Automated Discovery of CoAP-enabled IoT devices

Francesco Caturano[1], Jaime Jiménez[2], Simon Pietro Romano[1]

***Abstract*— This paper presents the design and implementation of a framework that allows for the automated discovery of IoT devices. We discuss how to properly combine a set of discovery mechanisms used in the framework of the Constrained Application Protocol (CoAP) in order to estabilish relations between CoAP-enabled devices. Such relations represent links that are 'tracked' by an ad-hoc implemented CoAP Crawler, which inspects the served resource state and chooses the link to follow from there, working as a Hypermedia-driven client. The Crawler is itself a Web-enabled device that dynamically draws an overlay network topology in which resources are represented as nodes and inter-device relationships as edges between pairs of nodes. This approach allows for the creation of a completely distributed, non-hierarchical, Peer-to-Peer overlay network. A Proof-Of-Concept implementation of the framework is described and preliminary performance evaluation results, in terms of response time and throughput with increasing network size conditions, are discussed.**

I. Introduction

Nowadays, Internet provides access to an huge amount of data, information and software. It hence becomes necessary to have mechanisms for finding and locating the resources users or systems are looking for. Vanthournout et al. [1] define Resource Discovery as "*the ability to locate resources that comply to a set of requirements given in a query*". The most popular and effective mechanism of Discovery over the Web is the so-called *crawler*, a software that automatically crawls Web pages to create a local index of resources of the most wide-spread publicly available Websites [2]. Though the basic concept of the discovery of resources on the Web can be extended to the Internet of Things, it can not be addressed in the exact same way, due to remarkable differences between the two environments. For instance, while a Web crawler can assume servers are highly available, IoT constrained devices suffer from extended periods of inactivity.

As part of the work in realizing the REST architecture for constrained IoT devices and networks, the Constrained RESTful Environments Working Group of the IETF has defined a serialization of a typed link as specified in the "Web Linking" standard [3], in order to address the discovery of resources hosted by constrained servers. The purpose of the CoRE Resource Discovery is to provide URIs for the resources hosted by servers, as long as other metadata such as attributes and possible link relations. End-points perform the discovery by issuing requests to the well-known URI "/.well-known/core", which is used as entry point for requesting the list of resources hosted by the server. Responses to such requests contain a payload formatted in the CoRE Link Format [16]. In many M2M (machine-to-machine) application scenarios, direct discovery can be impracticable due to sleeping nodes, lossy networks and poor support to multicast interactions. In these cases, it can be useful to introduce an entity called Resource Directory (RD) [4], which stores descriptions of resources hosted by other servers and provides a set of REST interfaces to perform discovery, resources registration and lookups on those resources. If a lookup interface is provided, clients use the RD to discover resources registered, performing queries for endpoints, resources, groups and domains. The result of a lookup query is the list of links to resources that match the type of lookup. In a distributed Web system, the resources hosted by servers are shared among applications, usually running in web browsers. The representations of those resources form the state of the application. Along with representations, clients and servers exchange descriptive metadata, called Hypermedia, about how to obtain state information. The REST design pattern claims that Hypermedia can drive the application state, allowing applications to read metadata and automatically consume resources. In a RESTful interaction model, hyperlinks describe how and where to retrieve state information [5]. Therefore, if resources are described through links, an automated discovery process provides sufficient information about actions to perform and functionalities to explore, that allow the state of the application to evolve autonomously [6].

The rest of this paper is organized as follows. Sec. II cathegorizes related works. Sec. III presents the design of a novel framework for the automated discovery of CoAP-enabled IoT devices. The implementation of the framework is briefly highlighted in Sec. IV. In the same section we also delve into the details of the "crawling" functionality for the visualization on a Web interface of the discovered resources, as well as of the inferred inter-relations. Results of experimental campaigns are presented and critically analyzed in Sec. V. Finally, Sec. VI concludes the paper by summarizing our main achievements, as well as identifying directions of future work.

II. Related Work

The attempt of creating a distributed architecture that supports discovery of IoT devices has been explored in the past. This section describes some approaches related to our work, pointing out both their advantages and limitations. Jouni et.

[1]University of Napoli Federico II, Department of Electrical Engineering and Information Technology, Via Claudio 21, 80125 Napoli, Italy. fr.caturano@studenti.unina.it, spromano@unina.it

[2]Ericsson Research, Hirsalantie 11, 02420 Jorvas Finland. jaime.jimenez@ericsson.com

al [7] propose a CoAP usage for RELOAD, a P2P signaling protocol allowing CoAP nodes to store resources in a peer-to-peer overlay. The results of this work show how a P2P architecture becomes recommendable when the frequency of the CoAP messaging and the size of the system increases, bringing many advantages in terms of self-organization and scalability. The paper [8] proposes a distributed resource directory architecture for M2M applications. A DHT-based P2P overlay network supports registration, discovery and lookup of devices. Preliminary experimental results are also discussed by the authors, showing how leveraging RESTful operations upon a P2P overlay network enables interoperability among heterogeneous devices. DHT is also the target architectural choice for [9], that, in combination with a novel IoT Gateway, aims to address both local and global Service Discovery. Services discovered locally are then published into a P2P overlay network, which supports complex query structures. However, studies like [10] have shown that structured distributed algorithms such as the ones adopted in DHT-based overlay networks, are not sustainable for constrained environments. Moreover, Evdokimov et al. [11], show how a DHT-based approach provides a high-level of scalability and reliability, as it removes central points of failure. However, it fails at enabling *trackability*, as many devices are not aware of each other when the network size increases. Work in [12] is a first attempt at applying Web oriented patterns (REST) to service discovery, so to enable integration of heterogeneous physical devices into digital information systems.
Finally, authors of the paper [13] provide a thorough categorization of the current technology landscape for resource discovery in IoT and also proposes a discovery framework based on REST APIs. The API manages registration and indexing of resources, whose metadata are represented using CoRE Link Format. However, the distributed approach is dropped, in favor of a more centralized one.
In the next sections, we describe the design and implementation of the proposed framework, showing how the combination of multiple CoAP discovery mechanisms allows for the usage of a decentralized architecture without the need of DHT algorithms or gateway intermediaries. Furthermore, we rely on a RESTful service discovery, which enables interoperability between different devices.

III. Design

The IoT devices discussed in this work are CoAP endpoints. Each of these devices implements one or more discovery mechanisms and builds logical connections with other nodes. Since this process is periodically triggered, new devices that are deployed at run-time can discover nodes already present on the network, build relations with them and vice-versa. Thanks to this ongoing process, the Crawler can try the different discovery mechanisms supported, in order to reach the highest number of devices on the network, prepare the data and show their resources on the Web Interface. An user can watch the devices join the network dynamically, inspect their resources, select the ones he/she is interested in and eventually interact with them.
These devices can be categorized based on the number of different discovery mechanisms that they are able to perform, as shown in the following table.

TABLE I: Discovery Categories

Discoverability			
	RD	Multicast Peer	Simple Peer
Unicast Inerface	✓	✓	✓
Multicast Interface	✓	✓	✗
Discovery Mechanism			
Multicast CoAP	✗	✓	Optional
CoRE RD	✗	✓	Optional
Research Across Peers	✗	✓	Optional
Pre-built relations	✓	✓	✓
Mutual Discovery	✗	✓	✓

The first part of Table I describes the physical means that a device can use to reach other things. A request to a Unicast Interface is issued on two occasions:

- A thing finds a Multicast CoAP server. The Multicast server starts a mutual discovery phase, contacting the device directly;
- Some devices may already have a set of pre-configured relations, for example a remote which controls a few light bulbs. This means that a light bulb already knows how to contact the remote control and vice-versa (holding the IP Address). At any time (at bootstrap, for instance) one of them can contact the other and retrieve its resources.

Second part of Table I lists the discovery mechanisms supported by each device. Multicast and Simple peers are equipped with all the different techniques. This means that they periodically flood the network to see if there is a device responding to Multicast requests. While doing this, if a Resource Directory is found, the devices register their own resources. A response to these requests carries the resources of the devices discovered, as well as the resources that such devices have in turn discovered themselves. Such a recursion allows devices to discover things that are not directly reachable with classic requests. "Searching across peers" is the fundamental requirement that allows devices (first of all, the crawler) to *follow* links, in order to reach the majority of devices on the network. Some devices may already have some pre-configured connections. At the end of the discovery process they store each others' resources.
Mutual Discovery is another important requirement that allows devices to establish connections and relations (links). By mutually discovering each other, two devices, regardless of who found whom first, simply exchange their resources.

The fact that many of the Simple Peer discovery mechanisms are marked as "Optional" in the standard, allows to configure them in different fashions and capture a lot of variation. There can be Simple Peers that discover both RDs and Multicast Peers, some that only discover Multicast Peers, or even some that do not perform any discovery.

The reason why the Crawler has not been included in the discussion so far is that it can be interpreted as a "discovery mechanisms aggregator": it tries every discovery technique, shows the devices discovered, their resources and, most importantly, the connections that things have built over time. Therefore, by the term "Crawler", we mean a functionality that, potentially, can be held by any device.

The same abstraction can be made elsewhere as well. In fact, a "device" is just a particular configuration of different discovery functions. The reason that led to the specific configuration described above is to catch the most variation possible and define the behavioral boundaries for the system. In the final architecture we will indeed consider also devices that do not support any discovery mechanism. The challenge is to see whether the Crawler is able to discover those as well.

By exchanging resources described with links, aggregate them in multiple points of the network, as well as builing logical links among devices, the resulting architecture is an unstructured P2P overlay network. Although the discussed devices assume different roles, the choice is to treat every device as equal, in order to focus on the exposed functionality. After all, having a hierarchical structure requires strong dependability assumptions, which are not likely to be met by constrained devices.

In the next sections, mechanisms allowing for the creation of an unstructured P2P overlay network are described in detail.

A. Mutual Discovery

Mutual Discovery is the simplest discovery mechanism: it allows devices to discover each other and exchange resources. The Sequence Diagram in Fig. 1 shows two Simple Peers attempting to discover each other. The second one does not perform any discovery other than the mutual, so this case falls under the category of "pre-built relations": it means that the connection between the two devices is not built dynamically, but is already statically configured.

Since the process is triggered periodically, devices filter resources that may have already been stored previously.

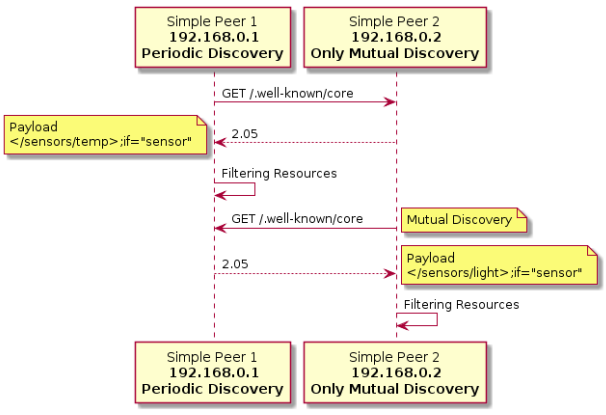

Fig. 1: Mutual Discovery Sequence Diagram

B. "hostedby" Relation Type

Once devices have exchanged their resources, they have somehow to relate them to the device that actually owns them. CoRE Link Format provides the attribute "rel", which describes the relation between resources. Hence, when the resource needs to be stored, a new link is created, containing the IP Address and the port of the device discovered. Then, an "anchor" is created, binding the link to the resource. At last, the relation attribute "hostedby" means that the resource indicated by the "anchor" attribute, is hosted by the device whose IP Address and port are reported in the URI section of the link just created. Storing resources and drawing logical links between them, allows to create a sort of distributed Resource Directory, which holds descriptions of resources and links for the entire network.

The following listings report, respectively, the structure of the above described web links and their representation in JSON format.

```
</sensors>;ct=40;title="Sensor Index",
</sensors/temp>;rt="temperature-c";if="sensor",
</sensors/light>;rt="light-lux";if="sensor",
<coap://172.0.24.6:5683>;
anchor="/sensors/temp";rel="hostedby";
```

Listing 1: Web Links before parsing

```
[{"href": "/sensors",
  "ct": "40",
  "title": "Sensor Index"},
{"href": "/sensors/temp",
 "rt":"temperature-c",
 "if":"sensor"},
{"href": "/sensors/light",
 "rt": "light-lux",
 "if": "sensor"},
{"href": "coap://172.0.24.6:5683",
 "anchor": "/sensors/temp",
 "rel": "hostedby"}]
```

Listing 2: JSON Object after parsing

C. Multicast Discovery

In Fig. 1, Simple Peer 1 has also the capability of periodically flooding the network to find CoAP servers that make themselves discoverable over a Multicast Interface. When one such server is found, the usual discovery process is performed. Multicast Peers are also able to mutually discover devices that find them. At the end, resources are exchanged and stored.

D. RD Discovery and Registration

Discovery of an RD follows the same rules mentioned before. A query string with the value "rt=core.rd*" is needed in the request URL. As a response, the Resource Directory lists the functionality supported. In this case, there exist a Registration Interface and a Lookup Interface for resources and endpoints. After discovering the supported functionality, the Multicast Peer sends a POST message to register its resources on the RD. In this way, a different device can issue all sorts of queries to the Lookup Interface and retrieve information of interest.

E. RD Lookup Query

Resource Directory Lookup Interface becomes useful when devices are not directly reachable, either for high packet-loss rates or temporary inactiveness. In the following scenario, the RD is the seed URL for the Crawler, which starts collecting links, asking for endpoints which have registered their resources. It does so by performing a simple query to the link "/rd-lookup/ep/". The response contains a link to the Multicast Peer.

F. Crawler Aggregation

The next scenario pictures the Crawler aggregating different discovery mechanisms. After the discovery phases described above, peers on the network have self-organized and have built connections, that is to say "hyperlinks". The crawler collects these links, visits the devices pointed by URLs and retrieves their resources. Moreover, during this process, it keeps track of the links between pairs of devices and draws the topology of the overlay network. Every device is identified by its IP Address and its resources, which are also shown on the Web Interface.

IV. Implementation

After describing the design requirements and a first sketch of the system structure, this section presents the choices made to realize the details of the architecture. Among the available CoAP implementations[1], *node-coap*[2] takes advantage of the growing attention towards new frameworks for programming network modules such as Node.js.*node-coap* is a client and server library for CoAP, based on the model of the *http* module. It follows the draft-18 of CoAP [14][3] and uses a generator/parser of CoAP packets for Node.js, called CoAP-packet[4]. Devices considered in this work host a collection of resources. Discovery of devices has both the purpose of exchanging those resources and establishing relations, so that potentially any other device can follow them. The analogy with navigating web pages by clicking on hyper-links adds up to resources represented as links. Using Javascript Object Notation in combination with Node.js allows natural usage of built-in methods to handle data representation.

A. Detailed Architecture

The attempt to make the discovery become automatic calls for the application to be based on the REST design principles. In particular, the so-called HATEOAS (Hypermedia As The Engine Of Application State) REST component allows to create an underlying continuous process of functionality discovery, that goes through the following states: an (i) issued Request (ii) identifies a Resource whose description (iii) is supplied within a Response. The application logic (iv) elaborates the content of the response and decides the next request to perform, based on the metadata that describe the payload of the response, allowing the process to start again from state (i). Every device in the reference implementation, follows this simple design principle.

Since every CoAP endpoint is a peer, from a high-level design point of view, it makes sense that every device has methods for both issuing requests and producing responses. However, to enlighten also the static part of the architecture, it is advisable to separate the client-side responsibilities from those of the server-side (as depicted in Fig. 2). The basic difference stands in the property, for the server, of storing descriptions of resources which carry Hypermedia Controls. Such descriptions provide the semantics that teaches to clients the relations between links so that they can decide, autonomously, what to do next. In general, Hypermedia Controls should be exposed by the server with a simple interface allowing clients to discover through links but also interact with devices, for example by submitting forms. This interface is, by nature, provided by a set of CRUD requests/responses. Since the focus of the work is mainly about discovery, rather then interaction, resources are assumed to be already created, updated or deleted. The client needs to be able to interpret the Hypermedia Controls. That means the ability of deciding where to direct the next request, thanks to the logic of the application.

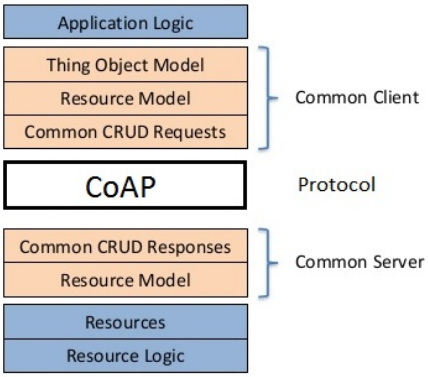

Fig. 2: Hypermedia Systems modular architecture

B. Simple Peer Implementation

According to the modular architecture just presented, the Simple Peer, in its basic configuration:

- receives requests over the /.well-known/core interface from devices which have pre-configured connections;
- answers by providing the list of resources hosted;
- extracts the IP address from the request just received and starts a Mutual Discovery process;
- receives a list of resources hosted by another device, processes them and creates new links via the "hostedby" relation type.

C. Multicast Peer Implementation

The Multicast Peer is intended to be another davice that accepts requests over a Multicast Interface. As the CoAP RFC suggests, the steps to discover a device over Multicast

[1]http://coap.technology/impls.html
[2]https://github.com/mcollina/node-coap
[3]Which has eventually become RFC7252
[4]https://github.com/mcollina/coap-packet

are exactly the same discussed in the previous paragraph. The only different aspect, obviously, is the address of the device, which is a Multicast Address suggested by the standard *Group Communication for the Constrained Application Protocol* [15]. In the reference implementation, the Multicast Peer is one of the most powerful, discovery-wise: it is the most easily reachable and supports all the widely discussed discovery mechanisms (Mutual Discovery, Multicast Discovery both for Peers and Resource Directories). To meet the requirements of the Constrained Devices, a big effort has been made to keep the implementation as light as possible. Two precautions have been taken. First, the device stores representations of resources discovered directly, for example via periodic discovery of other Multicast Peers, Mutual Discovery, or by following pre-configured relations. The rest of resources that it eventually retrieves (i.e., the "hostedby" ones) are not directly saved. Instead, a Map of IP addresses is created, which describes the devices discoverable through others. Eventually, the Multicast Peer, as the architecture starts to scale, will hold representations of many resources. These resources will be sent to any device that issues requests to the Multicast interface. To avoid overloading the network, only a maximum of 10 resources per discovery phase are actually sent. Since the discovery is done periodically, eventually the other peers will be able to discover every other device.

D. Resource Directory Implementation

Resource Directory is still a functionality that can be held by a single device, even if in most cases it operates on a separate module. It can be naturally structured as a Database, so the choice for the implementation has nonetheless fallen on MongoDB, an open-source document-oriented NoSQL database which stores data in the form of JSON-like objects. Resource Directory discovery follows the same rules discussed so far about Unicast/Multicast Interface and the well-known URI. To retrieve information about the available functionality for the lookup interface, one may fill the query string with the value "rt=core.rd*": this will result in a response indicating the types of lookup supported. In the reference implementation, queries for endpoints and resources are supported.

E. Crawler Implementation

The Crawler is a device which provides web integration, hosting on one side a functionality that enables the aggregation of multiple discovery mechanisms and on the other side the ability to show results in a human readable interface. The first functionality is achieved thanks to a crawling algorithm, whose ability is to try different discovery mechanisms until a point from which to start following links amongst devices is found. The other fundamental Crawler requirement is that it should be able to start its discovery activity from any point of the network, regardless of the kind of device discovered first. This requirement is fulfilled only by ensuring that the discovery is periodic and used to build relations between pairs of devices, that is what has been discussed so far. The Crawling algorithm makes use of a queue filled with URIs of resources to visit. Each visit consists of a request to the CoAP well-known address. As mentioned before, devices respond to these requests by providing both resources of their own and resources discovered, so the Crawler for each resource has to decide which results to publish and which to "follow". Hence, while resources hosted by devices are shown on the Web Interface, the ones discovered form the links pushed into the queue to continue the discovery activity. Resources are filtered based on the "hostedby" rel attribute.

F. Publishing Results

The web interface has three basic requirements: (i) identify each device by its IP Address, so that the user can "click on" and visualize its resources; (ii) visually represent both the position of each device and the logical links between pairs of devices on an overlay network topology; (iii) dynamically publish on the web page both resources and nodes representing devices joining the network over time.
Showing resources, adding nodes and drawing logical links to the topology while devices join the network dynamically, mandate the ability to handle real-time data communication. Socket.io uses the Websocket protocol to enable real-time bidirectional communication in the browser.

V. Experimental Results

The evaluation aims to deduce the performance of the system and the condition under which the related performance figures appear. The overlay network is tested from two points of view: (i) *Workload Characterization*, to extract significant application-level performance parameters that describe how the system works with a specific set of requests; (ii) *Capacity Test* on the bottleneck of the overlay network.

A. Workload Characterization

The purpose of the workload characterization is to draw up a profile of the system, under ordinary circumstances of work. High-Level parameters such as latency, byte count, and throughput are taken. The analysis conducted aims to find the amount of time that it takes for the Multicast Peer to discover 150 Simple Peers, throughout a time interval of 5 minutes.core link Requests are performed every 2 seconds. To ensure that each device performs the same amount of requests over time, average, the distribution of requests is normal. Figure 3 shows, on the left, that the amount of data exchanged suddenly drops and remains stable around that value, meaning that no more chunks of data need to be sent by the multicast peer, because every device was discovered. Per each container, the instant where this behavior manifests is extracted and a histogram of discovery times is drawn (Figure 3, right).
The term "mean" signifies that, if discovery is performed, on average, every 2 seconds, the Multicast Peer takes about 74 seconds to discover 150 devices.

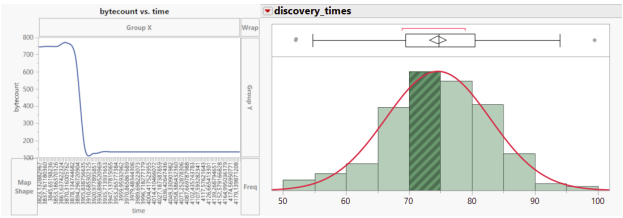

Fig. 3: Bytes exchanged (left) - Mean Discovery times (right)

B. Capacity Test

The purpose of the capacity test is to evaluate the system under increasing conditions of workload. A scenario with only one Multicast Peer - the real bottleneck of the network in terms of resources consumed - responding to requests coming from 150 Simple Peers is designed. To ensure that each client performs the same number of requests over time, a normal distribution of discovery times is induced. This means that a random number between 0 and the desired time period is chosen every time the request is performed. The boundaries of the time interval are chosen so that the sum of requests among the containers adds up to the desired rate of requests per minute.

The trial scenario is characterized by: (i) 12 experiments, starting from 300 requests/minute, ending with 180000 requests/minute; (ii) 2 minutes of duration for each experiment; (iii) Throughput (number of responses per minute) and Response Time as performance metrics.

Distributions of response times appeared skewed around specific values, so the median is a more significant central tendency index than the mean.

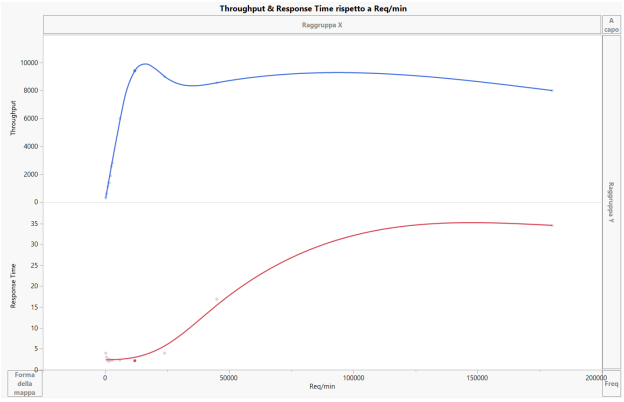

Fig. 4: Capacity Test

Figure 4 shows comparison of Throughput and Response Time curves. Around the same value of request rate, there is a decreasing trend for the Throughput curve and an increasing trend for the Response Time curve. This happens between 12000 req/min and 24000 req/min. In this interval the "Knee Capacity" value can be found, that is the optimal condition under which the system can operate, where Throughput is at its highest and response time at its lowest.

VI. CONCLUSIONS

In this paper we presented a distributed architecture for the automated discovery of CoAP-enabled IoT devices. We showed how the combination of several discovery mechanisms allows designing a completely decentralized architecture, with non need for gateways or complex DHT algorithms. The RESTful interaction model naturally lends itself to this purpose, allowing devices to exchange resources and build relations through simple interfaces. A Crawler takes advantage of the logical links estabilished between devices, performs multiple discovery mechanisms and draws up the topology of the overlay P2P network. At last, preliminary experimental results in terms of Throughput and Response Time are analyzed. A future work is to evaluate the scalability of the system, on occasions where the network size increases significantly.

REFERENCES

[1] Vanthournout, Koen, Geert Deconinck, and Ronnie Belmans. "A taxonomy for resource discovery." Personal and Ubiquitous Computing 9.2 (2005): 81-89.

[2] Cho, Junghoo, and Hector Garcia-Molina. "The evolution of the web and implications for an incremental crawler". Stanford, 1999.

[3] Nottingham, Mark. "Web linking". No. RFC 8288. 2017.

[4] Shelby, Zach, Carsten Bormann, and Srdjan Krco. "CoRE resource directory." (2013).

[5] M. Koster, (October 30, 2015), "Hypermedia Design for Machine Interfaces" [Blog Post], retrieved from http://iot-datamodels.blogspot.com/2015/10/hypermedia-design-for-machine-interfaces.html.

[6] M. Koster, (April 3, 2017), "Interactive Hypermedia and Asynchronous Machine Interaction using Hypermedia Controls" [Blog Post], retrieved from http://iot-datamodels.blogspot.com/2017/04/interactive-hypermedia-and-asynchronous.html?m=0.

[7] M.Jouni,J. Jiménez and S.Loreto, "Using RELOAD and CoAP for wide area sensor and actuator networking", 2014 EURASIP Journal on Wireless Communications and Networking, pp. 121, Springer.

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

[9] S.Cirani, L.Davoli, G.Ferrari, R.Léone, P.Medagliani, M.Picone, L.Veltri, "A scalable and self-configuring architecture for service discovery in the internet of things", IEEE Internet of Things Journal, 2014, pp. 508-521, IEEE

[10] S. Cirani, L. Davoli, M. Picone and L. Veltri, "Performance evaluation of a SIP-based constrained peer-to-peer overlay", 2014 International Conference on High Performance Computing & Simulation (HPCS), Bologna, 2014, pp. 432-435. doi: 10.1109/HPCSim.2014.6903717

[11] S.Evdokimov, B.Fabian, S.Kunz, N.Schoenemann, "Comparison of discovery service architectures for the internet of things", 2010 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing, 2010, pp. 237-244, IEEE

[12] D.Guinard, V.Trifa, S.Karnouskos, P.Spiess, D.Savio, "Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services", IEEE transactions on Services Computing, 2010, pp. 223-235, IEEE

[13] S.K.Datta, R.P.F Da Costa, C.Bonnet, "Resource discovery in Internet of Things: Current trends and future standardization aspects", 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT), 2015, pp. 542-547, IEEE

[14] Z. Shelby and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18, Internet Draft, June 28, 2013.

[15] Rahman, Akbar, and Esko Dijk. "Group communication for the constrained application protocol (CoAP)". No. RFC 7390. 2014.

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