Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 30, 2015

M. Koster
ARM Limited
A. Keranen
J. Jimenez
Ericsson

October 27, 2014

Publish-Subscribe in the Constrained Application Protocol (CoAP)
draft-koster-core-coap-pubsub-00

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 and message queuing functionality for CoAP that extends the capabilities for supporting nodes with long breaks in connectivity and/or up-time.

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] 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 [I-D.ietf-core-resource-directory]. The extensions enable publish-subscribe communication using a 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 [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, see [RFC6570], is used to describe the REST interfaces defined in this specification.

This specification makes use of the following additional terminology:

CoAP Publish-Subscribe (CoAP-PubSub) 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-PubSub 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-PubSub function set: A group of well-known REST resources that together provide the CoAP-PubSub service.

CoAP-PubSub Endpoint An endpoint that implements the CoAP-PubSub function set. A CoAP-PubSub endpoint has two potential modes, CoAP-PubSub Client and CoAP-PubSub 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-PubSub Broker

Figure 1 shows an example architecture of a CoAP-PubSub capable service. A Resource Directory (RD) 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-PubSub broker. Web clients subscribe to the state of the endpoint from the CoAP-PubSub broker, and publish updates to the endpoint state through the CoAP-PubSub broker. The CoAP-PubSub broker performs a store-and-forward function between web clients and the CoAP-PubSub capable endpoints. The CoAP-PubSub 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.

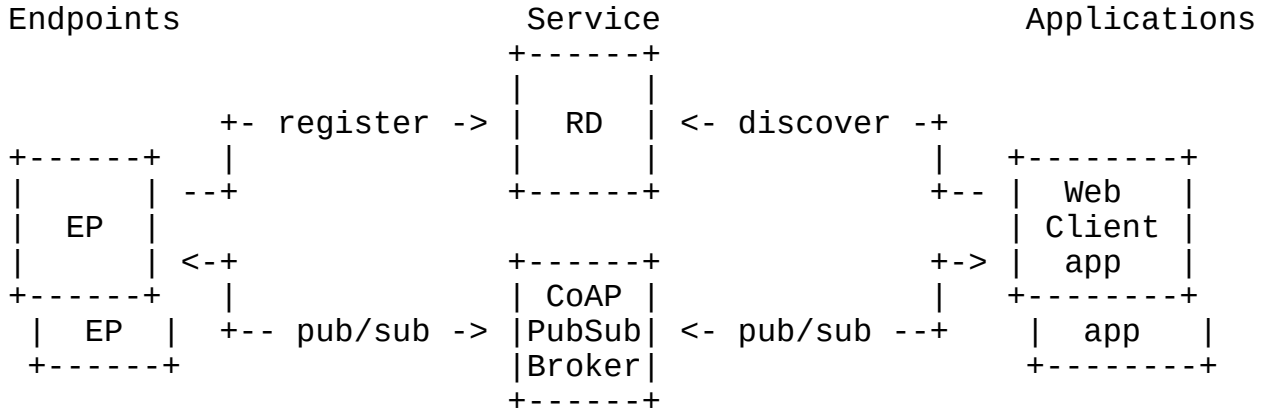

Figure 1: CoAP-PubSub Architecture

3.2. Client Endpoint

Client endpoints initiate all interactions with the RD and CoAP-PubSub broker. If the endpoint is an actuator it will need to either use CoAP Observe [I-D.ietf-core-observe] or periodically poll the PubSub broker to check for updates. A CoAP-PubSub client endpoint MUST use CoAP PUT operations to update its state on the PubSub 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-PubSub broker or RD server.

3.3. Server Endpoint

Server endpoint interactions require the CoAP-PubSub broker to perform the client role, initiating interaction with the server endpoint. The CoAP-PubSub broker MAY then use PUT operations to update state at the server endpoint, and MAY use GET or GET and Observe to subscribe to resources at the endpoint. Server mode endpoints are required to be reachable from the CoAP-PubSub 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-PubSub capabilities and using GET and 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-PubSub Registration and discovery

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

A server that implements a CoAP-PubSub broker MAY advertize this capability by registering the rt="core.pubsub" with an associated Resource Directory. If a server advertizes as a CoAP-PubSub Broker, it MUST support the transactions described in section 5 of this

document. As server that implements the CoAP-PubSub Broker MAY also implement sleeping endpoint and message queueing support referred to in Section 6 of this document.

4.1. Register CoAP-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-PubSub, it discovers the rt="core.pubsub" attribute at the RD service associated with the CoAP-PubSub broker and registers its CoAP-PubSub 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 "/pubsub/sensors/weather/barometer". An implementation MAY mix CoAP-PubSub resources and CoAP REST resources on the same endpoint. Endpoint registration proceeds as per normal RD registration.

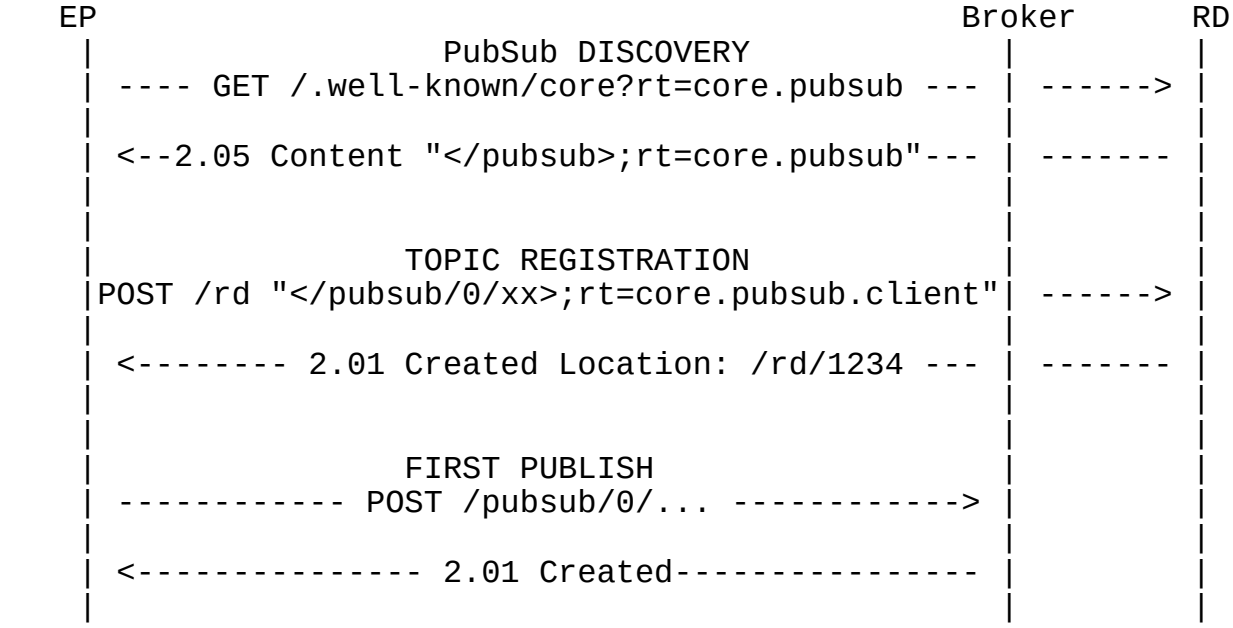

Figure 2: Discovery and Registration

4.2. Unregister Endpoint

CoAP-PubSub 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.

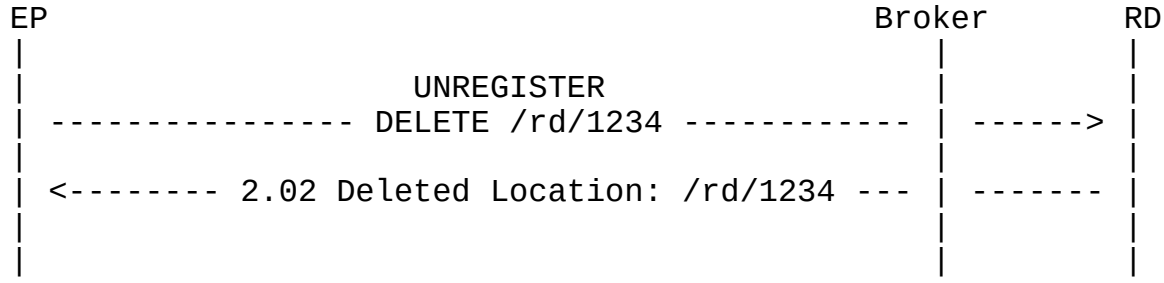

Figure 3: Unregister Endpoint

5. CoAP-PubSub Functions and Interactions

This section describes the transaction flows and interactions between CoAP-PubSub endpoints and CoAP-PubSub 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-PubSub 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-PubSub endpoints and CoAP-PubSub 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-PubSub broker

Client endpoint PUBLISHes updates to CoAP-PubSub broker. A CoAP-PubSub client endpoint MAY use PUT to publish state updates to the CoAP-PubSub broker.

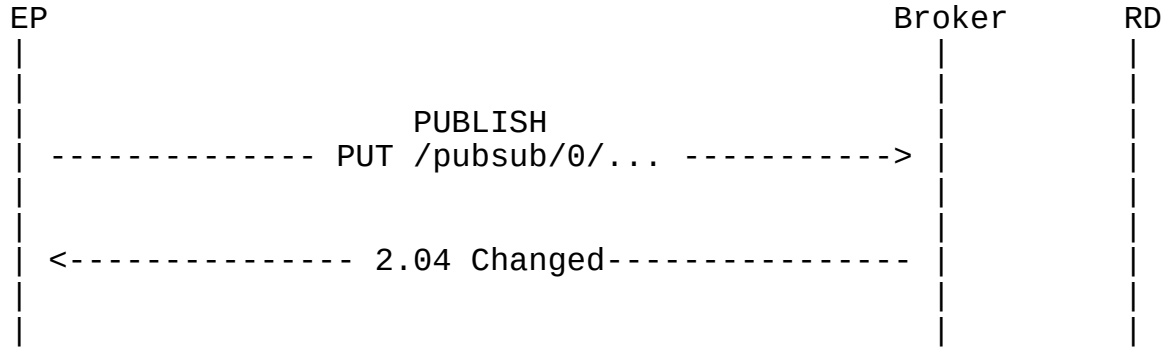

Figure 4: Client Role PUBLISH from EP to Broker

5.1.2. Client Endpoint SUBSCRIBE, Broker PUBLISH

Client mode endpoint subscribes to the topic at the CoAP-PubSub broker using GET and Observe. Published updates to the CoAP-PubSub 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-PubSub 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-PubSub Broker

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

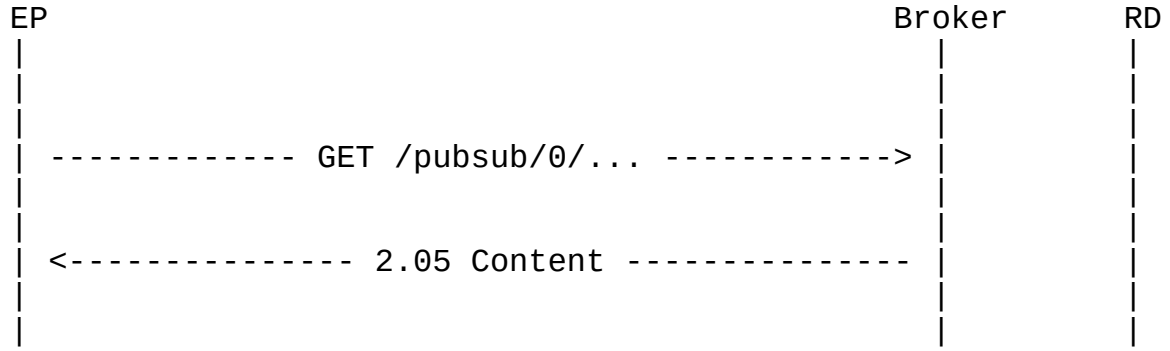

Figure 6: Client EP GET from CoAP-PubSub Broker

5.2. Server Role Endpoint Functions

This section describes the transaction flows and interactions between CoAP-PubSub endpoints and CoAP-PubSub 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-PubSub broker SUBSCRIBES to Server Role EP

The server mode endpoint requires the CoAP-PubSub broker to act as a client and subscribe to a resource on the endpoint using GET + Observe. A CoAP-PubSub broker MAY subscribe to topics registered by a server role endpoint at any time. A CoAP-PubSub broker MUST subscribe to a topic registered by a server role endpoint upon receiving a subscription on the associated topic. A CoAP-PubSub 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-PubSub Broker subscribing to a server role endpoint.

Figure 7: Broker SUBSCRIBE to Server Role EP

5.2.2. CoAP-PubSub Broker Publishes to Server Role Endpoint

CoAP-PubSub 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.

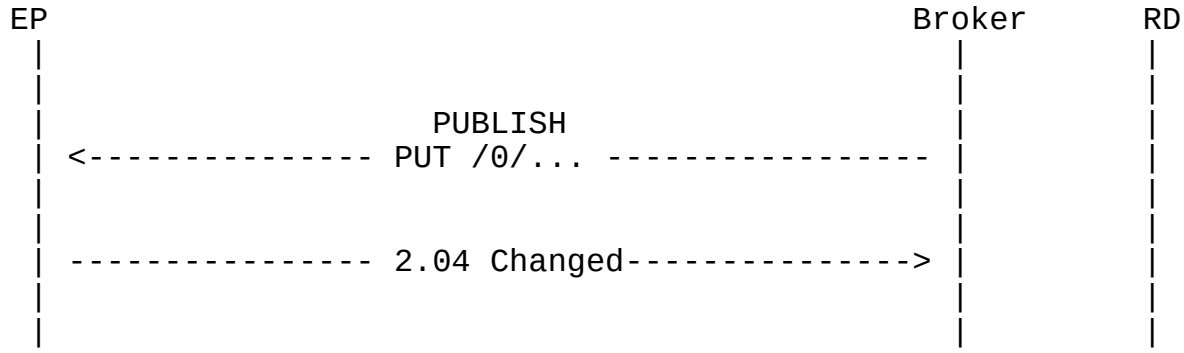

Figure 8: Broker PUBLISH to Server Role EP

5.2.3. CoAP-PubSub Broker GET from Server Role Endpoint

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

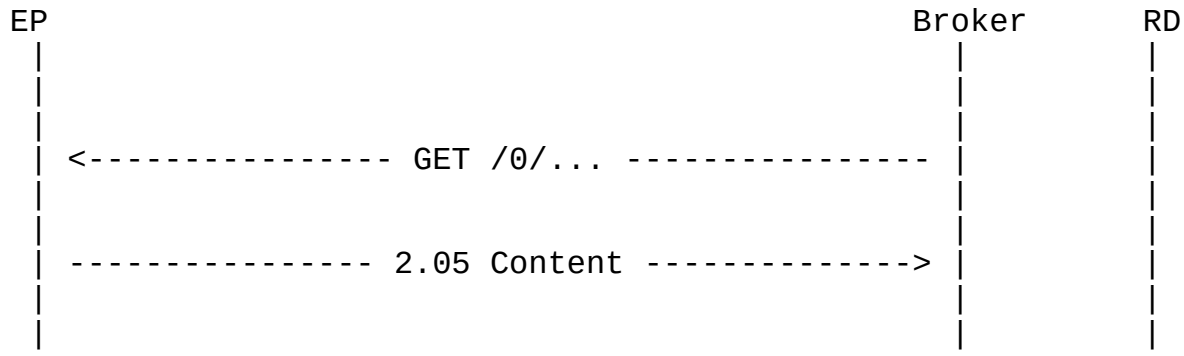

Figure 9: Broker GET from Server Role Endpoint

6. Enabling Multiple Publishers

6.1. Creating a Topic

After registration of the EP in the RD and discovering the CoAP-PubSub 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 PubSub function by doing a first publication as shown in Figure 2.

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

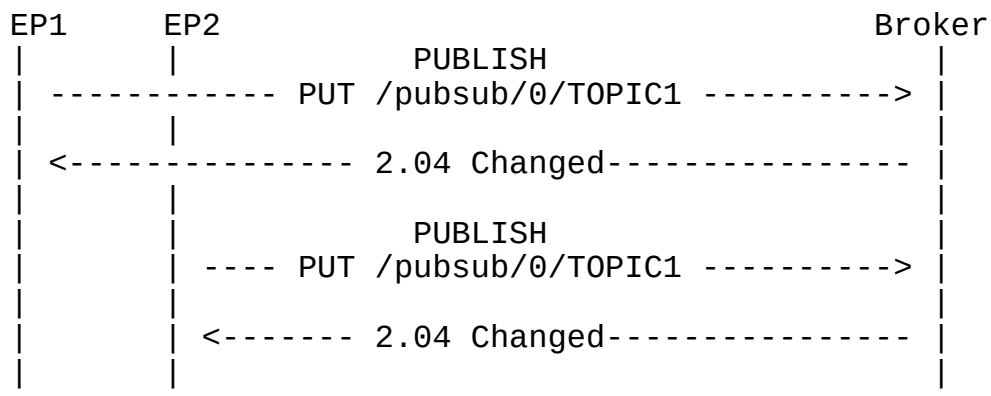

Figure 10: Multiple CoAP-PubSub EPs PUBLISH to Broker

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.

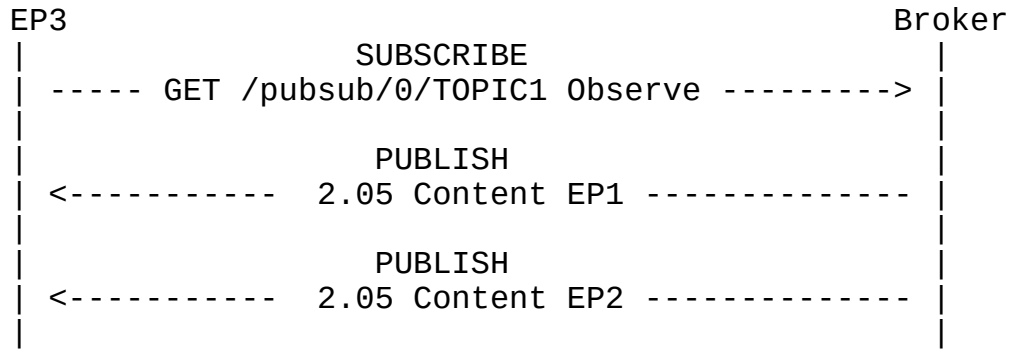

Figure 11: CoAP-PubSub Endpoint SUBSCRIBE to Broker

7. Sleep-Wakeup Operation and Message Queueing

A CoAP-PubSub broker MAY implement support for sleeping endpoints and queueing of messages as provided for in [OMALightweightM2M]

8. Security Considerations

CoAP-PubSub 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-PubSub 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-PubSub 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 confidentially 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-PubSub broker, the use of end-to-end encryption may also be envisioned. The CoAP-PubSub 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 [RFC6690].

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

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

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

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

10. Acknowledgements

The authors would like to thank Hannes Tschofenig, Zach Shelby, Mohit Sethi, and Anders Eriksson for their contributions and reviews

11. References

11.1. Normative References

[I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-14 (work in progress), June 2014.

[I-D.ietf-core-resource-directory]
Shelby, Z., Bormann, C., and S. Krco, "CoRE Resource Directory", draft-ietf-core-resource-directory-01 (work in progress), December 2013.

[OMALightweightM2M]
Open Mobile Alliance, "OMA LightweightM2M v1.0", http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases/oma-lightweightm2m-v1-0, 12 2013.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, March 2012.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

11.2. Informative References

[RFC5988] 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