Proceedings of the Federated Conference on Computer Science and Information Systems pp. 637–643
DOI: 10.15439/2018F278
ISSN 2300-5963 ACSIS, Vol. 15

FetchIoT: Efficient Resource Fetching for the Internet of Things

Badis Djamaa, Mohamed Amine Kouda, Ali Yachir, and Tayeb Kenaza
Ecole Militaire Polytehnique,
BP 17, Algiers, Algeria
{badis.djamaa, kouda.amine, ali.yachir, ken.tayeb}@gmail.com

***Abstract*—Finding the right resource at the right time and space is a key enabler for a wide adoption and spread of the Internet of Things (IoT). The Constrained Application Protocol (CoAP) and related standards are among the most prominent efforts working towards such a goal. Indeed, CoAP-related standards provide interesting mechanisms for resource discovery in both centralized and distributed architectures based on the CoAP's GET method. In this paper, we, first, highlight the limitations of GET-based discovery mechanisms. The paper, then proposes a new solution using the recently standardized FETCH method and develops its specifications, rules and semantics. The proposed solution is implemented in the recently released, secure and reliable OpenThread platform and compared with GET-based approaches in different home automation scenarios. Obtained results demonstrate the performance of FETCH-based discovery in achieving fine-grained, time-efficient and reliable discovery while preserving network resources.**

I. Introduction

WITH the growing number of sensors, actuators, devices, smartphones and embedded chips, along with the (r)evolution of computer and network technologies, the world is talking more and more about the Internet of Things (IoT); a term designating the extension of the Internet to everyday objects. When interconnected, these objects can form a network for measuring, storing, transferring, processing, and exchanging data between physical and virtual worlds. Today, such smart objects are able to discover, detect and exchange messages across the Internet thanks to the newly introduced protocols such as 6LoWPAN [1], RPL [2] and CoAP [3]. In fact, with the provided features, smart object networks can be built spontaneously and can be doted with capabilities of self-configuring, self-regulating and self-healing.

In IoT, the vision is that a significant number of new devices including refrigerators, clothes, cars, and traffic-lights will be dynamically connected to the Web for communication, command and control of the surrounding environment. This trend creates the so-called Web of Things (WoT) with the introduction of new constrained servers that have different features from traditional web servers and users. This pushes the WoT to face many challenges related mainly to the heterogeneous nature of networks constituted by these objects and their very limited capacity in terms of computing resources, communication capabilities, memory and energy. To overcome such challenges, the offered functionalities in the WoT are encapsulated as autonomous constrained REST (Representational State Transfer) resources [4] that are accessible from other objects or traditional Web services. This simplifies transparent integration of the physical world with the virtual one. However, to do so, there must be mechanisms to discover available resources and their capabilities with the minimum of human intervention. Thus, resource discovery becomes a fundamental requirement for the success of any IoT solution.

One of the main protocols implementing the REST-based mechanism for resource description and discovery in the web of things is CoAP [3]. Indeed, besides being the de-facto standard for data exchange in the WoT, CoAP provides distributed and centralized solutions for achieving resource discovery. It does so by employing the GET method for the sake of finding available resources in an IoT environment. For instance, a device searching for available temperature sensors in its environment issues a GET request to a *well-known* URI (Uniform Resource Identifier) asking for all sensors offering resources of type temperature. The queerer will get responses with the description of such resource and chooses the ones that best meets its needs.

This GET based mechanism is limited in many aspects that will be discussed and detailed in this paper. To overcome such issues, we introduce a new usage of the FETCH method [5] for the sake of efficient resource discovery in the IoT. The specification of such a usage along with the definition of rules allowing to achieve rich, expressive and compact resource discovery in the web of things are the main contributions of this paper.

Finally, it should be noted that to the best of our knowledge, this is the first paper introducing the use of the newly standardized FETCH method for resource discovery in the IoT. The paper also adds a resource discovery layer to the secure and reliable openThread networking protocol making the proposed approach ready for commercial deployments.

The remainder of this paper is organized as follows: Section II reviews related resource discovery work over CoAP. This will be followed by identifying the main issues of such approaches before proposing our approach and detailing its mechanisms in section III. section IV is devoted to implementing and evaluating the performance of the proposed approach in multiple IoT scenarios over the commercially-proven, secure Thread platform. The paper ends in Section V with a conclusion and directions for future work.

II. Related Work

Resource discovery is a well-investigated topic in traditional Networks with a plethora of solutions proposed in the literature. IoT objects have radically different features than traditional Web servers. As a result, traditional discovery approaches can not be applied directly and will not produce accurate and efficient results in many IoT scenarios. Consequently, new solutions are emerging for the IoT. Such solutions follow two main architectures, centralized and distributed, with standards being proposed for both architectures. The most promising ones are based on CoAP and/or DNS-SD [6]. While DNS-SD is starting to get intention, currently, CoAP-based discovery is the main standard solution in today's IoT applications.

CoAP-based resource discovery can be achieved via three main mechanisms, namely: CoAP Resource Directory (RD) [7], CoAP Distributed Resource Directory (DRD) [8], and CoAP Resource Discovery [9]. The main purpose of these mechanisms is to provide URIs, also called links, for the resources available within a server, as well as the attributes that describe them [9].

With the RD, all the resources offered by the servers are saved in a single directory so that clients can discover any required resource by looking up the RD. For instance, once the RD has been successfully discovered, a server can register its resources in the RD by performing a POST request to the path indicated by the RD. When a client wants to search the RD, they must issue a GET request to the RD. For this, the client uses a specific request to obtain the results that correspond to its interest. The use of GET imposes many constraints on the expressiveness of the request since the parameters must be "bundled up in some unspecified way into the URI" [5]. It should be noted that Following the success of RD, many solutions including [10] have been proposed. All, however, suffer from the same problems related to the use of GET for resource discovery.

In DRD-based discovery [8], before an object can register its resources, it must find an Entry Point (EP) at the DRD. The initial EP can be any object connected to the DRD. In order to find an EP, one method is to use a multicast address */.well-known*, where the object sends a POST request to that */.well-known* address to obtain the DRD information. Other means include searching for the nearest EP or DRD using dynamic discovery [8]. To improve this approach, work based on hierarchical repertoires has been proposed. Indeed, authors of [11], have introduced a usage of the REsource LOcation And Discovery (RELOAD) protocol [12] to discover CoAP resources. RELOAD forms an overlay network to provide storage and messaging services in a peer-to-peer (P2P) environment and allows applications to define specific use cases. For instance, [11] authors describe how to use CoAP with RELOAD to discover interconnected CoAP resources across a large geographic area. However, as with the RD, the discovery is based on GET, which in addition to the above limitations, only supports discovery in the CoRE Link Format (CLF) [9].

Direct approaches rely on IP multicast to achieve discovery [9]. Similarly to the above approaches, it uses the GET method to diffuse a request to all nodes in an IP domain targeting the well-known URI (*/.well-known/core?search**) of all nodes members of the group's multicast address. Unlike RD and DRD, here, not being able to filter the request may generate a huge number of irrelevant responses that consume network resources and slow its operations. Thus, direct resource discovery must include the *search** filter in its requests, which is still insufficient for fine-grained efficient discovery. Finally, a hybrid approach that tries to take advantages of both direct discovery and RD is proposed in [13]. However, similarly to the above, it also relies on the GET method to formulate the requests.

To the best of our knowledge, all CoAP-based discovery approaches deploy the GET method that limits their capabilities into achieving rich and concise discovery in IoT. Such limitations will be discussed in the following section before introducing our FETCH-based resource discovery for the IoT approach.

III. FETCH-based resource discovery for the IoT

Before introducing the FETCH-based resource discovery for the IoT, the following subsection identifies and discusses the main issues with the GET-based approach.

A. Issues with the GET-based resource discovery

As in HTTP, the GET method is used to obtain the complete representation of a resource, which can be refined according to the additional parameters conveyed in the request. However, using GET, a user/device can only allow the specification of a URI and the query parameters in CoAP options [3] as can be seen in Figure 1. Indeed, the GET method does not support the transmission of a payload detailing the request, which generates verbose replies (Figure 1) that consume energy and throughput. These restrictions have caused some applications to use the POST method for the sake of formulating queries with semantic alteration and standard compatibility violation [5].

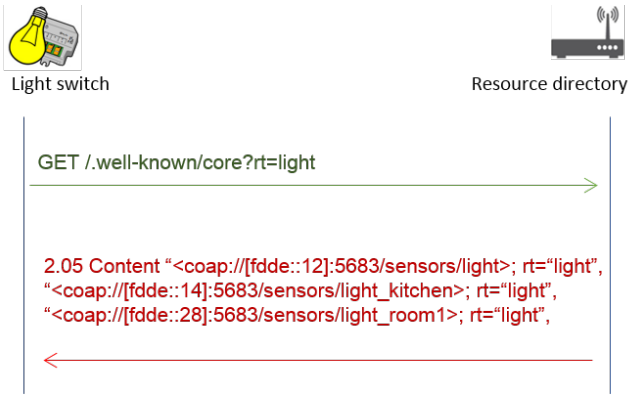

Fig. 1. An example of GET-based resource discovery

The following points summarize and discuss some of the major limitations and gaps related to using GET for the sake of resource discovery:

- Query parameters of GET-based requests are limited by the form and the size of the URI, which restricts their capacities to convey user/device requirements. This constraint imposes strict limitations on the expressiveness of requests, which may result in generating verbose replies that will, at the end, be discarded by the client.
- Despite some research efforts aiming to increase the number of filters included in a GET request, until now, CLF [9] limits this number to one by query. Indeed, a GET discovery request follows the scheme: (*/.well-known/core ?search**), where *search* represents a single parameter. For example: *GET /.well-known/core?rt=light*.
- The GET method does not support filtering based on logical operations (AND, OR, <,>,> = ...) between query parameters. These types of filtering are necessary to refine queries to receive only the most relevant answers.
- With GET, we can not include and/or exclude nodes/resources based on cached descriptions or any information known by a user or device. The use of these features is important for resource discovery in the IoT. Indeed, with such features, the user/device will have the possibility to specify the known resource descriptions in order not to include them in the answer. More importantly, having such an option will provide the possibility of specifying particular nodes to avoid or include, as well as the specification of the particular requirements in terms of security, reliability and the required quality of service.
- With GET, we can not specify/limit the search domain, the location of the searched resources, the maximum number of hops a query can reach, and so on. Such information is of paramount importance for a more fine-grained, effective and efficient discovery. For example, a client looking for temperature sensors in his immediate environment is not interested in having answers from all the temperature sensors available in the network.
- Finally, GET-based resource discovery only supports discovery operations in the CoRE link format [9], which limits its applicability when resources are described in other formats such as JSON, CBOR, EXI, etc.

From the above, and knowing that in IoT most objects are very constrained in terms of resources, a substitute mechanism that offers an explicit, compact and comprehensive expression of user requirements is required. Indeed, it is very important for an alternative approach to minimize congestion, excessive use of resources, energy consumption and latency, while offering more relevant results to better satisfy user requests. Such an approach will be the subject of the following sections.

B. The CoAP FETCH method

The FETCH method has been proposed in draft-ietf-core-etch-04 [5], which has just become RFC 8132 [14]. FETCH tries to provide a solution that covers the gap between the use of GET and POST. In the same way as POST, the parameters of FETCH are transmitted in the payload of the request rather than in the context of its URI. However, unlike POST, the semantics of FETCH is more specifically defined to ensure tasks similar to those of GET.

As defined in RFC 8132 [14], the FETCH method of CoAP is used to get a representation of a resource, providing a number of query parameters. Unlike GET, which requires a server to return a representation of the resource identified by the request URI (as defined by RFC 7252 [3]), the FETCH method is used by a client to request the server to produce a representation as described by the query parameters (including query options and payload) based on the resource specified by the effective URI. As a result, the payload returned in response to a FETCH request can not be assumed to be a complete representation of the resource identified by the effective request URI.

Using the FETCH method for the sake of resource discovery in IoT can remedy GET-related problems identified in the previous section. It is also extremely useful when efficient result filtering that preserves network resources is desired. For instance, if a client is only interested in the types of resources available at a server, it formulates a FETCH request asking of that part of the representation as can be seen in Figure 2. The server, then, replies only with the required information, which saves throughput and energy.

Furthermore, if several resources of similar types are provided by different objects and the client knows beforehand the existence of certain resources not meeting its requirements, it can indicate them in the request payload to avoid undesired replies. Thus, a client can exclude non-needed resources, nodes, and parameters by specifying them in the body of its request. In the same way, a client can specify the desired parameters, nodes, content-formats, etc. to be included. Moreover, a client/device can combine all their requirements and knowledge in a single request to further filter returned responses.

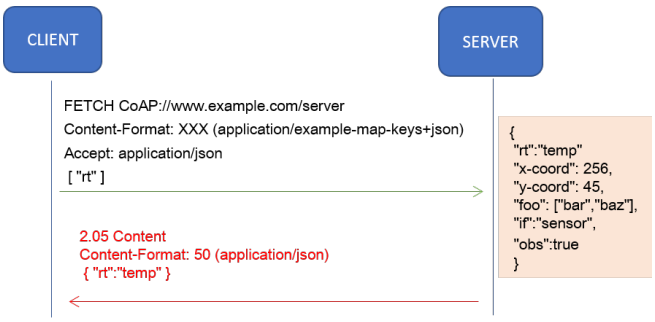

Fig. 2. A FETCH request in JSON

From the above, it is clear that the new FETCH method opens up promising prospects for successfully filtering the returned results, which can significantly reduce network traffic, congestion, collision problems and power consumption when performing resource discovery in constrained networks. In view of these advantages, the question, which will be ad-

dressed in the following section, is how to design an effective use of FETCH for resource discovery in the IoT.

C. FETCH-based Resource discovery for the IoT

Having discussed the limits of GET-based resource discovery and presented the opportunities provided by FETCH, this section introduces the proposed design of an effective scheme of FETCH-based resource discovery.

To do so, we took inspiration from the IGMPv3 protocol for the use of resource filtering based on the INCLUDE and EXCLUDE modes; EXINC for short. These modes are considered very adaptable to the context of resource discovery in the IoT. Indeed, it is useful to have a mechanism that allows to EXCLUDE the resources we do not need. At the same time, providing a technique allowing to INCLUDE the desired features is crucial for an fine-grained resource discovery. Indeed, with these two modes, several combinations of user/device discovery requirements can be formulated in the same request in a very compact format. Building on this, we have developed a scheme allowing to provide rich combinations of desired parameters to be included along with resources, nodes, and parameters to be excluded when replying. Such a scheme is presented in Figure 3, where a EBNF representation is given for the case of CoRE link format. It should be noted, however, that our scheme is not tight to CLF and can be adopted to any other description format including JSON, EXI, CBOR to allow resource discovery in formats other than CLF.

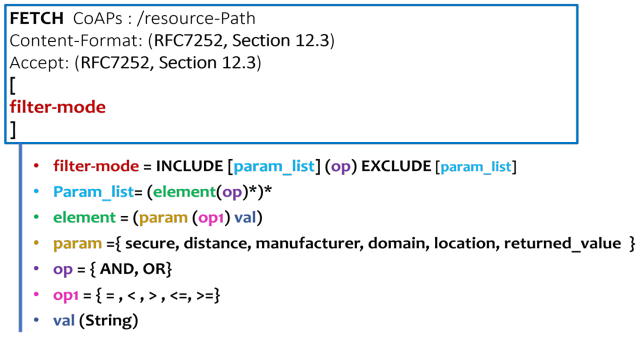

Fig. 3. Semantics of the FETCH-based resource discovery

In addition to the EXINC filtering technique, we propose to format the payload of the FETCH request in such a way as to take logical operators into account when specifying parameter lists to INCLUDE or EXCLUDE. For instance, a client may specify the parameters to be included AND/OR those to be excluded. It can also formulate the request to only get a specific part of the description in a specific media type. In this context, implementations can formulate a request payload of any media type that is compatible with the semantics of FETCH-based resource discovery, detailed in Figure 3. Finally, Figure 4 presents an example of a FETCH discovery request formulated in accordance of the proposed scheme.

To further highlight the importance of the proposed FETCH-based resource discovery, we employ it in a home automation scenario illustrated in Figure 5. In this scenario, the smart air-conditioner must get the average temperature of the house from the thermostats located in different rooms. However, this air conditioner only trusts in Nest thermostats that are secured with Thread and are located less than five hops away. The realization of such a scenario is not possible with GET-based discovery as defined in [9]. On the other hand, with FETCH, we can filter the results by using the proposed EXINC scheme. The FETCH request for this scenario along with the resolution process and returned results are shown in Figure 6.

Finally, the proposed FETCH-based discovery opens up promising prospects for significantly reducing network traffic, congestion, collisions, and power consumption during resource discovery. Note that the proposed technique is expandable to take into account more parameters and more combinations between these parameters in the same query. It is also designed to be adaptable for future uses. In addition, it is as valid with direct (distributed) discovery as with the centralized approaches and any other CoAP-based discovery approach. Indeed, it only requires changing the formulation of the queries by using FETCH with the semantics given above.

IV. Performance Evaluation

This section details the performance evaluation of the proposed method when compared with the GET-based approach adopted by CoAP. It starts by detailing the methodology and used tools, before presenting the evaluation scenarios and measured metrics along with discussing the obtained results.

A. Evaluation tools and Implementations

All our evaluations were carried out in the recently released Thread platform targeting IoT applications in home automation and similar environments. The choice of Thread is motivated by the fact that is a proven secure and reliable solution. It is, also, already implemented in many commercial products on sale for several years now. Moreover, this secure and open network protocol is built on a collection of existing standards similarly to environments such as Contiki [15]. Finally, a user can employ a smartphone, an application, and/or any device to communicate, directly or via the Cloud, with the Thread network.

Based on the open-source implementation of Thread, dubbed OpenThread [16], and inspired by Contiki, we have added a resource discovery layer over CoAP as specified in Figure 7. This layer contains three main components; namely: the request engine implementing the semantics of both GET and FETCH look-ups; the publication engine that is responsible on resource registration; and the resource description component. Subsequently, we developed prototypes of nodes that implement client and/or CoAP server along with the RD.

B. Evaluation protocol and scenarios

To evaluate our solution, we created different home automation and similar network configurations. Each configuration

```
FETCH CoAPS: ./well-known/
Content-Format: application/link-format
Accept: application/link-format
[
 [INCLUDE]: {secure=[Sval_1],Distance[=;<;>;<=;>=]Dval_1,location=[Nodes_List;],Domain=[domain],Manufacturer=[name]}
 [EXCLUDE]:
{secure=[Sval_2],Distance[=;<;>;<=;>=]Dval_2,location=[Nodes_List;],Domain=[domain],Manufacturer=[name]}
]
```

Fig. 4. Example of the proposed scheme for FETCH

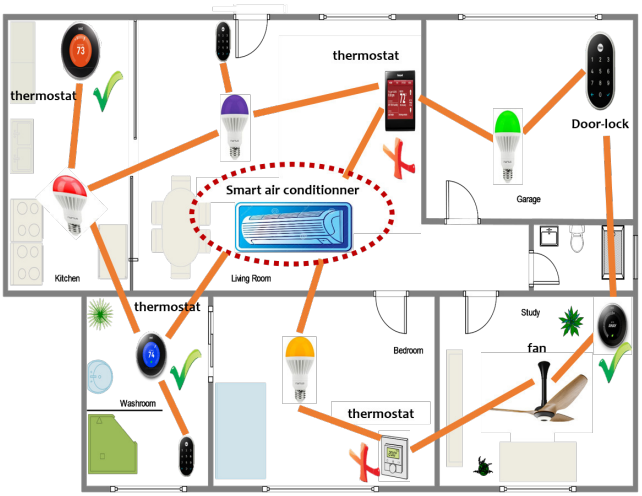

Fig. 5. A use-case of FETCH-based resource discovery

contains 13 nodes over which we run several single-hope and multi-hop scenarios for both centralized and distributed cases. Below are the details of multi-hop scenarios.

- Centralized discovery (CoAP-RD) in multi-hop networks: in this test, we designed a scenario where three servers will register their resources in the RD, then two concurrent clients consult the RD to obtain the descriptions of registered resources. To do so, clients send their requests in unicast, using the Thread unicast routing protocol, to the RD. This latter will respond with unicast messages containing the requested parts.
- Distributed discovery in multi-hop networks: in this test, one client sends a multicast request to discover required resources. This request will be propagated in the network by the multicast routing protocol MPL [17]. Once this request arrives to a node having the desired resources, it will respond directly to the client with a unicast response.

The above test cases were performed by simulations using virtual instances of Thread objects. Hence, environment-related parameters such as signal propagation, influence of obstacles and interference with other wireless signals are not taken into account. Also, the estimation of parameters, such as the consumed energy will not be possible. For each test case, we set the simulation time to 600 seconds and varied the request frequency. To put the results in context, we compared our FETCH-based discovery approach with the standard GET-based approach under the following performance metrics.

- Average discovery time: this parameter is defined as the average waiting time between the transmission of a request and the reception of the first response averaged over several requests. This metric is used to evaluate the efficiency of our solution in terms of latency.
- Discovery success rate: measures the number of received responses to all sent queries. This metric is used to evaluate the reliability of the proposed technique.
- Size of request/reply messages: accumulate the size of request/reply messages. It is defined as the ratio between the sum of request/reply sizes for each node over the total number of nodes.

By comparing our approach with that of GET under different networks and discovery scenarios, we aim to encompass most of the performance indicators of the proposed approach. Simulation parameters are summarized in Table I and obtained results are discussed in the following subsection.

TABLE I
SIMULATION PARAMETERS

Parameter	Value
Duration of one simulation	600 seconds
Number of iterations	5
Number of nodes	13
type of nodes	Thread CLi Instances
Message payload	Variable size
Network layer	IPv6 over 6LoWPAN
MAC layer	802.15.4 with enabled MAC security

C. Results and discussions

This section discusses the obtained results in both centralized and distributed scenarios.

1) Centralized approach: In this section, we will discuss the results obtained for the centralized approach of our FETCH-based solution (CoAP-RD-FETCH) and that of GET (CoAP-RD-GET). Obtained results are depicted in Figure 8.

Figure 8.(a) presents the average time of discovery of both approaches when varying the request frequency. As can be seen from this figure, discovery with FETCH achieved a lower discovery time compared to that of the GET method. This can be explained by the fact that GET generates more traffic since the response message contains descriptions of all available resources in the RD. It takes a longer time to reach the client because of its size and the congestion created in the network. However, the FETCH method performs efficient filtering so

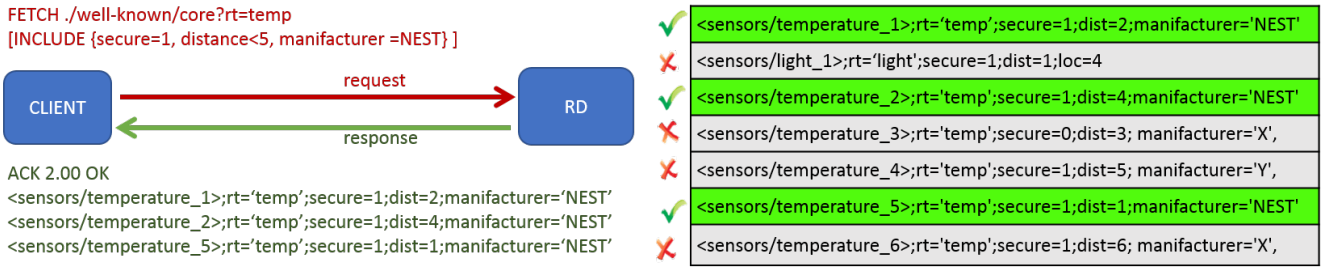

Fig. 6. Details of a FETCH-based resource discovery process

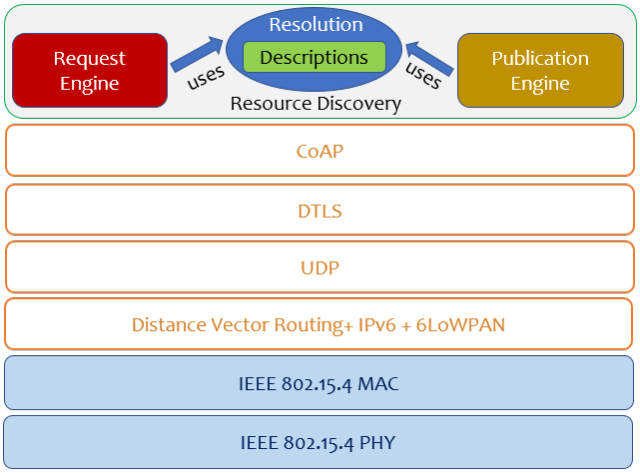

Fig. 7. Resource discovery implementation in OpenThread

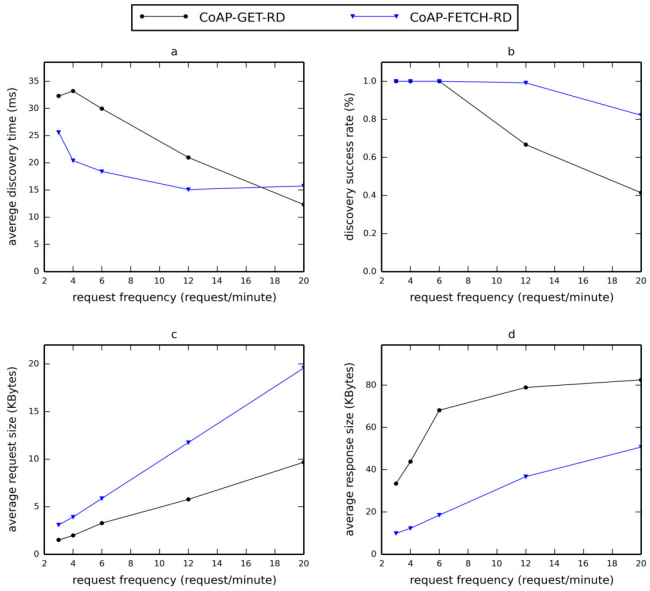

Fig. 8. Evaluation of proposed mechanisms in a unicast discovery scenario

that the RD only returns the desired resource descriptions, which significantly reduces the size of the response messages and also allows for faster routing (less congestion).

Concerning the average discovery success rate, Figure 8.(b) shows that both GET and FETCH registered a high success rate approaching 100 % in low request frequencies. This is due to the reliable routing protocol deployed in Thread along with the fact that the simulation environment is considered perfect. With a high query frequency, however, the discovery success rate becomes very low, approaching 40% for the GET method. This could be explained by the very high congestion of the network caused by the high frequency of query generation and the size of GET responses that may even lead to the elimination of responses at the transmission buffer. However, with the FETCH method, we notice that the rate does not decrease too much and approaches 80% at the highest frequency thanks to the minimized size of returned responses.

With regards to the size of requests, it is clear from Figure 8.(c) that the queries generated by FETCH are slightly larger compared to those generated by GET. This is due to the extra data that FETCH needs in its payload to specify the filter that will be used during the discovery process. This has the advantage of minimizing response size by only returning the desired resources. Knowing that a single query can match several large responses, this surplus provides an acceptable compromise. Indeed, it improves the relevance of responses along with the discovery time and network utilization as can be seen from Figure 8.(d). For instance, this figure shows that the difference in size, between the responses generated by GET and those generated by FETCH, is very important. This is due to the fact that, with FETCH, the RD only returned description parts that exactly matched user's specifications. This ensures both user satisfaction concerning discovery relevance and time as well as network fluidity with regards to congestion and resource utilization.

2) Distributed approach: This subsection discusses the results obtained on the distributed resource discovery scenario with the two discovery approaches: GET (CoAP-GET-multicast) and FETCH (CoAP-FETCH-multicast). Obtained results are depicted in Figure 9.

As can be seen from Figure 9.(a) FETCH-based resource discovery achieved noticeably lower discovery time compared to that achieved by GET-based discovery. This difference reaching up to 130 ms, shows the power of FETCH especially for multicast traffic, which is slow and expensive in terms of time and energy. With regards to the average discovery success

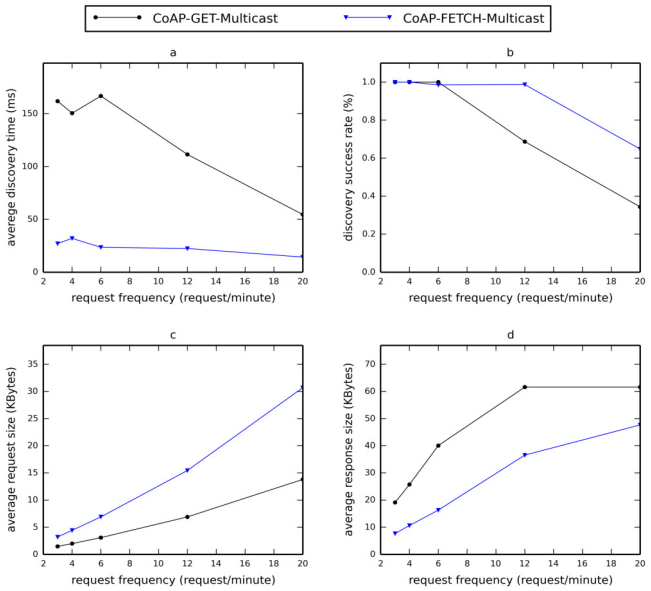

Fig. 9. Evaluation of the proposed mechanisms in a multicast discovery scenario

rate, Figure 9.(b) shows that both methods are equally reliable in both centralized and distributed approaches through the use of the MPL, which ensures the efficient routing of messages.

Finally, and concerning the size of generated messages, it is clear from Figure 9.(c) that the size of the multicast requests sent by FETCH is slightly larger than those of the GET. These results are similar to those of the centralized approach because it is the same client that sends the same requests. On the other hand, the size of returned responses is noticeably smaller in FETCH-based discovery in comparison with GET as can be seen from Figure 9.(d). Similarly to the centralized case, this is due to the fact that with FETCH the servers only returned the descriptions of the adequate resources.

By analyzing these results, we can confirm that the use of FETCH method for the sake of resource discovery is very important and outperforms GET in many aspects regarding the granularity, efficiency, and relevance of discovery along with resource utilization. Such performance is equally efficient and effective for both approaches (centralized and distributed). Therefore, an in-depth elaboration of the FETCH-based specification will open up more advanced and more efficient prospects for resource discovery in the IoT.

V. Conclusion

In this paper, a new CoAP-based discovery mechanism was proposed building on the newly standardized FETCH method. Obtained results demonstrated the capacities of FETCH to achieve fine-grained, expressive and efficient discovery when compared with the GET-based discovery adopted by CoAP. These achievements open up new horizons to formulate a compact and expressive resource discovery framework for the IoT. Our future work ports on the generalization of FETCH-based resource discovery to encompass a wide-range of IoT look-ups in view of proposing a specification of IoT resource discovery based on FETCH.

References

[1] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of ipv6 packets over ieee 802.15.4 networks," RFC 4944, RFC Editor, September 2007. http://www.rfc-editor.org/rfc/rfc4944.txt.

[2] P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander, "Rpl: Ipv6 routing protocol for low power and lossy networks," *RFC 6550*, 2012.

[3] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (coap)," RFC 7252, RFC Editor, June 2014. http://www.rfc-editor.org/rfc/rfc7252.txt.

[4] R. T. Fielding, *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

[5] P. Stok, C. Bormann, and A. Sehgal, "Patch and fetch methods for constrained application protocol (coap)," Internet-Draft draft-ietf-core-etch-04, IETF Secretariat, November 2016. http://www.ietf.org/internet-drafts/draft-ietf-core-etch-04.txt.

[6] S. Cheshire and M. Krochmal, "Dns-based service discovery," RFC 6763, RFC Editor, February 2013. http://www.rfc-editor.org/rfc/rfc6763.txt.

[7] Z. Shelby, M. Koster, C. Bormann, and P. V. der Stok, "Core resource directory," Internet-Draft draft-ietf-core-resource-directory-10, IETF Secretariat, March 2017. http://www.ietf.org/internet-drafts/draft-ietf-core-resource-directory-10.txt.

[8] M. Liu, T. Leppanen, E. Harjula, Z. Ou, A. Ramalingam, M. Ylianttila, and T. Ojala, "Distributed resource directory architecture in machine-to-machine communications," in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2013 IEEE 9th International Conference on*, pp. 319–324, IEEE, 2013.

[9] Z. Shelby, "Constrained restful environments (core) link format," RFC 6690, RFC Editor, August 2012. http://www.rfc-editor.org/rfc/rfc6690.txt.

[10] T. A. Butt, I. Phillips, L. Guan, and G. Oikonomou, "TRENDY: An adaptive and context-aware service discovery protocol for 6lowpans," in *Proceedings of the third international workshop on the web of things*, p. 2, ACM, 2012.

[11] J. Maenpaa, J. J. Bolonio, and S. Loreto, "Using RELOAD and CoAP for wide area sensor and actuator networking," *EURASIP Journal on Wireless Communications and Networking*, vol. 2012, no. 1, p. 121, 2012.

[12] C. Jennings, B. Lowekamp, E. Rescorla, S. Baset, and H. Schulzrinne, "Resource location and discovery (reload) base protocol," RFC 6940, RFC Editor, January 2014.

[13] B. Djamaa, A. Yachir, and M. Richardson, "Hybrid CoAP-based resource discovery for the Internet of Things," *Journal of Ambient Intelligence and Humanized Computing*, Feb. 2017.

[14] P. van der Stok, C. Bormann, and A. Sehgal, "Patch and fetch methods for the constrained application protocol (coap)," RFC 8132, RFC Editor, April 2017.

[15] "The official git repository for contiki." [Online] Available: https://github.com/contiki-os/contiki.

[16] "Openthread github website." [Online] Available: https://github.com/openthread/openthread.

[17] J. Hui and R. Kelsey, "Multicast protocol for low-power and lossy networks (mpl)," RFC 7731, RFC Editor, February 2016.