Edge-centric Distributed Discovery and Access in the Internet of Things

G. Tanganelli, C. Vallati and E. Mingozzi

***Abstract*—Massive diffusion of constrained devices communicating through low-power wireless technologies in the future Internet of Things will require in many scenarios the deployment of IoT gateways to allow applications to discover and access IoT resources. In this context, Fog/Edge Computing will represent an opportunity to deploy IoT gateways in proximity of IoT domains, meeting the requirements of applications needing low-latency interactions with devices. In this work, we present an Edge-centric distributed architecture to provide resource discovery and access services to IoT applications. The proposed approach leverages the CoRE Resource Directory interface and the CoAP protocol to expose a standard interface for global discovery and access. Multiple IoT gateways are federated through a P2P overlay implemented by a Distributed Hash Table (DHT), which is exploited to share the information on IoT resources available across multiple domains. The proposed solution is validated by means of a small-scale prototype, and extensively evaluated through emulation in large scale deployment in comparison to a centralized Cloud-based approach. Experimental results demonstrated that the proposed approach guarantees lower latencies than a centralized solution and can ensure scalability for small to medium sized deployments.**

***Index Terms*—IoT gateway, Fog Computing, CoAP, Resource Directory, Discovery, Access**

I. Introduction

EMERGING technologies, such as low-power wireless connectivity and new hardware for sensors and actuators, will represent the fundamental building blocks of the future Internet of Things (IoT), since they will allow producing low-cost and easy deployable embedded devices [1]. Integrating such devices into IoT applications, however, will represent a significant challenge, considering their constrained capabilities in terms of computing, memory and energy. Although recent standardization efforts enabled the use of the IPv6 protocol to allow end-to-end direct communication between constrained IoT devices and applications [2], in many practical deployments this solution may not be efficient or even unfeasible [3], considering that the access network may be highly unreliable, or IoT devices may often be offline, e.g., during sleeping cycles.

To handle this issue, an *IoT gateway* may be introduced as an intermediary logical entity between applications and constrained devices. An IoT gateway may support discovery of IoT resources, and access to them, thus behaving both as a *directory* to lookup, collecting information about available resources, and as a *broker*, handling application requests on behalf of constrained devices whenever they are unavailable. This approach is also endorsed by many existing standard or open source platforms such as ETSI oneM2M [4] or FIWARE [5].

A common approach in many existing IoT deployments is to place IoT gateways in a Cloud infrastructure to guarantee scalability, ease of deployment and low-cost management. However, with this approach, IoT gateways actually run in data centers far from the IoT infrastructure comprising the physical devices, and communicate with the latter through long-distance high-latency Internet core links.

On the other hand, in certain IoT scenarios, applications may have low-latency, privacy, or geographic constraints that can't be met by a Cloud-based solution. In order to support such applications, the *Fog/Edge Computing* paradigm has been recently proposed [6]. Fog/Edge Computing is an extension of the traditional Cloud Computing architecture that distributes advanced computing, storage, networking and management services closer to end users and things. This is accomplished by exploiting computing and storage capabilities deployed at the edge of the network, thus realizing a distributed run-time platform that provides key advantages such as real-time processing, rapid scalability, context-awareness and resource pooling [7]. For these reasons, Fog/Edge Computing can be exploited in many IoT scenarios, ranging from smart agriculture to smart cities [8]. As a matter of facts, placing IoT gateways in *Fog nodes* located in proximity of IoT devices is already commonly recognized by both the research community [9] and standardization bodies [10] as a major application scenario for Fog/Edge Computing.

Moreover, in future large-scale IoT systems, multiple IoT domains are expected to interwork with each other. In a Smart City environment, for instance, different IoT systems may be deployed in the same area, e.g., for intelligent transportation systems, environment monitoring, etc., each one operating for a different main goal [11]. To integrate different IoT infrastructures we will require the federation of such domains and, in particular, the cooperation of IoT gateways, in order to enable seamless discovery and access of IoT resources across multiple

G. Tanganelli, C. Vallati and E. Mingozzi are with the Department of Information Engineering, University of Pisa, Largo Lucio Lazzarino, 2, 56122, Pisa Italy. (e-mail: g.tanganelli@iet.unipi.it, c.vallati@iet.unipi.it, enzo.mingozzi@unipi.it).

G. Tanganelli, C. Vallati and E. Mingozzi, "Edge-Centric Distributed Discovery and Access in the Internet of Things," in IEEE Internet of Things Journal, vol. 5, no. 1, pp. 425-438, Feb. 2018. doi: 10.1109/JIOT.2017.2767381

domains.

In this work, we propose an Edge-centric distributed solution to federate different IoT gateways deployed in Fog nodes close to IoT domains. The goal is to provide applications with a common service for global discovery and access of IoT resources in the federated domain, irrespectively of their location. In order to preserve the distributed nature of the federation and continue ensuring low-latency interactions, the service is realized by IoT gateways at the Fog Layer through a structured Peer-to-Peer (P2P) overlay, implemented by means of a Distributed Hash Table (DHT), where information about all available IoT resources is stored for global lookup. P2P architectures have been successfully demonstrated to be both scalable and reliable (see, e.g., [12]), as well as resilient to flash-crowd effects and Denial-of-Service (DoS) attacks due to their fully distributed nature. For this reason, P2P has been already proposed and demonstrated to be effective also in industrial environments, e.g. in [13-14].

The overall concept is instantiated leveraging state-of-art technologies for IoT discovery and access. Specifically, the CoRE RD [3] and the CoAP protocol [15] have been adopted to implement the standard interface exposed to applications. On the other hand, for the implementation of the DHT, the eXtendible Metadata Hash Table (XMHT) has been adopted to exploit its extensible interface that resembles the CoRE Link Format commonly used to encode resource identities (URIs) along with metadata (link attributes).

In order to evaluate the feasibility of the proposed solution the latter has been fully implemented and validated by means of a small-scale prototype realized using off-the-shelf hardware. In addition, an extensive evaluation has been carried out by means of emulation to assess its performance in large-scale deployments as compared to a centralized Cloud-based approach. Results showed that a distributed Edge-centric solution successfully provides better performance in terms of latency, but may obviously suffer by the reduced available processing and storage capacity with respect to a Cloud infrastructure.

The rest of the paper is organized as follows. In Section II an overview of the related work is provided. Section III provides a brief overview of the enabling technologies for the proposed solution, which is described in detail in Section IV. In Section V the solution is validated by means of a small scale prototype, while Section VI presents a large scale performance evaluation by emulation. Eventually, in Section VII conclusions are drawn.

II. Related Work

Access and discovery are two basic functionalities essential to IoT applications. For this reason, Cloud-based IoT platforms, such as OpenIoT [16], or other commercial solutions [17], usually provide these basic services as part of their core functionalities. In order to foster interoperability, several standardization activities have been carried out to define a common interface to such services. In this context, it is particularly relevant the work carried out within the IETF CoRE Working Group that defined the CoAP protocol for RESTful access, and the Resource Directory (RD) interface for discovery. RD, described in details in Section III.B, defines a standard CoAP-based interface for a centralized lookup server.

An alternative, also specified by IETF, is DNS Service Discovery (DNS-SD) [18], a modified version of the well-known DNS protocol tailored for IoT applications. DNS-SD defines a communication protocol to interact with a centralized repository to which clients register their resources/services. To this aim, the standard DNS interface and its messages are extended to allow the discovery of available IoT services in a network. A distributed extension of the protocol, Multicast DNS (mDNS), has been also defined [19]. In mDNS, applications discover services available in the local network through multicast messages. Such solution, however, is tailored for small-scale systems, as multicast messages arise reliability issues on a large scale.

Following the Fog/Edge Computing paradigm, other architectures are currently under standardization. The OMA organization, for instance, is currently defining the Lightweight M2M (LWM2M) standard [20], which is a client/server architecture built on top of CoAP. In the LWM2M architecture, the IoT devices (LWM2M clients) interact with a LWM2M Server usually implemented on a local gateway. LWM2M servers are responsible to offer access and discovery to external applications. Even though LWM2M is based on a distributed architecture, the current specification does not consider the communication between LWM2M Servers to realize a federation of LWM2M Servers. Another architecture currently under development is the OneM2M standard [4]. The OneM2M standard is based on a distributed layered architecture composed of interconnected nodes. Each node can implement different functionalities, among them the Common Service Entity (CSE) can be implemented by nodes to expose a common interface to applications to access the IoT devices connected to the OneM2M network. Different CSE nodes can be interconnected to implement inter-domain access. However, this can be done only in a strictly hierarchical fashion.

Methodologies for node federation to implement distributed systems has been proposed in the past P2P interactions. The same popular approach based on DHT overlay networks has been adopted also in the IoT context. In [21], for instance, the authors propose a P2P architecture specifically designed for IoT applications. The work tackles only the discovery problem proposing a two-tier DHT overlay implemented by the gateways for each IoT network.

To the best of our knowledge, the only work that considers both access and discovery is [22]. The authors propose a P2P federation of gateways, relying on CoAP for communication. The architecture exploits a decentralized P2P rendez-vous service among the nodes for discovery. The proposed approach, however, is cumbersome due to the intensive negotiation and synchronization procedures that are required between nodes.

Although the approaches proposed in [21] and [22] have similarities with the solution proposed in this paper, they do not offer a standard interface for applications. This limitation requires application to implement a custom interface for discovery and access, for example to interact directly with the interface exposed by the DHT.

III. Enabling technologies

The architecture proposed in this work leverages some basic functional components in order to build a distributed overlay of gateways for global IoT resource discovery and access. Before delving into the details of our proposal, we provide a brief overview of the state-of-art technologies that we embraced to implement such components. In particular, as far as resource discovery and (mediated) access is concerned, we capitalize on standard protocols and services already and/or being specified by the IETF CoRE Working Group, i.e., Resource Directory (RD) [3] and the CoAP protocol [15], respectively. It is however worth highlighting that alternative technologies could be used as well, such as OMA LWM2M [20], the OMA NGSI 9/10 Context Management interfaces (adopted by FIWARE *IoT Discovery* and *IoT Broker* components) [5], or oneM2M [4], just to mention a few. In fact, the proposed approach is sufficiently general and independent of the discovery and access interfaces that are actually adopted.

A. Constrained Application Protocol (CoAP)

CoAP is a Web transfer protocol specified by the IETF CoRE Working Group optimized for resource-constrained devices [15]. CoAP implements a request/response model and provides a RESTful interface to access resources hosted by an origin server running on the constrained IoT device. A CoAP resource, whose state is usually related to some physical sensor/actuator, is made available under a Uniform Resource Identifier (URI); clients access the resource state using the four canonical REST methods: GET, PUT, POST and DELETE.

In addition, a recently specified extension, namely, *Observing Resources* [23], allows a CoAP client to also register its interest in the state of a resource, and then to start receiving asynchronous notifications from the server whenever the observed resource state changes. This feature enables remote monitoring of IoT resources in a very efficient manner: it is therefore a design requirement to support observation in our proposed architecture.

Finally, CoAP allows the deployment of one or more proxies as intermediaries between clients and servers. In particular, a *reverse-proxy* is by definition transparent to clients, and stands in for one or multiple origin servers, thus acting as a common broker to access dispersed IoT resources. A reverse-proxy improves efficiency (e.g., by implementing caching to support intermittently available devices) and scalability: multiple observe relationships for the same resource, especially if hosted by a very constrained device, can be easily managed if they are actually established with a more powerful proxy on behalf of the server [24]. We exploit a reverse-proxy in IoT gateways for resource access brokering.

B. CoRE Link Format

The CoRE WG has also specified a standard format, namely, the Link Format [25], to describe an IoT resource by means of an identifier, i.e., a URI, plus additional metadata (*link attributes*), which are useful to characterize the functionalities provided by the resource. The Link Format is based on Web Linking [26], which is extended to include a number of attributes specifically relevant for IoT resources. Among others, it is worth mentioning the Resource Type (*rt*) and the Interface Description (*if*). The *rt* attribute is an opaque string which allows to specify some application-specific semantic information associated to the resource, i.e., its type; and the *if* attribute is exploited to provide a name or URI indicating a specific interface definition used to interact with the target resource.

C. Resource Directory (RD)

The main objective of a resource discovery mechanism is to return resource URIs as a result of a lookup operation, complemented by additional metadata about those resources, and possible further link relations. The *Resource Directory* (RD) is a new entity under definition by the CoRE WG to allow discovery of resources [3]. The RD is a centralized directory that hosts descriptions of resources hosted by other dispersed servers, and allows lookup to be performed for those resources. To this aim, the RD implements a set of REST interfaces (*i*) for servers to discover the RD and register their set of resources; and (*ii*) for clients to discover resources by performing constrained lookup operations[1].

The *registration* interface allows a CoAP server (*endpoint*, according to RD terminology) to register its resources by providing a corresponding description using the Link Format. On the other hand, the *lookup* interface allows a client to discover registered resources and retrieve their Link Format description. In the simplest case, lookup can be done by specifying a given URI to discover if a specific resource is available and retrieve all the related metadata. More interestingly, one attribute and a corresponding value of interest can be specified instead of an URI; the RD then returns the list of all resources with a matching attribute value. The latter feature allows discovering resources by type or interface. It is therefore an important requirement to support this extended lookup operation also by the distributed directory implemented by the IoT gateway overlay proposed in this work.

IV. Edge-centric discovery and access

In this section, we describe the proposed solution for Edge-centric distributed IoT resource discovery and access.

We assume there are multiple independent IoT *domains*, each one being a logical grouping of devices that host IoT resources accessible through CoAP. How exactly the boundary extent of a domain is identified depends on the use case, but is not particularly relevant for the purpose of this work. As an example, it may be related to being part of the same vertical system, or it may be determined by some specific physical/geographic boundaries (e.g., a building or a neighborhood), or, more simply, it follows from the deployment of devices in the same

[1] The RD specification also allows to register and lookup entities other than resources, i.e., endpoints, groups of endpoints (for the purpose of group communication), and domains. These functionalities are not specifically relevant for the purpose of this work, and therefore are not considered hereafter.

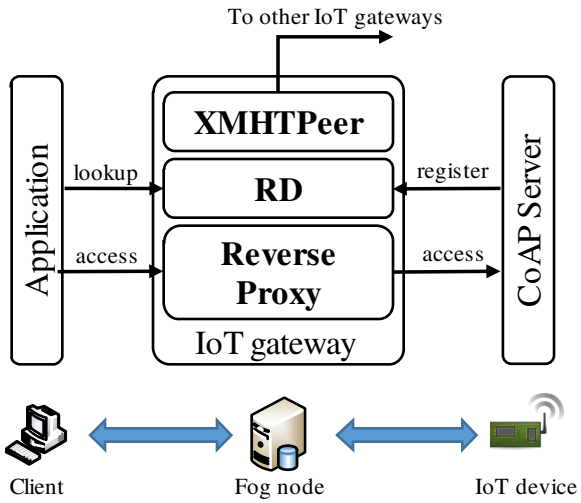


Fig. 1. IoT gateway functional view.

access network. As such, the size of an IoT domain may also span several scales, from a few devices in a smart home, to hundreds of sensors in an environmental monitoring system. We nevertheless assume that all IoT devices in the same domain are managed by the same IoT *gateway*. The latter is a *logical* entity that represents the point of contact to which an IoT *client application* is bound for discovering and accessing IoT resources in the managed domain. To this aim, the gateway implements the following functionalities accessible via dedicated interfaces:

- *registration*, which allows devices in the IoT domain to push the information about their hosted resources (i.e., URI and related metadata) into the gateway resource database. We assume that the gateway implements internally a CoRE RD, and resource registration is performed via the RD *registration* interface;
- *discovery*, which allows IoT clients to lookup information about resources hosted by devices in the managed IoT domain. IoT clients can be located anywhere, including on devices in the same IoT domain (e.g., for Machine-to-Machine applications). We assume that this is also implemented by the internal RD which enables discovery via its *lookup* interface;
- *access*, finally, which makes the gateway act as an intermediary broker between clients and all CoAP endpoints in the IoT domain for the purpose of accessing IoT resources. As such, the gateway implements a CoAP reverse-proxy, as defined in [15], coupled with the internal RD supporting discovery: URIs returned to clients as a result of lookup operations are translated consistently from registered URIs so that all CoAP requests are sent by clients to the intermediary gateway rather than directly to IoT devices.

The IoT gateway functional architecture is illustrated in Fig. 1, where, in addition to the already mentioned RD and Proxy components, a third component, namely, the XMHTPeer, is also included and will be introduced later in this section.

We further assume that IoT gateways are deployed in servers located close to IoT devices and away from the centralized Cloud, according to the Edge/Fog Computing paradigm. There are many advantages in deploying IoT gateways at the Edge. In particular, proximity makes communication more efficient as compared to using far-away centralized intermediaries; and low-latency autonomic control is enabled by exploiting co-located applications and services in the same IoT domain. In addition, the deployment of IoT gateways at the Edge offers better support for security and privacy, as they allow to store and manage sensitive information locally to the domain, thus avoiding their transmission over unsecure networks, or their storage into third-party Cloud infrastructures [27]. This is in fact one of the service scenarios explicitly considered by ETSI for Mobile Edge Computing [28].

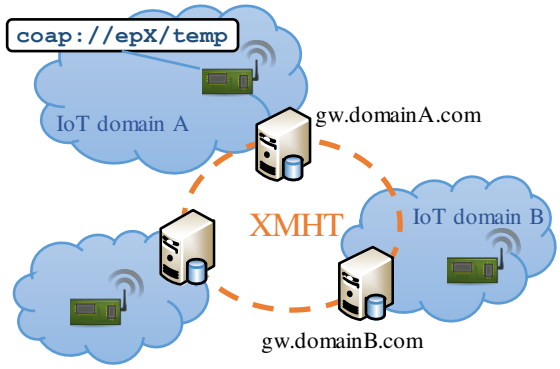


Fig. 2. Architecture overview.

In addition, gateway interfaces are bound to CoAP in order to enable the deployment also in constrained devices. Therefore, depending on the IoT domain size and use case, the IoT gateway could also be running on networked devices deployed at the Edge other than servers in micro/nano Edge data centers. Examples are home routers, set-top boxes, or even IoT devices with constrained yet sufficient computational and storage capabilities like, e.g., a smart device at home. Hereafter, we generically use the term *Fog node* to refer to the edge device where the IoT gateway is deployed [29].

Considering a scenario where multiple IoT domains operate independently of each other, we aim at providing a solution for *federating* the local services available in each domain, and build on top of them a *global* service for discovery and access, which allows client applications to discover and access any IoT resource, irrespectively of its actual location in a given domain. The objective is to keep the advantages of the original Edge-centric deployment, and therefore to avoid resorting to any higher-level functional entity, e.g., a "super" resource directory, deployed in the centralized Cloud. We then follow a loosely coupled approach, according to which IoT gateways cooperate with each other in a fully distributed manner to implement global discovery and access across multiple IoT domains.

The proposed architecture is illustrated in Fig. 2. As far as global discovery is concerned, P2P overlay networks are clearly the most suitable solution, since they naturally give a federated global perspective of different resources distributed across multiple domains, and provide many benefits in terms of efficiency, scalability, self-configuration, and robustness [30]. In the proposed solution, IoT gateways participate in a structured P2P overlay, implemented by means of a DHT where information about IoT resources is stored for global lookup. In particular,

we adopted a specific DHT implementation called *eXtensible Metadata Hash Table* (XMHT) [31], since it has unique extended features that allow to implement global lookup operations aligned to the RD lookup interface. The gateway's logical component, which is in charge of maintaining the P2P overlay, is the already mentioned XMHTPeer. The XMHTPeer is internally exploited by the RD in order to discover resources at the global level.

On the other hand, for the purpose of IoT resource access, we assume that each gateway acts as a proxy also for remote resources, i.e., resources managed by other gateways in foreign domains. In this manner, the distributed nature of the system is fully hidden to client applications, which access all IoT resources across different domains through a single point of contact, i.e., the gateway they are bound to. Such a communication approach provides many advantages. First, multiple caching points are enabled, both close to the client application and the CoAP server hosting the resource, therefore communication overhead and response times are likely to be reduced. Secondly, management of non-functional aspects, such as trustiness and QoS, is much more scalable, since it can be decoupled into provisioning rules for establishing relationships, e.g., trust or QoS agreements, between a client application and its bound domain, or between peering domains, thus avoiding also managing complex interactions between a domain and all foreign applications. Finally, extended logic can be implemented in the client's gateway to exploit the availability of foreign resources, as it will be further discussed below.

We detail in the following the proposed solution.

A. *Global discovery*

Global discovery across multiple domains is supported by a distributed global RD on top of the local RDs. To implement the latter, we make use of XMHT, a Pastry-like DHT implementation originally proposed for IoT discovery in [31][2]. XMHT allows to store and retrieve <*key,value*> pairs which are distributed among IoT gateways. The *key* is a hash of an opaque string (as in many DHT implementations), while the *value* is a *collection* of XMHT specific data structures called *enriched locators* (*e-locators* for short). Each *e-locator* includes a target URI, as well as additional metadata information in the form of name/value pairs. As such, it perfectly matches a link description according to the CoRE Link Format. The latter is therefore used, instead of the original format in [31], to avoid the need for any transcoding in the proposed solution.

XMHT allows to access the DHT through the following three methods: (1) *append*(*key,e-locator*), which adds the specified *e-locator* to the collection associated to the provided *key* – if there is no such *key*, a new pair is created and the collection is initialized with the provided *e-locator*; (2) *get*(*key*), which retrieves the collection of *e-locators* associated to the provided *key*; and, finally, (3) *unpublish*(*key*), and *unpublish*(*key,e-locator*), which are used to remove a *key* (and all associated *e-locators*), or an *e-locator* from the collection associated to the specified *key*, respectively. Thanks to this design, XMHT allows to group multiple IoT resource information under one single key, and to dynamically add and remove resources to/from this group. This feature is essential to our solution, and mainly motivates our choice for the DHT implementation. In fact, as it will be illustrated more in detail in the following, the description of all resources matching a given attribute-based lookup operation will be stored with XMHT in a collection associated to a key, whose value depends on the specific lookup parameters. In this manner, lookup of resources by type or interface is extended straightforwardly to the global scope, an essential requirement as highlighted in Section III.C.

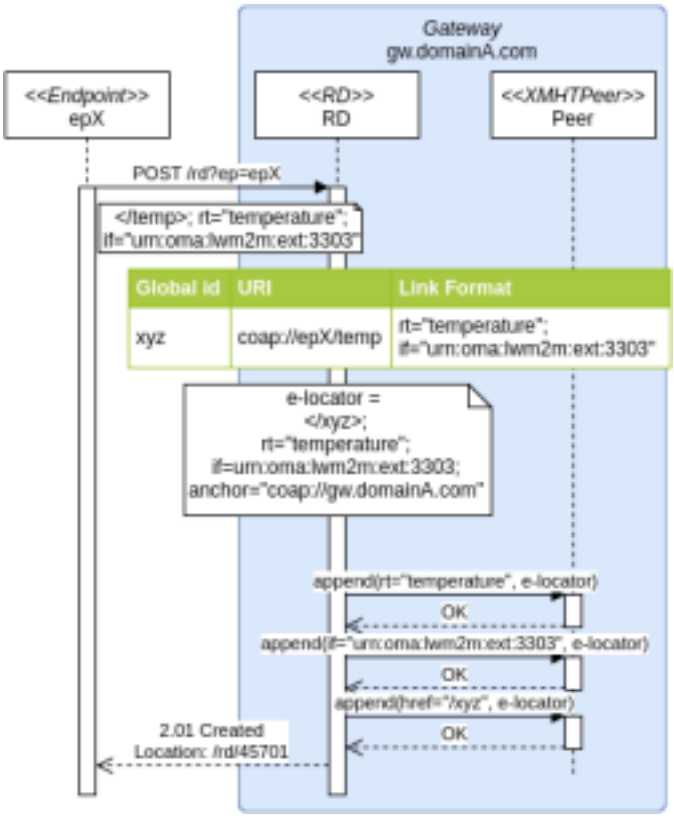


Fig. 3. Resource registration.

We now illustrate how the registration and discovery functions provided by the IoT gateway are extended to enable global discovery.

1) *Registration*

The registration operation is illustrated with the help of the example in Fig. 3. Following the RD specification [3], in order to register a new resource, an endpoint, say *epX*, in domain *A* sends a POST message to the RD in the corresponding IoT gateway, say *gw.domainA.com*, including the link description of the resource to be registered. In the example, the description includes a relative reference (URI-Reference [33]) to the resource (*</temp>*) and two link attributes with their respective values.

The RD then performs the following actions for each of the registered resources:

1. It generates a stable identifier with a global scope. This

[2] Although the XMHT implementation we adopt is based on Pastry, the proposed solution does not rely on any of Pastry specific features.

can be easily implemented, e.g., by calculating a hash function depending on the gateway name, the endpoint name, and the URI-Reference included in the description of the link. In the example, the resulting identifier is *xyz*=hash(*gw.domainA.com*:*epX*:*/temp*);

2. It stores the description of the registered resource in its database for local discovery, linked to the corresponding generated global id. This allows the local reverse-proxy to resolve URIs at access time. In the example, the received Link Format description of the link is stored, and the relative reference */xyz* is locally associated to the URI *coap://epX/temp*;
3. It creates a global link description of the resource by modifying the received one as follows: 1) the original URI-Reference is replaced with the newly created relative reference based on the global id; and 2) the *anchor* attribute is added, whose value is set to the URI identifying the IoT gateway function set. According to the CoRE Link Format specification, this will be used as the base URI for the relative reference to determine the target URI conveyed by the link description. In the example, the global link description is *</xyz>;rt="temperature";if="urn:oma:lwm2m:ext:3303";anchor="coap://gw.domainA.com"*. The latter defines the *e-locator* that will be stored in the DHT.

The RD completes the registration operation by pushing the registered resource information to XMHT for global lookup. Several *append*() operations may be performed. In particular, for each attribute on which a global lookup operation is allowed, if the attribute is included in the *e-locator*, then the latter is added to the corresponding collection invoking the *append*() method. The provided key is determined by the combined attribute name and value pair included in the *e-locator*. In the example in Fig. 3, this is therefore done for both the *rt* and *if* attributes, with key values *rt="temperature"* and *if="urn:oma:lwm2m:ext:3303"*, respectively. Finally, the *append*() method is invoked also to store the e-locator alone associated to its global id; in order to keep a key structure consistent with the cases above, the provided key in this case is *href="/xyz"*.

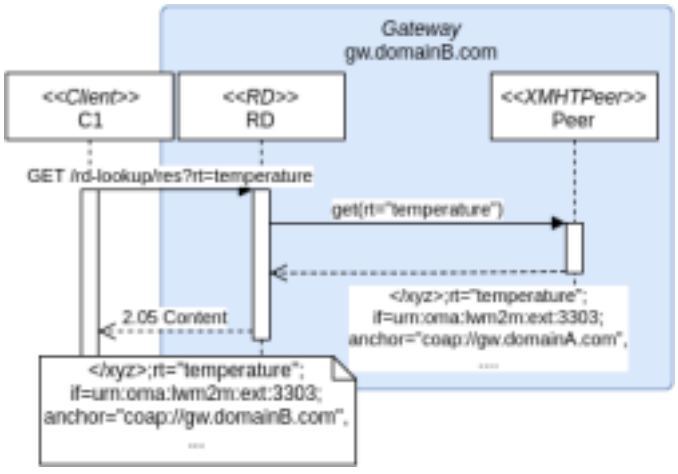


Fig. 4. Resource discovery.

[3] Should this not be a requirement, i.e., client applications are allowed to access resources through foreign gateways, is suffices to skip the anchor substitution.

2) Discovery

Once a resource is registered according to the procedure above, it can be discovered by any client application bound to an IoT gateway participating in the overlay. Global discovery operation is illustrated with the example in Fig. 4.

A client application discovers resources from any domain by using the *lookup* interface exposed by the RD in its point-of-contact IoT gateway. The RD retrieves this information from XMHT by determining the appropriate key based on the client query parameters. In the example in Fig. 4, the client queries its gateway, say *gw.domainB.com*, for all resources of type "temperature" registered in any of the federated IoT domains. By construction, this information is stored by XMHT as a collection of *e-locators* associated to the key *rt="temperature"*, which is then obtained by issuing a *get*(*rt="temperature"*) command.

For each *e-locator* in the retrieved collection, i.e., a global link description of a resource, RD performs the following two actions before responding to the client:

1. If the resource is not registered in the RD domain, i.e., it is a "foreign" resource, it temporary caches a copy of the link description in its local database. This is not needed to ensure correct operations but is rather done for optimization purposes. In fact, assuming that the client application will likely access one or more of the discovered resources through this gateway after lookup, the RD is then able to map the required resources to their target URIs (specified by the *anchor* attribute value) without the need to query XMHT again. In the example, the client performs a GET operation on the resource */xyz*, based on the cached information the RD can immediately instruct the local proxy to forward the request to the remote proxy *gw.domainA.com*. Cache management of 'foreign' resource information is however outside the scope of this work, and will not be discussed any further.
2. It substitutes the anchor attribute value with the URI identifying its own function set, in order to enforce the client application to access any resource, including foreign ones, through the local proxy. This is done to hide the distributed location of resources to applications, which is a design choice of the proposed solution[3]. In the example, the anchor value in the link description returned to the client application is then "*coap://gw.domainB.com*".

The collection of modified *e-locators* is then returned to the client application as a result of the discovery operation.

B. Global access

Seamless access to resources is provided to a client application by the Proxy component in its IoT gateway. As explained in previous sections, only URIs based on global identifiers are returned to applications as a result of a discovery operation. Therefore, incoming CoAP requests from client applications

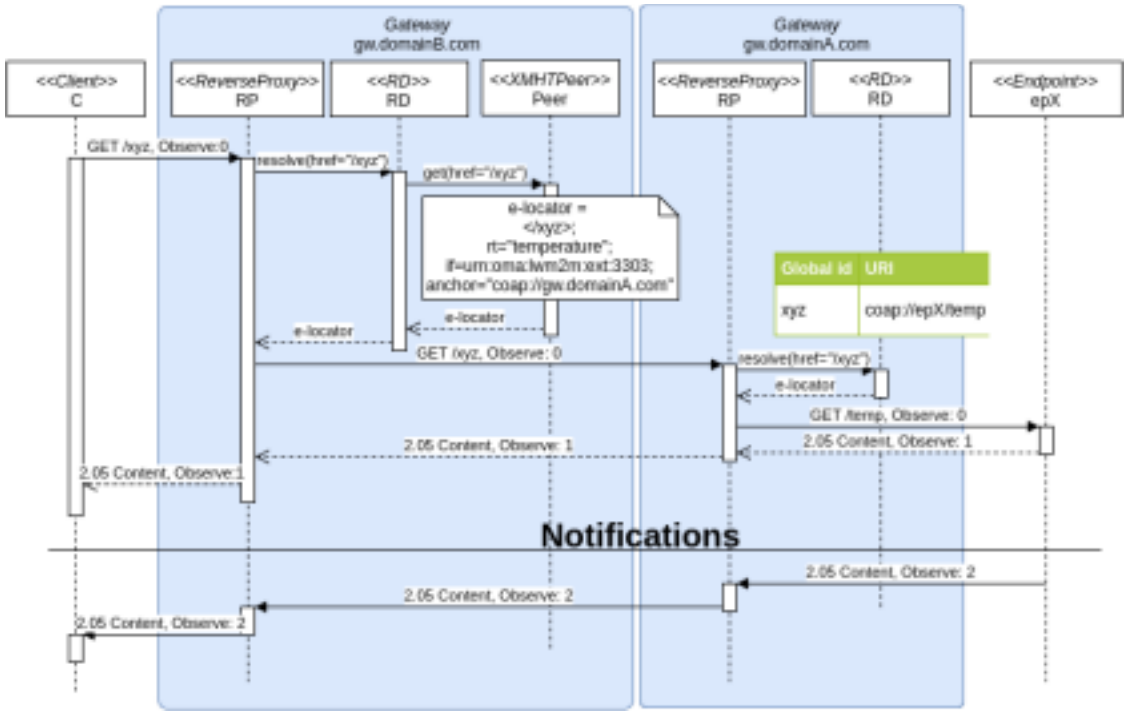


Fig. 5. Global access with observing.

only target such *global* URIs. When the Proxy receives a request from a client, it therefore needs to resolve the global URI into an URI useful for concretely accessing the resource. This is done with the help of the local RD. Only one of the following cases is possible. If the resource is hosted by an endpoint in the local domain, the RD has internally stored the mapping between the global id and the registered resource description (including the target URI), therefore URI resolution is straightforward. Otherwise, the resource is hosted in a foreign domain. In this case, a copy of its link description could be cached locally as a result of a recent discovery operation. URI resolution is again straightforward: the target URI is obtained by combining the base URI specified by the *anchor* attribute and the relative reference based on the global id. Finally, if the link description is not locally available, XMHT is queried to get the link description of the requested resource.

It is worth highlighting that communication through CoAP is adopted not only between (reverse) proxies and endpoints hosting IoT resources, but also between client and proxies, and between proxies in different gateways. While the former is a reasonable solution to cope with IoT devices that might be constrained along many dimensions (e.g., memory, computation, communication through low power and lossy network, energy), communication between applications and proxies could be also implemented by HTTP. However, besides adding a complexity due to protocol translation, this would prevent the overlay to unleash the full potential of the Observing Resources extension of CoAP on an end-to-end basis [23]. In our solution, observe relationships are managed by proxies, which allow to establish multiple observe relationships with the same resource in a scalable manner [34].

We illustrate in more detail in the following the global access procedure (with observing) with the help of the example in Fig. 5. A client application sends a GET request for the resource */xyz* to the Proxy in the IoT gateway *gw.domainB.com* including the observe option. The Proxy asks the local RD to resolve the relative reference to a target URI to forward the request to. In this example, the actual resource with global id *xyz* is located in a foreign domain, and we assume its link descriptor is not locally cached. The RD therefore queries XMHT by executing a *get*(*href="/xyz"*) method, which returns the stored *e-locator* including the CoRE link format global description of the resource. This is passed in turn to the Proxy, which checks if the resource is observable, and then forwards the GET request to the URI *coap://gw.domainA.com/xyz*, which addresses the Proxy in the IoT gateway managing the domain where the requested resource is hosted. Like the operation in domain B gateway, the proxy in domain A will query its local RD to resolve the reference. Since the resource is hosted in domain A, the mapping to the origin URI *coap://epX/temp* is now stored internally in the RD, and resolution is immediate without the need to access the DHT. Finally, the request is forwarded to the origin CoAP server on endpoint *epX*, and, if successful, notification messages start being delivered asynchronously to the remote client application through the two intermediate proxies.

C. Extensions

The proposed architecture is easily extendible with the implementation of new features that can be offered to applications. For example, the union of the different connected IoT *domains*, in a transparent way, may result in having multiple similar resources. One opportunity raised by this abundance of resources, for instance, could be the exploitation of *equivalent resources*, resources that can provide the same information/service. Let us suppose that an application is interested in the temperature in a certain area that is covered by two different IoT *domains* and in

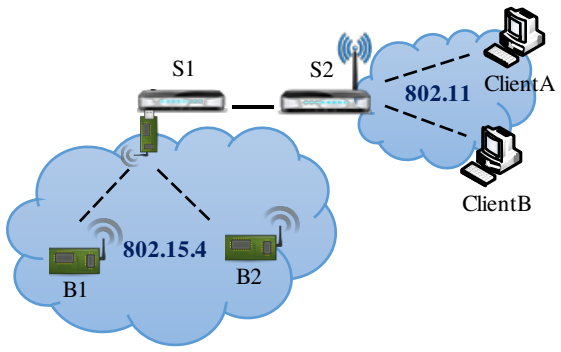


Fig. 6: Testbed.

both a temperature sensor is available. As far as the context of a specific temperature request in the area is concerned, the two temperature sensors are *equivalent*, and any of them can be exploited interchangeably, e.g. to enforce load balancing or power saving policies. Another opportunity is to aggregate different resources to obtain composite data, i.e., data derived from the composition of information coming from different sensors. An example of such *aggregated resources* could be a resource that aggregates the temperature values of a certain area to produce the average value. The creation of such *virtual resources* can be easily implemented in the proposed solution. A gateway that receives a discovery invocation can exploit the XMHT overlay to retrieve the list of resources matching the query and before replying back to the client, in case equivalent or aggregated resources are available, new *virtual resources* can be created. How this can be actually implemented, e.g., through semantic-based descriptions, goes however beyond the scope of this work.

V. Validation

In order to validate the proposed architecture and demonstrate its feasibility, a small-scale prototype using commercial off-the-shelf hardware has been implemented. The testbed architecture is illustrated in Fig. 6. Two IoT gateways are deployed to connect two IoT domains, one hosting IoT resources and another one hosting client applications. Fog nodes hosting the IoT gateways are realized by two Soekris boards net5501-60[4] (respectively S1 and S2), a popular embedded board that can run a fully featured Linux operating system. Each board connects to its local network - an IEEE 802.15.4 sensor network for S1 and a Wi-Fi network for S2 - and to an Ethernet network that emulates the backbone. The sensor network includes two Zolertia Z1 boards (respectively B1 and B2), a popular sensor platform that support the execution of the Contiki OS[5], which natively supports 6LoWPAN [2], RPL [35], and CoAP. Given the lack of an IEEE 802.15.4 transceiver on the Soekris boards, a Zolertia Z1 board[6] has been programmed to behave as a transceiver and attached to S1 to enable communication with the sensor network by means of a tunslip[7] interface. S2, instead, exploits its Wi-Fi transceiver to connect to the Wi-Fi network in which the CoAP client applications run.

TABLE I
DELAY FOR DIFFERENT COAP REQUESTS.

Request	Client A	Client B
GET */water*	53.8 ms ± 2.50 us	40.4 ms ± 2.55 us
GET */switch*	53.5 ms ± 3.05 us	40.4 ms ± 2.55 us
GET */light*	53.3 ms ± 2.50 us	42.3ms ± 3.30 us

The proposed solution for discovery and access is implemented in C++. The two boards are programmed as CoAP servers that provide measurements from physical sensors (a light sensor and a water consumption sensor) and access to an actuator (a light switch) through CoAP resources (two on B1 and one on B2). The standard CoAP implementation has been modified to introduce the initial RD registration phase to register all the exposed resources to the local IoT gateway.

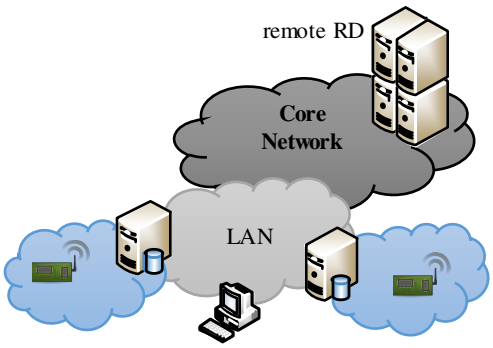
To validate the proposed solution, two different client applications are deployed in the Wi-Fi network, Client A and Client B, respectively. In order to measure the overhead introduced by the discovery operations, Client A is configured to access CoAP resources through the proposed discovery and access solution. Client B, instead, is configured to avoid discovery and perform only access through the Reverse Proxy interface, i.e. the client does not exploit the discovery functionality offered by the IoT gateway through XMHT, but the required resource is assumed to be known a priori. Each application client issues 300 requests per resource during each experiment. Experimental results are reported in Table I, that shows the average response delay defined as the time between the first CoAP request and the reception of the data. In order to ensure statistical soundness, 10 different runs for both the scenarios with Client A and Client B are executed and the average value reported with the 95% confidence interval. As expected, Client A shows a negligible overhead compared to Client B, in particular considering that the Sokeris boards are constrained embedded systems with limited resources that can represent a bottleneck for the DHT operations involving multiple TCP connections among gateways.

VI. Performance Evaluation

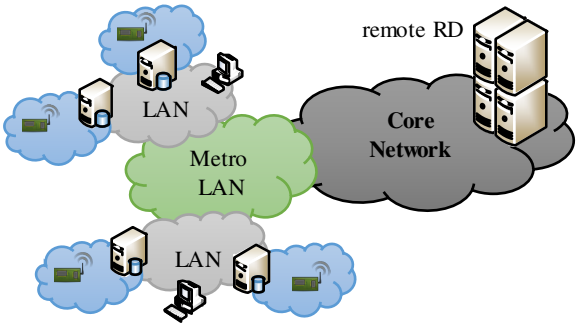
Results obtained from the prototype implementation demonstrated the feasibility of the proposed approach and the limited overhead introduced. In this section, we go further and analyze the proposed approach on a large scale. Carrying out such analysis using real devices is however not practical, as it requires the deployment of a large number of Fog nodes across different geographically distributed sites. For this reason, we decided to exploit an emulative approach to evaluate the performance of a large-scale network of Fog nodes in an easy-deployable and controllable manner with a high level of accuracy.

In the following, we first introduce the scenarios considered in our experiments, then we illustrate how experiments have been conducted, and finally we present the evaluation results.

[4] http://soekris.com/products/net5501.html
[5] http://www.contiki-os.org/
[6] http://zolertia.sourceforge.net/wiki/index.php/Z1
[7] TUN is a virtual network kernel device simulating a network layer device. The device is emulated at Layer 3 for routing purposes.



(a) *Local* scenario.



(b) *Metro* scenario.

Fig. 7: Network configurations.

TABLE II
LINK LATENCY PDF PARAMETERS

Link type	Probability density function
LAN	Normal; $\mu = 2ms, \sigma = 1ms$
Metro	Normal; $\mu = 5ms, \sigma = 1ms$
Core *Cloud-A*	Normal; $\mu = 40ms, \sigma = 5ms$
Core *Cloud-B*	Normal; $\mu = 100ms, \sigma = 5ms$

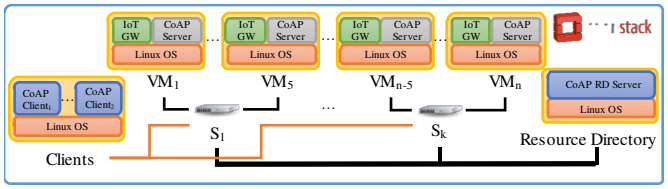


Fig. 8. Openstack experimental deployment.

A. *Evaluation Scenarios*

The goal of this evaluation is to compare the proposed Edge-centric distributed solution for global discovery and access to a centralized Cloud-based approach, verifying also its scalability under real conditions. To this aim, in addition to the proposed solution described in Section IV (*Edge* case hereafter for short), we consider an alternative solution where global discovery is implemented by means of a central RD service which is common to all IoT domains and deployed in a Cloud data center. In this case, referred to as *Cloud* case hereafter, all IoT gateways register their IoT resources to the remote RD, and client applications look up the remote RD instead of the local IoT gateway to discover resources. Moreover, the application directly access the discovered IoT resource through its managing IoT gateway as the only CoAP intermediary.

We design a set of experiments considering a variable number of IoT domains, each one comprising a set of IoT resources managed by one gateway deployed in a Fog node close to the IoT domain. All Fog nodes are connected to each other through a backhaul network that ensures low-latency communication. Two different network configuration scenarios are considered, as depicted in Fig. 7. The first network configuration (Fig. 7.a) represents a *local backhaul network* scenario (named *local* scenario hereafter) in which all Fog nodes are attached to the same LAN. The second network configuration (Fig. 7.b) represents instead a *metropolitan backhaul network* scenario (named *metro* scenario hereafter) in which Fog nodes are grouped into different LANs (five nodes per LAN) that are connected to each other through a Metro LAN. In the *Cloud* case, the remote RD is assumed to be located in a far data center, which is reachable by all Fog nodes and client applications through a core Internet link characterized by a long-distance latency.

All network latencies are random variables drawn from a Normal probability distribution. Table II reports the different mean and variance latency values (one-way) assumed for each type of network link. Two different values are used for the core Internet link in the *Cloud* case, corresponding to two typical scenarios for real long-distance latency values measured by a network operator[8]. We refer hereafter to the two corresponding cases as *Cloud-A* and *Cloud-B*, respectively.

In both scenarios, requests for discovery and access are generated according to a global Poisson arrival process. Different arrival rates are considered. Each request is randomly associated to an IoT resource in a uniform manner, irrespectively of its corresponding IoT domain. Moreover, in the *Edge* case, the request is also bound randomly to a local IoT gateway for request dispatching.

Two main performance metrics are collected. The *lookup delay* is defined as the time between the reception of a request by either the IoT gateway or the central RD, depending on the case, and the completion of the discovery process. The *access delay* is defined instead as the time between the completion of the discovery process and the completion of the access procedure. For each experiment, the mean value of each metric is estimated along with its 95% confidence interval. Every experiment has a variable duration and terminates when an overall number of 100000 requests are issued.

B. *Experimental methodology*

In order to carry out the experiments for the scenarios described in the previous section, we exploited the OpenStack[9] platform for emulation. OpenStack is a popular open-source virtualization framework that allows rapid deployment of *Virtual Machines* (VMs) with custom architecture and available resources. A set of VMs are deployed to emulate a distributed architecture of Fog nodes, as illustrated in Fig. 8. To this aim, an ad-hoc template has been prepared to emulate a Fog node

[8] http://www.verizonenterprise.com/about/network/latency/
[9] The OpenStack framework, https://www.openstack.org/

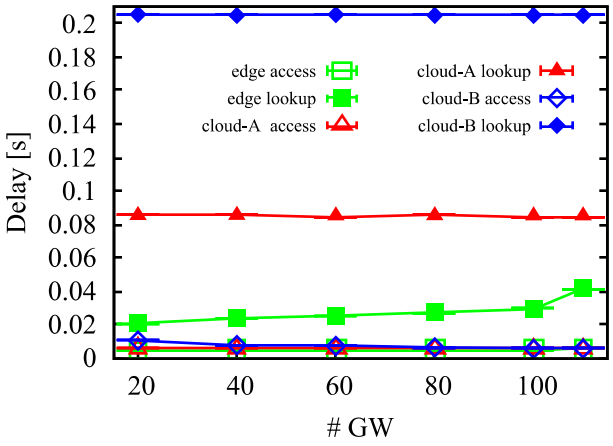


Fig. 9. Local scenario – Rate 60 [req/s].

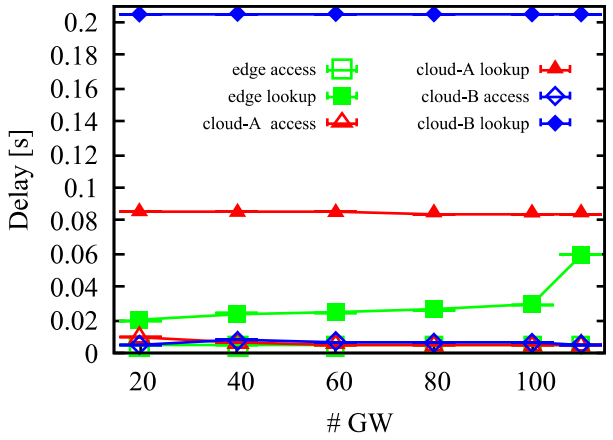


Fig. 10. Local scenario – Rate 100 [req/s].

running on an embedded device, i.e., a VM with a 32-bit computing architecture with 433 MHz CPU and 256MB of RAM (the same resources of a Soekris board) running the same OS and software of the prototype presented in Section V. In order to set the CPU speed, the *taskset*[10] Linux utility has been used to limit the VMs occupation of the host machine CPU. Without losing generality, CoAP servers are also running on the Fog node. Specifically, each Fog node hosts a CoAP server that exposes a certain number of IoT resources belonging to the same IoT domain.

The central RD is hosted on a separate VM with a 64-bit computing architecture, 4GB of RAM and two CPUs at 3.4GHz, on which a fully featured Linux OS is running. The VM hosts the central RD service that is used to emulate the *Cloud* case. RD stores the information about all IoT resources in a MySQL database. Network connections among VMs are emulated through the OpenStack Neutron module that allows the creation of virtual LANs for VMs. The emulation of heterogeneous network links is performed through *Netem*[11], a software that can introduce network delays between interfaces according to a given statistical distribution. In order to allow the communication between Fog nodes and the central RD in the *Cloud* case, an emulated Internet link between all the LANs and the VM hosting the RD is also configured.

Finally, an additional powerful VM (4GB of RAM and two CPUs at 3.4GHz) is deployed for the generation of application requests. Specifically, an application running in this VM generates CoAP requests according to a Poisson process with a configured rate, and sends them to the randomly selected gateway or the RD, depending on the case. To this aim, the VM is connected to all the LANs deployed in the scenario. In order to avoid bottlenecks in the request generation process, the application has been implemented exploiting multi-threaded programming in C++ language.

C. Evaluation Results

We consider two set of experiments. In the first set, for a given number of resources and a given request arrival rate, we vary the number of IoT domains, i.e. gateways, in order to assess the scalability of the proposed solution with respect to its distribution degree as compared to a fully centralized discovery service. On the other hand, in the second set, for a given number of resources and IoT domains, we vary the request arrival rate to evaluate the scalability with respect to offered load. We illustrate the evaluation results of the two sets in the following sections.

1) Distribution degree

For this set of experiments, we consider a fixed number of 1000 IoT resources, and we vary the number of IoT gateways between 20 and 110. IoT resources are uniformly distributed among the configured number of gateways in each experiment.

Fig. 9 shows the average access and lookup delays obtained in the *local* scenario for all cases, for a fixed arrival rate of 60 requests/s. As can be seen, the average access delay is in the order of few milliseconds, and values are pretty much the same in both the *Edge* and *Cloud* cases for any number of gateways. This means that, even though in the *Edge* case access is performed through an additional intermediary gateway as compared to *Cloud*, the corresponding processing overhead is negligible and does not affect the overall performance. On the other hand, we can observe that the access delay slightly decreases as the number of gateways increases. In fact, by increasing the number of gateways, the number of IoT resources per gateway decreases, and therefore the average load per gateway for proxying CoAP requests to origin servers decreases as well, which results in a lower processing delay per request.

Different conclusions can be drawn for the lookup delay, as it significantly differs in the *Edge* and *Cloud* cases. As expected, for both the *Cloud-A* and *Cloud-B* cases the lookup delay is significantly higher than for the *Edge* case, though independent of the number of gateways. This is mainly due to the high delay introduced by long-distance core network links that need to be traversed to reach the Cloud server, as compared to the much lower delays incurred by Fog nodes in the *Edge* case to communicate with each other over LANs only.

On the other hand, in the *Edge* case the lookup delay slowly increases as the number of gateways increases up to 100, and then shows a steep increment when the number further in-

[10] http://linuxcommand.org/man_pages/taskset1.html

[11] http://man7.org/linux/man-pages/man8/tc-netem.8.html

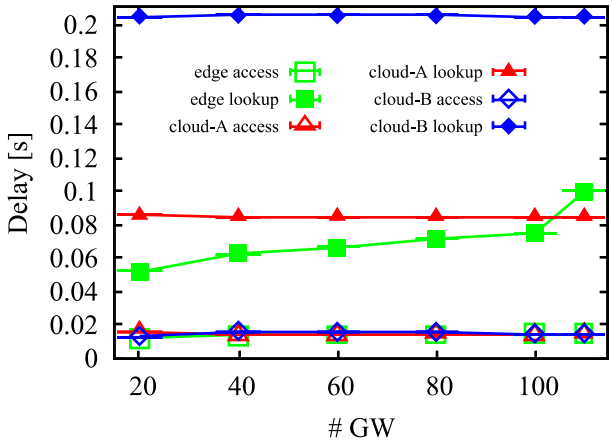


Fig. 11: Metro scenario – Rate 60 [req/s]

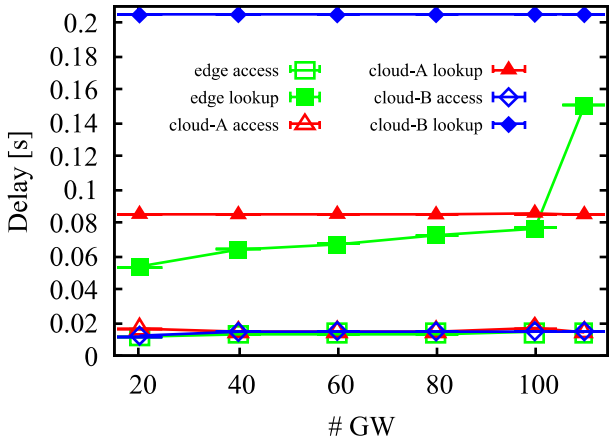


Fig. 12: Metro scenario – Rate 100 [req/s].

creases to 110. The first behavior is typical of a DHT-based implementation of a lookup service. In fact, multiple hops may be needed to lookup a resource in a DHT, and it is well known that the average number of such hops has a logarithmic increase with the number of nodes maintaining the DHT [32]. Although not directly devisable from Fig. 9, it can be numerically verified that the lookup delay actually increases logarithmically up to 100 gateways in the *Edge* case.

However, for 110 gateways the lookup delay deviates from the logarithmic increase and suddenly gets to about 60ms. After carefully analyzing this case, we discovered that, although the average load per gateway decreases as the number of gateways increases, nevertheless the amount of memory required by the XHMT implementation in use to manage an increasing number of connections actually increases as well. Therefore, considering that in these experiments Fog nodes run in constrained embedded devices, the steep increase of the lookup delay reveals that the implementation under evaluation enters congestion due to lack of memory starting from 110 gateways. Although this effect can be mitigated by a carefully optimized software implementation, this result shows a potential trade-off in adopting an Edge-centric distributed discovery solution between extending the distribution degree on one hand, and correctly provisioning adequate computation and storage resources in Fog nodes on the other hand. It is worth noting however that federating up to a hundred of IoT domains can be considered a limit case in most of the reasonable scenarios.

The same experiment has been carried out for different combinations of arrival rates (100 vs. 60 requests/s) and network configurations (*local* vs. *metro*), which all confirm the same conclusions drawn so far. In particular, Fig. 10 shows the results obtained for a request arrival rate of 100 requests/s in the *local* scenario, while Fig. 11 and Fig. 12 show the results in the case of the *metro* scenario, for a request arrival rate of 60 and 100 requests/s, respectively. The latter results show in addition how the performance in the *Edge* case also depends on network latencies at the access layer, i.e., between Fog nodes. In fact, in the *metro* scenario, the average one-way delay between two Fog nodes in different LANs is 9ms, as compared to 2ms in the *local* scenario. As can be seen, as the number of gateways increases up to 100, the average lookup delay becomes in the *metro* case very much comparable with that of the *Cloud-A* scenario, while still being clearly more efficient than the *Cloud-B* case. For 110 gateways, the lookup delay becomes even much higher in the *Edge* case with respect to the *Cloud-A* case, though however this result, as highlighted before, is very much implementation dependent for this set of experiments. As expected, we can conclude that an Edge-centric solution is as much more convenient as it can leverage proximity in terms of low-latency one-hop communications.

Finally, to get a better insight into this latter conclusion, we report in Fig. 13 and Fig. 14 the cumulative distribution function of the lookup and access delays for the *local* and *metro* scenarios, respectively, in the case of 60 IoT gateways and a request arrival rate of 100 requests/s. Specifically, the delays for the *Edge* and *Cloud-A* cases are compared. We can observe that, as far as access delay is concerned, also the delay distributions in both cases are practically overlapping. Moreover, in the *metro* scenario distributions are bimodal, since one-way delays are different depending on whether the accessed IoT resource is managed by a gateway attached to the same LAN as the client or not.

By considering lookup delays, instead, Fig. 13 and Fig. 14 confirm that the performance in the *Edge* and *Cloud-A* cases are very different. In the *Cloud-A* case, the delay has very little variability in both scenarios and is basically equal to the round-trip time to the remote RD. On the other hand, in the *Edge* case the delay is variable and actually distributed over a range of values in both scenarios. This is because the lookup operation entails a randomly variable number of hops in the P2P overlay to be completed. In addition, in the *metro* scenario the distribution is multimodal due to one-way average delays concentrated around two different values, as observed above. It is interesting to note, however, that in the *local* scenario Edge-centric discovery always outperforms the Cloud-based solution for mostly all the requests, whereas in the *metro* scenario a non-negligible percentile, i.e., 20%, of the requests in the *Edge* case experience a lookup delay worse than in the *Cloud-A* case, though the mean delay is better, and the load is far from the congestion threshold. This further confirms that, in terms of pure performance, the advantages of Edge-centric solutions with respect to a Cloud-

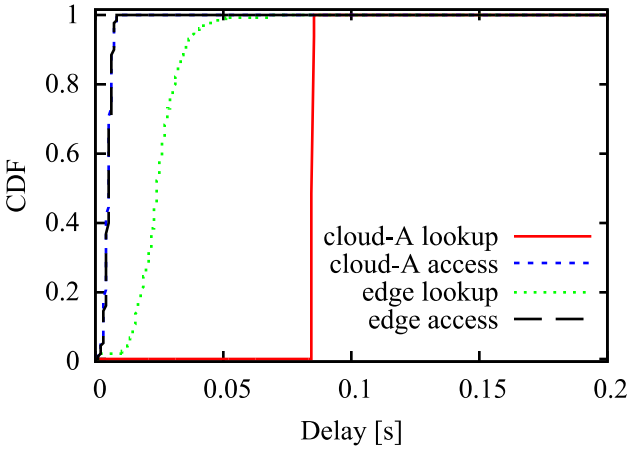


Fig. 13: Local scenario – 60 GWs - Rate 100 [req/s].

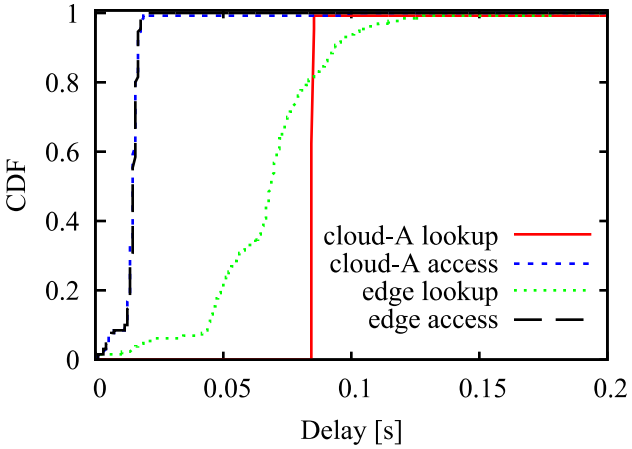


Fig. 14: Metro scenario – 60 GWs - Rate 100 [req/s].

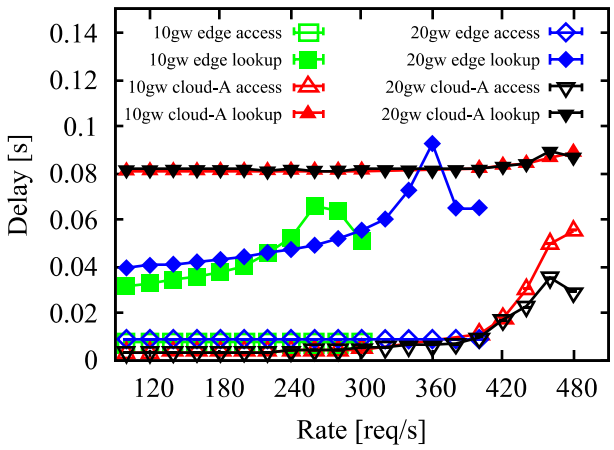


Fig. 15: Metro scenario –10 GWs and 20 GWs.

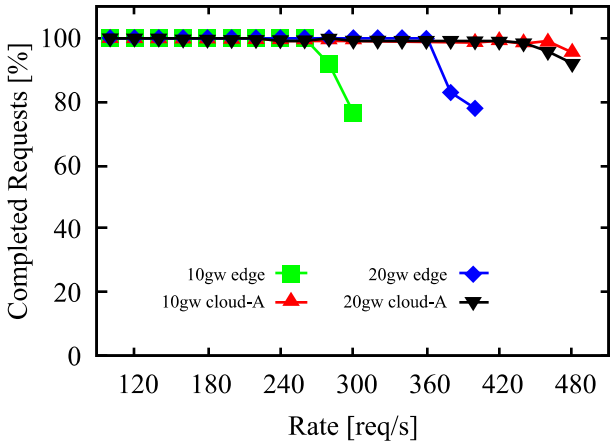


Fig. 16: Metro scenario – 10 GWs and 20 GWs.

based one very much depend on low network latencies available at the Edge of the network.

2) Offered Load

For this set of experiments, we consider a fixed number of 1000 IoT resources uniformly distributed among a given number of gateways, and we vary the request arrival rate between 100 and 480 requests/s. Without losing generality, we only report the results for the *metro* scenario, in both the *Edge* and *Cloud-A* cases, with 10 and 20 IoT gateways, respectively. Experiments with other configurations have been extensively conducted and led to similar conclusions.

Fig. 15 shows the average access and lookup delays as a function of the request arrival rate, i.e., the offered load. Since the latter spans in a range that is above the system capacity in all cases, it is worth to highlight that Fig. 15 reports results for *successfully completed requests* only, i.e., requests for which a CoAP response was successfully received from the IoT gateway managing the requested resource. In order to understand these results, it is then useful to also consider the fraction of the total number of requests that are successfully completed, which is reported as a percentage in Fig. 16.

Let us consider the *Cloud-A* case first. As can be seen, the lookup delay increases very slowly, thanks to the large processing and storage capabilities at the RD. Only at rates of about 450 request/s the delay starts increasing slightly faster, corresponding to some requests being dropped due to congestion at the RD. However, the bottleneck of the system is clearly the access to resource, whose delay steeply increases starting from 400 request/s. In fact, also in the *cloud* case access is managed by IoT gateways, which in all these experiments are deployed on embedded devices with constrained capabilities.

Interestingly, in the *Edge* case we observe the opposite behavior. In fact, since in this case discovery is also implemented by IoT gateways and is first used before access, it also first reaches congestion as the load increases. As can be seen, the lookup delay starts increasing sharply well before the access delay. The system becomes then congested at 280 requests/s and 380 requests/ rates for 10 and 20 gateways, respectively, when requests start being dropped immediately when received at IoT gateways. This also justifies why the delay decreases after reaching congestion: the more the requests dropped before entering the service, the less the number of requests that are actually successfully processed, and therefore the less the delay. On the other hand, it can be observed that, as expected, system capacity scales well with the number of gateways, i.e., the more the gateways, the more the available processing resources and then the system capacity. In fact, with 20 gateways congestion is reached at a higher rate than with 10 gateways. As already

mentioned, access is not an issue in this case: the access delay is nearly constant at all rates below the congestion limit.

Finally, by comparing the *Edge* and *Cloud-A* cases, it is confirmed that the proposed Edge-centric solution performs better than the Cloud-based one at all practical rates below the congestion limit, but it has also more limited capacity. Therefore, a trade-off need be considered. It is definitely possible to get better performance by exploiting computational capabilities at the Edge, but with a reduced overall capacity. Such limitations can be easily overcome by leveraging (unlimited) Cloud resources, but then there is a cost to pay in terms of performance since such resources are far from IoT domains.

VII. Conclusion

In this work, an Edge-centric distributed architecture to provide discovery and access services across multiple IoT domains has been proposed. The proposed solution is based on a DHT maintained by IoT gateways deployed in Fog nodes close to the IoT physical infrastructure, and leverages standard solutions including the IETF CoRE RD and CoAP protocol. The feasibility of the proposed solution, and its limited overhead, has been demonstrated through a concrete implementation tested in a small-scale prototype made of off-the-shelf hardware. Moreover, a virtual large-scale deployment of the proposed solution has been implemented through OpenStack, and different experiments have been carried out to evaluate the scalability of the proposed solution, and compare it to a Cloud-based one. Performance evaluation demonstrated the effectiveness of the proposed solution in providing lower latencies than a Cloud-based approach, and its scalability for small to medium-sized deployments.

Acknowledgments

The authors would like to thank Dr. Claudio Cicconetti for the helpful suggestions that helped to improve the quality of the paper.

Reference

[1] G. Kortuem, F. Kawsar, V. Sundramoorthy and D. Fitton, “Smart Objects as building blocks for the Internet of things,” in IEEE Internet Computing, vol. 14, no. 1, pp. 44-51, Jan.-Feb. 2010.

[2] G. Montenegro, N. Kushalnagar, J. Hui and D. Culler, “Transmission of IPv6 Packets over IEEE 802.15.4 Networks,” RFC 4944, Sept. 2007.

[3] Z. Shelby, M. Koster, C. Bormann, and P. van der Stok, “CoRE Resource Directory, Internet draft (work in progress),” IETF, Oct. 2016, draft-ietf-core-resource-directory-09.

[4] oneM2M Functional Architecture, TS-0001-V2.10, Aug. 2016.

[5] FIWARE IoT GE. [Online]. Available: https://catalogue.fiware.org/enablers/iot-broker.

[6] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. “Fog computing and its role in the internet of things,” In Proc. of the MCC workshop on Mobile Cloud computing (MCC '12). ACM, New York, NY, USA.

[7] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li and Lanyu Xu, “Edge Computing: Vision and Challenges”, IEEE Internet of Things Journal, Vol. 3, No. 5, October 2016, pp. 637-646.

[8] C. Perera, Y. Qin, J. C. Estrella, S. Reiff-Marganiec, and A. V. Vasilakos, “Fog Computing for Sustainable Smart Cities: A Survey,” ACM Comput. Surv., vol. 50, n. 3, June 2017.

[9] S. Abdelwahab, B. Hamdaoui, M. Guizani and A. Rayes, “Enabling Smart Cloud Services Through Remote Sensing: An Internet of Everything Enabler,” in IEEE Internet of Things Journal, vol. 1, no. 3, pp. 276-288, June 2014.

[10] Y. Chao Hu, M. Patel, D. Sabella, N. Sprecher and V. Young, “Mobile Edge Computing: A key technology towards 5G”. ETSI White Paper. Sept. 2015.

[11] J. Jin, J. Gubbi, S. Marusic and M. Palaniswami, “An Information Framework for Creating a Smart City Through Internet of Things,” in IEEE Internet of Things Journal, vol. 1, no. 2, pp. 112-121, Apr. 2014.

[12] S. Ioannidis and P. Marbach, “Absence of Evidence as Evidence of Absence: A Simple Mechanism for Scalable P2P Search,” in Proc. IEEE INFOCOM, Apr. 2009, pp. 576–584.

[13] B. Fabian, T. Ermakova, and C. Muller, “A Privacy-Enhanced Discovery Service for RFID-Based Product Information,” IEEE Transactions on Industrial Informatics, vol. 8, no. 3, pp. 707 –718, Aug. 2012.

[14] I. Delamer and J. Lastra, “Service-Oriented Architecture for Distributed Publish/Subscribe Middleware in Electronics Production,” IEEE Transactions on Industrial Informatics, vol. 2, no. 4, pp. 281 –294, Nov. 2006.

[15] Z. Shelby, K. Hartke, C. Bormann, “The Constrained Application Protocol (CoAP),” Internet RFC 7252, IETF, Jun. 2014.

[16] OpenIoT consortium. Open source solution for the internet of things into the Cloud. [Online]. Available: http://www.openiot.eu

[17] Xively, public Cloud for the internet of things. [Online]. Available: http://www.xively.com.

[18] S. Cheshire and M. Krochmal, “DNS-Based Service Discovery,” RFC 6763, 2013.

[19] S. Cheshire and M. Krochmal, “Multicast DNS,” RFC 6762, 2013.

[20] OMA LightweightM2M v1.0. [Online]. Available: http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases/oma-lightweightm2m-v1-0.

[21] S. Cirani et al., “A Scalable and Self-Configuring Architecture for Service Discovery in the Internet of Things,” IEEE Internet of Things Journal, vol. 1, no. 5, pp. 508-521, 2014.

[22] J. Mäenpää; J. Bolonio, and S. Loreto, “Using RELOAD and CoAP for wide area sensor and actuator networking,” EURASIP Wireless Communication Network. 2012.

[23] K. Hartke, “Observing Resources in the Constrained Application Protocol (CoAP),” Internet RFC 7641, IETF, Sept. 2015.

[24] E. Mingozzi, G. Tanganelli and C. Vallati, “CoAP Proxy Virtualization for the Web of Things,” 2014 IEEE 6th International Conference on Cloud Computing Technology and Science, Singapore, 2014, pp. 577-582.

[25] Z. Shelby, “Constrained RESTful Environments (CoRE) Link Format,” Internet RFC 6699, IETF, 2012.

[26] M. Nottingham, “Web Linking,” Internet RFC 4287, IETF, Oct. 2010.

[27] P. G. Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, “Edge-centric Computing: Vision and Challenges.” ACM Comput. Commun. Rev., vol. 45, no. 5, pp. 37–42, Sept. 2015.

[28] Mobile Edge Computing (MEC); Service Scenarios, ETSI GS MEC-IEG 004 V1.1.1, 2015.

[29] E. Marín-Tordera, X. Masip-Bruin, J. García-Almiñana, A. Jukan, G.-J. Ren, J. Zhu, “Do we all really know what a fog node is? Current trends towards an open definition, Computer Communications,” vol. 109, pp. 117-130, 2017.

[30] X. Shen, H. Yu, J. Buford, M. Akon (Eds.), “Handbook of Peer-to-Peer Networking,” Springer US, 2010.

[31] F. Andreini, F. Crisciani, C. Cicconetti, and R. Mambrini, “Context-aware location in the Internet of Things,” in Proc. IEEE GLOBECOM Workshops, Miami, FL, 2010.

[32] D. Rowstron, “Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems,” in Proc. IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), 2001.

[33] T. Berners-Lee, R. Fielding, L. Masinter, “Uniform Resource Identifier (URI): Generic Syntax,” Internet RFC 3986, IETF, Jan. 2005.

[34] G. Tanganelli, E. Mingozzi, C. Vallati, M. Kovatsch, “Efficient Proxying of CoAP Observe with Quality of Service Support,” In Proc. IEEE 3rd World Forum on Internet of Things (IEEE WF-IoT 2016), Reston (VA), USA, December 12-14, 2016.

[35] T. Winter, A. B. P. Thubert, T. Clausen, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, and J. Vasseur, “RPL: IPv6 routing protocol for low-power and lossy Networks,” Internet RFC 6550. IETF, 2012.