DINAS: A Lightweight and Efficient Distributed Naming Service for All-IP Wireless Sensor Networks

Michele Amoretti, *Member, IEEE*, Olivier Alphand, Gianluigi Ferrari, *Senior Member, IEEE*,
Franck Rousseau, *Member, IEEE*, and Andrzej Duda, *Member, IEEE*

***Abstract*—The Internet of Things (IoT) requires a compact naming scheme, which can also bring significant advantages to service registration and discovery. We propose a novel approach, denoted as distributed naming service, which provides a new naming scheme as well as an efficient service discovery protocol for wireless sensor networks. It is based on three pillars: 1) Bloom filters, to create compact names from node descriptions; 2) message propagation strategies, to publish and discover information—not only names—within the network; and 3) distributed caches, to store names within the network. In this paper, we assume ContikiMAC at layer 2, IPv6 and Routing Protocol for Low-Power and Lossy Networks (RPL) at layer 3, and we present two particular UDP-based message propagation strategies that take advantage of the RPL protocol at layer 3. We evaluate the performance of the proposed solutions through Contiki/Cooja simulations and on a real testbed, using the open and large scale FIT IoT-LAB.**

***Index Terms*—Naming and addressing, resource-constrained networks.**

I. Introduction

WE PROPOSE a novel approach to naming and service discovery in resource-constrained wireless sensor networks (WSNs), denoted as DIstributed NAming Service (DINAS). This paper extends our previous work [1] with the following additional contributions.

1) Improved description of the DINAS approach.
2) Introduction of a new binding propagation scheme called Routing Protocol for Low-Power and Lossy Networks (RPL)-DHT.
3) Extensive evaluation carried out with a hybrid simulation/emulation approach and also on a real testbed, using several topologies with different sizes, as well as a larger number of performance indicators.
4) Extended discussion of related work with a comparative perspective with respect to DINAS.

The development of tiny IP stacks, such as Contiki uIPv6 [2], has allowed the integration of everyday objects, sensors, and actuators within the Internet, which enables direct Internet access to such resource-constrained devices. Border routers allow routing between smart objects or sensor networks and traditional IP networks by providing end-to-end IP connectivity. On top of IP connectivity, the Internet of Things (IoT) requires standardized and highly scalable schemes to discover the names of smart objects and also the services that they may offer. Names are necessary to denote things (nodes, networks, data, services, etc.). They may be human-readable or only suitable for machine-to-machine communications: as names, they need to be globally or locally unique; but, as services, we may accept that several nodes provide the same resources and, therefore, have similar names.

The problems that have motivated this paper are as follows.

1) How to effectively and efficiently represent names within an IPv6-based WSN or in an IPv6-based network including several WSNs.
2) How to efficiently publish/lookup names within the networks or outside of them.

Besides solving the basic naming problem, our scheme also provides a new means for service discovery in sensor networks in which an efficient and scalable solution is still lacking.

The traditional Domain Name System (DNS) approach requires that a DNS server, placed at the border router, stores all bindings and replies to all name resolution queries. Such operations may result in a large overhead and excessive energy consumption. The centralized resource directory[1] proposed by the IETF CoRE Working Group suffers from the same limitations. The interest on a distributed service comes from the constraints of sensor networks. In many cases, information exchange among co-located sensors is sufficient and there is no need to interact with the sink (e.g., the control of air-conditioning, in which actuators consume temperatures published by neighboring sensors). In many cases, moreover, nodes may want to discover services offered in the network through

Manuscript received May 9, 2016; revised August 17, 2016 and October 31, 2016; accepted November 30, 2016. Date of publication December 15, 2016; date of current version June 15, 2017. The work of O. Alphand, F. Rousseau, and A. Duda was supported in part by the European Commission FP7 Project CALIPSO under Contract 288879 and in part by the French National Research Agency Project IRIS under Contract ANR-11-INFR-016. The work of G. Ferrari was supported by the European Commission FP7 Project CALIPSO under Contract 288879.

M. Amoretti and G. Ferrari are with the Department of Information Engineering, University of Parma, 43124 Parma, Italy (e-mail: michele.amoretti@unipr.it; gianluigi.ferrari@unipr.it).

O. Alphand, F. Rousseau, and A. Duda are with the Grenoble Informatics Laboratory, University of Grenoble, 38041 Grenoble, France (e-mail: olivier.alphand@imag.fr; franck.rousseau@imag.fr; andrzej.duda@imag.fr).

Digital Object Identifier 10.1109/JIOT.2016.2640317

[1] [Online]. Available: http://tools.ietf.org/html/draft-ietf-core-resource-directory-01

a naming service in a way similar to mDNS/Bonjour [3], [4]. Our approach enables such extensions.

DINAS is based on three "pillars."

1) Bloom filters (BFs) to create compact names from node and service descriptions.
2) A strategy for propagation of name-address bindings and queries within the network.
3) Caches, based on name similarity, to distribute name storage all over the network (instead of concentrating them at the border router).

Once a node has created its name as a BF, it propagates the information about the name-address binding to other nodes. The strategy for their propagation and caching policy in DINAS is that, upon reception of a name notification, a node decides whether to store the name and where to propagate it according to the content of the local cache. The strategy leads to grouping similar names—i.e., similar names are stored at the same nodes. Moreover, a node chooses the next hop based on the similarity of a name to be propagated to the names already stored in its local cache. Name requests (which are not stored) propagate in the network according to the same principle. In this paper, we also specialize the general propagation strategy to the case in which the network already builds its routing structure as a destination-oriented directed acyclic graph (DODAG) with RPL.[2] In this case, name propagation takes advantage of the network topology structure: each node sends its name binding to its parent, which, in turn, propagates it up the DODAG to the root node that sends the binding further down to a proper subgraph. Intermediate nodes on the propagation path store the binding in a cache so that they can resolve a given name. Name propagation is limited in depth so that only a portion of the nodes stores the binding: in particular, such nodes are placed at strategic points in the network, reachable in a few hops from any node.

This paper is organized as follows. In Section II, we describe the principles of DINAS. In Section III, we evaluate its performance, using its current implementation within Contiki. We discuss the related work in Section IV. Finally, we conclude this paper with an overview of future research directions in Section V.

II. Principles of DINAS

With DINAS, a node joining the network creates its name by encoding its descriptor (including the list of features, services, and the information that the node may provide) in a BF of a given size [5], [6]. DINAS handles the binding between a name and its IPv6 address so that nodes can resolve a name to obtain the address. To improve efficiency and distribute bindings over the network, nodes maintain caches of name-address bindings based on name similarity.

A. From Descriptors to Names

More formally, the construction of a node name starts with a m-bit BF B filled with zeros. Let $\mathrm{KW} = \{\mathrm{kw}_1, \mathrm{kw}_2, \ldots, \mathrm{kw}_n\}$ be a set of keywords associated with the node (the set of its characteristics, service descriptions) and let $H = \{h_1, h_2, \ldots, h_k\}$ be a set of hash functions, where $h_j : \mathrm{KW} \to \{0, m-1\}$. The name of the node is built as follows: $B[h_j(\mathrm{kw}_i)] \Leftarrow 1, \forall \mathrm{kw}_i \in \mathrm{KW}, \forall h_j \in H$. In other words, for every keyword kw_i, we set to 1 the BF's bits corresponding to the results of k hash functions computed on the keywords, as shown in Fig. 1. We assume that the names are unique, because we can always add a unique identifier as a keyword when filling a BF.

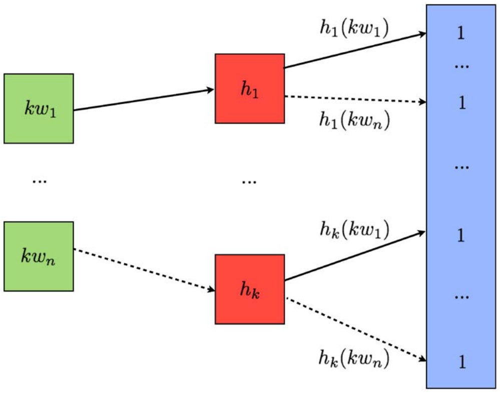


Fig. 1. Mapping keywords to a flat name with an m-bit BF.

We have used the basic BF because of the sensor node constraints: it is not easy (or even possible) to implement complex data structures on constrained devices like TMote Sky. To limit the memory usage, even the code must be as compact as possible. Enhanced versions of the BF such as compressed BFs can only slightly reduce the size of the filter while introducing some computational overhead [7].

To find if keyword kw_x is within the set of keywords encoded in a given BF B, we check the bits at positions $h_1(\mathrm{kw}_x)$, $h_2(\mathrm{kw}_x), \ldots, h_m(\mathrm{kw}_x)$. If one of them is 0, then kw_x is not in the set represented by B. Otherwise, we conjecture that kw_x is in the set. There is a probability of a "false positive" or, for historical reasons, a "false drop" if all bit positions are set to 1 and keyword kw_x is not in the set. We can choose parameters k and m so that the probability of this case is sufficiently low.

After inserting n keywords into a BF of size m, the probability that a particular bit is still 0 is the following:

$$p = \left(1 - \frac{1}{m}\right)^{kn}. \tag{1}$$

Hence, the probability of a false positive is

$$p_{f+} = (1-p)^k \approx \left(1 - e^{-kn/m}\right)^k. \tag{2}$$

Since k must be nonnegative integer, the right hand side of (2) is minimized for [6]

$$k = \left\lfloor \frac{m}{n} \ln 2 \right\rfloor. \tag{3}$$

In practice, one may choose a value lower than the optimal one to reduce computational overhead. For instance, if we want to describe each node with up to $n = 10$ keywords, with $m = 160$ and $k = 11$, the probability of false positives would be $p_{f+} = 4.5 \cdot 10^{-4}$.

[2] [Online]. Available: http://tools.ietf.org/html/rfc6550

We define *similarity* between BFs A and B as the number of bit positions with the same value, that is,

$$\sum_{i=1}^{m} A_i \oplus B_i.$$

Names in form of a BF allow aggregation of several keywords in one name, whereas other protocols may need an entry for each keyword. A request for a name resolution is also encoded as a m-bit BF so that the name resolution process consists of searching for names that match the request. The scheme also supports the resolution of subsets of names. For example, if we need to find the nodes that deliver the temperature of all rooms in a building, we can make a request for a BF created from "temperature" and the building name keywords such as

{data:Temperature; location:EnsimagBuilding;}.

B. Message Types

To maintain name-address bindings in the network and resolve names, DINAS defines four types of messages: 1) *notification*; 2) *request*; 3) *reply*; and 4) *neighbor announcement*. A notification contains a name-address binding: the BF name of a node and its IP address. A request includes the BF corresponding to the name to be resolved and the IP address of the node that tries to resolve the name. A reply provides the requested binding: the name and the associated IP address. Neighbor announcements are like notifications, but, once received by the direct neighbors of the notifier, they do not have to be further propagated. Notifications and requests also contain a supplementary field, denoted as D, whose role depends on the binding propagation strategy.

C. Similarity Cache

Nodes store name-address bindings in a similarity cache having the following structure.

1) Name: BF (m bits).
2) IP address corresponding to the name.
3) Similarity next-hop.
4) Timestamp.

The similarity next-hop is the IP address of the neighbor that has provided the name.

Upon receiving a notification, a node decides to add a name to the cache based on similarity—it adds the name if similarity of the name with the cached names exceeds threshold $T_1 \in \{1\%, 2\%, \ldots, 100\%\}$ (similarity expressed as a percentage). If the cache is empty, the name is added unconditionally. If there is already the name in the cache, the node updates the timestamp. If the cache is full, the name will replace the oldest one in the cache, if the similarity exceeds a threshold $T_2 > T_1$. In Fig. 2, the principles of similarity cache filling are shown.

D. Binding Propagation

In DINAS, nodes propagate name binding notifications in a proactive and periodic manner within the network and some nodes store them in similarity caches for further name resolution. The propagation process is selective, thus resulting in much less overhead than flooding, but, as a consequence, not all nodes are aware of all names. Therefore, name resolution requires propagation of name requests to find a node that stores a given name in its cache and can reply with the desired information. We limit the propagation of name requests to several hops to reduce overhead. Upon reception of a reply, nodes may store it in their caches to reinforce the distributed name resolution process: in fact, this name will appear in an increasing number of nodes. In this way, the most popular (i.e., requested) names are also the most replicated ones. To avoid inconsistencies, due to nodes joining/leaving or any topology changes, names are cached as soft states and, therefore, need to be refreshed periodically for maintenance.

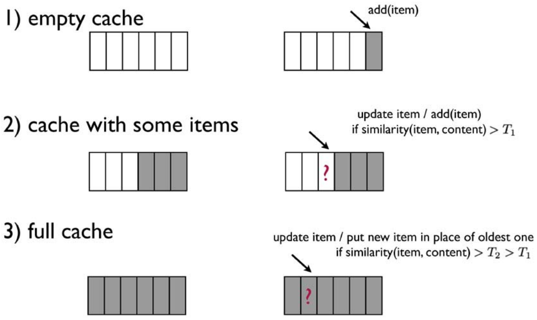

1) empty cache
add(item)
2) cache with some items
update item / add(item)
if similarity(item, content) > T_1
?
3) full cache
update item / put new item in place of oldest one
if similarity(item, content) > T_2 > T_1
?

Fig. 2. Principles of similarity cache filling.

Such a general binding propagation principle can be specialized in several different ways. One approach is $L3$-agnostic, with each node relying only on $L2$-based neighborhood information to decide where to propagate the message when its cache is still empty (i.e., the node does not know remote nodes other than its $L2$ neighbors). In the second approach, nodes may only rely on the information available at the application layer and are both $L3$- and $L2$-agnostic. Such an approach implies that, at startup, each node has to know in advance (or obtain from a known source) a limited number of other nodes. The third approach is $L2$-agnostic, but takes advantage of $L3$-specific information. In Sections II-E and II-F, we describe two binding propagation schemes for DINAS that follow the third approach.

E. RPL-UpDown

Assuming that each node runs RPL and knows the addresses of its preferred parent and children in the RPL DODAG, we propose an RPL-aware binding propagation protocol for DINAS denoted as RPL-UpDown. Fig. 3 illustrates the operational principle of RPL-UpDown.

After having joined the network and selected its preferred parent in the DODAG, a node starts periodically publishing its name, through a notification sent toward the DODAG root. The root then propagates the notification downward to a node chosen on the basis of the similarity of cached names.

More precisely, a node forwards the name notification to the neighbor from which it has received the most similar name to the one being notified. Fig. 4 shows an example in which node A sends a request for "00000100" to node B, instead of

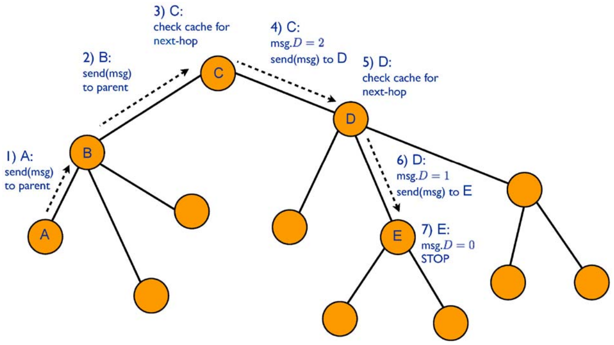
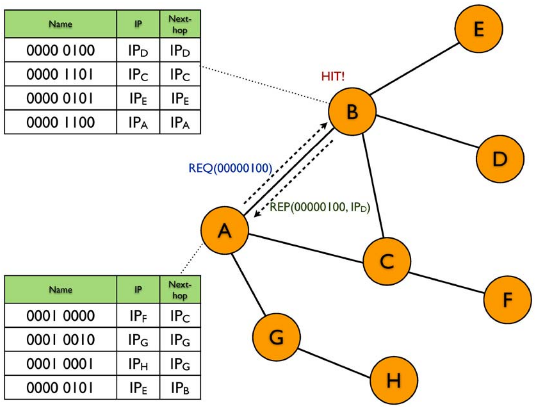


Fig. 3. RPL-UpDown propagation scheme for DINAS.



Name	IP	Next-hop
0000 0100	IP_D	IP_D
0000 1101	IP_C	IP_C
0000 0101	IP_E	IP_E
0000 1100	IP_A	IP_A

Name	IP	Next-hop
0001 0000	IP_F	IP_C
0001 0010	IP_G	IP_G
0001 0001	IP_H	IP_G
0000 0101	IP_E	IP_B

Fig. 4. Example of name resolution in RPL-UpDown.

C, because, according to the contents of *A*'s cache, *B* previously provided the most similar name ("00000101" associated with node *E*). As the cache aggregates similar names, node *B* has also name "00000100" in its cache, provided by node *D*. Thus, node *B* sends a reply to node *A* with the requested binding (00000100, *D*) and *A* updates consequently its cache.

The consequence of the propagation is that DINAS becomes more and more effective for increasing number of propagated names, because caches become more and more name-specific. Thus, a name notification and the related requests are forwarded toward the same node.

The notification propagates further on for *D* hops. Periodic name publication supports the name resolution process, as not requested names may disappear from the network, because of the content replacement policy for the cache. The RPL-UpDown scheme also caches replies, always respecting the similarity principles illustrated in Fig. 2.

The propagation of name resolution requests follows the same approach. When a node receives a name resolution request, it first checks if its name matches: if this is the case, the node sends a reply to the request issuer by providing its address in the name binding. Otherwise, the node checks its cache to find the name: if it does not have the binding, it propagates further the request.

As RPL-UpDown takes advantage of the DODAG structure built by RPL, it does not require to build and maintain its own overlay topology for name propagation and request resolution.

F. RPL-DHT

Another strategy is RPL-DHT,[3] whose principle is illustrated in Fig. 5. The upward propagation (up to the root) is similar to RPL-UpDown, but the root then propagates the notification downward to the child whose name is the most similar to the one to be published. The name is then further propagated downward in the DODAG, always applying the same principle.

The propagation stops at a node that does not have a neighbor with a name closer than its own to the name to be published. If the DODAG topology does not change over time, it is sufficient to store the name at the final destination, i.e., the last node that receives the name.[4] Otherwise, it is prudent to store the name also along the notification path (excluding the DODAG root, which is crossed by all paths), using the similarity cache approach described in Section II-C. Another obvious approach to face possible topology modifications is to periodically repeat the notification process.

The propagation of a name resolution request follows the same principle of a notification until it reaches the searched node or the final node in charge of the name resolution.

Like RPL-UpDown, RPL-DHT also takes advantage of the DODAG structure built by RPL, but it does not require to build and maintain its own overlay topology for name propagation and resolution. However, in RPL-DHT, message forwarding is only based on topological and naming considerations, not taking into account the content of the similarity cache.

III. Performance Evaluation

We have implemented DINAS in C on Contiki v2.7.[5] The developed code is available on GitHub.[6] The evaluation has been carried out with a hybrid simulation/emulation approach using Cooja to run the real code on emulated TMote Sky[7] devices in several different network topologies. Simulations have been executed on a server provided with 2 Intel Xeon Processors E5-2640 v3 at 2.60 GHz, 64GB of RAM and Ubuntu operating system. Every simulation has been executed ten times (using different seeds of the Cooja random number generator). By averaging over the obtained values, we drew the performance figures presented below. Moreover, DINAS has been also evaluated on a real testbed, as described in Section III-C.

In the considered scenarios, we have used ContikiMAC at layer 2 and IPv6 and RPL at layer 3. Nodes are organized in a DODAG, one of them (denoted as node 1) being the DODAG root. As mentioned above, all nodes are configured as TMote Sky. We also use the unit disk graph medium distance loss model with transmission range 100 m, interference range 120 m, and RX and TX success ratios set to 1. These assumptions mean that single-link (point-to-point) transmissions are

[3] DHT stands for *Distributed Hash Table*.

[4] At the final destination, the name is stored independently of its similarity with the names already stored in the cache.

[5] [Online]. Available:http://www.contiki-os.org/start.html

[6] [Online]. Available:https://github.com/ardarico/contiki/tree/dinas

[7] [Online]. Available:http://www.snm.ethz.ch/Projects/TmoteSky

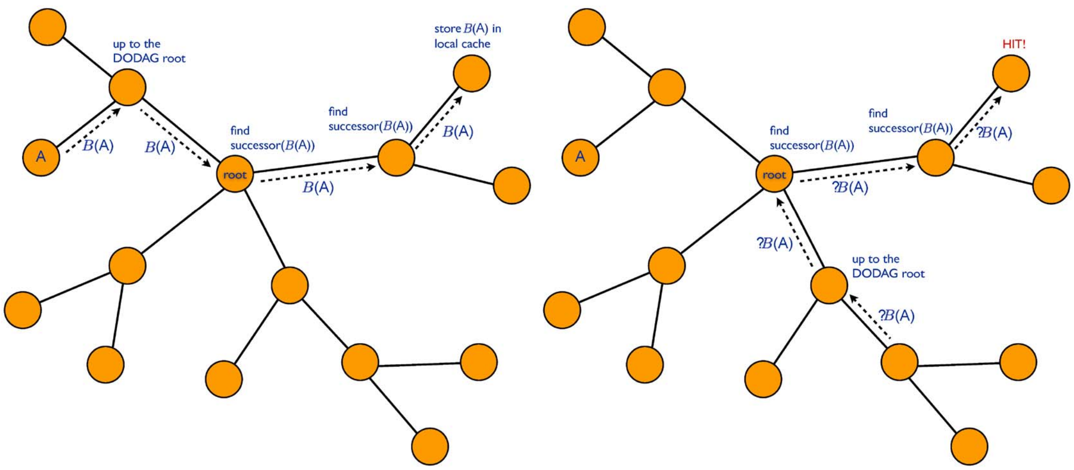


Fig. 5. RPL-DHT propagation scheme for DINAS: name notification (left) and lookup (right).

perfect, but collisions are always possible (with Cooja and Contiki, the whole stack is simulated/emulated including collisions/interference at the PHY/MAC layer). In particular, collisions are more likely at the sink and this is exacerbated in the centralized approach, where all messages are sent to the sink. The decentralized approach is less affected, especially when the maximum cache size C increases.

All nodes run two processes: one to generate and send DINAS notifications and requests and the other one to handle incoming messages. Each node builds its own name as a BF with $m = 40$ bits and $k = 7$ hash functions—these are the optimal parameters for a node description with $n = 4$ keywords (leading to $p_{f+} = 8 \cdot 10^{-3}$). More specifically, such a BF is filled with the following keywords.

1) networkService:_dinas._udp.local.
2) application:TemperatureMonitoring or application:CO2Monitoring.
3) space:Floor1 or space:Floor2.
4) space:Room-*loc*, where *loc* is an integer in the set $\{1, \ldots, N\}$, where N is the number of nodes.

Each room has its own specific *loc* value. The application associated with the node is: temperature monitoring, if *loc* is even; CO2 monitoring, if *loc* is odd. The floor is: 1 if $loc \leq N/2$; 2 if $loc > N/2$. In this way, we ensure that names are unique and sufficiently diversified. Moreover, it is possible to generate requests for names that exist for sure, allowing to measure the percentage of request hits in a consistent way.

The payload (i.e., the DINAS message) is 23 byte long: 5 bytes for the BF name, 16 bytes for the IPv6 address of the message issuer, 1 byte for configuration purposes (including the value of D), and 1 byte for the associated request number (used when the message is a reply, for performance measurement purposes). The actual DINAS implementation uses a payload of 24 bytes, because of the C struct padding. Such a payload size is very small with respect to the mDNS ADMC enhanced messages that are 82 byte long [3]. In theory, the payload limit to avoid fragmentation is $127 - 38 = 89$ bytes, where 38 is obtained as the sum of: 6 bytes for the PHY layer, 23 bytes for the 802.15.4 header (with checksum), and 9 bytes for 6LoWPAN compressed header, plus IPv6 and UDP. The total message size in the DINAS scenarios presented in this section is $38 + 24 = 62$ bytes.

A node publishes its name every T (dimension: [min]) with a notification message. In between, a node sends a request every τ min for a randomly generated name (among those that are possible and excluding its own name). A node sends M messages, of which $\lfloor \tau M/T \rfloor$ are notifications and $M - \lfloor \tau M/T \rfloor$ are requests. T has a low impact on the hit ratio *(HR)* as the network is static. In our tests: $T = 20$ min, $\tau = 2$ min, and $M = 20$ messages. Therefore, two notifications and 18 requests are sent by each node over a simulated period of about 40 min. Other tunable parameters are: C (i.e., the number of items that can be stored in the cache), D (i.e., in RPL-UpDown, the number of hops from the DODAG root), the cache thresholds T_1 and T_2 (defined in Section II-C). In the following, when we set values for C and D, we do not specify their units of measure to simplify the notation. All nodes store replies in the cache with the exception of the DODAG root (because its cache is targeted by all the notifications, which completely fill it). Once a name has been found by a node in its cache, the request is not forwarded, thus saving bandwidth and energy. Such a rule applies to this specific scenario, where requests target full names and no name is equal to another one.

We consider the following performance indicators.

1) *HR:* The percentage of name hits with respect to sent requests.
2) *Local HR (LHR):* The percentage of locally fulfilled requests with respect to the total number of hits.
3) *Average Response Time (ART):* The average time between a request and the corresponding reply (taking into account only the requests for which there is a reply).
4) *Total Traffic (TT):* The total number of "send()" operations for notifications and requests executed by the nodes in the network.
5) *Average Cache Occupation (ACO):* The average number of cached bindings per node.
6) *Max Cache Occupation (MCO):* The maximum number of cached bindings per node.

All indicators are averaged across all the nodes in the network.

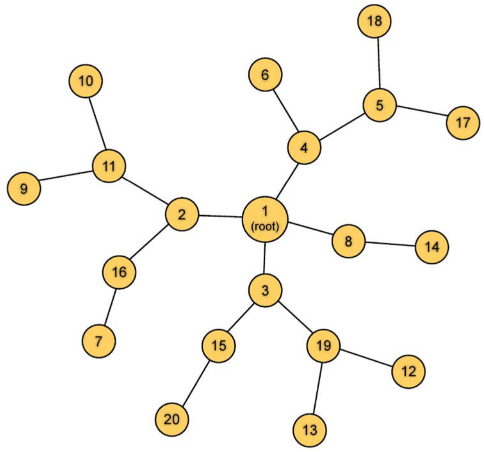


Fig. 6. Example DODAG with $N = 20$ nodes and $R_{\max} = 3$.

A. Small Networks

In the initial evaluation, we have used networks with $N = 20$ nodes, occupying an area of 500×500 m. The following results refer to a network in which the DODAG root is placed in the middle of the area (Fig. 6). The highest rank for this network is $R_{\max} = 3$. We have also tested cases in which the DODAG root is located on the border of the area, thus resulting in DODAGs with the highest rank $R_{\max} = 6$. We omit the related results, as the only difference is that the optimal value of D for RPL-UpDown is higher, because the DODAG rank is higher.

We have compared DINAS with RPL-UpDown to a centralized solution in which the DODAG root acts as a unique name service. Thus, the DODAG root has a cache sufficiently large (of size $C \geq N - 1$, to store the names of all other nodes). Communications between the DODAG root and the other nodes are only direct UDP unicasts. Notifications and requests can only be processed by the DODAG root, which is the only node able to send replies—as it is the only node that performs name caching. *TT* of the centralized solution can be analytically computed given the topology of the DODAG. Indeed, if the node rank is R, each notification and request to the DODAG root will cause R send() executions. We use the following equation:

$$TT = M \sum_{i=1}^{R_{\max}} i \cdot N_i \tag{4}$$

where $R_{\max}$ is the highest rank in the DODAG and N_i is the number of nodes with rank i. For the DODAG in Fig. 6, where $R_{\max} = 3$, $N_1 = 4$, $N_2 = 7$, $N_3 = 8$, since $M = 20$, it follows that $TT = 840$.

Another important aspect to investigate is how caches are filled. We analyzed the distribution of node names and the *self-similarity* of each cache, after a simulated time sufficiently long to guarantee that the caches are filled and their content is almost “stable”—we refer to this situation as the “steady-state.” Self-similarity is a measure of how much the names stored in one cache are similar. Cached names can be seen as a binary matrix. If the ith column has 70% of 0s and 30% of 1s, then the partial self-similarity index of the ith column (denoted as $PSSI_i$) is 0.7. The self-similarity index (*SSI*) is computed as the arithmetic average of the *PSSI*s. It is not possible to have $SSI = 1$, as the cache cannot contain copies of the same name (it could happen only if there were nonunique node names, but this is not the case of the simulated scenarios).

TABLE I
DINAS With RPL-UpDown: *HR* and *ACO* as a Function of (T_1, T_2), When $(C, D) = (9, 2)$

		T_1			
		30	50	70	90
T_2	50	95.13%, 8.99	95.13%, 8.99	/	/
	70	95.58%, 8.99	95.58%, 8.99	91.41%, 8.61	/
	90	96.13%, 8.98	96.13%, 8.98	94.91%, 8.74	35.2%, 1.37

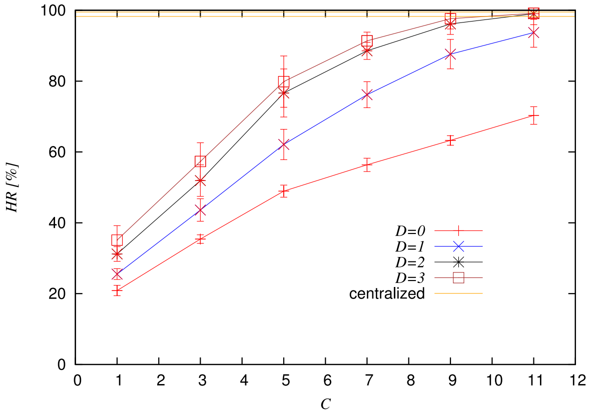


Fig. 7. DINAS with RPL-UpDown: *HR* as a function of C, with $T_1 = 30$ and $T_2 = 90$. Various values of D are considered.

Table I shows *HR* as a function of T_1 and T_2, when $(C, D) = (9, 2)$. The reader can observe that, among the tested configurations, those with $(T_1 < 70, T_2 \geq T_1)$ lead to the same values of *HR*. When T_1 is too high, it is difficult to fill the cache. If a cache is not filled, T_2 is non influential and, of course, name replication is reduced: hence, *HR* reduces. In the remainder of this section, the $(T_1, T_2) = (30, 90)$ configuration is assumed.

Fig. 7 shows *HR* as a function of C, considering various values of D. The best result ($HR = 99.16\%$) is given by the configuration with $C = 11$ and $D = 3$. It is comparable with the performance result of the centralized solution ($HR = 99.41\%$).

In Figs. 8 and 9, *TT* and *ART* are shown as functions of C, respectively. In both figures, various values of D are considered. The obtained results show that the centralized solution can be easily outperformed, in terms of *TT* and *ART*, by DINAS. The best configuration of DINAS appears to be the one with $C = 11$ and $D = 3$, leading to $TT = 565$ send() operations and $ART = 314$ ms. *TT* and *ART* of the centralized solution are higher, i.e., 840 send() operations and 368 ms, respectively. Having a large cache has pros and cons: it allows to reduce messaging, as the probability of already having the information in the local cache increases, but it requires more memory resources.

In Fig. 10, *LHR* is shown as a function of C, considering various values of D. As expected, the larger the cache, the higher the percentage of locally fulfilled requests, which

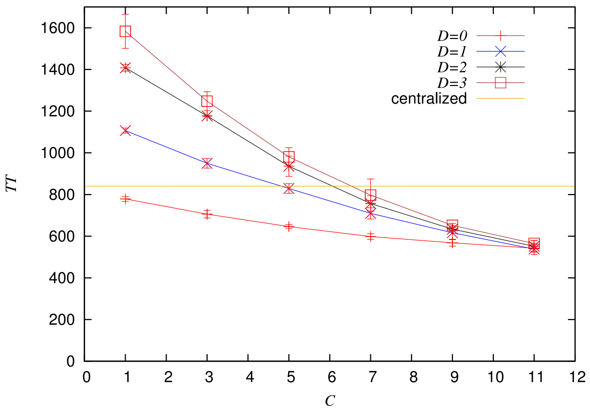


Fig. 8. DINAS with RPL-UpDown: *TT* as a function of *C*, with $T_1 = 30$ and $T_2 = 90$. Various values of *D* are considered.

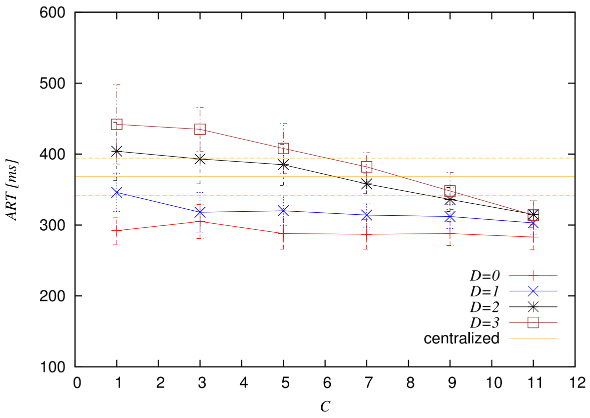


Fig. 9. DINAS with RPL-UpDown: *ART* as a function of *C*, with $T_1 = 30$ and $T_2 = 90$. Various values of *D* are considered.

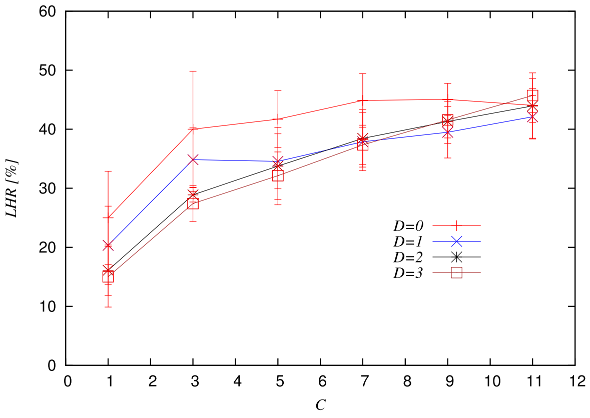


Fig. 10. DINAS with RPL-UpDown: *LHR* as a function of *C*, with $T_1 = 30$ and $T_2 = 90$. Various values of *D* are considered.

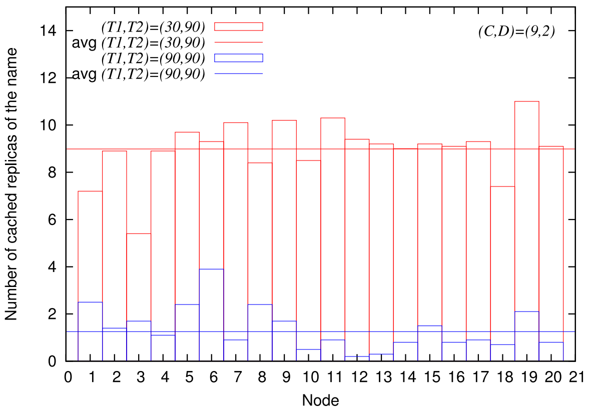


Fig. 11. DINAS with RPL-UpDown: Steady-state name distribution in caches, with $(C, D) = (9, 2)$, comparing $(T_1, T_2) = (30, 90)$ with $(T_1, T_2) = (90, 90)$.

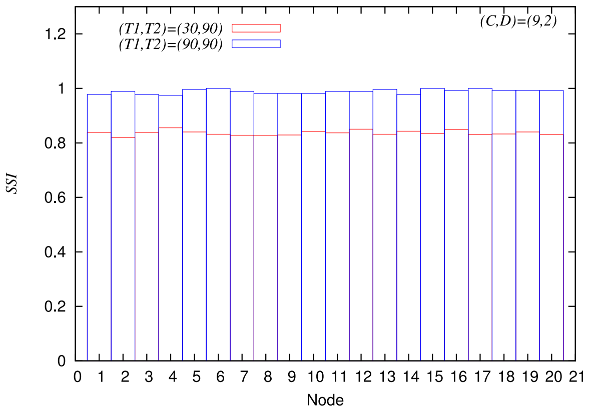


Fig. 12. DINAS with RPL-UpDown, $N = 20$: steady-state *SSI* of the caches, with $(C, D) = (9, 2)$, comparing $(T_1, T_2) = (30, 90)$ with $(T_1, T_2) = (90, 90)$.

becomes more evident when *D* increases. The similarity caching policy prevents the cache from being filled with all names even if the cache is sufficiently large to contain all names. Therefore, increasing *C* does not imply to have *LHR* → 100%.

Fig. 11 presents the numbers of cached replicas of a given name across the nodes. The proposed graphs refer to the $(C, D) = (9, 2)$ configuration, comparing $(T_1, T_2) = (30, 90)$ with $(T_1, T_2) = (90, 90)$. The results obtained show that, at the steady state, the names are almost evenly distributed in the caches. We recall that values are averaged over ten simulation runs. It can be observed that, as expected, when T_1 is too large, it is hard to completely fill the caches.

Fig. 12 shows the *SSI*s across the nodes. As in Fig. 11, the proposed graphs refer to the $(C, D) = (9, 2)$ configuration, comparing $(T_1, T_2) = (30, 90)$ with $(T_1, T_2) = (90, 90)$. Our results show that, once in the steady state, the *SSI* is high (above 0.8) for all caches. As expected, when $T_1 = 90$, *SSI* is higher—caches are less filled, but cached names are more similar to each other. However, less filled caches lead to a performance degradation with respect to the case with completely filled ones (as shown in Table I).

For DINAS with RPL-DHT, $HR = 100\%$ is always guaranteed (if the topology is stable, which is the case we are considering), with $ACO = 3.73$. The best configurations of RPL-UpDown and RPL-DHT are compared in Table II. We have studied RPL-DHT without and with redundancy: in the former case, names are stored at the destination node only; in the latter case (denoted as RPL-DHT-R for simplicity), names are stored all along the notification path with the exclusion of

TABLE II

COMPARISON BETWEEN THE BEST RPL-UPDOWN AND RPL-DHT CONFIGURATIONS, WHEN $N = 20$. RPL-UPDOWN IS WITH $(C, D) = (11, 3)$ AND $(T_1, T_2) = (30, 90)$. RPL-DHT-R MEANS RPL-DHT WITH REDUNDANCY, I.E., NAMES ARE CACHED ALONG THE NOTIFICATION PATH, WITH $(T_1, T_2) = (60, 90)$

Strategy	HR [%]	TT	$ART[ms]$	LHR [%]	ACO	MCO
RPL-UpDown	99.16	565	314	45.71	10.93	11
RPL-DHT	100	1235	494	2.38	1	7
RPL-DHT-R	100	980	377	9.72	2.83	10

TABLE III

SUMMARY (IN TERMS OF ASYMPTOTIC BEHAVIORS) OF THE PERFORMANCE RESULTS ILLUSTRATED IN FIGS. 13–17

Metric	Dependency on N	Reference Figure
HR	Linearly decreasing	13
MCO	Constant	14
ACO	Constant	15
ART	Constant	16
TT	Linearly increasing	17

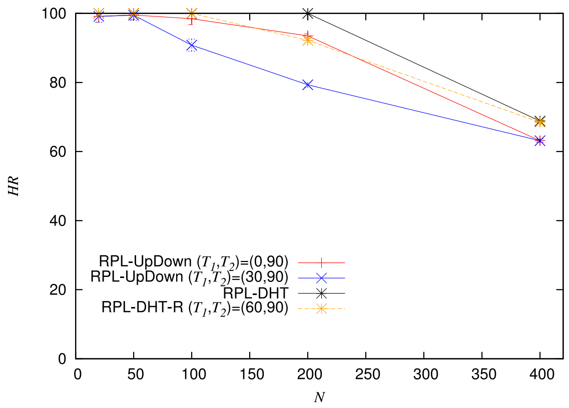


Fig. 13. RPL-UpDown and RPL-DHT: *HR* versus *N*.

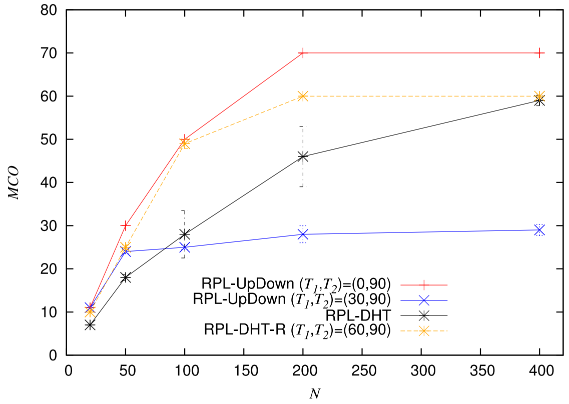


Fig. 14. RPL-UpDown and RPL-DHT: *MCO* versus *N*.

the sink unless it is the final destination for the name. Indeed, as the sink is traversed by all notification paths, its cache would be completely filled very quickly, leading to the possible overwriting of names it would be the only one responsible for. For both RPL-UpDown and RPL-DHT, we set $(T_1, T_2) = (30, 90)$. Instead, for RPL-DHT-R, we set $(T_1, T_2) = (60, 90)$, to reduce unnecessary cache filling. The reader can observe that RPL-UpDown has a much higher *ACO* value, but also lower *TT* and *ART* values (because of the higher *LHR* value). Therefore, if the memory is not a constraint, RPL-UpDown is to be preferred to RPL-DHT and RPL-DHT-R. Otherwise, RPL-DHT is the best choice, providing $HR = 100\%$ with considerably reduced memory requirements.

B. Large Networks

Both RPL-UpDown and RPL-DHT strategies have been tested in networks with increasing number of nodes, namely: 50, 100, 200, and 400. Tables IV–VII illustrate the best performance of the strategies, considering stable networks. In this case as well, we consider RPL-DHT without and with redundancy. Regarding RPL-UpDown, we have observed that, when N increases, if there is a constraint on the similarity cache (i.e., $T_1 > 0$, meaning that not all items will be cached), then *MCO* converges to a value lower than C. As a consequence, *HR* decreases when N increases, as the caches are not fully exploited. Instead, if $T_1 = 0$, *MCO* becomes equal to C and *HR* has acceptable values (~99%), provided that C and τ are suitably large. The performance of RPL-DHT, instead, is only affected by τ. It is worth noting that, regarding RPL-UpDown, $C = 70$ was the maximum reachable limit with TMote Sky devices. With larger values of C, the performance of RPL-UpDown, with $N \geq 200$ nodes, in terms of *HR*, would certainly be better. However, most names would be cached by the sink and the benefit, with respect to the centralized solution, would not be obvious. With RPL-DHT-R, instead, the maximum limit for the cache size is $C = 60$ for TMote Sky devices: such a cache size is not sufficient to guarantee $HR = 100\%$. With $N = 400$, $\tau = 2$ min, the request frequency is too high, considering that the requests are all driven toward the sink, regardless of the strategy. With $\tau = 4$ min, the performance of RPL-DHT is acceptable ($HR \sim 95\%$). In Figs. 13–17, the performance, in terms of *HR*, *MCO*, *ACO*, *ART*, and *TT* as functions of N (with $\tau = 2$ and $C = 70$), is investigated considering different strategies and configurations. The results, in terms of asymptotic behaviors of the considered performance metrics, are summarized in Table III. It can be concluded that the most "critical" performance metrics are *HR* and *TT*.

We have also studied the behavior of the considered strategies, in the presence of a failure probability $p_{\text{fail}} > 0$ at each node. In Fig. 18, we show how *HR* is affected by increasing values of p_{fail}, considering the RPL-UpDown and RPL-DHT-R configurations that guarantee the best performance. As expected, RPL-UpDown appears to be more robust than RPL-DHT-R.

In Fig. 19, we compare the distributions of cache sizes, still considering the best RPL-UpDown and RPL-DHT-R

TABLE IV
Comparison Between the Best RPL-UpDown and RPL-DHT Configurations, When $N = 50$. RPL-UpDown Is With $(C, D) = (30, 5)$ and $(T_1, T_2) = (30, 90)$. RPL-DHT-R Is With $(T_1, T_2) = (60, 90)$

Strategy	HR [%]	TT	$ART[ms]$	LHR [%]	ACO	MCO
RPL-UpDown	99.45	2814	455	23.67	15.638	24
RPL-DHT	100	3805	668	2.07	1	18
RPL-DHT-R	100	3478	559	5.82	3.79	25

TABLE V
Comparison Between the Best RPL-UpDown and RPL-DHT Configurations, When $N = 100$. RPL-UpDown Is With $(C, D) = (50, 15)$ and $(T_1, T_2) = (0, 90)$. RPL-DHT-R Is With $(T_1, T_2) = (60, 90)$

Strategy	HR [%]	TT	$ART[ms]$	LHR [%]	ACO	MCO
RPL-UpDown	98.45	9022	663	15.64	20.05	50
RPL-DHT	100	11218	950	0.76	1	28
RPL-DHT-R	100	9553	783	4.66	5.55	49

TABLE VI
Comparison Between the Best RPL-UpDown and RPL-DHT Configurations, When $N = 200$. RPL-UpDown Is With $(C, D) = (70, 15)$ and $(T_1, T_2) = (0, 90)$. RPL-DHT-R Is With $(T_1, T_2) = (60, 90)$

Strategy	HR [%]	TT	$ART[ms]$	LHR [%]	ACO	MCO
RPL-UpDown	93.48	29721	807	10.25	22.63	70
RPL-DHT	100	30468	1125	0.46	1	46
RPL-DHT-R	92.19	27546	960	3.36	6.97	60

TABLE VII
Comparison Between the Best RPL-UpDown and RPL-DHT Configurations, When $N = 400$. RPL-UpDown Is With $(C, D) = (70, 15)$ and $(T_1, T_2) = (0, 90)$. RPL-DHT-R Is With $(T_1, T_2) = (60, 90)$

Strategy	HR [%]	TT	$ART[ms]$	LHR [%]	ACO	MCO
RPL-UpDown ($\tau = 2$ min)	63.15	77733	820	5.08	15.04	70
RPL-UpDown ($\tau = 4$ min)	85.9	20649	808	2.33	8.14	70
RPL-DHT ($\tau = 2$ min)	68.82	62910	1086	0.33	0.95	59
RPL-DHT ($\tau = 4$ min)	94.9	31860	1141	0.26	0.96	59
RPL-DHT-R ($\tau = 2$ min)	68.48	59826	920	2.24	6.65	60
RPL-DHT-R ($\tau = 4$ min)	85.16	30085	966	1.84	6.47	60

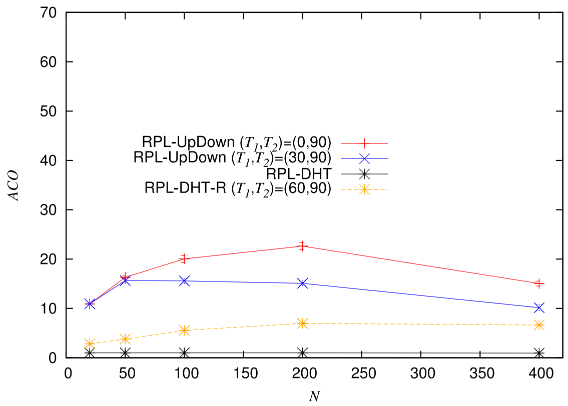


Fig. 15. RPL-UpDown and RPL-DHT: ACO versus N.

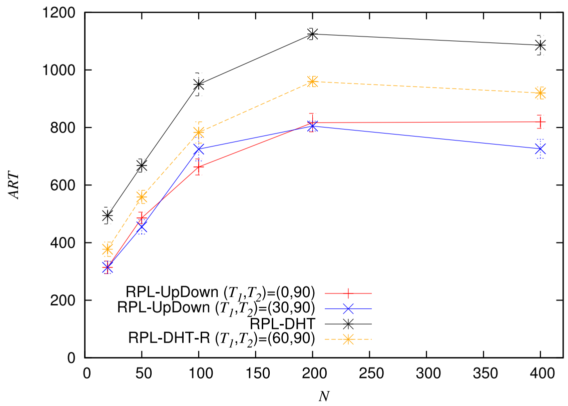


Fig. 16. RPL-UpDown and RPL-DHT: ART versus N.

configurations. We observe that in RPL-DHT-R the majority of nodes cache a few items. In RPL-UpDown, instead, the distribution is Gaussian-like and some nodes, namely the DODAG root and its children, have full caches.

In conclusion, the DODAG size should take into account the characteristics of the devices. With TMote Sky devices, the ideal DODAG size is between $N = 100$ and $N = 200$. If the network is not affected by node failures, RPL-DHT is the best solution in terms of memory occupation, at the expense of higher ART and TT. RPL-UpDown is more robust to node failures and more efficient in terms of ART and TT, as caches are better exploited (at the expense of higher ACO). RPL-DHT-R is an interesting tradeoff between RPL-DHT and RPL-UpDown.

C. Testbed Evaluation

In this section, we describe the experimental performance evaluation carried out at the open and large scale FIT IoT-LAB[8] infrastructure spread across different sites in France.

[8] [Online]. Available: https://www.iot-lab.info

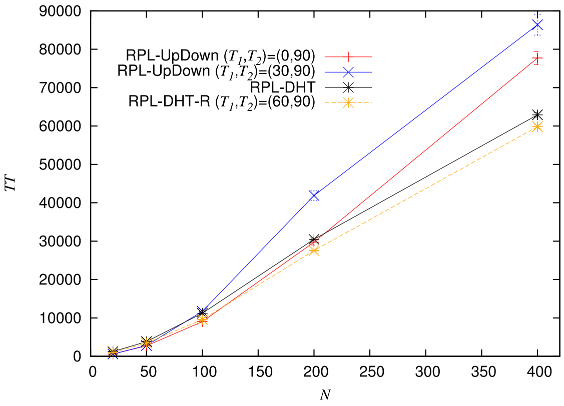


Fig. 17. RPL-UpDown and RPL-DHT: *TT* versus *N*.

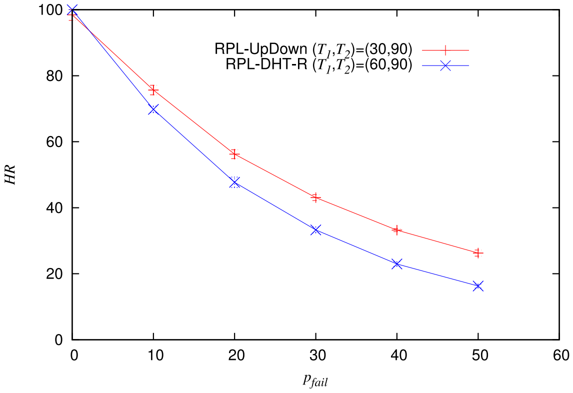


Fig. 18. RPL-UpDown and RPL-DHT: *HR* versus p_{fail}.

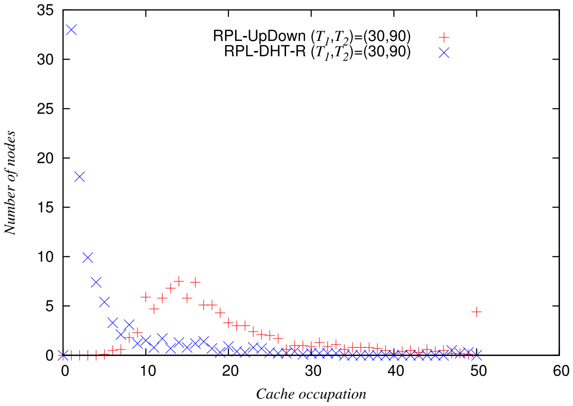


Fig. 19. RPL-UpDown and RPL-DHT: cache distribution (case $N = 100$).

The main motivation behind real experiments is to complement simulation results by observing how DINAS behaves in a realistic dynamic radio environment and how it copes with the resulting RPL topology changes. We use two different testbeds, one in Strasbourg and another one in Lille, to benefit from their heterogeneity in terms of physical topology and radio environment. While the original DINAS code could only run on TMote Sky nodes, since COOJA only emulates this platform, the code was then adapted to run on M3 Open Nodes[9] (ARM Cortex M3 with 64 KB RAM and 802.15.4 ATMEL radio), which are the most up-to-date devices deployed on IoT-LAB. Considering that Contiki v2.7 for M3 nodes was already made available by the IoT-LAB team[10] and that DINAS runs at the application layer, only a few adjustments were necessary to port the code. Most of the efforts were focused on automating experiments and getting as close as possible to previous COOJA topologies by tuning radio parameters and choosing relevant nodes to avoid too unstable RPL DODAGs. The IoT-LAB DINAS code is available at https://github.com/bobib22/{contiki-iotlab-dinas,iot-lab-dinas} for possible reproduction of the experiments. The repository also contains the logs of the detailed experimental results as well as the involved nodes and their radio configurations.

TABLE VIII
IoT-LAB EXPERIMENT PARAMETERS

Topology	20 nodes
Average path length	1.68 (Strasbourg), 2.04 (Lille)
Average max rank	3.86 (Strasbourg), 4.71 (Lille)
Radio	channel 26, Tx power -17dBm, Rx RSSI threshold -69dBm
DINAS	$C = 11$, $D = 4$, $T_1 = 30$ (60 for DHT-R), $T_2 = 90$
Traffic	Same as COOJA simulations except for message frequency 1 message every 10s
Number of experiments	40 for each DINAS propagation scheme (RPL-{UpDown, DHT, DHT-R}) and each site
Experiment duration	6 min

Table VIII summarizes the most characteristic parameters of our experiments. To investigate multihop topologies, we have changed the default radio transmission power and packet reception RSSI. Table IX presents the detailed results obtained by averaging over 40 experiments for each DINAS propagation strategy and for each site. The traffic pattern is similar to the one used in COOJA simulations, except for the message period reduced to 10 s, which allows one to run a significant number of experiments in an acceptable time while, at the same time, testing the robustness of DINAS in more strict traffic conditions. Therefore, in addition to the time required by nodes to boot and to be integrated in the RPL DODAG, the experiment duration drops from 40 min (simulated time) in COOJA to 6 min (real time) on IoT-LAB.

In Fig. 20, we show tw illustrative topologies formed during real experiments (the one on the left in Strasburg and the one on the right in Lille). The figure provides insights on the dynamically formed network topologies during such experiments and to which extent these topologies compare with those generated by COOJA. The DODAGs shown in Fig. 20 are obtained by overlapping all the DODAGs formed during the experiment. Therefore, each edge refers to a topological change either due to a (local or global) DODAG repair or to a change of a preferred parent advertising a better rank (according to the ETX metric). Despite being incomplete (i.e., no details about link duration or potential disconnection from the parent), the illustrative representation in Fig. 20 shows, for

[9] [Online]. Available: https://www.iot-lab.info/hardware/m3/

[10] [Online]. Available: https://github.com/iot-lab/contiki

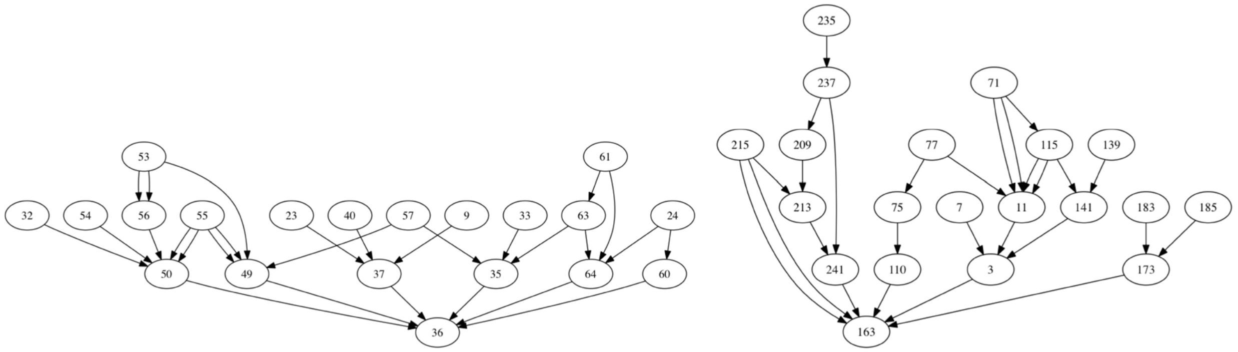


Fig. 20. DODAGs formed during two illustrative experiments: one on the Strasbourg platform (left) and the other one on the Lille platform (right).

TABLE IX
IoT-LAB Experimental Results With $N = 20$

Site	Strategy	HR [%]	TT	$ART[ms]$	LHR [%]	ACO	MCO
Strasbourg	RPL-UpDown	98.56	514	193	42.83	10.86	11
	RPL-DHT	96.08	1021	348	3.41	1	4.0
	RPL-DHT-R	97.12	762	260	10.22	2.41	11.0
Lille	RPL-UpDown	94.09	645	223	45.10	10.78	11.0
	RPL-DHT	89.03	1048	446	4.39	1	7
	RPL-DHT-R	94.68	909	434	11.06	3.18	9.0

TABLE X
Detailed Experimental Statistical Characterization of Packet Losses for $N = 30$ (IoT-LAB Strasbourg Testbed)

Strategy	$Resolved$ $requests$ [%]	$Lost$ $requests$	$Lost$ $replies$	$L2-$ $dropped\ frames$	$L3-$ $no\ route$	$L5-$ $deadend$	$Global$ $repairs$
RPL-UpDown	88.00	34.69	27.19	204.62	12.67	7.74	6.36
RPL-DHT	62.74	107.69	90.50	327.48	42.60	6.10	14.67
RPL-DHT-R	74.03	71.13	66.26	280.00	34.74	4.92	10.05

instance, that node 115 in the Lille testbed has experienced three topological changes (the preferred parent is either nodes 11 or 141) during the 6-min experiment—for the sake of clarity, we have chosen to display topologies among the most stable ones observed. The number of global repairs in Lille experiments ranges from 0 to 6, with an average (over all the experiments) number of global repairs per experiment equal to 3.64.

Considering the metrics *ACO*, *LHR*, *ART*, and *TT*, the experimental results, shown in Table IX, are comparable to the results obtained with COOJA (Table II). The slight differences can be explained by the differences in the involved topologies. Considering, for example, the *ART* metric, it can be seen that, in both cases, it is an increasing function of the average path length of the topology.

The main difference between simulations and experiments, for 20-node topologies, is observed in terms of *HR*. Despite a performance degradation in real experiments with respect to simulations, DINAS scales well since it still has high values of *HR* (over 94%) except for the RPL-DHT experiments on the Lille platform, where *HR* drops to approximately 89%.

This performance degradation comes from the increase of DINAS traffic (in the experiments, each node generates one message every 10 s instead of one message per minute) and more realistic radio interference conditions (with respect to the UDG distance loss model assumed in simulations) that lead to a larger number of collisions and, thus, to a larger number of packet drops, which in turn, creates topology instabilities.

We have run further experiments in harsher conditions (i.e., with higher spatial node density and a larger number of nodes) on the Strasbourg platform to get more insights into the considered name propagation performance. Table X presents a detailed statistical characterization of the packet losses and their distribution at each layer. *Resolved requests* (unlike *HR*) is a parameter corresponding to the percentage of requests for which a response is returned to the requester. The causes of unresolved requests can be outlined as follows, in order of the decreasing relevance.

1) At layer 2 *(L2 − dropped frames):* Drop of frames at the MAC layer.
2) At layer 3 *(L3 − no route):* Route missing when sending or relaying a notification/request/reply.
3) At layer 5 *(L5 − deadend):* Unresolved requests that do not reach a node with the requested name in its cache.

We now comment in more detail on the losses layer by layer.

First, *L2-dropped frames* correspond to the number of unicast packets (e.g., DINAS UDP datagrams or Unicast RPL packets like DAOs) that are dropped every time the CSMA/CA protocol (that handles retransmissions at layer 2) reaches the maximum number of retransmission attempts.

The *L3-no route* situation happens when a node wants to send or relay a DINAS message, but the next hop of the default route is missing. This temporary loss of a RPL parent or of a route is related to RPL global repairs as well as to the degradation of the ETX metric of the link with the preferred parent (which leads to an unacceptable rank and parent removal).

Finally, *L5-deadends* have different origins. First, the name might have been overwritten or not stored in the caches along the path followed by the requests. Second, the DINAS routing strategy might not route the request toward the right caches. Finally, the DINAS routing information might not be up-to-date with respect to RPL reconfigurations.

Table X confirms that the higher traffic generated by the DINAS propagation strategy degrades the performance. Therefore, RPL-DHT exhibits, averaging over 40 runs, 27% less resolved requests than RPL-UpDown. RPL-DHT also shows 127 more drops, 30 more *L3-no route* losses, and 8 more global repairs than RPL-UpDown. *L5-deadends* have little influence on performance degradation and are approximately the same in the three strategies.

DINAS packet drops happen at few nodes, especially at those belonging to higher density regions, i.e., at the sink and at its one-hop neighbors, where caches are fuller due to converging traffic. Another factor contributing to the performance degradation is related to global repairs triggered when inconsistencies are detected in the DODAG that reset the trickle timer of all nodes and lead to a non-negligible localized increase of signaling traffic (DIOs and DAOs) in the network. Thanks to redundancy, RPL-UpDown and RPL-DHT-R experience less collisions (and drops) since names are resolved earlier, when possible, in the DODAG, thus avoiding the bottleneck of the sink for both requests and replies. On the opposite, the RPL-DHT propagation scheme does not benefit from redundancy and, therefore, is less robust in the presence of a drastic traffic increase at the sink or in more mobile scenarios.

IV. Related Work

A. Naming

Concerning the general problem of naming, Balakrishnan *et al.* [8] proposed to use flat names for network elements. Such an approach is already used to name nodes in structured peer-to-peer (P2P) networks. However, if flat names are not obtained from the descriptions of the individuals, they are not flexible and, as a consequence, not useful. In their seminal work about content-centric networking (CCN), Jacobson *et al.* [9] proposed a naming scheme in which names are hierarchically structured with components encoded one by one. Such an approach is adopted, for example, in the architecture recently proposed by Waltari and Kangasharju [10]. However, as Andreolini and Lancellotti [11] have shown, it is possible to map a resource descriptor composed by several keywords to a unique machine-readable flat name, by means of a BF. A comparison between hierarchical and flat naming approaches has been recently proposed by Adhatarao *et al.* [12], where it is concluded that using flat names is likely to be much more scalable. DINAS adopts this latter approach.

Intanagonwiwat *et al.* [13] introduced *directed diffusion*, a data-centric approach to organize interest-based interactions among nodes in WSNs. In directed diffusion, tasks are named by a list of attribute-value pairs. A task description specifies an interest for data matching the attributes. For this reason, task descriptions are called “interests.” The data sent in response to interests are also named using a similar naming scheme. A node requests data by sending interests for named data. Data matching the interest are then drawn down toward that node. Intermediate nodes can cache, or transform, data and may direct interests based on previously cached data. For each active task, the sink periodically broadcasts an interest message to each of its neighbors.

With respect to directed diffusion, DINAS supports architectures in which every node may act as a sink. Moreover, DINAS does not support only interest-based interactions—all nodes are also allowed to publish their own descriptors. In this way, interests (which coincide with name requests, in DINAS) can find matching node descriptors that have been stored in the cache of an intermediate node. In other words, to enable the interaction between the requester and the source, it is not necessary that the request arrives at the source. Furthermore, DINAS messages are compact, as node descriptors and requests are encoded with BFs. Finally, DINAS is application-independent. More precisely, the way requesters and sources interact, after they find each other, is independent of DINAS.

More recently, Yue *et al.* [14] introduced the DataClouds architecture, whose basic building blocks are called *communities* and consist of users with common interests in data and information. Users within the same community are well connected in the network, so that desired data can be efficiently collected, disseminated, and shared among them. Data dissemination is achieved via community-oriented communications, where name resolution and data routing are restricted within each community. This approach improves scalability and is suitable to be used in conjunction with DINAS (whose scalability limits have been analyzed in Section III-B).

DIAT [15] is a distributed architecture for IoT with a strong emphasis on device virtualization and semantics. The virtual object layer (VOL) plays the role of bridging the gap between the physical and the cyber world. Moreover, the VOL enables the communication with the devices. As VO hosting may be decentralized, the architecture is scalable. However, as the authors admit, not all devices may be able to run a fully functional DIAT IoT Daemon, which includes the VOL and other upper layers. Thus, a depreciated IoT Daemon may run on constrained devices. With respect to DIAT’s VOL, DINAS supports any kind of device.

Li *et al.* [16] have proposed an IoT middleware architecture over information-centric network, based on named data networking (NDN) [17]. Rooted in CCN, NDN changes the semantics of network service from delivering the packet to a given destination address to fetching data identified by a given name. An interesting future direction for this paper is to adapt DINAS to the NDN context.

The SPITFIRE EU project [18], [19] considered the following technologies.

1) Constrained Application Protocol (CoAP)[11] (a lightweight RESTful transfer protocol for accessing

[11] [Online]. Available: http://www.ietf.org/rfc/rfc7252.txt

data on constrained devices) to connect sensors to the Internet and the Web.

2) RDF,[12] OWL,[13] and SPARQL[14] to allow machines to discover and understand the semantics of the data returned by the sensors.

Even though SPITFIRE's approach is interesting, its technologies for semantic annotation and reasoning may require important computational resources, so that scalability may be guaranteed only if reasoning tasks are executed by cloud computing facilities. Conversely, the CoAP protocol allows to benefit from the advantages of HTTP, without its drawbacks, in the context of sensor networks. CoAP keeps message overhead as small as possible and may get rid of TCP complexity by providing IoT-adapted interaction primitives over UDP.

In RFID systems, there are several standards for giving names to items. The standardization effort led by GS1 has recently resulted in the EPC Tag Data Standard (TDS),[15] also known as GEN2 RFID TDS, which specifies the following format for RFID tags:

urn:epc:id:*scheme:component1.component2. ...*

This scheme defines a uniform resource name (URN) which uniquely specifies an item, while being location-independent. The prefix urn:epc:id: is fixed; the *scheme* can be selected among sgtin (trade item), sgln (location), gdti (document), gsrn (service relation, e.g., loyalty card), etc.; and *component*s are numbers whose semantics depends on a particular scheme—for example sgtin requires three components: one for the company prefix, one for the item reference, and one for the serial number.

Schmidt *et al.* [20] proposed to share RFID tags by means of the Chord DHT, where resource keys are obtained by hashing their tags. A similar approach, introduced by Fabian *et al.* [21], uses the FreePastry DHT to share document fragments (obtained by means of the Shamir secret sharing scheme), whose keys are generated with hashing functions like SHA-1.

B. Service Discovery

Traditional IP-based service discovery protocols, such as Service Location Protocol[16] and Universal Description, Discovery, and Integration,[17] are hardly applicable in constrained environments such as 6LoWPAN networks. Beside centralized solutions, distributed directory solutions also exist, but rely on a few very powerful nodes that are not always available in sensor networks.

The IETF CoRE Working Group recognizes the need for finding sensors and interacting with them without human intervention. The most advanced one is the centralized CoAP-based resource directory approach.[18] The resource directory periodically announces itself (e.g., with Zeroconf [22]) or is discovered by sensors thanks to an anycast address or a specific CoAP request. At startup, the nodes register to the discovered resource directory that, in turn, polls each registered node. The response from each node is a descriptor that includes some information such as node ID and offered services. Finally, the proxy adds the node descriptor to a local table. Thus, the resource directory must be contacted to learn about other nodes in the WSN, a possible bottleneck and a single point of failure for the system. An IETF draft[19] further discusses the issues related to CoAP, specifies guidance on how CoAP should be used in a group communication context, and details an approach for using CoAP on top of IP multicast.

Regarding directory-less approaches, three categories are considered in the literature: push, pull, and hybrid. In push models, such as DEAPspace [23], service providers proactively send service advertisements. While latency is reduced, this approach introduces a large amount of traffic, making it unsuitable for highly dynamic networks. In pull models, requests issued by clients propagate across the network. From the latency point of view, pull models are less efficient that push models. On the other hand, pull models are more suitable for dynamic environments, as they generate less traffic. Hybrid push–pull models benefit from both aforementioned approaches. ADDER [24] is characterized by periodic advertisements, which reduces latency, but generates high push overhead. NanoSD [25] attempts to minimize packet sizes and service descriptions, but like ADDER, it does not propose advanced forwarding mechanisms to minimize push traffic.

In terms of interoperability, uBonjour [26] adapts traditional mDNS and DNS-SD operations to WSNs. Despite its message compression optimization [3], it still suffers from its original drawbacks: heavy messages, high communication overhead, and its need for multicast support that is not always available or efficient in multihop WSNs.

The efficient application-layer discovery protocol (EADP) [27] is based on the hybrid push–pull model. It provides advanced forwarding mechanisms for advertisement as well as two interesting features to minimize the push traffic. The first one is the use of the trickle algorithm[20] to exponentially increase the transmission window (during the push phase) as long as neighboring nodes do not show any inconsistencies among their shared data. The second one is the systematic aggregation of service descriptions (learned from neighboring nodes or locally owned) within advertisements, to gather with an independent consistency counter for each stored service that allows to only advertise inconsistent services every transmission window. However, the authors emphasize that EADP is not limited to a single service description, but do not explain how long descriptions could fit into small packets (as only around 61 bytes are left, at application layer, in 6LoWPAN networks). Furthermore, the use of limited flooding to discover services is not convincing and the proposed performance evaluation does not help,

[12] [Online]. Available: http://www.w3.org/RDF/

[13] [Online]. Available: http://www.w3.org/TR/owl-ref/

[14] [Online]. Available: http://www.w3.org/TR/sparql11-query/

[15] [Online]. Available: http://www.gs1.org/gsmp/kc/epcglobal/tds/

[16] [Online]. Available: http://tools.ietf.org/html/rfc2608

[17] [Online]. Available: http://www.uddi.org/pubs/uddi-v3.0.1-20031014.htm

[18] [Online]. Available: http://tools.ietf.org/html/draft-ietf-core-resource-directory-01

[19] [Online]. Available: http://tools.ietf.org/html/draft-ietf-core-groupcomm-24

[20] [Online]. Available: https://tools.ietf.org/search/rfc6206

because of the lack of information about the adopted routing protocol and the network topology.

Still directory-less and based on a hybrid push–pull model, 6LoWDIS [28] is a recently proposed application layer service discovery protocol for 6LoWPAN-based networks. 6LoWDIS is built on CoAP/HTTP and requires the support of multicast routing in the network layer.

DINAS is also directory-less and relies on a hybrid push-pull model for service discovery. Its main novelties, with respect to the state of the art, can be summarized as follows.

1) The name and the service descriptions of a node coincide (to solve two problems—naming and service discovery—with one protocol).
2) Push and pull forwarding are based on unicast and driven by the content of local caches (to optimize latency and overhead).

Finally, an interesting approach to large-scale service discovery has been proposed by Cirani *et al.* [4]. Small IoT networks are connected by means of IoT Gateways that participate in a layered P2P network. Each layer is organized as a DHT, characterized by a lookup time that is a logarithmically increasing in function of the number of peers. It is worth noting that structured overlay networks, such as DHTs, are affected by non-negligible maintenance overhead. Thus, IoT Gateways can be federated in a DHT, but resource-constrained IoT nodes cannot. The RPL-DHT binding propagation protocol introduced in the current paper is as effective as traditional DHT lookup protocols ($HR = 100\%$) but, not relying on a structured overlay network, is less efficient in terms of the lookup time.

V. Conclusion

In this paper, we have presented DINAS, a novel approach for publishing and retrieving information about names and services. DINAS creates names as BFs based on node or service descriptions. We have first outlined the main principles of DINAS, then we have investigated the performance of its Contiki implementation with a hybrid simulation/emulation approach and also on a real testbed. Performance results are encouraging in various scenarios, associated with different (in shape and size) DODAGs.

Regarding future work, we would like to test alternative message propagation schemes. One of this paper directions consists in implementing a propagation strategy that takes advantage of the $L3$ topology provided by LOADng,[21] a lightweight variant of AODV.[22] Furthermore, we plan to design an $L3$-agnostic propagation scheme with nodes relying only on $L2$-based neighborhood information.

Acknowledgment

The work reflects only the authors views; the European Community is not liable for any use that may be made of the information contained herein.

[21] [Online]. Available: http://tools.ietf.org/html/draft-clausen-lln-loadng-11

[22] [Online]. Available: https://tools.ietf.org/html/rfc3561

References

[1] M. Amoretti, O. Alphand, G. Ferrari, F. Rousseau, and A. Duda, “DINAS: A distributed naming service for all-IP wireless sensor networks,” in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Istanbul, Turkey, Apr. 2014, pp. 2781–2786.

[2] A. Dunkels, B. Gronvall, and T. Voigt, “Contiki—A lightweight and flexible operating system for tiny networked sensors,” in *Proc. IEEE Conf. Local Comput. Netw. (LCN)*, Tampa, FL, USA, Nov. 2004, pp. 455–462.

[3] R. Klauck and M. Kirsche, “Enhanced DNS message compression—Optimizing mDNS/DNS-SD for the use in 6LoWPANs,” in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops (PERCOM Workshops)*, San Diego, CA, USA, Mar. 2013, pp. 596–601.

[4] S. Cirani *et al.*, “A scalable and self-configuring architecture for service discovery in the Internet of Things,” *IEEE Internet Things J.*, vol. 1, no. 5, pp. 508–521, Oct. 2014.

[5] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, “Summary cache: A scalable wide-area Web cache sharing protocol,” *IEEE/ACM Trans. Netw.*, vol. 8, no. 3, pp. 281–293, Jun. 2000.

[6] A. Broder and M. Mitzenmacher, “Network applications of Bloom filters: A survey,” *Internet Math.*, vol. 1, no. 4, pp. 485–509, 2004.

[7] M. Mitzenmacher, “Compressed Bloom filters,” in *Proc. PODC*, vol. 1. Newport, RI, USA, 2001, pp. 144–150.

[8] H. Balakrishnan *et al.*, “A layered naming architecture for the Internet,” in *Proc. ACM SIGCOMM*, Portland, OR, USA, Aug. 2004, pp. 343–352.

[9] V. Jacobson *et al.*, “Networking named content,” in *Proc. ACM Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, Rome, Italy, Dec. 2009, pp. 1–12.

[10] O. Waltari and J. Kangasharju, “Content-centric networking in the Internet of Things,” in *Proc. 13th IEEE Annu. Consum. Commun. Netw. Conf. (CCNC)*, Las Vegas, NV, USA, Jan. 2016, pp. 73–78.

[11] M. Andreolini and R. Lancellotti, “A flexible and robust lookup algorithm for P2P systems,” in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, Rome, Italy, May 2009, pp. 1–8.

[12] S. S. Adhatarao, J. Chen, M. Arumaithurai, X. Fu, and K. K. Ramakrishnan, “Comparison of naming schema in ICN,” in *Proc. IEEE Int. Symp. Local Metropolitan Area Netw. (LANMAN)*, Rome, Italy, Jun. 2016, pp. 1–6.

[13] C. Intanagonwiwat, R. Govindan, and D. Estrin, “Directed diffusion: A scalable and robust communication paradigm for sensor networks,” in *Proc. ACM MobiCOM*, Boston, MA, USA, Aug. 2000, pp. 56–67.

[14] H. Yue, L. Guo, R. Li, H. Asaeda, and Y. Fang, “DataClouds: Enabling community-based data-centric services over the Internet of Things,” *IEEE Internet Things J.*, vol. 1, no. 5, pp. 472–482, Oct. 2014.

[15] C. Sarkar *et al.*, “DIAT: A scalable distributed architecture for IoT,” *IEEE Internet Things J.*, vol. 2, no. 3, pp. 230–239, Jun. 2015.

[16] S. Li *et al.*, “IoT middleware architecture over information-centric network,” in *Proc. IEEE Globecom Workshops*, San Diego, CA, USA, Dec. 2015, pp. 1–7.

[17] L. Zhang *et al.*, “Named data networking,” *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 66–73, Jul. 2014.

[18] D. Pfisterer *et al.*, “SPITFIRE: Toward a semantic Web of Things,” *IEEE Commun. Mag.*, vol. 49, no. 11, pp. 40–48, Nov. 2011.

[19] E. D. Porteer, I. Moerman, and P. Demeester, “Enabling direct connectivity between heterogeneous objects in the Internet of Things through a network-service-oriented architecture,” *EURASIP J. Wireless Commun. Netw.*, vol. 61, no. 1, pp. 1–14, 2011.

[20] L. Schmidt, N. Mitton, D. Simplot-Ryl, R. Dagher, and R. Quilez, “DHT-based distributed ALE engine in RFID middleware,” in *Proc. IEEE Int. Conf. RFID Technol. Appl. (RFID-TA)*, Sitges, Spain, Sep. 2011, pp. 319–326.

[21] B. Fabian, T. Ermakova, and C. Muller, “SHARDIS: A privacy-enhanced discovery service for RFID-based product information,” *IEEE Trans. Ind. Informat.*, vol. 8, no. 3, pp. 707–718, Aug. 2012.

[22] E. Guttman, “Autoconfiguration for IP networking: Enabling local communication,” *IEEE Internet Comput.*, vol. 5, no. 3, pp. 81–86, May/Jun. 2001.

[23] M. Nidd, “Service discovery in DEAPspace,” *IEEE Pers. Commun.*, vol. 8, no. 4, pp. 39–45, Aug. 2001.

[24] G. Oikonomou, I. Philips, L. Guan, and A. Grigg, “ADDER: Probabilistic, application layer service discovery for MANETs and hybrid wired-wireless networks,” in *Proc. IEEE Conf. Commun. Netw. Services Res. (CNSR)*, Ottawa, ON, Canada, May 2011, pp. 33–40.

[25] A. Kovacevic, J. Ansari, and P. Mahonen, “NanoSD: A flexible service discovery protocol for dynamic and heterogeneous wireless sensor networks,” in *Proc. IEEE Int. Conf. Mobile Ad Hoc Sensor Netw. (MSN)*, Hangzhou, China, Dec. 2010, pp. 14–19.

[26] R. Klauck and M. Kirsche, “Bonjour Contiki: A case study of a DNS-based discovery service for the Internet of Things,” in *Proc. 11th Int. Conf. Ad Hoc Netw. Wireless (ADHOC-NOW)*, Belgrade, Serbia, Jul. 2012, pp. 316–329.

[27] B. Djamaa, M. Richardson, N. Aouf, and B. Walters, “Towards efficient distributed service discovery in low-power and lossy networks,” *Wireless Netw.*, vol. 20, no. 8, pp. 2437–2453, 2014.

[28] K. Q. AbdelFadeel and K. Elsayed, “6LoWDIS: A lightweight service discovery protocol for 6LoWPAN,” in *Proc. IEEE Int. Conf. Commun. Workshops (ICC)*, Kuala Lumpur, Malaysia, May 2016, pp. 284–289.

Michele Amoretti (S’01–M’06) received the Laurea and Ph.D. degrees from the University of Parma, Parma, Italy.

He is currently an Assistant Professor with the Department of Information Engineering, University of Parma. He has authored or co-authored over 100 technical papers in refereed international journals, conference proceedings, and books. His current research interests include complex and adaptive distributed systems, service-oriented architectures and distributed middleware, wireless sensor networks, and quantum computing architectures and protocols.

Olivier Alphand received the Ph.D. degree from the Institut National Polytechnique de Toulouse, Toulouse, France, in 2005.

He is involved in several national and European projects. He has authored or co-authored several papers ranging from admission control and signaling in satellite networks to medium access control and routing protocols in sensor networks. His current research interest includes wireless networking.

Gianluigi Ferrari (S’96–M’98–SM’12) received the Laurea (*summa cum laude*) and Ph.D. degrees from the University of Parma, Parma, Italy.

He is currently an Associate Professor of Telecommunications with the Department of Information Engineering, University of Parma, and coordinates the Internet of Things (IoT) Laboratory. His scientific activity has led to around 300 works (in terms of papers, book chapters, patents, and books). His current research interests include signal processing, advanced networking/communication, and IoT and smart systems.

Prof. Ferrari was the recipient of several Technical/Best Paper Awards.

Franck Rousseau (M’04) received the Ph.D. degree from the Grenoble Institute of Technology, Grenoble, France, in 1999.

He is an Associate Professor with the Grenoble Institute of Technology/Ensimag. He was a Technical Staff Member with the OSF Research Institute, Grenoble. He is actively involved in national and European projects. His current research interests include communications and security for the Internet of Things, wireless mobile networks, and sensor networks.

Prof. Rousseau has served as a Program Committee Member for several networking conferences.

Andrzej Duda (M’10) received the Ph.D. degree from the Universitè de Paris-Sud, Orsay, France, in 1984, and the Habilitation Diploma degree from the University of Grenoble, Grenoble, France, in 1994.

He is a Professor with the Grenoble Institute of Technology/Ensimag, Grenoble. He is a Head of the Networks and Multimedia Group, Grenoble Informatics Laboratory, University of Grenoble. He was an Assistant Professor with the Université de Paris-Sud, from 1983 to 1986, a Chargé de Recherche with CNRS, Paris, France, from 1986 to 1995, and a Visiting Scientist with the MIT Laboratory for Computer Science, Cambridge, MA, USA, from 1992 to 1994. From 2002 to 2003, he was an Invited Professor with the Swiss Federal Institute of Technology, Lausanne, Switzerland. He authored or co-authored over 150 papers in the areas of performance evaluation, distributed systems, multimedia, and networks.