DINAS: a DIstributed NAming Service for All-IP Wireless Sensor Networks

Michele Amoretti*, Olivier Alphand¶, Gianluigi Ferrari‡, Franck Rousseau¶, Andrzej Duda¶
*SITEIA.PARMA, University of Parma, Italy
Contact email: michele.amoretti@unipr.it
¶Grenoble Institute of Technology, CNRS Grenoble Informatics Laboratory UMR 5217, France
‡Department of Information Engineering, University of Parma, Italy

***Abstract*—The Internet of Things requires a naming service that can also be beneficial for searching for services and applications. To avoid the traditional DNS approach in which a DNS server provides mapping of names to IP addresses in its domain, we propose a novel network service called DINAS (DIstributed NAming Service). It is based on three pillars: 1) Bloom filters for creating compact names from node descriptions, 2) distributed caches for storing names within the network, and 3) overlay routing strategies to publish and discover information—not only names—within the network. In this work, we assume ContikiMAC at Layer 2, 6LoWPAN at Layer 2.5, IPv6 and RPL (routing protocol) at Layer 3, and we present a particular UDP-based overlay routing strategy. We evaluate the proposal by Cooja simulations and compare its performance with a centralized naming service at the sink. The results show that DINAS outperforms the centralized solution.**

***Keywords*—Wireless Sensor Networks, Naming Service, Resource Discovery, Bloom filters**

I. Introduction

The development of tiny IP stacks such as Contiki uIPv6 [1] makes possible 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 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 and globally or locally unique, but as services, we may accept that several nodes provide the same name.

The problems that motivated our work are the following: i) how to effectively and efficiently represent names within an IPv6-based Wireless Sensor Network (WSN) or in an IPv6-based network including several WSNs? ii) how to efficiently publish/lookup names within the networks or outside of them?

The traditional DNS approach requires a DNS server placed at the border router that stores all bindings and replies to all name resolution queries. Such operation may result in a large overhead and excessive energy consumption. We propose *DINAS (DIstributed NAming Service)*, another approach that handles name resolution in a distributed manner. Names in DINAS are constructed as compact arrays of bits generated from node descriptions. The interest of a distributed service comes from the constraints of sensor networks: in many cases, the information from sensors may be required from nodes that are near to the sensors, so it is unnecessary to involve the sink (an example of such a situation is the control of air-conditioning, in which actuators locally operate based on the temperature gathered by sensors). Moreover, nodes, in many cases, may want to discover services offered in the network through a naming service in a similar way to mDNS/Bonjour. Our approach enables such extensions.

DINAS is based on three pillars: 1) Bloom filters for creating compact names from device descriptions, 2) a conservative strategy for publishing and resolving names within the network, 3) distributed caches for storing names within the network (instead of concentrating them at the border router).

Once a node has created its name as a Bloom filter, it propagates the information about the binding name-address to other nodes. We assume that the network builds its structure as a *Destination-Oriented Directed Acyclic Graph (DODAG)* with the RPL[1] protocol. Name propagation takes advantage of the structure: each node sends its name binding to its parent, that in turn propagates it up the DODAG to the sink, which sends the binding further down to a subgraph. Intermediate nodes on the propagation path store the binding in a cache so that they can resolve a given name. The choice of the nodes to propagate the binding down a subgraph depends on *proximity*—a node chooses the next one based on the similarity of names, which results in grouping similar names at the same nodes. Name propagation is limited in depth so that only a part of nodes stores the binding and they are placed at strategic points in the network, reachable in few hops from any node.

The rest of the paper is organized as follows. Section II describes the principles of DINAS. Section III evaluates its performance. We briefly discuss the related work in Section IV and conclude the paper in Section V.

II. DINAS

In this paper, we propose DINAS, a new approach to name resolution suitable for WSNs. A node joining the network

[1]http://tools.ietf.org/html/rfc6550

978-1-4799-3083-8/14/$31.00

encodes its descriptor (including a list of features, services, and the information that the node may provide) in a *Bloom filter* (BF) of a given size. The Bloom filter becomes the name of a node. DINAS handles the binding between a name and its IPv6 address so that nodes can resolve a name to obtain the address.

A. From descriptors to names

We propose to generate names from node descriptors: a name is an array of m bits. Similarly, a query for name resolution is also encoded as a m-bit array and the service matches the query with published names.

As Andreolini and Lancellotti have shown [2], it is possible to map several keywords to a machine-readable flat name by means of a data structure called a Bloom filter [3], [4]. Let B be an array of m bits representing the key for item r (assume the array B is initially filled with 0). Let $KW = \{kw_1, kw_2, ..., kw_n\}$ be a set of keywords associated with r, and let $H = \{h_1, h_2, ..., h_k\}$ be a set of hash functions, where $h_j : KW \rightarrow [0, m-1]$. Bloom filter B is built as follows: $B[h_j(kw_i)] \Leftarrow 1, \forall kw_i \in KW, \forall h_j \in H$: for every keyword kw_i, set to 1 the bits in the Bloom filter corresponding to the results of k hash functions computed on the keywords as shown in Fig.1.

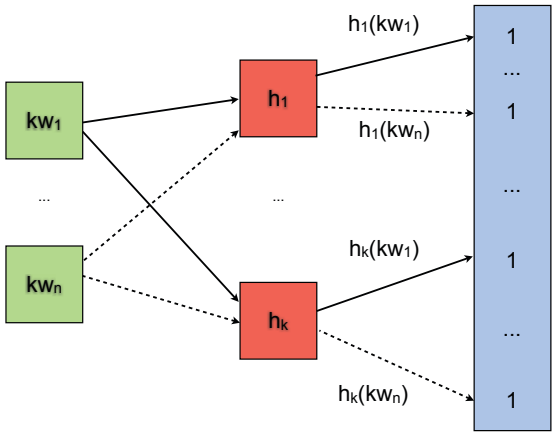

Fig. 1: Mapping keywords to a flat n bit name with a Bloom filter.

To find if keyword kw_x is within the set of keywords encoded in a given Bloom filter B, we check the bits at positions $h_1(kw_x)$, $h_2(kw_x)$, etc. If even one of them is 0, then certainly kw_x is not in the set represented by B. Otherwise, we conjecture that kw_x is in the set, although there is a certain probability that we are wrong. This is called a "false positive" or, for historical reasons, a "false drop". Parameters k and m should be chosen so that the probability of a false positive (and hence a false hit) is acceptable.

After inserting n keys into a Bloom filter of size m, the probability that a particular bit is still 0 is exactly

$$(1 - \frac{1}{m})^{kn}. \quad (1)$$

Hence, the probability of a false positive is

$$(1 - (1 - \frac{1}{m})^{kn})^k \approx (1 - e^{-kn/m})^k. \quad (2)$$

The right hand side is minimized for

$$k = \frac{m}{n} \ln 2. \quad (3)$$

In fact, k must be an integer and in practice, we might choose a value less than optimal to reduce computational overhead. For instance, if we want to describe each item with up to $n = 10$ keywords, with $m = 160$ and $k = 11$, the probability of false positives would be $p_{err} = 0.000458$. By properly choosing the parameters, we can make the probability of false positive sufficiently low.

We define selectivity of query σ as the ratio of bits set to 1 and the length m of the BF. It represents the number of potential hits returned by a query. When σ is low, the query is highly selective and only few items will match the query. On the other hand, when σ is high, most items will generate a hit.

DINAS compares two BFs in terms of the XOR distance: if they differ on m bits, their distance is m.

B. Operation and overlay routing

For the sake of simplicity, we will consider below that all nodes support DINAS. We also assume that nodes run RPL (not RPL-P2P[2]) and we rely on the fact that each node knows the addresses of its parent and children in the RPL DODAG.

With DINAS, nodes propagate names (or more precisely name-address bindings) in a proactive and periodic manner within the network up to some level D and some nodes selectively cache them for name resolution. Such selective propagation process results with much less overhead than flooding, but as a consequence, not all nodes are aware of all names. Thus, name resolution requires propagation of name queries to find a node that stores a given name in its cache and can reply with the information. We limit the propagation of name queries to several hops to reduce overhead. Nodes that receive a reply, store it in their caches, which reinforces the distributed name resolution process, because a name will appear in an increasing number of nodes.

DINAS messages are *name notification*, *name query*, and *reply*. A name notification contains the name binding: name and the IP address of the node that starts the notification process. A name query includes the requested name and the IP address of the node that starts the name resolution process. A reply provides the requested binding: the name and the associated IP address. Name notifications and queries also contain Time-To-Live, *i.e.*, the number of allowed propagation hops, used to limit the propagation process.

After having joined the network and selected its preferred parent in the DODAG, a node starts to periodically publish its name generated as described in II-A through a notification sent towards the DODAG root. The root then propagates the notification downward to a node chosen based on cache proximity. The notification propagates further on for a limited

[2]http://tools.ietf.org/html/rfc6997

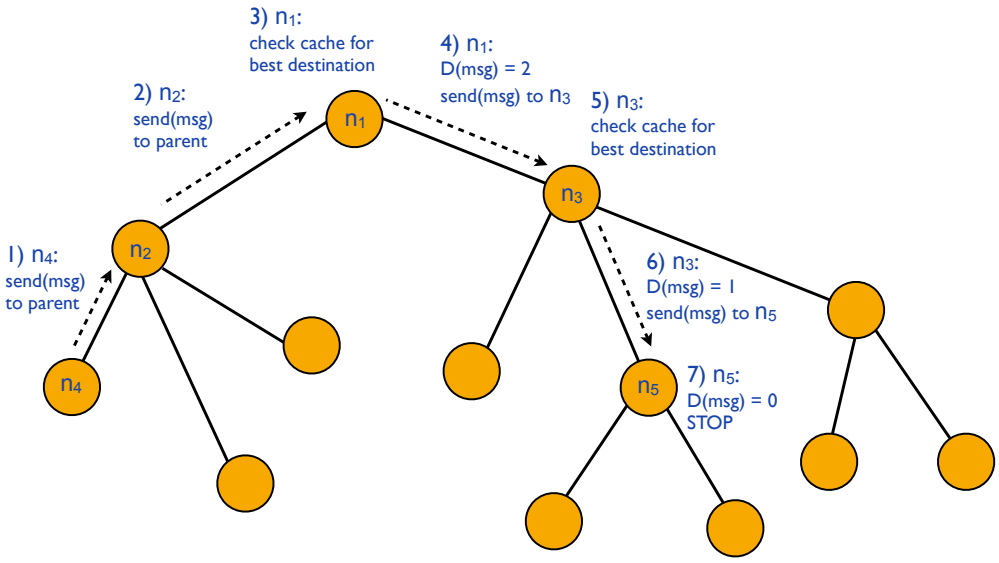

Fig. 2: DINAS "up-down" overlay routing.

number of hops. Such an overlay routing protocol, called "up-down", is illustrated in Figure 2. Periodic name publication supports the name resolution process, as names that are not queried may disappear from the network because of content replacement policies for the cache.

When a node receives a name notification, it stores it in its cache, if it is possible depending on the caching policy (refer to Section II-C), and propagates it to its parent (if the notification is on the upward path to the sink) or to its children (on the downward path from the sink). When a node receives a name resolution query, it first checks if its name matches. If it is the case, the node sends a reply to the query issuer, identifying itself as the owner of the queried name. Otherwise, the node checks its cache that may contain the name and propagates the query if not found.

As DINAS 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 query resolution.

The overlay routing protocol relies on the RPL DODAG. Nodes forward DINAS name notification messages to the sink and they are cached along the path. The sink may decide to propagate the messages to a subset of its children excluding the message sender. When the messages reach the sink, it selects the nodes for forwarding based on cache proximity. When the cache is empty, the nodes are randomly selected with an equal probability among children. In the current version of the protocol, replies are also cached. Moreover, the downward propagation is limited to D hops. If R is the rank of the message issuer, then the time-to-live of the message is $TTL = R + D$.

C. Caching

Independently of caching policy, the cached item has the following structure:

type (either notification or reply);
Bloom filter (m bits);
owner IP address;
provider IP address;
timestamp (*i.e.*, caching time).

The *proximity cache* is specialized in handling clusters of similar names: as more names are processed, the DINAS routing becomes better trained.

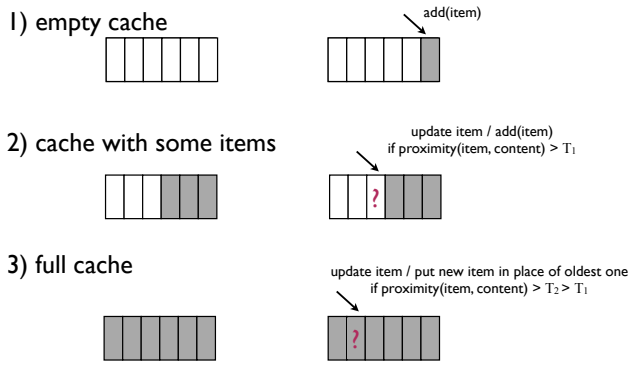

Fig. 3: The principle of the proximity cache.

As illustrated in Figure 3, an item is added to the cache only in three cases. First, if the cache is empty. Second, when the cache is not empty, but not full, if the % of total matching bits of cached items with the item to the cached is $> T_1$. If an item with the same BF is already cached, but owners are different, the new one is added to the cache. Otherwise, if the owner is the same, the timestamp of the cached item is updated to the current time. Third, when the cache is full, the % of total matching bits of cached items with the item to the cached is $> T_2 > T_1$. If an item with the same BF is already cached, but owners are different, the new one substitutes the old one in the

cache. If the owner is the same, the timestamp of the cached item is updated to the current time. $T_1, T_2 \in \{0, .., 100\}$ are integer thresholds.

DINAS names are BFs, hence similar names correspond to similar items. For example, a 32-bit BF (with 2 hash functions) obtained from the description (function:Sensing, object:Temperature) is

00000010010000000001000001000000

while the one obtained from (function:Sensing, object:Temperature, location:ENSIMAG-D327) is

01000010010000100001000001000000

The names differ by just 2 bits. A lookup for a more general description (function:Sensing, object:Temperature) would return both names, while a lookup for a more specific description (function:Sensing, object:Temperature, space:ENSIMAG-D327) would return only the second name.

If two or more subsequent queries for the same name arrive before the name expires at any of the caches along the way, the cached name is sent (in a reply message) to the interested nodes, thus avoiding to contact the name owner.

III. Performance evaluation

We have simulated several DINAS scenarios in Cooja[3] on a MacBook Pro with 2.4 GHz Intel Core 2 Duo and 4GB 1067 MHz DD3 RAM. Every simulation has been executed 10 times with different seeds for the random number generator of Cooja.

In the considered scenarios, we assume ContikiMAC at Layer 2, 6LoWPAN at Layer 2.5, IPv6 and RPL (routing protocol) at Layer 3. Nodes are organized in a RPL DODAG, one of them (node 1) being the DODAG root. All nodes are configured as Tmote Sky[4] hardware.

The RPL DODAG root is configured like all other nodes in terms of the cache size and performed operations. All nodes run two processes, one for sending DINAS messages, and the other one for handling incoming messages. Each node builds its own name as a Bloom filter B with $m = 16$ bits and $k = 6$ hash functions, the optimal parameters for a node description with $n = 2$ keywords. Such a Bloom filter is filled with the following keywords:

- "netservice-_dinas._udp.local",
- "location-loc",

where loc is the node ID, a unique integer (in range $[1, N]$, where N is the number of nodes). In this way, each node has a unique name. A node publishes its name every T minutes (with a notification message). In between, a node sends a query every τ minutes for a randomly generated name (among those that are possible and excluding the node own name). A node sends M messages of which $\tau M/T$ are notifications and $M - \tau M/T$ are queries. T has a low impact on the hit ratio as the network is static. In our tests, $T = 20$ minutes, $\tau = 2$ minutes, and $M = 20$ messages. Thus, 2 notifications and 18 requests are sent by each node over a simulated period of about 40 minutes. A simulation was repeated with 10 different seeds for the random number generator. The plotted values below are the results of averaging the values of the 10 runs. Other tunable parameters are: the size of the cache C, the downward propagation limitation D, the cache thresholds T_1 and T_2.

In the evaluation, we have used the topology illustrated in Figure 4 with $N = 20$ nodes. The DODAG root is placed in the middle of the area, we call it the DODAG X. Nodes have rank 3 at most.

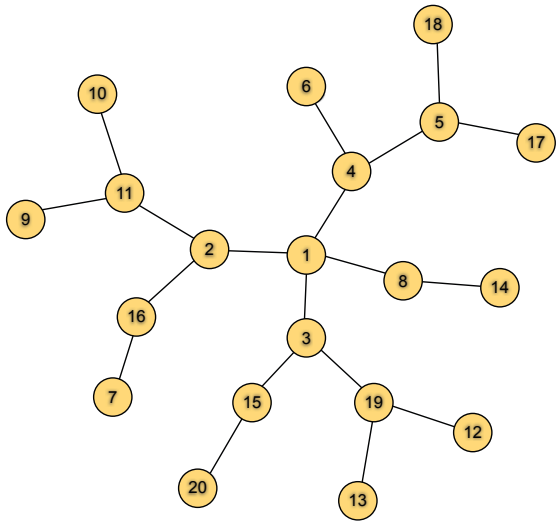

Fig. 4: RPL DODAG X used in the simulation.

We have considered the following performance indicators:

- Hit Ratio (HR), *i.e.*, the percentage of replies with respect to queries sent (considering the whole network);
- Average Response Time (ART), the average time between a query and the related reply (taking into account only the queries for which there is a reply);
- Total Traffic (TT), *i.e.*, the total number of "send()" operations for notifications and queries executed by the nodes in the network.

Message length is 24 bytes: 2 bytes for the Bloom filter, 19 bytes for the IPv6 address of the message issuer (in the string format), 1 byte for D, 1 byte for the message type, 1 byte for the flooding flag (we set it to 0, *i.e.*, no flooding). The packet size could be further reduced by using only 1 byte for D, message type and flooding flag, and, whenever possible, short representations of the IPv6 address. Conversely, the size of the Bloom filter may be larger, if more than $n = 2$ keywords are used.

We compare DINAS against a centralized solution in which the DODAG root (called a *sink*) acts as a unique resource directory. Thus, it has a large cache. Communications are only direct UDP unicasts with no overlay routing. Notifications and queries can only be processed by the sink, which is the only one that can answer. The centralized solution requires that the sink has $C = 19$ to store the names of all other nodes. With DODAG X, we have obtained $HR = 99.77\%$ (some packets

[3]We have used Contiki/Cooja v2.6. http://www.contiki-os.org/start.html
[4]http://www.snm.ethz.ch/Projects/TmoteSky

are lost due to collisions). TT of the centralized solution can be analytically computed given the topology of the RPL DODAG. Indeed, if the node rank is R, each notification and query to the sink will cause R "send()" executions. We use the following equation:

$$TT = M \sum_{i=1}^{R_{max}} i \cdot N_i, \quad (4)$$

where R_{max} is the higher rank in the RPL DODAG and N_i is the number of nodes with rank i. For DODAG X, $R_{max} = 3$, $N_1 = 4$, $N_2 = 7$, and $N_3 = 8$. Since $M = 20$, we have $TT = 840$.

Figure 5 shows HR as a function of C and D with $T_1 = 30$ and $T_2 = 90$. Such a configuration for the proximity cache is the most effective one among those we have tested (we omit the results of such an analysis, because of the space limits). The best result ($HR = 99.77\%$) is given by the configuration with $C = 11$ and $D = 3$. Couples (C, D) such as $(11, 1)$, $(9, 2)$, $(7, 3)$ also lead to good HR values. For $C < 7$ or $D = 0$, the HR is definitely too low.

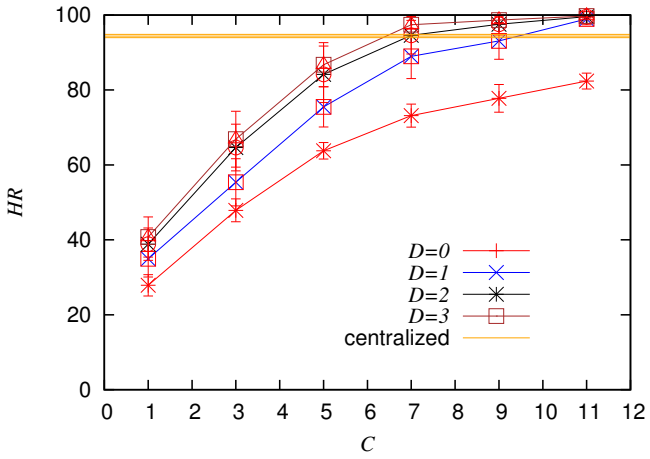

Fig. 5: DINAS "up-down", DODAG X: HR as a function of C and D, with $T_1 = 30$ and $T_2 = 90$.

Figures 6 and 7 show that the centralized solution can be easily outperformed by DINAS, in terms of TT and ART, respectively. Apparently, $C = 5$ is sufficient to have TT lower than the one of the centralized solution, but the corresponding HR values are worst. However, we observe that $C = 7$ and $D = 3$ is an interesting solution, leading to $TT = 565$ and $ART = 235 \pm 21\mu$s (with $HR = 97.41\%$). TT of the centralized solution is much higher (840, as illustrated above). The ART of the centralized solution is also higher that has been measured by means of several simulations—its value is $452 \pm 25\mu$s. Using the DINAS configuration that gives the higher HR value (99.77%), namely $C = 11$ and $D = 3$, we obtain $TT = 451$ and $ART = 160 \pm 11.6$. 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 requires more memory resources. Definitively, the (C, D) configuration depends on our simulation assumptions.

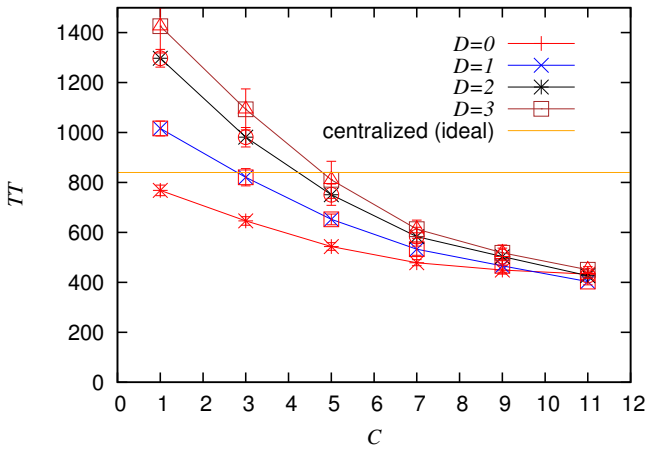

Fig. 6: DINAS "up-down", DODAG X: TT as a function of C and D, with $T_1 = 30$ and $T_2 = 90$.

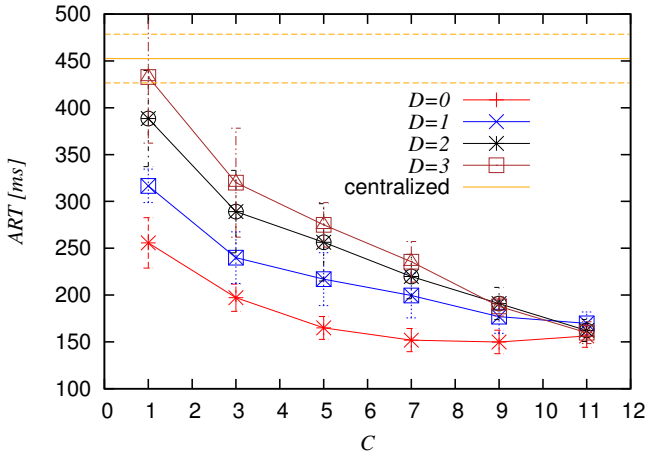

Fig. 7: DINAS "up-down", DODAG X: ART as a function of C and D, with $T_1 = 30$ and $T_2 = 90$.

IV. Related work

Concerning the general problem of naming, Balakrishnan proposed to use flat names for network elements [5]. Such an approach is already used to name nodes in structured (*i.e.*, DHT-based) 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. Instead, in their seminal work about content-centric networking (CCN), Jacobson *et al.* proposed a naming scheme in which names are hierarchically structured with components encoded one by one [6].

Intanagonwiwat *et al.* introduced *directed diffusion* [7], 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. 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 is then drawn down towards 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 (we call them queries) can find matching node descriptors that have been stored in the cache of an intermediate node. In other words, to start the interaction between the requester and the source, it is not necessary that the query arrives at the source. DINAS messages are compact, as node descriptors and requests are encoded in Bloom filters. Finally, DINAS is application-independent. How requesters and sources interact after they found each other, is independent from DINAS.

Currently, Zeroconf [8] is the most used technology for bootstrapping direct communication between two or more computing devices via IP. Zeroconf has several implementations, *e.g.*, Apple Bonjour, Avahi (for Linux machines), and Microsoft LLMNR. Klauck and Kirsche have recently developed *uBonjour*, a Zeroconf implementation for Contiki [9]. Zeroconf relies on DNS-SD, DNS records used to describe advertised services, and on Multicast DNS (mDNS).[5] Multicast DNS (mDNS) allows to perform DNS operations on a local link without a traditional DNS server. Thus a client acts as a DNS server that only answers DNS queries targeting its own resource records. It is possible since queries and answers are UDP packets sent to multicast addresses 224.0.0.251 (IPv4) and FF02::FB (IPv6), and UDP port 5353. mDNS raises the following issues: it is not fully distributed (it is a single link protocol), it does not provide a means to describe the capabilities of devices, it is too much wordy, and it does rely on multicast, which is not always available, especially in the context of WSNs. DINAS proposes a compact naming scheme, solves the multicast problem (by means of caching and overlay routing), and can be easily extended to support autonomous services.

CoAP[6] is a lightweight RESTful transfer protocol for accessing data on constrained devices. A typical CoAP-based WSN deployment requires to establish a resource directory[7] or a proxy (*e.g.* on a border router) to periodically announce itself (*e.g.* with Zeroconf) or to be discovered by sensors thanks to an anycast address or a specific CoAP request. At the startup, other 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 further discusses issues related to CoAP [10], 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.

[5]http://tools.ietf.org/html/rfc6762

[6]http://tools.ietf.org/html/draft-ietf-core-groupcomm-11

[7]http://tools.ietf.org/html/rfc6690

V. Conclusions

In this paper, we have presented DINAS, a novel network service for publishing and retrieving information about names and services. DINAS creates names as Bloom filters based on node or service descriptions. We have described the main principles of DINAS and its implementation under Contiki and Cooja. We have evaluated the proposal with Cooja simulations and compare its performance with a centralized naming service at the sink. The results show that DINAS outperforms the centralized solution.

The performance results are encouraging and we already plan new experiments with different (in shape and size) DODAGs. We would also like to test alternative overlay routing schemes, in particular, the schemes for mesh L3 topologies other than DODAGs.

VI. Acknowledgments

The work of O. Alphand, F. Rousseau, A. Duda, and G. Ferrari was partially supported by the European Commission FP7 project CALIPSO under contract 288879. 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. O. Alphand, F. Rousseau, and A. Duda were also supported by the French National Research Agency (ANR) project project IRIS under contract ANR-11-INFR-016.

References

[1] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki – a Lightweight and Flexible Operating System for Tiny Networked Sensors," in *29th Annual IEEE International Conference on Local Computer Networks*, Nov. 2004, pp. 455–462.

[2] M. Andreolini and R. Lancellotti, "A Flexible and Robust Lookup Algorithm for P2P Systems," in *23rd IEEE International Parallel & Distributed Processing Symposium (IPDPS 2009)*, May 2009, pp. 1–8.

[3] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol," *IEEE/ACM TRANSACTIONS ON NETWORKING*, vol. 8, no. 3, pp. 281–293, Jun. 2000.

[4] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," in *Internet Mathematics*, 2002, pp. 636–646.

[5] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish, "A Layered Naming Architecture for the Internet," in *ACM SIGCOMM'04*, Aug. 2004.

[6] V. Jacobson, D. K. Smetters, J. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking Named Content," in *ACM CoNEXT 2009*, Dec. 2009, pp. 1–12.

[7] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks," in *ACM MOBICOM '00*, Aug. 2000, pp. 56–67.

[8] E. Guttman, "Autoconfiguration for IP Networking: Enabling Local Communication," *IEEE Internet Computing*, vol. 5, no. 3, pp. 81–86, Jun. 2001.

[9] R. Klauck and M. Kirsche, "Bonjour Contiki: a Case Study of a DNS-based Discovery Service for the Internet of Things," in *11th International Conference on Ad Hoc Networks and Wireless (ADHOC-NOW 2012)*, Jul. 2012, pp. 1–13.

[10] A. Rahman and E. Dijk, "Group Communication for CoAP," IETF draft, May 2013. [Online]. Available: http://tools.ietf.org/html/draft-ietf-core-groupcomm-07