Thing-to-Thing Research Group
Internet-Draft
Intended status: Experimental
Expires: November 10, 2020

K. Hartke
Ericsson

May 9, 2020

Publish/Subscribe over the Constrained Application Protocol (CoAP) using the Constrained RESTful Application Language (CoRAL) draft-hartke-t2trg-coral-pubsub-01

Abstract

This document explores how the Constrained RESTful Application Language (CoRAL) might be used for enabling publish/subscribe-style communication over the Constrained Application Protocol (CoAP), which allows CoAP nodes with long breaks in connectivity and/or up-time to exchange data via a publish/subscribe broker.

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 https://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 November 10, 2020.

Table of Contents

1. Preamble

This document explores how CoAP Publish/Subscribe Broker [I-D.ietf-core-coap-pubsub] might look like if based on CoRAL [I-D.ietf-core-coral]. The exploration is done in the style of a self-contained specification rather than a description of changes.

2. Introduction

Constrained RESTful Environments (CoRE) realize the Representational State Transfer (REST) architectural style [REST] in a suitable form for constrained nodes (e.g., 8-bit microcontrollers with limited RAM and ROM) and constrained networks [RFC7228]. CoRE technologies like the Constrained Application Protocol (CoAP) [RFC7252] are aimed at machine-to-machine (M2M) applications like smart energy and building automation.

An important class of constrained nodes are devices that are intended to run for years on a small battery or by scavenging energy from their environment. These nodes have limited reachability, since they spend most of their time in a sleeping state with no network connectivity. Another important class of nodes are devices with limited reachability due to middle-boxes like Network Address Translators (NATs) and firewalls.

For these nodes, the client/server-oriented architecture of REST can be challenging when interactions are not initiated by the devices themselves. A publish/subscribe-oriented architecture where nodes are separated by a broker and data is exchanged via topics might fit these nodes better.

This document applies the idea of a "Publish/Subscribe Broker" to Constrained RESTful Environments. The broker enables store-and-forward data exchange between nodes, thereby facilitating the communication of nodes with limited reachability, providing simple many-to-many communication, and easing integration with other publish/subscribe systems.

2.1. Brokers and Topics

In this specification, publishing and subscribing is facilitated by a CoAP server, called the "publish/subscribe broker" (see Figure 1). Publishers and subscribers are CoAP clients that interact with this broker using a RESTful API.

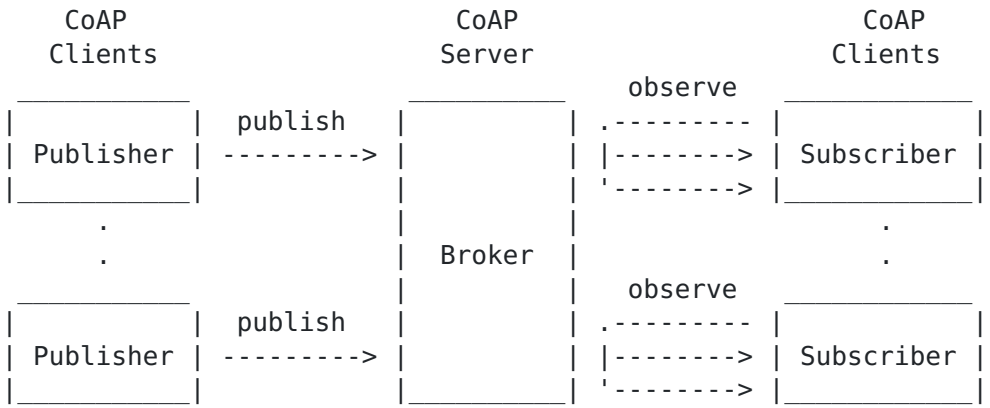

Figure 1: Publish/Subscribe over CoAP

Data is forwarded from publishers to subscribers via "topics" at the broker. This way, both publishers and subscribers do not need to have any knowledge each other; they just have to share the topic they're publishing and subscribing to.

Topics have to be created and configured before any data can be published. These are managed by the broker. Clients may propose new topics to be created; however, it is up to the broker to choose if and how a topic is created. The broker also decides the URI of each topic.

The creation, configuration, and discovery of topics at a broker is specified in Section 3.

2.2. Publish and Subscribe

A simple publish/subscribe mechanism can be implemented over CoAP by having publishers submit their data in PUT requests to a broker-managed resource and letting subscribers observe this resource [RFC7641].

However, the requirements may not always be simple: When observing a resource, notifications are sent on a best-effort basis. Subscribers are notified of as many state changes as possible; however, when a publisher is publishing faster than subscribers can be notified, subscribers might not see every publication. If this is not desired, an alternative mechanism needs to be be used. Such alternatives are supported, but their specification is out of this document's scope.

The simple publish/subscribe mechanism is specified in Section 4.

2.3. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Topics

The configuration side of a "publish/subscribe broker" consists of a collection of topics. These topics as well as the collection itself are exposed by a CoAP server as resources (see Figure 2).

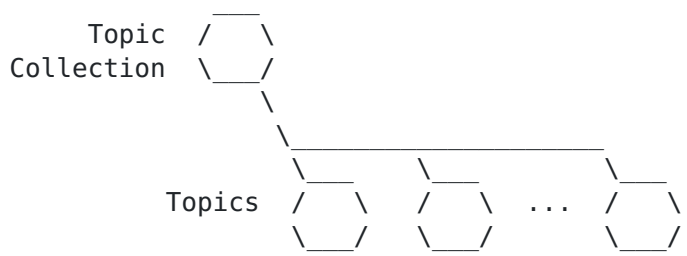

Figure 2: Resources of a Publish-Subscribe Broker

3.1. Topic Creation and Configuration

Each topic has a topic configuration. A CoAP client can create a new topic by submitting an initial configuration for the topic. It can also read and update the configuration of existing topics and delete them when they are no longer needed.

The configuration of a topic itself consists of a set of properties. These fall into one of two categories: configuration properties and status properties. Configuration properties can be set by a client and describe the desired configuration of a topic. Status properties are read-only, managed by the server, and provide information about the actual status of a topic.

When a client submits a configuration to create a new topic or update an existing topic, it can only submit configuration properties. When a server returns the configuration of a topic, it returns both the configuration properties and the status properties of the topic.

Every property has a type and a value. The type takes the form of an IRI [RFC3987]. This IRI serves only as an identifier; it must not be dereferenced by clients. The value can be either a Boolean value, an

integer, a floating-point number, a date/time value, a byte string, a text string, or a resource reference in the form of a URI [RFC3986].

3.1.1. Configuration Properties

The following configuration properties are defined:

TODO.

3.1.2. Status Properties

The following status properties are defined:

TODO.

3.1.3. Topic Configuration Representation

A topic configuration is represented as a CoRAL document [I-D.ietf-core-coral] containing the configuration properties and status properties of the topic as top-level elements.

Each property is represented as a link where the link relation type is the property type and the link target is the property value.

3.2. Topic Discovery

Topics can be discovered by a client on the basis of configuration properties and status properties. For example, a client could fetch a list of all topics that have a property of type "foo" or that have a property of type "bar" with the value 42. Alternatively, topics can also be discovered simply by getting the full list of all topics.

3.2.1. Topic List Representation

A list of topics is represented as a CoRAL document [I-D.ietf-core-coral] containing the topics in the list as top-level elements.

Each topic is represented as a link where the link relation type is <http://coreapps.org/pubsub#item> and the link target is the topic URI. This link can optionally contain (a subset of) the configuration properties and status properties of the topic as nested elements.

Each property is represented in the same way as in Section 3.1.3.

3.2.2. Filter Query Representation

TODO.

3.3. Interactions

3.3.1. Getting All Topics

A client can list a collection of topics by making a GET request to the collection URI.

On success, the server returns a 2.05 (Content) response with a representation of the list of all topics (see Section 3.2.1) in the collection.

Example:

```
=> 0.01 GET
   Uri-Path: pubsub
   Uri-Path: topics

<= 2.05 Content
   Content-Format: 65536

   item </pubsub/topics/1>
   item </pubsub/topics/2>
   item </pubsub/topics/3>
```

3.3.2. Getting Topics by Properties

A client can filter a collection of topics by submitting the representation of a topic filter (see Section 3.2.2) in a FETCH request to the topic collection URI.

On success, the server returns a 2.05 (Content) response with a representation of a list of topics in the collection (see Section 3.2.1) that match the filter.

Example:

```
=> 0.05 FETCH
   Uri-Path: pubsub
   Uri-Path: topics
   Content-Format: TODO

   TODO

<= 2.05 Content
```

```
   Content-Format: 65536

   item </pubsub/topics/1>
   item </pubsub/topics/2>
   item </pubsub/topics/3>
```

3.3.3. Creating a Topic

A client can add a new topic to a collection of topics by submitting a representation of the initial topic configuration (see Section 3.1.3) in a POST request to the topic collection URI.

If client just wants all the default configuration properties, it can simply submit an empty CoRAL document.

On success, the server returns a 2.01 (Created) response indicating the topic URI of the new topic.

Example:

```
=> 0.02 POST
   Uri-Path: pubsub
   Uri-Path: topics
   Content-Format: 65536

   foo "xyz"
   bar 42

<= 2.01 Created
   Location-Path: pubsub
   Location-Path: topics
   Location-Path: 1234
```

3.3.4. Reading the Configuration of a Topic

A client can read the configuration of a topic by making a GET request to the topic URI.

On success, the server returns a 2.05 (Content) response with a representation of the topic configuration (see Section 3.1.3).

Example:

```
=> 0.01 GET
   Uri-Path: pubsub
   Uri-Path: topics
   Uri-Path: 1234
```

```
<= 2.05 Content
   Content-Format: 65536
   Max-Age: 300

   foo "xyz"
   bar 42
```

3.3.5. Updating the Configuration of a Topic

A client can update the configuration of a topic by submitting the representation of the updated topic configuration (see Section 3.1.3) in a PUT request to the topic URI. Any existing properties in the configuration are replaced by this update.

On success, the server returns a 2.04 (Updated) response.

Example:

```
=> 0.03 PUT
   Uri-Path: pubsub
   Uri-Path: topics
   Uri-Path: 1234
   Content-Format: 65536

   foo "abc"

<= 2.04 Updated
```

3.3.6. Deleting a Topic

A client can delete a topic by making a DELETE request on the topic URI.

On success, the server returns a 2.02 (Deleted) response.

Any subscribers to the topic are automatically unsubscribed.

Example:

```
=> 0.04 DELETE
   Uri-Path: pubsub
   Uri-Path: topics
   Uri-Path: 1234

<= 2.02 Deleted
```

4. Publish/Subscribe

Unless a topic is configured to use a different mechanism, publish/subscribe is performed as follows: A publisher publishes to a topic by submitting the data in a PUT request to a broker-managed "topic data resource". This causes a change to the state of that resources. Any subscriber observing the resource [RFC7641] at that time receives a notification about the change to the resource state.

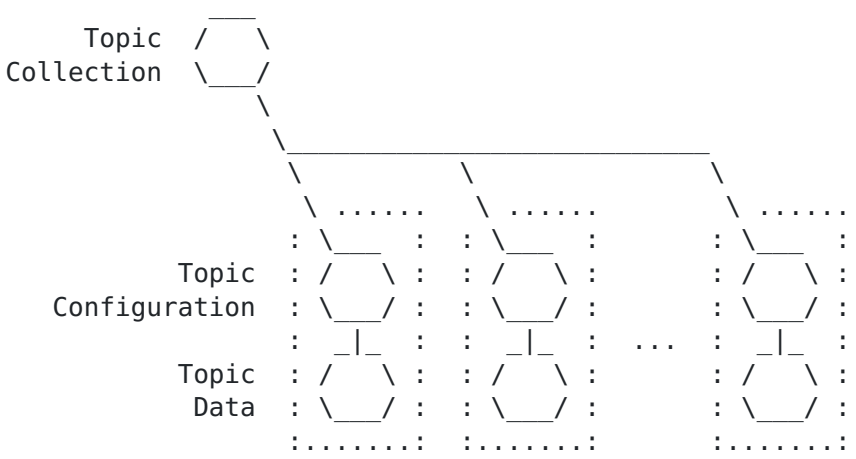

Figure 3: Resources of a Publish-Subscribe Broker

The topic data resource (which is different from the resource holding the topic configuration) does not exist until some initial data has been published to it. Before initial data has been published, the topic data resource yields a 4.04 (Not Found) response. If such a "half created" topic is undesired, the creator of the topic can simply immediately publish some initial placeholder data to make the topic "fully created".

All URIs for configuration and data resources are broker-generated. There does not need to be any URI pattern dependence between the URI where the data exists and the URI of the topic configuration. Topic configuration and data resources might even be hosted on different servers.

4.1. Topic Lifecycle

When a topic is newly created, it is first placed by the server into the HALF CREATED state (see Figure 4). In this state, a client can read and update the configuration of the topic and delete the topic. A publisher can publish to the topic data resource. However, a subscriber cannot yet observe the topic data resource.

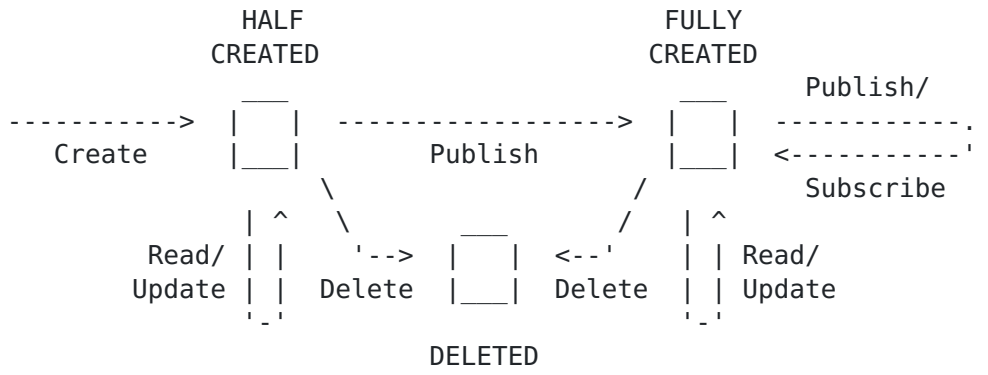

Figure 4: Lifecycle of a Topic

After a publisher publishes to the topic for the first time, the topic is placed into the FULLY CREATED state. In this state, a client can read and update the configuration of the topic and delete the topic; a publisher can publish to the topic data resource; and a subscriber can observe the topic data resource.

When a client deletes a topic, the topic is placed into the DELETED state and shortly after removed from the server. In this state, all subscribers are removed from the list of observers of the topic data resource and no further interactions with the topic are possible.

4.2. Rate Limiting

The server hosting a data resource may have to handle a potentially very large number of publishers and subscribers at the same time. This means the server can easily become overwhelmed if it receives too many publications in a short period of time.

In this situation, if a client is sending publications too fast, the server can return a 4.29 (Too Many Requests) response [RFC8516]. As described in RFC 8516, the Max-Age option [RFC7252] in this response indicates the number of seconds after which the client may retry. When a client receives a 4.29 (Too Many Requests) response, it should not send any new publication requests to the same topic data resource before the time indicated by the Max-Age option has passed.

4.3. Interactions

4.3.1. Publishing to a Topic

A client can publish data to a topic by submitting it in a PUT request to the topic data URI. The topic data URI is indicated by the status property of type <http://coreapps.org/pubsub#data> in the

topic configuration. (Note: The topic data URI is not identical to the topic URI!)

The data MUST be in the content format specified by the configuration property of type <http://coreapps.org/pubsub#accept> in the topic configuration.

On success, the server returns a 2.04 (Updated) response. However, when data is published to the topic for the first time, the server may instead return a 2.01 (Created) response.

If the request does not have an acceptable content format, the server returns a 4.15 (Unsupported Content Format) response.

If the client is sending publications too fast, the server returns a 4.29 (Too Many Requests) response [RFC8516].

Example:

```
=> 0.03 PUT
   Uri-Path: pubsub
   Uri-Path: data
   Uri-Path: 6578616d706c65
   Content-Format: 112

   [...SenML data...]

<= 2.04 Updated
```

4.3.2. Subscribing to a Topic

A client can get the latest published data and subscribe to newly published data by observing the topic data URI with a GET request that includes the Observe option [RFC7641].

On success, the server returns 2.05 (Content) notifications with the data.

Example:

```
=> 0.01 GET
   Uri-Path: pubsub
   Uri-Path: data
   Uri-Path: 6578616d706c65
   Observe: 0

<= 2.05 Content
   Content-Format: 112
```

```
      Observe: 10001
      Max-Age: 15

      [...SenML data...]

   <= 2.05 Content
      Content-Format: 112
      Observe: 10002
      Max-Age: 15

      [...SenML data...]

   <= 2.05 Content
      Content-Format: 112
      Observe: 10003
      Max-Age: 15

      [...SenML data...]
```

4.3.3. Unsubscribing from a Topic

A client can unsubscribe simply by cancelling the observation as described in Section 3.6 of RFC 7641.

4.3.4. Getting the Latest Published Data

A client can just get the latest published data by making a GET request to the topic data URI.

On success, the server returns 2.05 (Content) response with the data.

Example:

```
   => 0.01 GET
      Uri-Path: pubsub
      Uri-Path: data
      Uri-Path: 6578616d706c65


   <= 2.05 Content
      Content-Format: 112
      Max-Age: 15

      [...SenML data...]
```

5. Security Considerations

TODO.

6. IANA Considerations

TODO.

7. References

7.1. Normative References

[I-D.ietf-core-coral]
Hartke, K., "The Constrained RESTful Application Language (CoRAL)", draft-ietf-core-coral-03 (work in progress), March 2020.

[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>.

[RFC3987] Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, DOI 10.17487/RFC3987, January 2005, <https://www.rfc-editor.org/info/rfc3987>.

[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>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8516] Keranen, A., ""Too Many Requests" Response Code for the Constrained Application Protocol", RFC 8516, DOI 10.17487/RFC8516, January 2019, <https://www.rfc-editor.org/info/rfc8516>.

7.2. Informative References

[I-D.ietf-core-coap-pubsub] Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-pubsub-09 (work in progress), September 2019.

[REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

Acknowledgements

This document is based on a proposal for pub/sub broker improvements presented at IETF 105 in Montreal by Michael Koster, Ari Keranen, and Klaus Hartke. It borrows text from [I-D.ietf-core-coap-pubsub] by Michael Koster, Ari Keranen, and Jaime Jimenez.

Thanks to Christian Amsuess, Carsten Bormann, Rikard Hoeglund, Michael McCool, Francesca Palombini, Ivaylo Petrov, Jim Schaad, Peter van der Stok, and Marco Tiloca for helpful comments and discussions that have shaped the document.

Author's Address

Klaus Hartke
Ericsson
Torshamnsgatan 23
Stockholm SE-16483
Sweden

Email: klaus.hartke@ericsson.com