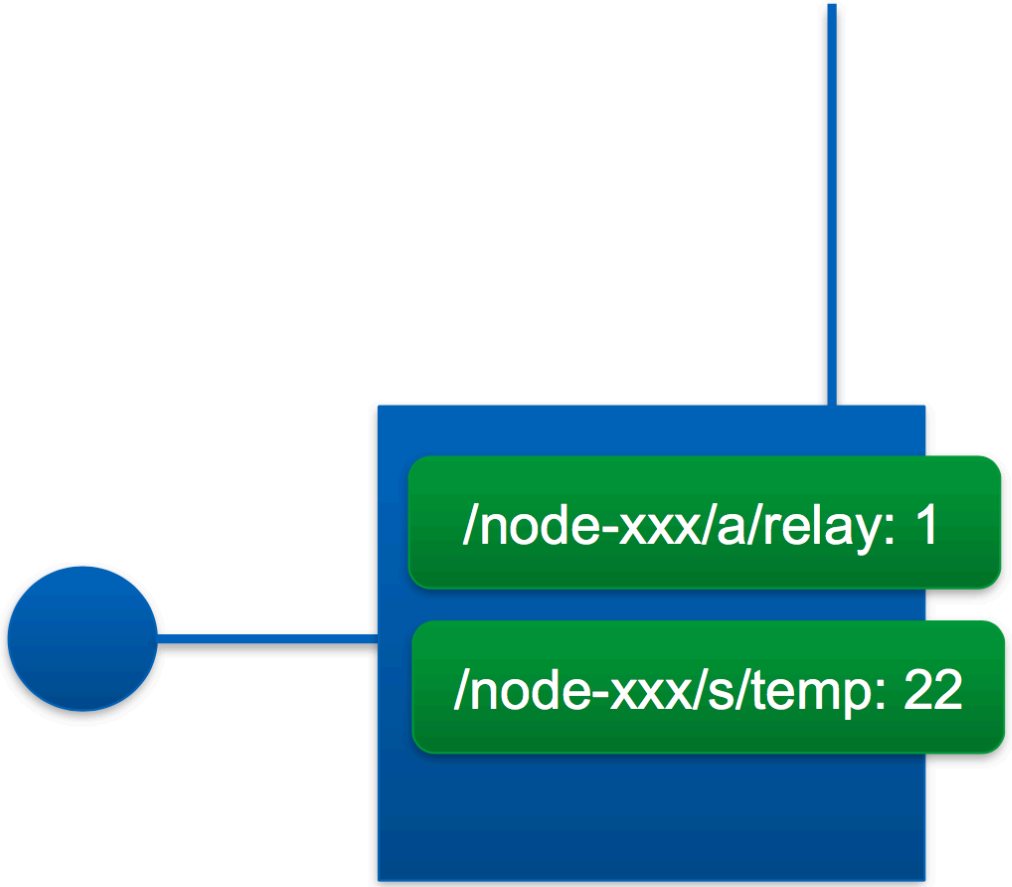
COAP RESOURCE OBSERVATION



COAP A RESTFUL PROTOCOL

- REpresentational State Transfer is a software architectural style for Web client–server
- Resources are represented as URL:
  - “example.com/profile/johnny”
  - “example.com/domain/sensor3/temp1“
- Resources can be retrieved and manipulated using VERBS:
  - GET, POST, PUT, DELETE • Example protocol: HTTP

COAP A RESTFUL PROTOCOL



GET /node-xxx/a/relay
-> 1

GET /node-xxx/s/temp
-> 22

POST /node-xxx/a/relay/1
-> OK

DELETE /node-xxx/a/relay
-> OK

COAP QOS

- Requests and response messages may be marked as “confirmable” or “non confirmable”.
- Confirmable messages must be acknowledged by the receiver with an ack packet.
- Non confirmable messages are “fire and forget”.
- Four message types

•Confirmable: CON

Some messages require an acknowledgement. These messages are called "Confirmable". When no packets are lost, each confirmable message elicits exactly one return message of type Acknowledgement or type Reset.

•Non–Confirmable: NON

Some other messages do not require an acknowledgement. This is particularly true for messages that are repeated regularly for application requirements, such as repeated readings from a sensor where eventual arrival is sufficient. A non-confirmable message always carries either a request or response, as well, and MUST NOT be empty

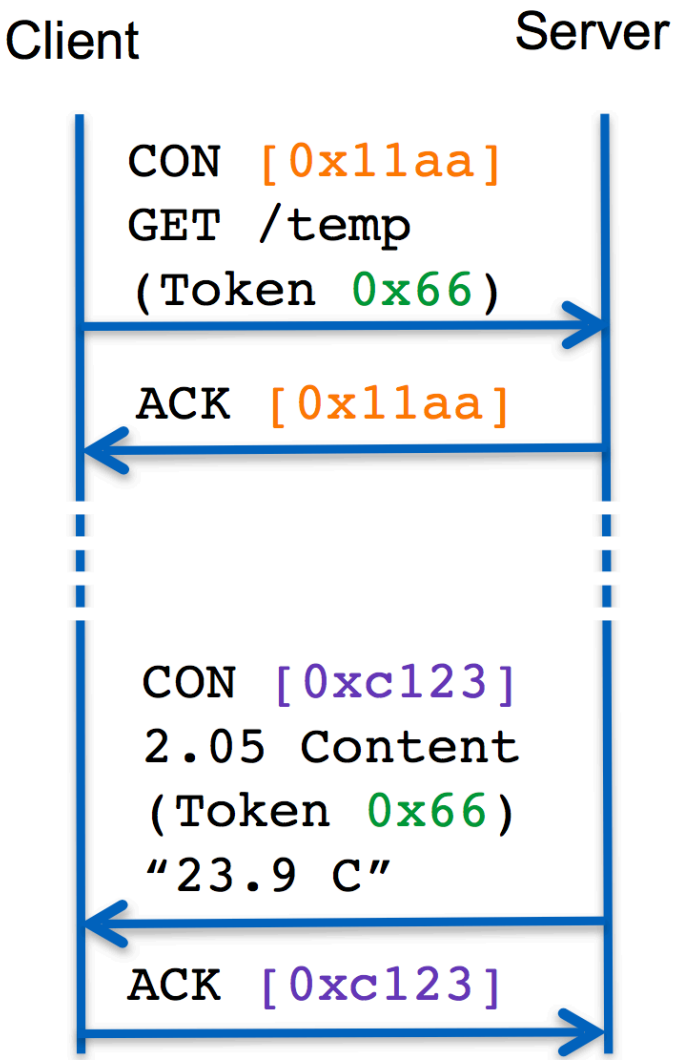
Acknowledgement: ACK

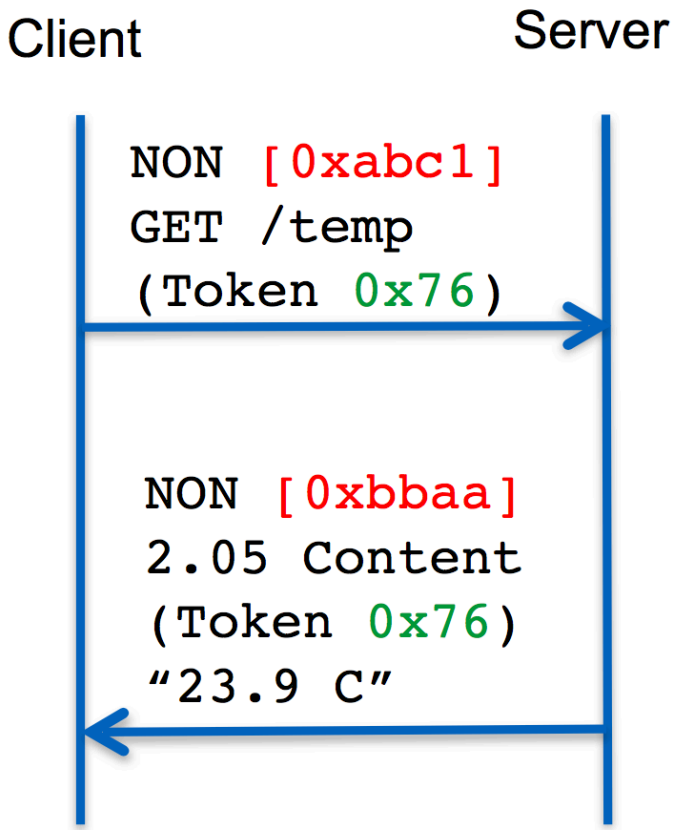
An Acknowledgement message acknowledges that a specific confirmable message (identified by its Message ID) arrived. It does not indicate success or failure of any encapsulated request.

The acknowledgement message MUST echo the Message ID of the confirmable message, and MUST carry a response or be empty.

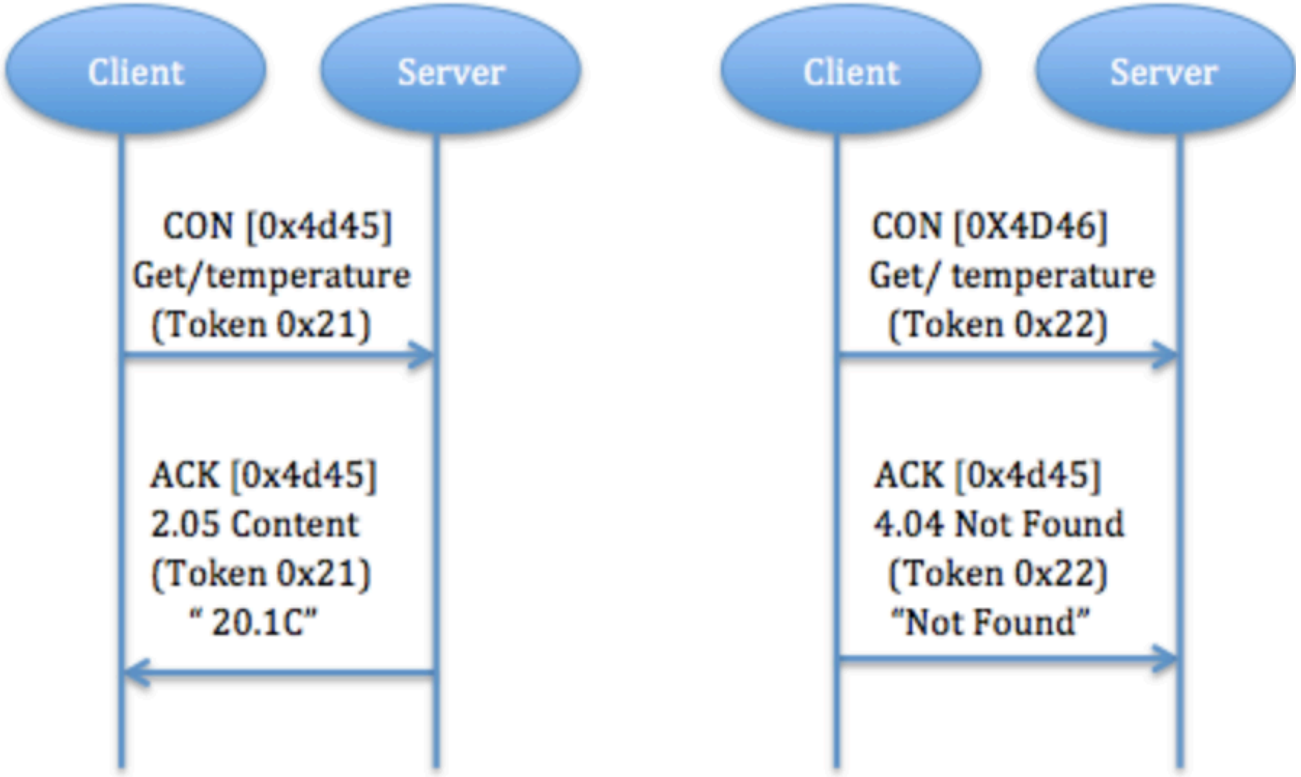
•Reset (3) A Reset message indicates that a specific confirmable message was received, but some context is missing to properly process it. This condition is usually caused when the receiving node has rebooted and has forgotten some state that would be required to interpret the message

COAP REQUEST/RESPONSE



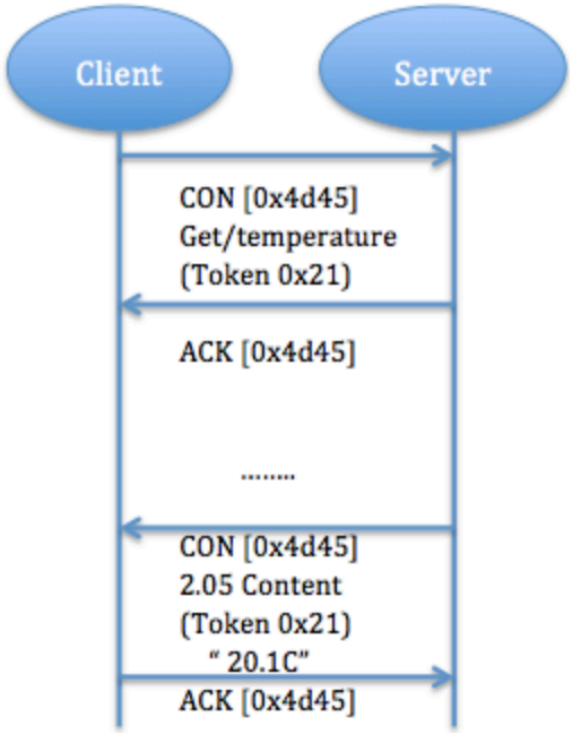


COAP REQUEST/RESPONSE



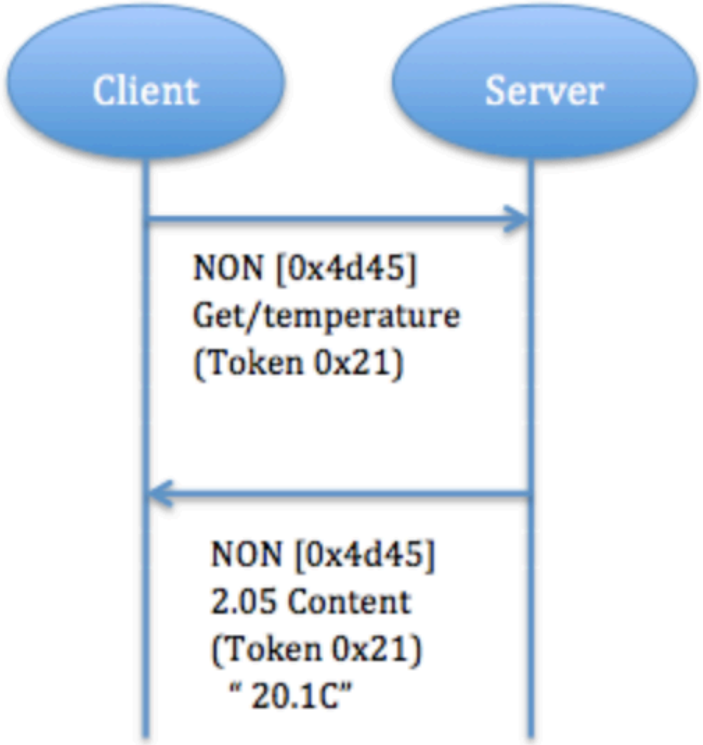
Piggy-backed: Client sends request using CON type or NON type message and receives response ACK with confirmable message immediately. In fig. 5, for successful response, ACK contain response message (identify by using token), for failure response, ACK contain failure response code.

COAP REQUEST/RESPONSE



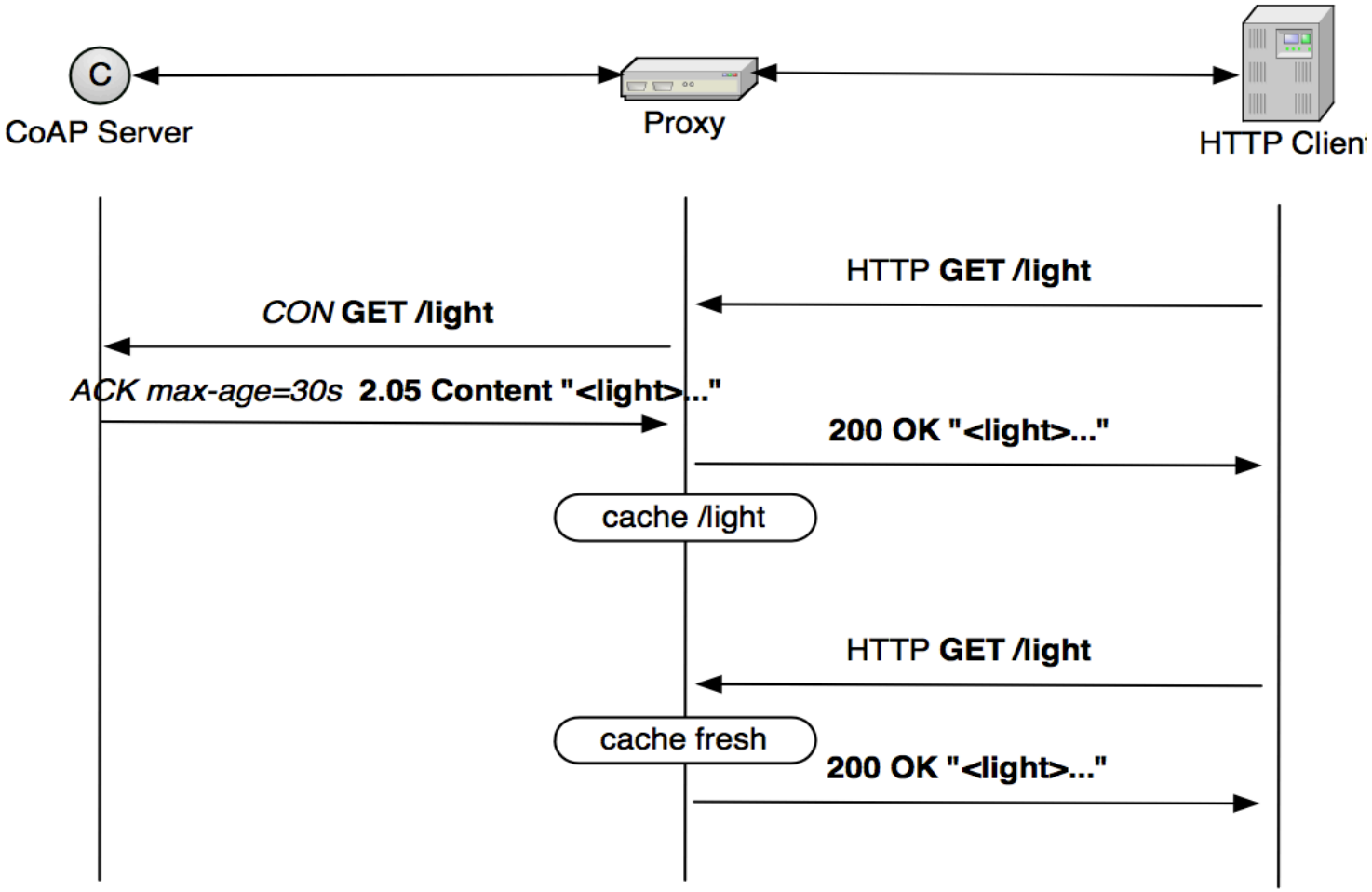
Separate response: If server receive a CON type message but not able to respond to this request immediately, it will send an empty ACK in case of client resend this message. When server ready to respond to this request, it will send a new CON to client and client reply a confirmable message with acknowledgment.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

COAP REQUEST/RESPONSE



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

COAP PROXY AND CACHING

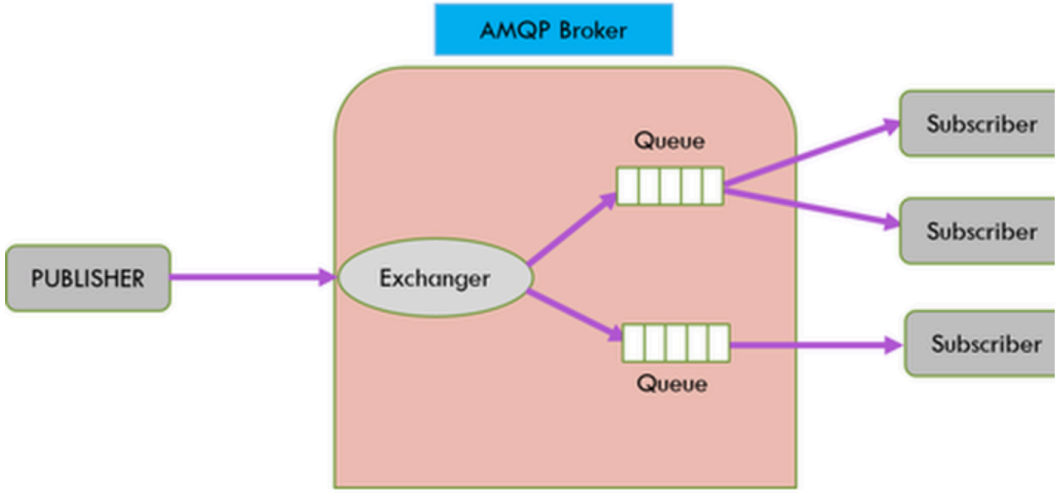


COAP IMPLEMENTATION

libcoap	*Client, Server*	*C*
Californium	*Client, Server*	*Java*
node-coap	*Client , Server*	*Node.js*
Erbium	*Client, Server*	*C*

OTHER APPLICTION LAYER PROTOCOLS

- The Advanced Message Queuing Protocol (AMQP) is an open standard application layer protocol for message-oriented middle ware.
- The features of AMQP are message orientation, queuing, routing (including point-to-point and publish-and-subscribe), reliability and security.
- The protocol runs over TCP/IP network protocol that provide ordered, lossless, bidirectional connections.



OTHER APPLICATION LAYER PROTOCOLS

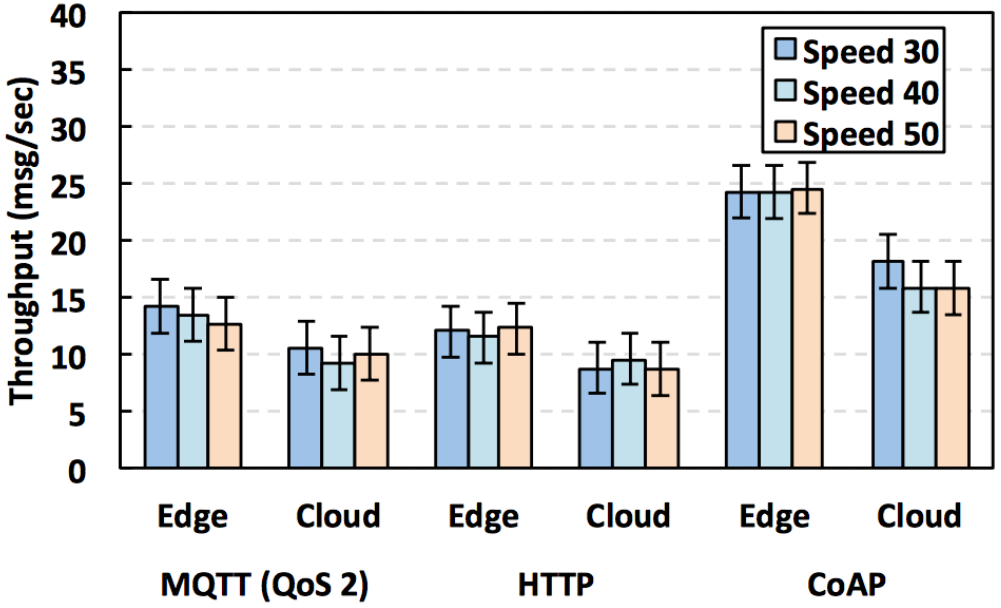
Extensible Messaging and Presence Protocol (**XMPP**) is a communications protocols for message-oriented middleware based on XML (Extensible Markup Language).
Originally was developed for near real-time instant messaging (IM), presence information, and contact list maintenance. Designed to be extensible , the protocol has been used also for publish-subscribe systems, signaling for VoIP, video, file transfer, gaming, the Internet of Things (IoT) applications.
Many XMPP features make it a preferred protocol by most IM applications and relevant within the scope of the IoT.
XMPP is secure and allows for the addition of new applications on top of the core protocols.
The **protocol runs over TCP/IP** network protocol that provide ordered, lossless, bidirectional connections.

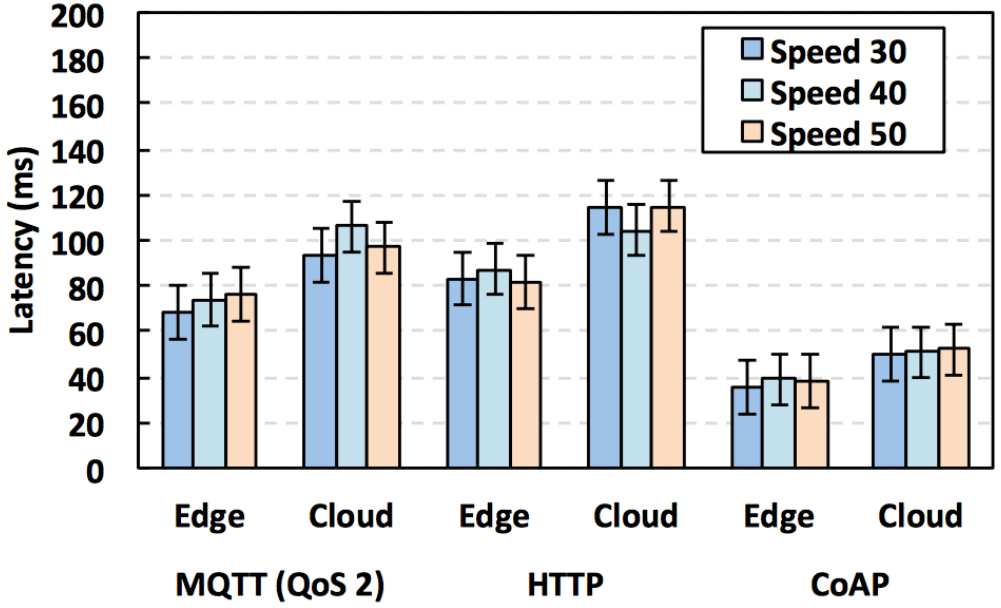
HTTP/COAP/MQTT

Criteria	HTTP	CoAP	MQTT
Architecture	Client/Server	Client/Server or Client/Broker	Client/Broker
Abstraction	Request/Response	Request/Response or Publish/ Subscribe	Publish/Subscribe
Header Size	Undefined	4 Byte	2 Byte
Message size	Large and Undefined (depends on the web server or the programming technology)	Small and Undefined (normally small to fit in single IP datagram)	Small and Undefined (up to 256 MB maximum size)
Semantics/Methods	Get, Post, Head, Put, Patch, Options, Connect, Delete	Get, Post, Put, Delete	Connect, Disconnect, Publish, Subscribe, Unsubscribe, Close
Quality of Service (QoS) /Reliability	Limited (via Transport Protocol - TCP)	Confirmable Message or Non-confirmable Message	QoS 0 - At most once QoS 1 - At least once QoS 2 - Exactly once
Transport Protocol	TCP	UDP, TCP	TCP (MQTT-SN can use UDP)
Security	TLS/SSL	DTLS/IPSEC	TLS/SSL
Default Port	80/443 (TLS/SSL)	5683 (UDP)/5684 (DTLS)	1883/8883 (TLS/SSL)

https://www.researchgate.net/publication/328051666_Service_Provisioning_in_Vehicular_Networks_Through_Edge_and_Cloud_An_Empirical_Analysis

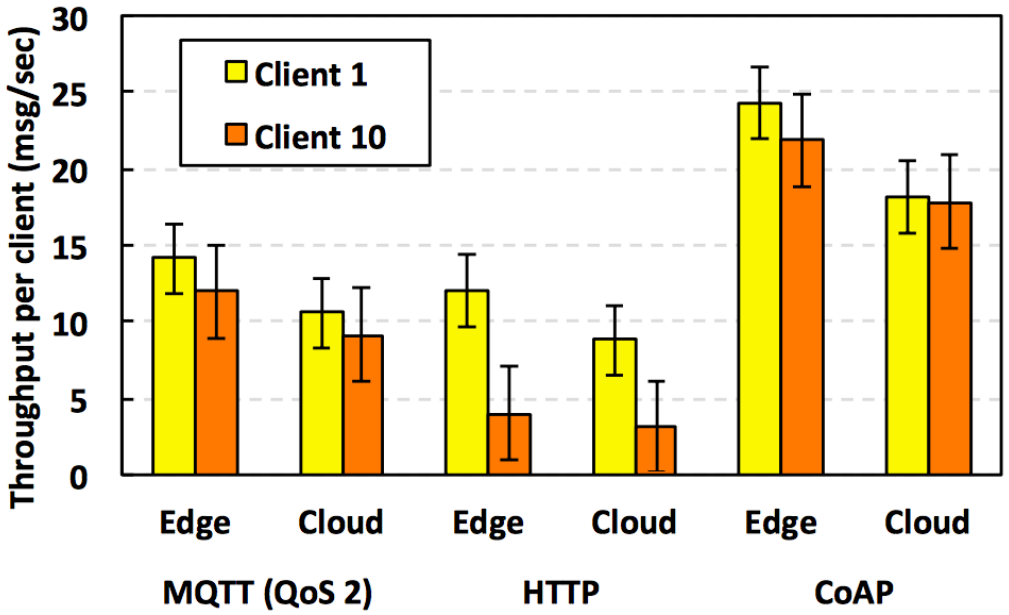
PERFORMANCE OF COAP, MQTT, AND HTTP IN VEHICULAR SCENARIOS

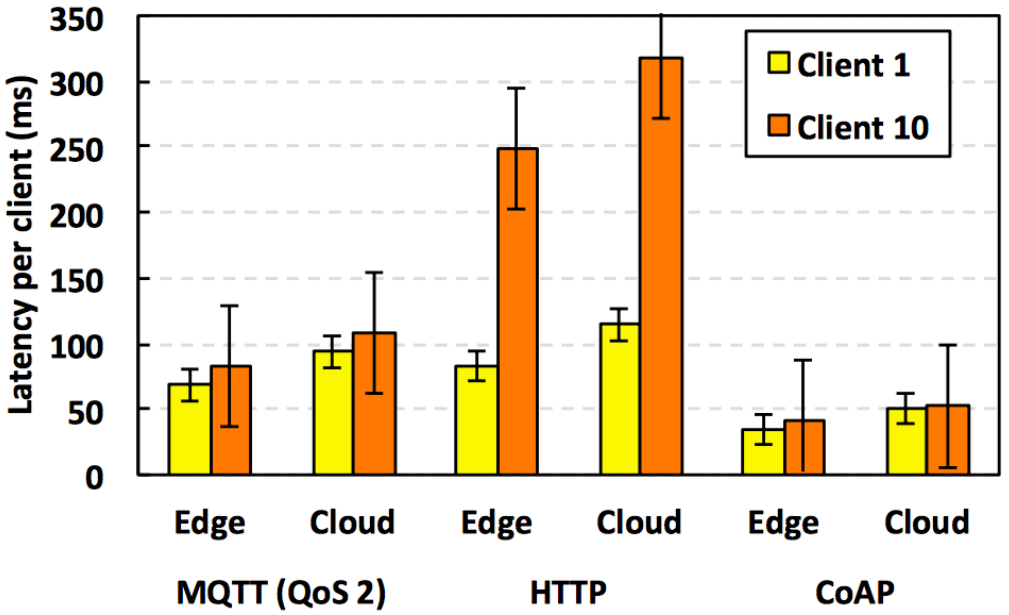




https://www.researchgate.net/publication/328051666_Service_Provisioning_in_Vehicular_Networks_Through_Edge_and_Cloud_An_Empirical_Analysis

PERFORMANCE OF COAP, MQTT, AND HTTP IN VEHICULAR SCENARIOS





https://www.researchgate.net/publication/328051666_Service_Provisioning_in_Vehicular_Networks_Through_Edge_and_Cloud_An_Empirical_Analysis

COMPARISON

Features	MQTT (MQTT-SN)	CoAP	XMPP	AMQP
Transport	TCP/IP UDP/IP (MQTT-SN)	UDP/IP	TCP/IP	TCP/IP
Communication model	Publish/subscribe	Request/Response	Request/Response Publish/subscribe	Publish/subscribe
Security	Medium-optional	Medium-optional	High-Mandatory	High-Mandatory
QoS Level	High	Moderate	Moderate	High
Header size	2 bytes	4 bytes	-	8 bytes
Constrained devices	Yes	Yes	No	No
Power consumption	Less	Medium	High	Medium
Assured delivery level	High	Medium	-	High

Protocol	Communication	Transport	Scalability	Security
RESTful HTTP	Client/server	TCP	Limited	HTTPS
CoAP	Client/server Publish/subscribe	UDP	Excellent	DTLS
MQTT-SN	Publish/subscribe	TCP/UDP	Excellent	TLS/SSL
XMPP	Client/server Publish/subscribe	TCP	Fair	TLS/SSL

Protocol	CoAP	XMPP	RESTful HTTP	MQTT
Transport	UDP	TCP	TCP	TCP
Messaging	Request/Response	Publish/Subscribe Request/Response	Request/Response	Publish/Subscribe Request/Response
2G, 3G, 4G Suitability (1000s nodes)	Excellent	Excellent	Excellent	Excellent
LLN Suitability (1000s nodes)	Excellent	Fair	Fair	Fair
Compute Resources	10Ks RAM/Flash	10Ks RAM/Flash	10Ks RAM/Flash	10Ks RAM/Flash
Success Storied	Utility Field Area Networks	Remote management of consumer white goods	Smart Energy Profile 2 (premise energy management/home services)	Extending enterprise messaging into IoT applications