Energy-Efficient Cluster-Based Service Discovery in Wireless Sensor Networks

Raluca Marin-Perianu, Hans Scholten, Paul Havinga and Pieter Hartel
University of Twente
Enschede, The Netherlands
Email: {r.s.marinperianu, j.scholten, p.j.m.havinga, pieter.hartel}@utwente.nl

***Abstract*— We propose an energy-efficient service discovery protocol for wireless sensor networks. Our solution exploits a cluster overlay, where the clusterhead nodes form a distributed service registry. A service lookup results in visiting only the clusterhead nodes. We aim for minimizing the communication costs during discovery of services and maintenance of a functional distributed service registry. We compare theoretically and by simulation the impact of the chosen clustering algorithm on the service discovery protocol.**

I. Introduction

Wireless Sensor Networks (WSNs) is an emerging technology that opens a wide perspective for future applications in ubiquitous computing and ambient intelligence. A typical application of a WSN consists of gathering data from large areas and processing it at a central location.

As the technology evolves, the sensor nodes are expected to self-organize and adapt in face of mobility, failures, changes of network tasks and requirements. Nodes are aware of their own capabilities and are able to cooperate with other nodes in the network, for the purpose of providing networking and system services. The focus changes toward sensor networks providing services to clients in a wide range of applications [11], [12].

In this context, a prerequisite for providing a service-oriented functionality is the ability to search for services in a WSN. A service discovery protocol has three participating entities: the *service provider*, the *service consumer* and the *service directory*. The latter represents a group of devices responsible for maintaining a distributed repository of service descriptions, which is accessed by the consumer in the search process. The result of a service lookup is the address of one or more service providers.

Designing a service discovery protocol for WSN environments implies a number of challenges. Since sensor nodes are likely to be battery powered, the first objective is to minimize the energy consumption. As the energy is spent mostly during communication, minimizing the energy consumption translates into minimizing the communication cost. The problem is challenging especially in large scale, dense networks, where significant traffic is generated due to the intrinsic broadcast nature of the wireless communication. A second challenge is to react rapidly to the network topology changes, which directly affect the consistency of the distributed directory.

The most straightforward method for searching in a WSN is based on flooding, which has the advantage of zero maintenance overhead. However, flooding has obvious limitations with regard to energy-efficiency and scalability. A better approach for WSNs is based on clustering, where a set of designated nodes acts as a distributed directory of service registrations for the nodes in their cluster. In this way, the communication costs are reduced, since the service discovery messages are exchanged only among the nodes from the distributed directory.

In this paper we propose a solution for service discovery in WSNs, based on clustering. The main goal is to minimize the energy consumed both during maintenance and discovery phases. The contributions of this paper are therefore:

- A lightweight clustering algorithm that builds a distributed directory of service registrations.
- An energy-efficient service discovery protocol that exploits the clustering structure.

We evaluate the performance of this integrated solution through simulations under different network densities and levels of dynamics. Additionally, to evaluate solely the clustering substrate, we compare our results to those of DMAC [2], a state of the art clustering algorithm.

The paper is organized as follows. We give an overview of related work in the field of service discovery in Section II. In Section III we present our design considerations regarding the proposed discovery solution. The clustering structure and the service discovery protocol are discussed in detail in Sections IV and V. Section VI presents the performance evaluation of the protocol, based on both our clustering algorithm and DMAC. Section VII presents a summary and future work.

II. Related work

In the following, we give a short overview of service discovery protocols designed for ad-hoc networks.

For energy-efficiency reasons, cross-layered solutions have been explored, where service discovery protocols piggyback on the routing messages to issue service request and get replies. Frank and Karl [6] rely on AODV [5], Wu and Zitterbart [15] use DSR [8]. These routing protocols use flooding to set up paths to destinations. Flooding limits the scalability of the routing protocols and consequently, of the service discovery protocols that rely on them.

DHT based peer-to-peer techniques have been proposed for service discovery in ad-hoc networks [1], due to their efficient lookup mechanism. However, this approach generates considerable network traffic and a high maintenance overhead, so it is not suitable for the WSN environment.

Kozat and Tassiulas [10] build a dominating set (or backbone) to which devices register their services. Due to the high density of nodes in the backbone (the dominating set is not independent), lots of loops are generated when a service discovery message travels the backbone nodes. To overcome this drawback, the backbone organizes in a source-based multicast tree. However, building and maintaining two overlays for the same purpose (the dominating set and the multicast tree) is expensive for resource-constraint sensor nodes.

Helmy [7] proposes a resource discovery protocol, where each node keeps track of a number of nodes within R hops away, that defines the zone of the node. As part of the zone information each node keeps routes to nodes and resource information. Moreover, a node has knowledge of a number of contact nodes outside its zone. The search method implies forwarding the requests to the contact nodes. However, maintaining a complete topological view over a number of hops, together with the knowledge on available resources is not feasible for the WSN environment.

III. Design considerations

In this section we discuss from the design perspective several techniques for reducing the communication cost during (1) discovery of services and (2) maintenance of the distributed directory.

The service discovery protocol uses an underlying clustering structure, where the clusterheads (or root nodes) form a distributed directory of service descriptions. During the discovery process, messages are exchanged among the clusterhead nodes. Therefore, the design issue for minimizing the discovery cost is that the root nodes have to be sparsely distributed on the deployment area. The clustering algorithm should construct an *independent set* of clusterheads, i.e. two root nodes are not allowed to be neighbors.

In the following, we give the design considerations for minimizing the communication cost during the maintenance of the distributed directory:

- *Make decisions based on 1-hop neighborhood information.* Clustering algorithms that require each node to have complete topology knowledge over a number of hops are expensive with regard to the maintenance cost. We aim to build a lightweight clustering structure that requires only the 1-hop neighborhood topology information.
- *Avoid chain reaction.* Several clustering algorithms [2] suffer from the chain reaction, where a single topology change in the network may trigger significant changes in clustering structure. For a distributed directory composed of clusterhead nodes, the chain reaction leads to high overhead for maintaining consistent service registries. Therefore, an energy-efficient solution should a avoid chain reaction, such that local topology changes determine only local modifications of the directory structure.
- *Distribute the knowledge on adjacent clusters among cluster members.* The knowledge on adjacent clusters should be distributed among the ordinary nodes. Only the root needs to know all the nearby clusters.

In theory, building an independent set of root nodes and avoiding a chain reaction comes at the expense of constructing clusters with an arbitrary height. However, in practice, we show that we can achieve small-height clusters without imposing a maximal height limit (see Section VI-D).

IV. Clustering algorithm

A. Network model

We model a wireless network as an undirected graph $G = (V, E)$, where V is the set of nodes and E is the set of links that directly connect two nodes. Two nodes u and v are neighbors if there is a direct communication channel between u and v. W Each node is assigned (1) a unique hardware identifier, termed the *address* of the node, and (2) a weight, termed the *capability grade*, representing an estimate of the node's dynamics and available resources. The higher the capability grade, the more suitable is the node for the clusterhead role. We make the following assumptions:

- The capability grades are unique, as the node hardware identifier may be used to break ties.
- The lower layers (such as MAC) filter out asymmetrical links, so that we can rely on bidirectional communication.
- A node is aware of its neighbors and their capability grades.
- The lower layers (such as transport) provide a reliable, best-effort message delivery service.

Our clustering structure is a *forest* composed of a set of disjoint *trees* or *clusters*. The *height* of the cluster is the longest path from the root node to a leaf. We say that two trees are *adjacent* if there are two nodes, one from each tree, that are connected through a link.

Given a node v, we use the following notation:

- $p(v)$ is the parent of v
- $r(v)$ is the root (or clusterhead) of the cluster of v
- $\Gamma(v)$ is the open neighborhood of v, $\Gamma(v) = \{u \in V \mid (u, v) \in E\}$
- $\Gamma^+(v)$ is the closed neighborhood of v, $\Gamma^+(v) = \Gamma(v) \cup \{v\}$
- $\Delta(v)$ is the set of children of node v, $\Delta(v) = \{u \in V \mid p(u) = v\}$
- $p_k(v)$ is the parent of order k, defined as $p_0(v) = v$, $p_{k+1}(v) = p(p_k(v))$
- $C(v)$ is the set of nodes that are part of the same cluster as v, $C(v) = \{u \in V \mid r(u) = r(v)\}$
- $T(v)$ is the set of nodes from the sub-tree rooted at v, $T(v) = \{u \in V \mid \exists k \text{ such that } p_k(u) = v\}$
- $R_u(v)$ is the set of adjacent clusters of node v, represented by their roots, that can be reached through node

u, where $u \in \Gamma(v)$ (u is the next hop on the path to the adjacent cluster):

  - if $u \in \Delta(v)$, then $R_u(v)$ is the set of root nodes of clusters adjacent to the sub-tree rooted at v. Formally, $R_u(v) = \{r \in V \setminus \{r(v)\} \mid \exists x \in T(u), \exists y \in \Gamma(x)$ such that $r = r(y)\}$;
  - if $u \in C(v) \setminus \Delta(v)$, then $R_u(v) = \emptyset$;
  - if $u \notin C(v)$, then $R_u(v) = \{r(u)\}$.

- $S(v)$ is the set of services provided by node v
- $S_u(v)$ is the set of services registered to v by $u \in \Delta(v)$. Formally, $\forall u \in \Delta(v),\ S_u(v) = \{S(x) \mid x \in T(u)\}$.

B. Construction of clusters

The construction of clusters follows the idea of a greedy algorithm, where nodes choose a neighbor with higher capability grade as *parent*, while other nodes that do not have such a neighbor are *roots*. The message *SetRoot* is used for propagating the address of the root node to all the members of the clusters. The *Initialization* phase and the event *SetRoot* from Algorithm 1 give a formal description for the construction of clusters. Briefly, the protocol works as follows:

- Nodes that have the highest capability grades among their neighbors declare themselves clusterheads and broadcast a *SetRoot* message announcing their roles.
- The remaining nodes choose as parent the neighbor with the highest capability grade.
- When a node receives a *SetRoot* message from its parent, it learns the cluster membership and rebroadcasts the *SetRoot* message.

C. Knowledge on adjacent clusters

Once the clustering structure is set up, the root nodes need to establish links to the adjacent clusters. The root nodes learn about the adjacent clusters from the nodes placed at the cluster borders. During the propagation of the broadcast message *SetRoot* down to the leaf nodes, the message is also received by nodes from adjacent clusters. These nodes store the adjacent root identity in their $R_u(v)$ sets and report it to their parents. The information is propagated up in the tree with a message which we term *UpdateInfo*. Through this message, nodes learn the next hops for the paths leading to the clusters adjacent to their sub-trees. In particular, the root nodes learn the adjacent clusters and the next hops on the paths to reach their clusterheads. Figure 1 gives an intuitive example of learning the adjacent clusters.

The events of receiving messages *SetRoot* and *UpdateInfo* from Algorithm 1 describe how the knowledge and the paths to adjacent clusters is updated for a given node v. Duplicate *UpdateInfo* messages are discarded: a node v sends the message *UpdateInfo* to its parent if and only if the set of known root nodes changes. This means that if v is informed about a root node from one neighbor, but it knows already about this root through another neighbor, v does not propagate the information to the parent again.

Algorithm 1 Clustering algorithm - node v (events/actions)

```
Initialization: // Parent is chosen
1. $r(v) \leftarrow \perp$; $R_m(v) \leftarrow \emptyset,\ \forall m \in \Gamma(v)$
2. choose $p(v) \in \Gamma^+(v)$ such that $c(p(v)) = max\{c(m) \mid m \in \Gamma^+(v)\}$
3. if $p(v) = v$ then
4.     $r(v) \leftarrow v$ // I am root
5.     Send $SetRoot\ (v,\ r(v))$ to neighbors
6. end if

SetRoot $(u, r)$: // Receive root $r$ from neighbor $u$
1. $R_0 = \bigcup_{m \in \Gamma(v)} R_m(v)$
2. if $(p(v) = u) \wedge (r(v) \neq r)$ then
3.     $r(v) \leftarrow r$
4.     Send $SetRoot(v,\ r(v))$ to neighbors
5.     $\forall m \in \Gamma(v),\ R_m(v) \leftarrow R_m(v) \setminus \{r(v)\}$
6. else if $(r(v) \neq r) \wedge (r \neq \perp)$ then
7.     $R_u(v) \leftarrow \{r\}$
8. else if $(r(v) = r)$ then
9.     $R_u(v) \leftarrow \emptyset$
10. end if
11. if $(v \neq p(v)) \wedge (R_0 \neq \bigcup_{m \in \Gamma(v)} R_m(v))$ then
12.     Send $UpdateInfo\ (v, \bigcup_{m \in \Gamma(v)} R_m(v))$ to $p(v)$
13. end if

UpdateInfo $(u, R)$: // Receive adjacent clusters $R$ from $u$
1. $R_0 = \bigcup_{m \in \Gamma(v)} R_m(v)$;
2. $R_u(v) \leftarrow R \setminus \{r(v)\}$
3. if $(v \neq p(v))) \wedge (R_0 \neq \bigcup_{m \in \Gamma(v)} R_m(v))$ then
4.     Send $UpdateInfo(v, \bigcup_{m \in \Gamma(v)} R_m(v))$ to $p(v)$
5. end if

LinkAdd $(u, c)$: // $u$ added to neighborhood, with capability $c$
1. $\Gamma(v) \leftarrow \Gamma(v) \cup \{u\}$
2. if $c > c(p(v))$ then
3.     if $(v \neq p(v))$ then
4.         Send $UpdateInfo\ (v, \emptyset)$ to $p(v)$
5.     end if
6.     $p(v) \leftarrow u$ // The new neighbor becomes parent
7.     Send $UpdateInfo\ (v, \bigcup_{m \in \Gamma(v)} R_m(v))$ to $p(v)$
8. end if
9. Send $SetRoot\ (v, r(v))$ to neighbors

LinkDelete $(u)$: // $u$ deleted from neighborhood
1. $R_0 = \bigcup_{m \in \Gamma(v)} R_m(v)$
2. $\Gamma(v) \leftarrow \Gamma(v) \setminus \{u\}$ // Remove neighbor
3. if $u = p(v)$ then
4.     choose $p(v) \in \Gamma^+(v)$ such that $c(p(v)) = max\{c(m) | m \in \Gamma^+(v)\}$
5.     if $p(v) = v$ then
6.         $r(v) \leftarrow v$
7.         Send $SetRoot\ (v, r(v))$ to neighbors
8.     else
9.         if $r(v) \neq r(p(v))$ then
10.            $r(v) \leftarrow r(p(v))$ // Update cluster membership
11.            $\forall m \in \Gamma(v),\ R_m(v) \leftarrow R_m(v) \setminus \{r(v)\}$
12.            Send $SetRoot\ (v, r(v))$ to neighbors
13.        end if
14.        Send $UpdateInfo\ (v, \bigcup_{m \in \Gamma(v)} R_m(v))$ to $p(v)$
15.    end if
16. else if $(v \neq p(v)) \wedge (R_0 \neq \bigcup_{m \in \Gamma(v)} R_m(v))$ then
17.     Send $UpdateInfo\ (v, \bigcup_{m \in \Gamma(v)} R_m(v))$ to $p(v)$
18. end if
```

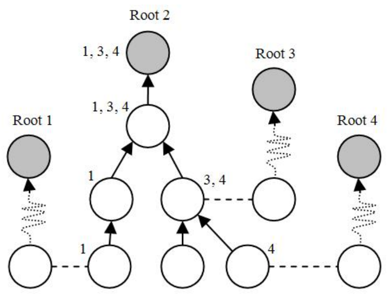

Fig. 1. Nodes learn from neighbors which are the adjacent clusters and propagate the knowledge to the parents.

Algorithm 2 Service registration - node v

```
UpdateInfo $(u, R, S)$:
// receive adjacent clusters $R$ and services $S$ from $u$
1: $R_0 = \bigcup_{m \in \Gamma(v)} R_m(v)$
2: $S_0 = \bigcup_{m \in \Delta(v)} S_m(v) \cup S(v)$
3: $\Delta(v) \leftarrow \Delta(v) \cup \{u\}$
4: $S_u(v) \leftarrow S$
5: $R_u(v) \leftarrow R \setminus \{r(v)\}$
6: if $(v \neq p(v)) \wedge ((R_0 \neq \bigcup_{m \in \Gamma(v)} R_m(v)) \vee (S_0 \neq \bigcup_{m \in \Delta(v)} S_m(v) \cup S(v)))$ then
7:     Send $UpdateInfo(v, \bigcup_{m \in \Gamma(v)} R_m(v), \bigcup_{m \in \Delta(v)} S_m(v) \cup S(v))$ to $p(v)$
8: end if
```

D. Maintenance in face of topology changes

We analyze how the clustering structure adapts to dynamic environments. We term the events regarding topology changes *LinkAdd* and *LinkDelete*. Algorithm 1 gives a detailed description of the behavior of node v when these events occur. In short, there are two situations where nodes adjust their cluster membership:

- A node discovers a new neighbor with a higher capability grade than its current parent. The node then selects that neighbor as its new parent.
- A node detects the failure of the link to its parent. The node then chooses as new parent the node with the highest capability grade in its neighborhood.

Besides reclustering, topology changes may also require modifications in the knowledge on adjacent clusters. The *SetRoot* message informs nodes about the cluster membership of their neighbors, while the *UpdateInfo* message is used for transmitting the updates from children to their parents. We distinguish the following situations:

- A node v detects a new neighbor from a different cluster. Consequently, v adds the root of that cluster to its knowledge.
- A node v switches from parent p_0 to p_1. Then v (1) notifies p_0 to remove the information associated with v and (2) sends the list of adjacent clusters to p_1.
- A node v detects the failure of the link to one of its neighbors u. As a result, v erases the knowledge associated with u.
- Any change of global knowledge at node v results in transmitting the message *UpdateInfo* from v to its parent.

V. Service discovery protocol

We now present the service discovery protocol, which relies on the clustering structure presented in Section IV.

A. Service registration

Each node keeps a registry of service descriptions of the nodes placed below in hierarchy. The root node knows all the service descriptions offered by the nodes in its cluster. Since the registration process requires unicast messages to be transmitted from children to parents, it can be easily integrated with the transfer of knowledge on adjacent clusters. Thus, the message *UpdateInfo* is used for both service registrations and transferring the knowledge on adjacent clusters. Algorithm 2 shows the integrated version of the *UpdateInfo* message, where a node updates the information on both the adjacent clusters and the known services.

In the following we describe how the distributed service registry is kept consistent when topology changes. In the case of a parent reselection, a child node v registers the services from its sub-tree with the new parent p_1, and notifies the old parent p_0 (if it is still reachable) to purge the outdated service information. The process is transparent for the other nodes in the sub-tree rooted at v. If the overall service information at p_0 and p_1 changes due to the parent reselection, the modifications are propagated up in the hierarchy.

B. Service discovery

The service discovery process uses the distributed directory of service registrations. Suppose a node in the network generates a service discovery request *ServDisc*. The request is first checked against the local registrations. In the case where no match is found, the message is forwarded to the parent. This process is repeated until the *ServDisc* message reaches the root of the cluster. When a root node receives a *ServDisc* message and it does not find a match in the local registry, the message is forwarded to the roots of the adjacent clusters. The next hop on the path leading to the adjacent cluster is decided by every node that acts as forwarder of the *ServDisc* message. Each node v along the path checks its $R_u(v)$ sets and picks a neighbor that has a path to the root of the adjacent cluster. In the case where a link is deleted and v cannot forward the *ServDisc* message, it chooses another neighbor that provides a path to destination. If such a neighbor does not exist, v informs its parent that it no longer has a route to the next cluster. The same procedure is repeated until all the paths to destination are tested. If the next cluster is not reachable, the root node erases the cluster from its knowledge.

The result of a service search is typically the address of one or more service providers. This response can be returned by the first node that finds a match in its registry for the requested service. However, in certain situations it may be preferable that the service provider itself issues a reply for the service request. Examples include applications where service descriptions change frequently, or cases where the reply incorporates more information than the address of the node. In these situations, the *ServDisc* message is forwarded down the cluster until it reaches the service provider. In the case where the link to the service provider is deleted or the service description is no longer valid, the service request is sent back to the root node which forwards it to the adjacent clusters.

The service discovery reply may follow the reverse cluster-path to the client, or any other path if a routing protocol is available. For the first case, if there is a cluster partition, the path can be reconstructed using the same search strategy as for the *ServDisc* message, where this time the service is the address of the client.

Caching the service discovery messages is a technique that allows us to cope with mobility. Root nodes cache the *ServDisc* messages for a limited period of time. If a newly arrived node registers a service for which there is a match in the cache, the root node can respond to the old service request. Moreover, when a root node is notified on a new adjacent cluster, it sends the valid service request entries from its cache to the new clusterhead. As a result, the overall hit ratio is improved.

Algorithm 3 describes the protocol, where replies are generated by nodes in the distributed directory and no caching is implemented. The message *ServDisc* has four parameters: the neighbor u that sends the request, the service description s, the final destination d of the message (typically a root node) and a flag f. The flag indicates whether the message is a fresh service discovery request, or it is a failure notification of a previous attempt to reach an adjacent cluster. In the latter case, the failed route is erased from the knowledge on adjacent clusters and another message is sent using an alternate path.

Algorithm 3 Service discovery - node v

```
ServDisc $(u, s, d, f)$:
// receive message ServDisc from neighbor $u$, requesting service $s$, destination $d$, flag $f$
1.  if $f = TRUE$ then
2.      if $s \in \bigcup_{m\in\Delta(v)} S_m(v) \cup S(v)$ then
3.          Service found; generate reply
4.      else if $p(v) = v$ then
5.          for all $r \in \bigcup_{m\in\Gamma(v)} R_m(v)$ do
6.              Pick $m \in \Gamma(v)$ such that $r \in R_m(v)$
7.              Send $ServDisc(v, s, r, TRUE)$ to $m$
8.          end for
9.      else if $d = r(v)$ then
10.         Send $ServDisc(v, s, d, TRUE)$ to $p(v)$
11.     else if $d \in \bigcup_{m\in\Gamma(v)} R_m(v)$ then
12.         Pick $m \in \Gamma(v)$ such that $d \in R_m(v)$
13.         Send $ServDisc(v, s, d, TRUE)$ to $m$
14.     else
15.         Send $ServDisc(v, s, d, FALSE)$ to $p(v)$
16.     end if
17. else
18.     $R_u(v) \leftarrow R_u(v) \setminus \{d\}$
19.     if $d \in \bigcup_{m\in\Gamma(v)} R_m(v)$ then
20.         Pick $m \in \Gamma(v)$ such that $r \in R_m(v)$
21.         Send $ServDisc(v, s, d, TRUE)$ to $m$
22.     else if $p(v) \neq v$ then
23.         Send $ServDisc(v, s, d, FALSE)$ to $p(v)$
24.     end if
25. end if
```

VI. Performance evaluation

In this section we evaluate the proposed clustering algorithm by comparing it to DMAC [2], when using both as distributed directory structures for the service discovery protocol. First, we briefly describe DMAC and provide a theoretical comparison between the two algorithms regarding the cluster density. Second, we introduce the general setting for both static and dynamic simulation experiments. Finally, we present the simulation results, including a performance evaluation of the service discovery protocol running on both structures under the same topological conditions.

In the following, we use the notation *C4SD* (Clustering for Service Discovery) for our proposed clustering algorithm. N represents the number of nodes in the network, r is the transmission range and a is the square side for a deployment area of size $a \times a$.

A. DMAC Clustering Algorithm

We choose DMAC as a viable clustering alternative for our service discovery protocol. Its simplicity and good performance results [3] make it suitable for sensor environments. DMAC achieves fast convergence, as nodes decide their roles based only on 1-hop neighborhood information. DMAC constructs the clusters based on unique weights assigned to nodes. The higher the weight, the more suitable is the node for the clusterhead role. The difference with our clustering algorithm is that DMAC imposes a maximum cluster height of one, whereas our protocol in principle may lead to arbitrary cluster height. For the construction of clusters, DMAC uses two types of broadcast messages, *Clusterhead* and *Join*, announcing the roles of the nodes to their neighbors. The role decision of a node is dependent on the decisions of the neighbors with higher weights. Therefore, a single topology change may trigger reclustering of a whole chain of dependent nodes, phenomenon called *chain reaction*. For a distributed directory composed of clusterhead nodes, the chain reaction leads to high overhead for maintaining consistent service registries. In Section VI-E we study the impact of the cluster height and the chain reaction over the performance of the service discovery protocol, in comparison with our proposed clustering solution.

B. Cluster density

The number of clusters is an important measure for the performance of a clustering algorithm that is intended to be used as a basis for a search mechanism. A high density of clusterheads leads to a large number of loops that occur during the discovery process.

We consider the nodes distributed on an area according to a homogeneous Poisson point process with density $\rho = N/a^2$. The spatial distribution of the root nodes for both clustering algorithms belongs to the family of *hard-core point proccesses*

[13], in which the constituent points are forbidden to lie closer together than a certain minimum distance. For our clustering algorithm, we approximate the cluster density by using the *Matérn hard-core process*. The retaining probability of nodes that become roots is the following:

$$P_{C4SD} = \frac{1}{\rho\pi r^2}(1 - e^{-\rho\pi r^2}) \tag{1}$$

This result enables us to compute the estimated number of clusters:

$$E_{C4SD} = P_{C4SD}N = \frac{a^2}{\pi r^2}(1 - e^{-\frac{N\pi r^2}{a^2}}) \tag{2}$$

The results obtained by Bettstetter [3] for the DMAC clustering algorithm indicate the following probability for a randomly chosen node to become clusterhead:

$$P_{DMAC} = \frac{1}{1 + \frac{\rho\pi r^2}{2}} \tag{3}$$

Thus, the estimation for the number of clusterheads in DMAC is:

$$E_{DMAC} = P_{DMAC}N = \frac{1}{\frac{1}{N} + \frac{\pi r^2}{2a^2}} \tag{4}$$

From Eq. 2 and 4 it can be easily shown that:

- $E_{C4SD} < E_{DMAC}$
- for r and a fixed, the function $f(N) = E_{DMAC} - E_{C4SD}$ is strictly increasing
- $\lim_{N\to\infty} E_{DMAC} = 2\lim_{N\to\infty} E_{C4SD}$

We can conclude that C4SD has a lower cluster density, and the difference in the number of clusters built by the two protocols increases with the network density. Moreover, C4SD almost halves the total number of clusters for saturated areas.

C. Simulation settings

For our experiments we use the OMNeT++ [14] simulation environment. We generate a random network, by placing N nodes uniformly distributed on a square area of size $a \times a$, where $a = 500m$. We consider links to be bidirectional, so nodes have the same transmission range, r. There is a link between two nodes if the distance between them is less or equal to r. Each node chooses a capability grade from a uniform distribution. Static nodes have higher capability grades than mobile nodes.

We test the performance of both clustering algorithms under the same topological conditions. We implement on DMAC the algorithm for maintaining the knowledge on adjacent clusters and for updating the service registry, using the *UpdateInfo* message. We use a heartbeat broadcast message periodically sent by every node to maintain the neighborhood information and to trigger the events *LinkAdd* and *LinkDelete*. The heartbeat is also used for the cluster setup and maintenance, replacing the *SetRoot* message for C4SD and the *Clusterhead* and *Join* messages for DMAC. The focus of our comparative simulations is the overhead induced by the *UpdateInfo* and *ServDisc* messages in dynamic environments.

For measuring the cluster height of C4SD we use the cyclic distance model for link formation, in order to avoid the border effects [3]. In this model, nodes at the border of the system area establish links via the borderline to the nodes located at the opposite side of the area. This setup approximates an area where nodes are distributed according to a Poisson point process [3].

In the dynamic experiments we use a simplified version of the random waypoint model [9]. We assume that the mobile nodes represent people walking, so the dynamics of the network is moderate. The transmission range is $r = 0.2a$. At the beginning, nodes are randomly placed on the simulation area, where they stay for a specified period of time. After this time expires, they choose a random destination and start moving towards that destination. Nodes are moving at $1m/s$, the approximate speed of a walking person. Upon arrival at the destination, nodes pause for 30 seconds before restarting the process. Due to the initialization problems that characterize the random waypoint mobility model [4], we discard the initial 1000 seconds of simulation time in each simulation trial and we count the number of messages for the next 1000 seconds. We average the results over at least 50 simulations.

D. Cluster height

In the first set of experiments we measure the average cluster height for our proposed clustering algorithm, and we show that it is a function only of the expected number of neighbors (or node degree). The expected node degree for a Poisson point process is [3]:

$$E(D) = N\frac{r^2\pi}{a^2} \tag{5}$$

We experiment with three transmission ranges: $0.1a$, $0.2a$ and $0.3a$. Figure 2 shows the results for these three values, as a function of the expected node degree, with the $5th$ and $95th$ percentile values as error bars. We can notice that for all the three transmission ranges, the points follow the same curve. Consequently, our first conclusion is that the average cluster height does not depend on the number of nodes, but it depends only on the expected number of neighbors. The second conclusion is that the average cluster height is lower than 2, and at least 95% of the clusters have the hight lower or equal to three. This important result indicates that we can achieve relatively small-height clusters without imposing a maximal hop diameter limit, which would increase the maintenance effort and generate the chain reaction effects.

E. Service discovery performance

We test the performance of the service discovery protocol using both DMAC and C4SD, under the same topological and mobility conditions. Due to the mentioned dissimilarities between the two protocols, we expect different behaviors when using them for discovery purposes: (1) the chain reaction of DMAC determines reclustering and re-registration of services with new clusterheads, implying higher maintenance overhead; (2) smaller-height clusters achieve faster convergence and higher hit ratio; (3) fewer clusters implies fewer loops in the discovery process and consequently, lower discovery overhead.

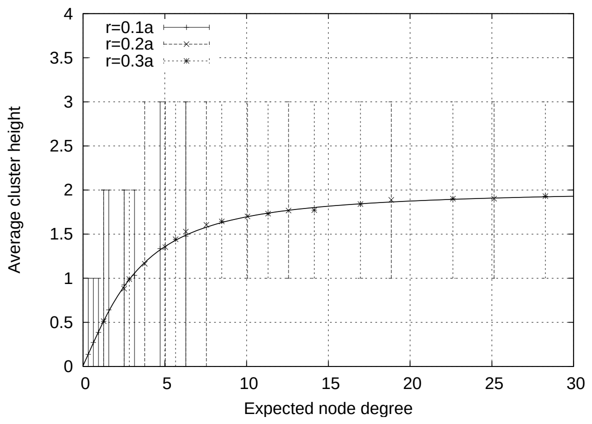

Fig. 2. Average cluster height with $5th$ and $95th$ percentiles.

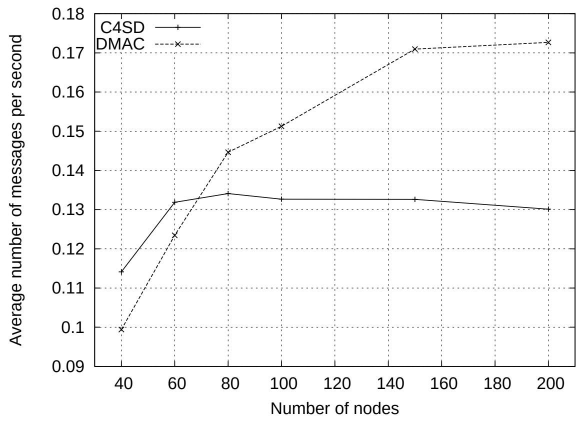

Fig. 3. The average number of *UpdateInfo* messages sent and received per node in one second depending on the number of nodes.

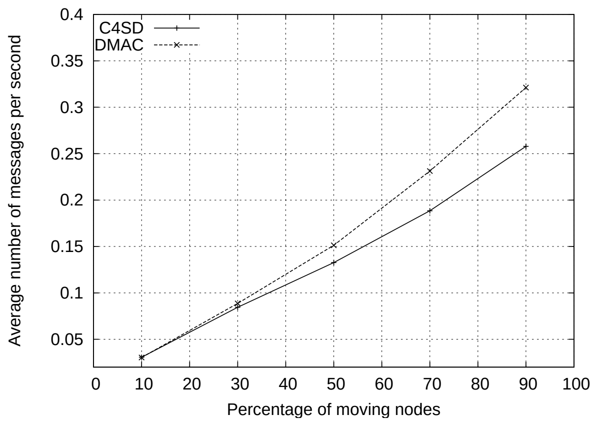

Fig. 4. The average number of *UpdateInfo* messages sent and received per node in one second depending on the percentage of moving nodes.

1) Maintenance overhead: In the first experiment we study the impact of the network density over the maintenance overhead (number of *UpdateInfo* messages), when 50% of the nodes are moving acording to the mobility model described in Section VI-C.

When a node moves from one cluster to another, the old service registration is deleted and a new registration is sent to the new clusterhead. However, the knowledge on adjacent clusters needs more overhead: when a node v moves from one cluster to another, all former neighbors of v must delete the information related to v and report the change to their parents, which can belong to different clusters. Similarly, all the new neighbors of v must add the information provided by v and send it to their parents. On the one hand, due to lower cluster density, C4SD has a lower overhead of maintaining the knowledge on adjacent clusters. On the other hand, the service registration is cheaper at DMAC due to the smaller cluster height. We are interested to examine the tradeoff of cumulative maintenance overhead with different network densities.

Figure 3 shows the average number of messages sent and received by a node in the network in one second of simulation time. For sparse networks, where there are few neighboring clusters, the DMAC protocol behaves better. For dense networks, the effort for maintaining the knowledge of adjacent clusters becomes prevalent over the overhead of service registrations, and thus C4SD overtakes DMAC.

We analyze the behavior further in terms of maintenance overhead when increasing the network mobility. Figure 4 shows the experimental results with 100 nodes and percentage of mobile nodes between 10% and 90%. We count the average number of messages per second sent and received by a node. C4SD behaves progressively better when increasing the network mobility. The reason is that the chain reaction inherent to DMAC triggers additional maintenance overhead of the directory structure, where the service information and knowledge on adjacent clusters needs to be updated at the new clusterheads. The more dynamic the network, the more probable is this reaction to occur.

2) Hit ratio: Since C4SD has an average cluster height bigger than DMAC, the convergence of service registrations is slower. In consequence, we expect DMAC to have a better hit ratio. For a fair comparison, we assume that each node provides exactly one service and for each service there is exactly one service provider. We generate random service requests from arbitrary chosen nodes. During 1000 seconds of simulation time we issue 10 service requests, with a delay of 100 seconds. The *ServDisc* messages are forwarded to the service provider (see Section V-B). If the service request reaches the matching service provider, we have a hit.

In our first experiments, no caching mechanism is involved. Figure 5 shows the results depending on the percentage of moving nodes. As expected, DMAC performs better than C4SD due to faster convergence. However, DMAC hit ratio drops similarly when increasing the network mobility. In our second set of experiments we implement a limited-time caching of service requests (see Section V-B). By implementing caching we obtain a high hit ratio for both protocols, which is above 0.98 for all mobility cases that we consider (see Figure 5).

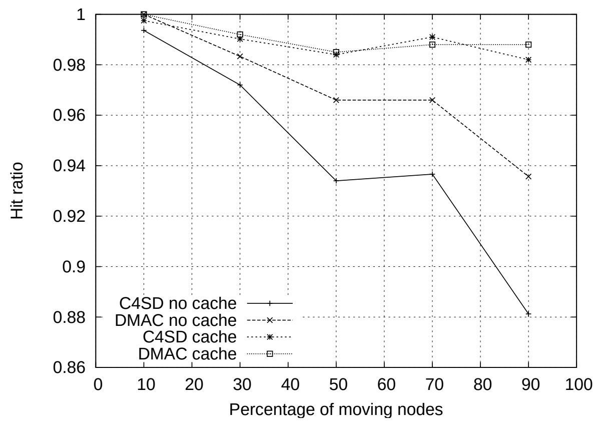

Fig. 5. Ratio of successful service requests depending on the percentage of moving nodes.

3) Discovery cost: We are interested in the number of *ServDisc* messages exchanged during one service discovery phase. Since C4SD has a lower cluster degree, we expect that it also experiences a lower discovery cost. Figure 6 shows the average number of service discovery messages per node, sent and received during one service discovery phase. We notice that caching implies more messages spent in the service discovery phase. The discovery cost is significantly smaller for C4SD, due to the lower cluster density. Moreover, DMAC experiences a rapid growth in the discovery cost when caching is implemented.

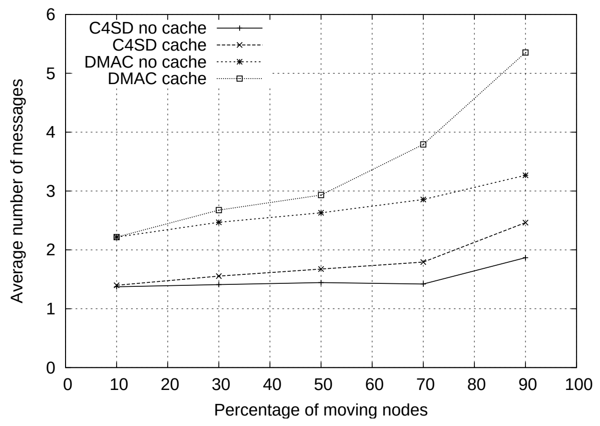

Fig. 6. The average number of *ServDisc* messages sent and received per node depending on the percentage of moving nodes.

VII. Conclusions

This paper proposes an energy-efficient solution to service discovery in wireless sensor networks. The discovery protocol relies on a clustering structure that offers distributed storage of service descriptions. The clusterheads act as directories for the services in their clusters. The structure ensures low construction and maintenance overhead, avoids the chain-reaction problems and keeps a sparse network of nodes in the distributed directory.

Our comparison with DMAC shows different performances of the service discovery protocol depending on the underlying clustering structure. We show that the chain reaction of DMAC determines reclustering and re-registration of services with new clusterheads, implying higher maintenance overhead. Our clustering algorithm achieves fewer clusters and consequently, lower discovery overhead. The smaller-height clusters of DMAC leads to faster convergence and higher hit ratio. The hit ratio is improved to more than 98% for both protocols if a mechanism of limited-time caching is implemented for service discovery messages. Our protocol has a lower discovery cost in both implementation alternatives.

For future work, we consider introducing dynamic capability grades, in order to avoid overloading the root and parent nodes with service registrations. The idea is that nodes that reach their memory limit decrease the capability grade and thus, a part of their children will register to other nodes.

References

[1] Magdalena Balazinska, Hari Balakrishnan, and David Karger. INS/Twine: A scalable peer-to-peer architecture for intentional resource discovery. In *International Conference on Pervasive Computing 2002*, pages 195–210, August 2002.

[2] Stefano Basagni. Distributed clustering for ad hoc networks. In *ISPAN '99*, pages 310–315, Washington, DC, USA, 1999. IEEE Computer Society.

[3] C. Bettstetter. *Mobility Modeling, Connectivity, and Adaptive Clustering in Ad Hoc Networks*. PhD thesis, Technische Universität München, Germany, October 2003.

[4] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *WCMC: Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, 2(5):483–502, 2002.

[5] Santanu Das, Charles E. Perkins, and Elizabeth M. Royer. Ad hoc on demand distance vector (AODV) routing. Internet-Draft Version 4, IETF, October 1999.

[6] Christian Frank and Holger Karl. Consistency challenges of service discovery in mobile ad hoc networks. In *MSWiM '04*, pages 105–114, New York, NY, USA, 2004. ACM Press.

[7] Ahmed Helmy. Efficient resource discovery in wireless AdHoc networks: Contacts do help. In *Resource Management in Wireless Networking*, volume 16, pages 419–471. Springer, 2005.

[8] D. Johnson, D. Maltz, and J. Broch. *DSR The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks*, chapter 5, pages 139–172. Addison-Wesley, 2001.

[9] David B Johnson and David A Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353, pages 153–181. Kluwer Academic Publishers, 1996.

[10] Ulas C. Kozat and Leandros Tassiulas. Service discovery in mobile ad hoc networks: An overall perspective on architectural choices and network layer support issues. *Ad Hoc Networks*, 2(1):23–44, June 2003.

[11] M. Marin-Perianu, T.J. Hofmeijer, and P. J. M.Havinga. Assisting business processes through wireless sensor networks. In *13th International Conference on Telecommunications (ICT 2006)*, 2006.

[12] Raluca Marin-Perianu, Hans Scholten, and Paul Havinga. CODE: A description language for wireless collaborating objects. In *Intelligent Sensors, Sensor Networks & Information Processing Conference (ISSNIP)*, pages 169–174. IEEE Computer Society Press, December 2005.

[13] Dietrich Stoyan, Wilfrid S. Kendall, and Joseph Mecke. *Stochastic Geometry and its Applications*. John Wiley and Sons, 1995.

[14] A. Varga. The omnet++ discrete event simulation system. In *ESM'01*, Prague, Czech Republic, June 2001.

[15] J. Wu and M. Zitterbart. Service awareness in mobile ad hoc networks. Boulder, Colorado, USA, March 2001. Paper Digest of the 11th IEEE Workshop on Local and Metropolitan Area Networks (LANMAN).