Enhanced publish/subscribe in CoAP: Describing advanced subscription mechanisms for the Observe extension

Markel Iglesias-Urkia
IK4-Ikerlan Technology Research Centre.
Pº J. M. Arizmendiarrieta, 2.
20500. Mondragón, Spain.
miglesias@ikerlan.es

Diego Casado-Mansilla
Deusto Institute of Technology.
Av. Universidades, 24.
48007. Bilbao, Spain.
dcasado@deusto.es

Simon Mayer
Pro2Future GmbH and Graz University of Technology.
Inffeldgasse 25F.
8010 Graz, Austria.
simon.mayer@pro2future.at

Aitor Urbieta
IK4-Ikerlan Technology Research Centre.
Pº J. M. Arizmendiarrieta, 2.
20500. Mondragón, Spain.
aurbieta@ikerlan.es

ABSTRACT

In the current Internet of Things (IoT) all sorts of devices and objects with diverse capabilities are being connected to the Internet and the Web. Consequently, new lightweight network protocols are also being developed to connect resource-constrained devices or networks with each other. One of these protocols is the Constrained Application Protocol, which provides a REST architecture to resource and network-constrained devices. Besides the RESTful client-server paradigm, CoAP supports a publish-subscribe model thanks to its Observe extension which mimics some of MQTT's functionalities. However, this extension has some limitations in the subscription mechanism. Hence, this paper proposes several enhancements that define new CoAP options and response codes for such mechanisms. Furthermore, a theoretical comparison of the current CoAP capabilities against the new possibilities is also provided. For that, an specific use case is proposed, and a comparison in terms of the overhead required to exchange payloads and to subscribe to state change notifications on resources.

ACM Classification Keywords

C.2.2 COMPUTER SYSTEMS ORGANIZATION: Network Protocols - Applications.

Author Keywords

Constrained Application Protocol; CoAP; Observe; Subscription; Notification; Publish/Subscribe.

IOT '18, October 15–18, 2018, Santa Barbara, CA, USA

 ISBN 978-1-4503-6564-2/18/10…$15.00
DOI: https://doi.org/10.1145/3277593.3277594

INTRODUCTION

With the development of the Internet of Things (IoT), all kinds of objects are being connected to the Internet. This objects are not necessarily electronic devices, but they can be other kinds of devices such as food, objects, clothes or mechanical machines. As such, they do not have high capacities regarding to computational or energy consumptions. This has made clear that new, more lightweight protocols are needed, where despite the resource constraints can achieve good communication between devices. Because of that, several organizations and companies have proposed different internet protocols, focusing on reducing energy consumption and network overhead.

One such protocols is the RFC 7252 - Constrained Application Protocol (CoAP) [21], standardized by the Internet Engineering Task Force (IETF) on June 2014. CoAP is a RESTful application layer protocol following the client-server on top of UDP. It can be securized using DTLS and uses 5683 for regular CoAP and 5684 for secure CoAP as default ports. There are two types of requests, CONfirmable for messages requiring an acknowledgment and NON-Confirmable messages which do not. CoAP uses a subgroup of HTTP as request codes (GET, PUT, POST and DELETE), whilst for responses it uses a subgroup of HTTP as well as new defined codes. Data can be represented in several content format, e.g. plain-text, JSON, CBOR or XML. The resources are addressable using request codes to an URI, being possible to add a query. A regular URI for CoAP looks like `coap://example.host:port/path/to/resource?query=value`.

However, some use cases require the publish-subscribe communication model in order to send push notifications when an event occurs, instead of constantly polling a resource. This allows to decrease the power consumption as well as the messages on a network. To address this issue, the IETF proposes

RFC 7641 - Observing Resources in the Constrained Application Protocol (CoAP) [7]. In this document, the IETF proposes to use a GET request with a special option to be able to register to a resource. This way, a server can know that a client wants to receive notifications when that resource's representation changes and can send a notification, which is a regular CoAP response, again with an observe option. The option is used to register or deregister in a request, and in a response it indicates a sequence number for reordering notifications. The server decides what a change in the representation of the resource is, that is, periodic changes based on time, value changes or value changes out of a range. Fig. 1 shows the communication of subscribing to a resource, with its response and two notifications.

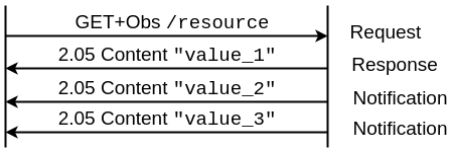

Figure 1. Subscribing to a resource and getting notifications.

Though, in some use cases the Observe extension for CoAP is not enough, due to the need for more complex mechanisms, as concluded in [10] and [9]. In those papers, the authors conclude that for a better fit of their proposal of a mapping of the IEC 61850 to CoAP, the subscription of the Observe extension needs to be enhanced. The needed enhancements are the possibility to subscribe to a CoAP resource while creating or changing its representation (PUT or POST request); not getting a resource representation when subscribing, but only when a notification happens; and being able to subscribe to a resource when sending a request to a related one, for instance requesting a configuration resource while subscribing to the resource the configuration affects to. This paper aims to fill that gap, offering more complex ways for handling resource subscriptions for clients which want to receive notifications. This work is a first step to propose new options and response codes for CoAP, and we also provide a short theoretical comparison of the overhead in a simple scenario.

The rest of this document is organized as follows. The next section presents the related work. Following, we propose new options and response codes for CoAP, to increase the capabilities of the subscribe mechanisms with the observe extension. Next, we validate our proposal comparing the overhead against the current options in a use case. After that, we provide a discussion on the results, also proposing future lines to continue the work. Finally, we conclude the paper.

RELATED WORK

The IETF's Constrained RESTful Environments (CoRE) group proposes several finished standards and drafts for resource-constrained devices and networks [8]. One of the finished standards is *Observing Resources in the Constrained Application Protocol* (RFC 7641) to enable devices to communicate using CoAP and the publish/subscribe communication paradigm. This extension can be divided in two parts, the subscription and the notification mechanisms. Since the publication of CoAP, a lot of work has been done to enhance CoAP notifications, but only limited improvements have been suggested for the subscription mechanisms. In the following, we briefly discuss these improvements before focusing on further enhancements of the subscription mechanisms in the rest of the paper.

IETF RFCs

Several improvements on top of the CoAP protocol have already been completed and published as RFCs, including the previously mentioned Observe, *Constrained RESTful Environments (CoRE) Link Format* (RFC 6690) [20], *Group Communication for the Constrained Application Protocol (CoAP)* (RFC 7390) [18], *Block-Wise Transfers in the Constrained Application Protocol (CoAP)* (RFC 7959) [2], and *PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)* (RFC 8132) [6].

IETF drafts

Apart from finished specifications, the IETF also publishes documents that are still being actively discussed within the community. For the present work, *Dynamic Resource Linking for Constrained RESTful Environments* (Dynlink) [23] and *Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)* (CoAP-PubSub) [13] are the most relevant, since they expand CoAP's publish-subscribe features, specifically regarding notification generation and delivery.

Dynlink defines parameters to configure conditional observation attributes. These parameters work not only with observing notifications, but also with push or polling requests. The attributes are included in the server on a table that is referred to as *binding table*, so when an event triggers, the server must check the table in order to decide whether a notification needs to be sent to subscribed clients. The defined parameters are: *bind* for the binding method (i.e. polling, observe or push); *pmin* for the minimum time to wait; *pmax* for the maximum time to wait; *st* for how much a value needs to change; *gt* for an upper limit value; *lt* for a lower limit value; and *band* for a bounded or unbounded value range.

CoAP-PubSub broker proposes to use a device as a broker for supporting nodes with limited availability. It decouples the clients and the server using an intermediary that centralizes the communication, thus being able to put clients or servers in sleep mode. Publishing clients push data using PUT or POST requests, while subscribed clients receive notifications or request information with GET requests. This document defines different functions to interact with the broker, i.e. *Discovery*, *Create*, *Publish*, *Subscribe*, *Unsubscribe*, *Read*, and *Remove*.

There are several other drafts, named active internet-drafts that do not relate directly with the Observe extension, but can be used in conjunction, such as *CoAP Simple Congestion Control/Advanced (CoCoA)* [1], *Representing Constrained RESTful Environments (CoRE) Link Format in JSON and CBOR* [14] and *CoRE Resource Directory* [22] among others. However, none of the finished specifications nor the active internet-drafts address the requirements for filling the gap presented in the introduction for more advanced subscription mechanisms.

Other proposals

Apart from the discussed IETF proposals, several works propose ways to enhance the observation of CoAP resources in the

related literature. These works focus on optimizing the notification number that the servers send, with different approaches, e.g. semantic meaning for the resources, using conditions through queries or options, or using intermediary devices.

In [12], Ketema et al. propose *conditional observation*, where clients specify notification criteria in conjunction with the Observe option. Instead of using a query (such as [24]), they authors propose new CoAP options, whose descriptions are drafted in [15]. The described options are *Minimum-Interval* and *Maximum-Interval*. Using 5 bits for the option, there are 32 different observation types, including *time series*, *maximum response time*, *minimum response time*, *step*, *All Values Less/Greater/Equal*, and *Periodic*. The authors also demonstrate that their proposed options are useful from an application and from a network efficiency point of view; still, this draft is long expired and abandoned.

Going beyond the conditional Observe, Mietz et al. [17] propose to use a new option, called *High-Level State Option*, to give semantic meaning to values that are exposed by a resource. To do that, a POST request is sent to the resource that includes this option, and the server creates a subresource with an arbitrary URI that can then be queried by the client. For example, when having a temperature resource, the subresource can be the meaning of the temperature, i.e. *cold*, *warm* or *hot*. This approach has three main advantages: i) it gives semantic meaning to the raw values of the sensors, ii) it needs fewer notifications and iii) it simplifies the management of the subscribed clients compared to the conditional Observe, as it does not need to manage them one by one.

Tanganelli et al. [25] describe another way of limiting the number of notifications sent. Their approach relies on proxies to support and optimize notification periods, similar to a broker. The difference to other proposals is that the proxy subscribes to the servers, instead of getting PUT requests from the publishing client. Each subscribed client can specify the conditions to be interested in the notifications but only the proxy subscribes to the server. A resource constrained client may not be computationally able to manage many subscribers so the proxy can act as the intermediary and manage the notifications it receives from the server to be able to notify each subscribed client with their time requirements. The authors propose an algorithm to calculate the optimal period for the proxy to receive the notification from the publishing client to comply with all the subscribed clients.

Ludovici et al. [16] also propose to reduce the sent notifications by specifying Quality of Service (QoS) parameters to support timeliness. In this proposal, 4 levels of priorities are used to determine the delivery order of notifications and to select which notifications a specific subscriber receives. This way, the client can specify which notifications can be dropped if the server can not manage all of them. An observing client demands a priority level when it subscribes and the server can accept that level or negotiate it. With this approach, observers with different roles can have different requirements, leading to a reduction on the consumed energy and improvement on the delivery delay and delivery ratio.

In [19], Sacramento et al. propose a framework for planning registration states, and aggregation and scheduling of notifications in proxies to maximize energy saving, use available bandwidth efficiently, and reduce delays. When there are multiple notification requests in a network optimization, issues such as the needed registration steps, energy saving, consistency and caching are raised. . CoAP and its Observe extension can use caches and proxies to improve scalability and efficiency. Optimizing those can improve the overall performance of a system. According to the authors, planning the proxies to aggregate and schedule notifications can achieve huge improvements. Continuing this work, in [5], Correia et al. take timeliness fairness into consideration. They argue that, since the network's lifetime is the time while *all* nodes are up, the timeliness of messages should be as evenly distributed as possible to maximize lifetime (since this will cause nodes to deplete their batteries in a balanced way). They propose to take into account fairness on registration steps, planning data forwarding through proxies in a way that registration steps impose equity among the nodes. In [4], the authors continue the work on [19] and [5] and add a heuristic approach for developing the algorithm to make the decisions on the registration steps of observers.

According to Ishaq et al. [11], CoAP's Observe and Group extensions do not work entirely well together because they have not been designed to work together and it is not possible to combine them to observe a group of resources, so the authors propose an entity named *Entity Manager* (EM) to manage the observation of groups. The EM pushes changes in a group – after aggregation – to the observing clients, instead of each resource from each group individually notifying each client, thereby reducing the amount of communication. To create entities (resource groups), a resource "/e" is defined which only supports POST request. The created entity acts as a proxy for the entire group and works like a regular resource: it can be polled, pushed to, and observed.

In [3], Choi et al. propose an extension of CoAP, using a clustering approach. The sensors are grouped into a cluster and a node acts as the head, aggregating and transmitting data as a proxy. This approach improves the performance in terms of bandwidth consumption and transmission time and allows many-to-many communications based on CoAP. The cluster head uses topics to represent the attributes of sensors and is the intermediary to be able to receive information from the sensors. The client nodes need to register to the cluster head with the topics and after that, the cluster receives messages and redirects them to the sensors. The authors define three different options: NORMAL to send normal condition of the server; DIRECT for packets needed to be send immediately; and LIMIT to indicate that the message has a deadline of stored time.

All these papers focus on the generation and delivery of notifications. However, none of them tackles the problems presented in [10] and [9], regarding the subscription mechanisms, as summarized in the Introduction.

ENHANCED COAP SUBSCRIPTION MECHANISMS

In this section, first we define the requirements for more advanced subscription mechanisms and then propose a solution.

- Subscribe with PUT/POST. With the current IETF specifications, it is not possible to update or create a resource and subscribe to that resource in just one step. With this requirement a client can create or update a resource and at the same time subscribe to it.
- Response does not have to be a resource representation in all cases; sometimes a "Subscribed" message can be sufficient. In some cases, a client may not be interested in the current situation of the resource, it may only need to be notified when something happens. Also, the resource may not have valid values or may even not exist yet.
- Subscribe through other resources. As explained in the use cases given in [10] and [9], some resources may be related to others. For example, a resource may configure the notification generation for another, thus, being able to change a configuration resource and to subscribe to the resource the configuration affects to is convenient to decrease network usage.

To shed light on these issues, we propose the creation of new CoAP options and new CoAP response codes. Dynamic environments or networks would specially benefit from them, as they focus on the subscription mechanisms and devices coming in and out of a network are more likely to be subscribing and unsubscribing more often.

Subscribe with PUT/POST

With the current Observe extension, PUT and POST requests cannot include the Observe option. Hence, we propose to use it in a similar way as GET requests. In this case, the interaction would be the same as GET requests with the Observe option. The server will answer with a "2.01 Changed" or a "2.04 Created" response and the Observe option with the identifier for the notification number. In this way, clients can create and update resources, and subscribe to them in a single step. After the first response, the following notifications will have the "2.05 Content" code and work exactly like the notifications in RFC 7641 [7]. The subscription and the notification process can be seen in Figure 2.

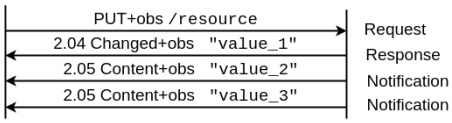

Figure 2. Update the values of a resource, subscribe to it and get notifications.

Same as for a regular GET subscription, if the subscription to a resource fails, the response will not include the Observe option.

Lightweight responses

To be able to subscribe to a resource and get an acknowledgment of the subscription without the representation, we propose a new CoAP option, named no-payload. We proposed option is the first one presented in Figure 9. We define the "24 - No payload" option. It is not critical, if a server does not recognize the option, it sends the current representation of the resource. This option requires the Observe option to be also part of the request. With this option new response codes are needed, presented in Figure 10, i.e. "2.10 Subscribed", "2.11 Created and Subscribed" and "2.14 Changed and Subscribed", implying that the client has been successfully subscribed, but there is no resource representation on the response. "2.15 Content and Subscribed" is for responses that include a payload but that payload is the representation of the resource the client is sending the request to, not from the resource the client is subscribing to. This way, the client gets the representation of the resource it sends the request to and subscribe to a related resource. The rationale for selecting the codes is to use the 2.1X for the subscribed responses and use the codes similar to the regular non subscribed codes:

- Subscribed: 2.10
- Created and Subscribed: 2.11, similar to 2.01 Created
- Changed and Subscribed: 2.14, similar to 2.04 Changed
- Content and Subscribed: 2.15, similar to 2.05 Content

Figure 3 shows the interaction for a GET request, with the client interested in receiving notifications, not in the current state of the resource. The client sends a GET, with Observe and no-payload options. The response is a "2.10 Subscribed" if everything goes well. If the subscription fails, a "5.00 Internal Server Error" or a "5.03 Service Unavailable" are the responses, depending on whether a MAX-AGE option is included. If other errors occur, regular CoAP error codes will be used.

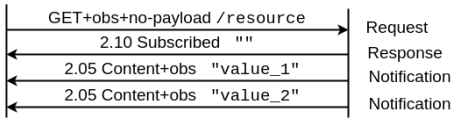

Figure 3. Subscription to a resource without getting the current state and get notifications.

With a simple PUT request, the response can be 2.01 or 2.04, depending on whether the resource previously existed or not. Figure 4 shows a extended PUT response with the Observe and no-payload options. As the resource existed previously, the response is "2.11 Changed and Subscribed", indicating that the resource has been correctly updated and that the client has been correctly subscribed, but as it does not need the representation of the resource, the server does not include it in the response. On the other hand, if the resource did not previously exist, the response will be a "2.14 Created and Subscribed". POST request work in the same way. If the subscription fails but the resources have been able to be created or updated, the server will respond with "2.01 Changed" or "2.04 Created" responses. If the error is of any other kind, regular CoAP error codes will be used.

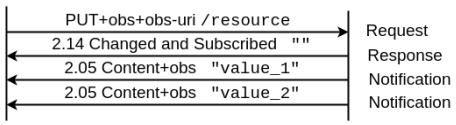

Figure 4. Change a resource, subscribe to it but do not receive a payload, then get notifications.

Subscribe through other resources

To be able to subscribe to a resource through a related resource, e.g. via a configuration resource, we propose a new option: "43 - Observe-uri". This option is useful for cases when a configuration resource is reachable and to be able to change or create that configuration with a PUT or POST request and subscribe to the resource to whom the configuration belongs in a single request, or to read the configuration resource with a GET. This is a critical option as the client is expecting to subscribe to another resource, thus if the server can not do that, the request can not be handled adequately.

Figure 5 shows how to request the configuration resource and at the same time subscribe to the resource the configuration affects to. The response will be a "2.15 Content" with the resource representation of the configuration resource. If the subscription has not been successful, the response will fall back to a "2.05 Content", also with the configuration resource's representation. For other problems, regular error codes apply. If the related resource does not exist, the response will be a "4.02 Bad-Option" with the observe-uri option present.

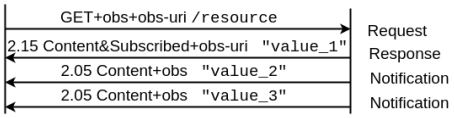

Figure 5. Get the representation of a resource, subscribe to a different resource and get notifications.

POST and PUT requests have a similar behaviour, they will get "2.01 Created" or "2.04 Changed" responses, depending on whether the resource previously existed or not. Figure 6 presents a PUT request to an already existing resource, and POST request have an analog behaviour. If the resource has been successfully created or updated but the subscription has not been successful, the response will not include the Observe option. For other problems, regular error codes apply. Same as a GET request, if the related resource does not exist, the response will be a 4.02 Bad-Option and the observe-uri option will be included.

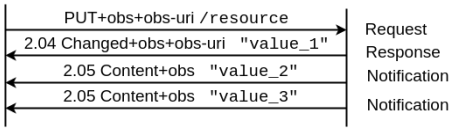

Figure 6. Update a resource, subscribe to a different resource and get notifications.

The observe-uri option can also be combined with the no-payload option. Figure 7 presents an interaction of a client subscribing to a related resource and which does not need the current state of the resource. At a glance, this request seem to not make any sense as clients could subscribe directly sending a GET request to the observe-uri with the no-payload option active and the result would be the same. However, with this interaction, the client can subscribe to a resource while checking the existence of the configuration resource. For that, the client sends a GET request, with the Observe, no-payload and observe-uri options. The response will be a "2.10 Subscribed" if everything went right. If an error occurs, previous handling mechanisms are followed, with regular error codes for general errors and with 4.02 Bad-Option and the observe-uri option response if the referenced uri does not exist or "5.00 Internal Server Error" or a "5.03 Service Unavailable" if the server can not handle the subscription, depending on whether a MAX-AGE option can be defined or not.

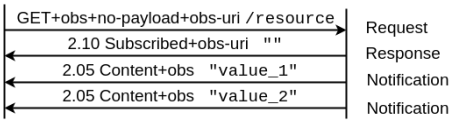

Figure 7. Poll a resource without getting the representation, subscribe to a different resource and get notifications.

With a PUT or POST request, the interaction is similar to GET requests. The response will be a "2.11 Changed and Subscribed" or a "2.14 Created and Subscribed" with the observe-uri option and no payload. If the client can not be subscribed, the server will fall back to a "2.01 Changed" or "2.04 Created" response code. If the resource observe-uri indicates does not exist, the error code will be "4.02 Bad Option" with the observe-uri option present. If the error is of any other kind, the handling must be done as per previous standards. Figure 8 shows a successful subscription with a PUT response to a resource that already existed.

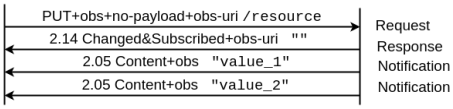

Figure 8. Update a resource without getting the representation, subscribe to a different resource and get notifications.

As it can be seen in all the interaction figures, these new option and response codes only affect the subscription mechanism. The notifications work exactly the same way they work with the previous observing standard. In the event of other errors occur during the subscription process, the errors must be handled the way they are handled with the current standards, e.g. "4.04 Not Found" response if the resource the client is sending a request to does not exist.

Figure 9 shows the new proposed options.

No	C	U	N	R	Name	Format	Length	Default
24					No-payload	empty	0	(none)
43	x	x	-	x	Observe-uri	string	0-255	(none)

Figure 9. New proposed options for CoAP.

Following RFC 7252's rules, our proposal is to use the code number 24 for "no-payload" option. This option is elective (non-critical), as it can be safely ignored. If ignored, the server sends the representation of the resource. It is safe to forward for a proxy for the same reason and it is not repeatable. The meaning of the option is to indicate whether a payload is required in the response, hence, it has no length nor format. The proposed "observe-uri" option is similar to the "uri-path" option, and we propose the 43 code number. It indicates to which resource the client wants to subscribe. It is critical,

unsafe to forward and repeatable, one option per resource path. The indicated resource path is represented in string format and has a maximum length of 255. The uri is relative to the resource of the uri-path of the option, hence, contrary to other options related to paths, ".." is allowed. To be able to subscribe to the resource uri represented with this option, it must be observable.

Figure 10 shows the new proposed response codes, codes that imply a successful subscription to notifications.

Code	Description
2.10	Subscribed
2.11	Created and Subscribed
2.14	Changed and Subscribed
2.15	Content and Subscribed

Figure 10. New response codes for CoAP.

VALIDATION

This is a first step and a theoretical analysis of the exchanged messages is provided as validation, an implementation and testing is out of scope of this work. For the analysis, a simple use case is presented, an alarm clock. The alarm clock has two resources, `time` and `time/alarm`. In this scenario, we compare the size of the request and the responses for different actions using the current CoAP option and codes, and the proposal of this document. Table 1 shows the actions and the messages that need to be exchanged to carry out those actions.

To measure the sizes of the messages, it has to be taken into account the size of each part of the header and the payload.

- Basic header: 4 bytes.
- Observe option: 2 bytes.
- Uri-Path option:
  - `time`: 5 bytes.
  - `alarm`: 6 bytes.
- Observe-uri option:
  - Request: `alarm`: 6 bytes.
  - Response: 1 byte, just the option.
- No-payload option: 1 byte.
- FF byte, to separate the header and the payload: 1 byte.
- Payload: 16 bytes. In this case, we include the payload also on 2.04 responses, to ascertain that the resource has been successfully updated.

Other options are left out of the comparison as they will be exactly the same for all cases, e.g. Uri-Host, Uri-Port, Content-format or Max-Age.

Figure 11 presents the bytes needed to be exchanged to be able to get, update and subscribe to the clock and to the alarm, using the already existing options and compared to our approach. The different uses are presented in Table 1, and the messages are grouped. The first column of each pair represents the bytes exchanged with the current options of CoAP, divided in the request and response (purple) and the subscription request and the response (blue). The second column, in green, represents the bytes exchanged with the new approach.

The bar chart of Figure 11 clearly shows that the bytes required for a successful subscription can be reduced using the options and response codes presented in this document. In each of the actions, the network usage is reduced significantly. On the one hand, not requiring a payload in the response significantly reduces the responses' size, as the payload is usually the most heavy part of a CoAP message. On the other hand, reducing the need for two interactions (action and subscription) to just one step has an even bigger impact on the overhead.

DISCUSSION

After presenting a proposal for new CoAP options and response codes and validating them, some conclusions can be provided. First, with these new option and responses a client is able to create or update a resource and subscribe to it in a single step, thus, reducing the needed messages from four (PUT or POST, response, subscription request and subscription response) to two (request and response).

Second, in some cases getting a current representation of the resource might not be necessary when subscribing to it. In the case of the work presented in [10] and [9], some reports are generated and a client subscribes to them. When subscribing to the reports, they might not exist yet or the client might not want to receive them yet. These reports can be large, needing to use the blockwise transfer extension of CoAP, thus requiring several messages with parts of the resource representation and acknowledgments. With the option of not sending the current representation this can be avoided.

Last, a client might want to create, update or get the representation of a resource while subscribing to a related one. This is specially interesting when a main resource includes a configuration resource, which can be consulted, updated or created while subscribing to the main resource.

With these options, we reduce the number and the size of the messages exchanged when subscribing to a resource. As it affects only the registration steps, the defined options are more optimal in dynamic networks or mobile devices, where devices subscribe and unsubscribe frequently, such as smart vehicles.

From this work, different lines for future work arise. As previously said, this is a first step with a proposal. The validation in this paper has been a theoretical analysis of the exchanged messages. The first line for future work is to implement these options in a library and take measurements in order to check the achieved improvement in terms of time to process and consumed energy in contrast to the current version.

After that first implementation, the next step is to include this options in the work presented in [10] and [9], where the needs addressed in this paper come from. This way, the mapping of the IEC 61850 to CoAP will be completed with a more optimal approach.

	Action	New Proposal	Previous way
1	Subscribe to an alarm, without needing to know when	GET+obs+no-payload /TIME/ALARM	GET+obs /TIME/ALARM
2	Get the current time and subscribe to an alarm	GET+obs+observe-uri /TIME subscribe to /TIME/ALARM	GET /TIME GET+obs /TIME/ALARM
3	Check that the time exists and subscribe to the alarm	GET+obs+no-payload+observe-uri /TIME subscribe to /TIME/ALARM	GET /TIME GET+obs /TIME/ALARM
4	Change the alarm and subscribe	PUT+obs /TIME/ALARM	PUT /TIME/ALARM GET+obs /TIME/ALARM
5	Change the alarm and without needing a response and subscribe	PUT+obs+no-payload /TIME/ALARM	PUT /TIME/ALARM GET+obs /TIME/ALARM
6	Change the time and subscribe to the alarm	PUT+obs+observe-uri /TIME subscribe to /TIME/ALARM	PUT /TIME GET+obs /TIME/ALARM
7	Change the time and subscribe without needing the alarm	PUT+obs+no-payload+observe-uri /TIME subscribe to /TIME/ALARM	PUT /TIME GET+obs /TIME/ALARM

Table 1. Action for a clock use cases with current mechanisms and with the new approach.

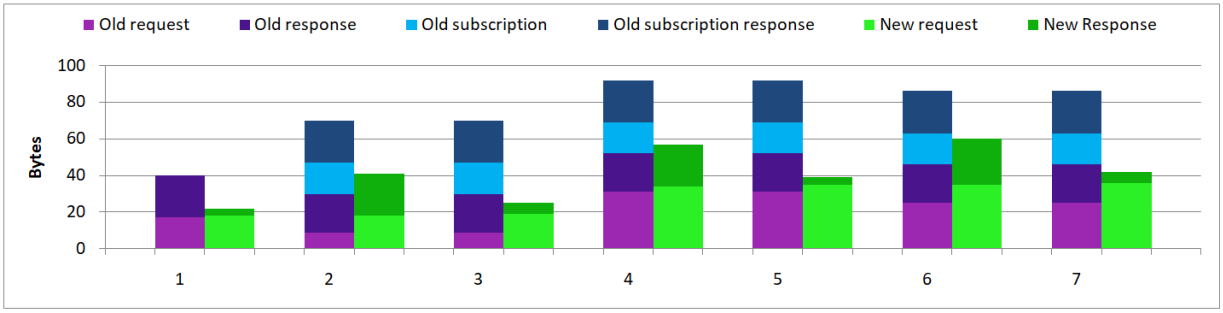

Figure 11. Overhead in the old and new approaches in different usages of a clock and alarm use case.

CONCLUSION

In this paper, we aim to extend CoAP's observe extension in terms of subscription mechanisms. For that, first we review the existing CoAP standard, drafts and papers and conclude that while enhancement for notification processes for CoAP's observe extension have been proposed, the subscription mechanisms have been left out of those proposals. Due to the needs arisen on [10] and [9], we propose two new options for CoAP, i.e. no-payload and observe-uri and four new response codes, i.e. 2.10, 2.11, 2.14 and 2.15. With this options, a client can subscribe to resources without getting the current state of the representation if it is not interested and can also subscribe while changing or creating a resource in one step. After presenting the options and how does a message exchange work, we validate the approach comparing the usage of the new options against the previous mechanisms in a simple alarm clock use case. We show that the bytes required to exchange information decrease with this approach. Finally, we offer a discussion of the results and future lines of work.

ACKNOWLEDGMENTS

This work has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under the MegaM@Rt2 project (Grant agreement No. 737494). This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation program and from Sweden, France, Spain, Italy, Finland & Czech Republic. Besides, this work has been partially supported by the Basque Government through the Elkartek program under the TEKINTZE project (Grant agreement No. KK-2018/00104).

REFERENCES

1. Carsten Bormann, August Betzler, Carles Gomez, and Ilker Demirkol. 2018. *CoAP Simple Congestion Control/Advanced*. Internet-Draft draft-ietf-core-cocoa-03. Internet Engineering Task Force. `https://datatracker.ietf.org/doc/html/draft-ietf-core-cocoa-03` Work in Progress.

2. Carsten Bormann and Zach Shelby. 2016. Block-Wise Transfers in the Constrained Application Protocol (CoAP). RFC 7959. (Aug. 2016). `DOI: http://dx.doi.org/10.17487/RFC7959`

3. Dong-Kyu Choi, Joong-Hwa Jung, Hyung-Woo Kang, and Seok-Joo Koh. 2017. Cluster-based CoAP for message queueing in Intemet-of-Things networks. In *International Conference on Advanced Communication Technology, ICACT*. IEEE, 584–588. `DOI: http://dx.doi.org/10.23919/ICACT.2017.7890157`

4. Noélia Correia, David Sacramento, and Gabriela Schutz. 2016. Dynamic Aggregation and Scheduling in CoAP/Observe-Based Wireless Sensor Networks. *IEEE Internet of Things Journal* 3, 6 (2016), 923–936. `DOI: http://dx.doi.org/10.1109/JIOT.2016.2517120`

5. Noélia Correia, Gabriela Schutz, and Alvaro Barradas. 2015. Fairness for CoAP/Observe based wireless sensor networks with aggregation deployment. In *IEEE World Forum on Internet of Things, WF-IoT 2015 - Proceedings*. IEEE, 110–115. `DOI: http://dx.doi.org/10.1109/WF-IoT.2015.7389036`

6. Peter Van der Stok, Carsten Bormann, and Anuj Sehgal. 2017. PATCH and FETCH Methods for the Constrained Application Protocol (CoAP). RFC 8132. (April 2017). DOI:http://dx.doi.org/10.17487/RFC8132

7. Klaus Hartke. 2015. Observing Resources in the Constrained Application Protocol (CoAP). RFC 7641. (Sept. 2015). DOI:http://dx.doi.org/10.17487/RFC7641

8. IETF. 2018. Constrained RESTful Environments (core). (2018). https://datatracker.ietf.org/wg/core/documents.

9. Markel Iglesias-Urkia, Diego Casado-Mansilla, Simon Mayer, and Aitor Urbieta. 2018. Validation of a CoAP to IEC 61850 Mapping and Benchmarking vs HTTP-REST and WS-SOAP. In *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*.

10. Markel Iglesias-Urkia, Aitor Urbieta, Jorge Parra, and Diego Casado-Mansilla. 2017. IEC 61850 meets CoAP: Towards the integration of Smart Grids and IoT standards. *ACM International Conference Proceeding Series* Part F132741 (2017). DOI: http://dx.doi.org/10.1145/3131542.3131545

11. Isam Ishaq, Jeroen Hoebeke, Ingrid Moerman, and Piet Demeester. 2016. Observing CoAP groups efficiently. *Ad Hoc Networks* 37 (2016), 368–388. DOI: http://dx.doi.org/10.1016/j.adhoc.2015.08.030

12. Girum Ketema, Jeroen Hoebeke, Ingrid Moerman, Piet Demeester, Li Shi Tao, and Antonio J. Jara. 2012. Efficiently observing Internet of Things resources. In *Proceedings - 2012 IEEE Int. Conf. on Green Computing and Communications, GreenCom 2012, Conf. on Internet of Things, iThings 2012 and Conf. on Cyber, Physical and Social Computing, CPSCom 2012*. IEEE, 446–449. DOI:http://dx.doi.org/10.1109/GreenCom.2012.70

13. Michael Koster, Ari Keränen, and Jaime Jimenez. 2018. *Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)*. Internet-Draft draft-ietf-core-coap-pubsub-04. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-core-coap-pubsub-04 Work in Progress.

14. Kepeng Li, Akbar Rahman, and Carsten Bormann. 2018. *Representing Constrained RESTful Environments (CoRE) Link Format in JSON and CBOR*. Internet-Draft draft-ietf-core-links-json-10. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-core-links-json-10 Work in Progress.

15. Shitao Li, Kepeng Li, Jeroen Hoebeke, Floris Van den Abeele, and Antonio J. Jara. 2014. *Conditional observe in CoAP*. Internet-Draft draft-li-core-conditional-observe-05. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-li-core-conditional-observe-05 Work in Progress.

16. Alessandro Ludovici, Eduardo Garcia, X. Gimeno, and Anna Calveras Augé. 2012. Adding QoS support for timeliness to the observe extension of CoAP. In *International Conference on Wireless and Mobile Computing, Networking and Communications*. IEEE, 195–202. DOI: http://dx.doi.org/10.1109/WiMOB.2012.6379074

17. Richard Mietz, Philipp Abraham, and Kay Römer. 2014. High-level states with CoAP: Giving meaning to raw sensor values to support IoT applications. In *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*. IEEE, 1–6. DOI: http://dx.doi.org/10.1109/ISSNIP.2014.6827637

18. Akbar Rahman and Esko Dijk. 2014. Group Communication for the Constrained Application Protocol (CoAP). RFC 7390. (Oct. 2014). DOI: http://dx.doi.org/10.17487/RFC7390

19. David Sacramento, Gabriela Schutz, and Noélia Correia. 2015. Aggregation and scheduling in CoAP/Observe based wireless sensor networks. In *IEEE International Conference on Communications*, Vol. 2015-September. IEEE, 654–660. DOI: http://dx.doi.org/10.1109/ICC.2015.7248396

20. Zach Shelby. 2012. Constrained RESTful Environments (CoRE) Link Format. RFC 6690. (Aug. 2012). DOI: http://dx.doi.org/10.17487/RFC6690

21. Zach Shelby, Klaus Hartke, and Carsten Bormann. 2014. The Constrained Application Protocol (CoAP). RFC 7252. (June 2014). DOI: http://dx.doi.org/10.17487/RFC7252

22. Zach Shelby, Michael Koster, Carsten Bormann, Peter Van der Stok, and Christian Amsüss. 2018a. *CoRE Resource Directory*. Internet-Draft draft-ietf-core-resource-directory-13. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-core-resource-directory-13 Work in Progress.

23. Zach Shelby, Matthieu Vial, Michael Koster, Christian Groves, Julian Zhu, and Bill Silverajan. 2018b. *Dynamic Resource Linking for Constrained RESTful Environments*. Internet-Draft draft-ietf-core-dynlink-05. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-core-dynlink-05 Work in Progress.

24. Zach Shelby, Matthieu Vial, Michael Koster, Christian Groves, Julian Zhu, and Bill Silverajan. 2018c. *Reusable Interface Definitions for Constrained RESTful Environments*. Internet-Draft draft-ietf-core-interfaces-11. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-core-interfaces-11 Work in Progress.

25. Giacomo Tanganelli, Carlo Vallati, Enzo Mingozzi, and Matthias Kovatsch. 2017. Efficient proxying of CoAP observe with quality of service support. In *2016 IEEE 3rd World Forum on Internet of Things, WF-IoT 2016*. IEEE, 401–406. DOI: http://dx.doi.org/10.1109/WF-IoT.2016.7845444