

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: January 5, 2015

M. Koster  
ARM Limited  
A. Keranen  
J. Jimenez  
Ericsson  
July 4, 2014

Message Queueing in the Constrained Application Protocol (CoAP)  
draft-koster-core-coapmq-00.txt

Abstract

The Constrained Application Protocol, CoAP, and related extensions are intended to support machine-to-machine communication in systems where one or more nodes are resource constrained, in particular for low power wireless sensor networks. This document defines publish-subscribe message queuing functionality for CoAP that extends the capabilities for supporting nodes with long breaks in connectivity and/or up-time.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 5, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Terminology . . . . .	3
3. Architecture . . . . .	4
3.1. RD Server with associated CoAP-MQ Broker . . . . .	4
3.2. Client Endpoint . . . . .	5
3.3. Server Endpoint . . . . .	5
3.4. Publish-Subscribe Topics . . . . .	5
4. CoAP-MQ Registration and discovery . . . . .	6
4.1. Register PubSub Endpoint . . . . .	6
4.2. Unregister Endpoint . . . . .	7
5. CoAP-MQ Functions and Interactions . . . . .	7
5.1. Client Role Endpoint Functions . . . . .	8
5.1.1. Client Endpoint PUBLISH to CoAP-MQ broker . . . . .	8
5.1.2. Client Endpoint SUBSCRIBE, Broker PUBLISH . . . . .	8
5.1.3. Client Endpoint GET from CoAP-MQ Broker . . . . .	9
5.2. Server Role Endpoint Functions . . . . .	9
5.2.1. CoAP-MQ broker SUBSCRIBES to Server Role EP . . . . .	10
5.2.2. CoAP-MQ Broker Publishes to Server Role Endpoint . . . . .	10
5.2.3. CoAP-MQ Broker GET from Server Role Endpoint . . . . .	11
6. Enabling Multiple Publishers . . . . .	11
6.1. Creating a Topic . . . . .	11
6.2. Publishing a Topic from Multiple Publishers . . . . .	12
6.3. Subscribing to a topic with multiple publishers . . . . .	12
7. Sleep-Wakeup Operation and Message Queueing . . . . .	13
8. Security Considerations . . . . .	13
9. IANA Considerations . . . . .	14
9.1. Resource Type value 'core.pubsub.client' . . . . .	14
9.2. Resource Type value 'core.pubsub.server' . . . . .	14
10. Acknowledgements . . . . .	14
11. References . . . . .	15
11.1. Normative References . . . . .	15
11.2. Informative References . . . . .	15
Authors' Addresses . . . . .	15

## 1. Introduction

The Constrained Application Protocol (CoAP) [6] supports machine to machine communication across networks of constrained devices. One important class of constrained devices includes devices that are intended to run for years from a small battery, or by scavenging

energy from their environment. These devices spend most of their time in a sleeping state with no network connectivity.

Devices may also have limited reachability due to certain middle-boxes, such as Network Address Translators (NATs) or firewalls. Such devices must communicate using a client role, whereby the endpoint is responsible for initiating communication.

This document specifies the means for nodes with limited reachability to communicate using simple extensions to CoAP and the CoRE Resource Directory [4]. The extensions enable publish-subscribe communication using a Message Queue (MQ) broker node that enables store-and-forward messaging between two or more nodes.

The mechanisms specified in this document are meant to address key design requirements from earlier CoRE drafts covering sleepy node support and mirror server.

## 2. Terminology

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and 'OPTIONAL' in this specification are to be interpreted as described in [1].

This specification requires readers to be familiar with all the terms and concepts that are discussed in [8] and [2]. Readers should also be familiar with the terms and concepts discussed in [6] and [4]. The URI template format, see [3], is used to describe the REST interfaces defined in this specification.

The following entities are used in this specification:

**CoAP Message Queue (CoAP-MQ) Service** A service provided by a node or system where CoAP messages sent by one endpoint to another are queued (stored) by intermediate node(s) and forwarded only when suitable, e.g., when the message recipient endpoint is not sleeping.

**CoAP-MQ Broker** A server node capable of storing messages to and from other nodes and able to match subscriptions and publications in order to route messages to right destinations.

**CoAP-MQ Endpoint** An endpoint that implements the MQ function set defined in Section 5. A CoAP-MQ endpoint has two potential roles, CoAP-MQ client and CoAP-MQ server

This specification makes use of the following additional terminology:

**CoAP Message Queue (CoAP-MQ) Service** A service provided by a node or system where CoAP messages sent by one endpoint to another are queued (stored) by intermediate node(s) and forwarded only when suitable, e.g., when the message recipient endpoint is not sleeping.

**CoAP-MQ Broker** A server node capable of storing messages to and from other nodes and able to match subscriptions and publications in order to route messages to right destinations.

**CoAP-MQ function set** A group of well-known REST resources that together provide the CoAP-MQ service.

**CoAP-MQ Endpoint** An endpoint that implements the CoAP-MQ function set. A CoAP-MQ endpoint has two potential roles, CoAP-MQ client and CoAP-MQ server.

**Publish-Subscribe (pub-sub)** A messaging paradigm where messages are published (e.g., to a broker) and potential receivers can subscribe to receive the messages.

**Topic** In Publish-Subscribe systems a topic is a unique identifying string for a particular item or object being published and/or subscribed to.

### 3. Architecture

#### 3.1. RD Server with associated CoAP-MQ Broker

Figure 1 shows an example architecture of a CoAP-MQ capable service. A Resource Directory service accepts registrations and registration updates from one or more endpoints and hosts a resource discovery service for one or more web application clients. State information is updated from the endpoints to the CoAP-MQ broker. Web clients subscribe to the state of the endpoint from the CoAP-MQ broker, and publish updates to the endpoint state through the CoAP-MQ broker. The CoAP-MQ broker performs a store-and-forward function between web clients and the CoAP-MQ capable endpoints. The CoAP-MQ broker is also responsible for acting as a proxy, returning the last published value to web clients or other endpoints on behalf endpoints that are sleeping.

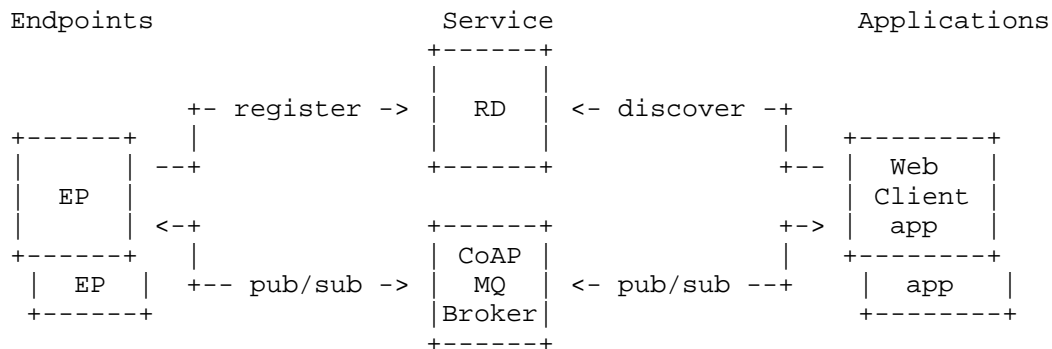


Figure 1: CoAP MQ Architecture

### 3.2. Client Endpoint

Client endpoints initiate all interactions with the RD and MQ broker. If the endpoint is an actuator it will need to either use CoAP Observe [I-D.ietf-core-observe] or periodically poll the MQ broker to check for updates. A CoAP-MQ client endpoint **MUST** use CoAP PUT operations to update its state on the MQ broker. An endpoint **SHOULD** update the RD periodically to indicate that it is still alive even if it has no pending data updates. Endpoints can operate in the client role even if not directly reachable from the CoAP-MQ broker or RD server.

### 3.3. Server Endpoint

Server endpoint interactions require the CoAP-MQ broker to perform the client role, initiating interaction with the server endpoint. The CoAP-MQ broker **MAY** then use PUT operations to update state at the server endpoint, and **MAY** use GET or GET+Observe to subscribe to resources at the endpoint. Server mode endpoints are required to be reachable from the CoAP-MQ broker. In a network containing both client and server endpoints, client endpoints **MAY** subscribe to server endpoints directly, in broker-less configurations, using RD or core-link-format metadata in .well-known/core to discover the CoAP-MQ capabilities and using GET+Observe to subscribe to the desired topics.

### 3.4. Publish-Subscribe Topics

Topic are strings used to identify particular resources and objects in publish-subscribe systems. Topics are conventionally formed as a hierarchy, e.g. "/sensors/weather/barometer/pressure".

Implementations are free to map topics to resources, reusing existing resource addressing schemes.

#### 4. CoAP-MQ Registration and discovery

An endpoint wishing to use a CoAP-MQ broker registers with an RD server that advertises a service having the the "core.mq" attribute. This indicates that there is a CoAP-MQ broker at the location returned by the discovery query as shown in Figure 2. The endpoint registers topics using the "rt=core.pubsub.client" or "core.pubsub.server" (or both) attributes to indicate intention to use CoAP-MQ and which roles are supported.

A server that implements a CoAP\_MQ broker MAY advertize this capability by registering the rt="core.mq" with an associated Resource Directory. If a server advertizes as a CoAP-MQ Broker, it MUST support the transactions described in section 5 of this document. As server that implements the CoAP-MQ Broker MAY also implement sleeping endpoint and message queueing support referred to in Section 6 of this document.

##### 4.1. Register PubSub Endpoint

Figure 2 shows the flow of the registration operation. Discovery proceeds as per CoRE Resource Directory[I-D.ietf-core-resource-directory-01]. When an endpoint wishes to use CoAP-MQ, it discovers the rt="core.mq" attribute at the RD service associated with the CoAP-MQ broker and registers its CoAP-MQ resources with the RD server by registering topics having the rt="core.pubsub" attribute. Topics are created using an initial POST operation to the registered topic or any valid sub-topic. For example, if the registered topic is "/sensors/weather", the sub-topic "/sensors/weather/barometer" is created using a POST to "/mq/sensors/weather/barometer". An implementation MAY mix CoAP-MQ resources and CoAP REST resources on the same endpoint. Endpoint registration proceeds as per normal RD registration.

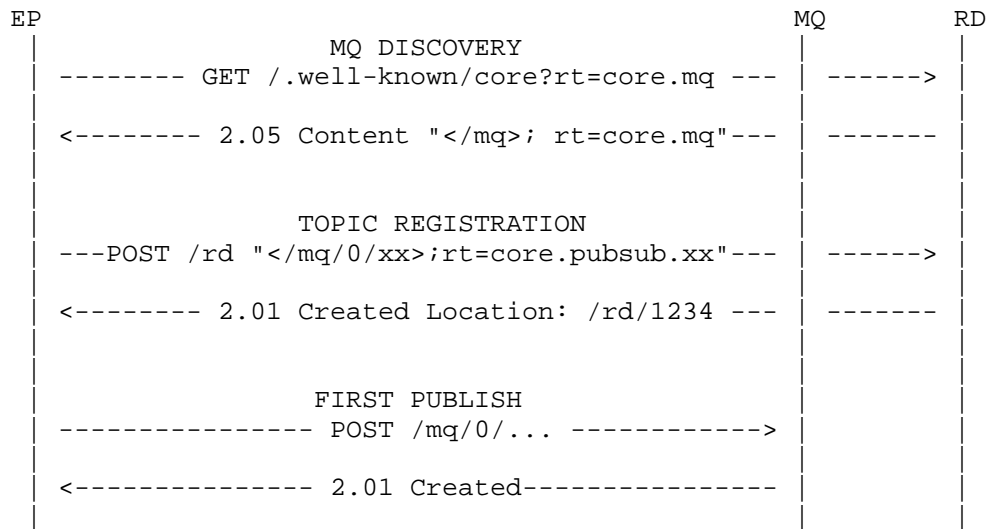


Figure 2: Discovery and Registration

#### 4.2. Unregister Endpoint

CoAP-MQ endpoints indicate the end of their registration tenure by either explicitly unregistering, as in Figure 3, or allowing the lifetime of the previous registration to expire.

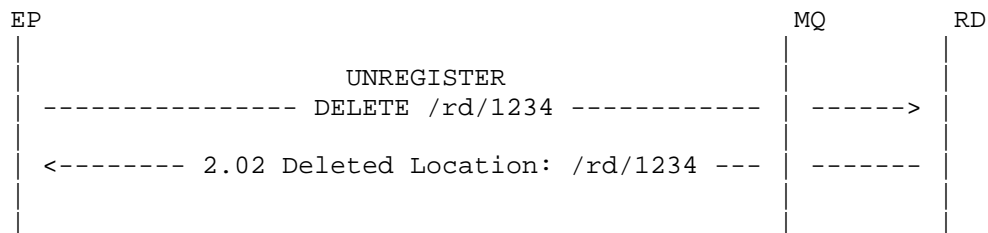


Figure 3: Unregister Endpoint

### 5. CoAP-MQ Functions and Interactions

This section describes the transaction flows and interactions between CoAP-MQ endpoints and CoAP-MQ brokers. Client endpoint functions are used by endpoints implementing the client role, for example to enable sleep/wakeup and partial connectivity. Server role endpoint functions are used by endpoints implementing the server role, for

example always on, reachable, endpoints. An endpoint implementation MAY support both client role and server role at an endpoint. A CoAP-MQ broker MUST implement support for both client role and server role endpoints.

### 5.1. Client Role Endpoint Functions

This section describes the transaction flows and interactions between CoAP-MQ endpoints and CoAP-MQ brokers where the endpoint supports the client role. A client registering the "core.pubsub.client" attribute MUST support the client role endpoint functions and interactions described in this section.

#### 5.1.1. Client Endpoint PUBLISH to CoAP-MQ broker

Client endpoint PUBLISH updates to CoAP-MQ broker.

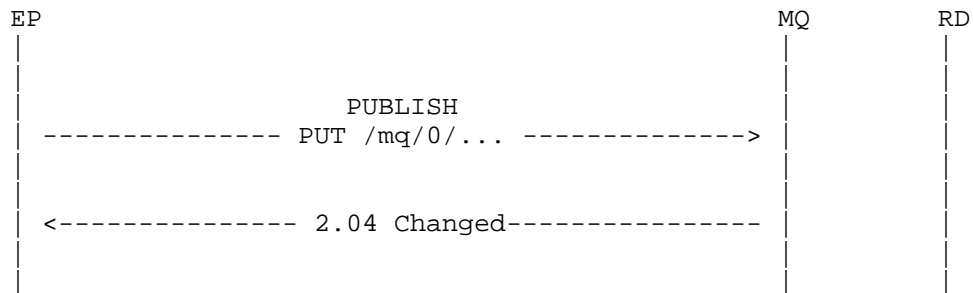


Figure 4: Client Role PUBLISH from EP to Broker

#### 5.1.2. Client Endpoint SUBSCRIBE, Broker PUBLISH

Client mode endpoint subscribes to the topic at the CoAP-MQ broker using GET+Observe. Published updates to the CoAP-MQ broker are published to the Endpoint using Observe response tokens. Client endpoint MAY update actuator or resource based on received values associated with responses. A CoAP-MQ broker MUST publish updates to subscribed endpoints upon receiving published updates on the associated topics.



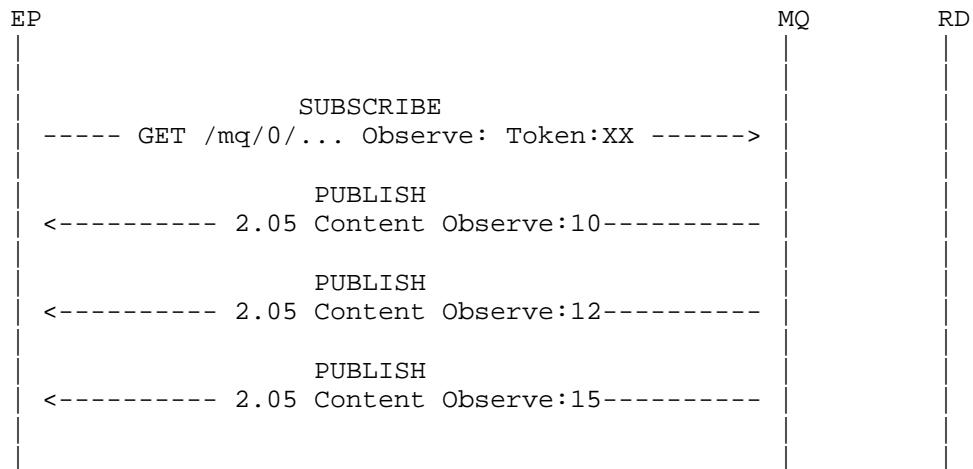


Figure 5: Client Role Endpoint SUBSCRIBE, Broker PUBLISH to Endpoint

#### 5.1.3. Client Endpoint GET from CoAP-MQ Broker

Client mode endpoint MAY issue GET to topic without Observe as needed to obtain last published state from the CoAP-MQ broker.

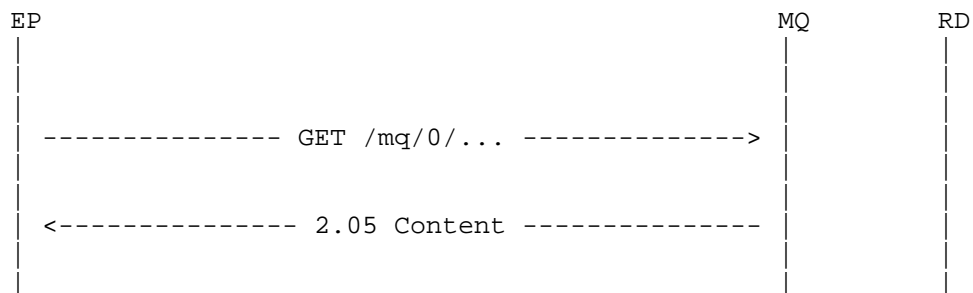


Figure 6: Client EP GET from CoAP-MQ Broker

#### 5.2. Server Role Endpoint Functions

This section describes the transaction flows and interactions between CoAP-MQ endpoints and CoAP-MQ brokers where the endpoint supports the server role. An endpoint registering the "core.pubsub.server" attribute MUST support these functions and interactions.

## 5.2.1. CoAP-MQ broker SUBSCRIBES to Server Role EP

The server mode endpoint requires the CoAP-MQ broker to act as a client and subscribe to a resource on the endpoint using GET + Observe. A CoAP-MQ broker MAY subscribe to topics registered by a server role endpoint at any time. A CoAP-MQ broker MUST subscribe to a topic registered by a server role endpoint upon receiving a subscription on the associated topic. A CoAP-MQ broker MUST forward state updates received from a publishing endpoint to all endpoints subscribed on the associated topic. Figure 7 shows the flow of a CoAP-MQ Broker subscribing to a server role endpoint.

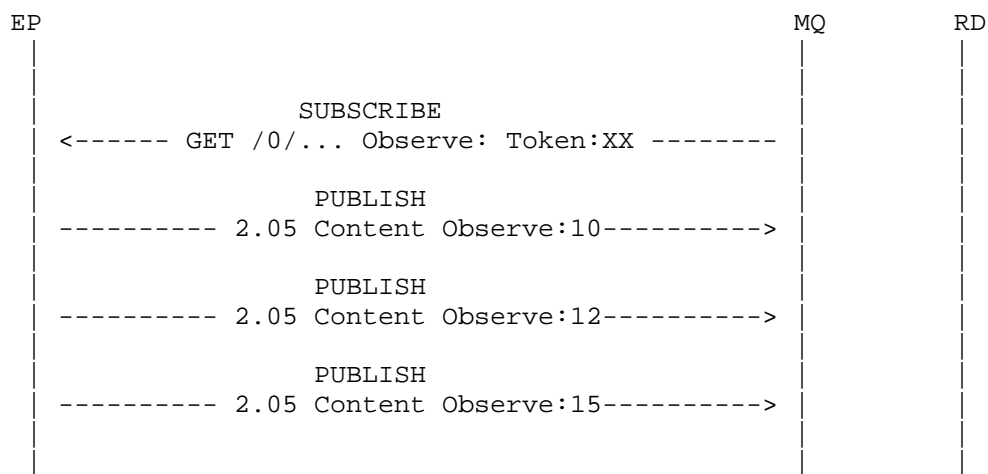


Figure 7: Broker SUBSCRIBE to Server Role EP

## 5.2.2. CoAP-MQ Broker Publishes to Server Role Endpoint

CoAP-MQ broker MUST update server mode endpoint using PUT when upon receiving updates published on the associated topics. Endpoint server MAY update actuator or resource upon receiving published state updates from the broker.

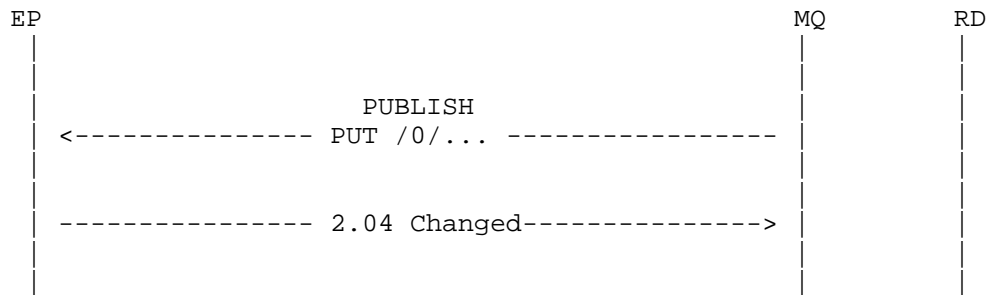


Figure 8: Broker PUBLISH to Server Role EP

### 5.2.3. CoAP-MQ Broker GET from Server Role Endpoint

CoAP-MQ broker MAY issue GET without Observe as needed to obtain state update from the server role endpoint.

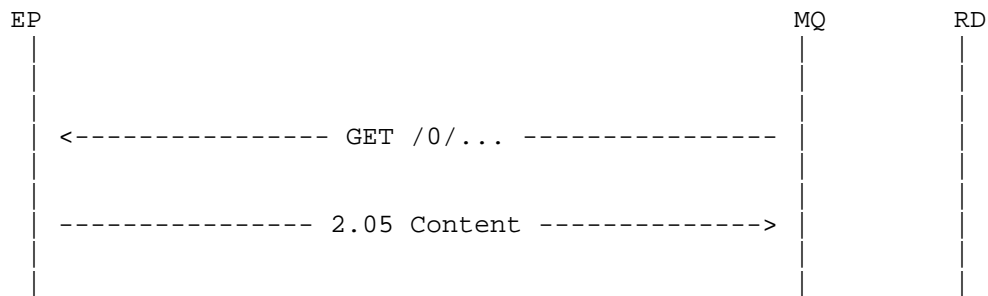


Figure 9: Broker GET from Server Role Endpoint

## 6. Enabling Multiple Publishers

### 6.1. Creating a Topic

After registration of the EP in the RD and discovering the CoAP-MQ function, a designated EP acting as publisher for a particular topic is responsible for creating such topic. To do so, it will have to register the new topic in the RD and create it on the MQ function by doing a first publication as shown in Figure 2.

After the topic has been created in the CoAP-MQ broker, the broker will be responsible of hosting this resource and to queue messages published on it as explained in Section 5

## 6.2. Publishing a Topic from Multiple Publishers

After the topic has been registered in the RD and is created in the CoAP-MQ broker, any device with the right access permissions can publish on that topic by using the topic field. For example in the following diagram, both EP1 and EP2 update the same topic that EP3 has previously subscribed to.

After the topic has been created in the CoAP-MQ Broker, the broker will be responsible of hosting this resource and to queue messages published on it as explained in Section 5

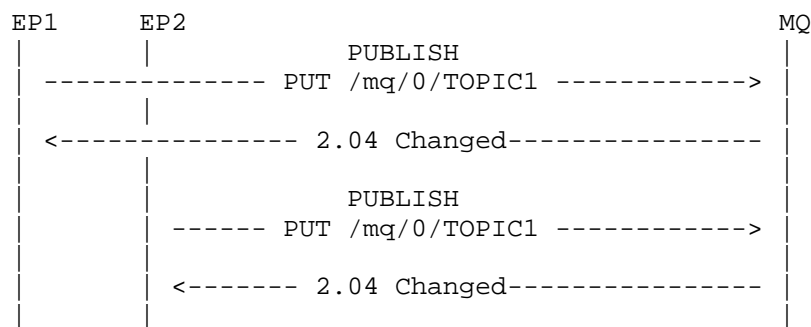


Figure 10: Multiple CoAP-MQ EPs PUBLISH to Broker

## 6.3. Subscribing to a topic with multiple publishers

Subscription to this topic is the same as in Section 5, since it acts as any other resource. Following the previous example, if EP3 is subscribed to the shared topic, it should receive two updates from both EP1 and EP2.

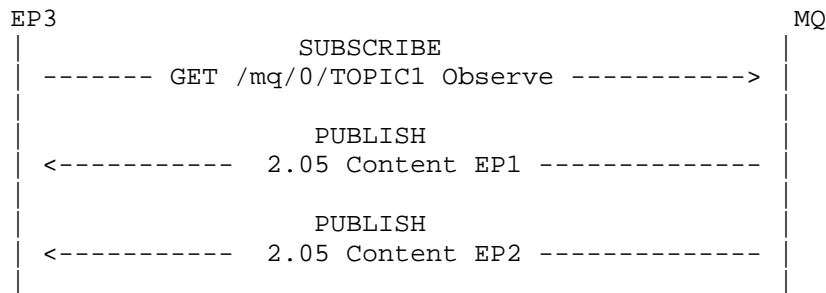


Figure 11: CoAP-MQ Endpoint SUBSCRIBE to Broker

## 7. Sleep-Wakeup Operation and Message Queueing

A CoAP-MQ broker MAY implement support for sleeping endpoints and queueing of messages as provided for in [7]

## 8. Security Considerations

CoAP-MQ re-uses CoAP [6], CoRE Resource Directory [4], and Web Linking [8] and therefore the security considerations of those documents also apply to this specification. Additionally, a CoAP-MQ broker and the endpoints SHOULD authenticate each other and enforce access control policies. A malicious EP could subscribe to data it is not authorized to or mount a denial of service attack against the broker by publishing a large number of resources. The authentication can be performed using the already standardized DTLS offered mechanisms, such as certificates. DTLS also allows communication security to be established to ensure integrity and confidentiality protection of the data exchanged between these relevant parties. Provisioning the necessary credentials, trust anchors and authorization policies is non-trivial and subject of ongoing work.

The use of a CoAP-MQ broker introduces challenges for the use of end-to-end security between the end device and the cloud-based server infrastructure since brokers terminate the exchange. While running separate DTLS sessions from the EP to the broker and from broker to the web application protects confidentiality on those paths, the client/server EP does not know whether the commands coming from the broker are actually coming from the client web application. Similarly, a client web application requesting data does not know whether the data originated on the server EP. For scenarios where end-to-end security is desirable the use of application layer security is unavoidable. Application layer security would then provide a guarantee to the client EP that any request originated at the client web application. Similarly, integrity protected sensor

data from a server EP will also provide guarantee to the client web application that the data originated on the EP itself. The protected data can also be verified by the intermediate broker ensuring that it stores/caches correct request/response and no malicious messages/requests are accepted. The broker would still be able to perform aggregation of data/requests collected.

Depending on the level of trust users and system designers place in the CoAP-MQ broker, the use of end-to-end encryption may also be envisioned. The CoAP-MQ broken would then only be able to verify the request/response message/commands and store-and-forward without being able to inspect the content. The solution for providing application layer security will depend on the utilized data encoding. For example, with a JSON-based data encoding the work from the JOSE working group could be re-used. Distribution of the credentials for accomplishing end-to-end security might introduce challenges if previously unknown parties need to exchange data.

## 9. IANA Considerations

This document registers two attribute values in the Resource Type (rt=) registry established with RFC 6690 [2].

### 9.1. Resource Type value 'core.pubsub.client'

- o Attribute Value: core.pubsub.client
- o Description: Section X of [[This document]]
- o Reference: [[This document]]
- o Notes: None

### 9.2. Resource Type value 'core.pubsub.server'

- o Attribute Value: core.pubsub.server
- o Description: Section Y of [[This document]]
- o Reference: [[This document]]
- o Notes: None

## 10. Acknowledgements

Hannes Tschofenig, Zach Shelby

## 11. References

### 11.1. Normative References

- [1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [2] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.
- [3] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, March 2012.
- [4] Shelby, Z., Bormann, C., and S. Krco, "CoRE Resource Directory", draft-ietf-core-resource-directory-01 (work in progress), December 2013.
- [5] Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-14 (work in progress), June 2014.
- [6] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.
- [7] Open Mobile Alliance, "OMA LightweightM2M v1.0", <http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases/oma-lightweightm2m-v1-0>, 12 2013.

### 11.2. Informative References

- [8] Nottingham, M., "Web Linking", RFC 5988, October 2010.

## Authors' Addresses

Michael Koster  
ARM Limited

Email: Michael.Koster@arm.com

Ari Keranen  
Ericsson

Email: ari.keranen@ericsson.com

Jaime Jimenez  
Ericsson

Email: [jaime.jimenez@ericsson.com](mailto:jaime.jimenez@ericsson.com)