Journal of Ambient Intelligence and Humanized Computing, Vol. 8, Issue 3, June 2017 pp. 357-372
DOI:10.1007/s12652-017-0450-3

Hybrid CoAP-based Resource Discovery for the Internet of Things

Badis Djamaa[a,b], Ali Yachir[a,c] and Mark Richardson[b]

[a]*Artificial Intelligence Laboratory, Military Polytechnic School (EMP), Bordj-El-Bahri, 16111, Algiers, Algeria*
[b]*Centre for Electronic Warfare, Cranfield University, Shrivenham, SN6 8LA, UK*
[c]*LISSI Laboratory, University Paris-Est Créteil (UPEC) , 94400 Vitry-sur-Seine, Paris, France*
b.djamaa@cranfield.ac.uk

Abstract

Enabling automatic, efficient and scalable discovery of the resources provided by constrained low-power sensor and actuator networks is an important element to empower the transformation towards the Internet of Things (IoT). To this end, many centralized and distributed resource discovery approaches have been investigated. Clearly, each approach has its own motivations, advantages and drawbacks. In this article, we present a hybrid centralized/distributed resource discovery solution aiming to get the most out of both approaches. The proposed architecture employs the well-known Constrained Application Protocol (CoAP) and features a number of interesting discovery characteristics including scalability, time and cost efficiency, and adaptability. Using such a solution, network nodes can automatically and rapidly detect the presence of Resource Directories (RDs), via a proactive RD discovery mechanism, and perform discovery tasks through them. Nodes may, alternatively, fall back automatically to efficient fully-distributed discovery operations achieved through Trickle-enabled, CoAP-based technics. The effectiveness of the proposed architecture has been demonstrated by formal analysis and experimental evaluations on dedicated IoT platforms.

Keywords: Service Discovery, Resource Directory, CoAP, Internet of Things, Trickle Algorithm, Contiki OS

1. Introduction

The emergence of Internet of Things (IoT) has introduced new protocols to the TCP/IP network stack in order to accommodate the constraints of interconnected low-power sensors and actuators, aka the *"fingers of the Internet"*. Such trend started by the adoption of the IPv6 protocol in order to facilitate integrations of the expected billions of smart devices to the future Internet architecture. IPv6 adaptation is, however, necessary to respond to devices' constraints, hence the introduction of the 6LoWPAN (*IPv6 over Low-power Wireless Personal Area Network*) adaptation layer (G. Montenegro et al. 2007). Routing in the IoT also requires novel methods, which account for the unreliable nature of Low-power and Lossy Networking (LLN) introducing thus, the RPL (*IPv6 Routing Protocol for LLNs*) (Winter et al. 2012) and MPL (*IPv6 Multicast Protocol for LLNs*) (REF) routing protocols. This set of standards has enabled smart physical-world devices to communicate over the Internet.

Having connected LLNs to the Internet, new application layer protocols are necessary to allow easy and seamless interactions in the IoT. To this end, service oriented computing along with the REST architectural style have been adopted in order to respond to the high heterogeneity of involved IoT entities. For instance, the Constrained Application Protocol (CoAP) (Z. Shelby, K. Hartke, and C. Bormann 2014) and its extensions have emerged as a main enabler of IoT. Indeed, CoAP and its extensions made it possible for the resources and services provided by smart objects to be easily queried and accessed through conventional web browsers, which sets ground for a web of things.

In order to fully exploit the potential of the LLN network stack into providing successful adoptions of IoT, seamless and automatic discovery of available resources is an imperative. Such resource discovery solutions can be achieved using a multitude of techniques depending on a number of parameters including network size, application requirements and available infrastructure. For instance, fully-distributed solutions (Djamaa et al. 2014; Djamaa and Richardson 2014; Shelby 2012) can be well suited for an infrastructure-less zero-configurable IoT network, while a centralized solution (A. Yachir et al. 2016; Zach Shelby et al. 2016) might be deployed for large-scale IoT networks with dedicated discovery servers.

Both discovery approaches are being actively investigated by standardization bodies such as the Internet Engineering Task Force (IETF). Many standard formats and protocols are also being considered for such tasks including CoAP, DNS and others. Motivated by CoAP's successful adoption in the IoT, CoAP-based resource discovery approaches have been widely investigated with propositions of standardized technics for both

distributed resource discovery (Shelby 2012) and centralized Resource Directories (RDs)(Zach Shelby et al. 2016). Both solutions make use of the CoRE link format (Shelby 2012) as default for describing available resources and enabling their access. However, each solution has its motivations, required resources and target applications. Thus, whilst, the distributed resource discovery mechanism enables discovery without needing dedicated servers by relying on IP multicast, the RD solution requires the availability of resource-rich directories and relies on unicast communications for realizing its operations. A combined solution might provide the best out of both, by switching between them, depending on user requirements, available resources and network context.

This article introduces efficient, scalable and adaptive hybrid centralized/distributed discovery of CoAP resources. Such solution builds upon and extends the Proactive RD Discovery (PRD) mechanism proposed in our conference paper (Djamaa and Yachir 2016). Indeed, PRD is being enhanced and extended to serve as a key pillar for achieving efficient Hybrid Resource Discovery (HRD) over CoAP. The other pillar of the proposed HRD solution is a novel usage of CoAP for Fully-distributed push-pull Resource Discovery (FRD) in order to ensure efficient discovery operations in the absence, congestion, and/or failures of RDs. The rules and mechanisms of switching seamlessly between these two wings of the proposed HRD architecture are also detailed. More precisely, this article presents the following additional contributions:

- Proposition of a hybrid resource discovery architecture enabling automatic switch between centralized and distributed discovery of things and their resources using CoAP.
- Enhancing and extending PRD by introducing two algorithms for efficient discovery of multiple RDs and removal of unavailable RD information, along with its usage as enabler of the proposed architecture.
- Proposition of a novel usage of CoAP for achieving efficient distributed push-pull resource discovery with the necessary changes ensuring compactness and backward compatibility.
- Design, implementation and evaluation of lightweight, reliable and cost-effective mechanisms to implement the proposed architecture.

The reminder of this paper is structured as follows. Section 2 discusses existing work related to resource discovery over CoAP. Section 3 introduces the proposed HRD architecture with show cases of its operations and main mechanisms. Sections 4 and 5 detail the two main components of the proposed architectures namely PRD and FRD. This is followed by presenting a big picture of the functioning of HRD when combining the two components in section 6. Section 7 provides an analysis of the time/cost performance of the proposed components, which is followed by an extensive experimental evaluation in section 8. The paper ends in section 9 by conclusions and ideas for future directions.

2. Related Work

Service/resource discovery solutions can be achieved using a multitude of techniques depending on a number of parameters including network size, application requirements and available infrastructure. For instance, fully-distributed solutions (Djamaa et al. 2014; Djamaa and Richardson 2014; Shelby 2012) can be well suited for an infrastructure-less, small-size, zero-configurable IoT network, while a centralized solution (A. Yachir et al. 2016; Zach Shelby et al. 2016) might be deployed for large-scale IoT networks, having dedicated resource-rich discovery servers. With the emergence of CoAP as a communication protocol for the IoT, these classic discovery approaches are being considered with solutions provided for centralized, fully-distributed and hierarchic discovery over CoAP. This section will focus on reviewing such discovery solutions.

For the centralized architecture, the CoRE working group at the IETF has proposed a resource directory solution (Zach Shelby et al. 2016). In such an architecture, the RD stores all resources offered by CoAP servers, so, requesters can discover any required resource just by querying the RD. The RD provides a registration interface allowing providers to register the description of their public resources at the directory by issuing POST requests. Clients then query the RD by issuing GET requests, via the RD look-up interface, looking for descriptions matching their requests. The default description format adopted by the RD is the CoRE link format (Shelby 2012), which is carried as a payload in a CoAP message. Such description has many resource attributes including resource type (rt), interface description (if) and path. To achieve RD operations, new attributes have been defined in the RD draft such as the endpoint attribute (ep) specifying the endpoint hosting the resource registered in the RD, and the lifetime attribute (lt) indicating a valid registration period. A context attribute (con) is also introduced in order to allow an endpoint to specify the $scheme$, $ip\_address$ and the $port$ on which it will be accessible using the following format $scheme{:}\ ip\_address{:}\ port$ (e.g. $coap{:}//[FDFD::123]{:}\ 61616$).

The RD makes registered resources available at the $/.well-known/core$ resource. A client aware of an RD (via its con attribute) issues a GET request to the RD in the following format: $/.well-known/core\{?search*\}$, with the filter $\{?search*\}$ containing known attributes about the required resources. However, to exploit the

RD functionalities, CoAP nodes must first discover its presence in the network. Following the success of the RD, other researches have tried to build upon it and provide additional functionalities. For instance, TRENDY (Butt et al. 2012) divides nodes to Group Leaders (GL) and Group Members (GM). GLs and GMs are constructed based on their locations (e.g., the nodes in a room are assigned one GL and the remaining become GMs). This mechanism is used by TRENDY in responding to LLN-specific requirements such as group discovery. However, TRENDY might induces high maintenance overhead to manage the formation of GLs and GMs and maintain the network consistent over time. Overall, this approach requires the presence of a resource-rich central directory able to store all available services and may suffer for bottleneck and single-point-of-failure issues.

To cope with such issues, hierarchic and overlay-based discovery solutions over CoAP have been proposed. In (Mäenpää, Bolonio, and Loreto 2012), a use of the REsource LOcation And Discovery (RELOAD) protocol (C. Jennings et al. 2014) to discover CoAP resources is proposed. RELOAD forms an overlay network to provide storage and messaging services in a peer-to-peer (P2P) network and lets it open for applications to define use-cases. Thus, (Mäenpää, Bolonio, and Loreto 2012) describes a use-case on how to use CoAP with RELOAD in order to discover CoAP resources internetworked over a wide geographical area. Being tightened to the RELOAD infrastructure, this proposal might not address IoT-specific requirements. Another overlay-based work, dubbed Distributed Resource Directory (DRD), is proposed in (Liu et al. 2013). Alternatively to RD, DRD defines an overlay to play the role of an RD. DRD is constructed using a Distributed-Hash-Table P2P overlay offering discovery, registration and proxy services for CoAP nodes. However, the authors do not specify how to construct and maintain the overlay. In summary, this approach generally assumes the availability of resource-rich nodes to play the role of distributed directories and needs synchronization between them to keep resource information updated, which might imply high maintenance overhead.

In the absence of any directory to store resource descriptions, nodes make use of multicast/broadcast of requests/advertisements in order to realize resource discovery. Three possible ways might be used to accomplish distributed resource discovery, namely push mechanisms, pull mechanisms, and hybrid mechanisms. CoAP resource discovery (Shelby 2012) provides a pull mechanism to discover resources available in CoAP networks. Thus, as by (Shelby 2012) specification, a GET request to the appropriate multicast address might be made for $/.well-known/core$. Matching nodes reply with a payload in the CoRE link format, similarly to that of RD and DRD. Note that in order to limit the number and size of responses, the request has to specify known attributes.

Obviously each method has its pros and cons. We argue that a hybrid centralized/distributed approach may provide the best out of both worlds. Indeed, hybrid unicast/multicast CoAP-based discovery can not only ensure unicast- and multicast-only service but also provides hybrid operations and backup schemes in situations of failures. However, innovative mechanisms are required in order to make it practical. The questions on: how to detect the presence of RDs efficiently; how to achieve cost and time effective distributed discovery tasks over CoAP; and how to decide when to switch between the two modes should be addressed. These points are being investigated in the remainder of this article.

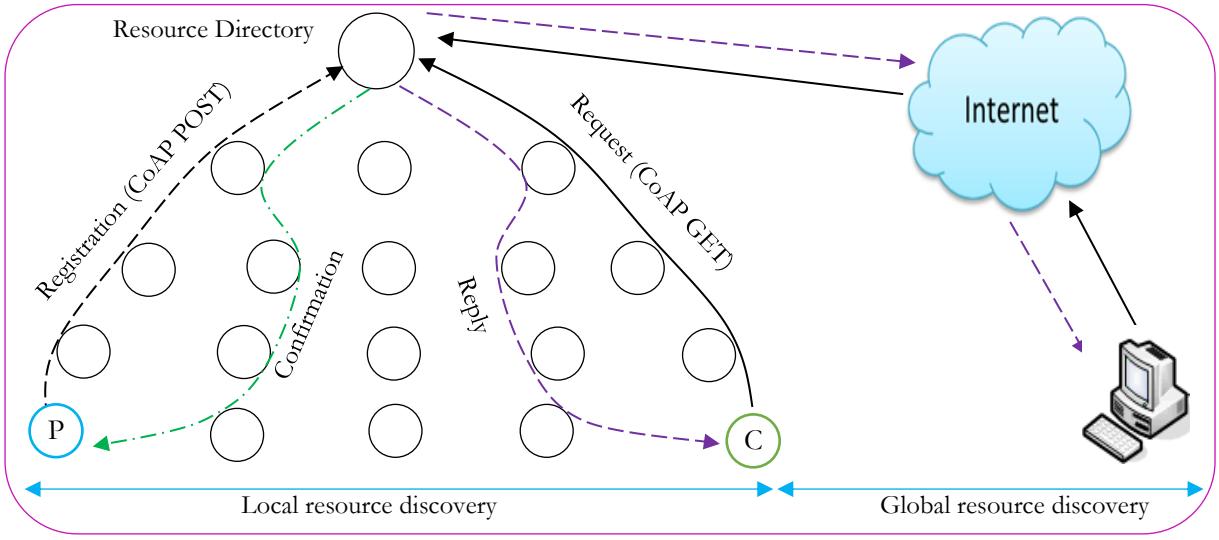
3. Hybrid unicast/multicast resource discovery (HRD)

This section presents an efficient architecture for hybrid unicast/multicast discovery based on CoAP. The main drive behind this architecture is providing continuous, efficient and reliable discovery depending on network state and available resources. Following this architecture, a node prefers to achieve the discovery of available resources via an RD if available as shown in Fig. 1 (a). Otherwise, the node falls-back to a fully-distributed discovery mechanism ensuring the continuity of services (Fig. 1 (b)). To realize such a discovery architecture two main points need to be treated. The first point focuses on how to achieve efficient, reliable and resource-lean detection of an RD in the network. The second treats the time-cost efficiency of the distributed mechanism ensuring fall-back services. Additionally, a third important point allowing seamless integration of both mechanisms is required. This latter should provide a simple and stateless way that ensures quick, efficient and transparent switch between the two discovery approaches while minimising the complexity of code, memory usage and computation operations.

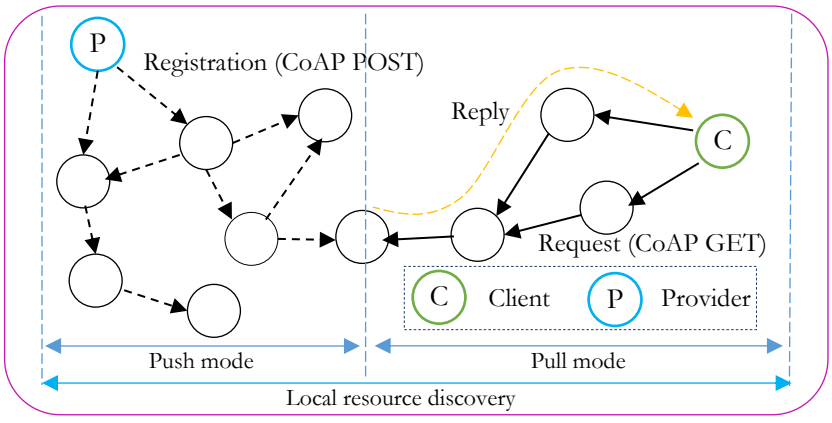
These three points are being developed in the remainder of this article. The first mechanism, in fact, presents a main building block for the efficiency of the proposed architecture. Indeed, before falling back to a fully-distributed approach, a node in the network starts by trying to detect the presence of an RD. Taking into account the number of nodes along with response time requirements makes reactive RD discovery fairly inefficient, which calls for efficient Proactive RD discovery (PRD). Detecting an RD, via PRD, enables both clients and providers to find RD information in their respective local directories and achieve unicast discovery operations. Otherwise, nodes switch to fully-distributed push-pull mechanisms. The time-cost efficiency of such a mechanism is a key

element of the proposed HRD architecture. To this end, we propose to adopt CoAP for a push-pull discovery protocol that ensures time efficiency through the push-mode and cost efficiency via the pull mode (Fig. 1 (b)).

The main advantage of the HRD architecture is seamless, automatic and efficient service discovery operations in different network scenarios. Take a scenario where the RD is not available in the network, our model will directly and automatically switch to a fully-distributed discovery (Fig. 1 (b)) with near zero-cost in time and network resources. Thus, providers failing to detect the RD automatically push their resources to neighboring nodes via multicast using the algorithm detailed in section 5. Clients also starts the direct discovery approach to locate available services in a zero-configuration, infrastructure-less manner. In the presence of an RD, however, thanks to the reliability of PRD (section 4), nodes find RD information locally and start discovery tasks through it. An important point of this architecture is that such a process is done at each discovery attempt ensuring continuity of service in case of failed or congested RDs while returning back to centralized discovery tasks after recovery. This way, our architecture prefers centralized discovery in the presence of non-congested RDs, ensures continuity of service at failures, congestion times and/or absence of RDs, and more importantly switch back to unicast discovery as soon as RDs become available. Finally, it should be noted that the proposed architecture natively supports both local and global discovery of available resources. For instance, an authorized remote client can query the RD over the Internet for available resources, while local clients might use both unicast and multicast discovery modes to discover available resources as shown in Fig. 1.



(a) Centralized resource discovery



(b) Fully-distributed push-pull resource discovery

Fig. 1 Overview of the proposed architecture

Having presented an overview of the proposed architecture and its enabling mechanisms, the following sections introduce and detail their functioning, interoperability and integrations.

4. Proactive RD Discovery (PRD)

4.1. PRD Overview

PRD allows a resource directory to proactively advertise its presence and provided capabilities for nodes to cache locally. The proposed mechanism makes use of CoAP messages and methods to achieve the discovery of an RD. It particularly adopts POST requests to proactively push RD information to the network in a CoAP message called Directory Advertisement (DA) as depicted in Fig. 2. The forwarding of such message is governed by a Trickle timer (Levis et al. 2011) as will be detailed in section 4.2. The advertised RD information is then cached for a specified lifetime, updated using either PUT or POST methods, and propagated in a wavelike pattern from nodes near the RD to those at the edge of the network as shown in Fig.2. Using Trickle ensures that, with time, all nodes will receive the RD's DA message. However, a node requiring to use RD services before receiving the DA can issue a single-hop multicast Directory Solicitation (DS) GET request looking for resources having the attribute $rt = core.rd$ (section 4.3). Upon reception of a DS request, a node having matching RD information may issue a response. Finally, PRD envisages a state maintenance mechanism (section 0) providing seamless reactions to network dynamics.

4.2. DA generation and forwarding

The RD generates and maintains its DA messages using a Trickle algorithm (Levis et al. 2011). Trickle allows nodes to get into a consistent state as quick as possible by relying on simple primitives. To do so, nodes periodically exchange their data to detect potential inconsistencies. When inconsistencies are detected, nodes shrink their transmission periods to the minimum interval size ($Imin$). In the absence of inconsistencies, the period is exponentially increased up to a configured maximum interval size ($Imax$). To minimize generated traffic, Trickle proposes a suppression mechanism. Thus, a node only transmits if the counted consistent received messages in an interval, via the consistency counter c, is less than the configured redundancy constant k. This Trickle behavior is summarized in the steps presented in Table 1.

Table 1 Trickle algorithm

Step	Action
1	Choose an interval size I randomly from $[Imin; Imax]$
2	When an interval starts, resets c to 0 and picks t randomly from $[I/2; I)$.
3	If receiving a consistent transmission, increments c.
4	At time t, only transmits if and only if c is less than k ($c < k$).
5	At an interval's end, double the interval size I up to $Imax$ and go to step 2.
6	If an *inconsistency* is received while $I > Imin$, put I to $Imin$ and go to step 2.

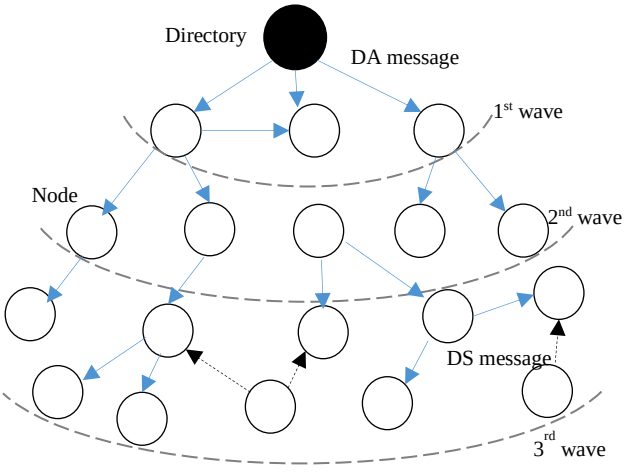


Fig. 2 Proactive RD discovery

evaluate_ incoming_post_request (DA)
IF *inconsistent_DA_from_new_RD* **THEN**
Create_new_resource_info (DA)
Create_new_trickle_timer (RD)
Trickle_inconsistency (RD) *//step 6 of Trickle*
ELSE IF (*inconsistent_DA_from_known_RD***) THEN**
update_existing_resource_info (DA)
Trickle_inconsistency (RD) *//step 6 of Trickle*
ELSE IF (*consistency***) THEN**
Trickle_consistency (RD) *//step 3 of Trickle*
ENDIF

Fig.3. DA Forwarding Algorithm

Using Trickle, in PRD, consists mainly of defining what constitutes a *consistent/inconsistent* DA transmission. To comply with the consistency model of Trickle, DA messages make use of the 16-bit $Message - ID$ field of a CoAP message. Thus, a consistency is defined as receiving a DA message from a specific RD with

the same $Message - ID$ as the cached one. Likewise, an *inconsistency* is specified as receiving a DA message of a specific RD having a different $Message - ID$ than the corresponding cached one. It should be noted that the $Message - ID$ of a DA is saved in a stable storage and only incremented by the RD when:

- The RD becomes enabled, reboots and/or when the lifetime of the previous DA is about to expire;
- The IP address and/or port number on which the RD is accessible change;
- Each time any advertised RD resource attribute changes (e.g., a change in the RD path).

Such DA message is then POSTed to the link-local multicast address. After receiving such a DA message, the receiver processes it and updates its resource list (creates a new resource if the information is from a new RD, updates *inconsistent* information of an existing RD, or discards a *consistent* message). The receiver then updates the corresponding Trickle timer accordingly as specified by the algorithm of Fig.3. It should be noted that PRD suppresses responses to DA messages similarly to CoAP group communication (A. Rahman and E. Dijk 2014). Finally, note that a node receiving a new DA message should create resource at a path inferred from the con attribute, which ensures its uniqueness, and includes it into its $/.well-known/core$ resource for the sake of making it discoverable to other nodes issuing DS messages.

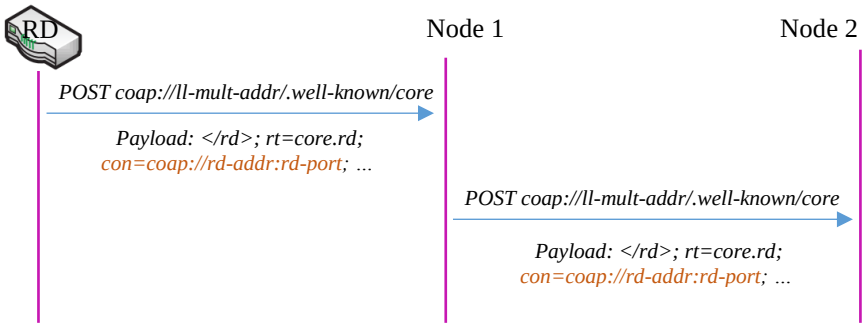


Fig. 3 Directory advertisements

4.3. DS generation and processing

In case of missing a DA, on-demand directory solicitations can be issued using the DS message. In addition to speeding up RD discovery, this functionality is particularly important for new nodes joining a network. For instance, a new node joining the network can discover the RD by issuing a DS message as shown in Fig. 4. DS messages are sent a MAX_DS_TRANSMISSIONS times separated by a DS_TRANSMISSION_INTERVAL. If still there are no responses, the originator switches to a slower transmission rate. The transmission of a DS is cancelled by receiving a response or a DA containing the requested information. For responding to DS messages, a node having matching resources generates a unicast response to be sent back to the DS originator, similarly to the standard resource discovery mechanism (Shelby 2012). Such response would have the format shown in Fig. 4. Note that, after receiving a response, a node might re-activate the Trickle timer to stay consistent.

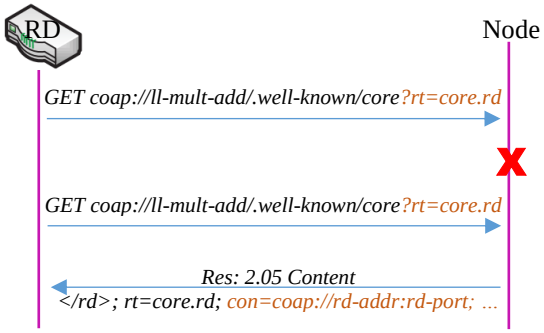


Fig. 4 Directory solicitation

Since multiple nodes might respond to a multicast DS, the congestion control mechanism suggested by CoAP should be used. Thus, nodes should insert a random delay, called leisure time, before issuing their responses. The lower bound of the leisure time can be approximated based on an estimate of the group size G, the data transfer

rate T and the response size S as follows: $LBLeisure = S \times GT$. If endpoints are not able to estimate such parameters, a default value of $5s$ might be used.

4.4. State maintenance

Being a proactive approach, PRD might suffer from network inconsistencies by keeping information about an RD which no longer exists. To deal with this, PRD envisages the following mechanisms.

The first enables a directory to advertise its graceful disappearance. This is done by issuing a DA message with the DELETE method to the path inferred from the con attribute of this RD which is unique for each RD as shown in Fig. 5. The forwarding of this message is governed by the same rules defined in section 4.2. Note, that a separated stopping Trickle timer that only manages DELETE messages might be preferred. This can have the advantage of handling DELETE requests differently from DA messages by using bigger values of k, for example. It also demonstrates a case were the infinite intervals of Trickle is not required (gossiping about a resource to be deleted a number of times is required for reliability purposes). Hence, for this mechanism to work efficiently, nodes should keep information about a deleted message for KEEP_DELETE_PERIOD before a definitive delete. Note that responses to DELETE messages are suppressed.

In addition to the explicit delete mechanism above, the lifetime-based soft deregistration mechanism (using the lt attribute) is responsible for deleting stale RD information. An RD should periodically generate new DA messages in order to refresh its information and confirms its presence. Finally, when contacting an RD and finding that it does not respond, nodes delete corresponding cached information.

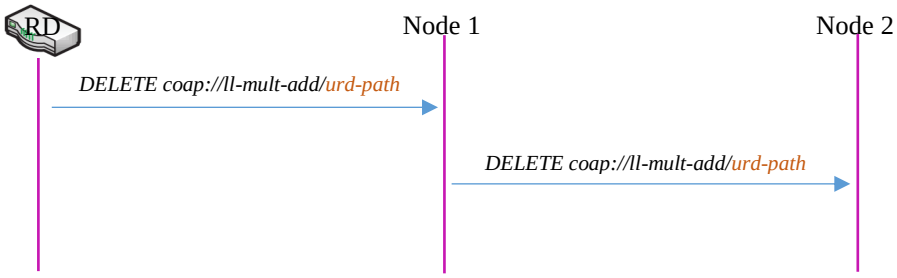


Fig. 5 Graceful suppression of RD information

4.5. Support of multiple RDs

Having multiple RDs might be preferable not only for redundancy reasons but also to support and provide different services. Indeed, a provider might be willing to register with an RD that supports a specific content-format/protocol. Similarly, a client might prefer to query an RD supporting its required parameters. PRD provides support of the discovery of multiple RDs in a network. To do so, each RD manages a separate Trickle. This requires nodes to keep separate Trickle timers per RD in a parallel Trickle approach. Such an approach might increase the burden on sensor/actuator nodes. However, taking into account the simplicity of Trickle code along with the small number of expected RDs in a network, this solution may present a fair trade-off. In section 6.1, an optimized version for supporting multiple RDs will be introduced.

Finally, it should be noted that by supporting discovery of multiple RDs, PRD can play a pivotal role into distributing hints allowing both providers and clients to be aware of available services of each RD and giving them all necessary information to access such services. For instance, PRD can distribute information about the content formats supported by an RD using the content type (ct) attribute. Clients and providers might use this information to select preferred content formats for interacting with RDs.

5. Fully-distributed pull-push resource discovery

Having presented PRD, this section introduces a CoAP-based Fully-distributed pull-push Resource Discovery (FRD) mechanism to achieve resource discovery in the absence of RDs. The dissemination part of the proposed mechanism builds upon that proposed in (Djamaa et al. 2014). The description part of the proposed mechanism leverages CoRE link formats with necessary adaptations as will be detailed below.

As depicted in Fig. 1 (b), FRD provides both pull and push-modes of distributed discovery. Indeed, FRD leverages a push-mode in which nodes proactively advertise descriptions of their own and cached resources. To

minimize resource utilization, FRD's push mode leverages on the advertisement algorithm proposed in (Djamaa et al. 2014), which is summarized in the following subsection. Additionally, FRD provides a pull mode for forwarding on-demand requests. For instance, a node not having required resources in its cache might issue a request to be propagated across the network. Upon finding a resource matching the requested criteria, a response is generated. A basic flooding algorithm or a multicast routing protocol (e.g., MPL) might be used to disseminate resource requests.

5.1. Trickle-based proactive advertisement

In (Djamaa et al. 2014), a Trickle-based mechanism is proposed for minimizing the traffic generated by proactive push of available resources. The mechanism mainly proposes another version of Trickle, which attaches the consistency counter to data items (resource descriptions in FRD). Thus, every resource description in a node's local directory has a consistency counter, which is updated following the below rules. In addition, and in order to minimize the generated maintenance traffic, a node only resets the consistency counters of its resources when starting new intervals. While this approach might not ensure strict consistency, it fits fully-distributed push-pull protocols well and gives it attractive proprieties. To achieve such proprieties each resource entry contained in an advertisement message (adv_msg) is modeled using mainly a vector (r, f, m) representing the resource description r, its sequence number f and a metric m (distance in hops). The former of the two parameters is used to ensure loop-free transmissions and it is incremented only by the provider, whilst the latter is used to limit entry's propagation; it is incremented by each forwarder.

The proposed advertisement algorithm in (Djamaa et al. 2014) decides on the eligibility of a resource entry for advertisements depending on its consistency. A resource entry in an adv_msg is considered consistent when the corresponding resource description is already in the node's local directory and considered as older, or announced as being far. More precisely, a consistent entry verifies the following points:

- The received entry (r, f, m) is already in the node's local directory, noted (r, f', m'), and has a lesser value of f $(f < f')$ or;
- The received entry (r, f, m) is already in the node's local directory, noted (r, f', m') and has the same value of f $(f = f')$ and a greater or equal value of m $(m \geq m')$.

A consistent entry is not eligible for advertisement. The algorithm only increments its consistency counter c. Otherwise, the entry is inconsistent. The algorithm proceeds to the registration of such an entry for the specified lifetime, increments and updates its distance m and reinitialise its consistency counter c to zero. If a first-inconsistency model is adopted, the Trickle timer is reset to allow its fast dissemination. Otherwise, the algorithm increments the *inconsistency* counter (ic) until reaching the configured value for resetting the timer. At time t of a Trickle interval, the algorithm includes in the outgoing message all entries whose distances are less than or equal the configured maximum advertisement distance having counters c less than the redundancy constant k $(c < k)$.

To illustrate the functioning of the above algorithm, let's take the example presented in Fig. 6, reproduced from (Djamaa et al. 2014), which depicts a network constructed of three nodes x, y and z. Node y's local directory before receiving an adv_msg is depicted in Fig. 6 (a) and contains three resource entries R1, R2 and R3 with their respective sequence numbers and distances from their providers. Upon receiving an adv_msg containing the resources R1, R2 and R4 with their respective values of f and m (Fig. 6 (b)), the algorithm investigates each entry. R1 is already present in node y's local directory and received with the same sequence number and distance $(f = f' and\ m = m')$, thus the algorithm only increments its c counter. R2 is already present in node y's local directory but received with a new sequence number $(f > f')$, the algorithm updates it and resets its c to zero. If the first-inconsistency approach is employed, the Trickle timer I is reset to $Imin$. R4 is new; the algorithm creates an entry for it with c set to zero. At time t, the algorithm loops over node y's local directory and includes in the outgoing advertisement entries with c counters less than k. Thus, if a constant $k = 1$ and a configured maximum advertisement distance = 4 are used, the outgoing advertisement contains R2 and R4 as illustrated in Fig. 6 (c).

Having summarized the functioning of the push-mode of FRD, the following sections detail its incorporation with CoAP. To this end, subsection 5.2 deploys CoRE link format for the push-mode and tweaks the proactive Trickle-based mechanism accordingly, while section 5.3 presents an optimized pull-mode for FRD.

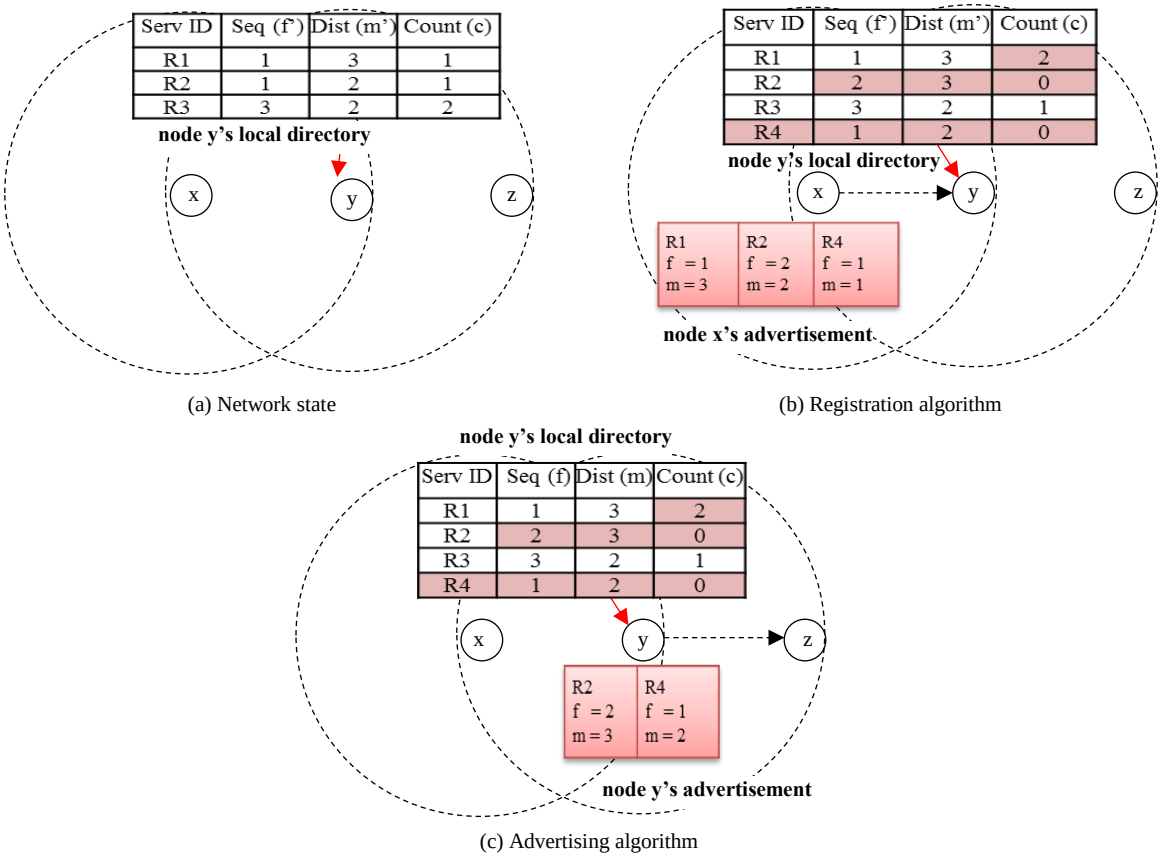


Fig. 6 An example of execution of the push-mode's dissemination algorithm

5.2. CoAP-based push resource discovery

This section introduces the necessary additional attributes to the CoRE link format in order to make the above dissemination algorithm compatible with CoAP along with new optimizations. Thus, advertisement messages have to follow the format imposed by CoAP (Fig. 7). To do so, we reused standard attributes whenever possible and added a few others. The fact of having multiple entries in a single advertisement message is achieved in CoAP by separating the multiple resource descriptions contained in the payload of a CoAP message by the ";" character.

Besides reusing the link format parameters introduced in (Zach Shelby et al. 2016), this section introduces the following attributes: the d attribute to represent the distance travelled by a resource description and the sequence number attribute of the description (seq). The following points present each attribute, its semantics and default value.

- con : this attribute is defined in with the following format $scheme: ip\_address: port$. The con attribute is mandatory for FRD since it allows to identify uniquely a provider of the resource description. It also enables to define the location of a resource in a unique manner by combing it with the resources path at its provider. This facilitates the update of such resources using the PUT method. It is also useful in the case of graceful departure of such a resource.
- seq : this is a newly added attribute to ease Trickle operations. It actually plays the role of the f factor described in the above section. Therefore, it is managed exactly as above and hence increased for each update.
- d: this is also a newly introduced attribute to limit the propagation of a resource description in the network. It plays exactly the same role as metric m defined in the previous section and hence managed in the same way.

With these attributes, the generic advertisement message described in section 5.1 will be mapped to a non-confirmable CoAP POST message as depicted in Fig. 7.

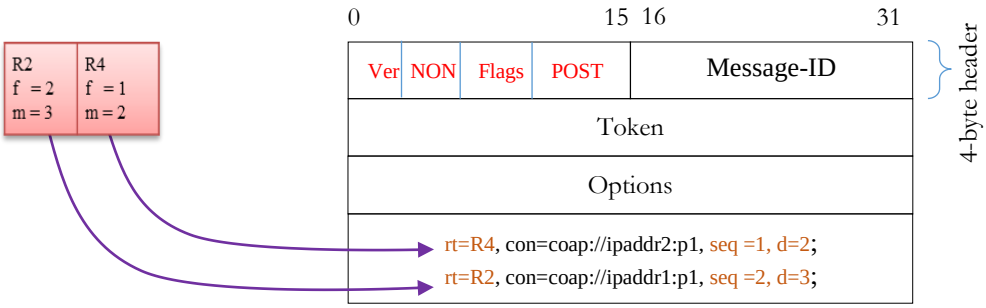


Fig. 7 CoAP mapping of the proposed push-model

5.3. Optimized CoAP-based pull mode

Besides limiting the distance of a request using the TTL field of the IPv6 header, the pull mechanism of the proposed FRD might render similarly to CoAP's distributed resource discovery. Hence, a client issues a multicast GET request to $/.well-known/core?search*$ sent to the allocated *all-coap-nodes* multicast address (Z. Shelby, K. Hartke, and C. Bormann 2014). Such requests risk to create a response storm problem. In addition, a client might be aware of some resources but requires others. In order to optimize the discovery, the newly introduced FETCH method (Stok, Bormann, and Sehgal 2016) will be used by FRD for the sake of minimizing the amount of responses along with specifying the required exact format of sought resources.

Unlike with GET, the request parameters are constituted of both URI options and multiple FETCH payloads. Thus, the client might insert in the payload of a FETCH request information about known resources in order to inform responders to eliminate such resources from their responses. Additionally, the client might specify the required content format and any useful information to accurately filter the responses only to the ones required. In the current version of FRD, this optimization is only implemented by the source of a request. Giving intermediate nodes the possibility of modifying the content of a FETCH request is envisaged in order to further minimize the number and size of expected responses. It should be noted that while featuring the above optimizations, FRD remains backword compatible with CoAP.

6. Hybrid resource discovery in practice

Having presented the two main pillars of HRD, this section presents the mechanisms allowing it to efficiently address the requirements of LLN discovery. Indeed, it shows an incorporation of PRD with FRD for efficient code and memory usage, and presents a big picture on the functioning of HRD.

6.1. PRD in FRD for code and memory optimizations

Because of the similarities between PRD and FRD forwarding algorithms, it might be preferable to consider a way of integrating them in a simple code that responds to both requirements. Although FRD's Trickle-based proactive advertisement does not need strict consistency in its generic form, it can provide strict consistency for a few crucial resources if needed. For instance, a flag in a resource entry of an adv_msg can indicate that this is an important resource, and hence nodes keep gossiping about it (as if it is one of their own resources) to ensure that it reaches all network entities.

With this propriety of FRD's Trickle algorithm, incorporating PRD functions into FRD translates into adding another CoRE link format attribute, resource criticality (rc), for specifying the criticality of a resource. rc, if present in a received resource entry, indicates that the resource is critical and hence must arrive at all nodes. Thus, rc must be present in all entries representing RD resources in order to indicate to FRD to keep gossiping about them at each interval as in PRD. In the absence of such an attribute or having its value to zero, the resource is assumed not critical and FRD, as described in section 5, is responsible on forwarding it. When not suing the first-inconsistency model, it is important to reset the Trickle timer of FRD when receiving the first resource entry having the rc attribute set to 1 in order to speed up its propagation. Finally, it should be noted that while the push algorithm of FRD can be disabled, it must be enabled to forward critical resources. Thus, a node receiving such resources must activate its Trickle timer if it is disabled.

Incorporating PRD in FRD provides another way of treating multiple RDs. Indeed, instead of using a Trickle timer per RD, FRD allows the information of multiple RDs to be managed by one Trickle timer. This optimizes the code, memory and even energy since larger messages generally consume less energy than the same amount of

data transmitted in smaller messages. This is especially true if the radio duty cycling protocol deploys a multicast burst forwarding mechanism (Djamaa et al. 2015).

6.2. Hybrid resource discovery in practice

Having introduced the generic architecture for hybrid centralized/distributed resource discovery for the IoT and introduced the components participating in realizing such an architecture, we present in this section a big picture for the functioning of such a solution.

In an ideal environment, an RD should not receive multicast advertisements and multicast requests. However, due to network losses and other factors, an RD might still receive such messages as some nodes might not be aware of its presence temporarily. Receiving such messages could give the RD hints that a client and/or provider is not aware of its presence. The RD can issue a confirmation to the provider or a response to the client informing them of its presence, which prompts them to switch to centralized discovery operations. Similarly, a node aware of the existence of an RD, knowing that the RD is still available, which receives multicast requests/advertisements might transmits via unicast directly to the RD. When the RD issues confirmations/responses the originator of the message, it will be aware of RDs presence. On the other hand, when an RD disappears from the network and in case of failure of the PRD state maintenance mechanisms (Section 0), some nodes might still hold its information temporarily. After a specific number of attempts at an RD, a provider and/or a client might switch to FRD to continue its operations.

Finally, it should be noted that the above mechanisms on switching between unicast and multicast discovery can ensure the correctness, accuracy and speed of discovery within the proposed architecture. For instance, a provider missing RD information for whatever reason will switch automatically and instantly to fully-distributed discovery by advertising its resources via FRD. When arriving to a neighbor having information about the RD, the resources will be sent via unicast to the RD, allowing thus a client to discover all available resources in the network by fetching the RD.

7. Formal performance analysis

This section provides a formal analysis of the two components of the proposed HRD architecture (PRD and FRD) and their combination. The parameters and notations used in the analysis are summarized in Table 2.

Table 2 parameters and notations

Parameter	***Meaning***
N	Number of nodes in the network
Nr	Number of new resources in an interval
R	Total number of resources in the network
RDs	Number of RD nodes in the network
D	Network Diameter
k	Trickle's redundancy constant

7.1. PRD analysis

With the default RD discovery (Shelby 2012) and since each node needs to issue a request in order to discover an RD, the number of generated application-layer requests is proportional to the number of nodes and hence it grows linearly with network size (application-layer overhead complexity in $O(N)$). Taking into account the fact that each request can result into multiple multi-hop multicast messages, the overhead generated by a request is considerably higher. In a scenario, where the multicast routing protocol deploys a simple flooding algorithm, each node will forward a single request at most once giving an overhead on $O(N)$ per request, which makes the actual number of transmissions generated for discovering an RD using the default mechanism grows on $O(N^2)$. Furthermore, since the RD has to respond to each node's request separately, the number of responses also grows linearly with the number of nodes. Moreover, taking into account the multi-hop aspect of LLN networks, a single RD response results into multiple transmissions, which are proportional to the diameter of the network (D). Besides overhead, the time of discovering an RD is also proportional to the diameter of the network. Thus, request propagation requires theoretically D transmission times with a similar time required to receive a reply.

Since the number of RDs is drastically smaller than the number of nodes in an LLN network, PRD provides an efficient solution to both overhead and time issues of the default RD discovery mechanism. Thus, having an

RD advertising its presence to the network, nodes will simply skip the discovery of the RD and start using unicast primitives to exploit its services. Indeed, using PRD, the overhead of discovering an RD is proportional to the number of RDs; which is very small compared to the number of nodes and hence the application-layer message complexity of PRD is in $O(RDs)$. Taking into account the multi-hop aspect of LLNs, the worst-case message complexity for an RD using flooding goes in $O(N)$. Note that using Trickle minimizes this complexity into $O(k)$ transmission per interval in a perfect lossless network, but in this analysis, we considered a worst-case scenario for both default resource discovery and PRD. Moreover, PRD saves the traffic that would be incurred by generating responses since no requests/responses are exchanged. Finally, the time complexity of the PRD can be zero since in most of the cases the RD information is locally available. Table 3 summarizes the message and time complexities of both approaches.

Table 3 PRD's message and time complexities.

RD discovery mechanism	App. Requests	Transmissions	App. Responses	Discovery Time
PRD	$O\ (RDs)$	$O\ (N \times RDs)$	0	0 or $O\ (D)$
Resource discovery	$O\ (N)$	$O\ (N^2)$	$O\ (N)$	$O\ (D)$

7.2. FRD analysis

In (Djamaa et al. 2014), a formal analysis of the proactive advertisement algorithm is presented. Such an analysis remains valid for FRD since the main difference resides in the description format and the only parameter changing is the size of a resource. Thus, this section adds on such an analysis to include the performance of PRD when integrated with FRD and compares it with FRD and a fixed-period push mechanism. As in (Djamaa et al. 2014), the analysis is limited to lossless single hop networks. The methodology presented in (Levis et al. 2004) may be used to generalize the discussion to lossy multi-hop networks. The performance is analyzed over one Trickle interval w.r.t the number of exchanged advertisements, the size of an advertisement, the total amount of generated push traffic and the inconsistency resolution time. This is done when assuming that a provider provides at most one resource. For ease of the analysis, we also assume that all the resources have the same $resource\_size$.

In FRD and since nodes are assumed to provide unique resources, each node will send its resource description even after it hears other nodes' consistent descriptions. This makes FRD depend only on the number of new resources (Nr) occurring in an interval, which gives the number of advertisements a good scalability as the number of new resources in an interval is generally much less than the total number of resources. When combining FRD with PRD, a node may send an extra message per RD in a worst case where RD information occur after sending that of a previous RD, making its message complexity on $O(Nr + RD)$. Finally, in fixed-period push algorithms where each node transmits once per interval, the number of messages scales linearly with the number of nodes.

FRD presents a very attractive characteristic regarding the maximum size of an advertisement as it is bounded by $(k + 1) \times resource\_size$ i.e. the maximum number of entries included in an advertisement message is $k + 1$. When combining PRD with FRD, critical resources should be included in the advertisement message increasing its size by the number of RDs in a worst case scenario. In a fixed-period push algorithm, however, where each advertisement contains all stored entries, the size of an advertisement might reach the number of resources R, giving it a linear scalability with the number of resources. With regards to the total amount of traffic generated in an interval, which has a direct impact on the communication's energy consumption, FRD sends in a worst case approximately $(k + 1) \times Nr$. When combining PRD with FRD the amount of generated traffic will be $(k + 1 + \mathrm{RDs}) \times Nr$. The same analysis gives fixed-period push algorithms a total amount of generated push traffic in $O\ (N \times R)$.

For inconsistency resolution time, FRD can detect and react to an inconsistency in a fixed time span in $O(1)$ since the first inconsistency resets the Trickle timer for fast propagations, which is in the same order of fixed-period push algorithms. A summary of this performance analysis is given in Table 4.

Table 4 summary of FRD's time and message complexities

algorithm/parameters	# messages	max adv_size	Total traffic	Inconsistency time
FRD	$\mathrm{O(Nr)}$	$\mathrm{O(K + 1)}$	$\mathrm{O((K + 1) \times Nr)}$	$\mathrm{O(1)}$
FRD + PRD	$\mathrm{O(Nr + RDs)}$	$\mathrm{O(K + 1 + }RDs\mathrm{)}$	$\mathrm{O((K + 1 + RD) \times (}RD + Nr\mathrm{))}$	$\mathrm{O(1)}$
Fixed-period push	$\mathrm{O(N)}$	$\mathrm{O(R)}$	$\mathrm{O(N \times R)}$	$\mathrm{O(1)}$

8. Performance Evaluation

Experimental evaluations of the proposed architecture are done in two parts. The first focuses on PRD performance, while the second evaluates the performance of FRD.

8.1. Experimental model and performance metrics

To evaluate the performance of the proposed HRD solution, we implemented it in Contiki OS. To put results into context and in order to provide a comprehensive evaluation, the mechanisms building our solution are compared to state-of-the-art algorithms. For instance, PRD is compared with RD's default discovery mechanism based on CoAP's built-in resource discovery. To allow resource discovery across a multi-hop network, the MPL implementation in Contiki was used. Note that such an implementation follows the specifications of draft-1 of MPL called Trickle Multicast (TM). For providing a fair comparison, the same Trickle parameters were used for both TM and PRD. For FRD, its performance is compared, in a home automation scenario, to that of CoAP's resource discovery using the SMRF multicast protocol (Oikonomou, Phillips, and Tryfonas 2013). By comparing PRD and FRD with state-of-the-art solutions, we aim to demonstrate the capabilities of the proposed architecture.

Two simulated network scenarios depicted in Fig. 8 were considered. A ten node scenario is used to validate PRD and show its advantages even in small-size networks. A 31 node scenario of a publically available network configuration[1] is used to show the performance of FRD in home automation and similar network scenarios. Emulated Sky motes (Polastre, Szewczyk, and Culler 2005) were used in order to show the lightweight aspect of our algorithms. Each mote in our scenarios can play the role of either a client requesting required resources or a provider registering its resources or both roles. In all cases, a node is required to be aware of the RD before accessing it or deciding to switch to the fully-distributed mode.

Table 5 simulation parameters

Parameter	***Value***
Simulation time	200 and 600 seconds
Number of nodes / iterations	10 and 31 / 10
Radio environment	UDGM
Communication range/bandwidth	50m / 250Kb/s
Network / Adaptation Layer	IPv6 / 6LoWPAN
MAC / RDC Protocol	CSMA-CA / CX-MAC
TM / PRD / FRD Trickle parameters	$Imin = 1s, \quad Imax = 8\,s, \quad k = 1$

For this evaluation, we developed applications running the two components of HRD above UDP in constrained 6LoWPAN networks running Contiki. At the routing layer, default resource discovery deploys TM to ensure multicast routing of RD discovery requests. RPL was operating in order to route unicast requests, registrations and responses between involved entities. At the link layer, the CX-MAC (Buettner et al. 2006) radio duty cycling protocol was used. Simulation configuration parameters are summarized in Table 5.

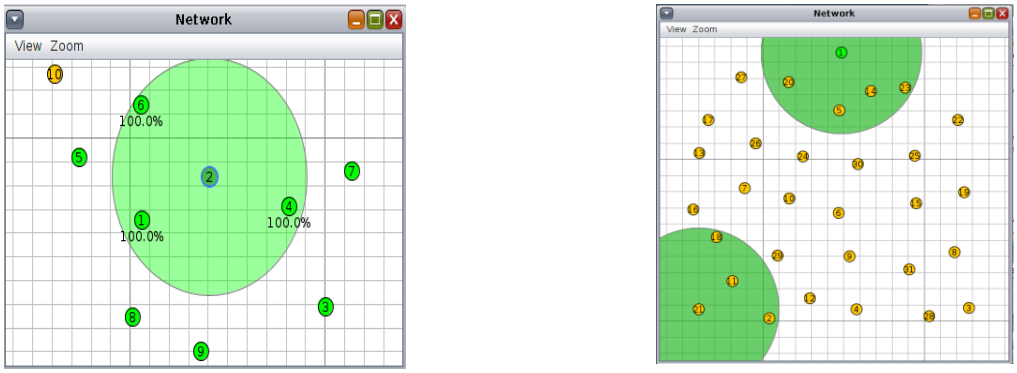


Fig. 8 network scenarios used in the evaluations

[1] https://github.com/contiki-os/contiki/blob/master/examples/ipv6/rpl-udp/rpl-udp.csc

To be able to draw conclusions on the time/cost performance of the evaluated approaches, the number of generated messages, the average discovery time, the average RD discovery time and the radio duty cycle, as proxy of energy consumption, were measured when varying the number of clients and providers in the network. The RD discovery time is measured as the time spent from sending a request until receiving the reply, averaged over all requests. The number of generated messages is measured at the routing layer for resource discovery and averaged over all nodes. The network duty cycle (the percentage of time spent in active state), as an indicator of energy consumption, is measured using Contiki's power profiler (Dunkels et al. 2011). Each experiment was repeated 10 times. The mean is reported in the graphs of Fig. 9 and Fig. 10.

8.2. Results and discussions

8.2.1. PRD discussions

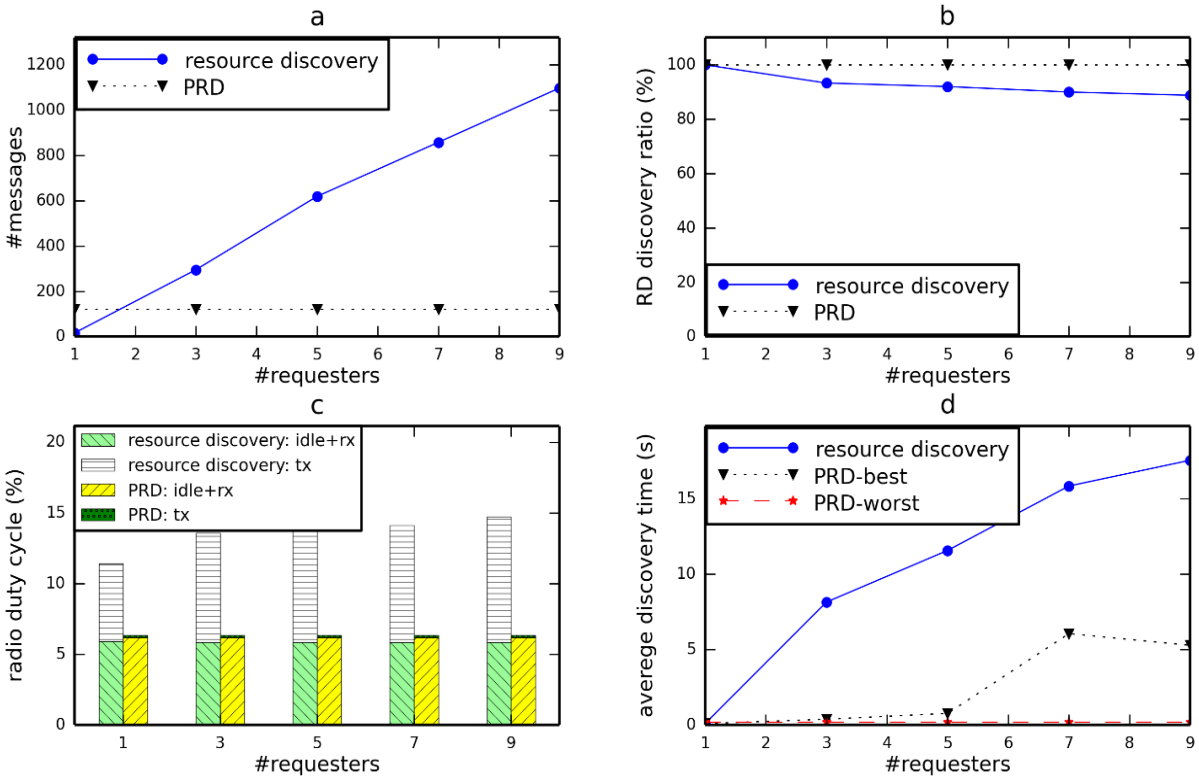


Fig. 9 PRD simulation results

As can be seen from Fig. 9 (a), the number of messages generated by the resource discovery mechanism increases drastically with increasing number of RD requesters in the network reaching about 10 times that generated by PRD for nine active requesters. The traffic generated by PRD, on the other hand, stayed independent of the number of requesters yielding a small generated traffic, which demonstrates the performance of PRD in terms of generated traffic. Concerning reliability, Fig. 9 shows that PRD can achieve a 100% RD discovery rate independently from the number of requesters thanks to Trickle reliability mechanisms making the information about the RD reaching each node. The default resource discovery mechanism, however, showed a decrease in discovery rates with increasing number of requesters reaching about 88% at 9 concurrent requesters. This decrease in reliability is expected to continue with increasing number of requesters because of an abundant number of concurrent multicast requests being forwarded.

Concerning energy consumption, Fig. 9 (c) shows that PRD achieves smaller radio duty cycles, hence smaller energy consumption when compared with resource discovery. Thus, even when resource discovery registered less number of messages, the energy consumed to deliver them was about twice that of PRD. This can be explained by the growing size of control traffic generated by the routing protocols along with the expensive cost of multicast communications. Finally, Fig. 9 (d) shows that in a best case, where all nodes requesting RD services find the information locally, PRD can achieve zero discovery time. In a worst case, where the requesters need to wait for the information to be pushed, the discovery time of PRD is still better than that of the default resource discovery mechanism.

8.2.2. FRD discussions

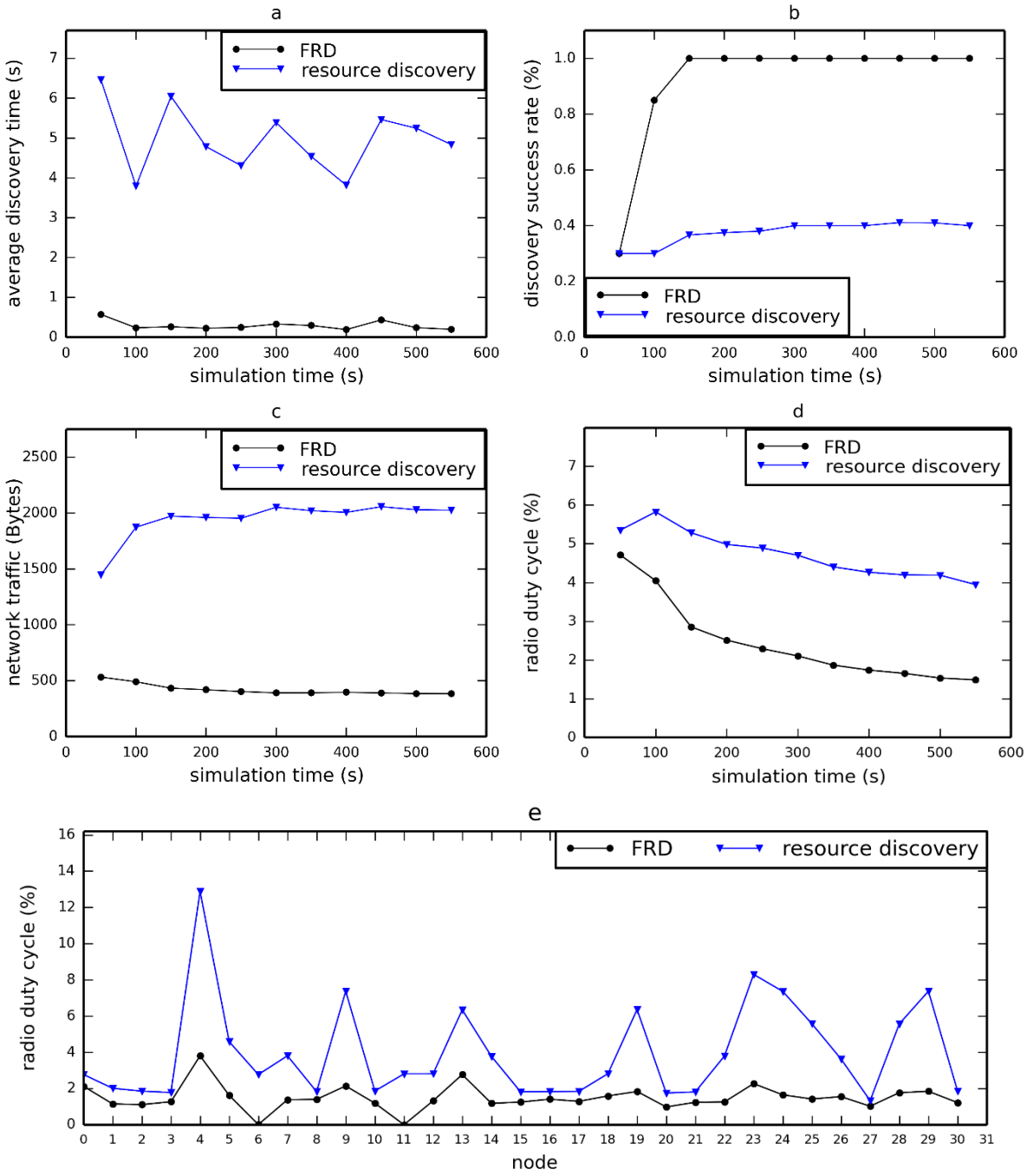


Fig. 10 FRD simulation results

Fig. 10 (a) shows that the discovery time of the default resource discovery registered a higher value of around six times that of PRD. This can be explained by the fact that resource discovery needs to wait for a request to get propagated throughout the whole network and the response to be sent back over long distances. FRD, however, benefited from the proactive push of available descriptions, which allowed sought descriptions to be found a limited number of hops from the client. This way FRD gained in both request and response times.

Concerning the discovery success rate, the graph in Fig. 10 (b) shows that the default discovery mechanism only achieved around 40% success rate. This low rate might be caused by generating big multicast traffic spanning long distances, which increased the chances of collisions and losses. FRD, however, achieved a satisfactory success rate nearing 100% towards the end of the simulation as shown in Fig. 10 (b). This is explained by the fact that at the start of the simulation, requests have to propagate throughout the network in order to find sought resources with high risk of losses. When resources get propagated through the network, clients find them nearby and from multiple sources increasing thus the discovery rate towards its maximum.

For the total amount of traffic generated in the network, Fig. 10 (c) shows that FRD outperforms by large the default resource discovery mechanism. This performance is partly achieved thanks to Trickle suppressions. Additionally, FRD saves a considerable amount of traffic by annulling request propagation over long distances thanks to the availability of sought descriptions in nearby devices, while resource discovery keeps on sending multicast requests across the whole network. Moreover, losses and collisions, diminishing the success rate (Fig. 10 (c)), might have incurred more retransmissions at lower stack layers. Note that this result might change in frequently dynamic environments as the generated push traffic may become more important.

Finally Fig. 10 (d) and (e) show the energy consumption aspects of both solutions. CoAP resource discovery consumed more energy as shown in Fig. 10 (d), which can be explained by the large number of multicast messages generated and propagated throughout the network. FRD consumed less energy thanks to minimizing the number of multicast packets. Fig. 10 (e) shows that CoAP resource discovery might not distribute energy consumption equitably between the nodes because it builds upon the RPL graph. Conversely, FRD nodes showed similar energy consumptions thanks to Trickle balanced forwarding decisions.

As a synthesis, both PRD and FRD showed important performance when compared with state-of-the-art standardized solutions. This gives the proposed architecture the necessary mechanisms for achieving efficient hybrid resource discovery.

9. Conclusions and Outlook

In this paper, a hybrid unicast/multicast resource discovery architecture is proposed. The architecture builds upon two main mechanisms, namely PRD and FRD allowing it to ensure efficient, scalable and quick discovery of available resources. Formal analysis along with extensive time-accurate emulations in dedicated IoT platforms have demonstrated the capabilities of such mechanisms concerning time efficiency, resource economy and reliability when compared with state-of-the-art solutions. Future work consist on extending the formal analysis to other scenarios along with further evaluations and optimizations in publicly available testbeds. A specification to be presented to the IETF is also planned.

References

A. Rahman, and E. Dijk. 2014. ‘Group Communication for the Constrained Application Protocol (CoAP)’. *RFC 7390, IETF*, October. http://tools.ietf.org/html/rfc7390.

A. Yachir, Y. Amirat, A. Chibani, and N. Badache. 2016. ‘Event-Aware Framework for Dynamic Services Discovery and Selection in the Context of Ambient Intelligence and Internet of Things’. *IEEE Transactions on Automation Science and Engineering* 13 (1): 85–102.

Badis Djamaa, Mark Richardson, Peter Barker, and Mohamed Aissani. to appear. ‘Multicast Burst Forwarding in Constrained Networks’. In . Glasgow.

Buettner, Michael, Gary V. Yee, Eric Anderson, and Richard Han. 2006. ‘X-MAC: A Short Preamble MAC Protocol for Duty-Cycled Wireless Sensor Networks’. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, 307–320. SenSys ’06. New York, NY, USA: ACM. doi:10.1145/1182807.1182838.

Butt, Talal Ashraf, Iain Phillips, Lin Guan, and George Oikonomou. 2012. ‘TRENDY: An Adaptive and Context-Aware Service Discovery Protocol for 6LoWPANs’. In *Proceedings of the Third International Workshop on the Web of Things*, 2:1–2:6. WOT ’12. Newcastle, United Kingdom: ACM. doi:10.1145/2379756.2379758.

C. Jennings, B. Lowekamp, E. Rescorla, S. Base, and H. Schulzrinne. 2014. ‘REsource LOcation And Discovery (RELOAD) Base Protocol’. *RFC 6940, IETF*, January. http://www.hjp.at/doc/rfc/rfc5911.html.

Djamaa, Badis, and Mark Richardson. 2014. ‘Towards Scalable DNS-Based Service Discovery for the Internet of Things’. In *Lecture Notes in Computer Science. Ubiquitous Computing and Ambient Intelligence. Personalisation and User Adapted Services*, 432–35. Lecture Notes in Computer Science 8867. Springer. http://link.springer.com/chapter/10.1007/978-3-319-13102-3_70.

Djamaa, Badis, Mark Richardson, Nabil Aouf, and Bob Walters. 2014. ‘Towards Efficient Distributed Service Discovery in Low-Power and Lossy Networks’. *Wireless Networks* 20 (8): 2437–53. doi:10.1007/s11276-014-0749-3.

Djamaa, Badis, and Ali Yachir. 2016. ‘A Proactive Trickle-Based Mechanism for Discovering CoRE Resource Directories’. *Procedia Computer Science* 83: 115–22. doi:10.1016/j.procs.2016.04.106.

Dunkels, Adam, Joakim Eriksson, Niclas Finne, and Nicolas Tsiftes. 2011. ‘Powertrace: Network-Level Power Profiling for Low-Power Wireless Networks’. Technical Report T2011:05. Swedish Institute of Computer Science. http://soda.swedish-ict.se/4112/.

G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. 2007. ‘Transmission of IPv6 Packets over IEEE 802.15.4 Networks’. *RFC 4944, IETF*, September. http://tools.ietf.org/html/rfc4944.

Levis, Philip, T. Clausen, J. Hui, O. Gnawali, and J. Ko. 2011. ‘The Trickle Algorithm’. *RFC 6206, IETF*. http://www.hjp.at/doc/rfc/rfc6206.html.

Levis, Philip, Neil Patel, David Culler, and Scott Shenker. 2004. ‘Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks’. In *In Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)*, 15–28.

Liu, Meirong, Teemu Leppanen, Erkki Harjula, Zhonghong Ou, Archana Ramalingam, Mika Ylianttila, and Timo Ojala. 2013. 'Distributed Resource Directory Architecture in Machine-to-Machine Communications'. In *IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 319–24. doi:10.1109/WiMOB.2013.6673379.

Mäenpää, Jouni, Jaime Jiménez Bolonio, and Salvatore Loreto. 2012. 'Using RELOAD and CoAP for Wide Area Sensor and Actuator Networking'. *EURASIP Journal on Wireless Communications and Networking* 2012 (1): 1–22.

Oikonomou, George, Iain Phillips, and Theo Tryfonas. 2013. 'IPv6 Multicast Forwarding in RPL-Based Wireless Sensor Networks'. *Wireless Personal Communications* 73 (3): 1089–1116. doi:10.1007/s11277-013-1250-5.

Polastre, J., R. Szewczyk, and D. Culler. 2005. 'Telos: Enabling Ultra-Low Power Wireless Research'. In *Fourth International Symposium on Information Processing in Sensor Networks, 2005. IPSN 2005*, 364–69. doi:10.1109/IPSN.2005.1440950.

Shelby, Zach. 2012. 'Constrained RESTful Environments (CoRE) Link Format'. *RFC 6690, IETF*. http://xml2rfc.tools.ietf.org/html/rfc6690.

Stok, P. van der, C. Bormann, and A. Sehgal. 2016. 'Patch and Fetch Methods for Constrained Application Protocol (CoAP)'. *Internet Draft, IETF*. http://tools.ietf.org/id/draft-ietf-mmusic-rfc2326bis-40.html.

Winter, T., P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. P. Vasseur, and R. Alexander. 2012. 'RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks'. *RFC 6550, IETF*, March.

Z. Shelby, K. Hartke, and C. Bormann. 2014. 'The Constrained Application Protocol (CoAP)'. *RFC 7252, IETF*. http://www.rfc-editor.org/rfc/pdfrfc/rfc7252.txt.pdf.

Zach Shelby, M. Koster, C. Bormann, and Peter van der Stok. 2016. 'CoRE Resource Directory'. *Internet Draft, IETF*, draft-ietf-core-resource-directory-08, , July. http://tools.ietf.org/html/draft-ietf-core-resource-directory-05.

Hybrid CoAP-based resource discovery for the Internet of Things

Djamaa, Badis

2017-02-16

Badis Djamaa, Ali Yachir, Mark Richardson. (2017) Hybrid CoAP-based resource discovery for the Internet of Things. Journal of Ambient Intelligence and Humanized Computing, Volume 8, Issue 3, June 2017, pp. 357-372

http://dx.doi.org/10.1007/s12652-017-0450-3