Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 7, 2019

M. Koster
SmartThings
A. Keranen
J. Jimenez
Ericsson

January 3, 2019

Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)

draft-ietf-core-coap-pubsub-06

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 a publish-subscribe Broker for CoAP that extends the capabilities of CoAP for supporting nodes with long breaks in connectivity and/or up-time.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] supports machine-to-machine communication across networks of constrained devices. CoAP uses a request/response model where clients make requests to servers in order to request actions on resources. Depending on the situation the same device may act either as a server, a client, or both.

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 have limited

reachability because they 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 middle-boxes often prevent connecting to a device from the Internet unless the connection was initiated by the device.

For some applications the client/server and request/response communication model is not optimal but publish-subscribe communication with potentially many senders and/or receivers and communication via topics rather than directly with endpoints may fit better.

This document specifies simple extensions to CoAP for enabling publish-subscribe communication using a Broker node that enables store-and-forward messaging between two or more nodes. This model facilitates communication of nodes with limited reachability, enables simple many-to-many communication, and eases integration with other publish-subscribe systems.

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 [RFC2119].

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC5988] and [RFC6690]. Readers should also be familiar with the terms and concepts discussed in [RFC7252] and [I-D.ietf-core-resource-directory]. The URI template format [RFC6570] is used to describe the REST API defined in this specification.

This specification makes use of the following additional terminology:

Publish-Subscribe (pub/sub): A messaging paradigm where messages are published to a Broker and potential receivers can subscribe to the Broker to receive messages. The publishers do not (need to) know where the message will be eventually sent: the publications and subscriptions are matched by a Broker and publications are delivered by the Broker to subscribed receivers.

CoAP pub/sub service: A group of REST resources, as defined in this document, which together implement the required features of this specification.

CoAP pub/sub Broker: A server node capable of receiving messages (publications) from and sending messages to other nodes, and able

to match subscriptions and publications in order to route messages to the right destinations. The Broker can also temporarily store publications to satisfy future subscriptions and pending notifications.

CoAP pub/sub Client: A CoAP client which is capable of publish or subscribe operations as defined in this specification.

Topic: A unique identifier for a particular item being published and/or subscribed to. A Broker uses the topics to match subscriptions to publications. A topic is a valid CoAP URI as defined in [RFC7252]

3. Architecture

3.1. CoAP Pub/sub Architecture

Figure 1 shows the architecture of a CoAP pub/sub service. CoAP pub/sub Clients interact with a CoAP pub/sub Broker through the CoAP pub/sub REST API which is hosted by the Broker. State information is updated between the Clients and the Broker. The CoAP pub/sub Broker performs a store-and-forward of state update representations between certain CoAP pub/sub Clients. Clients Subscribe to topics upon which representations are Published by other Clients, which are forwarded by the Broker to the subscribing clients. A CoAP pub/sub Broker may be used as a REST resource proxy, retaining the last published representation to supply in response to Read requests from Clients.

```
               Clients       pub/sub         Broker
               +-------+         |
               | CoAP  |         |
               |pub/sub|---------|------+
               |Client |         |      |    +-------+
               +-------+         |      +----| CoAP  |
                                 |           |pub/sub|
               +-------+         |      +----|Broker |
               | CoAP  |         |      |    +-------+
               |pub/sub|---------|------+
               |Client |         |
               +-------+         |
```

Figure 1: CoAP pub/sub Architecture

3.2. CoAP Pub/sub Broker

A CoAP pub/sub Broker is a CoAP Server that exposes a REST API for clients to use to initiate publish-subscribe interactions. Avoiding the need for direct reachability between clients, the Broker only needs to be reachable from all clients. The Broker also needs to have sufficient resources (storage, bandwidth, etc.) to host CoAP resource services, and potentially buffer messages, on behalf of the clients.

3.3. CoAP Pub/sub Client

A CoAP pub/sub Client interacts with a CoAP pub/sub Broker using the CoAP pub/sub REST API defined in this document. Clients initiate interactions with a CoAP pub/sub Broker. A data source (e.g., sensor clients) can publish state updates to the Broker and data sinks (e.g., actuator clients) can read from or subscribe to state updates from the Broker. Application clients can make use of both publish and subscribe in order to exchange state updates with data sources and data sinks.

3.4. CoAP Pub/sub Topic

The clients and Broker use topics to identify a particular resource or object in a publish-subscribe system. Topics are conventionally formed as a hierarchy, e.g. "/sensors/weather/barometer/pressure" or "/EP-33543/sen/3303/0/5700". The topics are hosted by a Broker and all the clients using the Broker share the same namespace for topics. Every CoAP pub/sub topic has an associated link, consisting of a reference path on the Broker using URI path [RFC3986] construction and link attributes [RFC6690]. Every topic is associated with zero or more stored representations and a content-format specified in the link. A CoAP pub/sub topic value may alternatively consist of a collection of one or more sub-topics, consisting of links to the sub-topic URIs and indicated by a link-format content-format. Sub-topics are also topics and may have their own sub-topics, forming a tree structure of unique paths that is implemented using URIs. The full URI of a topic includes a scheme and authority for the Broker, for example "coaps://10.0.0.13:5684/EP-33543/sen/3303/0/5700".

3.5. brokerless Pub/sub

Figure 2 shows an arrangement for using CoAP pub/sub in a "Brokerless" configuration between peer nodes. Nodes in a Brokerless system may act as both Broker and client. A node that supports Broker functionality may be pre-configured with topics that expose services and resources. Brokerless peer nodes can be mixed with client and Broker nodes in a system with full interoperability.

```
                       Peer         pub/sub         Peer
                     +-------+         |         +-------+
                     | CoAP  |         |         | CoAP  |
                     |pub/sub|---------|---------|pub/sub|
                     |Client |         |         |Broker |
                     +-------+         |         +-------+
                     | CoAP  |         |         | CoAP  |
                     |pub/sub|---------|---------|pub/sub|
                     |Broker |         |         |Client |
                     +-------+         |         +-------+
```

Figure 2: Brokerless pub/sub

4. CoAP Pub/sub REST API

This section defines the REST API exposed by a CoAP pub/sub Broker to pub/sub Clients. The examples throughout this section assume the use of CoAP [RFC7252]. A CoAP pub/sub Broker implementing this specification SHOULD support the DISCOVERY, CREATE, PUBLISH, SUBSCRIBE, UNSUBSCRIBE, READ, and REMOVE operations defined in this section. Optimized implementations MAY support a subset of the operations as required by particular constrained use cases.

4.1. DISCOVERY

CoAP pub/sub Clients discover CoAP pub/sub Brokers by using CoAP Simple Discovery or through a Resource Directory (RD) [I-D.ietf-core-resource-directory]. A CoAP pub/sub Broker SHOULD indicate its presence and availability on a network by exposing a link to the entry point of its pub/sub API at its .well-known/core location [RFC6690]. A CoAP pub/sub Broker MAY register its pub/sub REST API entry point with a Resource Directory. Figure 3 shows an example of a client discovering a local pub/sub API using CoAP Simple Discovery. A Broker wishing to advertise the CoAP pub/sub API for Simple Discovery or through a Resource Directory MUST use the link relation rt=core.ps. A Broker MAY advertise its supported content formats and other attributes in the link to its pub/sub API.

A CoAP pub/sub Broker MAY offer a topic discovery entry point to enable Clients to find topics of interest, either by topic name or by link attributes which may be registered when the topic is created. Figure 4 shows an example of a client looking for a topic with a resource type (rt) of "temperature" using Discover. The client then receives the URI of the resource and its content-format. A pub/sub Broker wishing to advertise topic discovery MUST use the relation rt=core.ps.discover in the link.

A CoAP pub/sub Broker MAY provide topic discovery functionality through the .well-known/core resource. Links to topics may be exposed at .well-known/core in addition to links to the pub/sub API. Figure 5 shows an example of topic discovery through .well-known/core.

Topics in the broker may be created in hierarchies (see {create}) with parent topics having sub-topics. For a discovery the broker may choose to not expose the sub-topics in order to limit amount of topic links sent in a discovery response. The client can then perform discovery for the parent topics it wants to discover the sub-topics.

The DISCOVER interface is specified as follows:

Interaction: Client -> Broker

Method: GET

URI Template: {+ps}/{+topic}{?q*}

URI Template Variables: ps := Pub/sub REST API entry point (optional). The entry point of the pub/sub REST API, as obtained from discovery, used to discover topics.

topic := The desired topic to return links for (optional).

q := Query Filter (optional). MAY contain a query filter list as per [RFC6690] Section 4.1.

Content-Format: application/link-format

The following response codes are defined for the DISCOVER operation:

Success: 2.05 "Content" with an application/link-format payload containing one or more matching entries for the Broker resource. A pub/sub Broker SHOULD use the value "/ps/" for the base URI of the pub/sub API wherever possible.

Failure: 4.04 "Not Found" is returned in case no matching entry is found for a unicast request.

Failure: 4.00 "Bad Request" is returned in case of a malformed request for a unicast request.

Failure: No error response to a multicast request.

```
          Client                                           Broker
            |                                                |
            | ------ GET /.well-known/core?rt=core.ps ---->>|
            | -- Content-Format: application/link-format ---|
            |                                                |
            | <<--- 2.05 Content                             |
            | </ps/>;rt=core.ps;rt=core.ps.discover;ct=40 --|
            |                                                |
```

Figure 3: Example of DISCOVER pub/sub function

```
          Client                                           Broker
            |                                                |
            | ---------- GET /ps/?rt="temperature" ------->>|
            |    Content-Format: application/link-format    |
            |                                                |
            | <<-- 2.05 Content                              |
            |   </ps/currentTemp>;rt="temperature";ct=50 ---|
            |                                                |
```

Figure 4: Example of DISCOVER topic

```
          Client                                           Broker
            |                                                |
            | -------- GET /.well-known/core?ct=50 ------->>|
            |    Content-Format: application/link-format    |
            |                                                |
            | <<-- 2.05 Content                              |
            |   </ps/currentTemp>;rt="temperature";ct=50 ---|
            |                                                |
```

Figure 5: Example of DISCOVER topic

4.2. CREATE

A CoAP pub/sub broker SHOULD allow Clients to create new topics on the broker using CREATE. Some exceptions are for fixed brokerless devices and pre-configured brokers in dedicated installations. A client wishing to create a topic MUST use a CoAP POST to the pub/sub API with a payload indicating the desired topic. The topic specification sent in the payload MUST use a supported serialization of the CoRE link format [RFC6690]. The target of the link MUST be a URI formatted string. The client MUST indicate the desired content format for publishes to the topic by using the ct (Content Format) link attribute in the link-format payload. Additional link target

attributes and relation values may be included in the topic specification link whena topic is created.

The client MAY indicate the lifetime of the topic by including the Max-Age option in the CREATE request.

Topic hierarchies can be created by creating parent topics. A parent topic is created with a POST using ct (Content Format) link attribute value which describes a supported serialization of the CoRE link format [RFC6690], such as application/link-format (ct=40) or its JSON or CBOR serializations. If a topic is created which describes a link serialization, that topic may then have sub-topics created under it as shown in Figure 7.

Ony one level in the topic hierarchy may be created as a result of a CREATE operation, unless create on PUBLISH is supported (see Section 4.3). The topic string used in the link target MUST NOT contain the "/" character.

A topic creator MUST include exactly one content format link attribute value (ct) in the create payload. If the Broker does not support the indicated format for both publish and subscribe, or if there is more than one "ct" value included in the request, the Broker MUST reject the operation with an error code of 4.00 "Bad Request".

Only one topic may be created per request. If there is more than one link included in a CREATE request, the Broker MUST reject the operation with an erroro code of 4.00 "Bad Request".

There is no default content format. If no ct is specified, the Broker MUST reject the operation with an error code of 4.00 "Bad Request".

A Broker MUST return a response code of "2.01 Created" if the topic is created and return the URI path of the created topic via Location-Path options. The Broker MUST return the appropriate 4.xx response code indicating the reason for failure if a new topic can not be created. Broker SHOULD remove topics if the Max-Age of the topic is exceeded without any publishes to the topic. Broker SHOULD retain a topic indefinitely if the Max-Age option is elided or is set to zero upon topic creation. The lifetime of a topic MUST be refreshed upon create operations with a target of an existing topic.

The CREATE interface is specified as follows:

Interaction: Client -> Broker

Method: POST

URI Template: {+ps}/{+topic}

URI Template Variables: ps := Pub/sub REST API entry point (optional). The entry point of the pub/sub REST API, as obtained from discovery, used to discover topics.

topic := The desired topic to return links for (optional).

Content-Format: application/link-format

Payload: The desired topic to CREATE

The following response codes are defined for the CREATE operation:

Success: 2.01 "Created". Successful Creation of the topic

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.03 "Forbidden". Topic already exists.

Failure: 4.06 "Not Acceptable". Unsupported content format for topic.

Figure 6 shows an example of a topic called "topic1" being successfully created.

```
       Client                                          Broker
         |                                               |
         | ---------- POST /ps/ "<topic1>;ct=50" ------->|
         |                                               |
         | <---------------- 2.01 Created ---------------|
         |               Location: /ps/topic1            |
         |                                               |
```

Figure 6: Example of CREATE topic

```
          Client                                         Broker
            |                                               |
            | ----- POST /ps/ "<parent-topic>;ct=40" ------>|
            |                                               |
            | <---------------- 2.01 Created ---------------|
            |           Location: /ps/parent-topic/         |
            |                                               |
            |-- POST /ps/parent-topic/ "<subtopic>;ct=50" ->|
            |                                               |
            | <---------------- 2.01 Created ---------------|
            |      Location: /ps/parent-topic/subtopic      |
            |                                               |
            |                                               |
```

Figure 7: Example of CREATE of topic hierarchy

4.3. PUBLISH

A CoAP pub/sub Broker MAY allow clients to PUBLISH to topics on the Broker. A client MAY use the PUT or the POST method to publish state updates to the CoAP pub/sub Broker. A client MUST use the content format specified upon creation of a given topic to publish updates to that topic. The Broker MUST reject publish operations which do not use the specified content format. A CoAP client publishing on a topic MAY indicate the maximum lifetime of the value by including the Max-Age option in the publish request. The Broker MUST return a response code of "2.04 Changed" if the publish is accepted. A Broker MAY return a "4.04 Not Found" if the topic does not exist. A Broker MAY return "4.29 Too Many Requests" if simple flow control as described in Section 7 is implemented.

A Broker MUST accept PUBLISH operations using the PUT method. PUBLISH operations using the PUT method replace any stored representation associated with the topic, with the supplied representation. A Broker MAY reject, or delay responses to, PUT requests to a topic while pending resolution of notifications to subscribers from previous PUT requests.

Create on PUBLISH: A Broker MAY accept PUBLISH operations to new topics using the PUT method. If a Broker accepts a PUBLISH using PUT to a topic that does not exist, the Broker MUST create the topic using the information in the PUT operation. The Broker MUST create a topic with the URI-Path of the request, including all of the sub-topics necessary, and create a topic link with the ct attribute set to the content-format value from the header of the PUT request. If topic is created, the Broker MUST return the response "2.01 Created"

with the URI of the created topic, including all of the created path segments, returned via the Location-Path option.

Figure 9 shows an example of a topic being created on first PUBLISH.

A Broker MAY accept PUBLISH operations using the POST method. If a Broker accepts PUBLISH using POST it shall respond with the 2.04 Changed status code. If an attempt is made to PUBLISH using POST to a topic that does not exist, the Broker SHALL return a response status indicating resource not found, such as HTTP 404 or CoAP 4.04.

A Broker MAY perform garbage collection of stored representations which have been delivered to all subscribers or which have timed out. A Broker MAY retain at least one most recently published representation to return in response to SUBSCRIBE and READ requests.

A Broker MUST make a best-effort attempt to notify all clients subscribed on a particular topic each time it receives a publish on that topic. An example is shown in Figure 10.

If a client publishes to a Broker with the Max-Age option, the Broker MUST include the same value for the Max-Age option in all notifications.

A Broker MUST use CoAP Notification as described in [RFC7641] to notify subscribed clients.

The PUBLISH operation is specified as follows:

Interaction: Client -> Broker

Method: PUT, POST

URI Template: {+ps}/{+topic}

URI Template Variables: ps := Pub/sub REST API entry point (optional). The entry point of the pub/sub REST API, as obtained from discovery, used to discover topics.

topic := The desired topic to return links for (optional).

Content-Format: Any valid CoAP content format

Payload: Representation of the topic value (CoAP resource state representation) in the indicated content format

The following response codes are defined for the PUBLISH operation:

Success: 2.01 "Created". Successful publish, topic is created

Success: 2.04 "Changed". Successful publish, topic is updated

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.04 "Not Found". Topic does not exist.

Failure: 4.29 "Too Many Requests". The client should slow down the rate of publish messages for this topic (see Section 7).

Figure 8 shows an example of a new value being successfully published to the topic "topic1". See Figure 10 for an example of a Broker forwarding a message from a publishing client to a subscribed client.

```
     Client                                           Broker
       |                                                |
       | ---------- PUT /ps/topic1 "1033.3"  --------> |
       |                                                |
       |                                                |
       | <--------------- 2.04 Changed---------------- |
       |                                                |
```

Figure 8: Example of PUBLISH

```
     Client                                           Broker
       |                                                |
       | -------- PUT /ps/exa/mpl/e "1033.3"  -------> |
       |                                                |
       |                                                |
       | <--------------- 2.01 Created---------------- |
       |             Location: /ps/exa/mpl/e            |
       |                                                |
```

Figure 9: Example of CREATE on PUBLISH

4.4. SUBSCRIBE

A CoAP pub/sub Broker MAY allow Clients to subscribe to topics on the Broker using CoAP Observe as described in [RFC7641]. A CoAP pub/sub Client wishing to Subscribe to a topic on a Broker MUST use a CoAP GET with the Observe option set to 0 (zero). The Broker MAY add the client to a list of observers. The Broker MUST return a response code of "2.05 Content" along with the most recently published value

if the topic contains a valid value and the Broker can supply the requested content format. The Broker MUST reject Subscribe requests on a topic if the content format of the request is not the content format the topic was created with.

If the topic was published with the Max-Age option, the Broker MUST set the Max-Age option in the valid response to the amount of time remaining for the value to be valid since the last publish operation on that topic. The Broker MUST return a response code of "2.07 No Content" if the topic has not yet been published to, or if Max-Age of the previously stored value has expired. The Broker MUST return a response code "4.04 Not Found" if the topic does not exist or has been removed.

The Broker MUST return a response code "4.15 Unsupported Content Format" if it can not return the requested content format. If a Broker is unable to accept a new Subscription on a topic, it SHOULD return the appropriate response code without the Observe option as per [RFC7641] Section 4.1.

There is no explicit maximum lifetime of a Subscription, thus a Broker may remove subscribers at any time. The Broker, upon removing a Subscriber, will transmit the appropriate response code without the Observe option, as per [RFC7641] Section 4.2, to the removed Subscriber.

The SUBSCRIBE operation is specified as follows:

Interaction: Client -> Broker

Method: GET

Options: Observe:0

URI Template: {+ps}/{+topic}

URI Template Variables: ps := Pub/sub REST API entry point (optional). The entry point of the pub/sub REST API, as obtained from discovery, used to discover topics.

topic := The desired topic to return links for (optional).

The following response codes are defined for the SUBSCRIBE operation:

Success: 2.05 "Content". Successful subscribe, current value included

Success: 2.07 "No Content". Successful subscribe, value not included

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.04 "Not Found". Topic does not exist.

Failure: 4.15 "Unsupported Content Format". Unsupported content format.

Figure 10 shows an example of Client2 subscribing to "topic1" and receiving a response from the Broker, with a subsequent notification. The subscribe response from the Broker uses the last stored value associated with the topic1. The notification from the Broker is sent in response to the publish received from Client1.

```
  Client1   Client2                                          Broker
     |         |                  Subscribe                     |
     |         | ----- GET /ps/topic1 Observe:0 Token:XX ----> |
     |         |                                                |
     |         | <---------- 2.05 Content Observe:10---------- |
     |         |                                                |
     |         |                                                |
     |         |                   Publish                      |
     | ---------|----------- PUT /ps/topic1 "1033.3"  --------> |
     |         |                   Notify                       |
     |         | <---------- 2.05 Content Observe:11 --------- |
     |         |                                                |
```

Figure 10: Example of SUBSCRIBE

4.5. UNSUBSCRIBE

If a CoAP pub/sub Broker allows clients to SUBSCRIBE to topics on the Broker, it MUST allow Clients to unsubscribe from topics on the Broker using the CoAP Cancel Observation operation. A CoAP pub/sub Client wishing to unsubscribe to a topic on a Broker MUST either use CoAP GET with Observe using an Observe parameter of 1 or send a CoAP Reset message in response to a publish, as per [RFC7641].

The UNSUBSCRIBE operation is specified as follows:

Interaction: Client -> Broker

Method: GET

Options: Observe:1

URI Template: {+ps}/{+topic}{?q*}

URI Template Variables: ps := Pub/sub REST API entry point
(optional). The entry point of the pub/sub REST API, as obtained
from discovery, used to discover topics.

topic := The desired topic to return links for (optional).

q := Query Filter (optional). MAY contain a query filter list as
per [RFC6690] Section 4.1.

The following response codes are defined for the UNSUBSCRIBE
operation:

Success: 2.05 "Content". Successful unsubscribe, current value
included

Success: 2.07 "No Content". Successful unsubscribe, value not
included

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.04 "Not Found". Topic does not exist.

Figure 11 shows an example of a client unsubscribe using the
Observe=1 cancellation method.

```
       Client                                           Broker
         |                                                |
         | ----- GET /ps/topic1 Observe:1 Token:XX ----> |
         |                                                |
         | <------------- 2.05 Content ----------------- |
         |                                                |
```

Figure 11: Example of UNSUBSCRIBE

4.6. READ

A CoAP pub/sub Broker MAY accept Read requests on a topic using the
the CoAP GET method if the content format of the request matches the
content format the topic was created with. The Broker MUST return a
response code of "2.05 Content" along with the most recently

published value if the topic contains a valid value and the Broker can supply the requested content format.

If the topic was published with the Max-Age option, the Broker MUST set the Max-Age option in the valid response to the amount of time remaining for the topic to be valid since the last publish. The Broker MUST return a response code of "2.07 No Content" if the Max-Age of the previously stored value has expired, or if the topic has not yet been published to.

The Broker MUST return a response code "4.04 Not Found" if the topic does not exist or has been removed. The Broker MUST return a response code "4.15 Unsupported Content Format" if the Broker can not return the requested content format.

The READ operation is specified as follows:

Interaction: Client -> Broker

Method: GET

URI Template: {+ps}/{+topic}

URI Template Variables: ps := Pub/sub REST API entry point (optional). The entry point of the pub/sub REST API, as obtained from discovery, used to discover topics.

topic := The desired topic to return links for (optional).

The following response codes are defined for the READ operation:

Success: 2.05 "Content". Successful READ, current value included

Success: 2.07 "No Content". Topic exists, value not included

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.04 "Not Found". Topic does not exist.

Failure: 4.15 "Unsupported Content Format". Unsupported content-format.

Figure 12 shows an example of a successful READ from topic1, followed by a Publish on the topic, followed at some time later by a read of the updated value from the recent Publish.

```
     Client1   Client2                                          Broker
       |          |                     Read                      |
       |          | --------------- GET /ps/topic1 -------------> |
       |          |                                               |
       |          | <---------- 2.05 Content "1007.1"------------ |
       |          |                                               |
       |          |                                               |
       |          |                    Publish                    |
       | ---------|----------- PUT /ps/topic1 "1033.3"  --------> |
       |          |                                               |
       |          |                                               |
       |          |                     Read                      |
       |          | --------------- GET /ps/topic1 -------------> |
       |          |                                               |
       |          | <----------- 2.05 Content "1033.3" ---------- |
       |          |                                               |
```

Figure 12: Example of READ

4.7. REMOVE

A CoAP pub/sub Broker MAY allow clients to remove topics from the Broker using the CoAP Delete method on the URI of the topic. The CoAP pub/sub Broker MUST return "2.02 Deleted" if the removal is successful. The Broker MUST return the appropriate 4.xx response code indicating the reason for failure if the topic can not be removed.

When a topic is removed for any reason, the Broker SHOULD remove all of the observers from the list of observers and return a final 4.04 "Not Found" response as per [RFC7641] Section 3.2. If a topic which has sub-topics is removed, then all of its sub-topics MUST be recursively removed.

The REMOVE operation is specified as follows:

Interaction: Client -> Broker

Method: DELETE

URI Template: {+ps}/{+topic}

URI Template Variables: ps := Pub/sub REST API entry point (optional). The entry point of the pub/sub REST API, as obtained from discovery, used to discover topics.

topic := The desired topic to return links for (optional).

Content-Format: None

Response Payload: None

The following response codes are defined for the REMOVE operation:

Success: 2.02 "Deleted". Successful remove

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.04 "Not Found". Topic does not exist.

Figure 13 shows a successful remove of topic1.

```
        Client                                          Broker
          |                                               |
          | ------------- DELETE /ps/topic1 ------------> |
          |                                               |
          |                                               |
          | <-------------- 2.02 Deleted ---------------- |
          |                                               |
```

Figure 13: Example of REMOVE

5. CoAP Pub/sub Operation with Resource Directory

A CoAP pub/sub Broker may register the base URI, which is the REST API entry point for a pub/sub service, with a Resource Directory. A pub/sub Client may use an RD to discover a pub/sub Broker.

A CoAP pub/sub Client may register links [RFC6690] with a Resource Directory to enable discovery of created pub/sub topics. A pub/sub Client may use an RD to discover pub/sub Topics. A client which registers pub/sub Topics with an RD MUST use the context relation (con) [I-D.ietf-core-resource-directory] to indicate that the context of the registered links is the pub/sub Broker.

A CoAP pub/sub Broker may alternatively register links to its topics to a Resource Directory by triggering the RD to retrieve it's links from .well-known/core. In order to use this method, the links must first be exposed in the .well-known/core of the pub/sub Broker. See Section 4.1 in this document.

The pub/sub Broker triggers the RD to retrieve its links by sending a POST with an empty payload to the .well-known/core of the Resource

Directory. The RD server will then retrieve the links from the .well-known/core of the pub/sub Broker and incorporate them into the Resource Directory. See [I-D.ietf-core-resource-directory] for further details.

6. Sleep-Wake Operation

CoAP pub/sub provides a way for client nodes to sleep between operations, conserving energy during idle periods. This is made possible by shifting the server role to the Broker, allowing the Broker to be always-on and respond to requests from other clients while a particular client is sleeping.

For example, the Broker will retain the last state update received from a sleeping client, in order to supply the most recent state update to other clients in response to read and subscribe operations.

Likewise, the Broker will retain the last state update received on the topic such that a sleeping client, upon waking, can perform a read operation to the Broker to update its own state from the most recent system state update.

7. Simple Flow Control

Since the Broker node has to potentially send a large amount of notification messages for each publish message and it may be serving a large amount of subscribers and publishers simultaneously, the Broker may become overwhelmed if it receives many publish messages to popular topics in a short period of time.

If the Broker is unable to serve a certain client that is sending publish messages too fast, the Broker SHOULD respond with Response Code 4.29, "Too Many Requests" [I-D.ietf-core-too-many-reqs] and set the Max-Age Option to indicate the number of seconds after which the client can retry. The Broker MAY stop creating notifications from the publish messages from this client and to this topic for the indicated time.

If a client receives the 4.29 Response Code from the Broker for a publish message to a topic, it MUST NOT send new publish messages to the Broker on the same topic before the time indicated in Max-Age has passed.

8. Security Considerations

CoAP pub/sub re-uses CoAP [RFC7252], CoRE Resource Directory [I-D.ietf-core-resource-directory], and Web Linking [RFC5988] and therefore the security considerations of those documents also apply

to this specification. Additionally, a CoAP pub/sub Broker and the clients SHOULD authenticate each other and enforce access control policies. A malicious client 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 pub/sub Broker introduces challenges for the use of end-to-end security between for example a client device on a sensor network and a client application running in a cloud-based server infrastructure since Brokers terminate the exchange. While running separate DTLS sessions from the client device to the Broker and from Broker to client application protects confidentially on those paths, the client device does not know whether the commands coming from the Broker are actually coming from the client application. Similarly, a client application requesting data does not know whether the data originated on the client device. 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 device that any request originated at the client application. Similarly, integrity protected sensor data from a client device will also provide guarantee to the client application that the data originated on the client device 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 pub/sub Broker, the use of end-to-end object security is RECOMMENDED as described in [I-D.palombini-ace-coap-pubsub-profile]. An example application that uses the CoAP pub/sub Broker and relies on end-to-end object security is described in [RFC8387]. When only end-to-end encryption is necessary and the CoAP Broker is trusted, Payload Only Protection (Mode:PAYL) could be used. The Publisher would wrap only the payload before sending it to the Broker and set the option Content-Format to application/smpayl. Upon receival, the Broker can read the unencrypted CoAP header to forward it to the subscribers.

9. IANA Considerations

This document registers one attribute value in the Resource Type (rt=) registry established with [RFC6690] and appends to the definition of one CoAP Response Code in the CoRE Parameters Registry.

9.1. Resource Type value 'core.ps'

- o Attribute Value: core.ps
- o Description: Section 4 of [[This document]]
- o Reference: [[This document]]
- o Notes: None

9.2. Resource Type value 'core.ps.discover'

- o Attribute Value: core.ps.discover
- o Description: Section 4 of [[This document]]
- o Reference: [[This document]]
- o Notes: None

9.3. Response Code value '2.07'

- o Response Code: 2.07
- o Description: No Content
- o Reference: [[This document]]
- o Notes: The server sends this code to the client to indicate that the request was valid and accepted, but the response may contain an empty payload. It is comparable to and may be proxied with the HTTP 204 No Content status code.

10. Acknowledgements

The authors would like to thank Hannes Tschofenig, Zach Shelby, Mohit Sethi, Peter van der Stok, Tim Kellogg, Anders Eriksson, Goran Selander, Mikko Majanen, and Olaf Bergmann for their contributions and reviews.

11. References

11.1. Normative References

[I-D.ietf-core-too-many-reqs]
Keranen, A., "Too Many Requests Response Code for the Constrained Application Protocol", draft-ietf-core-too-many-reqs-06 (work in progress), November 2018.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <https://www.rfc-editor.org/info/rfc6570>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <https://www.rfc-editor.org/info/rfc6690>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <https://www.rfc-editor.org/info/rfc7641>.

11.2. Informative References

[I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-15 (work in progress), August 2018.

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-18 (work in progress), December 2018.

[I-D.palombini-ace-coap-pubsub-profile]
Palombini, F., "CoAP Pub-Sub Profile for Authentication and Authorization for Constrained Environments (ACE)", draft-palombini-ace-coap-pubsub-profile-03 (work in progress), June 2018.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <https://www.rfc-editor.org/info/rfc5988>.

[RFC8387] Sethi, M., Arkko, J., Keranen, A., and H. Back, "Practical Considerations and Implementation Experiences in Securing Smart Object Networks", RFC 8387, DOI 10.17487/RFC8387, May 2018, <https://www.rfc-editor.org/info/rfc8387>.

Authors' Addresses

Michael Koster
SmartThings

Email: Michael.Koster@smartthings.com

Ari Keranen
Ericsson

Email: ari.keranen@ericsson.com

Jaime Jimenez
Ericsson

Email: jaime.jimenez@ericsson.com