A Scalable and Self-Configuring Architecture for Service Discovery in the Internet of Things

Simone Cirani, Luca Davoli, Gianluigi Ferrari, Rémy Léone, Paolo Medagliani, Marco Picone, and Luca Veltri

***Abstract*—The Internet of Things (IoT) aims at connecting billions of devices in an Internet-like structure. This gigantic information exchange enables new opportunities and new forms of interactions among things and people. A crucial enabler of robust applications and easy smart objects' deployment is the availability of mechanisms that minimize (ideally, cancel) the need for external human intervention for configuration and maintenance of deployed objects. These mechanisms must also be scalable, since the number of deployed objects is expected to constantly grow in the next years. In this work, we propose a scalable and self-configuring peer-to-peer (P2P)-based architecture for large-scale IoT networks, aiming at providing automated *service and resource discovery* mechanisms, which require no human intervention for their configuration. In particular, we focus on both local and global service discovery (SD), showing how the proposed architecture allows the local and global mechanisms to successfully interact, while keeping their mutual independence (from an operational viewpoint). The effectiveness of the proposed architecture is confirmed by experimental results obtained through a real-world deployment.**

***Index Terms*—Constrained Application Protocol (CoAP), distributed hash tables (DHTs), Internet of Things (IoT), peer-to-peer (P2P), self-configuration, service discovery (SD), zero-configuration (ZeroConf).**

I. Introduction

THE INTERNET of Things (IoT) is envisioned to bring together billions of devices, also denoted as smart objects, by connecting them in an Internet-like structure, allowing them to communicate and exchange information and to enable new forms of interaction among things and people. Smart objects are typically equipped with a microcontroller, a radio interface for communication, sensors and/or actuators. Smart objects are constrained devices, with limited capabilities in terms of computational power and memory. They are typically battery-powered, thus introducing even more constraints on energy consumption: this motivates the quest for energy-efficient technologies, communication/networking protocols and mechanisms. The Internet Protocol (IP) has been widely envisaged as the true IoT enabler, as it allows to bring the full interoperability among heterogeneous objects. As part of the standardization process which is taking place, new low-power protocols are being defined in international organizations, such as the IETF and the IEEE. At the application layer, the Constrained Application Protocol (CoAP) [1] has been designed to bring the REpresentational State Transfer (REST) paradigm, which was originally conceived for applications based on HTTP [2], to the IoT and is expected to become the standard communication protocol for constrained applications.

Together with application-layer protocols, suitable mechanisms for service and resource discovery should be defined. In particular, CoAP defines the term *service discovery* (SD) as the procedure used by a client to learn about the endpoints exposed by a server. A service is discovered by a client by learning the uniform resource identifier (URI) [3] that references a resource in the server namespace. *Resource discovery* is related to the discovery of the resources offered by a CoAP endpoint. In particular, M2M applications strongly rely on this feature to keep applications resilient to changes, without the need for any external human intervention. A *Resource Directory* (RD) [4] is a network element hosting the description of resources held on other servers, allowing lookups to be performed for those resources.

A crucial issue for the robust applications, in terms of resilience to changes that might occur over time (e.g., availability, mobility, and resource description), and the feasible deployment of (billions of) smart objects is the availability of mechanisms that minimize, if not remove, the need for human intervention for the configuration of newly deployed objects. In fact, the RESTful paradigm is intended to promote software longevity and independent evolution [5], which are both extremely important aspects for IoT and M2M applications deployed on smart objects that are expected to stay operational for long periods of time (e.g., years). Self-configuring service and resource discovery mechanisms should take into account the different scopes that these operations might have: 1) within a local scope, an SD mechanism should enable communication between geographically concentrated smart objects (i.e., residing in the same network) and 2) within a global (large-scale)

Manuscript received June 23, 2014; accepted August 29, 2014. Date of publication September 16, 2014; date of current version October 21, 2014. The work of S. Cirani, G. Ferrari, and L. Veltri was supported by the European Community's Seventh Framework Programme, area "Internetconnected Objects," under Grant 288879, CAL*IP*SO project—"Connect All *IP*-based Smart Objects!" The work of P. Medagliani was supported by the French National Research Agency (ANR) under Grant IRIS ANR-11-INFR-0016. The work of M. Picone was supported by Guglielmo srl, Reggio Emilia (RE), Italy.

S. Cirani, L. Davoli, G. Ferrari, M. Picone, and L. Veltri are with the Department of Information Engineering, University of Parma, 43124 Parma, Italy.

R. Léone and P. Medagliani are with Thales Communications and Security, 92622 Gennevilliers, France.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JIOT.2014.2358296

scope, it should enable communication between smart objects residing in different (and perhaps geographically distant) networks. These approaches should also be scalable, since the expected number of deployed objects is going to be on the order of billions.

Self-configuration is another crucial feature for the diffusion of IoT systems, where all the objects equipped with a radio interface are potential sources of information to be interconnected. An external operator managing a network first needs to configure the system. Clearly, if this operation is carried out manually, possible misconfigurations may arise. This is far more likely when thousands of devices are involved. In addition, an occasional manual network reconfiguration may cause a remarkable system outage: for instance, in an industrial plant, machines may need to be stopped during a normal technical intervention. For this reason, a self-configurable IoT system is expedient to prevent long outages and configuration errors.

Peer-to-peer (P2P) networks have been designed to provide some desirable features for large-scale systems, such as scalability, fault-tolerance, and self-configuration. The main feature that makes P2P networks appealing is the fact that as the number of participating nodes increases, the overall system capacity (in terms of processing and storage capabilities) increases as well. This challenges classical client/server architectures, where an increase in the number of clients may bring the system to saturation and/or failure. P2P networks arrange participating nodes in an overlay network, built on top of an existing network, such as the Internet. The algorithm through which the overlay is created can be used to make a distinction between structured and unstructured P2P networks. Structured P2P networks, such as distributed hash tables (DHTs), are built using *consistent hashing* algorithms, which guarantee that the routing of requests takes a deterministic and upper-bounded number of hops for completion, at the cost of having network traffic for managing and maintaining the overlay. Historically, P2P networks have been associated with file sharing applications, such as eMule[1] and BitTorrent.[2] The decrease in the popularity of file sharing applications has cooled down the interest toward P2P, even though notable applications, such as Skype, historically, use a P2P overlay as backbone to provide a scalable and efficient service. However, the features that P2P networks have been designed for are very appealing for IoT scenarios, where large-scale and robust applications need to be supported. IoT thus represents an opportunity of redemption and renaissance for P2P.

In this work, we propose a scalable and self-configuring architecture for service and resource discovery in IoT aiming at providing mechanisms, which require no human intervention for configuration and simplifying the deployment of IoT applications. Our approach is based on: P2P technologies, at a *large scale*, to provide a distributed large-scale service discovery infrastructure; and, at a *local scale*, zero-configuration (Zeroconf) mechanisms. Information on resources provided by smart objects attached to a local wireless network is gathered by a special boundary node, referred to as "IoT Gateway," which is also part of a P2P overlay used to store and retrieve such information, resulting in a distributed and scalable RD. As will be shown, the performance of the global SD depends only on the number of peers in the P2P overlay: this makes the proposed approach directly scalable when the size of the IoT network increases. The presence of a local SD at the IoT Gateways, instead, makes the process of discovery of new resources automatic. In particular, in our experimental tests, we refer to CoAP for the description of the available endpoints. To the best of our knowledge, this is the first work that provides an architecture and mechanisms that integrate SD on both global and local scales into a unique self-configuring system. We also provide some preliminary results obtained by an implementation and a real-world deployment of our architecture, thus demonstrating its feasibility.

We point out that the proposed architecture is built upon components designed to be absolutely agnostic regarding the format of service and resource descriptors, in order to avoid the introduction of application-specific constraints. In fact, the architecture provides mechanisms for publishing and retrieving information, mapped to service or RD URIs, which can be represented in any suitable content format for service/resource description, either already available, such as CoRE Link Format [6], or foreseeable. The adoption of standard description formats is mandatory to guarantee maximum interoperability, but it is a service's responsibility to enforce this practice. It is also important to remark that IoT applications should be implemented according to the REST paradigm and the definition of CoAP is intended exactly to accomplish this. Client applications, in order to comply with the RESTful paradigm, must follow the Hypermedia as the Engine of Application State (HATEOAS) principle [2], which forbids applications from driving interactions based on out-of-band information rather than on hypermedia. The existence of prerequisites, in terms of resource representation, is a violation of the REST paradigm and cannot be attributed to the SD architecture. The latter is, on the contrary, extremely flexible as it can possible handle any resource description format. The absence of content-related dependencies leads to more robust implementations, in terms of longevity and adaptability to changes that resource descriptions might undergo.

This paper is organized as follows. In Section II, an overview of related work on service and resource discovery mechanisms for IoT systems is provided. In Section III, a detailed description of the basic building blocks of our architecture is given. In Section IV, we describe the architecture for large-scale SD, whereas Section V illustrates the architecture for Zeroconf local service and resource discovery. In Section VI, experimental results based on the deployment of the proposed architecture are presented. Finally, in Section VII, we draw our conclusion.

II. Related Work

In the literature, there already exist some mechanisms implementing SD. Most of them, however, have been conceived for local area networks (LANs) and, then, have been extended for constrained IPv6 over low-power wireless personal area networks (6LoWPAN) networks. One of these mechanisms is

[1][Online]. Available: http://www.emule-project.net/

[2][Online]. Available: http://www.bittorrent.com/

Universal Plug and Play (UPnP) [7], a protocol that allows to automatically create a device-to-device network. However, as UPnP uses TCP as transport protocol and XML as the message exchange format, it is not suited for constrained devices.

Another proposed mechanism is based on the Service Location Protocol (SLP) [8], [9] through which computers and devices can find services in LANs without prior configuration. Devices use SLP to announce in the local network their available services, which are grouped into scopes, i.e., simple strings that allow to classify services. The use of SLP may be important in large-scale IoT scenarios, in order to make SD automatic. However, SLP is not targeting constrained devices, such as those used in IoT. In addition, it relies on centralized approaches, which may be prone to failures. Our approach, instead, is based on a redundant and distributed mechanism for resource location based on a P2P overlay. Finally, by now, no SLP implementation is available for Contiki-based devices.

An alternative to UPnP is Zeroconf [10] networking protocol, which allows to automatically create computer networks based on the TCP/IP Internet stack and that does not require any external configuration. Zeroconf implements three main functionalities: 1) automatic network address assignment; 2) automatic distribution and resolution of host names; and 3) automatic location of network services. The automatic network assignment intervenes when a node first connects to the network. The host name distribution and resolution is implemented using multicast DNS (mDNS) [11], a service that has the same interfaces, packet formats, and semantic of the standard Domain Name System (DNS) messages to resolve host names in networks that do not include a local name server. Concerning the SD phase, Zeroconf implements the DNS-based Service Discovery (DNS-SD) [12]. Using standard DNS queries, a client can discover, in a given domain, the named instances of the service of interest.

Within the field of ubiquitous computing, P2P Interactive Agent eXtensions (PIAX), a P2P platform for geographic service location, has been proposed [13], [14]. Unlike our approach, in PIAX, every node is a peer of the overlay. This approach is not suitable for IoT, since many nodes are constrained in terms of processing capabilities. In addition, PIAX does not provide a URI resolution service, so that it can only take care to route the query to the correct area of the network but not to resolve the endpoint to be contacted.

Efforts have been done to adapt these solutions to the world of constrained devices. In [15], Busnel *et al*. introduce a P2P overlay to perform broadcast or anycast in wireless sensor networks (WSNs) without any centralized element. Sensors are clustered according to their types into specific layers. However, in [15], Busnel *et al*. take into account neither the local SD nor the computational complexity due to the existence of nodes belonging to different layers. In [16], instead, Gutierrez *et al*. introduce the separation between WSNs and P2P networks. Their focus is on exploiting these two types of network to develop a feedback loop to allow developers to define self-managing behaviors. However, Gutierrez *et al*. [16] do not take into account aspects like energy efficiency, self-discovery of resources, or large-scale deployments. In [17], an automatic discovery mechanism has been implemented: each node is responsible for announcing itself to the main gateway through HELLO messages. These messages are sent either in response to a discovery request or proactively sent in an automatic way. The gateway will then be in charge of addressing the requests coming from external networks to the right nodes. In [18], Kovacevic *et al*. propose NanoSD, a lightweight SD protocol, designed for highly dynamic, mobile, and heterogeneous sensor networks. This solution requires extensive multicast and broadcast messages to keep track of service information of the neighboring nodes. Another solution is presented in [19], where a RESTFul web service is developed using HTTP-based SD. However, it does not provide management and status maintenance of existing services. Finally, in [20], Butt *et al*. divide the network in groups, assigning different roles to the nodes in each group. In this way, embedding a directory agent into the border router, a more effective scalability can be handled. However, this architecture tends to be too fragile in the presence of failures of the central border router. In addition, the protocol focuses on in-network service location, but it lacks coordination with other similar entities, thus preventing large-scale discovery.

A few works related to SD in IoT systems are also appearing. In [21], Jara *et al*. sketch an architecture for large-scale SD and location. However, they rely on a centralized solution exposing a search engine to make the integration of distributed service directories feasible. In [22], Paganelli and Parlanti exploit an underlying distributed P2P overlay to support more complex queries, such as multiattribute and range queries. This approach is more focused on service resolution rather than on the creation of the overlay by automatically discovering existing services. Unlike our approach, which aims at being transparent and agnostic of the underlying technology, the P2P overlays have been presented in [23]–[25] focusing especially on RFID for supply chains.

CoAP natively provides a mechanism for SD and location [1]. Each CoAP server must expose an interface */.well-known/core* to which the RD or, more generally, a generic node can send requests for discovering available resources. The CoAP server will reply with the list of resources and, for each resource, with an attribute that specifies the format of the data associated to that resource. CoAP, however, does not specify how a node joining the network for the first time must behave in order to announce itself to the RD node. In [26], this functionality is extended to multicast communications. In particular, multicast resource discovery is useful when a client needs to locate a resource within a limited scope, and that scope supports IP multicast. A *GET* request to the appropriate multicast address is made for */.well-known/core*. Of course, this multicast resource discovery works only within an IP multicast domain and does not scale to larger networks that do not support end-to-end multicast. However, in CoAP, there is no specification on how a remote client can lookup into the RD and query for the resource of interest.

Centralized approaches for SD, such as, for instance, the RD of the CoAP protocol, suffer from scalability and availability limitations and are prone to attacks, such as *denial of service* (DoS). Possible alternatives to this problem may consist of the use of distributed hast tables (DHTs). Key/value pairs are stored in a DHT and any participating node can efficiently retrieve the value associated with a given key. Responsibility for maintaining the mapping from keys to values is distributed among the nodes, in such a way that a change in the set of participants causes a minimal amount of disruption (consistent hashing). This allows a DHT to scale to extremely large numbers of nodes and to handle continuous node arrivals, departures, and failures. Different algorithms and protocols have been already proposed for DHTs; the most significant are Chord [27] (for its simplicity) and Kademlia [28] (for its efficiency). Some works have been published also on the use of P2P for SD. In [29], Yulin *et al*. combine P2P technology and centralized Universal Description Discovery and Integration (UDDI) technology to provide a flexible and reliable service discovery approach. In [30], Kaffille *et al*. apply the concepts of DHTs to the SD creating an overlay P2P to exchange information about available services without flooding the entire network. However, these approaches do not take into account constraints and requirements of IoT. In the following, we will detail our P2P implementation for large-scale service/resource discovery in IoT networks, extending the P2P DHT solution by taking into account the requirements of scalability and self-configuration typical of constrained networks.

III. IoT Gateway

As anticipated in Section I, the SD architecture proposed in this work relies on the presence of an "IoT Gateway." By combining different functions, the IoT Gateway provides both IoT nodes and standard (nonconstrained) nodes with service and resource discovery, proxying, and (optionally) caching and access control functionalities. In this section, the internal architecture of the IoT Gateway and its associated functions will be detailed.

A. *Proxy Functionality*

The IoT Gateway interacts, at the application level, with other IoT nodes through CoAP and may act as both CoAP client and CoAP server. More precisely, according to CoAP specifications, it may act as CoAP *origin server* and/or *proxy*. The CoAP specification defines an *origin server* as a CoAP server on which a given resource resides or has to be created, while a *proxy* as a CoAP endpoint which, by implementing both the server and client sides of CoAP, forward requests to an origin server and relays back the received responses. The *proxy* may also (optionally) perform caching and protocol translation (cross proxy).

The presence of a proxy at the border of an IoT network can be very useful for a number of reasons:

1) to protect the constrained network from the outside, i.e., for security reasons (DoS attacks);
2) to integrate with the existing web through legacy HTTP clients;
3) to ensure high availability of resources through caching;
4) to reduce network load of constrained devices;
5) to support data formats that might not be suitable for constrained applications, e.g., XML.

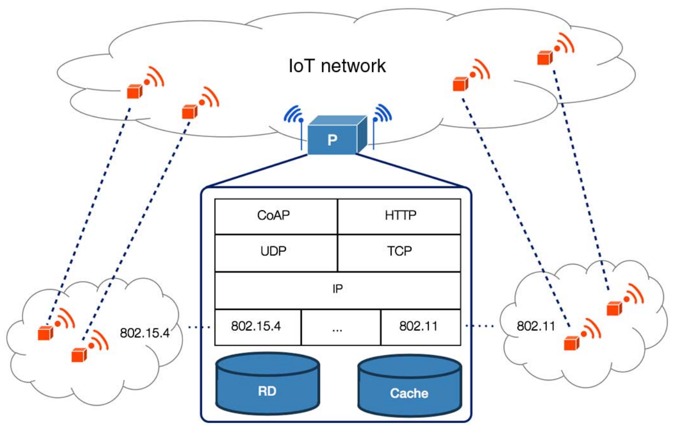


Fig. 1. Architecture of the IoT Gateway with internal layers and caching/RD capabilities.

In Fig. 1, a layered view of the IoT Gateway node is presented.

In addition to standard CoAP proxying behavior, the IoT Gateway may also act a HTTP-to-CoAP proxy by translating HTTP requests to CoAP requests (and vice-versa). Just like a standard CoAP proxying, an HTTP-to-CoAP proxy can integrate two different operational modes: 1) *reverse-proxy*, when, by translating incoming HTTP requests to CoAP requests, it provides access to resources that are created and stored by CoAP nodes within the IoT network (acting as CoAP servers); and 2) *origin-server*, when it acts as both HTTP and CoAP server, by letting CoAP nodes residing in the IoT network (and acting as clients) create resources through CoAP POST/PUT requests, and by making such resources available to other nodes through HTTP and CoAP. The latter operational mode is particularly suited for duty-cycled IoT nodes, which may post resources only during short wake-up intervals. In Fig. 2, the difference between a reverse-proxy and an origin-server is shown.

From an architectural point of view, the IoT Gateway is composed by the following elements.

1) An *IP gateway* managing IPv4/IPv6 connectivity among smart objects in heterogeneous networks (i.e., IEEE 802.15.4, IEEE 802.11.x, and IEEE 802.3) for interconnecting devices operating in different networks by providing an IP layer to let nodes communicate seamlessly.
2) A *CoAP origin server,* which can be used by CoAP clients within the network to post resources which will be maintained by the server on their behalf.
3) A *HTTP-to-CoAP reverse proxy*, optionally equipped with caching capabilities, which can be used for accessing services and resources that are available in an internal constrained network.

The IoT Gateway is, therefore, a network element that coordinates and enables full and seamless interoperability among highly heterogeneous devices, which: may operate different protocols at the link and/or application layers; may not be aware of the true nature of the nodes providing services and resources; and may be geographically distant.

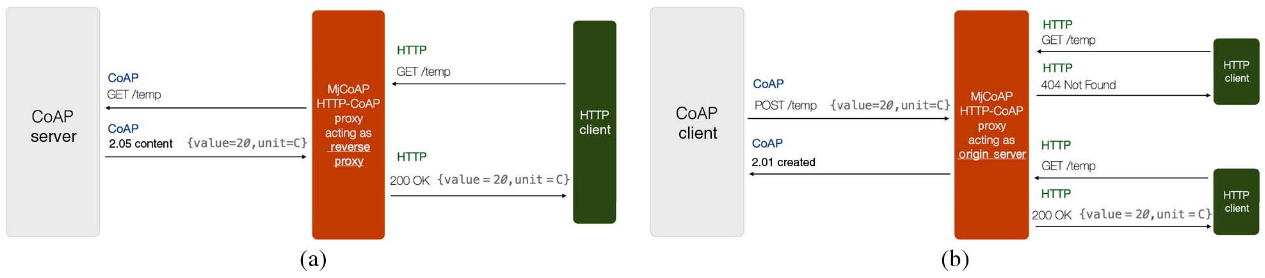

CoAP server
CoAP
GET /temp
CoAP
2.05 content {value=20,unit=C}
MjCoAP HTTP-CoAP proxy acting as reverse proxy
HTTP
GET /temp
HTTP
200 OK {value = 20,unit = C}
HTTP client
(a)
CoAP client
CoAP
POST /temp {value=20,unit=C}
CoAP
2.01 created
MjCoAP HTTP-CoAP proxy acting as origin server
HTTP
GET /temp
HTTP
404 Not Found
HTTP
GET /temp
HTTP
200 OK {value = 20,unit = C}
HTTP client
HTTP client
(b)

Fig. 2. HTTP-to-CoAP proxy acting as (a) reverse proxy and (b) origin server.

B. Service and Resource Discovery

SD aims at getting the host port of the CoAP servers in the network, while resource discovery allows to discover the resources that a CoAP server manages. Because of its role in managing the life cycle of nodes residing in its network, the IoT Gateway is naturally aware of the presence of the nodes and the available services and resources. When the IoT Gateway detects that a CoAP node has joined its IP network, it can query the CoAP node asking for the list of provided services (this is done in CoAP, by sending a GET request to the */.well-known/core* URI). Such information (the RD) is then locally maintained by the IoT Gateway and successively used to route incoming requests to the proper resource node. According to this mechanism, the IoT Gateway may act as a RD for the CoAP nodes within the network.

In Section IV, we detail how IoT Gateways can be federated in a P2P overlay in order to provide a distributed and global service and RD that can be used to discover services at a global scale. In Section V, we then provide a Zeroconf solution to discover resources and services within a local scope with no prior knowledge or intervention required on any node of the network; this allows the IoT Gateways to populate and update their resource and service directories.

IV. P2P-Based Large-Scale SD Architecture

As stated in Section III, IoT Gateways can be federated in a P2P overlay in order to provide a large-scale SD mechanism. The use of a P2P overlay can provide several desirable features as follows.

1) *Scalability*: P2P systems are typically designed to scale and increase their capacity as the number of participants increases.
2) *High availability*: P2P systems are inherently robust since they have no single-point of failure and the failure of a node does not compromise the overall availability of the services and resources provided.
3) *Self-configuration*: P2P systems provide mechanisms to let the overlay reorganize itself automatically when nodes join and leave, requiring no direct intervention for configuration.

These features fit perfectly in IoT scenarios, where billions of objects are expected to be deployed. Among several possible approaches to implement P2P overlays, *structured* P2P overlays, such as DHTs, provide some interesting features, such as efficient storage and lookup procedures, which result in deterministic behaviors with respect to unstructured overlays, which rely on flooding techniques for message routing. In the remainder of this section, we propose a P2P-based approach to provide a scalable and self-configuring architecture for service discovery at a global scale.

IoT Gateways are organized as peers of a structured P2P overlay, which provides an efficient name resolution for CoAP services. The large-scale SD architecture presented in this work relies on two P2P overlays: 1) the distributed location service (DLS) [31] and 2) the distributed geographic table (DGT) [32], [33]. The DLS provides a name resolution service to retrieve all the information needed to access a resource (of any kind) identified by a URI. The DGT builds a distributed geographical knowledge, based on the location of nodes, which can be used to retrieve a list of resources matching geographic criteria. The combination of these two P2P overlay systems allows to build a distributed architecture for large-scale service discovery with the typical features of P2P networks (scalability, robustness, and self-configuration), yet enabling the unique feature of service and resource discovery on a geographical basis. In the following, we first detail the DLS and DGT; then, we describe the overall envisioned system architecture.

A. DLS

The DLS is a DHT-based architecture, which provides a name resolution service based on storage and retrieval of bindings between a URI, identifying some resources (e.g., a web service), and the information that indicates how such a resource can be accessed [31]. Basically, the DLS implements a location service, which can be used to store and retrieve information for accessing services and resources. Together with each contact URI, some other information can be stored, such as the expiration time, an access priority value, and, optionally, a human-readable text (e.g., a contact description or a name).

The service provided by DLS can be considered similar to that of the DNS, since it can be used to resolve a name to retrieve the information needed to access the content related to that name. However, the DNS has many limitations that the DLS overcomes, as follows.

1) The DNS applies only to the Fully-Qualified Domain Name (FQDN) and not to the entire URI.
2) The DNS typically has long propagation times (further increased by the use of caching), which are not suited to highly dynamic scenarios, such as those encompassing node mobility.

3) The DNS basically provides the resolution of a name which results in an IP address, but it does not allow to store and retrieve additional useful information related to the resolved URI, such as the description and the parameters of the hosted service.

Another important aspect that makes the use of the DLS preferable with respect to the DNS is its robustness. In fact, if a DNS server is unreachable, then resolution cannot be performed. On the contrary, P2P overlays do not have single-point of failures that might cause service disruption, resulting in a more robust, dynamic, and scalable solution.

A DLS can be logically accessed through the two simple methods: 1) *put(key,value)* and 2) *get(key)*, where *key* is a Resource URI (actually its hash), whereas *value* is a structured information that may include location information (e.g., a contact URI) together with a display name, expiration time, priority value, etc. The *get(key)* method should return the set of the corresponding values (actually the contact information) associated with the targeted resource. The removal of a resource is performed by updating an existing resource through a put operation with expiration time set to 0. This mapping allows to achieve support for 1) *mobility*: it is sufficient to put and replace an old resource with an updated one, which considers the new position of the resource and 2) *replication*: it is sufficient to execute several *put* operations for the same resource in order to have multiple replicas diffused in the DHT.

The DLS interface can be easily integrated with existing networked applications, such as a middleware layer offering services to applications and working on top of standard transport protocols. Different RPC protocols, such as dSIP [34] and RELOAD [35], may be used for messaging, regardless of the actual selected DHT algorithm (e.g., Chord or Kademlia).

B. DGT

The DGT [32], [33] is a structured overlay scheme, built using directly the geographical location of the nodes. Unlike DHTs, with a DGT each participant can efficiently retrieve node or resource information (data or services) located near any chosen geographic position. In such a system, the responsibility for maintaining information about the position of active peers is distributed among nodes, so that a change in the set of participants causes a minimal amount of disruption.

The DGT is different from other P2P-based localization systems, where geographic information is routed, stored, and retrieved among nodes organized according to a structured overlay scheme. The DGT principle consists in building the overlay taking directly into account the geographic positions of nodes, allowing to build a network where overlay neighbors are also geographic neighbors and no additional messages are needed to obtain the closest neighborhood of a peer. The main difference between the DGT and the DHT-based P2P overlays is the fact that the DGT overlay is structured in such a way that the messages are routed based exclusively on the geographic locations of nodes, rather than on keys that have been assigned to nodes. Typically, DHTs arrange hosts at unpredictable and unrelated points in the overlay, deriving keys through hashing functions. At the opposite, the DGT ensures that hosts that are geographically close are guaranteed to be neighbors in the overlay.

The DGT provides a primitive *get(lat, lon, rad)*, which returns a list of nodes which fall inside the circular region centered in *(lat,lon)* with radius *rad*. Each node that provides a service can be looked up. The *get* primitive is used to localize the list of nodes in a certain geographic region. It might be possible to extend the *get* primitive by introducing query filters, which may be used to return only matching services. The DGT does not provide a generic *put* primitive that can be invoked on the overlay as a whole. However, it is possible to extend the classical DGT behavior with a generic *put* primitive, which consists of the detection of a list of peers in a given area (through the native DGT *get* primitive) and, subsequently, to invoke a *put* method directly on each of the detected peers.

C. P2P-Based Architecture for Large-Scale Service Discovery

The mechanisms presented in Sections IV-A and IV-B are the key ingredients of a large-scale SD architecture. In Fig. 3, an illustrative representation of the system architecture is shown. Several IoT Gateways managing their respective networks are interconnected through the two P2P overlays. Each IoT Gateway is at the same time a DLS peer and a DGT peer. The data structures of the overlays are separated, since they pertain to different operations of the overall architecture. The DLS and DGT overlays are loosely coupled. In fact, the IoT Gateway uses: the DLS to publish/lookup the details of resources and services; and the DGT to publish its presence or discover existing IoT Gateways in a given geographic area. This separation allows the IoT Gateway to access the services provided by each overlay as a "black-box," without any risk of direct interference between the overlays. In fact, the IoT Gateway is responsible to implement the behavior required by the SD architecture. The lifecycle of an IoT Gateway is shown in Fig. 4 and can be described as follows.

1) Upon start up, the IoT Gateway joins the DLS and DGT overlays.
2) The IoT Gateway publishes its presence in the DGT by issuing a *DGT.put(lat, lon,* URI_{GW}) request.
3) When the IoT Gateway detects a new CoAP node in the network, through any suitable means (e.g., Zeroconf), it fetches the node's Local Resource Directory (LRD) through a CoAP GET request targeting the */.well-known/core* URI. The LRD is filled with JSON-WSP[3] documents or similar formats (such as CoRE Link Format) containing the description of all the resources that are hosted by the CoAP node and the information to be used to access them; at this point, the resources included in the fetched node's LRD are added to the IoT Gateway's LRD.

[3]Java Script Object Notation Web-Service Protocol (JSON-WSP) is a web-service protocol that uses JSON for service description, requests, and responses. It has been designed to cope with the lack of service description specification with documentation in JSON-RPC, a remote procedure call protocol in JSON format.

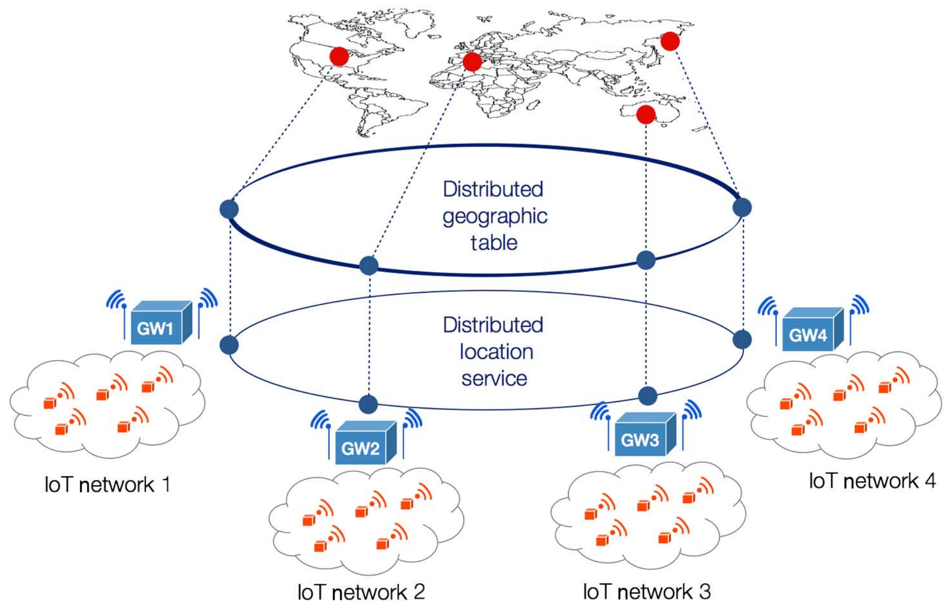


Fig. 3. Large-scale SD architecture. IoT Gateway nodes act as peers of two different P2P overlays. The DLS overlay is used for discovering resources and services as a "white-pages" service providing a name resolution service to be used to retrieve the information needed to access a resource. The DGT is used as a "yellow-pages" service to learn about the existence of IoT Gateway nodes in a certain geographical neighborhood.

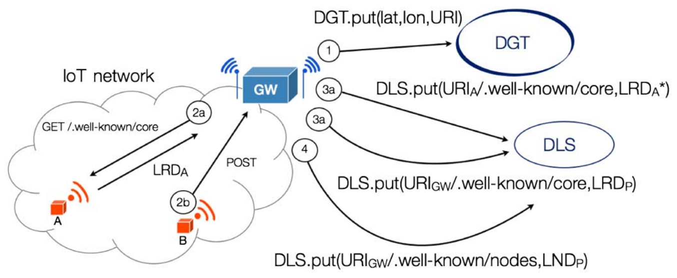

DGT.put(lat,lon,URI)
1
DGT
IoT network
GW
3a
DLS.put(URI$_A$/.well-known/core,LRD$_A$*)
2a
GET /.well-known/core
3a
DLS
LRD$_A$
POST
4
DLS.put(URI$_{GW}$/.well-known/core,LRD$_P$)
2b
A
B
DLS.put(URI$_{GW}$/.well-known/nodes,LND$_P$)

Fig. 4. Messages exchanged when a new node joins the network. First, the IoT Gateway discovers the resources of a new CoAP server or stores them on behalf of a CoAP client. Finally, DGT and DLS are updated with information about the new node.

4) If the IoT Gateway is willing to let the resources be reachable through it, it will modify its LRD to include the references of the URLs to be used to reach the resources through the IoT Gateway, obtaining a new LRD, denoted as LRD*; the IoT Gateway could also delete from the LRD all the references directly related to this resource, in order to avoid that a resource could be accessed without the IoT Gateway relaying messages.
5) The IoT Gateway publishes the LRD* in the DLS through a *DLS.put(URI_{node}/.well-known/core,LRD*)* request.
6) The IoT Gateway keeps track of the list of nodes that are in its managed network, by adding the node in a Local Node Directory (LND).
7) The IoT Gateway publishes the LND pair in the DLS through a *DLS.put(URI_{GW}/.well-known/nodes,LND)* request.
8) If, in addition, the IoT Gateway acts as origin server, it stores its own resources, which will then be published as soon as it receives CoAP POST requests from CoAP clients residing in the inner network.

Steps 3) to 7) are repeated for each CoAP node detected in the network. By publishing all the LRDs in the DLS, a Distributed Resource Directory (DRD) is obtained. The DRD provides a global knowledge of all the available resources. The use of LNDs provides a census of all the nodes that are within a certain network. Location information is managed with JSON-WSP or CoRE Link Format documents, which provide all the details related to parameters and return values. This is similar to WSDL documents, in a more compact, yet as descriptive, format than XML. As soon as a node joins a local network and discovers the presence of an IoT Gateway (it can be assumed that either the IoT Gateway address is hard coded or the node joins the RPL tree finding out the presence of the IoT Gateway—other mechanisms may also be possible), the node announces its presence—we point out that this phase is optional, in the sense that other discovery mechanisms can be adopted. When the IoT Gateway detects this advertisement, it issues a *GET /.well-known/core* to the node, in order to discover its available resources. The node, in return, replies by sending a JSON-WSP or CoRE Link Format document describing its exposed resources, the URI to access them, and their data format. Finally, the IoT Gateway will parse this response and will populate the DLS and DGT accordingly. If other IoT Gateways are present within a certain network, they can act as additional access points for a resource: this can be achieved by publishing an $LRD*'$ containing the URLs related to them. This will lead to highly available and robust routing in very dynamic scenarios, where IoT Gateways may join and leave the network. Should one want to provide fault-tolerance, information replication mechanisms can be also introduced [36].

In the proposed architecture, the DLS can be interpreted as a "white-pages" service to resolve the name of a service, in the form of a URI, to get all the information needed to access it. Similarly, the DGT can be interpreted as a "yellow-pages"

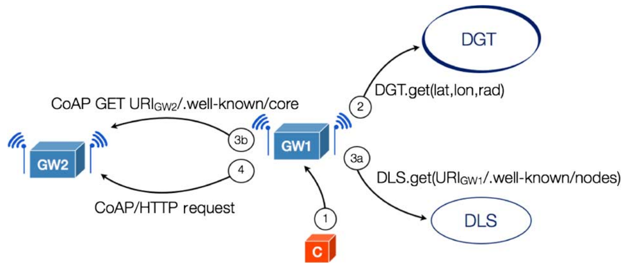

DGT
DGT.get(lat,lon,rad)
2
CoAP GET URI$_{GW2}$/.well-known/core
3b
GW1
GW2
3a
4
DLS.get(URI$_{GW1}$/.well-known/nodes)
CoAP/HTTP request
1
DLS
C

Fig. 5. Data retrieval operations: 1) the client C contacts a known IoT Gateway GW1; 2) GW1 accesses the DGT to retrieve the list of IoT Gateways available in a given area; 3a) GW1 selects one of these IoT Gateways, namely GW2; 3b) GW1 discovers the nodes managed by the GW2 through the DLS or directly by contacting GW2; 4) finally, GW1 queries the node, associated with the resource of interest, managed by GW2.

service, used to retrieve a list of available services matching geographic location criteria, i.e., in the proximity of a geographic position. Note that the DGT is just one possible solution to get matching services, but other mechanisms might be adopted. Other solutions, for instance, may not be related to geographic locations, but matching criteria can be characterized differently, e.g., based on taxonomies/semantics—this is the case if the search is carried out by the type of service rather than by geographical location. The distinction between the lookup services provided by DLS and DGT avoids the inclusion, in the URI, of service or resource information that can dynamically change (such as the location), thus making it possible to support also mobility of services and resources. The DGT and the DLS run in parallel, and the IoT Gateways of a IoT subnetwork act as peers of both the DLS and the DGT. The resulting architecture is very flexible and scalable, in terms of nodes that may join and leave the network at any time. In fact, as explained in Sections IV-A–IV-C, the nature of DLS and DGT P2P overlay networks allows to add new IoT Gateways without requiring the recomputation of the entire hash table. Vice versa, only the nodes responsible for maintaining the resources close to the joining node must update their hash tables in order to include the resources of the new node.

A client, in need for retrieving data from a resource and with no information about the URI to contact, must perform the operations shown in Fig. 5. In particular, the client can perform SD through the mediation of a known IoT Gateway, which is part of the DLS and DGT overlays. The procedure can be detailed as follows (the first five steps are explicitly shown in Fig. 5).

1) The client contacts a known IoT Gateway in order to access the DLS and DGT overlays for SD.
2) The client uses the DGT to retrieve a list of IoT Gateways that are in the surroundings of a certain geographical location through a *DGT.get(lat, lon, rad)* message.
3a) The IoT Gateway selects one of the IoT Gateways returned by the DGT and discovers the list of its managed nodes, through a *DLS.get*(URI_{GW1}*/.well-known/nodes)* request.
3b) The IoT Gateway discovers the resources that are reachable i) by executing a *DLS.get*(URI_{node}*/.well-known/core)* procedure or ii) by issuing a CoAP GET request for URI_{GW2}*/.well-known/core*.
4) The IoT Gateway interacts with the resource by issuing CoAP or HTTP requests targeting the selected resource through the appropriate IoT Gateway; the client can then contact the URI of the resource either directly through CoAP (if supported by the IoT Gateway) or HTTP (by delegating to the IoT Gateway the HTTP-CoAP request translation).
5) Once the command has been transmitted to the CoAP server, the latter will reply with the requested data.
6) If supported, the response will be through CoAP to the client; otherwise, the IoT Gateway will be in charge of response translation.

V. Zeroconf-Based Local SD for Constrained Environments

SD within a local network can be performed using several mechanisms. In scenarios, where a huge number of devices is involved or external human intervention is complicated, it is desirable that all devices can automatically adapt to the surrounding environment. The same considerations apply to devices that do not reside in a particular environment but are characterized by mobility, such as smartphones. In both cases, an SD mechanism, which requires no prior knowledge of the environment, is preferable. In this section, we propose a novel lightweight Zeroconf-based mechanism for service and resource discovery within local networks.

A. Architecture

The considered local SD mechanism is based on the Zeroconf protocol suite. The local SD involves the following elements.

1) IoT nodes (smart objects) belonging to an IoT network.
2) An IoT Gateway, which manages the IoT network and acts as RD.
3) Client nodes, which are interested in consuming the services offered by the IoT nodes.

We assume that the use of IP multicast is supported within the local network and Dynamic Host Configuration Protocol (DHCP) [37] is the dynamic configuration for the IP layer.

B. SD Protocol

There are essentially two relevant scenarios for the application of the proposed SD protocol.

1) A new device offering some service is added to the network and starts participating actively.
2) A client, which is interested in consuming the services offered by the nodes already present in the network, discovers the available services.

In the former scenario, the procedure for adding a new service to the network can be performed in two different ways, depending whether: 1) the smart object can be queried for its services (using the */.well-known/core* URI) or 2) it posts the information related to the services it is offering on the IoT Gateway, which acts as a RD. The difference between

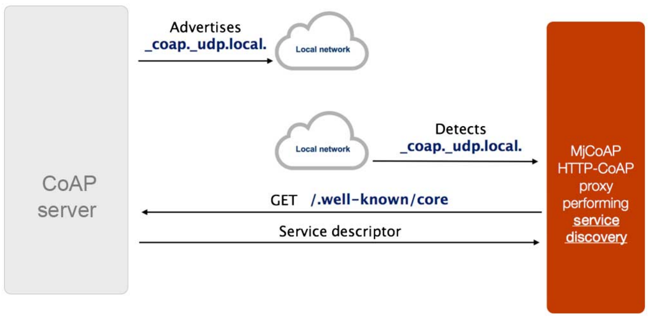

CoAP server
Advertises _coap._udp.local.
Local network
Local network
Detects _coap._udp.local.
GET /.well-known/core
Service descriptor
MjCoAP HTTP-CoAP proxy performing service discovery

Fig. 6. Service advertisement by CoAP server detected by HTTP-to-CoAP proxy.

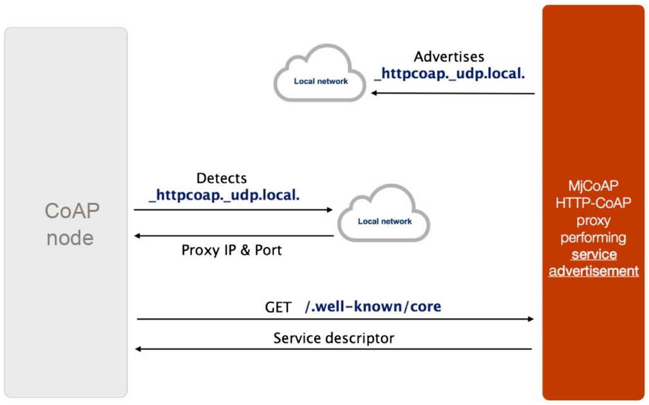

CoAP node
Local network
Advertises _httpcoap._udp.local.
Detects _httpcoap._udp.local.
Local network
Proxy IP & Port
GET /.well-known/core
Service descriptor
MjCoAP HTTP-CoAP proxy performing service advertisement

Fig. 7. Service advertisement by HTTP-to-CoAP proxy detected by CoAP client.

the two scenarios can also be interpreted by characterizing the smart object as a CoAP server or as a CoAP client, respectively.

In case the device acts as a CoAP (origin) server, the service discovery procedure, shown in Fig. 6, is as follows.

1) The IoT node joins the network and announces its presence by disseminating a DNS-SD message for a new service type *_coap._udp.local*.
2) The IoT Gateway, listening for events related to service type *_coap._udp.local*., detects that a new node has joined the network.
3) The IoT Gateway queries the new node for its provided services by sending a CoAP GET request targeting the URI */.well-known/core*.
4) The IoT node replies with a JSON-WSP or CoRE Link Format document describing the offered services.
5) The IoT Gateway updates the list of services that it manages on behalf of the constrained nodes residing in the network, thus making these services consumable by clients residing outside of the IoT network (e.g., remote Internet hosts, which may be unaware of the constrained nature of the network where the service of interest is located).

If the device acts as a CoAP client, instead, the service discovery procedure, shown in Fig. 7, is as follows.

1) The proxy, which is a module of the IoT Gateway, announces its presence, periodically, by disseminating a DNS-SD message for a new service type *_httpcoap._udp.local*.;
2) The joining smart object, which is listening for events related to the service type advertised by the IoT Gateway (*_httpcoap._udp.local*.), detects that an IoT Gateway is available in the network.
3) The smart object sends a CoAP GET request to the URI */.well-known/core* to get a description of the services that the IoT Gateway provides and other information that might be used to detect the most suitable proxy for the client.
4) The IoT Gateway replies with a JSON-WSP or CoRE Link Format document describing the services it provides.
5) The smart object processes the payload and then sends a CoAP POST/PUT request to the IoT Gateway to store resources to be made available to external clients.

In this scenario, the IoT Gateway does not simply forward incoming requests and relay responses, but it acts as a server both toward: 1) the generator of the resource (CoAP client) from which it receives CoAP POST requests and 2) external clients, to which it appears as the legitimate origin server, since the generator of the data is not a CoAP server.

When a client needs to discover the available services, the procedure comprises the following steps.

1) The client sends a CoAP or HTTP request to the proxy targeting the URI */.well-known/core*.
2) The proxy replies with a JSON-WSP or CoRE Link Format document describing all the services managed on behalf of the nodes.
3) The client then uses the received information to perform subsequent CoAP or HTTP requests in order to consume the required services.

The use of IP multicast (i.e., mDNS) offers the main advantage of avoiding to set "*a priori*" the actual network address of any present device, thus eliminating the need for any configuration.

VI. Implementation Results

The solutions presented in the previous sections may apply to many large IoT scenarios where scalable and reliable service and resource discovery is required. In particular, we focus on a smart infrastructure surveillance scenario, where given areas of interest can be monitored by means of the deployment of wireless devices. Each device (smart object) is characterized by the type of the collected data and by its position. A system user may then be interested either in directly contacting a given resource (e.g., a sensor) or having the list of all available resources in a given area. Such wireless sensors are grouped into low-power wireless networks with one or more gateways acting as interfaces between the resulting constrained wireless network and the rest of the network (namely, in the considered scenario, the Internet).

In order to validate the feasibility of the proposed solution and to evaluate its performance, an extensive experimentation has been carried out targeting the reference smart infrastructure

TABLE I
LOCAL SD METRICS

Metric	Description	Dimension
QC	Query client time: the time needed by a node acting as client to send a DNS-SD query and receive a response	ms
QS	Query server time: the time needed by a node acting as server to construct and send a response back to a DNS-SD client	ms

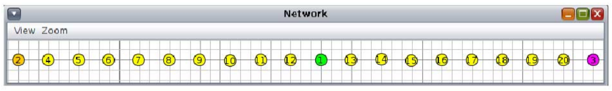
surveillance scenario. The performance evaluation focuses on both the local and large-scale SD mechanisms described in Sections IV and V, respectively.

A. Local SD

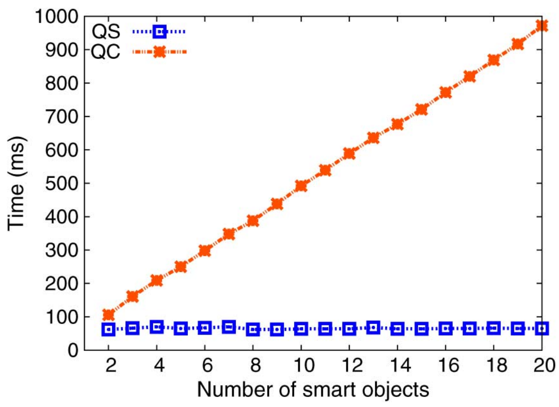
The first phase of experimental performance analysis focuses on the discovery of new CoAP services (associated with constrained devices) available in the local network.

The performance evaluation of the Zeroconf-based local service discovery strategy has been conducted using Zolertia Z1 Contiki nodes, simulated in the Cooja simulator. The Contiki software stack running on each node has been configured in order to fit in the Z1's limited available memory, in terms of both RAM and ROM-Z1 nodes feature nominal 92 kB ROM (when compiling with 20 bit architecture support) and an 8 kB RAM. In practice, the compilation with the Z1 nodes has been performed with a 16 bit target architecture, which lowers the amount of available ROM to roughly 52 kB. The simulated smart objects run Contiki OS, uIPv6, RPL, NullMAC, and NullRDC. The software stack deployed on the smart objects includes our lightweight implementation of the mDNS [11] and DNS-SD [12] protocols, developed in order to minimize memory footprint and to include all the needed modules in the smart objects. The implementations comply with the IETF standards defined in the RFCs and can be replaced by any other compatible implementation, should no particular constraint on the code size be present. The local SD mechanism has been tested on IEEE 802.15.4 networks formed by Contiki nodes arranged in linear and grid topologies. The performance indicator is the time needed to perform a DNS-SD query—from the DNS-SD client perspective—and to process an incoming DNS-SD query and respond—from the DNS-SD server perspective—(dimension: [ms]). The impact of the number of constrained nodes (and, therefore, the number of needed hops) in the network is analyzed. All the results have been obtained by performing 100 SD runs on each configuration. The specific performance metrics are detailed in Table I.

In Fig. 8(a), the considered linear topology, with a maximum of 20 nodes deployed in Cooja, is shown. In particular, node 1 is the 6LoWPAN Border Router (6LBR), which is the root of the RPL tree; node 2 is the node acting as DNS-SD server;



(a)



(b)

Fig. 8. (a) Linear topology considered for multihop Zeroconf-based SD. (b) Average time (dimension: [ms]) of Zeroconf-based SD on Contiki nodes with linear topology.

and node 3 is the node acting as DNS-SD client. The distance between nodes has been set so that the query must follow a multihop path consisting of as many hops as the number of nodes in the network. In Fig. 8(b), the corresponding performance, in terms of QC/QS times, as functions of the number of smart objects, is shown. The QS time has a nearly constant value around 65 ms, since the processing time is independent of the number of nodes in the network. The QC time is a linear function of the number of hops (which, in our scenario, coincides with the number of nodes), since the query packet has to be relayed by each intermediate node to reach the DNS-SD server node.

More complex bidimensional topologies have also been tested in order to evaluate grid-like deployments. Different sizes and arrangements for grids have been considered, as shown in Fig. 9. In all cases: node 1 is the 6LBR; node 2 is the node acting as DNS-SD server; and node 3 is the node acting as DNS-SD client. The topologies in Fig. 9 are denoted as: 1) Grid-A (3 hops); 2) Grid-B (4 hops); 3) Grid-C (6 hops); and 4) Grid-D (5 hops). The corresponding performance of service resolution, in terms of QC/QS times, is shown in Fig. 10. Just like in the linear case, the QS time is independent of the network size (around 65 ms are still needed by the DNS-SD server-side processing). As the number of nodes participating in the network increases, the QC time increases as well, due to the need for multihop communications from client to server. It can be observed that, in the case of Grid-D, even though the number of nodes is larger than in the case of Grid-C, the QC time is shorter. This is due to the fact that the distance between the nodes has been decreased from 40 to 30 m (to minimize collisions due to the use of NullMAC) and, therefore, the total number of hops from the client to the server decreases. In general, it can be concluded that, at a fixed node density, the QC time is a linear function of the number of hops.

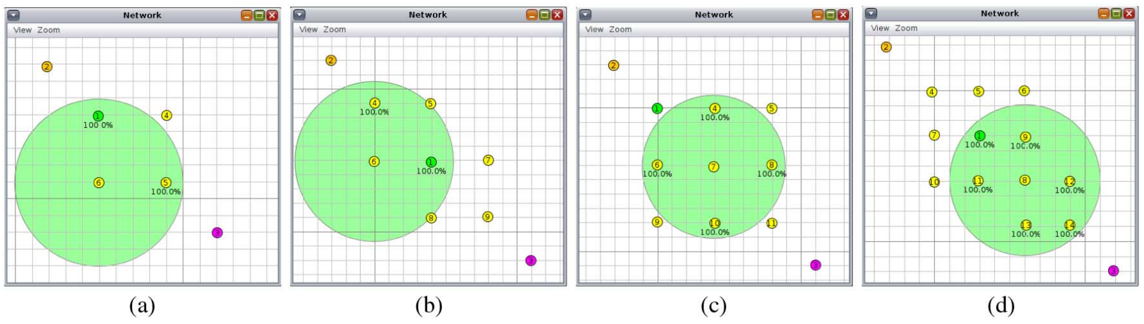


Fig. 9. Grid topologies considered for bidimensional deployments of smart objects.

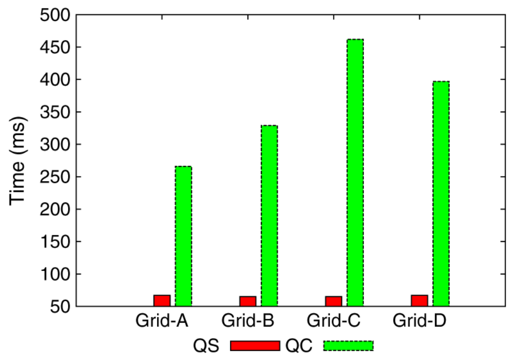


Fig. 10. Average QC/QS times of the Zeroconf-based SD in the grid topologies shown in Fig. 9 (dimension: ms).

B. Large-Scale SD

The second performance evaluation phase focuses on a P2P overlay where multiple IoT Gateways join the network in order to store new services into the DLS overlay and retrieve references to existing ones. The aim of this evaluation is to test the validity of the proposed approach with different configurations and, in particular, to measure the average time required by an IoT Gateway to complete the three main actions in the network (JOIN, PUT, and GET) with different sizes of the P2P overlay. We focus only on the evaluation of the DLS overlay since the published content pertains to IoT services and resources and, therefore, it represents the component of the proposed SD architecture that is directly related to IoT services and resources. The DGT allows to achieve a structured geographical network that can be used to efficiently discover available nodes based on location criteria in a content-agnostic way, which is what the DGT has been designed for and thoroughly evaluated, both in simulative environments and real-world deployments [38], [39].

The performance evaluation has been carried out considering several configurations, with different numbers of IoT Gateways (which are also the peers of the overlay). Each IoT Gateway acts as boundary node of a wireless network with CoAP-aware sensor nodes. The DLS overlay uses a Kademlia DHT and the dSIP protocol for P2P signalling [34], [40]. Both are implemented in Java. The P2P overlay contains up to 1000 nodes deployed over an evaluation platform composed of four cluster hosts, where each computing host is an 8-CPU Intel Xeon E5504 at 2.00 GHz, 16 GB RAM running the Ubuntu 12.04 operating system. The number of nodes in the P2P network has been split evenly among all cluster hosts (up to 250 peers per cluster host), which were connected using a traditional switched ethernet LAN. The HTTP-to-CoAP proxy functionality relies on two different implementations: 1) one based on the mjCoAP library [41], an open-source Java-based RFC-compliant implementation of the CoAP protocol and 2) the other based on the Californium platform [42]. Both HTTP-to-CoAP proxies are written in Java and provide their own local SD mechanisms. The use of two different types of HTTP-to-CoAP proxy shows clearly how the overlay can be easily developed and integrated with currently available technologies. The sensor nodes are either Arduino boards or Java-based emulated CoAP nodes (just for emulating large network scenarios). Each performance result is obtained by averaging over 40 executions of PUT and GET procedures for each size of the overlay.

As anticipated, the following performance metrics are of interest: 1) elapsed time for a JOIN operation (dimension: ms); 2) number of rounds for PUT operations (adimensional); and 3) number of rounds for GET operations (adimensional). The selection of the number of rounds for PUT and GET operations, rather than their times, is expedient to present performance results that are independent of the actual deployment environment. For the JOIN operation, the average total time required to completion is shown in order to provide a practical measurement of the complexity of this operation. However, the very nature of all operations relies on a common iterative procedure (as explained in [28]), thus making it possible to intuitively derive the behavior of all operations in terms of time and rounds.

The performance results are shown in Fig. 11. As expected, the complexity, in terms of (a) JOIN time and (b) and (c) numbers of rounds for PUT/GET operations, is a logarithmically increasing function of the number of peers. In Fig. 11, the experimental data are directly compared with the following logarithmic fitting curves [43]:

$$\begin{aligned}\text{Join time} &\simeq 16.5 + 61.29 \cdot \log n\\ \text{\# of Rounds PUT} &\simeq -5.75 + 3.44 \cdot \log n\\ \text{\# of Rounds GET} &\simeq -0.40 + 0.15 \cdot \log n.\end{aligned}$$

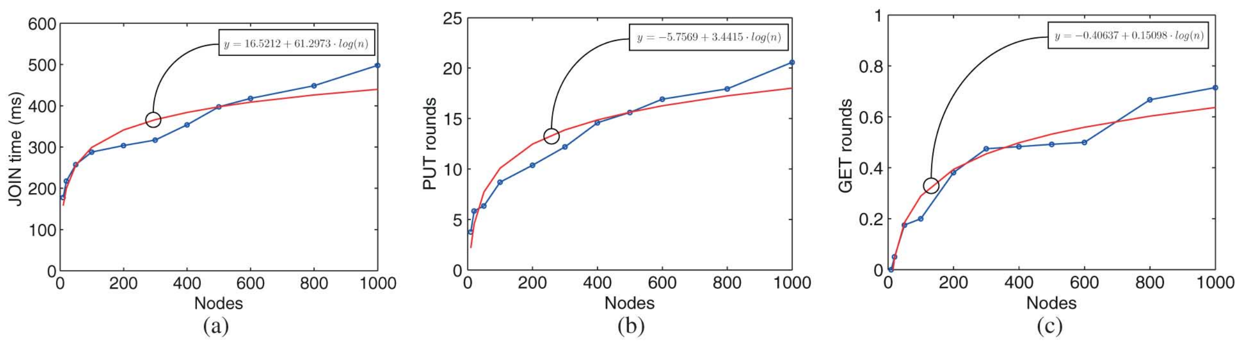


Fig. 11. Experimental results collected to evaluate the performance of the DLS overlay, showing: (a) the average elapsed time (dimension: [ms]) for JOIN operations; (b) the average number of rounds (adimensional) for PUT operations; and (c) the average number of rounds (adimensional) for GET operations on the DLS toward the number of active IoT Gateways in the P2P network. Plotted data have also been used to construct fitting curves (in red); the formula of the fitting curve is reported in the top-right corner.

This clearly proves the scalability brought by the use of a P2P approach, confirming the formal analysis and results discussed in [28].

VII. Conclusion

In this paper, we have introduced a novel architecture for self-configurable, scalable, and reliable large-scale service discovery. The proposed approach provides efficient mechanisms for both local and global SD. First, we have described the IoT Gateway and the functionalities that this element must implement to perform resource and SD. Then, we have focused on large-scale distributed resource discovery exploiting a proper P2P overlays, namely DLS and DGT, which implement, respectively, "white-pages" and "yellow-pages" services. Finally, we have shown a solution for automated local SD that allows to discover resources available in constrained WSNs and to publish them into the P2P overlay with no need for any prior configuration (Zeroconf). An extensive experimental performance evaluation of the proposed local and large-scale SD mechanisms has been performed. For the local SD, experimentation has been conducted on Contiki-based nodes operating in constrained (IEEE 802.15.4) networks with RPL in the Cooja simulator. The large-scale SD mechanism has been deployed and tested on P2P overlays of different sizes, spanning from a few to 1000 peers, in order to evaluate the performance in terms of scalability and self-configuration. The obtained results show that the time required for service resolution in the Zeroconf-based approach for local SD is linearly dependent on the number of hops in the path between the client and server node. Considering large-scale SD, the adoption of a P2P overlay provides scalability in terms of time required to perform the basic publish/lookup operations. In conclusion, the easy and transparent integration of two different types of overlays shows the feasibility and reliability of a large-scale architecture for efficient and self-configurable service and resource discovery in IoT networks.

Acknowledgment

The authors would like to thank M. Antonini, University of Parma, for his valuable contributions and support.

References

[1] Z. Shelby, K. Hartke, and C. Bormann. (Jun. 2013). *Constrained Application Protocol (CoAP)*. RFC 7252 (Proposed Standard), Internet Engineering Task Force [Online]. Available: http://tools.ietf.org/html/rfc7252

[2] R. Fielding *et al*., (Feb. 2013). *Hypertext Transfer Protocol—HTTP/1.1*. Internet Engineering Task Force RFC 2616 [Online]. Available: http://tools.ietf.org/html/rfc2616

[3] T. Berners-Lee, R. Fielding, and L. Masinter. (Jan. 2005). *Uniform Resource Identifier (URI): Generic Syntax*. Internet Engineering Task Force, RFC 3986 [Online]. Available: http://tools.ietf.org/html/rfc3986

[4] Z. Shelby, C. Bormannand, and S. Krco. (Dec. 2013). *CoRE Resource Directory*, Internet Engineering Task Force, Internet-Draft draft-ietf-Core-Resource-Directory-01 (Proposed Standard) [Online]. Available: http://tools.ietf.org/id/draft-ietf-core-resource-directory-01.txt

[5] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dessertation, Dept. Inform. Comput. Sci., Univ. California, Oakland, CA, USA, 2000 [Online]. Available: http://www.ics.uci.edu/fielding/pubs/dissertation/top.htm

[6] Z. Shelby. (2012, Aug.). *Constrained RESTful Environments (CoRE) Link Format*. RFC 6690 (Proposed Standard), Internet Engineering Task Force [Online]. Available: http://www.ietf.org/rfc/rfc6690.txt

[7] UPnP Forums. (1999) [Online]. Available: http://www.upnp.org/

[8] E. Guttman, C. Perkins, and J. Veizades. (Jun. 1999). *Service Location Protocol, Version 2*. Internet Engineering Task Force, RFC 2608 [Online]. Available: http://tools.ietf.org/html/rfc2608

[9] E. Guttman. (Jan. 2002). *Vendor Extensions for Service Location Protocol, Version 2*. Internet Engineering Task Force, RFC 3224 [Online]. Available: http://tools.ietf.org/html/rfc3224

[10] Zeroconf Website. (1999) [Online]. Available: http://www.zeroconf.org/

[11] S. Cheshire and M. Krochmal. (Feb. 2013). *Multicast DNS*. Internet Engineering Task Force, RFC 6762 [Online]. Available: http://tools.ietf.org/html/rfc6762

[12] S. Cheshire and M. Krochmal. (Feb. 2013). *DNS-Based Service Discovery*. Internet Engineering Task Force, RFC 6763 [Online]. Available: http://tools.ietf.org/html/rfc6763

[13] Piax Website (2004). [Online]. Available: http://www.piax.org/en

[14] Y. Kaneko, K. Harumoto, S. Fukumura, S. Shimojo, and S. Nishio, "A location-based peer-to-peer network for context-aware services in a ubiquitous environment," in *Proc. Sym. Appl. Internet Workshops*, Jan. 2005, pp. 208–211.

[15] Y. Busnel, M. Bertier, and A.-M. Kermarrec, "Solist or how to look for a needle in a haystack? A lightweight multi-overlay structure for wireless sensor networks," in *Proc. IEEE Int. Conf. Netw. Commun. Wireless Mobile Comput. (WiMob'08)*, Avignon, France, Oct. 2008, pp. 25–31.

[16] G. Gutierrez, B. Mejias, P. Van Roy, D. Velasco, and J. Torres, "WSN and P2P: A self-managing marriage," in *Proc. 2nd IEEE Int. Conf. Self-Adaptive Self-Organ. Syst. Workshops (SASOW'08)*, Oct. 2008, pp. 198–201.

[17] J. Leguay, M. Lopez-Ramos, K. Jean-Marie, and V. Conan, "An efficient service oriented architecture for heterogeneous and dynamic wireless sensor networks," in *Proc. 33rd IEEE Conf. Local Comput. Netw., (LCN'08)*, 2008, pp. 740–747.

[18] A. Kovacevic, J. Ansari, and P. Mahonen, "NanoSD: A flexible service discovery protocol for dynamic and heterogeneous wireless sensor networks," in *Proc. 6th Int. Conf. Mobile Ad-Hoc Sensor Netw. (MSN'10)*, 2010, pp. 14–19.

[19] S. Mayer and D. Guinard, "An extensible discovery service for smart things," in *Proc. 2nd Int. Workshop Web of Things (WoT'11)*, 2011, pp. 7:1–7:6.

[20] T. A. Butt, I. Phillips, L. Guan, and G. Oikonomou, "TRENDY: An adaptive and context-aware service discovery protocol for 6LoWPANs," in *Proc. 3rd Int. Workshop Web of Things (WOT'12)*, 2012, pp. 2:1–2:6.

[21] A. Jara *et al*., "Mobile digcovery: A global service discovery for the Internet of Things," in *Proc. 27th Int. Conf. Adv. Inf. Netw. Appl. Workshops (WAINA'13)*, 2013, pp. 1325–1330.

[22] F. Paganelli and D. Parlanti, "A DHT-based discovery service for the Internet of Things," *J. Comp. Netw. Commun.*, vol. 2012, 2012, doi:10.1155/2012/107041.

[23] N. Schnemann, K. Fischbach, and D. Schoder, "P2P architecture for ubiquitous supply chain systems," in *ECIS*, S. Newell, E. A. Whitley, N. Pouloudi, J. Wareham, and L. Mathiassen, Eds., 2009, pp. 2255–2266.

[24] S. Shrestha, D. S. Kim, S. Lee, and J. S. Park, "A peer-to-peer RFID resolution framework for supply chain network," in *Proc. 2nd Int. Conf. Future Netw. (ICFN'10)*, 2010, pp. 318–322.

[25] P. Manzanares-Lopez, J. P. Muoz-Gea, J. Malgosa-Sanahuja, and J. C. Sanchez-Aarnoutse, "An efficient distributed discovery service for {EPCglobal} network in nested package scenarios," *J. Netw. Comput. Appl.*, vol. 34, no. 3, pp. 925–937, 2011.

[26] A. Rahman and E. Dijk. (2014, Sep.). *Group Communication for CoAP*. Internet Engineering Task Force, Internet-Draft Draft-Ietf-Core-Groupcomm-24 [Online]. Available: http://tools.ietf.org/id/draft-ietf-core-groupcomm-24.txt

[27] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, pp. 149–160, Aug. 2001 [Online]. Available: http://doi.acm.org/10.1145/964723.383071

[28] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the XOR metric," in *Proc. 1st Int. Workshop Peer-to-Peer Syst.*, (IPTPS'01), 2002, pp. 53–65 [Online]. Available: http://dl.acm.org/citation.cfm?id=646334.687801

[29] N. Yulin, S. Huayou, L. Weiping, and C. Zhong, "PDUS: P2P-based distributed UDDI service discovery approach," in *Int. Conf. Serv. Sci. (ICSS)*, 2010, pp. 3–8.

[30] S. Kaffille, K. Loesing, and G. Wirtz, "Distributed service discovery with guarantees in peer-to-peer networks using distributed hashtables," in *Proc. Int. Conf. Parallel Distrib. Process. Tech. Appl. (PDPTA)*, 2005, pp. 578–584.

[31] S. Cirani and L. Veltri, "Implementation of a framework for a DHT-based distributed location service," in *Proc. 16th Int. Conf. Software, Telecommun. Comput. Netw., (SoftCOM'08)*, Sep. 2008, pp. 279–283.

[32] M. Picone, M. Amoretti, and F. Zanichelli, "GeoKad: A P2P distributed localization protocol," in *Proc. 8th IEEE Int. Conf. Pervasive Comput. Commun. Workshops (PERCOM)*, 2010, pp. 800–803.

[33] M. Picone, M. Amoretti, and F. Zanichelli, "Proactive neighbor localization based on distributed geographic table," *Int. J. Pervasive Comput. Commun.*, vol. 7, pp. 240–263, 2011.

[34] D. Bryan, B. Lowekamp, and C. Jennings. (Feb. 2007). *dSIP: A P2P Approach to SIP Registration and Resource Location*. Internet Engineering Task Force, Internet-Draft draft-bryan-p2psip-dsip-00 [Online]. Available: http://tools.ietf.org/id/draft-bryan-p2psip-dsip-00.txt

[35] C. Jennings, B. Lowekamp, E. Rescorla, S. Baset, and H. Schulzrinne. (2013, Feb.). *REsource LOcation And Discovery (RELOAD) Base Protocol*. Internet Engineering Task Force, Internet-Draft Draft-ietf-p2psip-Base-26 [Online]. Available: http://tools.ietf.org/html/draft-ietf-p2psip-base-26

[36] P. Gonizzi, G. Ferrari, V. Gay, and J. Leguay, "Data dissemination scheme for distributed storage for IoT observation systems at large scale," *Inf. Fusion*, vol. 22, pp. 16–25, 2015 [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1566253513000444

[37] R. Droms. (1997, Mar.). *Dynamic Host Configuration Protocol*. Internet Engineering Task Force, RFC 2131 [Online]. Available: http://tools.ietf.org/html/rfc2131.txt

[38] M. Picone, M. Amoretti, and F. Zanichelli, "Evaluating the robustness of the DGT approach for smartphone-based vehicular networks," in *Proc. IEEE 36th Conf. Local Comput. Netw. (LCN)*, Oct. 2011, pp. 820–826.

[39] M. Picone, M. Amoretti, and F. Zanichelli, "A decentralized smartphone based traffic information system," in *Proc. 2012 IEEE Intell. Veh. Symp.* Jun. 2012, pp. 523–528.

[40] S. Cirani and L. Veltri, "A Kademlia-based DHT for resource lookup in P2PSIP," Obsolete Internet Draft, Oct. 2007.

[41] S. Cirani, M. Picone, and L. Veltri, "mjCoAP: An open-source lightweight Java CoAP library for Internet of Things applications," in *Workshop Interoperability and Open-Source Solutions Internet Things, Conjunction (SoftCOM'14): Proc. 22nd Int. Conf. Software, Telecommun. Comput. Netw.*, Split, Croatia, Sep. 2014, 16 pp.

[42] M. Kovatsch, M. Lanter, and Z. Shelby, "Californium: Scalable cloud services for the Internet of Things with coap," in *Proc. 4th Int. Conf. Internet of Things (IoT'14)*, Cambridge, MA, USA, Oct. 2014, pp. 1–6.

[43] E. W. Weisstein. (2014 Sep., 30). *Least Squares Fitting–Logarithmic* [Online]. Available: http://mathworld.wolfram.com/LeastSquaresFitting Logarithmic.html

Simone Cirani received the Dr. Ing. (Laurea) degree in computer science (*cum laude*) and Ph.D. degree in information technologies from the University of Parma, Parma, Italy, in 2007 and 2011, respectively.

He is a Postdoctoral Research Associate with the Department of Information Engineering, University of Parma. His research research interests include Internet of Things, peer-to-peer networks, network security, pervasive computing, and mobile application development.

Luca Davoli received the B.Sc. and M.Sc. degrees in computer science engineering from the University of Parma, Parma, Italy, in 2011 and 2013, respectively, and is currently working toward the Ph.D. degree in information engineering at the University of Parma.

His research interests include peer-to-peer networks, security and protocols for the Internet of Things, and pervasive computing.

Gianluigi Ferrari is currently an Associate Professor of Telecommunications with the University of Parma, Parma, Italy. He was a Visiting Researcher with the University of Southern California (USC), Los Angeles, CA, USA, from 2000 to 2001, Carnegie Mellon University (CMU), Pittsburgh, PA, USA, from 2002 to 2004, King Mongkut's Institute of Technology Ladkrabang (KMITL), Bangkok, Thailand, in 2007, and Universitè Libre de Bruxelles (ULB), Brussels, Belgium, in 2010. Since 2006, he has been the Coordinator of the Wireless Ad-Hoc and Sensor Networks (WASN) Laboratory, Department of Information Engineering. As of today, he has been published extensively in the areas of wireless *ad hoc* and sensor networking, adaptive digital signal processing, and communication theory. He currently serves on the Editorial Boards of several international journals.

Prof. Ferrari was the recipient of the Paper/Technical Awards at IWWAN06, EMERGING10, BSN 2011, ITST-2011, SENSORNETS 2012, EvoCOMNET 2013, and BSN 2014.

Rémy Léone received the Master's degree in network from the Université Pierre et Marie at Telecom Paris Tech (INFRES) Laboratory and Thales Communications and Security, Gennvilliers, France.

His research interests include wireless sensors networks.

Paolo Medagliani received the Master's degree and Ph.D. degree in information technologies from the University of Parma, Parma, Italy, in 2006 and 2010, respectively.

He is an Advanced Studies Engineer with the Networking Group, Thales Communications and Security, Gennevilliers, France. He has authored several international conference papers, journal papers, and patents. His research interests include adaptation of wireless sensor networks for energy-aware systems and, more in general, networking and medium access problematic in scenarios with constrained nodes, performance analysis and design of wireless sensor networks, and *ad hoc* networks.

Marco Picone received the Laurea (*cum laude*) degree in computer engineering and Ph.D. degree in information technologies from the University of Parma, Parma, Italy, in 2008 and 2012, respectively.

He is a Postdoctoral Research Associate with the University of Parma. In 2011, he was a Research Visitor with the Computer Laboratory, University of Cambridge, Cambridge, U.K. His research interests include mobile and pervasive computing, location based services, distributed systems, wireless sensor networks, and the Internet of Things.

Luca Veltri received the Laurea degree in telecommunication engineering and Ph.D. degree in communication and computer science from the University of "Rome La Sapienza," Rome, Italy, in 1994 and 1999, respectively.

Since 2002, he has been an Assistant Professor with the University of Parma, Parma, Italy, where he currently teaches classes on telecommunication networks and network security. His research interests include P2P systems, future internet, and network security.